Behavior Composition Optimisation

A thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

Nitin Kumar Yadav
M.B.B.S, Ms.CS

School of Computer Science and Information Technology,
College of Science, Engineering, and Health,
RMIT University,
Melbourne, Victoria, Australia.

January 6, 2014

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; and, any editorial work, paid or unpaid, carried out by a third party is acknowledged.

Nitin Kumar Yadav
School of Computer Science and Information Technology
RMIT University
January 6, 2014

Acknowledgments

I am grateful to my supervisor, Dr. Sebastian Sardina, who introduced me to the fascinating world of theoretical computer science. He taught me to appreciate the beauty in elegant mathematics and to consistently strive for perfection. His encouragement, guidance and support made this endeavour possible.

I thank Prof. Giuseppe De Giacomo and Dr. Paolo Felli for hosting my research visit at La Sapienza University in Rome. Discussions with Dr. Giuseppe De Giacomo provided me with further in-depth understanding in areas related to this thesis. I am thankful to Dr. Lawrence Cavedon for helping me in polishing the thesis and for his support and encouragement throughout my candidature.

I would also like to thank my thesis reviewers, Professor Wiebe van der Hoek and Associate Professor Diego Calvanese, who provided encouraging and detailed feedback, and for their insightful thoughts.

Finally, I am grateful to my family for all their encouragement, love, and understanding. My parents, who have for ever supported me in all my ventures. My brother, who always stood by me. My partner, Anita, who had to bear the highs and lows, and for always believing in me. Rudra Vijai Singh for his guidance and wisdom, and Umang Mahindra for always providing the guiding force to make unconventional conventions. A heartfelt thank you.

Credits

Portions of the material in this thesis have previously appeared in the following publications:

- N. Yadav and S. Sardiña. Decision theoretic behavior composition. In L. Sonenberg, P. Stone, K. Tumer, and P. Yolum, editors, *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 575–582. IFAAMAS, 2011
- N. Yadav and S. Sardiña. Qualitative approximate behavior composition. In L. F. del Cerro, A. Herzig, and J. Mengin, editors, *Proceedings of the European Conference on Logics in Artificial Intelligence (JELIA)*, volume 7519 of *Lecture Notes in Computer Science*, pages 450–462. Springer, 2012. ISBN 978-3-642-33352-1
- M. Ramirez, N. Yadav, and S. Sardiña. Behavior composition as fully observable non-deterministic planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS).* AAAI Press, 2013
- N. Yadav, P. Felli, G. DeGiacomo, and S. Sardiña. On the supremal realizability of behaviors with uncontrollable exogenous events. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).* AAAI Press, 2013

The thesis was written in the Eclipse editor on Linux Mint, and typeset using the $\LaTeX\,2_\varepsilon$ document preparation system.
All trademarks are the property of their respective owners.

Contents

List of Figures

Abstract

The behavior composition problem involves automatic synthesis of a controller that is able to "realize" (i.e., implement) a desired target specification by suitably controlling a collection of already available, partially controllable, behaviors running in a partially predictable shared environment. A *behavior* in our context refers to an already existing functionality such as the logic of a device, a service, a standalone component, etc; whereas a *target specification* represents the desired non-existent functionality that is meant to be obtained through the available behaviors. Previous work in behavior composition has exclusively aimed at synthesising exact controllers, those that bring about the desired specification completely. One open issue has resisted principled solutions: *if the target specification cannot be completely implemented, is there a way to realize it "optimally"?* In this doctoral thesis, we propose qualitative and quantitative *optimisation* frameworks that are able to accommodate composition problems that do not admit the "perfect" coordinating controller. In the qualitative setting, we rely on the formal notion of simulation to define *realizable fragments* of a target specification and show the existence of a *unique supremal* realizable fragment for a given problem instance. In addition, we extend the qualitative framework by introducing *exogenous uncontrollable events* to represent observable contingencies. In the quantitative setting, we provide a *decision theoretic* approach to behavior composition by quantifying the uncertainties present in the domain. In all cases, we provide effective techniques to compute optimal solutions and study their computational properties.

CHAPTER 1

Introduction

"Not to be absolutely certain is, I think, one of the essential things in rationality."

–Bertrand Russell

Utilizing simple reusable modular components to obtain a complex functionality is a well known design methodology across various fields, e.g., software engineering [Heineman et al. 2005, Sametinger 1997], robotics [Brooks et al. 2005, Brugali et al. 2007], and hardware such as consumer electronics [Van Ommering et al. 2000]. For example, a recent study[1] shows that an average U.S. household owns more than 20 electronic devices such as televisions, tablets, phones, game consoles, etc. As the uptake of such devices continues to increase, these themselves are becoming smarter, as is the case, for example, with automatic vacuum cleaners [Kim et al. 1998] and smart televisions. Usually, each electronic device will offer its own separate control mechanism. For instance, to operate a vacuum cleaner one has to use the buttons provided on it. Of course, as the number of devices used in day-to-day lives will increase, handling each one of them individually will become impractical, thereby requiring home automation systems [Humphries et al. 1997, Gill et al. 2009]. For example, an automated home user could dim the lights, decrease the speaker volume, and ask a robot to make coffee [Tsai et al. 2010] from her mobile phone, while sitting on a couch and communicating with her friend. Other domains of applicability where modular components are used to build complex systems include composition of web services [Berardi et al. 2003a; 2005], factory robotic settings [Saffiotti and Broxvall 2005] and agent programming [De Giacomo et al. 2010a]. At the core, be it a home automation system, web service composition, or a factory setting, the challenge is to build complex functionalities by integrating and reusing available simpler devices. A desirable feature in such cases is to provide an abstract framework to model all these scenarios and *automatically* generate the required *glue* that integrates the available components.

[1]http://www.reuters.com/article/2012/04/05/idUS164780+05-Apr-2012+BW20120405

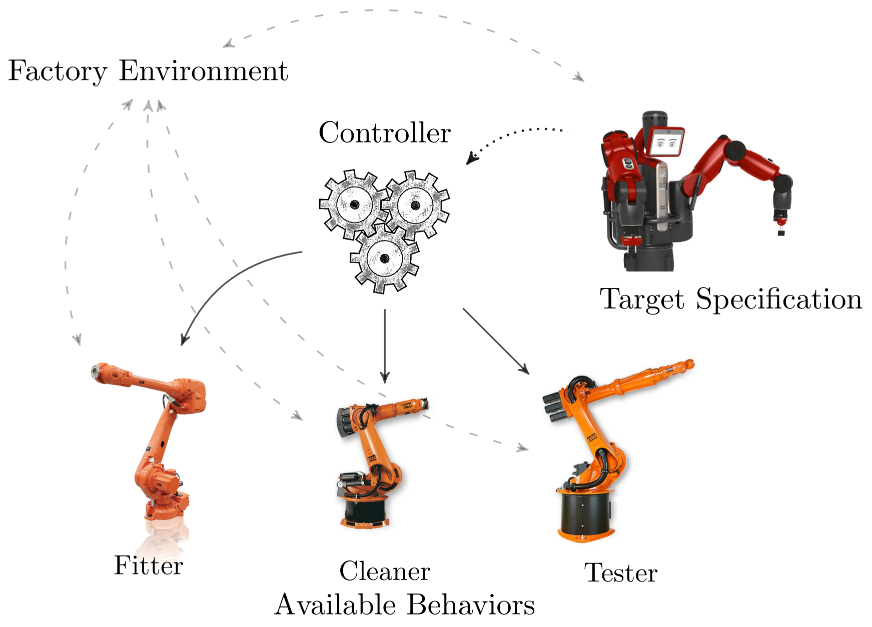


Figure 1.1: A factory setting for behavior composition.

As an example consider a factory scenario to manufacture cars, as pictured in Figure 1.1, where a part of the assembly line consists of robotic arms including a *fitter*, *cleaner*, and a *tester*. The fitter arm cuts and fits metallic sheets for the car's roof, the cleaner arm cleans off the metal scraps and fillings, and the tester arm tests the car roof for any faults. Additionally, both fitter and cleaner robots consumes cold water: the fitter arm uses it for cutting metal and the cleaner arms uses it to clean. Instead of operating each of these devices individually, the user wishes to operate just one global robotic device (the "target" device). Of course, such a device does not exist in reality. However, from an automation perspective, since the user knows what functionality she desires (the target device), one looks for a *controller* which can control and coordinate the installed devices such that *it appears as if the user is actually using a global robotic arm*. Then, we say that the target specification can be "realized" using the devices the user owns. The controller in this setting is the "glue" integrating the installed devices and providing an interface between them and the end user. For example, if the user asks the controller to clean, it is the responsibility of the controller to delegate the request to a device that can execute it (e.g., cleaner robot). In other words, as desired, the end user will only interact with the controller and not with the individual devices present.

Similarly, an ecommerce system, such as a travel booking site or an online merchandise store, may be built using a collection of web services. For example, a travel booking system may require third-party web services to book flights, arrange for local transport, search for accommodation, and transfer money via payment gateways. Each of these individual services may be called on to perform a specific function which may be a part of the complete system. For example, the payment gateway service can be used to accept money from the user when she buys flight tickets. Moreover, such systems will often require interaction between multiple services in order to complete a task. For instance, repeated search listings may require calling the payment gateway service multiple times in order to get the current minute exchange rates. The motivation in such a scenario is to look for an *orchestrator*, i.e. a controller, that can co-ordinate and monitor the individual web services in order to implement the complete online store functionality.

The question then is: *how to generate such a controller module automatically?* The *behavior composition problem* address this question formally.

1.1 The behavior composition problem

Since any of the individual devices owned by the end user is insufficient for achieving the overall desired functionality, the end user requires a controller that is able to manage the available devices to help her implement it. One could either build such a controller manually by a “try-test” procedure or construct it automatically in a way that guarantees it to be correct. Manual construction of such controllers is tedious, time consuming, and error prone; moreover it is difficult to check if one actually does exist. Automatic construction, on the other hand, does not need every output controller to be verified for correctness if a sound technique ensures this. The behavior composition problem deals with *automatic construction* of such a controller that is *able to manage the devices a user owns in order to implement a complex functionality she desires.*

1.1.1 Components of behavior composition

The term *behavior* in *behavior composition* refers to the abstract operational logic of a device or program. For example, a behavior could represent the dynamics of how a web service operates, or the internal logic of a vacuum cleaner, or the operational mechanics of a robot. Hence, the existing installed devices are called *available behaviors*; the desired complex functionality is called the *target specification*; and the glue which serves to coordinate between the components is called the *controller*. In addition, the available behaviors together with the shared environment are referred to as the *available system.*

In many instances, the dynamics of the available behaviors depends on shared resources; for example, a robot can clean only if water is available in a shared tank. Moreover, a behavior’s actions may in turn affect shared resources (after an exhaustive clean, the water tank may become empty). To model this shared space where behaviors are meant to operate, the notion of an *environment* is used. The environment serves to model shared resources and the preconditions/effects for the execution of actions by the devices.

Hence, the execution of a behavior may depend on the state of the environment, but not directly on the presence of other available behaviors. For instance, the presence of a fitter robot does not affect the working of a cleaner robot but the water tank in the environment affects both fitter and cleaner robots.

In addition, one may have *incomplete information* about the shared environment and the available behaviors. For instance, one may not know beforehand if the shared water tank will become empty after a single round of cutting. However, one can *observe* when the water tank is empty. Thus, in general, the environment is *fully observable* and *partially predictable*. Similarly, the cleaner robot may stop working if its waste collection bin becomes full with metal scraps. Hence, we say that the available behaviors are *partially controllable*. On the other hand, since the user knows what she desires, the target specification is *fully controllable*.

The composition problem is challenging not only because each of the available devices could themselves be complex (e.g., a robotic arm), but also, in order to realize the target functionality, one might need to consider multiple permutations of how each of the devices could be used. For instance, it may only be possible to use a cleaner after the fitter arm has finished cutting the metallic sheet or one may not be allowed to view available seat numbers before purchasing the flight ticket. Partial predictability of the environment and partial controllability of the available behaviors imposes further challenges as a controller will need to be *intelligent* enough to ensure the complete target can be implemented irrespective of such uncertainties arising in the domain.

Informally, the behavior composition task can be stated as follows:

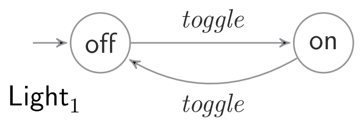
> *Given the set of behaviors available to the user, the (partially predictable) shared environment in which these (partially controllable) available behaviors are to operate, and the target specification the user wants to achieve, automatically build a controller that will be able to honor all the user's requests (as per the target specification).*

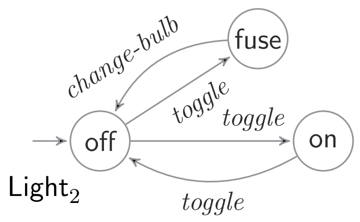
1.1.2 Modelling behavior composition components

From a problem perspective, the available system (consisting of available behaviors and the shared environment) and the desired target specification constitute the inputs to a behavior composition problem. If a problem instance has a solution, then one obtains a controller that will *realize* (i.e., achieve) the desired target specification. Otherwise, we say that the problem instance is unsolvable; in other words, the target specification cannot be achieved with the devices at hand.

Technically, the components of the behavior composition problem are modeled using *transition systems* [Baier et al. 2008]. Transitions systems are similar to automata and can be understood as directed graphs with nodes representing *states* and edges representing *transitions*. The transitions help in modelling how the states evolve as a result of actions performed by the device. In behavior composition, the available behaviors are modelled as *nondeterministic* transition systems to represent partial controllability; the environment is modelled via a *nondeterministic* transition system to depict partial predictability; and the target specification is a *deterministic* transition system to model full controllability. Intuitively, nondeterminism in this setting means that after executing an action it may not be possible to know how a behavior or the environment may evolve. Hence, nondeterminism helps in modeling the uncertainty in the problem. On the other hand, with deterministic behaviors one knows exactly how the device will evolve after an action execution.

To illustrate the above, consider two transitions systems shown below representing light devices:





Both lights have a toggle switch to turn them on and off. However, Light$_2$'s bulb may blow as a result of turning it on (it will need a change of bulb if that happens). Hence, Light$_2$ is modelled by a nondeterministic transition system to represent the incomplete information regarding the effect of switch toggling. In contrast, Light$_1$ is modelled via a deterministic transition system as we know its precise state after every action. Observe the *devilish* nature of nondeterminism in this context as compared to *angelic* nondeterminism as in the case of language acceptance in automata theory [Hopcroft et al. 2007].

Using a formal model to ground the core components of the behavior composition problem provides multiple advantages. First, it provides a clean framework for representing abstract scenarios such as home automation, service composition, agent planning, etc. Second, it provides a basis for developing computational approaches to automatically build controllers. Third, a formal framework is receptive to detailed mathematical analysis over properties such as computation complexity and component compatibility [Bordeaux et al. 2005]. For example, properties such as robustness to device failures [Sardina et al. 2008] of different approaches can be evaluated. Fourth, it allows use of formal notions such as *simulation* [Milner 1971] to capture solution concepts, as well as leveraging existing state-of-the-art tools for other formal frameworks, such as automated planning [Ghallab et al. 2004] and model checking [Alur et al. 2002], for controller synthesis. Finally, it serves as a basis to develop and evaluate various extensions to the problem.

Formally, the (classical) behavior composition problem can be stated as follows:

Given an available system $\mathcal{S}$ composed of a set of behaviors $\mathcal{B}_1, \ldots, \mathcal{B}_n$, $n \geq 1$, meant to operate in a shared environment $\mathcal{E}$, and a target specification $\mathcal{T}$,

> *automatically compute a controller* C *that will realize target specification* $\mathcal{T}$ *in the available system* $\mathcal{S}$*.*

We shall say a controller C is an *exact composition* of target specification $\mathcal{T}$ in system $\mathcal{S}$ if C realizes $\mathcal{T}$ in $\mathcal{S}$.

1.2 An open issue in behavior composition

The behavior composition problem has been extensively investigated in the AI literature (see [De Giacomo et al. 2013] for an extensive review). In fact, various techniques [De Giacomo and Felli 2010, De Giacomo and Sardina 2007, Lustig and Vardi 2009, Sardina et al. 2008, Stroeder and Pagnucco 2009] already exist to compute exact compositions. However, one important open issue has resisted principled solution:

> *If the target behavior specification cannot be realized in the available system, is there a way to realize it "optimally"?*

Many (if not most) realistic problem instances will have *no* complete realization and a (merely) "no solution" output may be extremely unsatisfactory, especially in large problem instances where one may have spent large amounts of time and resources in attempting to solve the problem. For example, in our factory automation scenario, the user might want to delay the cleaning of the roof after its testing has been done in order to conserve water (faulty parts will not need cleaning). However, this may not be possible as the tester robot may move the faultless part to the next assembly line right after testing. In such a case, the classical composition framework will simply convey that the problem instance is unsolvable because the motivation there is to solve the problem in its totality. In other words, if a composition problem is solvable, the classical framework will provide the user with an exact composition that will realize the desired target specification in the given system. However, for unsolvable instances the classical framework does not compute anything meaningful. The need for dealing with approximate solution concepts was first recognized by Stroeder and Pagnucco [2009], but the authors left this as important future work and this open issue has resisted principled approaches till now. Since many realistic scenarios will lack a technically complete solution, the ability to deal with such problems in better ways by providing optimal solutions will make behaviour composition applicable to a wider range of scenarios. Thus the overarching objective of this doctoral thesis is to address the following question:

> *What is an optimal solution if the target specification* $\mathcal{T}$ *cannot be realized in available system* $\mathcal{S}$*?*

1.3 Behavior composition optimisation

For an unsolvable behavior composition problem, the "optimal solution" could be found in multiple ways. One could suggest additional behaviors or extra functionality required in existing behaviors that will render the given target specification realizable. Alternatively, one could construct non-exact controllers which will achieve the target specification to the best possible extent by utilizing only the existing behaviors. One could also require extra domain information such as which parts of the target are more important and then build a controller that maximizes the implementation of the essential target parts. In this thesis we focus on the last two approaches by proposing qualitative and quantitative frameworks for behavior composition optimisation.

1.3.1 Qualitative behavior composition optimisation

The core of this thesis focuses on a *qualitative approach* to behavior composition optimisation. An optimal solution in this setting is found by checking how much of the target specification can be realized without requiring any further domain knowledge or additional available behaviors. The underpinning idea in the proposed approach is to shift the focus from synthesising a controller to synthesising parts of the specification which can be fully realized in the given system. Intuitively, the focus is to *look for those parts of the target specification that can be realized with the available modules*, and provide the maximal of them as a solution. We name such parts of target specifications as *Realizable Target Fragments (RTFs)* and show that RTFs are closed under their union. This union property is key in the sense that it allows more general (better) RTFs to be obtained from smaller RTFs and forms the basis for showing that the union of all possible RTFs is the largest possible realizable fragment - the *Supremal Realizable Target Fragment (SRTF).* In contrast to classical behavior composition, the qualitative optimisation framework ensures that a meaningful and practically useful solution is always returned.

In order to compute the SRTF for instances involving only deterministic available behaviors, we reduce the qualitative optimisation problem to a particular safety game [Bloem et al. 2011]. In such a game the target and controller jointly play against the available system. Intuitively, the target and the controller "win" the game if they can always ensure that only those actions that can be executed by an available behavior (in the context of the environment) are requested. Unfortunately, the two player game approach works only for deterministic available behaviors. Therefore, for computing the SRTF for the general case, we rely on a "belief-space" [Bonet and Geffner 2000] construction technique to compute the possible states where a system could be from the target's perspective.

Observe that similar to the classical setting, all the uncertainty (nondeterminism) in the qualitative optimisation framework remains unobservable. However, in actual settings a user can often observe the contingencies, e.g., blowing of a light bulb or running out of fuel. In order to embed such uncertain but observable effects, inspired by literature on *discrete event systems* [Wonham and Ramadge 1987, Cassandras and Lafortune 2006], we introduce *uncontrollable exogenous events* in the qualitative optimisation framework. These events are uncontrollable in the sense that their occurrence is not under the control of the available behavior and so they cannot be requested by the end user. Importantly, we show how to compute SRTFs involving exogenous events by parsimoniously adapting the belief-space construction approach used in the non-exogenous qualitative optimisation framework.

1.3.2 Quantitative behavior composition optimisation

Our second approach to behavior composition optimisation is a *quantitative* one relying on extra domain knowledge. If such extra information is available, then one can quantify the sources of uncertainty in the behavior composition problem (nondeterminism in the available system and importance of target requests) and develop a stochastic framework in which the task will be to generate a controller that has maximum likelihood to realize the target specification. Note, availability of extra domain information to the modeller is a reasonable assumption in many settings. For example, domain knowledge for a garden setting may include that watering plants is a more frequent and important action request than collecting fruits, or the failure rate of a garden cleaner due to sand deposition.

The outcome of the quantitative approach is a *decision-theoretic* [French 1986] behavior composition framework, in which the task is to maximize the so-called *expected realizabability* of a target behavior in the given available system. Expected realizability then implies the probability that a target specification will be realized in the system by a given controller. The optimal solution in the quantitative framework is an *optimal composition controller* that *maximizes* the expected target realizability. Clearly, such a stochastic framework is able to output "optimal" controllers even for problems that do not allow exact controllers. In addition, we characterize when an optimal controller is also an exact composition, thus subsuming the classical setting. We provide a technique to compute optimal composition controllers by encoding the problem as a particular kind of Markov decision process [Puterman 2005].

1.4 Contributions

This thesis is the first example of behavior composition optimisation frameworks that are equipped to cater for unsolvable composition problem instances. The main contributions of this thesis are as follows:

1. We provide a *qualitative optimisation approach* to address unsolvable composition problems without requiring extra domain knowledge.

- We show the *soundness* and *completeness* of supremal realizable target fragments (SRTFs) with respect to controllers.
- We prove key desirable properties of the framework such as the *uniqueness* of SRTFs (up to simulation equivalence).
- We provide *effective techniques* for computing SRTFs and discuss their respective computational complexity.
  - We use LTL synthesis and ATL model checking techniques to build SRTFs for problems involving only deterministic available behaviors.
  - We provide a "belief-space" construction technique for the general case involving nondeterministic available behaviors.

2. We extend the qualitative optimisation model by introducing *uncontrollable exogenous events* for modelling cases where uncertainties can be observed by the user.
   - Based on the user's observability of exogenous events we provide two solution types, namely, *conditional* and *conformant* SRTFs.
     - If the user has the capability to observe such events, then she can use a conditional SRTF.
     - If the user does not have observability over exogenous events, then the SRTF must be conformant.
   - We show that conditional and conformant SRTFs can be generated by parsimoniously modifying the "belief-space" technique for non-exogenous based SRTFs.
3. We develop a decision theoretic framework for *quantitative behavior composition optimisation* for cases where extra domain information is readily available.
   - We develop a probabilistic framework based on quantification of sources of uncertainty in the behavior composition problem.
   - We introduce a notion of "expected realizability" of target specifications to evaluate controllers.
   - We provide a technique for computing optimal controllers for the decision theoretic setting by reducing the problem to Markov decision processes.

1.5 Publications

Many of the results presented in this thesis have already been published in mainstream AI venues including IJCAI, JELIA, and AAMAS. The qualitative optimisation approach to behavior composition was first introduced in [Yadav and Sardiña 2012] and further refined in [Yadav et al. 2013]. The quantitative optimisation approach has been published in [Yadav and Sardiña 2011].

1.6 Thesis outline

The rest of the thesis is organized as follows:

- Chapter 2 provides the relevant background to this thesis. In this chapter, we formally introduce the classical behavior composition problem and define what constitutes a solution to the problem, along with outlines of different techniques to compute such solutions. In addition, we discuss interesting extensions of the behavior composition problem found in the literature.

- In Chapter 3 we present our *qualitative optimisation* framework and prove its key properties. We formally define *realizable target fragments* (RTFs), and *supremal realizable target fragments* (SRTFs), along with their union operation. We prove that SRTFs are *unique* up to simulation equivalence, a surprising though desired property. In addition, we argue for the nondeterministic relaxation of target specifications.

- In Chapter 4 we present effective techniques for computing the supremal target behaviors for both deterministic and nondeterministic available systems. To that end, we rely on LTL synthesis and ATL model checking for computing SRTFs for deterministic available systems and provide a sort of "belief-space" construction technique for nondeterministic available systems.

- Chapter 5 introduces the notion of *uncontrollable exogenous events* as events that may occur spontaneously in available behaviors. We extend the qualitative optimisation framework with such exogenous events and show how the definition of, and techniques to compute, SRTFs can be parsimoniously adapted.

- In Chapter 6 we provide a *quantitative optimisation* framework for behavior composition by viewing the problem from a decision theoretic perspective. The quantitative framework relies on extra domain knowledge to quantify uncertainties in the domain. We introduce the notion of *maximal controllers* as optimal controllers that maximize the target realizability and show how to compute them by using MDP solvers.

- Finally, in Chapter 7 we conclude the thesis by discussing the contributions and describing possible future work.

CHAPTER 2

Background

"The point of philosophy is to start with something so simple as not to seem worth stating, and to end with something so paradoxical that no one will believe it."
–Bertrand Russell

To understand any problem, its components and key ingredients must be precisely stated. We begin by introducing *transition systems*, the core ingredient for the formalisms presented in this thesis, along with some of their properties. We formally define the behavior composition problem and briefly discuss the relevant literature on behavior composition. The aim here is not to present all the results on the topic, but to paint a picture with enough formal, and intuitive, detail to appraise this thesis. Note that some of the formal definitions are borrowed from [Baier et al. 2008, De Giacomo et al. 2013, Sardina et al. 2008].

2.1 Transition systems

Transition systems are widely used in computer science to model abstract behavior of real devices [Baier et al. 2008]. They can be understood as directed graphs with nodes representing *states* and edges representing *transitions* among those states. For a given model of a device, its states portray the possible internal situation the device could be in, whereas, the transitions model how these situations can change as a result of actions performed using the device. Intuitively, transitions encode the effects on the device as a result of action execution. Consider a simplistic model of a vacuum cleaner with two states, as shown in Figure 2.1. The states, empty and full, represent the status of the vacuum cleaner's dust bin. When there is a lot of dust to be collected, a *clean* action may cause the bin to fill. If this happens, the vacuum cleaner will evolve to the state full after a *clean* action. Otherwise, the vacuum cleaner remains in the empty state. Notice that the rest of the vacuum cleaner's functionalities are not captured by this model. Indeed, one only models the aspects of the device relevant to the given problem, the rest being abstracted away. Formally:

Definition 2.1 (Transition system). A *finite transition system* is a tuple $\mathcal{T} = \langle S, A, s_0, \delta\rangle$ where:

- S is the finite set of $\mathcal{T}$'s states;
- A is the finite set of $\mathcal{T}$'s actions;
- $s_0 \in S$ is $\mathcal{T}$'s initial state; and
- $\delta \subseteq S \times A \times S$ is $\mathcal{T}$'s transition relation.

□

A transition $\langle s, a, s'\rangle \in \delta$, also written as $s \xrightarrow{a} s'$ in $\mathcal{T}$, denotes that action a executed in state s may lead the system to successor state s'. Based on the constraints over the transition relation, one may define two kinds of transition systems: *deterministic* and *nondeterministic*. A transition system is *deterministic* if there is no state $s \in S$ and action $a \in A$ for which there exists two transitions $s \xrightarrow{a} s'$ and $s \xrightarrow{a} s''$ in $\mathcal{T}$ with $s' \neq s''$. Intuitively, after executing an action, a deterministic transition system evolves to at most a single successor state. On the other hand, in a nondeterministic transition system, after executing an action the number of possible successors may be more than one. Note that the device is always in a single state at any point in time, even after executing a nondeterministic action. For instance, given two transitions $s \xrightarrow{a} s'$ and $s \xrightarrow{a} s''$ in $\mathcal{T}$, after executing action a from state s, $\mathcal{T}$ will evolve to either s' or s''. However, *before* executing action a from state s one cannot determine if the successor state will be s or s'; it is only *after* the transition system has evolved that one knows whether s or s' ensued.

Example 2.1. The transition system for the vacuum cleaner depicted in Figure 2.1 can be formally represented by the tuple $\mathcal{T}_{vc} = \langle\{\mathsf{empty}, \mathsf{full}\}, \{\mathit{clean}, \mathit{reset}\}, \mathsf{empty}, \delta\rangle$ where $\delta = \{\langle\mathsf{empty}, \mathit{clean}, \mathsf{empty}\rangle, \langle\mathsf{empty}, \mathit{clean}, \mathsf{full}\rangle, \langle\mathsf{full}, \mathit{reset}, \mathsf{empty}\rangle\}$. $\mathcal{T}$ is a nondeterministic transition system since after executing the *clean* action from the empty state the vacuum cleaner may either remain in the same state or evolve to the full state. □

To represent the execution of a transition system, we introduce the notion of *traces* and *histories* of a transition system. Informally, a trace of a transition system is an alternating sequence of states and actions, capturing a possible evolution that the system may go through. Formally, a *trace* of a transition system $\mathcal{T} = \langle S, A, s_0, \delta\rangle$ is a, possibly infinite, sequence of the form $\tau = s^0 \xrightarrow{a^1} s^1 \xrightarrow{a^2} \cdots$ such that *(i)* $s^0 = s_0$; and *(ii)* $\langle s^i \xrightarrow{a^{i+1}} s^{i+1}\rangle \in \delta$, for all $i \geq 0$. A *history* is a finite prefix of a trace ending in a state.

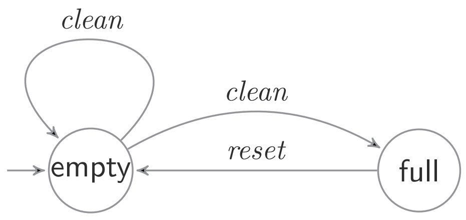


Figure 2.1: A simple model of a vacuum cleaner.

Let $h = s^0 \xrightarrow{a^1} \cdots \xrightarrow{a^\ell} s^\ell$, with $\ell \geq 0$, be a history of $\mathcal{T}$. The *length* of h, denoted by $|h|$, is the number of transitions included in it; that is, $|h| = \ell$, and the i^{th} state of the history h, where $0 \leq i \leq |h|$, is denoted by $h[i]$; that is, $h[i] = s^i$. The notions of length and state index can be extended to infinite traces. Given a, possibly infinite, trace $\tau = s^0 \xrightarrow{a^1} s^1 \xrightarrow{a^2} \cdots$, its i^{th} state, where $i \geq 0$, is denoted by $\tau[i]$; that is $\tau[i] = s^i$. Observe that the notion of traces and histories is akin to words of languages in the context of automata theory [Hopcroft et al. 2007], with actions corresponding to the alphabet of the automaton. However, here we include both states and actions as part of the system executions, whereas a word in a language only contains alphabet symbols. In fact, as we will see later, states play an important role when it comes to describing the behavior of a transition system.

Example 2.2. One of the possible traces of the vacuum cleaner $\mathcal{T}_{vc}$ is $\mathsf{empty} \xrightarrow{clean} \mathsf{empty} \xrightarrow{clean} \mathsf{full} \xrightarrow{reset} \mathsf{empty} \cdots$ and a possible history of length 2 is $\mathsf{empty} \xrightarrow{clean} \mathsf{empty} \xrightarrow{clean} \mathsf{full}$. $\square$

2.2 Behavioral equivalence of transition systems

Transition systems are abstract machines used to model devices or components of interest. One is often interested to know how two given models compare; for example, is one more general than the other or are they equivalent. Two well known measures for comparing transition systems throughout Computer Science are the notions of language equivalence [Hopcroft et al. 2007] and simulation [Milner 1971]. We start by formally defining these concepts and then discuss their properties in the context of (possibly non-deterministic) transition systems.

2.2.1 Language equivalence

Language equivalence for transition systems is the analogue of language equivalence for automata. Let $\mathcal{T} = \langle S, A, s_0, \delta \rangle$ be a transition system. We denote the set of all traces of $\mathcal{T}$ by $\Delta_\mathcal{T}$, that is:

$$\Delta_\mathcal{T} = \{s^0 \xrightarrow{a^1} s^1 \xrightarrow{a^2} \cdots \mid s^0 \xrightarrow{a^1} s^1 \xrightarrow{a^2} \cdots \text{ is a trace of } \mathcal{T}\}.$$

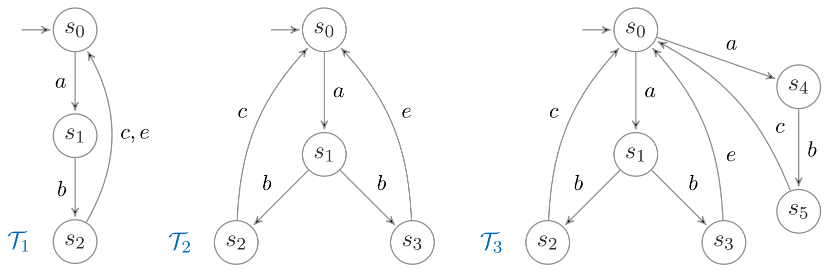


Figure 2.2: Example depicting different behavioral equivalences for transition systems.

Given a trace $\tau = s^0 \xrightarrow{a^1} s^1 \xrightarrow{a^2} \cdots$ of $\mathcal{T}$, we denote by $\tau^{\uparrow S}$ the sequence $a^1 a^2 \cdots$ obtained by projecting out $\mathcal{T}$'s states. We extend this projection to the set of all traces of $\mathcal{T}$:

$$\Delta_{\mathcal{T}}^{\uparrow S} = \{\tau^{\uparrow S} \mid \tau \in \Delta_{\mathcal{T}}\}.$$

Given two transition systems $\mathcal{T}_i = \langle S_i, A_i, s_{i0}, \delta_i \rangle$, for $i \in \{1, 2\}$, $\mathcal{T}_1$'s language is *included* in $\mathcal{T}_2$'s language iff $\Delta_{\mathcal{T}_1}^{\uparrow S} \subset \Delta_{\mathcal{T}_2}^{\uparrow S}$ and $\mathcal{T}_1$ and $\mathcal{T}_2$ are *language equivalent* iff $\Delta_{\mathcal{T}_1}^{\uparrow S} = \Delta_{\mathcal{T}_2}^{\uparrow S}$. Note that we ignore states when comparing language. The following example clarifies this.

Example 2.3. Figure 2.2 depicts three language equivalent transition systems $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ over a common set of actions. Observe that $\mathcal{T}_2$ and $\mathcal{T}_3$ are nondeterministic and have a branching structure, whereas, $\mathcal{T}_1$ is deterministic and has a linear structure. Observe that state s_1 in $\mathcal{T}_2$ and $\mathcal{T}_3$ offers more options when selecting the transition with label b as compared to transition system $\mathcal{T}_1$. $\square$

As we can see, the syntactic and the *branching* structure of the transition systems play no role when comparing their language equivalence. To compare the branching nature of transition systems, we need a stronger measure of equivalence – simulation.

2.2.2 Simulation

The formal notion of simulation was introduced by Milner [1971] and provides a finer comparison of nondeterministic transition systems than provided by language equivalence. Intuitively, a transition system $\mathcal{T}_1$ "simulates" another system $\mathcal{T}_2$ if $\mathcal{T}_1$ is able to *match* all of $\mathcal{T}_2$'s moves. Algebraically, simulation is a relation that captures the similarity in the behavior of two transition systems.

Definition 2.2 (Simulation). Let $\mathcal{T}_i = \langle S_i, A_i, s_{i0}, \delta_i \rangle$, where $i \in \{1, 2\}$, be two transition systems. A simulation relation of $\mathcal{T}_2$ by $\mathcal{T}_1$ is a binary relation $Sim \subseteq S_2 \times S_1$ such that $\langle s_2, s_1 \rangle \in Sim$ implies that:

- for all transitions $\langle s_2, a, s_2' \rangle \in \delta_2$ in $\mathcal{T}_2$, there exists a transition $\langle s_1, a, s_1' \rangle \in \delta_1$ in $\mathcal{T}_1$; such that

- $\langle s_2', s_1' \rangle \in \mathit{Sim}$.

We say that a state $s_2 \in T_2$ is <u>*simulated*</u> by a state $s_1 \in T_1$ (or s_1 <u>*simulates*</u> s_2), denoted by $s_2 \preceq s_1$, *iff* there exists a simulation relation *Sim* of $\mathcal{T}_2$ by $\mathcal{T}_1$ such that $\langle s_2, s_1 \rangle \in \mathit{Sim}$. □

The relation $\preceq$ is a *preorder* and is the largest simulation relation, in that all simulation relations are contained in it. Informally, $s_2 \preceq s_1$ is intended to mean that state s_1 in $\mathcal{T}_1$ can "mimic" all moves of state s_2 in $\mathcal{T}_2$, and that this property is propagated in their corresponding successor states. The transition system $\mathcal{T}_1$ simulates $\mathcal{T}_2$, denoted by $\mathcal{T}_2 \preceq \mathcal{T}_1$, iff their initial states are in simulation; that is, $s_{20} \preceq s_{10}$. The notation $\mathcal{T}_1 \prec \mathcal{T}_2$ implies $\mathcal{T}_1 \preceq \mathcal{T}_2$ and $\mathcal{T}_2 \not\preceq \mathcal{T}_1$. In addition, we say two transition systems are simulation equivalent if they simulate each other. Formally:

Definition 2.3 (**Simulation equivalence**)**.** Two transition systems $\mathcal{T}_1$ and $\mathcal{T}_2$ are <u>*simulation equivalent*</u>, denoted by $\mathcal{T}_1 \sim \mathcal{T}_2$, if $\mathcal{T}_2 \preceq \mathcal{T}_1$ and $\mathcal{T}_1 \preceq \mathcal{T}_2$. □

Example 2.4. Consider the three transition systems shown in Figure 2.2. Although, $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ are language equivalent, they are not simulation equivalent. One can verify that $\mathcal{T}_1$ simulates both $\mathcal{T}_2$ and $\mathcal{T}_3$ but the opposite is not true; that is, $\mathcal{T}_3 \preceq \mathcal{T}_2 \prec \mathcal{T}_1$. After the action sequence $a \cdot b$, transition system $\mathcal{T}_1$ will be in state s_2 and $\mathcal{T}_2$ could be in state s_2 or s_3. State s_2 of $\mathcal{T}_1$ is able to execute actions c and e, whereas state s_2 of $\mathcal{T}_2$ can execute only c and state s_3 of $\mathcal{T}_2$ can execute only e. In addition, the transition systems $\mathcal{T}_2$ and $\mathcal{T}_3$ are simulation equivalent; that is, $\mathcal{T}_2 \sim \mathcal{T}3$. □

Below, we formally state the well known result that simulation is a stricter notion of equivalence than trace equivalence.

Theorem 2.1 (Baier et al. [2008], Theorem 7.70)**.** *Let $\mathcal{T}_i = \langle S_i, A_i, s_{i0}, \delta_i \rangle$ for $i \in \{1,2\}$ be two transition systems. If $\mathcal{T}_2 \preceq \mathcal{T}_1$, then $\Delta^{\uparrow S}_{\mathcal{T}_2} \subseteq \Delta^{\uparrow S}_{\mathcal{T}_1}$.*

2.2.3 Bisimulation

Finally, we introduce *bisimulation* [Milner 1989, Baier et al. 2008], a behavioral equivalence notion even stricter than simulation equivalence. Intuitively, bisimulation requires the two transition systems to *exactly* match each others' moves.

Definition 2.4 (**Bisimulation**)**.** Let $\mathcal{T}_i = \langle S_i, A_i, s_{i0}, \delta_i \rangle$, where $i \in \{1,2\}$, be two transition systems. A <u>*bisimulation relation*</u> of $\mathcal{T}_2$ by $\mathcal{T}_1$ is a binary relation $\mathit{Bisim} \subseteq S_2 \times S_1$ such that $\langle s_2, s_1 \rangle \in \mathit{Bisim}$ implies that:

- for all transitions $\langle s_1, a, s_1' \rangle \in \delta_1$ in $\mathcal{T}_1$, there exists a transition $\langle s_2, a, s_2' \rangle \in \mathcal{T}_2$, such that $\langle s_2', s_1' \rangle \in \mathit{Bisim}$; and

- for all transitions $\langle s_2, a, s'_2\rangle \in \delta_2$ in $\mathcal{T}_2$, there exists a transition $\langle s_1, a, s'_1\rangle \in \mathcal{T}_1$, such that $\langle s'_2, s'_1\rangle \in \mathit{Bisim}$.

We say that a state $s_2 \in T_2$ is *bisimulated* by a state $s_1 \in T_1$ (or s_1 *bisimulates* s_2), denoted by $s_2 \cong s_1$, *iff* there exists a bisimulation relation *Bisim* of $\mathcal{T}_2$ by $\mathcal{T}_1$ such that $\langle s_2, s_1\rangle \in \mathit{Bisim}$. □

Note that the $\cong$ relation is an *equivalence* relation; that is, it is *reflexive*, *transitive*, and *symmetric*. Two transition systems $\mathcal{T}_1$ and $\mathcal{T}_2$ are *bisimilar*, denoted by $\mathcal{T}_2 \cong \mathcal{T}_1$, iff their initial states are in bisimulation; that is, $s_{20} \cong s_{10}$. Reverting to the three transition systems shown in Figure 2.2, see that $\mathcal{T}_2$ and $\mathcal{T}_3$ are simulation equivalent, however, they are not bisimilar. The transition $s_0 \xrightarrow{a} s_4$ of $\mathcal{T}_3$ can only be matched by transition $s_0 \xrightarrow{a} s_1$ of $\mathcal{T}_2$. Next, observe that transition system $\mathcal{T}_3$, after executing b from s_4, cannot match the move $s_3 \xrightarrow{e} s_0$ of $\mathcal{T}_2$ by any transition. Hence, transition systems $\mathcal{T}_2$ and $\mathcal{T}_3$ are not bisimilar.

Theorem 2.2 (Baier et al. [2008], Theorem 7.64). *Let $\mathcal{T}_i = \langle S_i, A_i, s_{i0}, \delta_i\rangle$ for $i \in \{1, 2\}$ be two transition systems. If $\mathcal{T}_2 \cong \mathcal{T}_1$, then $\mathcal{T}_2 \sim \mathcal{T}_1$. However, $\mathcal{T}_2 \sim \mathcal{T}_1$ and $\mathcal{T}_2 \ncong \mathcal{T}_1$ is possible.*

Theorem 2.2 shows that bisimulation is a stronger measure of equivalence than simulation in general. Note, all these notions collapse for deterministic systems; that is, for deterministic systems language equivalence, simulation equivalence, and bisimulation are equally powerful [Girard and Pappas 2007, Baier et al. 2008]. In terms of computational complexity, checking if a transition system simulates (bisimulates) another transition system can be performed in polynomial time [Baier et al. 2008, Balcázar et al. 1992] with respect to the size of the input transition systems. In comparison, checking for language equivalence is PSPACE-complete [Kanellakis and Smolka 1990].

2.3 The classical behaviour composition problem

Behavior composition deals with checking for the existence of a controller for a collection of available devices to realize a desired complex functionality. For example, consider a scenario where one has multiple robots to maintain a garden at home. The regular maintenance of the garden requires cleaning of the dirt, watering of the plants, plucking the ripe fruits and flowers, etc. Now, instead of operating multiple garden robots, it will be ideal to have just one super-bot that can do all these tasks as required by the user. However, since this super-bot does not exist in reality, one can then look for a controller that is able to manage the available devices for the user in order to achieve the desired functionality of garden maintenance. The idea is that the user will interact only with the controller rather than the individual robots she has, and it will appear as if she is operating just one device. The problem in behavior composition is to check if such a controller can be automatically generated given the devices the user owns and the desired complex (non-existent) functionality she requires.

The composition problem itself has attracted interest from multiple research communities such as LTL synthesis [Lustig and Vardi 2009], web services [Berardi et al. 2003b;a], and reasoning about action [Sardina and De Giacomo 2009]. Roots of behavior composition lie in the area of automated service composition [Berardi et al. 2003b, Rao and Su 2005, Berardi et al. 2003a, Calvanese et al. 2008, Sirin et al. 2004], where Berardi's thesis [Berardi 2005] formalised web services as finite state automatons and provided techniques to compute such controllers. Later, Patrizi's doctoral work [Patrizi 2009] extended the service composition framework to include general nondeterministic devices, thereby, making the problem relevant to a wider range of audience. Interested readers can find a detailed survey of behavior composition by De Giacomo et al. [2013].

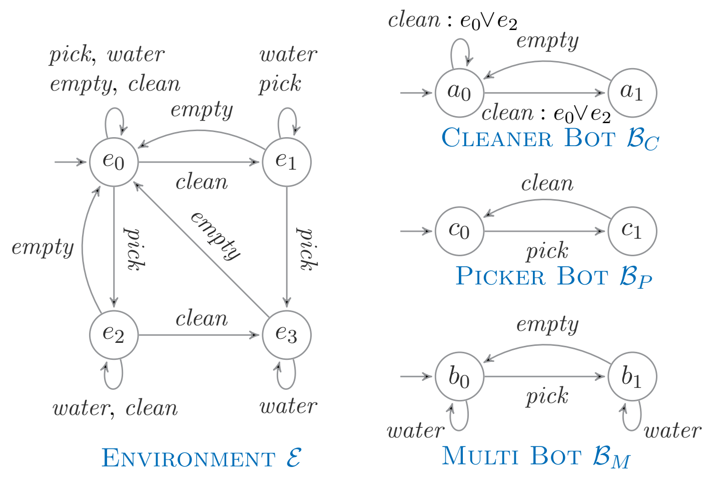
We shall present the behavior composition framework from [Sardina et al. 2008, De Giacomo et al. 2013] along with the relevant variations studied in the literature. We call the framework presented in this section the *classical* behavior composition framework to differentiate from the extensions we introduce in the later chapters.

2.3.1 Classical framework

In behavior composition terminology, the available devices at hand which need to be controlled, in order to obtain a given complex functionality are called the *available behaviors*. A *target specification* stands for such a complex functionality that is desired but not directly available and is therefore meant to be "realized" by suitably composing the available behaviors in the system. These available behaviors and the target are meant to be executed in a shared space called the *environment*. The environment can be viewed as a common playground where the available behaviors and the target operate. As such, the environment serves as a common medium to share information and resources between devices, and to reflect uncertainties that may arise outside of them. From a modelling point of view, the environment, available behaviors, and the target specification are represented using transition systems.

Environment

The available behaviors and the target operate in a shared *fully observable* space called the *environment*. This allows the modelling of preconditions and effects of actions as well as providing means of communication between the behaviors. Consider a garden scenario consisting of a common garbage bin and a number of robots which empty the bin when full. A required constraint to conserve energy is that a robot should only be able to execute the *empty* action if the garbage is full. Obviously, the garbage bin cannot be modelled as part of a single robot. Instead, the environment has to be used to provide such a shared



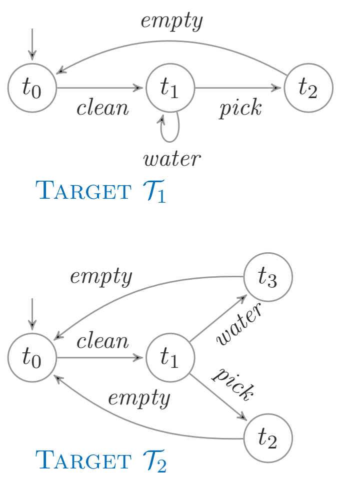


Figure 2.3: Behavior composition in a garden.

resource. That is, before attempting to *empty* the garbage bin, a robot will check if it is full or not in the environment. In other words, the precondition of the empty action is dependent on the shared garbage bin, which is a part of the environment. Formally:

Definition 2.5 (Environment). An *environment* is a tuple $\mathcal{E} = \langle E, A, e_0, \rho \rangle$, where:

- E is the finite set of environment's states;
- A is a finite set of shared actions;
- $e_0 \in E$ is the initial state;
- $\rho \subseteq E \times A \times E$ is the transition relation among states: $\langle e, a, e' \rangle \in \rho$, or $e \xrightarrow{a} e'$ in $\mathcal{E}$, denotes that action a performed in state e may lead the environment to a successor state e'.

□

Observe that the environment is modelled as a nondeterministic transition system. This is essential to model *incomplete information* about the effects of actions, similar to action theory [Reiter 2001]. For instance, in the garden scenario, after a *clean* action, the garbage bin may become full or it may still have more space. That is, there is uncertainty about the status of the garbage bin before a *clean* action. Therefore, to be able to represent settings like this, we allow the environment to be *nondeterministic* in general. Below, we present this gardening scenario of ours which will be used as an example of a behavior composition problem.

Example 2.5. Let us consider behavior composition applied in a garden setting. Figure 2.3 presents such a scenario, where multiple gardening bots are used to keep the garden healthy. The garden environment $\mathcal{E}$ allows *picking* the fruits, *watering* and *cleaning* the garden, and *emptying* the waste bins. In our garden, after a single *clean* (*pick*) action the garden may be cleared of all the dirt (fruits) or it might still contain some dirt (fruits). This uncertainty is modeled by the nondeterminism of *clean* and *pluck* actions. Take for instance the *clean* action. In state e_0 and e_2 the garden is dirty whereas in states e_1 and e_3 it is clean. A single *clean* (*pick*) action from e_0 may result nondeterministically in either e_0 or e_1. One can observe similar dynamics for the *pick* action in states e_0 and e_1. The other two actions, *water* and *empty*, are deterministic. In addition, as defined by system, emptying the waste bins always resets the environment. □

Available behaviors and system

An available behavior is basically an abstraction of a program, operational dynamics, or logic of a device. The available behaviors are considered to be passive; that is, they do not execute actions on their own, instead they are operated by an external agent. The agent directs the behavior to perform one of the allowed actions, execution of which may cause the behavior to evolve, thereby providing the agent a new set of available actions.

Obviously, behaviors interact with, and operate within, the environment. Hence, they are equipped with the ability to test conditions (i.e., guards) on the environment, to determine applicability as needed. For instance, a cleaning robot can check if the garden is dirty. We use guards only on the environment states and not on states of other behaviors because the internal working of an available behavior, and hence information about its states, is hidden from other available behaviors. For example, a vacuum cleaner's functionality cannot depend on the internal state of a light bulb as the vacuum cleaner does not have means to access the bulb's internal state. In case it requires a room to be lit for cleaning, it can only do so by checking if the environment is in a state with adequate lighting.

Definition 2.6 (Behavior). A <u>*behavior*</u> over an environment $\mathcal{E} = \langle E, A, e_0, \rho\rangle$ is a tuple $\mathcal{B} = \langle B, b_0, G, \varrho\rangle$, where: [1]

- B is the finite set of behavior's states;
- $b_0 \in B$ is the initial state;
- G is a set of *guards*; that is, Boolean functions $g : E \mapsto \{\texttt{true}, \texttt{false}\}$;

[1]Some formalisations [De Giacomo and Sardina 2007, De Giacomo and Felli 2010] use final states in the behavior definition, we omit them wlog.

- $\delta \subseteq B \times G \times A \times B$ is the behavior's transition relation, where $\langle b, g, a, b' \rangle \in \varrho$, or $b \xrightarrow{g,a} b'$ in $\mathcal{B}$, denotes that action a executed in behavior state b, when the environment is in a state e such that $g(e) = \texttt{true}$, may lead the behavior to a successor state b'.

$\square$

We say that a behavior $\mathcal{B} = \langle B, b_0, G, \varrho \rangle$ over an environment $\mathcal{E} = \langle E, A, e_0, \rho \rangle$ is *deterministic* if there is no behavior state $b \in B$ and no environment state $e \in E$ for which there exist two transitions $b \xrightarrow{g_1,a} b'$ and $b \xrightarrow{g_2,a} b''$ in $\mathcal{B}$ such that $b' \neq b''$ and $g_1(e) = g_2(e) = \texttt{true}$. In general, similar to the environment, behaviors are *nondeterministic*; that is, given a state and an action, there may be several transitions to different successor states whose guards evaluate to `true`. Nondeterminism in available behaviors is strongly tied to the notion of *controllability*. Before executing a nondeterministic action, the agent is uncertain of the resulting state of the behavior, and hence, also of the next set of available actions. As a consequence, nondeterministic behaviors are said to be *partially controllable* by the agent. In comparison, in a deterministic behavior, the resulting state is always known before executing an action, and hence, deterministic behaviors are *fully controllable*. For brevity, we omit the guard from a transition if that transition can be executed in all environment states. More formally, $b \xrightarrow{a} b'$ in $\mathcal{B}$ represents $b \xrightarrow{a,g_\top} b'$ such that $g_\top(e) = \texttt{true}$ for all $e \in E$.

We call the collection of the available behaviors at hand along with the environment as the *system*. Formally:

Definition 2.7 (**System**)**.** A <u>*system*</u> is a tuple $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ built from an *environment* $\mathcal{E}$ and a number of predefined, possibly nondeterministic, *available behaviors* $\mathcal{B}_i$, where $i \leq n$, over $\mathcal{E}$. $\square$

Example 2.6. Consider again the garden scenario depicted in Figure 2.3. Our garden is maintained by three garden bots, namely, a cleaner bot $\mathcal{B}_C$ to clean the garden, a picker bot $\mathcal{B}_P$ to pick fruits, and a multipurpose bot $\mathcal{B}_M$. Observe that the *clean* action is nondeterministic in the cleaner bot $\mathcal{B}_C$; after a *clean* action the bot's internal bin may be full and the bot may evolve to the state a_1, else it remains in the state a_0. In addition, to save energy the cleaner bot only cleans when it can sense dirt. This is reflected in the guards present on the *clean* action in $\mathcal{B}_C$; that is, the *clean* action can only be performed when the environment is in states e_0 or e_2 where indeed the garden is dirty. The remaining garden bots, picker bot $\mathcal{B}_P$ and multipurpose bot $\mathcal{B}_M$, are deterministic. $\square$

Target specification

A target specification represents the fully controllable desired behavior to be obtained through the available behaviors in the context of the shared environment.

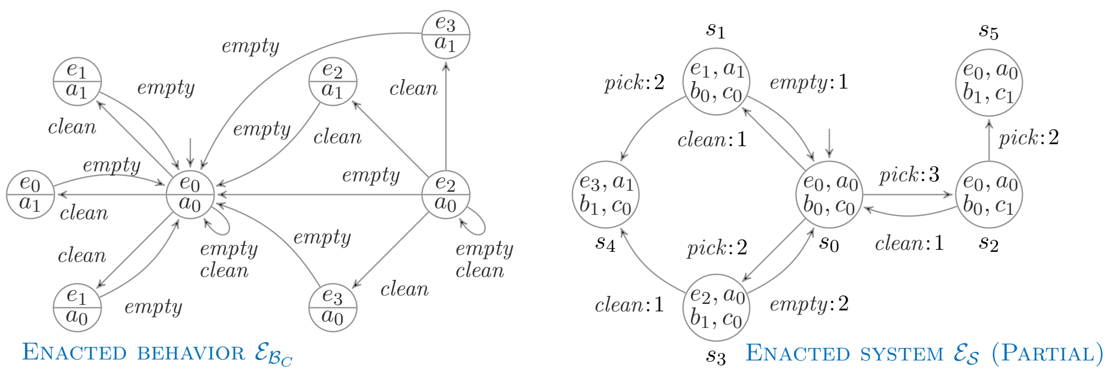


Figure 2.4: Enacted cleaner bot and partial enacted system for the garden example.

Definition 2.8 (Target specification). A *target specification* is a deterministic behavior over a fully observable shared environment. □

We say the target behavior to be *virtual* in the sense that it is not available, instead its behavior ought to be actualized by using the available system at hand.

Example 2.7. Figure 2.3 shows two possible target behaviors for our garden example. Both targets $\mathcal{T}_1$ and $\mathcal{T}_2$ start by *cleaning* the garden first, followed by either *watering* the plants or *picking* the fruits, and finally *emptying* the waste bins. However, target $\mathcal{T}_2$ requires the *empty* action to be done after *picking* the fruits as well as after *watering* the garden, whereas $\mathcal{T}_1$ requires *emptying* of waste bins only after *picking*. □

Enacted behaviors

Behaviors do not function in a standalone manner, rather they operate in a shared environment. As a result, some of the behavior actions may be not executable. For instance in the garden example, if the environment is in state e_3, then the cleaner bot $\mathcal{B}_C$ cannot execute the *clean* action. Hence, the actual capabilities of a behavior depend on both, the behavior itself and the environment in which it operates. The functionality that emerges from a behavior operating in an environment is denoted by its *enacted behavior*.

Definition 2.9 (Enacted behavior). Given a behavior $\mathcal{B} = \langle B, b_0, G, \varrho \rangle$ over an environment $\mathcal{E} = \langle E, A, e_0, \rho \rangle$, we define the *enacted behavior* of $\mathcal{B}$ over $\mathcal{E}$ as a tuple $\mathcal{E}_{\mathcal{B}} = \langle S, A, s_0, \delta \rangle$, where:

- $S = B \times E$ is the finite set of $\mathcal{E}_{\mathcal{B}}$'s states, given a state $s = \langle b, e \rangle$, we denote b by $\mathsf{beh}(s)$ and e by $\mathsf{env}(s)$;
- A is the set of actions in $\mathcal{E}_{\mathcal{B}}$;

- $s_0 \in S$, with $\mathsf{beh}(s_0) = b_0$ and $\mathsf{env}(s_0) = e_0$, is the initial state of $\mathcal{E}_{\mathcal{B}}$;
- $\delta \subseteq S \times A \times S$ is the transition relation, where $\langle s, a, s' \rangle \in \delta$, or $s \xrightarrow{a} s'$ in $\mathcal{E}_{\mathcal{B}}$, iff: *(i)* $\mathsf{env}(s) \xrightarrow{a} \mathsf{env}(s')$ in $\mathcal{E}$; and *(ii)* $\mathsf{beh}(s) \xrightarrow{g,a} \mathsf{beh}(s')$ in $\mathcal{B}$, with $g(\mathsf{env}(s)) = \mathtt{true}$ for some $g \in G$.

□

Observe that an action is executable from an enacted behavior state if it is executable from both the respective behavior and environment states, and there exists a behavior guard which evaluates to $\mathtt{true}$. Technically, enacted behavior $\mathcal{E}_{\mathcal{B}}$ is the synchronous product of the behavior and the environment, and represents all possible executions obtained from those of behavior $\mathcal{B}$ once guards are evaluated and actions are performed in the environment $\mathcal{E}$. In general, the sources of nondeterminism in enacted behaviors are twofold: the environment (effects of actions on the environment are nondeterministic); and the behavior itself (which may be nondeterministic). For the enacted behavior $\mathcal{E}_{\mathcal{T}}$ of a target specification $\mathcal{T}$ (*enacted target*) in environment $\mathcal{E}$, we denote the state of the target in an enacted target state s of $\mathcal{E}_{\mathcal{T}}$ by $\mathsf{tgt}(s)$ instead of $\mathsf{beh}(s)$.

Example 2.8. The enacted behavior of the cleaner bot $\mathcal{B}_C$ over the garden environment $\mathcal{E}$ is depicted in Figure 2.4. Note that after executing the *clean* action from the initial state, the enacted behavior could evolve to 4 possible successor states. This is due to the combined nondeterminism of the *clean* action in the environment and of the cleaner bot. □

All available behaviors in a system act in the same environment in an *interleaved* fashion; that is, at a single step an action is executed by only one behavior. As a result, the behavior executing that action and the environment evolve; all other behaviors remain stationary in their respective states. The so called *enacted system* is used to refer to the joint behavior that emerges from such an interleaved execution of the available behaviors.

Definition 2.10 (Enacted system). Let $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ be a system, where $\mathcal{E} = \langle E, A, e_0, \rho \rangle$ and $\mathcal{B}_i = \langle B_i, b_{i0}, G_i, \varrho_i \rangle$, for $i \in \{1, \ldots, n\}$. The <u>*enacted system*</u> behavior of $\mathcal{S}$ is the tuple $\mathcal{E}_{\mathcal{S}} = \langle S, A, \{1, \ldots, n\}, s_0, \delta \rangle$, where:

- $S = B_1 \times \cdots \times B_n \times E$ is the finite set of $\mathcal{E}_{\mathcal{S}}$'s states, when $s = \langle b_1, \ldots, b_n, e \rangle$, we denote b_i by $\mathsf{beh}_i(s)$, for $i \in \{1, \ldots, n\}$, and e by $\mathsf{env}(s)$;
- $s_0 \in S$ with $\mathsf{beh}_i(s_0) = b_{i0}$, for $i \in \{1, \ldots, n\}$, and $\mathsf{env}(s_0) = e_0$, is $\mathcal{E}_{\mathcal{S}}$'s initial state;
- $\delta \subseteq S \times A \times \{1, \ldots, n\} \times S$ is $\mathcal{E}_{\mathcal{S}}$'s transition relation, where $\langle s, a, k, s' \rangle \in \delta$, or $s \xrightarrow{a,k} s'$ in $\mathcal{E}_{\mathcal{S}}$, iff:
  - $\mathsf{env}(s) \xrightarrow{a} \mathsf{env}(s')$ in $\mathcal{E}$;

– $\mathsf{beh}_k(s) \xrightarrow{g,a} \mathsf{beh}_k(s')$ in $\mathcal{B}_k$, with $g(\mathsf{env}(s)) = \mathtt{true}$, for some $g \in G_k$; and
– $\mathsf{beh}_i(s) = \mathsf{beh}_i(s')$, for $i \in \{1, \ldots, n\} \setminus \{k\}$.

$\square$

Note that the enacted system is analogous to the enacted behavior, except for being suitably extended to a set of behaviors. Technically, the enacted system behavior $\mathcal{E}_\mathcal{S}$ is the asynchronous product of the available behaviors plus the synchronous product with the environment. The interleaved asynchronous execution of the behaviors is evident by the presence of behavior index k in the transitions. The presence of this index makes explicit which behavior in the system is the one performing the action in the transition—all other behaviors remain stationary.

Example 2.9. A part of the enacted system for the garden example can be seen in Figure 2.4. The indexes 1, 2, and 3 in the transitions refer to the cleaner bot $\mathcal{B}_C$, the multipurpose bot $\mathcal{B}_M$, and the picker bot $\mathcal{B}_P$, respectively. From the initial state s_0 the *pick* action can be executed by two behaviors, the multipurpose bot $\mathcal{B}_M$ or the picker bot $\mathcal{B}_P$; however, the *clean* action can only be performed by the cleaner bot $\mathcal{B}_C$. $\square$

Conceptually, the enacted system encompasses the *complete* functionality that can be achieved by asynchronously operating all the available behaviors. Informally, then, one checks if the given target specification can be achieved by choosing which behavior to activate in a step by step manner such that it appears that the target is being executed in the environment. When this is possible, we say the target is *realizable* in the given system.

2.4 Composition – solution to the problem

A *controller* is an external agent able to activate, stop, and resume any of the available behaviors, and to instruct them to execute an allowed action from their current state. The controller has *full observability* on the available behaviors; that is, it can keep track (at runtime) of their current states. As argued by De Giacomo and Sardina [2007], full observability is the natural choice in this context. Since available behaviors are already suitable abstractions for *actual* modules, if details have to be hidden, this can be done by means of nondeterminism within the abstract behaviors exposed. Roughly speaking, we look for a controller that can realize (i.e., implement) the target behavior by suitably operating the available ones.

To formally define controllers, we first extend the notion of traces for transition systems to enacted behaviors and enacted systems.

Definition 2.11 (Trace and History). A *trace* for a given enacted behavior $\mathcal{E}_{\mathcal{B}} = \langle S, A, s_0, \delta\rangle$ is a, possibly infinite, sequence of the form $s^0 \xrightarrow{a^1} s^1 \xrightarrow{a^2} \cdots$, such that *(i)* $s^0 = s_0$; and *(ii)* $s^j \xrightarrow{a^{j+1}} s^{j+1}$ in $\mathcal{E}_{\mathcal{B}}$, for all $j > 0$. A *history* is just a finite prefix $h = s^0 \xrightarrow{a^1} \cdots \xrightarrow{a^\ell} s^\ell$ of a trace. We denote s^ℓ by $\mathsf{last}(h)$, and ℓ by $|h|$. $\square$

Notions of trace and history extend naturally to enacted system: system traces have the form $s^0 \xrightarrow{a^1,k^1} s^1 \xrightarrow{a^2,k^2} \cdots$, and system histories have the form $s^0 \xrightarrow{a^1,k^1} \cdots \xrightarrow{a^\ell,k^\ell} s^\ell$. Functions $\mathsf{last}(\cdot)$ and $|\cdot|$ are extended in the obvious way.

Definition 2.12 (Controller). Let $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E}\rangle$ be a system and $\mathcal{H}$ be the set of all histories of the enacted system $\mathcal{E}_{\mathcal{S}}$. A *controller* for system $\mathcal{S}$ is a partial function $\mathsf{C} : \mathcal{H} \times A \mapsto \{1, \ldots, n\}$ which, given a system history $h \in \mathcal{H}$ and a target action $a \in A$, returns the index of a behavior that will execute the requested action a. $\square$

Next, following De Giacomo et al. [2013], we define when a controller realizes the given target specification in the available system.

Given a behavior composition problem, we say a controller is a solution to the problem if starting from the initial states of the target and the system, the controller can always delegate current and subsequent target action requests to an available behavior in the system. Informally, one first defines when a controller realizes a single trace of the target. Then, a controller is said to be *exact* if it can realize all the target traces. Note, since the target is deterministic, its behavior is fully characterized by the set of its traces (see Section 2.2 for details).

Due to nondeterminism present in the environment and the available behaviors, after executing an action the system may evolve to any of the possible successor states. Therefore, there could be multiple enacted system histories *induced* by a controller while realizing a target trace. Obviously, an exact controller will be able to delegate the next target action to an available behavior in all of such possibilities. Let us next formalise this notion of induced enacted system histories.

Definition 2.13 (Histories induced by a controller [De Giacomo et al. 2013]). Let $\tau = s^0_{\mathcal{T}} \xrightarrow{a^1} s^1_{\mathcal{T}} \xrightarrow{a^2} \ldots$ be a trace of the enacted target behavior $\mathcal{E}_{\mathcal{T}}$ of target specification $\mathcal{T}$ in environment $\mathcal{E}$. The set of histories of enacted system $\mathcal{E}_{\mathcal{S}} = \langle S, A, \{1, \ldots, n\}, s_0, \delta\rangle$ induced by controller C on trace τ is the set $\mathcal{H}_{\tau,\mathsf{C}} = \bigcup_{\ell\geq 0} \mathcal{H}^\ell_{\tau,\mathsf{C}}$, where:

- $\mathcal{H}^0_{\tau,\mathsf{C}} = \{s_0\}$;
- $\mathcal{H}^{j+1}_{\tau,\mathsf{C}}$ is the set of all $(j+1)$-length histories $h \xrightarrow{a^{j+1},k^{j+1}} s^{j+1}$ such that:
  - $h \in \mathcal{H}^j_{\tau,\mathsf{C}}$;
  - $\mathsf{env}(s^{j+1}) = \mathsf{env}(s^{j+1}_{\mathcal{T}})$;

– $k^{j+1} = \mathsf{C}(h, a^{j+1})$; that is, at history h, action a^{j+1} in trace τ is delegated to available behavior $\mathcal{B}_{k^{j+1}}$; and

– $\mathsf{last}(h) \xrightarrow{a^{j+1},k^{j+1}} s^{j+1}$ in $\mathcal{E}_{\mathcal{S}}$; that is, behavior $\mathcal{B}_{k^{j+1}}$ can actually execute action a^{j+1}.

$\square$

Informally, $\mathcal{H}_{\tau,\mathsf{C}} \subseteq \mathcal{H}$ represents the set of all possible enacted system histories that could result when controller C realizes target trace τ. Observe that we require the environment evolution to be synchronized over enacted target and enacted system traces. This is natural because we expect the environment to behave consistently for the system and the target. For instance, in the garden scenario, if the waste bin gets filled after two *clean* actions, then it will be so for both the system and the target. Since the enacted target trace already encodes the environment evolution that occurred, the induced enacted system histories ought to comply with the same environment evolution.

Example 2.10. Consider a controller C_1 for the garden example, where C_1 delegates all *clean* action requests to the cleaner bot $\mathcal{B}_C$ and *pick* action requests to the multi-bot $\mathcal{B}_M$. Let $\tau = \langle t_0, e_0\rangle \xrightarrow{clean} \langle t_1, e_1\rangle \xrightarrow{pick} \langle t_2, e_3\rangle$ be a trace of the enacted target $\mathcal{E}_{\mathcal{T}_1}$. Then, the set of enacted system traces induced by C_1 on τ include $\langle a_0, b_0, c_0, e_0\rangle \xrightarrow{clean,1} \langle a_0, b_0, c_0, e_1\rangle \xrightarrow{pluck:2} \langle a_0, b_1, c_0, e_3\rangle$ and $\langle a_0, b_0, c_0, e_0\rangle \xrightarrow{clean,1} \langle a_1, b_0, c_0, e_1\rangle \xrightarrow{pick,2} \langle a_1, b_1, c_0, e_3\rangle$, where 1 and 2 are behavior indexes of the cleaner bot $\mathcal{B}_C$ and multi-bot $\mathcal{B}_M$, respectively. $\square$

Intuitively, a controller realizes an enacted target trace, if starting from its initial state and following the trace thereupon, all induced enacted system histories can be extended by successfully delegating the next target request to one of the available behaviors in the system.[2] Since a deterministic transition system is characterized by all its traces, a controller *realizes* a target specification if it realizes all its (enacted target's) traces. Given a behavior composition problem with system $\mathcal{S}$ and target $\mathcal{T}$, a controller which realizes $\mathcal{T}$ in $\mathcal{S}$ is called an *exact* controller for $\mathcal{T}$ in $\mathcal{S}$. A solution to such a problem, that is an exact controller, is called a *composition*.

Definition 2.14 (Composition). A controller $\underline{C \textit{ realizes enacted target trace } \tau}$ if: for all $\mathcal{E}_{\mathcal{S}}$ histories $h \in \mathcal{H}_{\tau,\mathsf{C}}$ it holds that; if $|h| < |\tau|$, then $\mathsf{C}(h, a^{|h|+1}) = k$ and $\mathsf{last}(h) \xrightarrow{a^{|h|+1},k} s'$ in $\mathcal{E}_{\mathcal{S}}$ for some s', where $k \in \{1, \ldots, n\}$ and $\mathcal{E}_{\mathcal{S}}$ is the enacted system for system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E}\rangle$. A controller C $\underline{\textit{realizes}}$ a target behavior $\mathcal{T}$ in a system $\mathcal{S}$ iff C realizes all traces of enacted target $\mathcal{E}_{\mathcal{T}}$ in $\mathcal{S}$. A controller C is a $\underline{\textit{composition}}$ for a target specification $\mathcal{T}$ in system $\mathcal{S}$ if C realizes $\mathcal{T}$ in $\mathcal{S}$. $\square$

[2]Note, since we do not consider final states here (all states are final), there is no need to check the synchronisation of the final states of the enacted target and enacted system [De Giacomo et al. 2013].

We use the terms *exact controller* and *composition* interchangeably. Formally, the classical behavior composition problem can be stated as follows:

> *Given a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ and a deterministic target behavior $\mathcal{T}$ over $\mathcal{E}$, synthesize a* composition C *for $\mathcal{T}$ in $\mathcal{S}$.*

Example 2.11. Returning to the garden example, note that Figure 2.3 depicts two target specifications, $\mathcal{T}_1$ and $\mathcal{T}_2$. For $\mathcal{T}_1$ and $\mathcal{T}_2$, an exact controller exists only for the target specification $\mathcal{T}_1$. One can verify that the trace $\langle t_0, e_0 \rangle \xrightarrow{clean} \langle t_1, e_0 \rangle \xrightarrow{water} \langle t_3, e_0 \rangle \xrightarrow{empty} \langle t_0, e_0 \rangle$ of the enacted target $\mathcal{E}_{\mathcal{T}_2}$ cannot be realized in system $\mathcal{S}$, hence, the target specification $\mathcal{T}_2$ does not have a solution in $\mathcal{S}$. □

Note that any controller that can fully realize a target in a given system is an exact controller. In fact, for a given behavior composition problem there may be more than one exact controller. This may be due to redundant functionalities present in the available system or to having multiple copies of the same behavior. To formally capture all possible exact controllers for a given composition problem, the notion of *composition generator* [De Giacomo et al. 2013] is used.[3] Here, we present a more general definition of composition generator as compared to what is found in the literature.[4]

Definition 2.15 (Composition generator). Let $\mathcal{H}$ be the set of enacted system histories of system $\mathcal{S} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E}\}$ and $\mathcal{T}$ be a given target specification over joint actions A. Then, a *composition generator* for $\mathcal{T}$ in $\mathcal{S}$ is *the* function $\mathsf{CG} : \mathcal{H} \times A \rightarrow 2^{\{1,\ldots,n\}}$ such that: a controller C is an exact controller for $\mathcal{T}$ in $\mathcal{S}$ iff for all enacted system histories $h \in \mathcal{H}$ and actions $a \in A$, $\mathsf{C}(h, a) \in \mathsf{CG}(h, a)$. □

Interestingly, if there exists a solution for a given behavior composition problem, then its composition generator is *unique* [De Giacomo et al. 2013].

In terms of computational complexity, checking the existence of an exact composition for given target specification in a system is EXPTIME-hard [Muscholl and Walukiewicz 2007, Sardina et al. 2007]. Muscholl and Walukiewicz [2007] show the lower exponential bound by reducing the problem of checking if an alternating Turing machine with polynomial space bound loops on a given input to the problem of checking simulation between deterministic behaviors and a given target specification. Interestingly, involving nondeterministic behaviors does not change the computational complexity [Sardina et al. 2007].

[3]Note a slight departure from the terminology used in literature [Sardina et al. 2008, De Giacomo et al. 2013], we use the term composition generator instead of controller generator to clearly specify aggregation of compositions rather than controllers (which may not be exact).

[4]The commonly used definition [Sardina et al. 2008, De Giacomo et al. 2013] relies on an extended simulation relation and depends on the proven link between this extended simulation relation and exact controllers.

Theorem 2.3. *Let $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ be a system and $\mathcal{T}$ a target specification. The problem of checking the existence of a controller C able to realize $\mathcal{T}$ in $\mathcal{S}$ is* EXPTIME-*hard.*

Observe that both the exact controller and composition generator are defined as mathematical functions. It still remains to be seen whether such functions can actually be computed, and more importantly, represented finitely.

2.5 Synthesising compositions

Various techniques have been used to synthesise controllers for the classical behavior composition problems, including PDL satisfiability [De Giacomo and Sardina 2007], direct search-based approaches [Stroeder and Pagnucco 2009], ATL/LTL synthesis [Sardina and De Giacomo 2008, De Giacomo and Felli 2010], safety games [De Giacomo et al. 2013], and computation of special kind of simulation relation [Sardina et al. 2008, Berardi et al. 2008]. We briefly outline these techniques here.

2.5.1 Synthesising compositions via PDL-satisfiability

Propositional Dynamic Logic (PDL) is an extension of modal logic designed to reason about computer programs over propositional variables [Harel et al. 2000]. Syntactically, the language of PDL consists of two basic expressions: *propositions* and *programs*. Atomic propositions are Boolean variables and atomic programs are single instructions that can be executed in one step. Given a set of atomic propositions $\mathcal{P}$ and atomic programs Φ, complex formulae and programs can be built inductively by using PDL operators as follows [Harel et al. 2000]:

$$\begin{aligned} \varphi \rightarrow &\quad p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \Rightarrow \varphi_2 \mid [\alpha]\varphi \mid \langle\alpha\rangle\varphi \mid \texttt{true} \mid \texttt{false}; \\ \alpha \rightarrow &\quad a \mid \alpha_1 \cup \alpha_2 \mid \alpha_1 ; \alpha_2 \mid \alpha^* \mid \varphi?, \end{aligned}$$

where, $p \in \mathcal{P}$ is an atomic proposition and $a \in \Phi$ is an atomic program. The modal formula $\langle\alpha\rangle\varphi$ intuitively means that there is an execution of program α such that the execution terminates in a state satisfying φ. Its dual $[\alpha]\varphi$ implies that φ holds in all terminating executions of α.

Similar to modal logic, PDL formulae are interpreted over *Kripke frames*. A Kripke frame is a pair $\langle K, \mathfrak{m} \rangle$ where, K is a non-empty set of states and $\mathfrak{m}$ is a meaning function [Harel et al. 2000]. The meaning function $\mathfrak{m}$ serves a dual purpose: for each atomic proposition $p \in \mathcal{P}$, $\mathfrak{m}(p) \subseteq K$ returns the set of states in K in which p is $\texttt{true}$; and for each atomic program $a \in \Phi$, $\mathfrak{m}(a) \subseteq K \times K$ returns a set of state pairs $\langle k, k' \rangle$ such that executing program a from a state k may result in state k'. Intuitively, one can check if a formula of the type $\langle\alpha\rangle\varphi$ holds in a state k by following the sequence(s) of states inductively induced by $\mathfrak{m}(\alpha)$ and then checking if φ holds in the terminating state of any of these executions. Given a formula φ over a set of atomic propositions $\mathcal{P}$ and atomic programs Φ, the satisfiability problem in PDL is to check for the existence of a Kripke frame $\langle K, \mathfrak{m}\rangle$ such that φ is satisfiable in $\langle K, \mathfrak{m}\rangle$.

Observe that a non-atomic program is inductively built using regular expressions $(\cup, ;, *)$ and the core components of behavior composition have automaton-like structures. De Giacomo and Sardina [2007] leverage this similarity and provide an encoding schema to model the dynamics of enacted system evolution, target specification, and controller delegation via a PDL formula. In particular, given a behavior composition problem with a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E}\rangle$ and target specification $\mathcal{T}$ over joint actions A, the atomic programs Φ consist of actions A and the atomic propositions $\mathcal{P}$ consist of *(i)* propositions for each state in the behaviors, environment and the target; *(ii)* propositions for all possible controller (action-behavior index) delegations; and *(iii)* a special proposition undef to denote no behavior delegation is possible. Using these, a formula $\varphi_{\mathcal{S},\mathcal{T}}$ is built encoding the transition relation of the available behaviors, environment, and the target. In addition, the formula $\varphi_{\mathcal{S},\mathcal{T}}$ embeds constraints to ensure that only the delegated behavior, environment, and target evolve after an action is executed. De Giacomo and Sardina [2007] show that an exact controller exists for target $\mathcal{T}$ in system $\mathcal{S}$ iff the PDL formula $\varphi_{\mathcal{S},\mathcal{T}}$ is satisfiable. As pointed out earlier, there may be more than one exact controller, consequently there may be more than one model satisfying $\varphi_{\mathcal{S},\mathcal{T}}$. However, PDL satisfiability returns only one such model, therefore, only one possible exact controller can be computed at a time.

Every satisfiable formula in PDL admits a finite model which can be exponential in size with respect to the formula size in worst case [Harel et al. 2000]. This property of PDL, known as the *small model property*, implies that controllers for behavior composition, which were defined as mathematical functions, can indeed be computed and finitely represented. Moreover, the satisfiability problem for PDL is EXPTIME-complete [Harel et al. 2000], hence, one can check for existence of compositions in exponential time.

While, in theory, reduction to PDL provides a technique to solve behavior composition problems, there are no efficient PDL theorem provers to leverage. In addition, for solvable problem instances we obtain only a single controller instead of a composition generator.

2.5.2 Synthesising compositions via search

For a behavior composition problem, the enacted system represents the complete functionality that can be achieved as a result of operating the available behaviors in the given environment. Conceptually, the enacted system provides the feasibility boundary for a given behavior composition problem; that is, any realizable target specification ought to be within this boundary. Hence, in order to *search* for a composition (and composition generator), one only needs to consider the possibilities contained within the enacted system.

Stroeder and Pagnucco [2009] suggest a *forward search* approach to search for compositions. A state in their search based technique encapsulates a snapshot of the given composition problem [Stroeder and Pagnucco 2009]; it consists of the current state of the target, current state of the enacted system, an action that was requested by target in the previous step, a behavior that could have performed that action, and a set of possible action requests that the target may request next. The search proceeds in two phases [Stroeder and Pagnucco 2009]: an *expansion* phase and a *marking* phase. During the expansion phase the algorithm generates successor search states, starting from the initial state, allocating each possible target action request to an available behavior. In the marking phase, the algorithm checks for illegal states amongst the expanded ones and marks them. A state is considered illegal if there is no behavior which can be delegated an action request, or if any of its nondeterministic successor states is illegal. An exact controller exists for a given problem if the initial state is not marked.

Unlike the PDL-satisfiability approach, the search based technique computes the composition generator rather than a composition. Observe that, in practice, only the *reachable* parts from the initial state of the enacted system need to be searched. Hence, the forward search approach is efficient in cases where the target specification requires only a small subset of the enacted system functionality.

2.5.3 Synthesising compositions via simulation

Recall that the behavior composition problem considers existence of a controller able to control the available behaviors such that it appears as if one is actually executing the target behavior. At a high level, the controller is *restricting* the enacted system behavior (see Definition 2.10) to *match* the enacted target behavior. This "matching" of behavior very closely resembles the well known notion of simulation [Milner 1971]. Berardi et al. [2008] and Sardina et al. [2008] exploited this connection between behavior composition and simulation to derive a technique to generate composition generators.

Due to the nondeterminism present in the environment and the available behaviors, the standard notion of simulation is inadequate; instead an extension, called ND-*simulation*, is used. Informally speaking, the notion of ND-simulation requires the simulation property to be maintained across all nondeterministic evolutions.

Definition 2.16 (**ND-simulation** [De Giacomo et al. 2013, Sardina et al. 2008])**.** Let $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ be a system, $\mathcal{T}$ be the target behavior, and let $\mathcal{E}_\mathcal{S} = \langle S, A, \{1, \ldots, n\}, s_0, \delta_\mathcal{S} \rangle$ and $\mathcal{E}_\mathcal{T} = \langle T, A, t_0, \delta_\mathcal{T} \rangle$ be the enacted system and enacted target, respectively. An ND-simulation [Sardina et al. 2008] relation of $\mathcal{E}_\mathcal{T}$ by $\mathcal{E}_\mathcal{S}$ is a relation $\mathcal{R} \subseteq T \times S$ such that $\langle t, s \rangle \in \mathcal{R}$ implies:

1. $\mathsf{env}(t) = \mathsf{env}(s)$;

2. for all $a \in A$, there exists a $k \in \{1, \ldots, n\}$ such that for all transitions $t \xrightarrow{a} t'$ in $\mathcal{E}_\mathcal{T}$:

- there exists a transition $s \xrightarrow{a,k} s'$ in $\mathcal{E}_\mathcal{S}$ with $\mathsf{env}(t') = \mathsf{env}(s')$; and
- for all transitions $s \xrightarrow{a,k} s'$ in $\mathcal{E}_\mathcal{S}$ with $\mathsf{env}(t') = \mathsf{env}(s')$, it is the case that $\langle t', s' \rangle \in \mathcal{R}$.

□

Intuitively, an enacted target-system state pair $\langle t, s \rangle$ is in $\mathcal{R}$ iff *(i)* t and s share the same environment; and *(ii)* for all possible successors of t, there exists a behavior which can evolve from s such that, despite nondeterminism, the corresponding successors of t and s are in ND-simulation. We say an enacted target state t is ND-simulated by enacted system state s, denoted by $t \preceq_{\text{ND}} s$, iff there exists an ND-simulation relation $\mathcal{R}$ such that $\langle t, s \rangle \in \mathcal{R}$. The enacted system $\mathcal{E}_\mathcal{S}$ ND-simulates the enacted target $\mathcal{E}_\mathcal{T}$, denoted by $\mathcal{E}_\mathcal{S} \preceq_{\text{ND}} \mathcal{E}_\mathcal{T}$, iff their initial states are in ND-simulation: that is, $t_0 \preceq_{\text{ND}} s_0$. Similar to simulation, the relation $\preceq_{\text{ND}}$ is the largest ND-simulation relation; that is, all the ND-simulation relations are contained in it.

Coming back to composition controllers, Sardina et al. [2008] show that an exact controller exists for a given problem instance, if the enacted system ND-simulates the enacted target. This is an important result in the context of this thesis, so we state it formally:

Theorem 2.4 (Sardina et al. [2008])**.** *Let $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ be a system, $\mathcal{T}$ be the target specification, and let $\mathcal{E}_\mathcal{S} = \langle S, A, \{1, \ldots, n\}, s_0, \delta_\mathcal{S} \rangle$ and $\mathcal{E}_\mathcal{T} = \langle T, A, t_0, \delta_\mathcal{T} \rangle$ be the enacted system and enacted target, respectively. An exact controller C exists for $\mathcal{T}$ in $\mathcal{S}$ iff $\mathcal{E}_\mathcal{T} \preceq_{\text{ND}} \mathcal{E}_\mathcal{S}$.*

As noted earlier, there may be more than one exact controller for a given problem. Consequently, there may be more than one ND-simulation relation between the enacted target and the enacted system. However, all of these ND-simulation relations will be subsumed by the $\preceq_{\text{ND}}$ relation, the largest ND-simulation relation. This provides an important characterisation of the composition generator. More importantly, the $\preceq_{\text{ND}}$ relation can be computed by an elegant regression algorithm called *NDS* [Sardina et al. 2008]. The *NDS* algorithm first constructs all possible enacted target-system state pairs, and then iteratively removes the pairs which violate the local requirements of ND-simulation. The algorithm terminates when no more such pairs can be removed, implying that all remaining enacted state pairs are in ND-simulation.

In terms of time complexity, computing $\preceq_{\text{ND}}$ is polynomial in the size of the model; however, the model itself is exponential in the number of available behaviors. As a result, the ND-simulation based technique uses exponential time. However, as compared to the PDL based approach, we can now synthesize composition generators without an increase in time complexity.

Note that in order to compute the composition generator, the *NDS* algorithm constructs the enacted system and the enacted target separately. In contrast to the forward search approach, this will be inefficient in practice for problem instances where the target may require a small subset of enacted system functionality. In addition, the *NDS* algorithm uses explicit representations for the state pairs; thus it suffers from the state explosion problem [Burch et al. 1992], very-well known to the model-checking community. An efficient way to tackle state explosion is by using compact representations such as ordered binary decision diagrams (OBDD) [Meinel and Theobald 1998]. Hence, a practically amenable method is to rely on tools already built by the verification and model checking communities, such as JTLV [Pnueli et al. 2010](Java-based improved version of TLV [Pnueli and Shahar 1996]), MOCHA [Alur et al. 1998], NUGAT (based on NUSMV [Cimatti et al. 2002]), LILY [Jobstmann and Bloem 2006], and MCMAS [Lomuscio and Raimondi 2006, Lomuscio et al. 2009]. We present two such techniques next, one relying on LTL synthesis and the other on ATL model checking.

2.5.4 Synthesising compositions via LTL-synthesis

LTL synthesis involves automatic synthesis of programs from their specifications. De Giacomo and Patrizi [2010] use the two player game [Piterman et al. 2006] approach to LTL synthesis to automatically build the composition generator (i.e., a program) from a given target and system (i.e., specification). Roughly speaking, the game is played between two players, namely, the *environment-player* and *system-player*, where at each turn, the former moves and the latter responds. In the context of behavior composition, the environment-player is the target along with the system, and the system-player is the controller. Intuitively, the moves of the target/system (that is, the environment-player) involve requesting actions as per the target specification and evolving the enacted system as per the controller delegation. The controller (that is, the system-player) responds by delegating the requested action to an available behavior. The objective of the controller is to always be able to reply to the target/system such that it can honor the requested actions. Since the aim is to *always* satisfy a property, such a game is called a "safety" game.

Technically, the game is played over a safety game structure which consists of a mutually exclusive set of environment and system variables–let us call them $\mathcal{X}$ and $\mathcal{Y}$, respectively–and rules for updating their values. Let X (Y) be the set of all possible evaluations of variables in $\mathcal{X}$ ($\mathcal{Y}$). Then, a game state $\langle x, y\rangle \in X \times Y$ consists of a complete assignment of values to environment and system variables. The game proceeds as follows, from a given initial state the environment-player updates its set of variables (i.e., variables in $\mathcal{Y}$); then the system-player responds by updating the variables it controls (i.e., variables in $\mathcal{X}$), and so on. It is assumed that the system-player can see the environment-player's move before playing. Both environment-player and system-player can only update their variables as per the rules defined as part of the game structure. The system-player wins a game if it can ensure the given formula holds over infinite plays, else the environment-player wins. The task is then to synthesize a winning reply-strategy for the system such that the goal holds in all possible (infinite) “plays” that may ensue in the game when the system-player follows such a strategy. A state is called “winning” if there is a winning strategy from it.

De Giacomo and Patrizi [2010] show that the composition generator can be synthesised by building a winning strategy for a particular safety-game. The core idea behind the translation is as follows: the controller should *always* be able to satisfy the target's request, no matter how the system evolves (legally) or what action the target requests (compliant with its specification). Appropriately, the environment, behaviors, and the target comprise the *environment-player*; and the controller is the *system-player*. The formula that the system-player must ensure is $\Box\neg fail$, where $fail$ denotes an infeasible controller delegation; that is, the delegated behavior cannot execute the requested action. Then, a given problem instance accommodates an exact controller if the *system-player* can win the game starting from the initial state, and a composition generator can be extracted from the set of all winning states [De Giacomo and Patrizi 2010, De Giacomo et al. 2013].

2.5.5 Synthesising compositions via **ATL** model checking

Alternating-time temporal logic (ATL) [Alur et al. 2002] is a logic for reasoning about the ability of a group of agents (i.e., a coalition) in a multi-agent game structure. In the composition setting, each of the available behaviors, environment, target, and the controller can be considered to be an agent.

ATL formulae are built by combining propositional formulas, the usual temporal operators—namely, $\bigcirc$ (“in the next state”), $\Box$ (“always”), $\Diamond$ (“eventually”), and $\mathcal{U}$ (“strict until”)—and a *coalition path quantifier* $\langle\langle Ag \rangle\rangle$ taking a set of agents Ag as parameter. Intuitively, an ATL formula $\langle\langle Ag \rangle\rangle\phi$, where Ag is a set of agents, holds in an ATL structure if by suitably choosing their moves, the agents in Ag *can force* ϕ *true*, no matter how other agents happen to move. The semantics of ATL is defined in so-called *concurrent game structures* where, at each point, all agents simultaneously choose their moves from a finite set, and the next state deterministically depends on such choices.

De Giacomo and Felli [2010] show that the composition generator (i.e., a structure representing all exact compositions) can be synthesised by resorting to ATL model checking. In order to reduce a behavior composition problem, for a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ and target $\mathcal{T}$, to an ATL model checking problem, they basically define an ATL structure $\mathcal{M}_{\langle \mathcal{S}, \mathcal{T} \rangle}$ with one agent per available and target behavior, and one distinguished agent *contr* representing the controller. A state $\langle b_1, \ldots, b_n, t_s, a, e, \mathsf{ind} \rangle$ in such a model encodes the current state b_i of each available behavior, the current state t_s of the target, the current action a being requested by the target, the current state e of the environment, and the index of the available behavior to which the last action was delegated. The initial states of $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}$ encode all possible initial configurations of the composition framework, i.e., initial states for all behaviors and a legal initial request. The transition function of the structure $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}$ is made to encode all legal evolutions of the composition instance. The task then involves model checking the formula $\langle\langle \mathit{contr} \rangle\rangle \Box (\bigwedge_{i=1,\ldots,n} \mathit{state}_i \neq \mathit{error}_i)$ (against structure $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}$) which states that the controller agent has a strategy so that none of the n available behaviors end up in an error state. A behavior arrives to a distinguished "error"state if it is ever delegated an action that it cannot perform. As a result, the controller agent ought to ensure it always delegates actions in a way so as to satisfy every potential request; that is, it has to solve the composition problem. Finally, De Giacomo and Felli [2010] show how to *extract* a correct composition generator from the set of *winning states* $[\varphi]_{\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}}$, namely, all those states q in $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}$ from where the controller has a winning strategy. Intuitively, a winning state for them is one in which the current request is legally honored to some available behavior and *all* corresponding successor states are winning.

2.6 Variations of the classical behavior composition problem

The assumptions used in the classical behavior composition framework can be relaxed to cater for interesting problem extensions applicable to certain practical settings. For example, one could consider the available behaviors to be *partially observable* by the controller [De Giacomo et al. 2009], the available behaviors may act *concurrently* instead of one at a time [Sardina and De Giacomo 2008], or the controller itself may be assumed to be *distributed* [Sardina et al. 2007] instead of centralized. We briefly outline these extensions next.

De Giacomo et al. [2009] drop the full observability assumption by restricting the controller to have partial observability over the available behavior states. In order to do so, they associate an *observability* function with each available behavior; this is a standard way to handle partial observability in planning [Bertoli et al. 2001b] and decision theory formalisms [Monahan 1982, Kaelbling et al. 1998]. Intuitively, an observability function exports the perceivable information from each state. As a result, the controller may not know the actual state of a behavior, but instead can only track the exported observation. Hence, the authors [De Giacomo et al. 2009] extend the controller to consider histories over such observations instead of histories over actual states. A controller in such an instance is considered exact, if in spite of incomplete information it can guarantee the realisability of the target specification. Interestingly, unlike for automated planning [Haslum and Jonsson 2000, Rintanen 2004], the complexity class of the problem with partial observability remains the same as for the classical setting with full observability. This is due to the fact that partial observability is not at the level of enacted system, but is at the behavior level. In comparison, in automated planning such a decomposition does not exist, and therefore, the entire domain becomes partially observable.

There are cases [Lundh et al. 2008, Saffiotti and Broxvall 2005, Kim et al. 2004], where one may want to realize *multiple target specifications* rather than just one. Sardina and De Giacomo [2008] cater for this requirement by defining *concurrent composition*, wherein multiple target specifications can be realized by allowing multiple actions to be requested, one per target specification, at the same time. In this flavour of behavior composition, a problem instance consists of a set of available behaviors and set of target specifications, along with an environment. Here, a controller has two responsibilities: one, as usual, to delegate one of the requested actions to an available behavior, and second to progress one of the target agents, whose requested action was successfully delegated. To prevent starving a target agent, only one pending request is allowed per target. That is, a controller is supposed to be *fair* between the multiple targets. A controller is deemed exact if it can ensure all target specifications can be eventually realized. Intuitively, an exact controller carefully selects which action to realize next such that the other pending, current and future, action requests are not jeopardized.

A third possible relaxation lies in the *decentralisation* of control. This is of particular interest in cases where the available behaviors are distributed and not present in one central location, as in the case of the *RoboCup* domain [Bredenfeld 2006] and robot ecologies [Tilden 1993]. Sardina et al. [2007] tackle this by modelling message exchange between distributed controllers as a synchronisation mechanism. More importantly, they show that for every decentralized exact controller there exists a central exact controller and vice-versa.

In addition to the relaxation of the classical settings, behavior composition framework has been extended in the context of advanced planning [De Giacomo et al. 2010a;b]. Observe that the target specification in the classical setting is of a procedural nature, that is, the user of the target requires certain actions to be done. De Giacomo et al. [2010a] consider target specifications involving declarative goals [van Riemsdijk et al. 2005]; that is, the user requires certain goals to be achieved without specifying how to achieve them. The task in such a setting is to synthesize plans that will achieve the current requested goal in a way such that they will not compromise the feasibility of satisfying future goal requests. Second, De Giacomo et al. [2010b] consider a component based framework to solve a generalized planning problem under strong fairness. The behaviors used in their framework are more expressive because of the use of strong fairness constraints. In their framework, one can embed constraints such as: after several clean attempts the garden will eventually be free of dirt. Allowing of such constraints accommodates more expressive types of incomplete information which may impact the solvability of a problem instance [De Giacomo et al. 2010b].

We close this chapter by highlighting that behavior composition has received considerable attention in recent years. All the approaches outlined above synthesize an exact controller, when one exists, but for unsolvable problem instances, as in the case of target specification $\mathcal{T}_2$ in the garden example (Figure 2.3), these techniques do not output any useful information. Stroeder and Pagnucco [2009] were the first to recognize the need to cater for unsolvable problem instances. An important feature of their forward search based approach is that it is suited for finding *approximations* to the problem by modifying the marking phase. That is, one could potentially *relax* the marking of nondeterministic successors or search states in which all action requests cannot be delegated, in order to return close enough solutions. However, the authors did not provide any concrete modifications or formalisations of such approximations, and left it as important future work.

2.7 Summary

The classical setting for the composition problem has been extensively studied and enjoys efficient methods to solve the problem via state-of-the-art tools. However, a pressing question has resisted concrete answers: *How to deal with unsolvable problem instances?* The main focus of this thesis is to address this question in a principled manner. We shall shift the perspective at the behavior composition problem from a *feasibility* perspective to an *optimisation* one, thus providing meaningful (partial) solution concepts for unsolvable problem instances.

To summarise:

- Behavior composition deals with control and coordination of available behaviors (see Definition 2.6) to realize a complex, unavailable, target specification (see Definition 2.8) in a shared environment (see Definition 2.5).
- A controller is called a *composition* (see Definition 2.14), a solution to the problem, if it realizes the given target specification in the system.
- Techniques to automatically synthesise compositions include PDL satisfiability [De Giacomo and Sardina 2007], forward search [Stroeder and Pagnucco 2009], synthesis of a special kind of simulation relation [Sardina et al. 2008], LTL synthesis [De Giacomo and Patrizi 2010], and ATL model checking [De Giacomo and Felli 2010].
- The classical composition problem, along with the partial observable, distributed, and concurrent extensions, are all EXPTIME-complete.
- No known technique(s) exist to handle unsolvable composition problem instances.

CHAPTER 3

Supremal Realizable Target Fragments

"The greatest challenge to any thinker is stating the problem in a way that will allow a solution."
–Bertrand Russell

Classical behavior composition can be seen as a *yes/no* (decision) problem. Wherein the problem is to check the existence of an exact controller for a given target specification to be implemented via a set of available behaviors acting in shared environment, and the answer is either a "yes" or a "no." Though the techniques for behavior composition construct an exact controller if there exists one, unsolvable problem instances are not accompanied by any useful information (except that a composition does not exist for the given problem instance). Rather than just providing a negative answer, a more practically useful approach will be to provide additional insights into the unsolvability of that particular problem instance: for example, information such as which part(s) of the target specification is causing the problem to be unsolvable, or even better returning the *best* possible solution(s). In other words, we would like to translate the decision problem in behavior composition from a feasibility problem to an *optimisation* problem.

Recall that the behavior composition problem has three core components, namely, the available system (built from a collection of existing modules and a shared environment), the target specification, and a controller. In unsolvable cases, one could potentially "optimize" any of these elements. For instance, one could look at enhancing the system with new capabilities, shrinking the target specification, or returning the best possible controller. In the first case, the end user may be presented with additional capabilities (new behaviors or extensions to existing behaviors) required to render the given target specification realizable. Note that acquiring such capabilities may or may not be physically possible as these additional devices may not exist in reality. In comparison, the second and third optimisation strategies involve ways of maximising the best that can

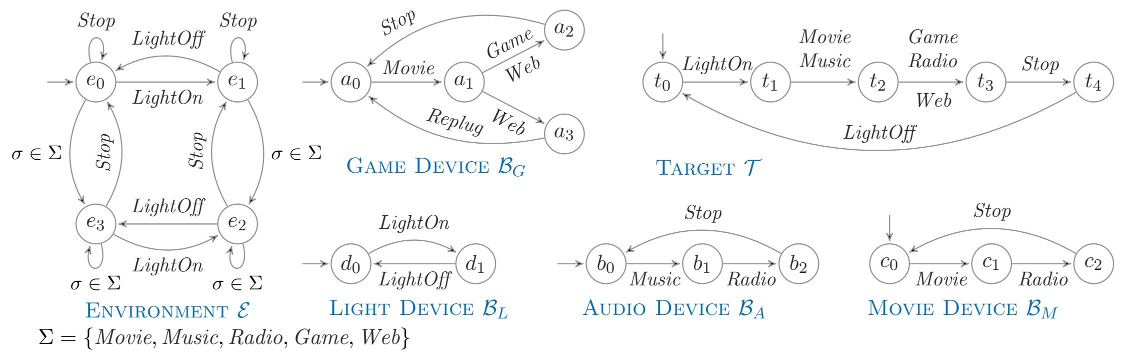


Figure 3.1: Behavior composition in ambient spaces.

be done with the existing devices at hand without requiring any further capabilities or domain knowledge, and will be the focus of this thesis.

Example 3.1. Consider a smart home scenario with a media room that has a collection of devices for entertainment of users, as shown in Figure 3.1. The media room environment $\mathcal{E}$ allows toggling the lights on/off, and playing and stopping various media options. The room is equipped with four devices, namely, a game device $\mathcal{B}_G$ to play games, browse the web, and watch movies; an audio device $\mathcal{B}_A$ for listening music from compact discs and radio; a movie device $\mathcal{B}_M$ to watch movies and listen to radio; and a light device $\mathcal{B}_L$ to switch lights on and off. Observe that in the room environment, operating the media devices and toggling the lights can be done independently. Amongst the devices present, only the game device $\mathcal{B}_G$ is nondeterministic: if the game device loses its Internet connection, executing the *Web* action will necessitate replugging the network cable. One can check that the target specification $\mathcal{T}$ does not have an exact controller. This is because, after the $t_1 \xrightarrow{Music} t_2$ request, the $t_2 \xrightarrow{Game} t_3$ target request cannot be delegated to any available device. $\square$

Informally, the best possible controller for the given target specification, is one that may not be exact, but will realize as much of the original target specification as possible. On the other hand, optimisation of the target specification results in a partial specification that can be implemented fully in the given system. We formalize both these optimisation strategies in this chapter, and in fact show that these are equivalent perspectives on behavior composition optimisation.

3.1 Extended framework

We begin by extending the classical framework to cater for these optimised components. We slightly relax the classical framework by allowing target specifications to be *nondeter-*

ministic. The objective for doing so is *not* so much to capture incomplete information, or allow the target specification to be partially controllable. Instead, the aim is to allow target requests to embed more implicit information, thus allowing *more expressive* specifications to be obtained. For example, consider two target specifications $\mathcal{T}$ and $\hat{\mathcal{T}}_2$ shown in Figures 3.1 and 3.3, respectively. Target specification $\hat{\mathcal{T}}_2$ forces the user to reveal their choice of *Game* or *Radio* before choosing the *Movie* action; in comparison, specification $\mathcal{T}$ allows the users to *delay* this choice to until the movie has finished. In this context, we say that $\hat{\mathcal{T}}_2$ has more information embedded into the specification as compared to the target module $\mathcal{T}$. In other words, target specification $\mathcal{T}$ allows greater "freedom of choice" to the user. Technically, this means that the user of a nondeterministic target specification has to choose in *advance* between different target traces having a common prefix. Informally, such a pre-selection provides more information on subsequent action requests to the controller. Indeed, in some cases this additional information may be the difference between a problem being solvable or unsolvable.

To ensure nondeterministic targets are still fully controllable, we extend the definition of a controller to account for target transition requests instead of action requests. For the rest of this section let $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ be an available system, $\mathcal{T} = \langle T, t_0, G, \varrho \rangle$ be a target specification, and $\mathcal{H}$ be the set of all histories of the enacted system $\mathcal{E}_\mathcal{S}$ (Definition 2.10 on page 24).

Definition 3.1 (**Nondeterministic transition-based controller**)**.** A controller for a potentially nondeterministic target specification $\mathcal{T}$ in a system $\mathcal{S}$ is a partial function $\mathsf{C} : \mathcal{H} \times (T \times G \times A \times T) \mapsto \{1, \ldots, n\}$ which, given a system history $h \in \mathcal{H}$ and a target transition $\langle t, g, a, t' \rangle \in \varrho$, returns the index $\mathsf{C}(h, t, g, a, t')$ of a behavior that will execute the action a in the requested transition $t \xrightarrow{g,a} t'$ of $\mathcal{T}$. □

For legibility, we write $\mathsf{C}(h, t \xrightarrow{g,a} t')$ to compactly denote $\mathsf{C}(h, t, g, a, t')$. The definition of an exact controller (i.e., a composition) extends naturally to account for transition requests instead of action requests.

Definition 3.2 (**Histories induced by a transition-based controller**)**.** Let $\tau = s^0_\mathcal{T} \xrightarrow{a^1} s^1_\mathcal{T} \xrightarrow{a^2} \ldots$ be a trace of the enacted target behavior $\mathcal{E}_\mathcal{T}$. We define the set of histories of enacted system $\mathcal{E}_\mathcal{S} = \langle S, A, \{1, \ldots, n\}, s_0, \delta \rangle$ <u>*induced by controller C on trace* τ</u>, as the set $\mathcal{H}_{\tau,\mathsf{C}} = \bigcup_{\ell \geq 0} \mathcal{H}^\ell_{\tau,\mathsf{C}}$, where:[1]

- $\mathcal{H}^0_{\tau,\mathsf{C}} = \{s_0\}$;
- $\mathcal{H}^{j+1}_{\tau,\mathsf{C}}$ is the set of all $(j+1)$-length histories $h \xrightarrow{a^{j+1},k^{j+1}} s^{j+1}$ such that:
  - $h \in \mathcal{H}^j_{\tau,\mathsf{C}}$;

[1]Recollect that for a given enacted target state $s = \langle t, e \rangle$, t and e are denoted by $\mathtt{tgt}(s)$ and $\mathsf{env}(s)$, respectively.

- $\mathsf{env}(s^{j+1}) = \mathsf{env}(s_{\mathcal{T}}^{j+1})$;
- $k^{j+1} = \mathsf{C}(h, \mathsf{tgt}(s^j) \xrightarrow{g^{j+1}, a^{j+1}} \mathsf{tgt}(s^{j+1}))$; that is, at history h, action a^{j+1} in trace τ is delegated to available behavior $\mathcal{B}_{k^{j+1}}$;
- there exists $g^{j+1} \in G$ such that $g^{j+1}(\mathsf{env}(s_{\mathcal{T}}^{j})) = \mathtt{true}$; that is, target $\mathcal{T}$ can execute action a^{j+1} from state $\mathsf{tgt}(s_{\mathcal{T}}^{j})$ when environment is in state $\mathsf{env}(s_{\mathcal{T}}^{j})$;
- $\mathsf{last}(h) \xrightarrow{a^{j+1}, k^{j+1}} s^{j+1}$ in $\mathcal{E}_{\mathcal{S}}$; that is, behavior $\mathcal{B}_{k^{j+1}}$ can actually execute action a^{j+1}.

□

Observe that the definition of *induced enacted system traces* is same as Definition 2.13 (see page 26), except for the fact that action requests are replaced by transition requests. While this has no impact when dealing with deterministic targets, it guarantees full controllability for nondeterministic ones. As before, we say that a controller *C realizes enacted target trace τ* if for all $\mathcal{E}_{\mathcal{S}}$ histories $h \in \mathcal{H}_{\tau,\mathsf{C}}$: if $|h| < |\tau|$, then $\mathsf{C}(h, \mathsf{tgt}(s_{\mathcal{T}}^{|h|}) \xrightarrow{g^{|h|+1}, a^{|h|+1}} \mathsf{tgt}(s_{\mathcal{T}}^{|h|+1})) = k$ and $\mathsf{last}(h) \xrightarrow{a^{|h|+1}, k} s'$ in $\mathcal{E}_{\mathcal{S}}$ for some s', where $g^{|h|+1}(\mathsf{env}(s_{\mathcal{T}}^{|h|})) = \mathtt{true}$. Finally, we define a transition-based exact controller, the same as before, as relying on the set of enacted target traces it realizes.

Definition 3.3 (Transition-based composition). A controller C *realizes* a possibly nondeterministic target behavior $\mathcal{T}$ in a system $\mathcal{S}$ iff C realizes all traces of enacted target $\mathcal{E}_{\mathcal{T}}$ in $\mathcal{S}$. A controller that realizes $\mathcal{T}$ in $\mathcal{S}$ is called a *transition-based composition* (exact controller) for $\mathcal{T}$ in $\mathcal{S}$. $\textsc{ExactComp}(\mathcal{S}, \mathcal{T})$ denotes the set of all exact controllers for a target $\mathcal{T}$ in system $\mathcal{S}$. □

Note that the definition of an exact controller still relies on enacted target traces, even for nondeterministic target specifications. This is because, to ensure full controllability of the target module, we allow traversing of nondeterministic targets via transitions instead of actions. Semantically, this is equivalent to treating nondeterministic targets as deterministic ones as *the* successor state is always unique, and known. The next result shows that one can easily obtain an action-based exact controller in the classical setting from a transition-based exact controller.

Theorem 3.1. *Let C be a transition-based exact controller for a deterministic target specification $\mathcal{T}$ and system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$. Then C^d is an action-based exact controller for $\mathcal{T}$ in $\mathcal{S}$ (as in the classical setting), where C^d is defined as follows:*

$$\begin{aligned} \mathsf{C}^d(h,a) = \mathsf{C}(h,t,g,a,t') \text{ where } h = s^0 \xrightarrow{a^1} \cdots s^\ell \text{ and there exists} \\ t^0 \xrightarrow{g^1,a^1} \cdots \xrightarrow{g^{\ell-1},a^{\ell-1}} t \xrightarrow{g,a} t' \in \Delta_{\mathcal{T}} \text{ such that} \\ g^i(\mathit{env}(s^{i-1})) = g(\mathit{env}(s^\ell)) = \mathtt{true} \text{ for } 0 < i \leq \ell. \end{aligned}$$

Proof. First, note that C^d is well defined: since $\mathcal{T}$ is deterministic there is at most one target trace to justify a given system history. Second, we show C^d is a composition for $\mathcal{T}$ in $\mathcal{S}$ as follows. Assume that C^d is not an exact controller for $\mathcal{T}$ in $\mathcal{S}$. Therefore, there exists a trace $\tau = s^0_{\mathcal{T}} \xrightarrow{a^1} s^1_{\mathcal{T}} \xrightarrow{a^2} \ldots$ of enacted target $\mathcal{E}_{\mathcal{T}}$ which cannot be realized by C^d. Hence, there exists a system history $h = s^0 \xrightarrow{a^1} \cdots s^\ell$, where $\ell \geq 0$, induced by C^d, such that $\mathsf{C}^d(h, a^{\ell+1})$ is undefined and $\mathsf{C}^d(h[0,i], a^{i+1}) \in \{1, \ldots, n\}$ for $0 \leq i < \ell$, where $h[0,i]$ is the i length prefix of history h. Consequently, $\mathsf{C}(h, t^\ell, g^{\ell+1}, a^{\ell+1}, t^{\ell+1})$ is undefined, where $t^\ell = \mathtt{tgt}(s^\ell_{\mathcal{T}})$, $t^{\ell+1} = \mathtt{tgt}(s^{\ell+1}_{\mathcal{T}})$, and $g^{\ell+1}(\mathsf{env}(s^\ell)) = \mathtt{true}$. However, this is absurd because C is an exact controller, therefore it cannot be undefined for a legal transition request compliant with a given system history. Thus, C^d is an exact controller for $\mathcal{T}$ in $\mathcal{S}$. $\square$

Lastly, note that the framework presented here is a *strict* extension to the classical framework. By definition, all deterministic target specifications are trivially valid nondeterministic specifications; however, the converse is not true.

Relaxation to nondeterministic target behaviors, along with suitable extension of controller definition, is the *only* extension required to the classical composition framework. Surprisingly, although this is a minor departure from the classical setting, it will enable us to capture solution concept(s) for unsolvable problems in a principled manner.

3.2 Maximal compositions

Suppose that we are given a target specification $\mathcal{T}$ and an available system $\mathcal{S}$, and that, as expected for many problems, there is no exact composition for $\mathcal{T}$ in $\mathcal{S}$—the target specification cannot be fully realized in the system. Merely returning a "no solution" outcome is clearly highly unsatisfactory. In such cases, what does it mean for a controller C_1 to achieve "a better realization" of $\mathcal{T}$ in $\mathcal{S}$ than a controller C_2?

To answer such a question in a qualitative manner and without requiring further domain knowledge, we first rely on the extent to which these two controllers are able to honour arbitrarily long sequences of enacted target requests. Intuitively, we say that a controller C_1 *dominates* another controller C_2 if C_1 realizes all the traces of enacted target $\mathcal{E}_{\mathcal{T}}$ in system $\mathcal{S}$ that are realized by C_2, and possibly more. Let $\Delta^{\mathsf{C}}_{\langle \mathcal{S},\mathcal{T}\rangle}$ denote the set of enacted target traces realized by a controller C in system $\mathcal{S}$. Formally:

$$\Delta^{\mathsf{C}}_{\langle \mathcal{S},\mathcal{T}\rangle} = \{\tau \in \Delta_{\mathcal{E}_{\mathcal{T}}} \mid \mathsf{C} \text{ realizes } \tau \text{ in } \mathcal{S}\}.$$

Recall that $\Delta_{\mathcal{E}_{\mathcal{T}}}$ is the set of all traces of transition system $\mathcal{E}_{\mathcal{T}}$. Then:

Definition 3.4 (Controller dominance). Given two controllers C_1 and C_2 for a target specification $\mathcal{T}$ in system $\mathcal{S}$, C_1 <u>*dominates*</u> controller C_2, denoted by $\mathsf{C}_1 \geq \mathsf{C}_2$, *iff* $\Delta^{\mathsf{C}_2}_{\langle \mathcal{S},\mathcal{T}\rangle} \subseteq \Delta^{\mathsf{C}_1}_{\langle \mathcal{S},\mathcal{T}\rangle}$. $\square$

As usual, $\mathsf{C}_1 > \mathsf{C}_2$ is equivalent to $\mathsf{C}_1 \geq \mathsf{C}_2$ and $\mathsf{C}_2 \not\geq \mathsf{C}_1$; that is, $\Delta^{\mathsf{C}_2}_{\langle \mathcal{S},\mathcal{T}\rangle} \subset \Delta^{\mathsf{C}_1}_{\langle \mathcal{S},\mathcal{T}\rangle}$. Then, *maximal* compositions, the "best" compositions, are those for which there is no other controller that can realize strictly more traces of the enacted target in the system.

Definition 3.5 (Maximal composition). A controller C is said to be a *maximal composition* (for a target on a system) *iff* for every other controller C', if $\mathsf{C}' \geq \mathsf{C}$, then $\mathsf{C} \geq \mathsf{C}'$. □

We use MaxComp$(\mathcal{S},\mathcal{T})$ to denote the set of all maximal compositions for $\mathcal{T}$ in $\mathcal{S}$.

Example 3.2. Consider two controllers C_1 and C_2 for the target specification $\mathcal{T}$ in our smart house scenario (see Figure 3.1). Whereas controller C_1 allocates all transition requests to the light device $\mathcal{B}_L$, controller C_2 delegates media requests to the audio device $\mathcal{B}_A$ and lights requests to the light device $\mathcal{B}_L$. Thus C_1 realizes just one trace; that is, $\Delta^{\mathsf{C}_1}_{\langle \mathcal{S},\mathcal{T}\rangle} = \{\langle t_0, e_0\rangle \xrightarrow{LightOn} \langle t_1, e_1\rangle\}$. On the other hand, C_2 realizes this trace as well as trace $\langle t_0, e_0\rangle \xrightarrow{LightOn} \langle t_1, e_1\rangle \xrightarrow{Music} \langle t_2, e_2\rangle \xrightarrow{Radio} \langle t_3, e_2\rangle \xrightarrow{Stop} \langle t_4, e_1\rangle$ (and all its prefixes). Therefore, $\Delta^{\mathsf{C}_1}_{\langle \mathcal{S},\mathcal{T}\rangle} \subset \Delta^{\mathsf{C}_2}_{\langle \mathcal{S},\mathcal{T}\rangle}$ and $\mathsf{C}_2 > \mathsf{C}_1$ holds. Other dominant controllers exist when all four behaviors are used. □

As one would expect, in solvable problem instances, the set of maximal compositions and the set of exact compositions (see Definition 3.3) coincide:

Theorem 3.2. *Let $\mathcal{S}$ be a system and $\mathcal{T}$ a target specification such that $\mathcal{T}$ has a transition-based exact controller in $\mathcal{S}$. Then,* ExactComp$(\mathcal{S},\mathcal{T})$ = MaxComp$(\mathcal{S},\mathcal{T})$.

Proof. We prove ExactComp$(\mathcal{S},\mathcal{T})$ and MaxComp$(\mathcal{S},\mathcal{T})$ are subsets of each other.

- ExactComp$(\mathcal{S},\mathcal{T}) \subseteq$ MaxComp$(\mathcal{S},\mathcal{T})$: Let C be an exact controller for $\mathcal{T}$ in $\mathcal{S}$. Now, suppose that C is not a maximal composition of $\mathcal{T}$ in $\mathcal{S}$; that is $\mathsf{C} \notin$ MaxComp$(\mathcal{S},\mathcal{T})$. Then, there must exist a controller $\mathsf{C}' \in$ MaxComp$(\mathcal{S},\mathcal{T})$ such that C' dominates C; that is $\mathsf{C}' > \mathsf{C}$. Hence, $\Delta^{\mathsf{C}}_{\langle \mathcal{S},\mathcal{T}\rangle} \subset \Delta^{\mathsf{C}'}_{\langle \mathcal{S},\mathcal{T}\rangle}$. Therefore, there exists a trace τ of enacted target $\mathcal{E}_\mathcal{T}$ such that $\tau \in \Delta^{\mathsf{C}'}_{\langle \mathcal{S},\mathcal{T}\rangle}$ and $\tau \notin \Delta^{\mathsf{C}}_{\langle \mathcal{S},\mathcal{T}\rangle}$. But this is not possible; as C is an exact controller of $\mathcal{T}$ in $\mathcal{S}$, it realizes all traces of $\mathcal{T}$; that is $\Delta^{\mathsf{C}}_{\langle \mathcal{S},\mathcal{T}\rangle} = \Delta_{\mathcal{E}_\mathcal{T}}$. Therefore, controller C' does not exist. Thus, $\mathsf{C} \in$ MaxComp$(\mathcal{S},\mathcal{T})$.

- MaxComp$(\mathcal{S},\mathcal{T}) \subseteq$ ExactComp$(\mathcal{S},\mathcal{T})$: Let C' be a maximal composition and C an exact controller for $\mathcal{T}$ in $\mathcal{S}$ (we know there is at least one). Since C' is maximal, we know that $\mathsf{C}' \geq \mathsf{C}$; that is $\Delta^{\mathsf{C}}_{\langle \mathcal{S},\mathcal{T}\rangle} \subseteq \Delta^{\mathsf{C}'}_{\langle \mathcal{S},\mathcal{T}\rangle}$. Also, since C is exact for $\mathcal{T}$ in $\mathcal{S}$, we have $\Delta^{\mathsf{C}}_{\langle \mathcal{S},\mathcal{T}\rangle} = \Delta_{\mathcal{E}_\mathcal{T}}$. Hence, $\Delta_{\mathcal{E}_\mathcal{T}} \subseteq \Delta^{\mathsf{C}'}_{\langle \mathcal{S},\mathcal{T}\rangle}$. Thus, C' is also an exact composition for $\mathcal{T}$ in $\mathcal{S}$; that is $\mathsf{C}' \in$ ExactComp$(\mathcal{S},\mathcal{T})$.

□

Indeed, maximal compositions serve as exact controllers for solvable behavior composition problems, and in cases for which a full realization is impossible, they capture the *optimal* controllers that one could hope for.

Example 3.3. Consider two maximal non-exact controllers C_1 and C_2 for target $\mathcal{T}$ in the smart home system depicted in Figure 3.1. Controller C_1 allocates all *Movie* requests to game device $\mathcal{B}_G$, whereas C_2 uses movie device $\mathcal{B}_M$ for *Movie* requests. Since target specification $\mathcal{T}$ is not solvable in the smart home system, maximal controllers C_1 and C_2 do not realize all the enacted target traces. Indeed, C_1 will not realize any trace with an action sequence where *Movie* is followed by *Radio*. Similarly, C_2 will not realize any trace where request for *Movie* is followed by a request for *Game*. Of course, other maximal controllers also exist for the target specification $\mathcal{T}$ that may utilize both game device $\mathcal{B}_G$ and movie device $\mathcal{B}_M$. □

Whereas maximal compositions provide a way of capturing optimal solutions for instances with no exact solution, they suffer from two important limitations. First, maximal compositions do not convey any useful insights on how well the instance can be solved. Even if we are given the set of traces that a maximal composition realizes, it will be difficult to reconstruct what it means in terms of the problem specification. As a consequence, using a maximal non-exact composition may yield dead-end executions where no further actions can be performed; that is, a maximal composition may become stuck. Second, the end user manages *only* the target specification; that is, the end user is the one requesting the domain actions and not the one delegating them. Hence, providing the end user with a maximal composition for the original target specification is not practically useful since without knowing the internals of the controller, the user cannot know which actions she may request next. For example, it may happen that no subsequent requests can be delegated after a particular initial request, in that case the user may decide to avoid this initial request. In comparison, for solvable problem instances, the user does not need to distinguish between transition requests since irrespective of which one she chooses, a maximal (exact) composition will always be able to honor the future requests. In short, just obtaining maximal compositions is not enough; one has to know *how* to use it because full realizability is no longer a guarantee in unsolvable problem instances.

In the next section, the most important in this thesis, we shall look at optimal solutions from a different perspective that is arguably more intuitive and computationally more amenable, than dealing with maximal controller functions.

3.3 Supremal realizable target fragments (SRTFs)

An alternative to maximal compositions, though related (as we will show later in Section 3.5), perspective is to ask what parts or aspects of a target specification $\mathcal{T}$ can indeed be brought about in system $\mathcal{S}$. For instance, which part of the target specification should be removed to render the remaining specification realizable. More concretely, we are interested in the following task:

> *Given an available system $\mathcal{S}$ and a target specification $\mathcal{T}$, find a fragment $\hat{\mathcal{T}}$ of $\mathcal{T}$ that can be fully realized in $\mathcal{S}$ (by a composition $\hat{\mathsf{C}}$ for $\hat{\mathcal{T}}$ in $\mathcal{S}$) where $\hat{\mathcal{T}}$ is "as close as possible" to the original target specification $\mathcal{T}$.*

Intuitively, a target fragment is simply a partial specification of the original target behavior. A *realizable* target fragment is a partial specification (of the given target) that is in fact solvable in the available system; that is, there exists an exact controller for it. Then, a *supremal realizable target fragment* is the largest realizable target fragment. Of course, in fully solvable problem instances, the original target and its supremal realizable target fragment will coincide.

In this section, we make these high level concepts concrete in three incremental steps:

1. We define a *target fragment*;
2. We define a *realizable target fragment* as a target fragment having an exact composition; and
3. We define a *supremal realizable target fragment* as the largest realizable target fragment.

When it comes to comparing behaviors and their fragments, simulation is an ideal choice; the preorder property of the simulation relation is well suited to defining behavioral hierarchies. To account for the shared environment, we extend the notion of simulation as follows:

Definition 3.6 (e-simulation). Given two target behaviors $\mathcal{T}_i = \langle T_i, G_i, t_{i0}, \varrho_i \rangle$, for $i \in \{1,2\}$, over an environment $\mathcal{E} = \langle E, A, e_0, \rho \rangle$, an E-*simulation* of $\mathcal{T}_1$ by $\mathcal{T}_2$ is a binary relation $Sim \subseteq T_1 \times T_2$ where $\langle t_1, t_2 \rangle \in Sim$ iff

- for all transitions $\langle t_1, a, g_1, t_1' \rangle \in \varrho_1$ there exists a transition $\langle t_2, a, g_2, t_2' \rangle \in \varrho_2$, such that for all states $e \in E$, if $g_1(e) = \mathtt{true}$ then $g_2(e) = \mathtt{true}$; and
- $\langle t_1', t_2' \rangle \in Sim$.

We say a state t_1 of $\mathcal{T}_1$ is E-simulated by a state t_2 of $\mathcal{T}_2$ in the context of environment $\mathcal{E}$, denoted by $t_1 \preceq_{\mathcal{E}} t_2$, if there exists an E-simulation relation Sim of $\mathcal{T}_1$ by $\mathcal{T}_2$ such that $\langle t_1, t_2 \rangle \in Sim$. Target behavior $\mathcal{T}_1$ is E-simulated by target behavior $\mathcal{T}_2$ in the context of environment $\mathcal{E}$, denoted by $\mathcal{T}_1 \preceq_{\mathcal{E}} \mathcal{T}_2$, if their initial states are in E-simulation; that is $t_{10} \preceq_{\mathcal{E}} t_{20}$. □

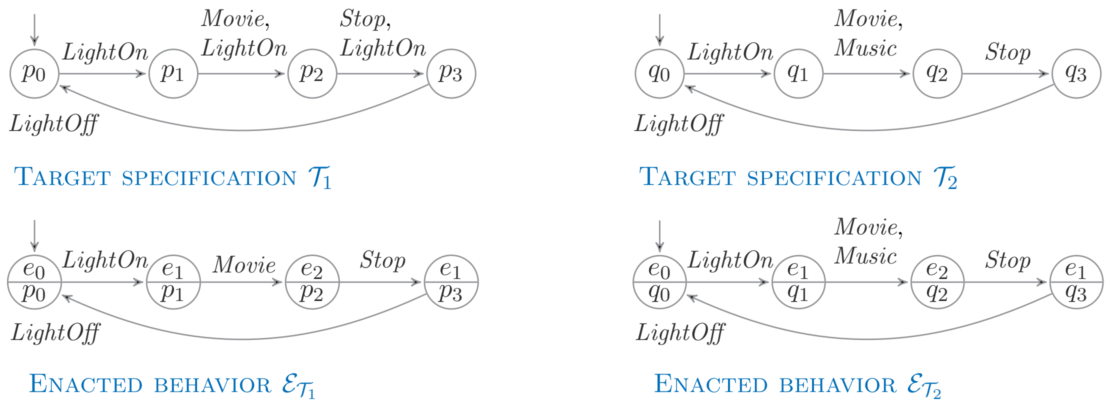

Target specification $\mathcal{T}_1$
Target specification $\mathcal{T}_2$
Enacted behavior $\mathcal{E}_{\mathcal{T}_1}$
Enacted behavior $\mathcal{E}_{\mathcal{T}_2}$
LightOn
Movie, LightOn
Stop, LightOn
LightOff
Movie, Music
Stop
Movie

Figure 3.2: Example to differentiate E-simulation between behaviors and simulated between enacted behaviors.

Similar to the standard simulation notion, $\mathcal{T}_1 \prec_{\mathcal{E}} \mathcal{T}_2$ implies $\mathcal{T}_1 \preceq_{\mathcal{E}} \mathcal{T}_2$ and $\mathcal{T}_2 \not\preceq_{\mathcal{E}} \mathcal{T}_1$; and $\mathcal{T}_1 \sim_{\mathcal{E}} \mathcal{T}_2$ implies $\mathcal{T}_1 \preceq_{\mathcal{E}} \mathcal{T}_2$ and $\mathcal{T}_2 \preceq_{\mathcal{E}} \mathcal{T}_1$. Conceptually, for two target specifications $\mathcal{T}_1$ and $\mathcal{T}_2$, $\mathcal{T}_1 \prec_{\mathcal{E}} \mathcal{T}_2$ means that *when target behaviors* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *are placed in the environment* $\mathcal{E}$, the enacted behavior of $\mathcal{T}_1$ in $\mathcal{E}$ will be subsumed by the enacted behavior of $\mathcal{T}_2$ in $\mathcal{E}$.

We use an E-simulation to define target fragments. Note that the definition of a target fragment is extremely simple.

Definition 3.7 (Target fragment). A target specification $\mathcal{T}_1$ is a *fragment* of another target specification $\mathcal{T}_2$ in the context of an environment $\mathcal{E}$ iff $\mathcal{T}_1 \preceq_{\mathcal{E}} \mathcal{T}_2$. □

Observe that Yadav and Sardiña [2012] consider target fragments for composition problems in the absence of the shared environment, hence it is enough for them to define target fragments relying only on the standard notion of simulation between target specifications. However, in the context of the environment, guards need to be considered and hence the standard notion of simulation cannot be directly applied to compare target behaviors. A naive way to extend their definition to incorporate environment is to consider simulation between enacted targets. Indeed, doing so makes the definition of target fragments simpler as guards are compiled away in the enacted behaviors. However, Definition 3.7 is not only stricter than considering simulation between enacted target behaviors, but is also more natural in the sense that defining target fragments based on enacted target behaviors may lead to counter-intuitive examples.

Example 3.4. Consider the target specifications $\mathcal{T}_1$ and $\mathcal{T}_2$ shown in Figure 3.2 for the media room scenario. Target $\mathcal{T}_1$ allows requesting of multiple *LightOn* actions, whereas $\mathcal{T}_2$ allows *LightOn* only from its initial state. As a consequence, $\mathcal{T}_1$ is not E-simulated by $\mathcal{T}_2$; that is $\mathcal{T}_1 \not\preceq_{\mathcal{E}} \mathcal{T}_2$, where $\mathcal{E}$ is the media room environment shown in Figure 3.1. Clearly, and also intuitively, $\mathcal{T}_1$ is not a fragment of the target specification $\mathcal{T}_2$. However, note that the media room environment only allows toggling the light switch; that is multiple *LightOn* actions are not allowed, as is evident in the enacted system $\mathcal{E}_{\mathcal{T}_1}$ of the target $\mathcal{T}_1$. Now, comparing the enacted systems $\mathcal{E}_{\mathcal{T}_1}$ and $\mathcal{E}_{\mathcal{T}_2}$ of respective target behaviors $\mathcal{T}_1$ and $\mathcal{T}_2$, one can check that $\mathcal{E}_{\mathcal{T}_2}$ strictly simulates $\mathcal{E}_{\mathcal{T}_1}$, as $\mathcal{E}_{\mathcal{T}_2}$ allows an extra *Music* action after turning the room lights on. Hence, if only simulation between enacted systems is considered, then $\mathcal{T}_1$ will be a fragment of $\mathcal{T}_2$. This is counter-intuitive as specification $\mathcal{T}_1$ "seems" to allow more *LightOn* actions than specification $\mathcal{T}_2$. □

The above example depicts the fineness of the E-simulation between target specifications as compared to simulation between enacted targets; we show this formally.

Theorem 3.3. *Let $\mathcal{T}_i = \langle T_i, G_i, t_{i0}, \varrho_i \rangle$, where $i \in \{1,2\}$, be two target specifications over an environment $\mathcal{E} = \langle E, A, e_0, \rho \rangle$ and $\mathcal{E}_{\mathcal{T}_i}$ be their respective enacted target behaviors. Then, $\mathcal{T}_1 \preceq_{\mathcal{E}} \mathcal{T}_2$ implies $\mathcal{E}_{\mathcal{T}_1} \preceq \mathcal{E}_{\mathcal{T}_2}$, but $\mathcal{E}_{\mathcal{T}_1} \preceq \mathcal{E}_{\mathcal{T}_2}$ and $\mathcal{T}_1 \not\preceq_{\mathcal{E}} \mathcal{T}_2$ may hold.*

Proof. We show the strictness of E-simulation in two steps.

- $\mathcal{T}_1 \preceq_{\mathcal{E}} \mathcal{T}_2$ implies $\mathcal{E}_{\mathcal{T}_1} \preceq \mathcal{E}_{\mathcal{T}_2}$: Let $\mathcal{E}_{\mathcal{T}_i} = \langle S_i, A, s_{i0}, \delta_i \rangle$ for $i \in \{1,2\}$ be the respective enacted target behaviors of $\mathcal{T}_1$ and $\mathcal{T}_2$ over environment $\mathcal{E}$. We define a function $f : S_1 \to S_2$ where $f(s_1) = s_2$ such that $\mathsf{env}(s_1) = \mathsf{env}(s_2)$ and $\mathsf{tgt}(s_1) \preceq_{\mathcal{E}} \mathsf{tgt}(s_2)$. That is, f takes an enacted target state s_1 of $\mathcal{E}_{\mathcal{T}_1}$ and returns a corresponding enacted target state s_2 of $\mathcal{E}_{\mathcal{T}_2}$ such that s_1 and s_2 are in the same environment state and their respective target states are in E-simulation. Note, there may be multiple such functions as a state in $\mathcal{T}_1$ could be E-simulated by multiple states in $\mathcal{T}_2$. We nondeterministically pick one such f (there is at least one). Now, consider a relation $\mathcal{R} \subseteq S_1 \times S_2$ such that $\langle s_1, s_2 \rangle \in \mathcal{R}$ iff $s_2 = f(s_1)$. We show that $\mathcal{R}$ is a simulation relation of $\mathcal{E}_{\mathcal{T}_1}$ by $\mathcal{E}_{\mathcal{T}_2}$. Consider a tuple $\langle s_1, s_2 \rangle \in \mathcal{R}$:

  1. If there is no transition $\langle s_1, a, s_1' \rangle$ in $\mathcal{E}_{\mathcal{T}_1}$, then $\langle s_1, s_2 \rangle$ trivially obeys the simulation requirement.
  2. If there exists a transition $\langle s_1, a, s_1' \rangle$ in $\mathcal{E}_{\mathcal{T}_1}$, then there exists:
     - A guard $g_1 \in G_1$ such that $g_1(\mathsf{env}(s_1)) = \mathtt{true}$ and $\langle \mathsf{tgt}(s_1), g_1, a, t_1' \rangle$ in $\mathcal{T}_1$; that is action a can be executed from state $\mathsf{tgt}(s_1)$ of target $\mathcal{T}_1$ when environment is in state $\mathsf{env}(s_1)$; and
     - A transition $\langle \mathsf{env}(s_1), a, e' \rangle$ in $\mathcal{E}$; that is the environment itself allows the execution of action a.

  Since $\mathsf{tgt}(s_1) \preceq_{\mathcal{E}} \mathsf{tgt}(s_2)$, there exists a transition $\langle \mathsf{tgt}(s_2), g_2, a, t_2' \rangle$ such that $g_2(\mathsf{env}(s_2)) = \mathtt{true}$ and $t_1' \preceq_{\mathcal{E}} t_2'$. Moreover, as $\mathsf{env}(s_1) = \mathsf{env}(s_2)$, there exists a transition $\langle s_2, a, s_2' \rangle$ in $\mathcal{E}_{\mathcal{T}_2}$ with $\mathsf{env}(s_2') = e'$ and $\mathsf{tgt}(s_2') = t_2'$. Hence, $s_2' = f(s_1')$ and so $\langle s_1', s_2' \rangle \in \mathcal{R}$. Therefore, $\mathcal{R}$ is a simulation relation of $\mathcal{E}_{\mathcal{T}_1}$ by $\mathcal{E}_{\mathcal{T}_2}$.

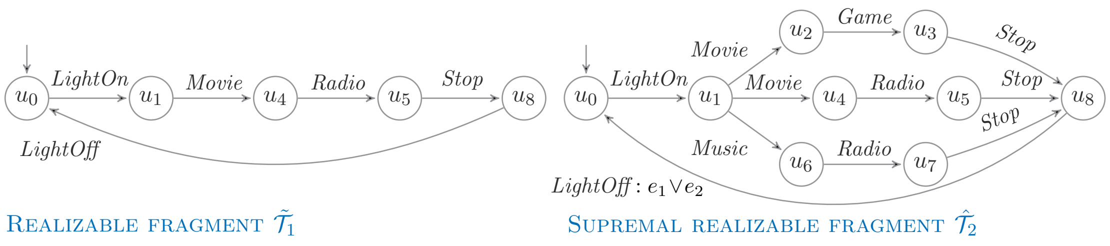


Figure 3.3: Realizable target fragments for the ambient spaces example.

- $\mathcal{E}_{\mathcal{T}_1} \preceq \mathcal{E}_{\mathcal{T}_2}$ and $\mathcal{T}_1 \not\preceq_{\mathcal{E}} \mathcal{T}_2$: Example 3.4 shows such an instance.

$\square$

In the enacted target behavior, the environment restricts the target specification to only allowable transitions. Hence, if we rely on simulation between enacted target behaviors to define target fragments, a target fragment will be legally allowed to contain inoperative transitions in the context of the given environment as such transitions will be compiled away in their respective enacted behaviors.

Example 3.5. Compare the target specifications $\mathcal{T}$, $\tilde{\mathcal{T}}$, and $\hat{\mathcal{T}}$ shown in Figures 3.1 (page 40) and 3.3, respectively. Observe that, in the media room environment $\mathcal{E}$, $\hat{\mathcal{T}}$ is a fragment of $\mathcal{T}$ ($\hat{\mathcal{T}} \prec_{\mathcal{E}} \mathcal{T}$), and $\tilde{\mathcal{T}}$ is a fragment of both $\hat{\mathcal{T}}$ and $\mathcal{T}$ ($\tilde{\mathcal{T}} \prec_{\mathcal{E}} \hat{\mathcal{T}} \prec_{\mathcal{E}} \mathcal{T}$). Moreover, technically, $\mathcal{T}$ is a fragment of itself ($\mathcal{T} \preceq_{\mathcal{E}} \mathcal{T}$). $\square$

An interesting issue arises if the original target specification (part of the problem input) has transitions which can never be executed in the given environment; fragments of such a target may still have these environment incompatible elements. Note that *classical behavior composition does not differentiate between target specifications having such inoperative transitions.* For example, both target specifications $\mathcal{T}_1$ and $\mathcal{T}_2$ shown in Figure 3.2 have an exact controller in the media room system $\mathcal{S} = \langle \mathcal{B}_G, \mathcal{B}_A, \mathcal{B}_M, \mathcal{B}_L, \mathcal{E} \rangle$. However, interestingly, in the target specification $\mathcal{T}_1$, the end user cannot request the *LightOn* action from state p_1. The reason for this seemingly counter-intuitive notion lies in the problem perspective taken. Classical behavior composition views the problem from the controller perspective; that is, it answers the following question:

> *Given a system $\mathcal{S}$ and a target specification $\mathcal{T}$, does there exist an exact controller C for $\mathcal{T}$ in $\mathcal{S}$?*

As we have already seen, an exact controller (see Definition 3.3) is only interested in the target traces that can actually occur (in the given environment), and as a result, the controller is not affected even if the target specification has transitions which will never occur in the context of the environment. In comparison, if we take the perspective of the user agent using the target specification, then the agent must be able to request any transition as per the specification if it accommodates an exact controller. However, this may not be the case in classical composition as we saw in the example above, where the specification allowed two consecutive *LightOn* actions but the environment disallowed them.

This, of course, is not a shortcoming of classical behavior composition; there the target specification to be realized is an *input* to the problem. That is, the end user gives the specification that she wants to be realized and so the specification is assumed to be meaningful in the given (fully observable) environment. On the other hand, when optimising an unsolvable problem instance, a realizable target specification is an *output* of the problem. Therefore, one needs to get rid of impossible transitions of the target. Specially, in cases when the target specification is not realizable and the end user is given a realizable fragment of the original specification, that fragment ought to be such that the user of the fragment can request all what is given to her. In short, the question which we are interested in is this:

> *Given a system $\mathcal{S}$ and a target specification $\mathcal{T}$, which is the largest fragment of $\mathcal{T}$ that can be fully realized in $\mathcal{S}$.*

To ensure that the realizable fragments of $\mathcal{T}$ do not have any parts incompatible with the environment, we introduce the notion of *effective* fragments in the context of the environment. Intuitively, a fragment is effective if each of its traces is matched by some trace of the enacted system; that is, each target trace can be legally executed in the environment. In order to formally define effective target fragments, we need a technical notion of projecting environment guards from the target behavior. Let $\mathcal{T} = \langle T, G, t_0, \varrho \rangle$ be a target specification over an environment $\mathcal{E} = \langle E, A, t_0, \rho \rangle$. The transition system obtained by projecting out $\mathcal{T}$'s guards, denoted by $\mathcal{T}^{\uparrow G}$, is defined as $\mathcal{T}^{\uparrow G} = \langle T, A, t_0, \varrho^{\uparrow G} \rangle$, where $\varrho^{\uparrow G} = \{\langle t, a, t' \rangle \mid \langle t, a, g, t' \rangle \in \varrho\}$. Intuitively, we say that a target specification $\mathcal{T}$ is effective in the environment $\mathcal{E}$ if the transition system obtained by projecting out $\mathcal{T}$'s guards (that is, $\mathcal{T}^{\uparrow G}$) is simulated by the enacted behavior of $\mathcal{T}$ in $\mathcal{E}$.

Definition 3.8 (Effective target fragment). Given a target specification $\mathcal{T} = \langle T, G, t_0, \varrho \rangle$ over an environment $\mathcal{E} = \langle E, A, t_0, \rho \rangle$, we say $\mathcal{T}$ is <u>*effective*</u> in $\mathcal{E}$ iff $\mathcal{T}^{\uparrow G} \preceq \mathcal{E}_{\mathcal{T}}$, where $\mathcal{E}_{\mathcal{T}}$ is the enacted behavior of target $\mathcal{T}$ in environment $\mathcal{E}$. □

Example 3.6. Consider the target specification $\mathcal{T}_1$ shown in Figure 3.2 operating in the media room environment $\mathcal{E}$. If we compare $\mathcal{T}^{\uparrow G}$ and the enacted target $\mathcal{E}_{\mathcal{T}_1}$, trivially, $\mathcal{T}^{\uparrow G} \not\preceq \mathcal{E}_{\mathcal{T}_1}$. As evidence, see that the trace $p_0 \xrightarrow{LightOn} p_1 \xrightarrow{LightOn} p_2$ of $\mathcal{T}^{\uparrow G}$ cannot be matched by any trace of $\mathcal{E}_{\mathcal{T}_1}$. Hence, $\mathcal{T}_1$ is not an effective specification in the context of the environment $\mathcal{E}$. □

We now have all the required technical machinery to introduce the main notion of this chapter, namely, realizable target fragments and supremal realizable target fragments.

Definition 3.9 (Realizable target fragment (RTF)). A *realizable target fragment (RTF)* $\tilde{\mathcal{T}}$ of target $\mathcal{T} = \langle T, G, t_0, \varrho \rangle$ in system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ with environment $\mathcal{E} = \langle E, A, e_0, \rho \rangle$ is a tuple $\tilde{\mathcal{T}} = \langle \tilde{T}, \tilde{G}, \tilde{t_0}, \tilde{\varrho} \rangle$, where:

1. $\tilde{\mathcal{T}} \preceq_{\mathcal{E}} \mathcal{T}$; that is, $\tilde{\mathcal{T}}$ is a fragment of target $\mathcal{T}$;
2. $\tilde{\mathcal{T}}$ is *effective* in $\mathcal{E}$; that is, $\tilde{\mathcal{T}}$ does not contain environment incompatible parts; and
3. $\tilde{\mathcal{T}}$ has an exact composition in $\mathcal{S}$; that is, $\tilde{\mathcal{T}}$ can be realized (solved) in system $\mathcal{S}$.

$\square$

Despite being fully solvable, an RTF will generally provide "*less*" than the original target specification. First, an RTF may be missing certain executions altogether. In the smart house scenario, RTF $\tilde{\mathcal{T}}$ does not account for the action sequence *LightOn* · *Music* · *Game* · *Stop* · *LightOff*. Second, and more interesting, an RTF may require the user to commit earlier to future possible request choices. In that sense, a user of target $\hat{\mathcal{T}}$ needs to decide, when requesting *Movie* in state u_1, if she will later play a *Game* or listen to *Radio*. Notice such extra "temporal" information is not required at state t_1 in the original target $\mathcal{T}$. It is exactly to accommodate this feature that we have departed from the standard view of deterministic target specifications and allowed nondeterministic specifications.

Trivially, the smallest RTF for any problem instance is the "empty" one; that is, an RTF with just one state and no transitions. Obviously, between full target specification and the trivial empty one, there lies a spectrum of RTFs. Among these, we are interested in those that are "closest" to the original target, in that the minimum possible is given up and remain realizable. In other words, a "supremal" RTF is one which cannot be further extended.

Definition 3.10 (Supremal realizable target fragment (SRTF)). A target behavior $\hat{\mathcal{T}}$ is an *supremal realizable target fragment (SRTF)* of target $\mathcal{T}$ on system $\mathcal{S}$ *iff*:

1. $\hat{\mathcal{T}}$ is a *realizable* fragment (RTF) of $\mathcal{T}$ in $\mathcal{S}$; and
2. there is no RTF $\hat{\mathcal{T}}'$ of $\mathcal{T}$ in $\mathcal{S}$ such that $\hat{\mathcal{T}} \prec_{\mathcal{E}} \hat{\mathcal{T}}'$; that is, $\mathcal{T}$ cannot be optimized any further than $\hat{\mathcal{T}}$ by a strictly more general target fragment.

$\square$

That is, a supremal fragment is one such that there is no other realizable fragment which accounts for a larger specification.

Example 3.7. Returning to the target specifications $\tilde{\mathcal{T}}$, $\hat{\mathcal{T}}$, and $\mathcal{T}$ in the ambient spaces example shown in Figures 3.1 and 3.3, $\tilde{\mathcal{T}}$ and $\hat{\mathcal{T}}$ are realizable in the media room, whereas $\mathcal{T}$ is unrealizable. Both $\tilde{\mathcal{T}}$ and $\hat{\mathcal{T}}$ target specifications are realizable fragments of $\mathcal{T}$. However, $\hat{\mathcal{T}}$ is the supremal realizable behavior of $\mathcal{T}$ in the media room system. □

3.4 Uniqueness of SRTFs

In the previous section, we saw that multiple distinct RTFs may exist for a given behavior composition problem. What remains to be seen is whether these RTFs can be combined in some way to generate more general realizable specifications. More importantly, one would like to know if there exists a single *unique* SRTF (for a given problem instance), or similar to RTFs one has to search for multiple distinct SRTFs. As one will observe, allowing nondeterminism in the target specification is key to these issues. We start by presenting an interesting property of RTFs: a union of two RTFs is an RTF. In other words, the set of RTFs are closed under RTF union. To that end, let us first define what such a union amounts to.

Definition 3.11 (Union of two RTFs). Let $\mathcal{T}_i = \langle T_i, G_i, t_{i0}, \varrho_i\rangle$, where $i \in \{1, 2\}$, be two target specifications over a given environment $\mathcal{E} = \langle E, A, e_0, \rho\rangle$ such that T_1 and T_2 are disjoint sets of states (i.e., $T_1 \cap T_2 = \emptyset$).[2] Then, the target specification resulting from the <u>*union*</u> of $\mathcal{T}_1$ and $\mathcal{T}_2$ is given by the tuple $\mathcal{T}_1 \cup \mathcal{T}_2 = \langle T_{1+2}, G_{1+2}, t_0, \varrho_{1+2}\rangle$ such that $t_0 \notin T_1$ and $t_0 \notin T_2$, where:

- $T_{1+2} = T_1 \cup T_2 \cup \{t_0\}$;
- $G_{1+2} = G_1 \cup G_2$;
- $t_0 \in T$ is the initial state;
- $\varrho_{1+2} = \varrho_1 \cup \varrho_2 \cup \{\langle t_0, g, a, t'\rangle \mid \langle t_{10}, g, a, t'\rangle \in \varrho_1\} \cup \{\langle t_0, g, a, t'\rangle \mid \langle t_{20}, g, a, t'\rangle \in \varrho_2\}$ is the transition relation.

□

Conceptually, when combining two target specifications $\mathcal{T}_1$ and $\mathcal{T}_2$, one creates a new initial state and adds transitions from this initial state to states that can be reached by a single action from initial states of $\mathcal{T}_1$ and $\mathcal{T}_2$.

Example 3.8. Figure 3.4 shows two target specifications, $\mathcal{T}_1$ and $\mathcal{T}_2$, and their union, $\mathcal{T}_1 \cup \mathcal{T}_2$, for the media room scenario. Observe that we create a new initial state t_0 in the union $\mathcal{T}_1 \cup \mathcal{T}_2$ and all the transitions of $\mathcal{T}_1$ and $\mathcal{T}_2$ are same in $\mathcal{T}_1 \cup \mathcal{T}_2$ except for additional transitions from state t_0. From the initial state t_0, transition $t_0 \xrightarrow{LightOn} u_1$ takes the user

[2]The states can be renamed wlog to ensure T_1 and T_2 are disjoint.

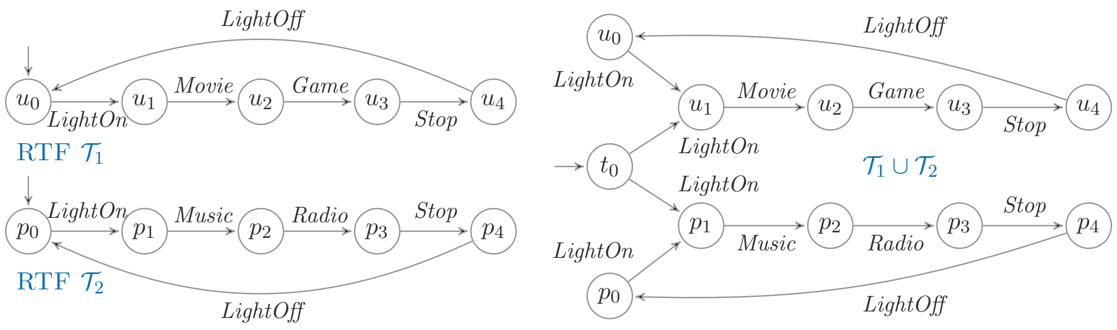


Figure 3.4: Union of two target specifications.

to target specification $\mathcal{T}_1$ whereas transition $t_0 \xrightarrow{LightOn} p_1$ takes the user to specification $\mathcal{T}_2$. Observe that $\mathcal{T}_1$ and $\mathcal{T}_2$ are both E-simulated by their union $\mathcal{T}_1 \cup \mathcal{T}_2$, but the opposite is not true; that is $\mathcal{T}_1 \cup \mathcal{T}_2 \not\preceq_{\mathcal{E}} \mathcal{T}_1$ and $\mathcal{T}_1 \cup \mathcal{T}_2 \not\preceq_{\mathcal{E}} \mathcal{T}_2$. □

Observe that given two target specifications $\mathcal{T}_1$ and $\mathcal{T}_2$, their union $\mathcal{T}_1 \cup \mathcal{T}_2$ will simulate both $\mathcal{T}_1$ and $\mathcal{T}_2$. As evident from the transition relation of $\mathcal{T}_1 \cup \mathcal{T}_2$, the new initial state t_0 mimics the behavior of initial states of $\mathcal{T}_1$ and $\mathcal{T}_2$. For all transitions $\langle t_{10}, g, a, t' \rangle$ in $\mathcal{T}_1$, by definition, $\mathcal{T}_1 \cup \mathcal{T}_2$ has a transition $\langle t_0, g, a, t' \rangle$, hence $t_{10} \preceq t_0$. By similar reasoning we can get that $t_{20} \preceq t_0$.

We now formally prove that the union of two RTFs is also an RTF.

Theorem 3.4. *Let $\tilde{\mathcal{T}}_1$ and $\tilde{\mathcal{T}}_2$ be two RTFs for target specification $\mathcal{T}$ in system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$. Then $\tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2$ is an RTF of $\mathcal{T}$ in $\mathcal{S}$.*

Proof. We show that the result of a RTF union, as defined above, adheres to the RTF definition (see Definition 3.9). Let $\mathcal{T}_i = \langle T_i, G_i, t_{i0}, \varrho_i \rangle$, where $i \in \{1, 2\}$, be two RTFs of target specification $\mathcal{T} = \langle T, G, t_0, \varrho \rangle$ in system $\mathcal{S}$ and $\mathcal{T}_1 \cup \mathcal{T}_2 = \langle T_{1+2}, G_{1+2}, t_0^+, \varrho_{1+2} \rangle$.

- $\mathcal{T}_1 \cup \mathcal{T}_2 \preceq_{\mathcal{E}} \mathcal{T}$: We define a relation $\mathcal{R} \subseteq (T_{1+2} \times T)$ such that $\langle t', t \rangle \in \mathcal{R}$ iff *(i)* $t' \preceq_{\mathcal{E}} t$, where t' is a state in $\mathcal{T}_1$ or $\mathcal{T}_2$, or *(ii)* $t' = t_0^+$ and $t = t_0$. Note, $\mathcal{R}$ is not empty since we know that $\mathcal{T}_1 \preceq_{\mathcal{E}} \mathcal{T}$ and $\mathcal{T}_2 \preceq_{\mathcal{E}} \mathcal{T}$. In addition, for all states t' such that $t_0^+ \xrightarrow{g,a} t'$ in $\mathcal{T}_1 \cup \mathcal{T}_2$, there exists a state $t \in T$ such that $t' \preceq t$. Moreover, by definition, $\mathcal{R}$ is an E-simulation relation of $\mathcal{T}_1 \cup \mathcal{T}_2$ by $\mathcal{T}$ with $\langle t_0^+, t_0 \rangle \in \mathcal{R}$.

- $\mathcal{T}_1 \cup \mathcal{T}_2 \preceq_{\mathcal{E}} \mathcal{T}$ is *effective* in $\mathcal{E}$: Since $\mathcal{T}_1$ and $\mathcal{T}_2$ are RTFs of $\mathcal{T}$ in $\mathcal{S}$, $\mathcal{T}_1$ and $\mathcal{T}_2$ are effective in $\mathcal{E}$. Hence, $\mathcal{T}_1^{\uparrow G} \preceq \mathcal{E}_{\mathcal{T}_1}$ and $\mathcal{T}_2^{\uparrow G} \preceq \mathcal{E}_{\mathcal{T}_2}$. Suppose $\mathcal{R}_1$ and $\mathcal{R}_2$ are simulation relations between $\mathcal{T}_1^{\uparrow G}$–$\mathcal{E}_{\mathcal{T}_1}$ and $\mathcal{T}_2^{\uparrow G}$–$\mathcal{E}_{\mathcal{T}_2}$, respectively. Then, by definition, $\mathcal{R}_1 \cup \mathcal{R}_2 \cup \{\langle t_0^+, (t_0^+, e_0) \rangle\}$ is a simulation relation of $\mathcal{T}_1 \cup \mathcal{T}_2$ by $\mathcal{E}_{(\mathcal{T}_1 \cup \mathcal{T}_2)}$. Therefore, $\mathcal{T}_1 \cup \mathcal{T}_2$ is effective in $\mathcal{E}$.

- $\mathcal{T}_1 \cup \mathcal{T}_2$ has an exact composition in $\mathcal{S}$: Let C_1 and C_2 be exact compositions of $\mathcal{T}_1$ and $\mathcal{T}_2$ in $\mathcal{S}$, respectively. We define a controller C_{1+2} as follows:

$$\mathsf{C}_{1+2}(h, t_1 \xrightarrow{g,a} t_2) = \begin{cases} \mathsf{C}_1(h, t_1 \xrightarrow{g,a} t_2) & \text{if } t_1 = t_0 \text{ and } t_1 \in T_1 \\ \mathsf{C}_2(h, t_1 \xrightarrow{g,a} t_2) & \text{if } t_1 = t_0 \text{ and } t_2 \in T_2 \\ \mathsf{C}_1(h, t_1 \xrightarrow{g,a} t_2) & \text{if } t_1 \xrightarrow{g,a} t_2 \text{ in } \mathcal{T}_1 \\ \mathsf{C}_2(h, t_1 \xrightarrow{g,a} t_2) & \text{otherwise.} \end{cases}$$

The controller C_{1+2} delegates the transition requests arising from $\mathcal{T}_1$ as per C_1 and transition requests arising from $\mathcal{T}_2$ as per C_2. Since C_1 and C_2 realize all respective enacted traces of $\mathcal{T}_1$ and $\mathcal{T}_2$, C_{1+2} realizes all enacted traces of $\mathcal{T}_1 \cup \mathcal{T}_2$. Hence, C_{1+2} is an exact composition of $\mathcal{T}_1 \cup \mathcal{T}_2$ in $\mathcal{S}$.

□

Union of RTFs can be seen as the addition operation, and in fact, similar to addition, the union of RTFs is *commutative*, *associative*, and has an *identity element.*

Theorem 3.5. *Let $\tilde{\mathcal{T}}_1$ and $\tilde{\mathcal{T}}_2$ be two RTFs for target specification $\mathcal{T}$ in system $\mathcal{S}$. Then:*

1. *$\tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2 \sim_{\mathcal{E}} \tilde{\mathcal{T}}_2 \cup \tilde{\mathcal{T}}_1$; that is, union of RTFs is commutative.*
2. *$(\tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2) \cup \tilde{\mathcal{T}}_3 = \tilde{\mathcal{T}}_1 \cup (\tilde{\mathcal{T}}_2 \cup \tilde{\mathcal{T}}_3)$; that is, union of RTFs is associative.*

Proof. We prove both the properties below:

1. Let $\tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2 = \langle T_{1+2}, G_{1+2}, t_0, \varrho_{1+2} \rangle$, $\tilde{\mathcal{T}}_2 \cup \tilde{\mathcal{T}}_1 = \langle T_{2+1}, G_{2+1}, t_0, \varrho_{2+1} \rangle$ and $\mathcal{R} \subseteq T_{1+2} \times T_{2+1}$ be a relation such that $\langle t_{1+2}, t_{2+1} \rangle \in \mathcal{R}$ iff $t_{1+2} = t_{2+1}$. Trivially, $\mathcal{R}$ is a E-simulation relation of $\tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2$ by $\tilde{\mathcal{T}}_2 \cup \tilde{\mathcal{T}}_1$. Following same lines of reasoning a relation $\mathcal{R}$ will be an E-simulation relation of $\tilde{\mathcal{T}}_2 \cup \tilde{\mathcal{T}}_1$ by $\tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2$.
2. Let us build the RTF $(\tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2) \cup \tilde{\mathcal{T}}_3$ of $\mathcal{T}$ in $\mathcal{S}$. Resolving $\tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2$ we get $(\tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2) \cup \tilde{\mathcal{T}}_3 = \mathcal{T}_{1+2} \cup \mathcal{T}_3$, where $\mathcal{T}_{1+2} = \langle T_{1+2}, G_{1+2}, t_0', \varrho_{1+2} \rangle$ is built as per the RTF union definition. Similarly, $\mathcal{T}_{1+2} \cup \mathcal{T}_3 = \langle T_{1+2+3}, G_{1+2+3}, t_0, \varrho_{1+2+3} \rangle$. Following the same steps, $\tilde{\mathcal{T}}_1 \cup (\tilde{\mathcal{T}}_2 \cup \tilde{\mathcal{T}}_3) = \mathcal{T}_1 \cup \langle T_{2+3}, G_{2+3}, t_0', \varrho_{2+3} \rangle = \langle T_{1+2+3}, G_{1+2+3}, t_0, \varrho_{1+2+3} \rangle$.

□

Hence, RTFs can be combined in any order. To show the existence of an identity element, it suffices to state that $\tilde{\mathcal{T}} \cup \tilde{\mathcal{T}}_0$ is simulation equivalent to $\tilde{\mathcal{T}}$, where $\tilde{\mathcal{T}}_0 = \langle \{t\}, \emptyset, t, \emptyset \rangle$. Observe that $\tilde{\mathcal{T}}_0$ is an RTF of *any* target specification $\mathcal{T}$ in *any* system $\mathcal{S}$; in fact, it is the *smallest* universal RTF.

Importantly, RTFs, partial target specifications which can be fully implemented in the given system, can be combined generate more *general* target specifications. Technically, the union of two RTFs will e-simulate (Definition 3.6) both the individual RTFs.

Theorem 3.6. *Let $\tilde{\mathcal{T}}_1$ and $\tilde{\mathcal{T}}_2$ be two RTFs for target specification $\mathcal{T}$ in a system $\mathcal{S}$. Then $\tilde{\mathcal{T}}_1 \preceq_{\mathcal{E}} \tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2$.*

Proof. Let $\tilde{\mathcal{T}}_1 = \langle T_1, G_1, t_{10}, \varrho_1 \rangle$, $\tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2 = \langle T_{1+2}, G_{1+2}, t_0, \varrho_{1+2} \rangle$ and $\mathcal{R} \subseteq T_{1+2} \times T_1$ be a relation such that $\langle t_{1+2}, t_1 \rangle \in \mathcal{R}$ iff $t_{1+2} = t_1$ or $t_{1+2} = t_0$ and $t_1 = t_{10}$. Since $\tilde{\mathcal{T}}_1$ is fully contained in $\tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2$; that is $T_1 \subseteq T_{1+2}$, $\varrho_1 \subseteq \varrho_{1+2}$ and $t_{10} \preceq t_0$, $\mathcal{R}$ is an E-simulation relation of $\tilde{\mathcal{T}}_1$ by $\tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2$. $\square$

Observe that since union of RTFs is an associative operation, it automatically implies that if $\tilde{\mathcal{T}}_1$ and $\tilde{\mathcal{T}}_2$ are two RTFs for target specification $\mathcal{T}$ in a system $\mathcal{S}$, then $\tilde{\mathcal{T}}_2 \preceq_{\mathcal{E}} \tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2$. Usually, the resulting union of two RTFs will be a *better* RTF as compared to the individual RTFs with respect to the original target specification and the available system. Formally, given two RTFs $\tilde{\mathcal{T}}_1$ and $\tilde{\mathcal{T}}_2$ of target specification $\mathcal{T}$ in a system $\mathcal{S}$, $\tilde{\mathcal{T}}_1 \cup \tilde{\mathcal{T}}_2 \not\preceq_{\mathcal{E}} \tilde{\mathcal{T}}_1$ may hold (see Example 3.8 for such an instance). Since RTFs are closed under union (Theorem 3.4), one can combine RTFs to build *optimal* RTFs.

Theorem 3.7. *Let $\mathcal{S}$ be a system, $\mathcal{T}$ a target specification, and $\mathcal{T}^* = \bigcup_{\tilde{\mathcal{T}} \text{ is an RTF of } \mathcal{T} \text{ in } \mathcal{S}} \tilde{\mathcal{T}}$. Then, $\mathcal{T}^*$ is an SRTF of target specification $\mathcal{T}$ in systen $\mathcal{S}$.*

Proof. Assume that $\mathcal{T}^*$ is not a SRTF of $\mathcal{T}$ in $\mathcal{S}$ and suppose $\hat{\mathcal{T}}$ is a SRTF of $\mathcal{T}$ in $\mathcal{S}$. From Theorem 3.4 we know that $\mathcal{T}^*$ is a RTF of $\mathcal{T}$ in $\mathcal{S}$. Interestingly, note that by definition $\hat{\mathcal{T}}$ is also a RTF of $\mathcal{T}$ in $\mathcal{S}$, therefore $\hat{\mathcal{T}}$ is contained in $\mathcal{T}^*$. Applying Theorem 3.6 we get that $\hat{\mathcal{T}} \preceq_{\mathcal{E}} \mathcal{T}^*$, which is a contradiction. Therefore, $\mathcal{T}^*$ is a SRTF of $\mathcal{T}$ in $\mathcal{S}$. $\square$

A surprising and much desired consequence of Theorem 3.7 is that any $\tilde{\mathcal{T}}$ which is E-simulation equivalent to $\mathcal{T}^*$ is also "the" SRTF (we focus on semantics not syntax here). It follows then that SRTFs are unique up to E-simulation equivalence. Indeed, there might be two SRTFs that are syntactically (structurally) different, however, if they are simulation equivalent then they express (semantically) the same target specification.

Theorem 3.8. *SRTFs are unique up to E-simulation equivalence.*

Proof. Let $\hat{\mathcal{T}}$ be a SRTF of target specification $\mathcal{T}$ in system $\mathcal{S}$. From Theorem 3.7 we know that $\mathcal{T}^*$, union of all RTFs, is a SRTF of $\mathcal{T}$ in $\mathcal{S}$. Therefore, $\hat{\mathcal{T}} \preceq_{\mathcal{E}} \mathcal{T}^*$. Suppose that $\mathcal{T}^*$ is not E-simulated by $\hat{\mathcal{T}}$; that is $\mathcal{T}^* \not\preceq_{\mathcal{E}} \hat{\mathcal{T}}$. Then, it follows that $\hat{\mathcal{T}} \prec_{\mathcal{E}} \mathcal{T}^*$. An absurd since we know $\hat{\mathcal{T}}$ is a SRTF of $\mathcal{T}$ in $\mathcal{S}$. Hence our assumption is wrong and $\mathcal{T}^*$ is indeed E-simulated by $\hat{\mathcal{T}}$. Consequently, it holds that $\hat{\mathcal{T}}$ and $\mathcal{T}^*$ are E-simulation equivalent. $\square$

Undoubtedly, uniqueness of SRTFs is an important desirable property as the target specification's user can be given a single (optimised) SRTF, and she can be sure that this is *the* best possible specification that can be realized.

3.5 Imported controllers

In contrast with maximal controllers, supremal fragments are specified in the *same* language as the original problem. The user can thus decide to request actions as per the new target fragment with guaranteed full realizability. Nonetheless, one may still ask in which sense these solutions are "correct." To answer that, we show that using an exact composition for a supremal target behavior amounts to using a maximal composition for the original target. In other words, if the user wants to request actions as per the original target specification, then a maximal composition can be computed using SRTF's exact composition. Once a supremal target behavior $\hat{\mathcal{T}}$ is obtained, one may either use such new target directly or consider "importing" its exact compositions into the original target module $\mathcal{T}$. Hopefully, in the latter case, the imported controllers will turn out to be the best possible controllers for the original target; that is, a maximal composition. To that end, we define what it means to "import" a controller $\mathsf{C}_{\mathcal{T}'}$ designed for one target specification $\mathcal{T}'$ into another target specification $\mathcal{T}$.

We start by defining the family of functions that are meant to explain sequences of action requests in a target. Informally, the function $\mathsf{Expl}_{\mathcal{T}}(\sigma)$ outputs a history of the target $\mathcal{T}$ compatible with the given sequence of actions σ. Formally, a function $\mathsf{Expl}_{\mathcal{T}} : A^* \mapsto \mathcal{H}_{\mathcal{T}}$ is a <u>*target explanatory*</u> function for a target $\mathcal{T}$ if for any action sequence $\sigma = a^1 \cdot \dots \cdot a^\ell \in A^*$, with $\ell \geq 0$, it is the case that $\mathsf{Expl}_{\mathcal{T}}(\sigma) = t^0 \xrightarrow{g^1,a^1} \cdots \xrightarrow{g^\ell,a^\ell} t^\ell \in \mathcal{H}_{\mathcal{T}}$, where $\mathcal{H}_{\mathcal{T}}$ is the set of $\mathcal{T}$'s histories. In general, there will be many of such functions, since the same sequence of action requests can arise from different runs of a nondeterministic target. For instance, sequence $\mathit{LightOn} \cdot \mathit{Movie}$ can be explained in two ways on target specification $\hat{\mathcal{T}}$ (see the Figure 3.3), namely, via histories $u_0 \xrightarrow{\mathit{LightOn}} u_1 \xrightarrow{\mathit{Movie}} u_2$ and $u_0 \xrightarrow{\mathit{LightOn}} u_1 \xrightarrow{\mathit{Movie}} u_4$.

Using target explanatory functions, we next characterize the set of so-called *induced* controllers. Suppose we have a controller $\mathsf{C}_{\mathcal{T}'}$ for a target $\mathcal{T}'$ (in a system $\mathcal{S}$). An induced controller (from controller $\mathsf{C}_{\mathcal{T}'}$) for a target behavior $\mathcal{T}$ is one that handles requests from $\mathcal{T}$ *as if they were requests issued as per specification $\mathcal{T}'$*. Recall that a controller for a system $\mathcal{S}$ outputs the behavior index to which a given transition-action request is delegated to at a certain system history.

Definition 3.12 (Induced controller). A controller $\mathsf{C}^{\mathcal{T}'}_{\mathcal{T}}$ is an <u>*induced controller*</u> (from controller $\mathsf{C}_{\mathcal{T}'}$ on target $\mathcal{T}'$) for target $\mathcal{T}$ over system $\mathcal{S}$ if there exists a target explanatory function $\mathsf{Expl}_{\mathcal{T}'}(\cdot)$ for $\mathcal{T}'$ such that for every system history $h \in \mathcal{H}_{\mathcal{S}}$ and transition $t_1 \xrightarrow{g,a} t_2$ in $\mathcal{T}$, the following holds (recall that $h^{\uparrow S}$ denotes the sequence of actions in history h):

$$\mathsf{C}^{\mathcal{T}'}_{\mathcal{T}}(h, t_1 \xrightarrow{g,a} t_2) = \begin{cases} \mathsf{C}_{\mathcal{T}'}(h, t_1' \xrightarrow{g',a} t_2') & \mathsf{Expl}_{\mathcal{T}'}(h^{\uparrow S} \cdot a) = t^0 \xrightarrow{g'^1,a^1} \cdots \xrightarrow{g'^{|h|},a^{|h|}} t_1' \xrightarrow{g',a} t_2' \\ \text{undefined} & \mathsf{Expl}_{\mathcal{T}'}(h^{\uparrow S} \cdot a) \text{ is undefined} \end{cases}$$

□

That is, $\mathcal{T}$'s request $t_1 \xrightarrow{g,a} t_2$ is delegated at history h as controller $\mathsf{C}_{\mathcal{T}'}$ would delegate request $t'_1 \xrightarrow{g',a} t'_2$ from target $\mathcal{T}'$ if h's request leaves target $\mathcal{T}'$ in state t'_1 and the current requested action a is indeed explained by transition request $t'_1 \xrightarrow{g',a} t'_2$ in $\mathcal{T}'$. When there is no explanation in the $\mathcal{T}'$ for h—$\mathsf{Expl}(\cdot)$ is undefined—the induced controller is left undefined. Note that different ways of explaining the original target's sequences of requests (i.e., different explanatory functions) yield different induced controllers.

Finally, an *imported* controller is a maximal (i.e., non-strictly dominated) controller within the family of induced controllers—the "best" induced controllers. Technically:

Definition 3.13 (Imported controllers). The set of <u>*imported controllers*</u> from C on $\mathcal{T}$ into target $\mathcal{T}'$, denoted $\Omega^{\mathcal{T}'}_{\langle \mathsf{C},\mathcal{T}\rangle}$ is the set of all controllers $\hat{\mathsf{C}}$ for $\mathcal{T}'$ such that:

1. $\hat{\mathsf{C}}$ is an induced controller from C on target $\mathcal{T}$ for $\mathcal{T}'$; and
2. there is no other induced controller C' such that $\mathsf{C}' > \hat{\mathsf{C}}$.

$\square$

First, we show that supremal target behaviors amount to better, or more precisely "never worse," imported controllers.

Theorem 3.9. *Let $\tilde{\mathcal{T}}_1$ and $\tilde{\mathcal{T}}_2$ be two RTFs of target $\mathcal{T}$ in system $\mathcal{S}$, and let $\tilde{\mathsf{C}}_1$ and $\tilde{\mathsf{C}}_2$ be exact compositions of $\tilde{\mathcal{T}}_1$ and $\tilde{\mathcal{T}}_2$, respectively. Suppose also that $\tilde{\mathcal{T}}_2 \preceq_{\mathcal{E}} \tilde{\mathcal{T}}_1$ (i.e, $\tilde{\mathcal{T}}_1$ E-simulates $\tilde{\mathcal{T}}_2$). Then, for every controller $\mathsf{C}_1 \in \Omega^{\mathcal{T}}_{\langle \tilde{\mathsf{C}}_1,\tilde{\mathcal{T}}_1\rangle}$, there is no controller $\mathsf{C}_2 \in \Omega^{\mathcal{T}}_{\langle \tilde{\mathsf{C}}_2,\tilde{\mathcal{T}}_2\rangle}$ such that $\mathsf{C}_2 > \mathsf{C}_1$ holds.*

Proof. Assume controllers $\mathsf{C}_1 \in \Omega^{\mathcal{T}}_{\langle \tilde{\mathsf{C}}_1,\tilde{\mathcal{T}}_1\rangle}$ and $\mathsf{C}_2 \in \Omega^{\mathcal{T}}_{\langle \tilde{\mathsf{C}}_2,\tilde{\mathcal{T}}_2\rangle}$ exist such that $\mathsf{C}_2 > \mathsf{C}_1$. Let Expl_1 and Expl_2 be the target explanatory functions on which C_1 and C_2 are built upon, respectively. Now, consider a target explanatory function Expl'_1 for $\tilde{\mathcal{T}}_1$ such that $\mathsf{Expl}'_1(h)$ E-simulates $\mathsf{Expl}_2(h)$ state-wise (i.e., at each step). Formally, for all histories $h \in \mathcal{H}_{\mathcal{S}}$ if $\mathsf{Expl}_2(h) = t_2^0 \xrightarrow{g_2^1,a^1} \cdots t_2^{|h|}$ then, $\mathsf{Expl}'_1(h) = t_1^0 \xrightarrow{g_1^1 a^1} \cdots t_1^{|h|}$ such that for all $i \leq |h|$ it is the case that $t_2^i \preceq_{\mathcal{E}} t_1^i$. Note, such function Expl'_1 exists since $\tilde{\mathcal{T}}_1$ E-simulates $\tilde{\mathcal{T}}_2$. Next, consider the induced controller C'_1 built upon target explanatory function Expl'_1. Then, the set of enacted behavior ($\mathcal{E}_{\mathcal{T}}$) traces realized by C'_1 subsumes the set of traces realized by C_2; that is $\Delta^{\mathsf{C}'_1}_{\langle \mathcal{S},\mathcal{T}\rangle} \subseteq \Delta^{\mathsf{C}_2}_{\langle \mathcal{S},\mathcal{T}\rangle}$. Hence, $\mathsf{C}'_1 \geq \mathsf{C}_2$ holds (i.e., C'_1 dominates C_2). Since, by assumption, $\mathsf{C}_2 > \mathsf{C}_1$, it follows that $\mathsf{C}'_1 > \mathsf{C}_1$, an contradiction since C_1 is not strictly dominated by any induced controller from C to $\mathcal{T}$.

$\square$

In other words, if $\tilde{\mathcal{T}}_1$ is as good a realizable fragment as $\tilde{\mathcal{T}}_2$, then $\tilde{\mathcal{T}}_1$'s imported controllers will not be worse than those imported from $\tilde{\mathcal{T}}_2$. More importantly, the next result demonstrates that importing controllers from a SRTF yields maximal compositions (for the original target specification), and that, together, these maximal compositions account for every trace of the original enacted target that could ever be realized. In other words, $\Omega^{\mathcal{T}}_{\langle\hat{\mathsf{C}},\hat{\mathcal{T}}\rangle}$ is sound and "complete."

Theorem 3.10. *Let $\hat{\mathcal{T}}$ be an SRTF of target specification $\mathcal{T}$ in system $\mathcal{S}$, and let $\hat{\mathsf{C}}$ be an exact composition for $\hat{\mathcal{T}}$. Then,*

- *For all $C \in \Omega^{\mathcal{T}}_{\langle\hat{\mathsf{C}},\hat{\mathcal{T}}\rangle}$, it holds that $C \in \textsc{MaxComp}(\mathcal{S},\mathcal{T})$; that is, C is a maximal composition (Definition 3.5); and*
- *$\bigcup_{C\in\Omega^{\mathcal{T}}_{\langle\hat{\mathsf{C}},\hat{\mathcal{T}}\rangle}} \Delta^{C}_{\langle\mathcal{S},\mathcal{T}\rangle} = \bigcup_{C\in\textsc{MaxComp}(\mathcal{S},\mathcal{T})} \Delta^{C}_{\langle\mathcal{S},\mathcal{T}\rangle}$; that is, all imported controllers together account for all realizable enacted target traces.*

Proof. The proof uses an auxiliary definition to enhance a behavior to account for a set of enacted traces. If $\mathcal{T} = \langle T, G, t_0, \varrho\rangle$ is a behavior and Δ is a set of enacted target traces of some other target behavior $\mathcal{T}'$ over environment $\mathcal{E} \in \mathcal{S}$ (wlog we assume $\mathcal{T}'$ and $\mathcal{T}$ have disjoint sets of states), we define $\mathcal{T}_{+\Delta} = \langle T^+, G^+, t_0, \varrho^+\rangle$ as follows:

- $T^+ = T \cup \{\mathsf{tgt}(s) \mid s \text{ is a state in some trace in } \Delta\}$;
- $G^+ = G \cup \{g_s \mid g_s(\mathsf{env}(s)) = \mathtt{true}, \text{ where } s \text{ is a state in some trace in } \Delta\}$;
- $\varrho^+ = \varrho \cup \{\langle t_0, g_{s_0}, a_1, \mathsf{tgt}(s_1)\rangle \mid g_{s_0}(\mathsf{env}(s_0)) = \mathtt{true}, s_0 \xrightarrow{a_1} s_1 \cdots \in \Delta\} \cup$ $\{\langle \mathsf{tgt}(s_i), g_{s_i}, a_{i+1}, \mathsf{tgt}(s_{i+1})\rangle \mid g_{s_i}(\mathsf{env}(s_i)) = \mathtt{true}, s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \in \Delta, i \geq 1\}$.

Informally, we extend $\mathcal{T}$ with a disjoint sub-transition system that can produce exactly those traces in Δ. See, this is well-defined as $\mathcal{T}'$ is finite, and so will $\mathcal{T}_{+\Delta}$. For the first claim, we assume there exists $\mathsf{C} \in \Omega^{\mathcal{T}}_{\langle\hat{\mathsf{C}},\hat{\mathcal{T}}\rangle}$ such that $\mathsf{C} \notin \textsc{MaxComp}(\mathcal{S},\mathcal{T})$. Hence, there exists a controller $\mathsf{C}' \in \textsc{MaxComp}(\mathcal{S},\mathcal{T})$ such that $\Delta^{\mathsf{C}}_{\langle\mathcal{S},\mathcal{T}\rangle} \subset \Delta^{\mathsf{C}'}_{\langle\mathcal{S},\mathcal{T}\rangle}$. We next enhance $\hat{\mathcal{T}}$ with the set of traces realized by C'; that is, we build $\hat{\mathcal{T}}_{+\Delta^{\mathsf{C}'}_{\langle\mathcal{S},\mathcal{T}\rangle}}$, and extend $\hat{\mathsf{C}}$ to $\hat{\mathsf{C}}'$ such that $\hat{\mathsf{C}}'$ mimics C' for transition requests arising out from $\hat{\mathcal{T}}$'s extension (i.e., requests from traces in $\Delta^{\mathsf{C}'}_{\langle\mathcal{S},\mathcal{T}\rangle}$). Formally,

$$\hat{\mathsf{C}}'(h, t_1 \xrightarrow{g,a} t_2) = \begin{cases} \hat{\mathsf{C}}(h, t_1 \xrightarrow{g,a} t_2) & t_1 \xrightarrow{g,a} t_2 \text{ in } \mathcal{T} \\ \mathsf{C}'(h, t_1 \xrightarrow{g,a} t_2) & t_1 = t_{10} \text{ and } t_2 \notin T \\ \mathsf{C}'(h, t_1 \xrightarrow{g,a} t_2) & \text{otherwise.} \end{cases}$$

Observe, *(i)* $\hat{\mathcal{T}}_{+\Delta^{\mathsf{C}'}_{\langle\mathcal{S},\mathcal{T}\rangle}}$ is E-simulated by $\mathcal{T}$ as $\hat{\mathcal{T}} \preceq_{\mathcal{E}} \mathcal{T}$ and $\Delta^{\mathsf{C}'}_{\langle\mathcal{S},\mathcal{T}\rangle}$ are the traces of enacted target $\mathcal{E}_{\mathcal{T}}$; and *(ii)* $\hat{\mathsf{C}}'$ is an exact controller for $\hat{\mathcal{T}}_{+\Delta^{\mathsf{C}'}_{\langle\mathcal{S},\mathcal{T}\rangle}}$ in $\mathcal{S}$. Hence, the extension $\hat{\mathcal{T}}_{+\Delta^{\mathsf{C}'}_{\langle\mathcal{S},\mathcal{T}\rangle}}$ is an RTF of $\mathcal{T}$ in $\mathcal{S}$. Moreover, since $\hat{\mathcal{T}}$ is an SRTF of $\mathcal{T}$ in $\mathcal{S}$, it holds that $\hat{\mathcal{T}}_{+\Delta^{\mathsf{C}'}_{\langle\mathcal{S},\mathcal{T}\rangle}} \preceq_{\mathcal{E}} \hat{\mathcal{T}}$. Because there is a way to evolve $\hat{\mathcal{T}}$ so as to mimic all traces in $\Delta^{\mathsf{C}'}_{\langle\mathcal{S},\mathcal{T}\rangle}$, there must exist an induced controller C^* from $\hat{\mathsf{C}}$ into $\mathcal{T}$ such that $\Delta^{\mathsf{C}'}_{\langle\mathcal{S},\mathcal{T}\rangle} \subseteq \Delta^{\mathsf{C}^*}_{\langle\mathcal{S},\mathcal{T}\rangle}$. This together with the original assumption implies that $\Delta^{\mathsf{C}}_{\langle \mathcal{S},\mathcal{T}\rangle} \subset \Delta^{\mathsf{C}^*}_{\langle \mathcal{S},\mathcal{T}\rangle}$, or what is the same, $\mathsf{C}^* > \mathsf{C}$, which is a contradiction since C is an imported controller.

For the second claim, assume there exists a realizable trace τ of $\mathcal{E}_{\mathcal{T}}$ such that τ is *not* realized by any imported controller. Let C' be a controller realizing τ. We build the enhanced behavior $\hat{\mathcal{T}}_{+\{\tau\}}$ and extend $\hat{\mathsf{C}}$ to $\hat{\mathsf{C}}'$ so that $\hat{\mathsf{C}}'$ mimics C' for requests arising from $\hat{\mathcal{T}}$'s extension. Now, $\hat{\mathcal{T}}_{+\{\tau\}}$ is an RTF of $\mathcal{T}$ and $\hat{\mathcal{T}}$ does not E-simulate $\hat{\mathcal{T}}_{+\{\tau\}}$ (otherwise τ would be accounted for by some induced controller), which is a contradiction since $\hat{\mathcal{T}}$ is an SRTF of $\mathcal{T}$ in $\mathcal{S}$. □

Theorems 3.9 and 3.10 are important as they establish the relationship between optimising the target specification and optimizing its controller. *Optimizing target specifications implies maximising controllers.* Indeed, maximal compositions and SRTFs are, somehow, equivalent perspectives at behavior composition optimisation. A direct, and expected, consequence is that if an SRTF is simulation equivalent to the target specification, then there is an exact composition for the composition problem at hand.

Corollary 3.11. *Let $\hat{\mathcal{T}}$ be a supremal target behavior of target specification (SRTF) $\mathcal{T}$ on system $\mathcal{S}$ such that $\hat{\mathcal{T}} \sim_{\mathcal{E}} \mathcal{T}$, and let $\hat{\mathsf{C}}$ be an exact composition for $\hat{\mathcal{T}}$ in $\mathcal{S}$. Then, every imported controller in $\Omega^{\mathcal{T}}_{\langle \hat{\mathsf{C}},\hat{\mathcal{T}}\rangle}$ is an exact composition for $\mathcal{T}$ in $\mathcal{S}$.*

Proof. Since $\hat{\mathcal{T}}$ has an exact composition and $\hat{\mathcal{T}}$ can mimic $\mathcal{T}$ (due to $\mathcal{T} \preceq_{\mathcal{E}} \hat{\mathcal{T}}$); $\mathcal{T}$ has an exact composition: just delegate requests as done with $\hat{\mathcal{T}}$. Let C be an exact composition for $\mathcal{T}$ in $\mathcal{S}$, and assume next that there exists $\mathsf{C}' \in \Omega^{\mathcal{T}}_{\langle \hat{\mathsf{C}},\hat{\mathcal{T}}\rangle}$ such that C' is *not* an exact composition for $\mathcal{T}$ in $\mathcal{S}$. Hence, $\mathsf{C} > \mathsf{C}'$ follows, which is a contradiction since $\mathsf{C}' \in \textsc{MaxComp}(\mathcal{S},\mathcal{T})$ due to Theorem 3.10. □

To conclude, optimising the target specification results in the so-called supremal realizable target behavior (SRTF), which is the best fragment of the original target specification accommodating an exact controller. While SRTFs and maximal compositions are equivalent perspectives at behavior composition optimisation, SRTFs are more user friendly than maximal compositions. Since SRTFs are in the same language as the problem input, an specifications designer can compare the original specification with its SRTF (for example to know which parts require extra information). More importantly, since SRTFs are unique up to simulation equivalence, a system user can be given the SRTF with the guarantee that this is the best possible solution she could have.

3.6 Deterministic RTFs

The relaxation of target specification to nondeterministic, but fully controllable, behaviors lead to some elegant properties of SRTFs, such as uniqueness up to simulation equivalence

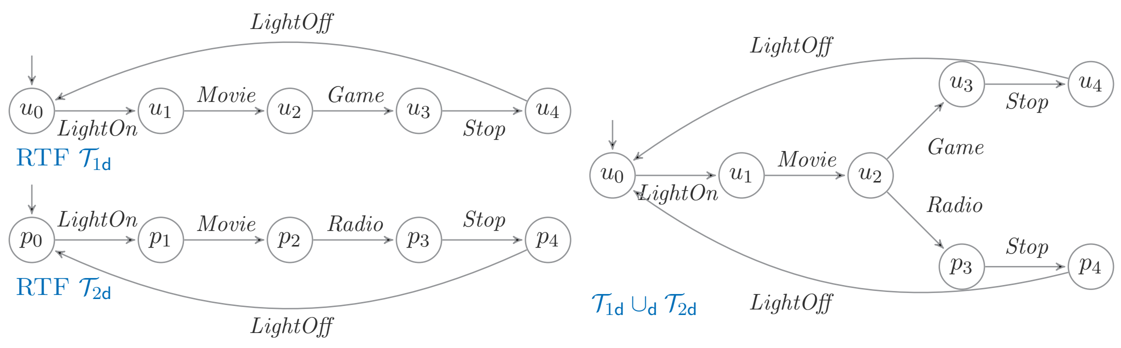


Figure 3.5: Union of two target specifications restricted to deterministic RTFs.

(Theorem 3.8). However, one may ask if this introduction of nondeterminism is at all necessary. An obvious question that arises is:

> *"What optimisation strategies exist while remaining strictly in the realm of classical behavior composition; that is, only dealing with deterministic target specifications?"*

To answer such a question, we look at what properties of nondeterministic RTFs still hold for deterministic RTFs. Note that we retain the same concept of what constitutes RTFs and SRTFs. To recap, an RTF is a target fragment which has an exact controller in the given system, and an SRTF is the largest amongst such fragments,

Definition 3.14 (DRTF and DSRTF). A DRTF (DSRTF) is a deterministic RTF (DSRTF) for a given a deterministic target specification $\mathcal{T}$ and system $\mathcal{S}$. □

Recall that for deterministic transition systems, language equivalence and simulation are equivalent notions [Girard and Pappas 2007] (also see Section 2.2). Therefore, we define the union of two DRTFs as a target specification whose language is the union of the languages of the given DRTFs. Formally, given two DRTFs $\mathcal{T}_i = \langle T_i, G_i, t_{i0}, \varrho_i \rangle$, where $i \in \{1, 2\}$, for a target specification $\mathcal{T}$ and system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$, the target specification resulting from the union of $\mathcal{T}_1$ and $\mathcal{T}_2$ is any deterministic target specification $\mathcal{T}_1 \cup_{\mathsf{d}} \mathcal{T}_2$ such that (below $\mathcal{E}_{\mathcal{T}}$ denotes enacted system of $\mathcal{T}$ in $\mathcal{E}$):

$$\mathcal{L}(\mathcal{E}_{(\mathcal{T}_1 \cup_{\mathsf{d}} \mathcal{T}_2)}) = \mathcal{L}(\mathcal{E}_{\mathcal{T}_1}) \cup \mathcal{L}(\mathcal{E}_{\mathcal{T}_2}).$$

Note that, structurally, multiple target specifications may depict such a union; however, all of them are semantically equivalent (they generate the same language). Since language equivalence and simulation are equivalent for deterministic transition systems, two deterministic language equivalent target specifications will also be simulation equivalent [Girard and Pappas 2007]. Then, as a consequence of Corollary 3.11, if one such union has an

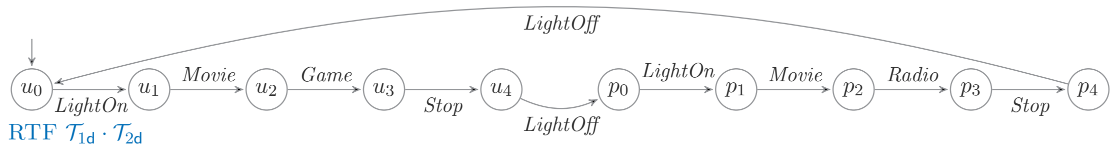


Figure 3.6: Concatenation of deterministic RTFs $\mathcal{T}_{1d}$ and $\mathcal{T}_{2d}$.

exact controller in a given system, then all possible unions will also accommodate an exact controller.

Let us focus on what it means for the end user to use deterministic specifications. Fundamentally, such a restriction on target specifications abolishes the user's freedom to *pre-select*, in advance, different target traces having a common prefix. For instance, compare the nondeterministic RTF $\hat{\mathcal{T}}_2$ shown in Figure 3.3 (see page 49) and the deterministic target specification $\mathcal{T}_{1d} \cup_d \mathcal{T}_{2d}$ depicted in Figure 3.5. In specification $\hat{\mathcal{T}}_2$, the user has a choice to make at state u_1: to select transition *Movie* resulting in u_2 or u_4 (there are two ways to request *Movie*). As a result of such a selection, the controller has more information on the possible future requests. In particular, if $u_1 \xrightarrow{Movie} u_4$ is selected, then the only next possible request is the *Radio* action, and the controller knows that. From the user's perspective, she is committing in advance to certain future requests. Observe, due to structural limitations in deterministic specifications, such an "advance" selection is not possible. As a consequence, the target specifications will no longer be able to embed such extra information related to sequencing of successor actions. Due to this, union of DRTFs may not be a DRTF. More importantly, there might be more than one deterministic DSRTF for a given composition instance.

Theorem 3.12. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two DRTFs of a target specification $\mathcal{T}$ in system $\mathcal{S}$. Then:*

- *The union of $\mathcal{T}_1$ and $\mathcal{T}_2$ may not be a DRTF; and*
- *Multiple DSRTFs for $\mathcal{T}$ in $\mathcal{S}$ may exist.*

Proof. Figure 3.5 presents such an instance for the smart home scenario with two DRTFs $\mathcal{T}_{1d}$ and $\mathcal{T}_{2d}$, and their union $\mathcal{T}_{1d} \cup_d \mathcal{T}_{2d}$. Observe that, first, the resulting union $\mathcal{T}_{1d} \cup_d \mathcal{T}_{2d}$ is no longer a DRTF as it is not realizable in the media room system. There is no controller that can realize *both* the traces $\langle e_0, e_0\rangle \xrightarrow{LightOn} \langle u_1, e_1\rangle \xrightarrow{Movie} \langle u_2, e_2\rangle \xrightarrow{Game} \langle u_3, e_3\rangle$ and $\langle e_0, e_0\rangle \xrightarrow{LightOn} \langle u_1, e_1\rangle \xrightarrow{Movie} \langle u_2, e_2\rangle \xrightarrow{Radio} \langle u_3, p_3\rangle$. And second, check that $\mathcal{T}_1$ and $\mathcal{T}_2$ are both semantically *distinct* DSRTFs. $\mathcal{L}(\mathcal{E}_{\mathcal{T}_1}) \neq \mathcal{L}(\mathcal{E}_{\mathcal{T}_2})$; that is, they are not language equivalent, and there is no other DRTF $\mathcal{T}'$ such that $\mathcal{T}_1 \prec_{\mathcal{E}} \mathcal{T}'$ and $\mathcal{T}_2 \prec_{\mathcal{E}} \mathcal{T}'$. □

Although DSRTFs are no longer unique (semantically) for deterministic target specifications, a user may still be given a set of distinct DSRTFs to choose from. Interestingly, for a given problem instance, the set of all DSRTFs may not even be finite. For example, consider DSRTFs $\mathcal{T}_{1\mathsf{d}}$ and $\mathcal{T}_{2\mathsf{d}}$ for the media room scenario (see Figure 3.5). The concatenation of $\mathcal{T}_{1\mathsf{d}}$ and $\mathcal{T}_{2\mathsf{d}}$, as shown in Figure 3.6, is indeed a DSRTF ($\mathcal{T}_{1\mathsf{d}} \not\preceq_{\mathcal{E}} \mathcal{T}_{1\mathsf{d}} \cdot \mathcal{T}_{2\mathsf{d}}$ and $\mathcal{T}_{2\mathsf{d}} \not\preceq_{\mathcal{E}} \mathcal{T}_{1\mathsf{d}} \cdot \mathcal{T}_{2\mathsf{d}}$).[3] Interestingly, multiple concatenations (with different ordering) involving $\mathcal{T}_{1\mathsf{d}}$ and $\mathcal{T}_{2\mathsf{d}}$ result in unique DSRTFs of target specification $\mathcal{T}$ in media room system $\mathcal{S}$. Since the concatenation operation can be done any number of times, the set of all DSRTFs is also infinite:

Theorem 3.13. *There exists a behavior composition problem with target specification $\mathcal{T}$ and system $\mathcal{S}$ such that the set of all DSRTFs is infinite.*

These results show that significant desirable properties are lost if we restrict RTFs to deterministic specifications.

One could consider different behavior restrictions, other than mentioned in this chapter, to categorize partial target specifications. For example, target fragments which are "cuts" of the original target specification. In such target cuts, one is only allowed to remove transitions from the original target behavior. That is, the transition relation of a target fragment is a subset of the transition relation of the original target behavior. In such a framework, one will then look for minimal subsets of the transition relation (of the original target) that should be removed in order to guarantee full realizability. Although outside the scope of this thesis, we believe that the optimisation framework presented in this chapter subsumes such syntactic restriction(s). Obviously, it remains to be seen, for the nondeterministic case, if the SRTFs can actually be computed and represented finitely.

3.7 Summary

In this chapter, we defined two equivalent behavior composition optimisation notions, namely, SRTFs and maximal compositions. To that end, we relaxed the classical composition setting by allowing for nondeterministic target specifications. SRTFs are the alternate fully realizable specifications closest to the original target and are unique up to E-simulation equivalence. On the other hand, maximal compositions are the best controllers for the original target specification. We showed the correctness of SRTFs by providing a technique to obtain maximal compositions by way of importing the exact composition from the SRTF into the original target.

Initially, the work of Girard and Pappas [2007] appeared to be extremely related to our objectives, as it proposes a notion of transition system approximation based on the

[3]Intuitively, we concatenate two DRTFs by joining the states terminating in the initial state of first DRTF to the initial state of the second DRTF. Concatenation of DRTFs is not a DRTF in general.

notion of simulation. However, their work differs in *what* is being approximated. In the most general notion of simulation, only some aspects of states are observable and two states in simulation are meant to coincide on their observable aspects. In Girard and Pappas's account, an approximate transition system is allowed to differ on such observables up to some extent: s simulates s' implies s can (always) replicate all moves of s' and s's observation is "similar" to that of s'. It follows then that the approximating transition system *must* still be able to mimic *all* actions of the approximated system. In our framework, there is no notion of state observations (every state has the same observations) and hence we only focus on the similarities of states in terms of the potential behavior they can generate.

To summarise:

- We relaxed the classical composition framework to allow nondeterministic target specifications.
- Optimising controller and target specification are two different, but equivalent, perspectives to optimise the behavior composition problem.
- Optimising controllers results in *maximal compositions* - best possible controllers that one can achieve for a given composition problem.
- E-simulation relation was used to capture target fragments (partial specifications) for a given composition problem.
- A realizable target fragment (RTF) is a target fragment which accommodates an exact controller.
- Optimising target specifications yields *supremal realizable target behaviors (SRTFs)* - the union of all RTFs.
- SRTFs are unique up to E-simulation equivalence.
- A composition problem is solvable if the given target specification is E-simulation equivalent to its SRTF.
- Important properties are lost if one restricts to deterministic target specifications (classical framework).

CHAPTER 4

Computing supremals

"The problems are solved, not by giving new information, but by arranging what we have known since long."

–Ludwig Wittgenstein

The behavior composition optimisation framework of Chapter 3 provides a constructive *algebraic* definition of supremal realizable target fragments (SRTF). Briefly, given a target specification $\mathcal{T}$ and system $\mathcal{S}$, the SRTF of $\mathcal{T}$ in $\mathcal{S}$ is the union of *all* $\mathcal{T}$'s RTFs in system $\mathcal{S}$. Interestingly, for a given composition problem, there will be an infinite number of RTFs due to structural variations.[1] Obviously, one cannot take the union of all RTFs in such a case. Hence, from a computational perspective, one is naturally interested in the computability aspect of SRTFs. A pressing question is whether we can not only represent SRTFs finitely, but also be able to compute them. We address such a question positively in this chapter and focus on finding effective techniques to compute SRTFs for unsolvable composition instances.

Various efficient techniques exist for computing exact compositions in the classical composition setting (see Section 2.5 for a survey), including synthesis via safety games [De Giacomo and Patrizi 2010, De Giacomo et al. 2013] and ATL model checking [De Giacomo and Felli 2010]. However, all those techniques aim at synthesising *exact* composition controllers that would solve a given instance completely. In the context of our work, we are instead interested in *computing SRTFs*. A surprising result we show here is that one can also use LTL synthesis and ATL model checking to synthesize an SRTF for a possibly unsolvable composition problem, involving only *deterministic* available behaviors (and to extract the corresponding composition generator). Unfortunately, when nondeterministic behaviors are involved, reduction to these techniques proves to be insufficient. In order to

[1]Two transitions systems may be structurally different but still be simulation equivalent.

compute SRTFs for target specifications meant to be realized in nondeterministic systems, we provide an alternate approach via a particular "belief-level" system.

4.1 Computing SRTFs for deterministic systems

We start by tackling the problem of computing SRTFs for composition problems involving only deterministic available behaviors. In particular, we present two techniques, namely, LTL synthesis via safety games and ATL model checking. The motivation behind this is the availability of software tools, such as TLV [Pnueli and Shahar 1996], NUGAT,[2] ANZU [Jobstmann et al. 2007], and MOCHA [Alur et al. 1998], providing effective procedures for strategy computation and convenient languages for representing problem instances in a modular and high-level manner.

4.1.1 Safety games

The classical behavior composition problem can be seen as an adversarial game played between the controller on one side and the available system together with the target on the other. In this game, the target requests actions in such a manner so as to make it difficult for the controller to delegate these requests, and the available system (the target's partner) tries to evolve, after the controller's delegation, such that the controller gets stuck (cannot honor subsequent target requests). On the other side, the controller tries to delegate the current target request in such a way that irrespective of how the system evolves after the delegation and what subsequent target request arise, it can continue to successfully delegate target requests.

To simplify, the controller tries to ensure it can *always* honor target requests, a safety condition in the context of linear temporal logic. For classical composition, De Giacomo and Patrizi [2010] showed how to compute the composition generator (i.e., a structure representing all exact compositions) by building a winning strategy for a particular safety-game (see Section 2.5.4). Inspired by such a reduction, we show how to synthesize the SRTF, along with its composition generator, for a given composition problem containing only deterministic available behaviors.

Preliminaries

Automatic synthesis of programs from a given specification has been extensively studied in the literature [Bloem et al. 2011, Pnueli and Rosner 1989a;b, Manna and Wolper 1984, Asarin et al. 1995]. Conceptually, the problem is to automatically synthesize a program such that it satisfies a given temporal specification. One of the approaches to solve this synthesis problem is via a two-player game. The game consists of two players, namely,

[2] https://es.fbk.eu/index.php?n=Tools.NuGaT

an *environment-player* and a *system-player*,[3] where each player controls their respective set of variables. To differentiate the game terminology from the one of behavior composition, we suffix the player names in the context of games with the word "player", that is, environment-player and system-player. Let us denote the set of variables controlled by environment-player and system-player as $\mathcal{X}$ and $\mathcal{Y}$, respectively. Moves of the players consist of updating their respective variables according to certain pre-defined rules that are specific to each game. At each step of the game, the environment-player moves by updating its variables in $\mathcal{X}$ and the system-player replies by updating its own set of variables in $\mathcal{Y}$. It is assumed that the system-player can see the environment-player's previous move. The goal of the system-player is for the game to satisfy a given temporal property, an LTL formula, regardless of the environment-player's moves. Similar to adversarial games, the players respond to the opponent's moves, each trying to constrain their opponent in order to win the game.

The game itself is played over a *game structure* that defines *(i)* the set of variables over which the game is played, with the rules governing how these variables are updated, *(ii)* the start states of the game, and *(ii)* the temporal property capturing the winning condition for the system-player. Formally, a *game structure* [Bloem et al. 2011] is a tuple $\mathcal{G} = \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \Theta, \varrho_e, \varrho_s, \varphi \rangle$, where:

- $\mathcal{V} = \{v_1, \ldots v_n\}$ is the finite set of state variables that range over finite domains $V_1 \ldots V_n$, respectively. A *state* $\vec{s} = \langle s_1, \ldots, s_n \rangle \in V$ is a possible assignment to the variables, where $V = V_1 \times \cdots \times V_n$ is the set of all possible valuations over $\mathcal{V}$. $\mathcal{X} \subseteq \mathcal{V}$ is the set of input variables controlled by the environment-player, and $\mathcal{Y} = \mathcal{V} \setminus \mathcal{X}$ is the set of output variables controlled by the system-player. Let X and Y denote the set of all possible valuations over $\mathcal{X}$ and $\mathcal{Y}$, respectively. By a slight abuse of notation, we denote a state $\vec{s} = \langle \vec{x}, \vec{y} \rangle$, where $\vec{x} \in X$ is a valuation of variables in $\mathcal{X}$ and $\vec{y} \in Y$ is a valuation of variables in $\mathcal{Y}$.

- Θ is a formula representing the initial states of the game. It is a boolean combination of expressions of the form $(v_i = s_i)$, such that $v_i \in \mathcal{V}$ is a variable and $s_i \in V_i$ is its assignment, where $i \leq n$. Given a game state $\vec{s}$, we write $\vec{s} \models \Theta$ if $\vec{s}$ satisfies Θ in the standard way. In general, Θ may not contain all the variables from $\mathcal{V}$, hence there might be multiple initial states for a game.

- $\varrho_e \subseteq V \times X$ is the environment-player's transition relation. A tuple $\langle \vec{s}, \vec{x} \rangle \in \varrho_e$ means that when the game is in a state $\vec{s}$, then $\vec{x}$ is a possible legal move for the environment-player.

- $\varrho_s \subseteq V \times X \times Y$ is the system-player's transition relation. A tuple $\langle \vec{s}, \vec{x}, \vec{y} \rangle \in \varrho_s$ means that when the game is in a state $\vec{s}$ and the environment-player updated its

[3]The players are different from the environment and system components of behavior composition.

variables to $\vec{x}$, then $\vec{y}$ is a possible legal move for the system-player.

- φ is an LTL formula, denoting the winning condition for the system-player.

When the goal is a safety one; that is, one of the form "always ϕ," the game is said to be a *safety game*. Intuitively, the system-player's objective is to always be able to reply to the environment-player's moves so as to satisfy a given (goal) temporal property, while the environment-player tries to avoid this. Technically, the task is to synthesize a winning reply-strategy for the system-player such that either the system-player enforces a game state from which the environment-player has no more moves left, or the goal holds in all possible infinite "plays" that may ensue when the system follows such a strategy. A state is deemed "winning" if there is a winning strategy from it. We borrow the technical notions from [Bloem et al. 2011] to formalize the notion of winning in such a game.

A game state $\langle\vec{x},\vec{y}\rangle$ is a *successor* of a state $\vec{s}$ iff $\varrho_e(\vec{s},\vec{x})$ and $\varrho_s(\vec{s},\vec{x},\vec{y})$ hold. A *play* is a, possibly infinite, sequence of the form $\vec{s_0}\vec{s_1}\ldots$ such that *(i)* $\vec{s_0} \models \Theta$; that is, $\vec{s_0}$ is an initial state; and *(ii)* for each $i \geq 0$, $\vec{s}_{i+1}$ is a successor of $\vec{s_i}$. When a play $\sigma = \vec{s_0}\ldots\vec{s_n}$ is finite, we denote the last state of σ by $\mathsf{last}(\sigma)$; that is, $\mathsf{last}(\sigma) = \vec{s_n}$. A play $\sigma = \vec{s_0}\vec{s_1}\ldots$ is *winning for the system-player* [Bloem et al. 2011] if either: *(i)* σ is finite and there is no assignment $\vec{y}$ over variables in $\mathcal{Y}$ such that $\varrho_e(\mathsf{last}(\sigma),\vec{y})$ holds; or *(ii)* σ is infinite and it models φ. Otherwise, the play σ is winning for the environment-player. A strategy for the system-player is a partial function $f : V^+ \times X \rightarrow Y$ such that for every finite play $\sigma = \vec{s_0}\ldots\vec{s_n}$ if there exists $\vec{x} \in X$ where $\varrho_e(\vec{s_n},\vec{x})$ holds, then $\varrho_s(\vec{s_n},\vec{x},f(\sigma,\vec{x}))$ holds. A play $\sigma = \langle\vec{x_0},\vec{y_0}\rangle\langle\vec{x_1},\vec{y_1}\rangle\ldots$ is *compliant* with a given system-player strategy f if for all $i \geq 0$ we have $f(\langle\vec{x_0},\vec{y_0}\rangle\ldots\langle\vec{x_i},\vec{y_i}\rangle,\vec{x}_{i+1}) = \vec{y}_{i+1}$. A strategy f is *winning* for the system-player from a state $\vec{s} \in V$ (winning state) if all plays starting from $\vec{s}$ that are compliant with f are winning for the system-player. A strategy f is *winning* for a game if f is a winning strategy from all the initial states of the game; and a game is winning for the system-player if there exists a winning strategy for that game. The *winning set* of a game $\mathcal{G}$, denoted by $\mathcal{G}^W$, is the set of all winning states for the system-player in that game.

Computationally, the winning set for a game structure can be computed using a fixpoint algorithm [De Giacomo et al. 2013, Bloem et al. 2011]. Intuitively, for a safety game formula $\Box\phi$ (where $\Box$ is the temporal operator denoting "always"), the fixpoint algorithm first computes the set of game states (W) that satisfy the formula ϕ. Then from this set W, the algorithm iteratively removes game states from where the system-player cannot ensure the successor states to be in W. The algorithm terminates when no more states can be removed from W; that is, when a fixpoint is reached. In terms of computational complexity, the winning set for a safety game can be computed in time polynomial to the state space ($|V|$) [Piterman et al. 2006, De Giacomo et al. 2013], which is exponential in the number of variables in $\mathcal{V}$.

Synthesising SRTFs via safety games

We now encode a given (unsolvable) behavior composition problem containing only deterministic available behaviors to a safety game structure, so that the problem's SRTF and its composition generator can be synthesized.

The key to achieving this resides in carefully selecting the variables under the system-player's control. The overriding idea in our encoding is to include *both* target requests and controller delegations under the system-player's control, whereas the environment-player is only in charge of the evolution of the available behaviors and the environment. Though having the system-player, which stood for the composition generator in the classical setting, select the request at each step may appear counter-intuitive, it yields the right semantics for our objective. The intuition behind this is to allow the system-player to control the game evolution in such a way that it can (always) choose only those target requests that can be successfully delegated to an available behavior. In contrast, in De Giacomo and Patrizi [2010]'s encoding (c.f. Section 2.5.4), the system-player was in charge only of the controller delegations; the target requests were under the environment-player's control. Indeed, in the classical setting one has to guarantee the realizability of the complete target specification, and hence all legal target requests ought to be catered for.

Let $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E}\rangle$ be a system, with deterministic available behaviors $\mathcal{B}_i = \langle B_i, G_i, b_{i0}, \varrho_i\rangle$ for $1 \leq i \leq n$, environment $\mathcal{E} = \langle E, A, e_0, \rho\rangle$, and let $\mathcal{T} = \langle T, G, t_0, \varrho\rangle$ be a target specification behavior. For technical convenience, we use a special "dummy" value ($\sharp$) to denote "no action" and to define the initial state of the game. The safety game structure $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle} = \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \Theta, \rho_s, \rho_c, \Box\varphi\rangle$ is defined as follows:

1. $\mathcal{V} = \{\mathbf{b_1}, \ldots, \mathbf{b_n}, \mathbf{e}, \mathbf{t_s}, \mathbf{g}, \mathbf{a}, \mathbf{t_d}, \textsf{\textbf{ind}}\}$ is the set of game variables where variables $\mathbf{b_i}$ can take values from $B_i \cup \{\sharp\}$, $\mathbf{e} \in E \cup \{\sharp\}$, $\mathbf{t_s}, \mathbf{t_d} \in T \cup \{\sharp\}$, $\mathbf{g} \in G \cup \{\sharp\}$, $\mathbf{a} \in A \cup \{\sharp\}$, and $\textsf{\textbf{ind}} \in \{1, \ldots, n, \sharp\}$. A *game state* is of the form $w = \langle b_1, \ldots, b_n, e, t_s, g, a, t_d, \textsf{ind}\rangle$ denoting that behavior $\mathcal{B}_i$ is in state b_i where $i \leq n$, environment is in the state e, and that the current request $t_s \xrightarrow{g,a} t_d$ is to be delegated to available behavior $\mathcal{B}_{\textsf{ind}}$.

2. $\mathcal{X} = \{\mathbf{b_1}, \ldots, \mathbf{b_n}, \mathbf{e}\}$ is the set of environment-player controlled variables and $\mathcal{Y} = \{\mathbf{t_s}, \mathbf{g}, \mathbf{a}, \mathbf{t_d}, \textsf{\textbf{ind}}\}$ is the set of system-player controlled variables.

3. $\Theta = \bigwedge_{i=1}^{n}(\mathbf{b_i} = \sharp) \wedge (\mathbf{e} = \sharp) \wedge (\mathbf{t_s} = \sharp) \wedge (\mathbf{g} = \sharp) \wedge (\mathbf{a} = \sharp) \wedge (\mathbf{t_d} = t_0) \wedge (\textsf{\textbf{ind}} = \sharp)$ defines the initial distinguished state $\langle \sharp, \ldots, \sharp, \sharp, \sharp, \sharp, \sharp, t_0, \sharp\rangle$ of the game.

4. ρ_e is the environment-player's transition relation, such that $(\vec{s}, \langle b'_1, \ldots, b'_n, e'\rangle) \in \rho_e$, where $\vec{s} = \langle b_1, \ldots, b_n, e, t_s, g, a, t_d, \textsf{ind}\rangle$, *iff* one of the following holds:

   a) $\vec{s} \models \Theta$; that is, $\vec{s}$ is initial, $b'_i = b_{i0}$ for $i \in \{i, \ldots, n\}$, and $e' = e_0$;

   b) $\textsf{ind} \neq \sharp$ and

i. there exists guard $g_{\mathsf{ind}} \in G_{\mathsf{ind}}$ such that $g_{\mathsf{ind}}(e) = \mathtt{true}$ and $b_{\mathsf{ind}} \xrightarrow{g_{\mathsf{ind}};a} b'_{\mathsf{ind}}$ in $\mathcal{B}_{\mathsf{ind}}$; that is, behavior $\mathcal{B}_{\mathsf{ind}}$ can execute action a;

ii. there exists transition $e \xrightarrow{a} e'$ in $\mathcal{E}$; that is, the environment allows execution of action a;

iii. $b_i = b'_i$ for all $i \in \{i, \ldots, n\} \setminus \{\mathsf{ind}\}$; that is, all other behaviors remain still.

5. ρ_s is the system-player's transition relation, such that $(\vec{s}, \vec{x}', \vec{y}') \in \rho_s$, where $\vec{s} = \langle b_1, \ldots, b_n, e, t_s, g, a, t_d, \mathsf{ind} \rangle$, $\vec{x}' = \langle b'_1, \ldots, b'_n, e' \rangle$, and $\vec{y}' = \langle t'_s, g', a', t'_d, \mathsf{ind}' \rangle$, *iff*

   a) there exists guard $g' \in G$ where $g'(e') = \mathtt{true}$, $t'_s \xrightarrow{g',a'} t'_d$ in $\mathcal{T}$, and $t'_s = t_d$; that is, the target requests a next legal transition; or

   b) $a' = g' = \sharp$ and $t'_s = t_d = t'_d$; that is, the target requests a (self loop) transition containing dummy action.

6. Formula $\varphi = \neg\Theta \wedge \mathbf{a} \neq \sharp \wedge \mathbf{ind} \neq \sharp \supset (\exists g, b').\varrho_{\mathsf{ind}}(b_{\mathsf{ind}}, g, a, b')$ states that if a game state is not initial, and a domain action is being requested and delegated to an available behavior, then that behavior is capable of a transition on that action from its current state.

Conceptually, a game state in our encoding captures a snapshot of a problem instance. Structurally, a game state consists of the current states of the available behaviors and the environment, a target request, and the behavior index which will execute the requested action. The game evolves as follows: from the initial state the only move available for the environment-player (4a) is to update the behaviors and environment to their respective initial states and the system-player responds by updating the transition request (5a) to any legal transition arising from the initial state of the target behavior. Alternatively, the system-player can also update the transition request to contain the special no action (5b). Note that there are no restrictions placed on the ind variable (controller delegations): the controller is free to delegate the target request to any behavior (including behaviors which cannot execute the action in the transition). If the system-player updates its variables to denote a legal transition request and a behavior index such that the behavior with that index can execute the requested action, then the environment-player updates its variables (4b) as per the respective behavior and environment evolutions resulting from executing the requested action. Otherwise, if the system-player updates the ind variable to $\sharp$ or the target requests a transition with $\sharp$, then the environment-player has no legal moves left. Intuitively, in this game, a winning state is either one in which the current request is legally honored by some available behavior and it has a successor winning state, or a state in which the target action or the controller delegation is $\sharp$ (after playing $\sharp$, the environment-player has no moves defined).

Note the implicit existential quantification on the subsequent request, as compared to the universal quantification implied in the case of classical composition [De Giacomo and

Patrizi 2010, De Giacomo et al. 2013]. This is because, to compute an exact composition in the classical setting, where from a target state *all* subsequent requests must be satisfiable.

Our encoding is, arguably, simpler than the one used in the classical case [De Giacomo and Patrizi 2010, De Giacomo et al. 2013]. In particular, we drop the infinite-play assumption that requires the game structure to be such that *all* the resulting plays are infinite. This implies that at every step of the game, both the environment-player and system-player should always be able to move legally.[4] Dropping this infinite-play requirement results in an encoding intuitively closer to the actual composition problem in which target specifications could be finite, eliminating the need for fake loops.

It is not a surprise that our game encoding is such that the system-player can *always* win the game. Observe, if the system-player updates the target action to $\sharp$, then the environment-player has no legal moves left. Once this happens the resulting play will be finite and the system-player will win such a game. Note that we allow the system-player to play the $\sharp$ target action at any step of the game (see condition 5b of the encoding).

Theorem 4.1. *Let $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$ be a safety game, as defined above, where $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E}\rangle$ is a system and $\mathcal{T} = \langle T, t_0, G, \varrho\rangle$ is a target specification. Then, $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$ is always winning for the system-player.*

Proof. The proof amounts to showing the existence of a winning strategy from the initial state of all game instances. We prove a stronger claim here: we show the existence of a *universal* winning strategy for the system-player. Let $\vec{s_0}$ be the distinguished initial state of the game and $\vec{x_1}$ the first move of the environment-player. Then, the strategy $f_u(\vec{s_0}, \vec{x_1}) = \vec{y_1} = \langle t_0, \sharp, \sharp, t_0, \sharp\rangle$ is a winning strategy for the system-player. First, note that the environment player has only one possible move from the initial state $\vec{s_0}$ (4a). Second, $\vec{y_1}$ is a legal move for the system-player (5b) because $\varrho_e(\vec{s_0}, \vec{x_1})$ and $\varrho_s(\vec{s_0}, \vec{x_1}, \vec{y_1})$ hold. Finally, note that once the system-player moves as per the strategy f_u, the environment-player has no legal moves left. Hence, the play $\vec{s_0}\langle \vec{x_1}, \vec{y_1}\rangle$ is finite and winning for the system-player. Thus, the strategy f_u is a winning strategy for the game. □

The above result may seem trivial or contrived, but it is strongly tied to the properties of the optimisation framework presented in the previous chapter. To elaborate, first note that *all* behavior composition problem instances have an RTF, namely, the *universal* RTF (see page 54) consisting of just one state and an empty transition relation. This is precisely what Theorem 4.1 depicts. In contrast with the classical setting, a problem instance may have an exact composition or not; that is, a system-player may win or lose the game. However, in the optimisation setting, since there always exists an RTF (and an SRTF), a system-player ought to win the game. Secondly, at the conceptual

[4]In [De Giacomo and Patrizi 2010, De Giacomo et al. 2013] infinite plays are required for the winning condition of the system-player.

level, a *universal* winning strategy exists for the system-player because there exists a *universal* RTF. Observe that this universal winning strategy is equivalent to the target not requesting any action.

Of course, since the system-player can always win the game, it can do so at the initial state or after successfully delegating a sequence of target transitions. Conceptually, each of such winning plays corresponds to an RTF. As one would expect, the union of all these winning plays should encode the SRTF. Again, this is strongly tied to the property of the optimisation framework that states that the union of all RTFs results in the SRTF (see Theorem 3.7). However, by encoding to a safety game, we now provide *finiteness* and *computability* to the algebraic definition of SRTF; that is, SRTFs are indeed finite and can actually be computed. To that end, let us define the transition system represented by the winning set of an encoded safety game.

Concretely, given the maximal winning states W of the game $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$ for a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E}\rangle$ with deterministic available behaviors $\mathcal{B}_i = \langle B_i, G_i, b_{i0}, \varrho_i\rangle$ for $1 \leq i \leq n$, environment $\mathcal{E} = \langle E, A, e_0, \rho\rangle$, and target behavior $\mathcal{T} = \langle T, t_0, G, \varrho\rangle$, we define behavior $\underline{\mathit{XtractBeh(W)}} = \langle \hat{T}, \hat{t_0}, G, \hat{\varrho}\rangle$, where:

- $\hat{T} = \{\langle b_1, \ldots, b_n, e, t_s\rangle \mid \langle b_1, \ldots, b_n, e, t_s, g, a, t_d, \mathsf{ind}\rangle \in W\}$. Given a game state $w = \langle b_1, \ldots, b_n, e, t_s, g, a, t_d, \mathsf{ind}\rangle$ let $\mathsf{st}(w) = \langle b_1, \ldots, b_n, e, t_s\rangle$.
- $\hat{t_0} = \langle b_{10}, \ldots, b_{n0}, e_0, t_0\rangle$ is the initial state of $\mathsf{XtractBeh}(W)$; and
- $\hat{\varrho}(\mathsf{st}(\langle b_1, \ldots, b_n, e, t_s, g, a, t_d, \mathsf{ind}\rangle), g, a, \mathsf{st}(w'))$ holds *iff*:
  - $\langle b_1, \ldots, b_n, e, t_s, g, a, t_d, \mathsf{ind}\rangle, w' \in W$;
  - $a \neq \sharp$, $\mathsf{ind} \neq \sharp$; that is, an action is delegated to an available behavior; and
  - $\langle w, \vec{x}, \vec{y}\rangle \in \rho_s$, where $\langle \vec{x}, \vec{y}\rangle = w'$.

Each state $\hat{t} = \langle b_1, \ldots, b_n, e, t\rangle$ of the behavior $\mathsf{XtractBeh}(W)$ is extracted from a winning state $w = \langle b_1, \ldots, b_n, e, t, g, a, t', \mathsf{ind}\rangle$, where $b_1, \ldots, b_n$ are states of behaviors $\mathcal{B}_1, \ldots, \mathcal{B}_n$, e is the environment state, $t \xrightarrow{g,a} t'$ is a transition of behavior $\mathcal{T}$, and ind is the index of behavior that will execute the action a. Intuitively, the transition relation of $\mathsf{XtractBeh}(W)$ encodes all the transition requests that can be honored *while remaining in the winning states*. Note that there will be no outgoing transitions from a state $\mathsf{st}(w)$ in $\mathsf{XtractBeh}(W)$ if w has $\sharp$ as the action request or an incorrect behavior delegation.

Given a game state $w = \langle b_1, \ldots, b_n, e, t, g, a, t', \mathsf{ind}\rangle$, let $\mathsf{comp}_{\mathcal{S}}(w)$ denote the enacted system state $\langle b_1, \ldots, b_n, e\rangle$ (i.e., $\mathsf{comp}_{\mathcal{S}}(w) = \langle b_1, \ldots, b_n, e\rangle$), $\mathsf{comp}_{\mathsf{req}}(w)$ denote the target transition $t \xrightarrow{g,a} t'$ (i.e., $\mathsf{comp}_{\mathsf{req}}(w) = t \xrightarrow{g,a} t'$), $\mathsf{comp}_{\mathcal{T}}(w)$ denote the target state t (i.e., $\mathsf{comp}_{\mathcal{T}}(w) = t$), $\mathsf{comp}_{\mathcal{E}}(w)$ denote the environment state e (i.e., $\mathsf{comp}_{\mathcal{E}}(w)$=e), and $\mathsf{comp}_{\mathcal{I}}(w)$ denote the behavior index ind (i.e., $\mathsf{comp}_{\mathcal{I}}(w) = \mathsf{ind}$). By a slight abuse of notation we extend the comp family of functions to the relevant components of $\mathsf{XtractBeh}(W)$'s states.

We are now able to present our main claim formally, namely, that the set of all the winning states encode the SRTF of the given target specification in the system.

Theorem 4.2. *Let W be the maximal set of winning states for safety game $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$, where $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E}\rangle$ is a given available system with deterministic available behaviors $\mathcal{B}_i = \langle B_i, G_i, b_{i0}, \varrho_i\rangle$ for $1 \leq i \leq n$, environment $\mathcal{E} = \langle E, A, e_0, \rho\rangle$, and $\mathcal{T} = \langle T, t_0, G, \varrho\rangle$ is a target behavior. Then, $\mathsf{XtractBeh}(W)$ is the SRTF for target $\mathcal{T}$ in system $\mathcal{S}$.*

Proof. For eligibility we represent the behavior $\mathsf{XtractBeh}(W)$ by $\hat{\mathcal{T}} = \langle \hat{T}, \hat{t_0}, G, \hat{\varrho}\rangle$. The proof amounts to showing that *(i)* $\hat{\mathcal{T}}$ is an RTF of $\mathcal{T}$ in $\mathcal{S}$, and *(ii)* $\hat{\mathcal{T}}$ is the SRTF of $\mathcal{T}$ in $\mathcal{S}$.

1. $\hat{\mathcal{T}}$ is an RTF: Due to the definition of $\hat{\varrho}$ in $\hat{\mathcal{T}}$ and ρ_s in the safety game $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$, it holds that $\hat{t} \xrightarrow{g,a} \hat{t}' \in \hat{\varrho}$ if $\mathsf{comp}_{\mathcal{T}}(\hat{t}) \xrightarrow{g,a} \mathsf{comp}_{\mathcal{T}}(\hat{t}') \in \varrho$. Now, consider the relation $\mathcal{R} \subseteq \hat{T} \times T$ such that $(\hat{t}, t) \in \mathcal{R}$ if $\mathsf{comp}_{\mathcal{T}}(\hat{t}) = t$. Then, for a tuple $(\hat{t}, t) \in \mathcal{R}$, for all transitions $\hat{t} \xrightarrow{g,a} \hat{t}'$ in $\hat{\mathcal{T}}$ there exists a transition $t \xrightarrow{g,a} t'$ in $\mathcal{T}$ such that $(\hat{t}', t') \in \mathcal{R}$. Trivially, $(\hat{t_0}, t_0) \in \mathcal{R}$, thus, $\mathcal{R}$ is an E-simulation relation of $\hat{\mathcal{T}}$ by $\mathcal{T}$, i.e., $\hat{\mathcal{T}} \preceq_{\mathcal{E}} \mathcal{T}$.

   Now, we show that $\hat{\mathcal{T}}$ is an *effective* target fragment (Definition 3.8). Let $\mathcal{E}_{\hat{\mathcal{T}}} = \langle S, A, s_0, \varrho_S\rangle$ be the enacted target behavior of $\hat{\mathcal{T}}$ in environment $\mathcal{E}$ and $\hat{\mathcal{T}}^{\uparrow G}$ be the transition system obtained by projecting $\hat{\mathcal{T}}$'s guards. Consider the relation $\mathcal{P} \subseteq \hat{T} \times S$ such that $(\hat{t}, s) \in \mathcal{P}$ if $\hat{t} = \mathsf{tgt}(s)$ and $\mathsf{comp}_{\mathcal{E}}(\hat{t}) = \mathsf{env}(s)$.[5] From condition 5a of the safety game encoding, we know that for a transition $\hat{t} \xrightarrow{a} \hat{t}'$ there exists guard $g \in G$ such that $g(\mathsf{comp}_{\mathcal{E}}(\hat{t})) = \mathtt{true}$ and $\mathsf{comp}_{\mathcal{E}}(\hat{t}) \xrightarrow{a} \mathsf{comp}_{\mathcal{E}}(\hat{t}')$ in $\mathcal{E}$. Therefore, for a tuple $(\hat{t}, s) \in \mathcal{P}$, $\hat{t} \xrightarrow{a} \hat{t}'$ in $\hat{\mathcal{T}}^{\uparrow G}$ implies that there exists a transition $s \xrightarrow{a} s'$ in $\mathcal{E}_{\hat{\mathcal{T}}}$ such that $\mathsf{tgt}(s') = \hat{t}'$ and $\mathsf{env}(s') = \mathsf{comp}_{\mathcal{E}}(\hat{t}')$. Clearly, $\langle \hat{t_0}, s_0\rangle \in \mathcal{P}$. Thus, $\hat{\mathcal{T}}^{\uparrow G}$ is simulated by $\mathcal{E}_{\hat{\mathcal{T}}}$.

   Next, we show that $\hat{\mathcal{T}}$ has an exact composition in $\mathcal{S}$; that is, $\hat{\mathcal{T}}$ is simulated by the enacted system $\mathcal{E}_{\mathcal{S}} = \langle S, A, s_0, \varrho_S\rangle$.[6] Let $\mathcal{Z} \subseteq \hat{T} \times \mathcal{S}$ be a relation such that $(\hat{t}, s) \in \mathcal{Z}$ iff $\mathsf{comp}_{\mathcal{S}}(\hat{t}) = s$. Consider a tuple $(\hat{t}, s) \in \mathcal{Z}$, a transition $\hat{t} \xrightarrow{g,a} \hat{t}'$ from $\hat{t}$, and game states w, w' such that $\mathsf{st}(w) = \hat{t}$ and $\mathsf{st}(w') = \hat{t}'$. From (winning) game states w and w' we can conclude that $s \xrightarrow{a,\mathsf{comp}_{\mathcal{I}}(w)} s'$ in $\mathcal{E}_{\mathcal{S}}$ where a is the action in transition $\mathsf{comp}_{\mathsf{req}}(w)$, and $s' = \mathsf{comp}_{\mathcal{S}}(w') = \mathsf{comp}_{\mathcal{S}}(\hat{t}')$. Thus, $(\hat{t}', s') \in \mathcal{Z}$, and so $\mathcal{Z}$ is a simulation relation of $\hat{\mathcal{T}}$ by $\mathcal{S}$. Clearly, $(\hat{t_0}, s_0) \in \mathcal{Z}$ as $\mathsf{comp}_{\mathcal{S}}(\hat{t_0}) = s_0$, therefore $\hat{\mathcal{T}} \preceq \mathcal{E}_{\mathcal{S}}$.

2. $\hat{\mathcal{T}}$ is the SRTF: Let $\tilde{\mathcal{T}} = \langle \tilde{T}, \tilde{G}, \tilde{t_0}, \tilde{\varrho}\rangle$ be the SRTF of $\mathcal{T}$ in $\mathcal{S}$. Therefore, by definition of SRTFs (Definition 3.10, item (1)), we have that $\hat{\mathcal{T}} \preceq_{\mathcal{E}} \tilde{\mathcal{T}} \preceq_{\mathcal{E}} \mathcal{T}$. We use proof by contradiction to show that $\hat{\mathcal{T}}$ and $\tilde{\mathcal{T}}$ are E-simulation equivalent. Assume that $\hat{\mathcal{T}}$ does

[5]Recall that $\mathsf{tgt}(s)$ and $\mathsf{env}(s)$ denote the target state and environment state components of the enacted state s, respectively.

[6]Note that for deterministic behaviors ND-simulation and simulation are equivalent.

not E-simulate $\tilde{\mathcal{T}}$, i.e., $\tilde{t_0} \not\preceq_{\mathcal{E}} \hat{t_0}$. Therefore, there exists a trace $\tilde{\tau} = \tilde{t^0} \xrightarrow{\tilde{g}^1,a^1} \cdots \xrightarrow{\tilde{g}^n,a^n} \tilde{t^n}$ of $\tilde{\mathcal{T}}$ such that for all traces $\hat{\tau} = \hat{t^0} \xrightarrow{g^1,a^1} \cdots \xrightarrow{g^n,a^n} \hat{t^n}$ of $\hat{\mathcal{T}}$, there exists a transition $\tilde{t^n} \xrightarrow{\tilde{g}^{n+1},a^{n+1}} \tilde{t^{n+1}}$ in $\tilde{\mathcal{T}}$ for which there is no transition $\hat{t^n} \xrightarrow{g^{n+1},a^{n+1}} \hat{t^{n+1}}$ in $\hat{\mathcal{T}}$. That is, $\tilde{\tau}$ cannot be E-simulated by any trace of $\hat{\mathcal{T}}$.

Let us consider the safety game $\mathcal{G}_{\langle \mathcal{S}, \tilde{\mathcal{T}} \rangle}$ between $\tilde{\mathcal{T}}$ and $\mathcal{S}$. Since $\tilde{\mathcal{T}}$ has an exact composition in $\mathcal{S}$, the maximal set of winning states $\tilde{W}$ will accommodate all $\tilde{\mathcal{T}}$'s transition requests. Let $\tilde{W}_{\tilde{\tau}} \subseteq \tilde{W}$ be the set of winning states catering for $\tilde{\tau}$'s transitions; that is, $\langle b_1, \ldots, b_n, \tilde{t_s}, \tilde{g}, a, \tilde{t_d}, \mathit{ind} \rangle \in \tilde{W}_{\tilde{\tau}}$ if $\tilde{t_s} \xrightarrow{\tilde{g},a} \tilde{t_d} = \tilde{t^i} \xrightarrow{\tilde{g}^{i+1},a^{i+1}} \tilde{t^{i+1}}$ for some $i \leq n$. Note that the transition $\tilde{t^n} \xrightarrow{\tilde{g}^{n+1},a^{n+1}} \tilde{t^{n+1}}$ which breaks the simulation of $\tilde{\mathcal{T}}$ by $\hat{\mathcal{T}}$ is also included. Now consider the set of states in the safety game between $\mathcal{S}$ and $\mathcal{T}$ defined by: $U = \{\langle b_1, \ldots, b_n, e, t_s, g, a, t_d, \mathit{ind} \rangle \mid \langle b_1, \ldots, b_n, e, \tilde{t_s}, \tilde{g}, a, \tilde{t_d}, \mathit{ind} \rangle \in \tilde{W}_{\tilde{\tau}}, \tilde{t_s} \preceq_{\mathcal{E}} t_s, \tilde{t_d} \preceq_{\mathcal{E}} t_d, g(e) = \tilde{g}(e) = \mathtt{true}\}$. That is, the states are identical to winning states in $\tilde{W}_{\tilde{\tau}}$ except for the transition requests. The transition requests of $\tilde{\mathcal{T}}$ are replaced by the transition requests from $\mathcal{T}$ such that the corresponding states are in simulation. Note that these states are not only legal game states but also winning for the safety game $\mathcal{G}_{\langle \mathcal{S}, \mathcal{T} \rangle}$, i.e., $U \subseteq W$: allocation of simulating transitions to the same indexes as in winning states of $\mathcal{G}_{\langle \mathcal{S}, \tilde{\mathcal{T}} \rangle}$ will still be winning in $\mathcal{G}_{\langle \mathcal{S}, \mathcal{T} \rangle}$. Therefore, U contains winning states having transition requests $t \xrightarrow{g,a} t'$ of $\mathcal{T}$, corresponding to $\tilde{\tau}$'s transition $\tilde{t^n} \xrightarrow{\tilde{g}^{n+1}a^{n+1}} \tilde{t^{n+1}}$ such that $\tilde{t^n} \preceq_{\mathcal{E}} t$ and $\tilde{t^{n+1}} \prec_{\mathcal{E}} t'$. Consequently, there will be a transition $\hat{t^n} \xrightarrow{g^{n+1},a^{n+1}} \hat{t^{n+1}}$ in trace $\hat{\tau}$ of $\hat{\mathcal{T}}$ where $\tilde{t^n} \preceq_{\mathcal{E}} \mathsf{comp}_{\mathcal{T}}(\hat{t^n})$ and $\tilde{t^{n+1}} \preceq_{\mathcal{E}} \mathsf{comp}_{\mathcal{T}}(\hat{t^{n+1}})$, which contradicts the assumption. Therefore, $\hat{\mathcal{T}}$ and $\tilde{\mathcal{T}}$ are E-simulation equivalent and hence $\hat{\mathcal{T}}$ is the SRTF of $\mathcal{T}$ in $\mathcal{S}$.

□

Observe that the winning states not only include the feasible transition requests, but also the indexes of the behaviors that can execute the requested actions (in those transition requests). Thus, it is not hard to see that the composition generator for $\mathsf{XtractBeh}(W)$ can be extracted by keeping those behavior delegations that transition a winning game state into another winning state in $\mathcal{G}_{\langle \mathcal{S}, \mathcal{T} \rangle}$.

Theorem 4.3. *Given the maximal winning set W for the safety game between a system $\mathcal{S}$ and target specification $\mathcal{T}$, the composition generator for $\mathsf{XtractBeh}(W)$ in $\mathcal{S}$ is given by the function:*

$$\mathit{CG}(h, \hat{t}, g, a, \hat{t}') = \{\mathit{comp}_{\mathcal{I}}(w) | w \in W, \mathit{comp}_{\mathcal{S}}(w) = \mathit{last}(h), \mathit{st}(w) \xrightarrow{g,a} \hat{t}' \text{ in } \mathit{XtractBeh}(W)\}.$$

Proof. We proceed in two steps. First, we show that a controller generated from CG is an exact compositions for $\mathsf{XtractBeh}(W)$ in S. Second, we show that all composition of $\mathsf{XtractBeh}(W)$ in $\mathcal{S}$ can be generated from CG.

1. Let C be a controller for $\mathsf{XtractBeh}(W)$ in $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ such that for all system histories h and target transitions $\hat{t} \xrightarrow{g,a} \hat{t}'$ in $\mathsf{XtractBeh}(W)$ it is the case that $\mathsf{C}(h, \hat{t}, g, a, \hat{t}') \in \mathsf{CG}(h, \hat{t}, g, a, \hat{t}')$. Assume that C is not an exact composition for $\mathsf{XtractBeh}(W)$ in system $\mathcal{S}$. Hence, there exists a trace $\tau = s^0_{\mathcal{T}} \xrightarrow{a} s^1_{\mathcal{T}} \cdots$ of the enacted target $\mathcal{E}_{\mathsf{XtractBeh}(W)}$ which cannot be realized by C in $\mathcal{S}$. Suppose that C can realize τ up to length ℓ, but cannot delegate the next transition action a arising from a target request $\hat{t} \xrightarrow{g,a} \hat{t}'$. Let h be a system history resulting from realising the ℓ length prefix of τ such that $\mathsf{C}(h, \hat{t}, g, a, \hat{t}')$ and $\mathsf{CG}(h, \hat{t}, g, a, \hat{t}')$ are undefined. Since states $\hat{t}$ and $\hat{t}'$ are from $\mathsf{XtractBeh}(W)$, there exists winning states $w, w' \in W$ such that $\mathsf{st}(w) = \hat{t}$, $\mathsf{st}(w') = \hat{t}'$, and $\langle w, w' \rangle \in \rho_s$, where ρ_s is the system-player's transition relation.

   Consequently, there exists a behavior index $\mathsf{comp}_{\mathcal{I}}(w)$ such that behavior $\mathcal{B}_{\mathsf{comp}_{\mathcal{I}}}$ can execute action a when the enacted system is in a state $\mathsf{comp}_{\mathcal{S}}(w)$ (the resulting enacted system state will be $\mathsf{comp}_{\mathcal{S}}(w')$). Therefore, $\mathsf{C}(h, \hat{t}, g, a, \hat{t}')$ and $\mathsf{CG}(h, \hat{t}, g, a, \hat{t}')$ will include $\mathsf{comp}_{\mathcal{I}}(w)$. As a result, the transition request $\hat{t} \xrightarrow{g,a} \hat{t}'$ resulting after length ℓ prefix of τ will be realized, thereby contradicting our assumption. Hence, C is an exact composition of $\mathsf{XtractBeh}(W)$ in system $\mathcal{S}$.

2. We use proof by contradiction to show that all compositions for $\mathsf{XtractBeh}(W)$ in $\mathcal{S}$ can be generated from CG. Assume that there exists a composition C of $\mathsf{XtractBeh}(W)$ in $\mathcal{S}$, such that C cannot be generated from CG. Thus, there exists a history h of system $\mathcal{S}$ and an h compatible transition request $t \xrightarrow{g,a} t'$ such that $\mathsf{C}(h, \hat{t}, g, a, \hat{t}') \notin \mathsf{CG}(h, \hat{t}, g, a, \hat{t}')$. Let $\mathsf{C}(h, \hat{t}, g, a, \hat{t}') = \mathsf{ind}$; that is, when the enacted system is in a state $\mathsf{last}(h)$, behavior $\mathcal{B}_{\mathsf{ind}}$ can successfully execute action a. Next, consider the game state $w = \langle b_1, \ldots, b_n, e, t, g, a, \hat{t}', \mathsf{ind} \rangle$ where $\langle b_1, \ldots, b_n, e \rangle = \mathsf{last}(h)$. Clearly, there exists a game state $w' = \langle b'_1, \ldots, b'_n, e', t', g', \sharp, \hat{t}', \mathsf{ind}' \rangle$ where $\langle b'_1, \ldots, b'_n, e' \rangle$ is a resulting enacted system state after executing action a by behavior $\mathcal{B}_{\mathsf{ind}}$ from $\mathsf{last}(h)$. Moreover, note that w' is a winning state (environment-player has no moves defined when requested action is $\sharp$) and $\langle w, w' \rangle \in \rho_s$. However, $w \notin W$, or CG would have accounted for the behavior index ind. This is a contradiction, since W is a maximal winning set, therefore any winning state w must be in W. Therefore, CG is the composition generator for $\mathsf{XtractBeh}(W)$ in $\mathcal{S}$.

$\square$

Theorems 4.2 and 4.3 show that SRTFs along with their composition generators can be computed by using synthesis via safety games. In terms of computational complexity, synthesising the maximal set of winning states can be performed in polynomial time wrt to the game state space [Bloem et al. 2011]. Once the maximal winning set has been computed, the SRTF can be extracted in polynomial time wrt the size of the set of maximal winning states.

Theorem 4.4. *Given the maximal winning set W for a safety game, the function $\mathsf{XtractBeh}(W)$ can be computed in time polynomial to $|W|$.*

Proof. To construct the states of $\mathsf{XtractBeh}(W)$, an algorithm needs only to visit each winning state in W once. The initial state of $\mathsf{XtractBeh}(W)$ can be computed in constant time. Finally, the transition relation of $\mathsf{XtractBeh}(W)$ can be built by visiting each winning state pair $w, w' \in W$ once; that is, $|W|^2$ pairs in the worst case. $\square$

Analogously, the composition generator for $\mathsf{XtractBeh}(W)$ in $\mathcal{S}$ can be computed in time polynomial in $|W|$. However, since the size of such space is exponential on the number of available behaviors, computing the SRTF and its composition generator can be performed in exponential time (for deterministic systems). Observe that, in the worst case, the optimisation problem itself is (at least) exponential in time, since it subsumes the classical behavior composition problem (which is exponential even under deterministic behaviors). Specifically, to check if a problem has an exact composition one can compute its optimal approximation and check if it is simulation equivalent to the original target (which can be performed in polynomial time with respect to the target specifications [Baier et al. 2008]).

Theorem 4.5. *Checking if a given target specification is the SRTF of a given behavior composition problem with only deterministic available behaviors is* EXPTIME*-complete.*

Theorem 4.5 shows that, in a deterministic setting, the computational complexity of optimising a behavior composition problem and checking the existence of an exact composition is the same. This is desirable because within the same time complexity as of the classical behavior composition problem, one can compute the optimal (best) target specification (that can actually be realized) as well as check if the original composition problem is solvable. Indeed, in the optimisation framework, unlike the classical setting, the end user is always guaranteed to obtain a constructive outcome (i.e., the SRTF and its composition generator) which can be used in practice.

4.1.2 ATL model checking

In this section, we consider yet another practically amenable way of implementing safety games, namely, Alternating-time Temporal Logic (ATL) model checking, to synthesise SRTFs. The advantage of reducing the composition problem to that of ATL reasoning is that it provides access to advanced model checking techniques and tools, such as MCMAS [Lomuscio et al. 2009], that are in active development within the agent community. ATL has already been used in the classical setting to synthesize exact compositions [De

Giacomo and Felli 2010] (see Section 2.5.5 for an overview). Here, we show how to synthesise the SRTF, along with its composition generator, in the optimisation setting by parsimoniously altering De Giacomo and Felli [2010]'s encoding.

Preliminaries

Alternating-time Temporal Logic [Alur et al. 2002] is a temporal logic for reasoning about the ability of agent coalitions in *multi-agent game structures*. It is a widely used logic to verify open systems where each component of the system can be modeled as an ATL agent. ATL formulae are built by combining propositional formulas, the usual temporal operators—namely, $\bigcirc$ ("in the next state"), $\Box$ ("always"), $\Diamond$ ("eventually"), and $\mathcal{U}$ ("strict until")—and a *coalition path quantifier* $\langle\langle A\rangle\rangle$ taking a set of agents A as parameter. As in CTL, which ATL extends, temporal operators and path quantifiers are required to alternate. Intuitively, an ATL formula $\langle\langle A\rangle\rangle\phi$, where A is a set of agents, holds in an ATL structure if by suitably choosing their moves, the agents in A *can force* ϕ *true*, no matter how other agents happen to move.

The semantics of ATL is defined in so-called *concurrent game structures* where, at each point, all agents simultaneously choose their moves from a finite set, and the next state deterministically depends on such choices. In comparison to well know Kripke structures, transitions in concurrent game structures involve simultaneous moves by all the agents (components) of the model, whereas a transition in a Kripke structure represents a single step of a (closed) system. Thus, concurrent agent structures provide appropriate semantics to model open (reactive) systems. More concretely, a *concurrent game structure* is a tuple $\mathcal{M} = \langle \mathcal{A}, Q, \mathcal{P}, Act, d, \mathcal{V}, \sigma\rangle$, where:

- $\mathcal{A} = \{1, \ldots, k\}$ is a finite set of agents;
- Q is the finite set of states;
- $\mathcal{P}$ is the finite set of propositions;
- Act is the set of all domain actions;
- $d : \mathcal{A} \times Q \mapsto 2^{Act}$ indicates all available actions for an agent in a state. Given a state $q \in Q$, $\mathcal{D}(q) = \times_{i=1}^{|\mathcal{A}|} d(i, q)$ denotes the set of all legal joint-moves in q;
- $\mathcal{V} : Q \mapsto 2^{\mathcal{P}}$ is the valuation function stating what is true in each state; and
- $\sigma : Q \times Act^{|\mathcal{A}|} \mapsto Q$ is the transition function mapping a state q and a joint-move $\vec{a} \in \mathcal{D}(q)$ to the resulting next state q'.

A *path* $\lambda = q_0q_1\cdots$ in a structure $\mathcal{M}$ is a, possibly infinite, sequence of states such that for each $i \geq 0$, there exists a joint-move $\vec{a_i} \in \mathcal{D}(q_i)$ for which $\sigma(q_i, \vec{a_i}) = q_{i+1}$. We use $\lambda[i] = q_i$ to denote the i-th state of λ, Λ to denote the set of all paths in $\mathcal{M}$, and

$\Lambda(q)$ to denote those starting in q. Also, $|\lambda|$ denotes the length of λ as the number of state transitions in λ: $|\lambda| = \ell$ if $\lambda = q_0q_1\ldots q_\ell$, and $|\lambda| = \infty$ if λ is infinite. When $0 \leq i \leq j \leq |\lambda|$, then $\lambda[i,j] = q_iq_{i+1}\ldots q_j$ is the finite subpath between the i-th and j-th steps in λ. Finally, a _computation_ in $\mathcal{M}$ is an infinite path in Λ.

To provide semantics to formulas $\langle\langle\cdot\rangle\rangle\varphi$, ATL relies on the notion of agent strategies. Technically, an ATL _strategy_ for an agent *agt* is a function $f_{agt} : Q^+ \mapsto Act$, where $f_{agt}(\lambda q) \in d(agt, q)$ for all $\lambda q \in Q^+$, stating a particular action choice of agent *agt* at path λq. A _collective strategy_ for a group of agents $A \subseteq \mathcal{A}$ is a set of strategies $F_A = \{f_{agt} \mid agt \in \mathcal{A}\}$ providing one specific strategy for each agent $agt \in A$. For a collective strategy F_A and an initial state q, $\mathit{out}(q, F_A)$ denotes the set of all _possible outcomes_ of F_A starting at state q. Intuitively, it is the set of all computations that may ensue when the agents in A behave as prescribed by F_A, and the remaining agents follow any arbitrary strategy [Alur et al. 2002]. Formally, given a collective strategy F_A for agents A, a computation $\lambda = q_0q_1\cdots$ is in the set $\mathit{out}(q_0, F_A)$ if for all positions $i \geq 0$, there is a joint-move $\langle a_1, \ldots, a_{|\mathcal{A}|}\rangle \in \mathcal{D}(q_i)$ such that *(i)* $a_{agt} = f_{agt}(\lambda[0,i])$ for all agents $agt \in A$, and *(ii)* $\sigma(q_i, \langle a_1, \ldots, a_{|\mathcal{A}|}\rangle) = q_{i+1}$. The semantics for the coalition modality is then defined as follows (here ϕ is a *path formula*; that is, it is preceded by $\bigcirc$, $\Box$, or $\mathcal{U}$, and $\mathcal{M}, \lambda \models \phi$ is defined in the usual way [Alur et al. 2002]):

$$\mathcal{M}, q \models \langle\langle A\rangle\rangle\phi \textit{ iff } \text{there is a collective strategy } F_A \text{ such that for all computations } \lambda \in \mathit{out}(q, F_A), \text{ we have } \mathcal{M}, \lambda \models \phi.$$

Given a concurrent game structure $\mathcal{M}$ and an ATL formula ϕ, the *model checking problem* of ATL asks for the set of states in $\mathcal{M}$ that satisfy formula ϕ. Let $[\phi]_{\mathcal{M}}$ denote the *maximal* set of states of $\mathcal{M}$ that satisfy ϕ. A state q in $\mathcal{M}$ is said to be *winning* for ϕ if $q \in [\phi]_{\mathcal{M}}$. Computationally, such a desired set of states can be computed in time polynomial in the size of the game structure and the length of the given formula [Alur et al. 2002]. Observe that the coalition modality only allows for implicit (existential) quantification over strategies. However, the model checking algorithms generally return *all* satisfying states; that is, $[\phi]$ represents the outcome of all possible strategies which the agents in coalition may follow to enforce the given formula.

Synthesising SRTFs via ATL model checking

For the classical setting (see Section 2.5.5), De Giacomo and Felli [2010] defined an ATL structure $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}$ with one agent for each available behavior and target specification, and one distinguished agent *contr* representing the controller. The task then involves model checking the special formula $\varphi = \langle\langle \mathit{contr}\rangle\rangle\Box(\bigwedge_{i=1,\ldots,n} \mathsf{state}_i \neq \mathit{error}_i)$ (against structure $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}$), which states that the controller agent has a strategy so that none of the n available behaviors end up in an *error* state. A behavior reaches a distinguished "error" state if the controller delegates an unfeasible action to it.

Surprisingly, it turns out that one can readily adapt De Giacomo and Felli's reduction to actually synthesize an SRTF for a, possibly non-solvable, *deterministic* composition problem (and to extract the corresponding controller generator). In line with the LTL synthesis approach, the key to this is to *include the target behavior in the coalition* so that the joint-strategy also includes selecting which transition from the actual target may be requested. We first show how to construct a concurrent game structure for ATL from a given behavior composition problem. Following that, we present the formula to be checked in such a model in order to get the SRTF.

Let $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ be an available system, with deterministic available behaviors $\mathcal{B}_i = \langle B_i, G_i, b_{i0}, \varrho_i \rangle$, for $1 \leq i \leq n$, environment $\mathcal{E} = \langle E, A, e_0, \rho \rangle$, and let $\mathcal{T} = \langle T, G, t_0, \varrho \rangle$ be a target specification. In line with the encoding for the classical setting, we modify each available behavior $\mathcal{B}_i$ by adding a new disconnected error state $error_i$, for each $1 \leq i \leq n$, and environment $\mathcal{E}$ by adding a disconnected error state $error_{\mathsf{env}}$. The error state captures wrong delegations by the controller, i.e., a behavior reaches the error state if it cannot execute the delegated action from its current state. We define the *concurrent game structure*, for a system $\mathcal{S}$ and target $\mathcal{T}$, as the tuple $\mathcal{M}_{\langle \mathcal{S}, \mathcal{T} \rangle} = \langle \{1, \ldots, n, \mathit{env}, \mathit{tgt}, \mathit{contr}\}, Q, \mathcal{P}, Act, d, \mathcal{V}, \delta \rangle$, where:

- There are $n + 3$ agents: one per available behavior (agents $1, \ldots, n$), one agent for the environment (agent env), one agent for the target module (agent tgt), and one agent for the controller (agent contr).
- The states Q of the game structure consists of the following finite range functions:
  - $\mathsf{state}_i \in B_i \cup \{error_i\}$, for $i \leq n$, returns the current state of behavior $\mathcal{B}_i$, and $\mathsf{state}_{\mathcal{E}} \in E \cup \{error_{\mathsf{env}}\}$ returns the current state of the environment;
  - $\mathsf{sch} \in \{i, \ldots, n, \mathsf{stop}\}$ returns the index of the available behavior that performed the last transition request. A special action stop is included to denote that no behavior was selected;
  - $\mathsf{req} \in \varrho \cup \{\mathsf{stop}\}$ returns the next transition request of the target. Given a transition request $r = \langle t_s, g, a, t_d \rangle$, we denote its action a by $\mathsf{act}(r)$. A special request stop is included to indicate that the target wants to stop requesting transitions;
- $\mathcal{P}$ is the set of propositions asserting value assignments to the above defined functions;
- $\mathcal{V}$ is the mapping from a game state q to the values returned by the above defined functions. For convenience, we write $\mathsf{state}_i(q) = b$ instead of $(\mathsf{state}_i = b) \in \mathcal{V}(q)$;
- $Act = \bigcup_{i \leq n} B_i \cup E \cup \{1, \ldots, n\} \cup \varrho \cup \{\mathsf{stop}\}$ is the set of domain actions;

- The function $d(j,q)$ captures the moves available to agent j at state q, and is defined as follows:
  - Available behaviors ($j \in \{1,\ldots,n\}$):

$$d(j,q) = \begin{cases} \{error_j\}, \text{ if there does not exist a transition } \mathsf{state}_j(q) \xrightarrow{g_j,a,} b_j \text{ in } \mathcal{B}_j \\ \quad \text{where } g_j(\mathsf{state}_\mathcal{E}(q)) = \mathtt{true},\ a = \mathsf{act}(\mathsf{req}(q)), \text{ and } \mathsf{req}(q) \neq \mathsf{stop}. \\ \{b \mid b \in \varrho_j(\mathsf{state}_j(q), g_j, \mathsf{act}(\mathsf{req}(q))), g_j(\mathsf{state}_\mathcal{E}(q)) = \mathtt{true}\}, \\ \quad \text{if there exists a transition } \mathsf{state}_j(q) \xrightarrow{g_j,a,} b_j \text{ in } \mathcal{B}_j \text{ where} \\ \quad g_j(\mathsf{state}_\mathcal{E}(q)) = \mathtt{true} \text{ and } a = \mathsf{req}(q) \neq \mathsf{stop}. \\ \{\mathsf{state}_j(q)\}, \text{ if } \mathsf{req}(q) = \mathsf{stop}. \end{cases}$$

  - Environment:

$$d(\mathit{env},q) = \begin{cases} \{error_{\mathit{env}}\}, \text{ if there does not exist a transition } \mathsf{state}_\mathcal{E}(q) \xrightarrow{a} e' \text{ in } \mathcal{E} \\ \quad \text{where } a = \mathsf{act}(\mathsf{req}(q)) \text{ and } \mathsf{req}(q) \neq \mathsf{stop}. \\ \{e \mid e \in \rho(\mathsf{state}_\mathcal{E}(q), \mathsf{act}(\mathsf{req}(q)))\}, \text{ if there exists a transition} \\ \quad \mathsf{state}_\mathcal{E}(q) \xrightarrow{\mathsf{act}(\mathsf{req}(q))} e' \text{ in } \mathcal{E} \text{ and } \mathsf{req}(q) \neq \mathsf{stop}. \\ \{\mathsf{state}_\mathcal{E}(q)\}, \text{ if } \mathsf{req}(q) = \mathsf{stop}. \end{cases}$$

  - Target behavior:

$$d(\mathit{tgt},q) = \begin{cases} \{\langle t,g,a,t''\rangle \in \varrho \mid g(\mathsf{state}_\mathcal{E}(q)) = \mathtt{true}, \mathsf{req}(q) = \langle t',g',a',t\rangle \in \varrho\} \\ \quad \cup\ \{\mathsf{stop}\}, \text{ if } \mathsf{req}(q) \neq \mathsf{stop}. \\ \{\mathsf{stop}\}, \text{ if } \mathsf{req}(q) = \mathsf{stop}. \end{cases}$$

  - Controller:

$$d(\mathit{contr},q) = \begin{cases} \{1,\ldots,n\}, \text{ if } \mathsf{req}(q) \neq \mathsf{stop}. \\ \{\mathsf{stop}\}, \text{ otherwise.} \end{cases}$$

- $\delta : Q \times Act \mapsto Q$ is the transition function, where $\delta(q, j_1, \ldots, j_n, j_{\mathit{env}}, j_{\mathit{tgt}}, j_{\mathit{contr}}) = q'$ if:
  - $\mathsf{req}(q) \neq \mathsf{stop}$ (the target requested a transition from its specification) and
    - $\mathsf{sch}(q') = j_{\mathit{contr}}$; that is, we store the index of the behavior to which the previous request was delegated by the controller;

  * $\mathsf{state}_i(q') = j_i$ if $i = j_{contr}$; that is, we evolve the behavior which was chosen by the controller;
  * $\mathsf{state}_i(q') = \mathsf{state}_i(q)$ for $i \in \{1, \dots, n\} \setminus j_{contr}$; that is, all other behaviors remain still;
  * $\mathsf{state}_{\mathcal{E}}(q') = j_{env}$; that is, we update the environment's state; and
  * $\mathsf{req}(q') = j_{tgt}$; that is, the target chooses its next transition request.
- $\mathsf{req}(q) = \mathsf{stop}$ (the target requested stop) and
  * $\mathsf{sch}(q') = j_{contr}$;
  * $\mathsf{state}_i(q') = \mathsf{state}_i(q)$ for $i \in \{1, \dots, n\}$; that is, all behaviors remain in their current states;
  * $\mathsf{state}_{\mathcal{E}}(q') = \mathsf{state}_{\mathcal{E}}(q)$; that is, environment state remains same; and
  * $\mathsf{req}(q') = \mathsf{stop}$; that is, the next target request is stop.

Conceptually, and structurally, a state in the encoded ATL game structure consists of the current states of the available behaviors and the environment, the current pending transition request from the target, and the previous controller delegation. Observe that actions of each agents are strongly linked to the propositions set in the next state. Specifically, an action of each available behavior and the environment is one of the following: *(i)* the successor of the current state if it can execute the current requested action; or *(ii)* an error state if the behavior cannot execute a legal (non stop) action; or *(iii)* the current state if the current target request is stop.

The target behavior moves by requesting either a legal transition as per the specification or the special stop action. As one would expect, once the target agent chooses the stop action, all future target agent actions are limited to the stop action. The last agent in our encoding, the controller, selects one of the available behaviors to execute the requested (non stop) action. The transition function of the game encodes the evolutions of all the components; that is, available behaviors, environment, target request, and the controller. To elaborate, for a state q, where $\mathsf{req}(q) \neq \mathsf{stop}$, and a move vector $\langle j_1, \dots, j_n, j_{env}, j_{tgt}, j_{contr} \rangle$, the next resulting state q' is such that only the state of behavior $\mathcal{B}_{j_{contr}}$ in q' is updated; all the other behaviors remain still. In addition to the delegated behavior, the state of environment in q' is updated to its action j_{env}. Similarly, the target request and the controller delegation in q' are updated to their actions j_{tgt} and j_{contr}, respectively. And for a state q, where $\mathsf{req}(q) = \mathsf{stop}$, a successor state of q is such that all the available behaviors and environment stay in their current state. We observe that our model is similar to the one used in [De Giacomo and Felli 2010] except that the target agent's requests involve transitions rather than actions and for inclusion of the special request stop.

Lastly, we model check the following ATL formula in the structure model $\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}$:

$$\tilde{\varphi} = \langle\langle \textsf{contr}, \textsf{tgt} \rangle\rangle \Box (\bigwedge_{i \in \{1,\ldots,n,\mathcal{E}\}} \textsf{state}_i \neq \mathit{error}_i).$$

Intuitively, the idea behind formula $\tilde{\varphi}$, as opposed to formula φ, is that the *coalition is now in control of what can be requested* (and what should not be). This suggests that the coalition has the ability to select which parts of the target can be executed without driving the available system into an "error" state (due to an impossible fulfilment of a request). In this case, similar to the LTL synthesis approach, a winning state in $[\tilde{\varphi}]_{\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}}$ is one in which either the target requests actions such that the controller can (always) legally delegate them to an available behavior, or the target requests stop from a non error state. By design, it is the case that the target agent can always do the stop action. Notably, the game semantics is defined such that once a stop is requested, the behaviors and the environment remain in their respective current states in the next, and all subsequent, successor states. This approach empowers the target agent to decide on the stop action when all available behaviors are in a state such that the next target transition cannot be delegated, thus avoiding error states.

Aligned with the LTL synthesis approach where the safety game $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$ is always winning for the system-player, in the ATL model checking technique, the formula $\tilde{\varphi}$ is satisfiable in *all* encoded models $\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}$. This is not surprising, rather it is expected, as in our encoding the *tgt* agent can do the stop action at any state, thereby continuously remaining in the same non-error state. Therefore, even before model checking $\tilde{\varphi}$, one knows that it will be satisfiable in any such encoded model $\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}$. In fact, the aim here is not to verify the satisfiability of $\tilde{\varphi}$, unlike in the classical setting where one checks the existence of an exact composition, but rather to use the model checking technology to compute the set $[\tilde{\varphi}]$. It follows then that one can extract an SRTF from the *maximal* winning set $[\tilde{\varphi}]_{\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}}$, as the following result demonstrates.

Theorem 4.6. *Let $[\tilde{\varphi}]_{\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}}$ be the maximal set of states satisfying $\tilde{\varphi}$ in the encoded concurrent game structure $\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}$ for a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E}\rangle$ and target specification $\mathcal{T} = \langle T, G, t_0, \varrho\rangle$. Then, behavior $\mathcal{T}_{[\tilde{\varphi}]} = \langle \hat{T}, G, \hat{t_0}, \hat{\varrho}\rangle$ is the SRTF for $\mathcal{T}$ in $\mathcal{S}$, where:*

- $\hat{T} = \{\langle \textsf{state}_1(q), \ldots, \textsf{state}_n(q), \textsf{state}_{\mathcal{E}}(q), t\rangle \mid q \in [\tilde{\varphi}]_{\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}}, \textsf{req}(q) = \langle t, g, a, t'\rangle\} \cup \{\hat{t_0}\}$. *Given a state $q \in [\tilde{\varphi}]_{\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}}$ we denote $\langle \textsf{state}_1(q), \ldots, \textsf{state}_n(q), \textsf{state}_{\mathcal{E}}(q), t\rangle$ by $\textsf{st}(q)$, enacted system state $\langle \textsf{state}_1(q), \ldots, \textsf{state}_n(q), \textsf{state}_{\mathcal{E}}(q)\rangle$ by $\textsf{st}_{\mathcal{S}}(q)$, and target state t by $\textsf{st}_{\mathcal{T}}(q)$;*
- $\hat{t_0} = \langle b_{10}, \ldots, b_{n0}, e_0, t_0\rangle$ *is the initial state of $\mathcal{T}_{[\tilde{\varphi}]}$;*
- $\hat{\varrho}(\textsf{st}(q), g, a, \textsf{st}(q'))$, *where $\textsf{req}(q) = \langle \textsf{st}_{\mathcal{T}}(q), g, a, \textsf{st}_{\mathcal{T}}(q')\rangle$,* iff *it is the case that:*
  - $q, q' \in [\tilde{\varphi}]_{\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}}$; *and*

– $\delta(q, j_1, \ldots, j_{n+3}) = q'$ *for some* $j_1, \ldots, j_{n+3}$.

Proof. The proof amounts to showing that *(i)* $\mathcal{T}_{[\tilde{\varphi}]}$ is an RTF of $\mathcal{T}$ in $\mathcal{S}$, and *(ii)* $\mathcal{T}_{[\tilde{\varphi}]}$ is the SRTF of $\mathcal{T}$ in $\mathcal{S}$.

1. $\mathcal{T}_{[\tilde{\varphi}]}$ is an RTF: Due to the definition of $\hat{\varrho}$ in $\mathcal{T}_{[\tilde{\varphi}]}$ and move function d for target agent *tgt* in the encoding $\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}$, it holds that $\hat{t} \xrightarrow{g,a} \hat{t}' \in \hat{\varrho}$ if $\mathsf{st}_{\mathcal{T}}(\hat{t}) \xrightarrow{g,a} \mathsf{st}_{\mathcal{T}}(\hat{t}') \in \varrho$. Now, consider the relation $\mathcal{R} \subseteq \hat{T} \times T$ such that $(\hat{t}, t) \in \mathcal{R}$ if $\mathsf{st}_{\mathcal{T}}(\hat{t}) = t$. Then, for a tuple $(\hat{t}, t) \in \mathcal{R}$, for all transitions $\hat{t} \xrightarrow{g,a} \hat{t}'$ in $\mathcal{T}_{[\tilde{\varphi}]}$ there exists a transition $t \xrightarrow{g,a} t'$ in $\mathcal{T}$ such that $(\hat{t}', t') \in \mathcal{R}$. Note that $\mathcal{R}$ is an E-simulation relation of $\mathcal{T}_{[\tilde{\varphi}]}$ by $\mathcal{T}$. Since by definition $(\hat{t_0}, t_0) \in \mathcal{R}$, we have that $\mathcal{T}_{[\tilde{\varphi}]} \preceq_{\mathcal{E}} \mathcal{T}$.

   Next, we show that $\mathcal{T}_{[\tilde{\varphi}]}$ is an *effective* target fragment (Definition 3.8). Let $\mathcal{E}_{\mathcal{T}_{[\tilde{\varphi}]}} = \langle S, A, s_0, \varrho_S\rangle$ be the enacted target behavior of $\mathcal{T}_{[\tilde{\varphi}]}$ in environment $\mathcal{E}$ and $\mathcal{T}_{[\tilde{\varphi}]}^{\uparrow G}$ be the transition system obtained by projecting $\mathcal{T}_{[\tilde{\varphi}]}$'s guards. Consider the relation $\mathcal{P} \subseteq \hat{T} \times S$ such that $(\hat{t}, s) \in \mathcal{P}$ if $\hat{t} = \mathsf{tgt}(s)$ and $\mathsf{state}_{\mathcal{E}}(q) = \mathsf{env}(s)$, where $\mathsf{st}(q) = \hat{t}$.[7] Then, $\mathcal{P}$ is the simulation relation of $\mathcal{T}_{[\tilde{\varphi}]}^{\uparrow G}$ by $\mathcal{E}_{\mathcal{T}_{[\tilde{\varphi}]}}$: From the move function d for target agent *tgt* defined in $\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}$, we know that for a transition $\hat{t} \xrightarrow{g,a} \hat{t}'$ there exists guard $g \in G$ such that $g(\mathsf{state}_{\mathcal{E}}(q)) = \mathtt{true}$, where $\mathsf{st}(q) = \hat{t}$, and since the environment agent cannot be in the error state we know that $\mathsf{state}_{\mathcal{E}}(q) \xrightarrow{a} \mathsf{state}_{\mathcal{E}}(q')$ in $\mathcal{E}$, where $\mathsf{st}(q) = \hat{t}$ and $\mathsf{st}(q') = \hat{t}'$. Hence, for a tuple $(\hat{t}, s) \in \mathcal{P}$, $\hat{t} \xrightarrow{a} \hat{t}'$ in $\mathcal{T}_{[\tilde{\varphi}]}^{\uparrow G}$ implies that there exists a transition $s \xrightarrow{a} s'$ in $\mathcal{E}_{\mathcal{T}_{[\tilde{\varphi}]}}$ such that $(\hat{t}', s') \in \mathcal{R}$. Trivially, $(\hat{t_0}, s_0) \in \mathcal{R}$, therefore $\mathcal{E}_{\mathcal{T}_{[\tilde{\varphi}]}}$ simulates $\mathcal{T}_{[\tilde{\varphi}]}^{\uparrow \mathcal{G}}$. Thus, $\mathcal{T}_{[\tilde{\varphi}]}$ is an effective fragment of $\mathcal{T}$ in $\mathcal{S}$.

   Next, we show that $\mathcal{T}_{[\tilde{\varphi}]}$ has an exact composition in $\mathcal{S}$; that is, $\mathcal{T}_{[\tilde{\varphi}]}$ is simulated by the enacted system $\mathcal{E}_{\mathcal{S}} = \langle S, A, s_0, \varrho_S\rangle$.[8] Let $\mathcal{Z} \subseteq \hat{T} \times \mathcal{S}$ be a relation such that $(\hat{t}, s) \in \mathcal{Z}$ iff $\mathsf{st}_{\mathcal{S}}(q) = s$, where $\mathsf{st}(q) = \hat{t}$. Consider a tuple $(\hat{t}, s) \in \mathcal{Z}$, a transition $\hat{t} \xrightarrow{g,a} \hat{t}'$ from $\hat{t}$, and game states q, q' such that $\mathsf{st}(q) = \hat{t}$ and $\mathsf{st}(q') = \hat{t}'$. From (winning) game states q and q' we can conclude that $s \xrightarrow{a,\mathsf{sch}(q')} s'$ in $\mathcal{E}_{\mathcal{S}}$ where a is the action in transition $\mathsf{req}(q)$, and $s' = \mathsf{st}_{\mathcal{S}}(q')$ such that $\mathsf{st}(q') = \hat{t}'$. Thus, $(\hat{t}', s') \in \mathcal{Z}$, and so $\mathcal{Z}$ is a simulation relation of $\mathcal{T}_{[\tilde{\varphi}]}$ by $\mathcal{S}$. Clearly, by definition of $\mathcal{Z}$, $(\hat{t}_0, s_0) \in \mathcal{Z}$, therefore $\mathcal{T}_{[\tilde{\varphi}]} \preceq \mathcal{E}_{\mathcal{S}}$.

2. $\mathcal{T}_{[\tilde{\varphi}]}$ is the SRTF: Let $\tilde{\mathcal{T}} = \langle \tilde{T}, \tilde{G}, \tilde{t_0}, \tilde{\varrho}\rangle$ be the SRTF of $\mathcal{T}$ in $\mathcal{S}$. Therefore, by definition of SRTFs (Definition 3.10), we have that $\mathcal{T}_{[\tilde{\varphi}]} \preceq_{\mathcal{E}} \tilde{\mathcal{T}} \preceq_{\mathcal{E}} \mathcal{T}$. We use proof by contradiction to show that $\mathcal{T}_{[\tilde{\varphi}]}$ and $\tilde{\mathcal{T}}$ are E-simulation equivalent. Assume that $\mathcal{T}_{[\tilde{\varphi}]}$ does not E-simulate $\tilde{\mathcal{T}}$, i.e., $\tilde{t_0} \not\preceq_{\mathcal{E}} \hat{t_0}$. Therefore, there exists a trace $\tilde{\tau} = \tilde{t^0} \xrightarrow{\tilde{g}^1,a^1} \cdots \xrightarrow{\tilde{g}^n,a^n} \tilde{t^n}$ of $\tilde{\mathcal{T}}$ such that for all traces $\hat{\tau} = \hat{t^0} \xrightarrow{g^1,a^1} \cdots \xrightarrow{g^n,a^n} \hat{t^n}$ of

[7]Recall that $\mathsf{tgt}(s)$ and $\mathsf{env}(s)$ denote the target state and environment state components of the enacted state s, respectively.

[8]Note that for deterministic behaviors ND-simulation and simulation are equivalent.

$\hat{\mathcal{T}}$, there exists a transition $\tilde{t^n} \xrightarrow{\tilde{g}^{n+1},a^{n+1}} \tilde{t^{n+1}}$ in $\tilde{\mathcal{T}}$ for which there is no transition $\hat{t^n} \xrightarrow{g^{n+1},a^{n+1}} \hat{t^{n+1}}$ in $\mathcal{T}_{[\tilde{\varphi}]}$. That is, $\tilde{\tau}$ cannot be E-simulated by any trace of $\mathcal{T}_{[\tilde{\varphi}]}$.

Consider the concurrent game structure $\mathcal{M}_{\langle\mathcal{S},\tilde{\mathcal{T}}\rangle}$ for target specification $\tilde{\mathcal{T}}$ and system $\mathcal{S}$. Since $\tilde{\mathcal{T}}$ has an exact composition in $\mathcal{S}$, the maximal set of winning states $\tilde{W} = [\tilde{\varphi}]_{\mathcal{M}_{\langle\mathcal{S},\tilde{\mathcal{T}}\rangle}}$ will accommodate all $\tilde{\mathcal{T}}$'s transition requests. Let $\tilde{W}_{\tilde{\tau}} \subseteq \tilde{W}$ be the set of winning states catering for $\tilde{\tau}$'s transitions; that is, $\langle b_1, \ldots, b_n, e, \tilde{t_s}, \tilde{g}, a, \tilde{t_d}, ind\rangle \in \tilde{W}_{\tilde{\tau}}$ if $\tilde{t_s} \xrightarrow{\tilde{g},a} \tilde{t_d} = \tilde{t^i} \xrightarrow{\tilde{g}^{a+1},a^{i+1}} \tilde{t^{i+1}}$ for some $i \leq n$. Note that the transition $\tilde{t^n} \xrightarrow{\tilde{g}^{n+1},a^{n+1}} \tilde{t^{n+1}}$ which breaks the simulation of $\tilde{\mathcal{T}}$ by $\mathcal{T}_{[\tilde{\varphi}]}$ is also included. Now consider the set of states in the concurrent game structure $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}$ for target $\mathcal{T}$ and system $\mathcal{S}$ defined by: $U = \{\langle b_1, \ldots, b_n, t_s, g, a, t_d, ind\rangle \mid \langle b_1, \ldots, b_n, \tilde{t_s}, \tilde{g}, a, \tilde{t_d}, ind\rangle \in \tilde{W}_{\tilde{\tau}}, \tilde{t_s} \preceq_{\mathcal{E}} t_s, \tilde{t_d} \preceq_{\mathcal{E}} t_d\}$. That is, the states are similar to winning states in $\tilde{W}_{\tilde{\tau}}$ except for the transition requests. The transition requests of $\tilde{\mathcal{T}}$ are replaced by the transitions requests from $\mathcal{T}$ such the corresponding states are in E-simulation. Note that these states are not only legal game states but also winning in the game structure $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}$, i.e., $U \subseteq W$: allocation of simulating transitions to the same indexes as in the winning states of $\mathcal{M}_{\langle\mathcal{S},\tilde{\mathcal{T}}\rangle}$ will still be winning in $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}$. Therefore, U contains winning states having transition requests $t \xrightarrow{g^{n+1},a^{n+1}} t'$ of $\mathcal{T}$, corresponding to $\tilde{\tau}$'s transition $\tilde{t^n} \xrightarrow{\tilde{g}^{n+1},a^{n+1}} \tilde{t^{n+1}}$ such that $\tilde{t^n} \preceq_{\mathcal{E}} t$ and $\tilde{t^{n+1}} \preceq_{\mathcal{E}} t'$. Consequently, there will be a $\hat{\tau}$'s transition $\hat{t^n} \xrightarrow{g^{n+1},a^{n+1}} \hat{t^{n+1}}$ in $\mathcal{T}_{[\tilde{\varphi}]}$ where $\tilde{t^n} \preceq_{\mathcal{E}} \mathsf{st}_{\mathcal{T}}(\hat{t^n})$ and $\tilde{t^{n+1}} \preceq_{\mathcal{E}} \mathsf{st}_{\mathcal{T}}(\hat{t^{n+1}})$, which contradicts the assumption. Therefore, $\mathcal{T}_{[\tilde{\varphi}]}$ and $\tilde{\mathcal{T}}$ are E-simulation equivalent and hence $\mathcal{T}_{[\tilde{\varphi}]}$ is the SRTF of $\mathcal{T}$ in $\mathcal{S}$.

□

Given a composition problem, one can now use ATL model checking to compute its SRTF. We denote the SRTF, of a target specification $\mathcal{T}$ in a system $\mathcal{S}$, encoded in $[\tilde{\varphi}]_{\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}}$, by $\mathcal{T}_{[\tilde{\varphi}]}$. Note that the concurrent game states also store the "good" controller delegations that ensure the system never reaches an error state. From these, the controller generator for $\mathcal{T}_{[\tilde{\varphi}]}$ can then be extracted by keeping those behavior delegations that transition a winning game state into another winning state in $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}$.

Theorem 4.7. *Given the SRTF $\mathcal{T}_{[\tilde{\varphi}]}$ of a target specification $\mathcal{T}$ and system $\mathcal{S}$, the composition generator of $\mathcal{T}_{[\tilde{\varphi}]}$ in $\mathcal{S}$ is given by the function:*

$$\mathit{CG}(h, \hat{t}, g, a, \hat{t}') = \{\mathit{sch}(q') \mid \mathit{st}_{\mathcal{S}}(q) = \mathit{last}(h), \mathit{st}(q') = \hat{t}', \mathit{st}(q) \xrightarrow{g,a} \hat{t}' \text{ in } \mathcal{T}_{[\tilde{\varphi}]},\ q, q' \in [\tilde{\varphi}]\}.$$

Proof. First, we show that a controller generated from CG is an exact composition for $\mathcal{T}_{[\tilde{\varphi}]}$ in $\mathcal{S}$. Second, we show that all compositions of $\mathcal{T}_{[\tilde{\varphi}]}$ in $\mathcal{S}$ can be generated from CG.

1. Let C be a controller for $\mathcal{T}_{[\tilde{\varphi}]}$ in system $\mathcal{S} = \langle\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E}\rangle$ such that for all system histories h and transitions $\hat{t} \xrightarrow{g,a} \hat{t}'$ it is the case that $\mathsf{C}(h, \hat{t}, g, a, \hat{t}') \in \mathsf{CG}(h, \hat{t}, g, a, \hat{t}')$.

Assume that C is not an exact composition for $\mathcal{T}_{[\tilde{\varphi}]}$ in system $\mathcal{S}$. Hence, there exists a trace $\tau = s^0_{\mathcal{T}} \xrightarrow{a} s^1_{\mathcal{T}} \cdots$ of the enacted target $\mathcal{E}_{\mathcal{T}_{[\tilde{\varphi}]}}$ which cannot be realized by C in $\mathcal{S}$. Suppose that C can realize τ up to length ℓ, but cannot delegate the next transition action a arising from a target request $\hat{t} \xrightarrow{g,a} \hat{t}'$. Let h be a system history resulting from realising the ℓ length prefix of τ, such that $\mathsf{C}(h, \hat{t}, g, a, \hat{t}')$ and $\mathsf{CG}(h, \hat{t}, g, a, \hat{t}')$ are undefined. Since states $\hat{t}$ and $\hat{t}'$ are from $\mathcal{T}_{[\tilde{\varphi}]}$, there exists winning (non error) states $q, q' \in [\tilde{\varphi}]$ such that $\mathsf{st}(q) = \hat{t}$ and $\mathsf{st}(q') = \hat{t}'$. Consequently, there exists a behavior index $\mathsf{sch}(q')$ such that behavior $\mathcal{B}_{\mathsf{sch}(q')}$ can execute action a when the enacted system is in a state $\mathsf{st}_{\mathcal{S}}(q)$ (the resulting enacted system state will be $\mathsf{st}_{\mathcal{S}}(q')$). Therefore, $\mathsf{C}(h, \hat{t}, g, a, \hat{t}')$ and $\mathsf{CG}(h, \hat{t}, g, a, \hat{t}')$ will include $\mathsf{sch}(q')$. As a result, the transition request $\hat{t} \xrightarrow{g,a} \hat{t}'$ resulting after ℓ length prefix of τ will be realized, thereby contradicting our assumption. Hence, C is an exact composition of $\mathcal{T}_{[\tilde{\varphi}]}$ in $\mathcal{S}$.

2. We use proof by contradiction to show that all compositions for $\mathcal{T}_{[\tilde{\varphi}]}$ in $\mathcal{S}$ can be generated from CG. Assume that there exists a composition C of $\mathcal{T}_{[\tilde{\varphi}]}$ in $\mathcal{S}$, such that C cannot be generated from CG. Thus, there exists a history h of system $\mathcal{S}$ and an h compatible transition request $\hat{t} \xrightarrow{g,a} \hat{t}'$ such that $\mathsf{C}(h, \hat{t} \xrightarrow{g,a} \hat{t}') \notin \mathsf{CG}(h, \hat{t} \xrightarrow{g,a} \hat{t}')$. Let $\mathsf{C}(h, \hat{t} \xrightarrow{g,a} \hat{t}') = \mathsf{ind}$; that is, when the enacted system is in a state $\mathsf{last}(h)$, behavior $\mathcal{B}_{\mathsf{ind}}$ can successfully execute action a. Next, consider the game states q, q' such that $\mathsf{st}_{\mathcal{S}}(q) = \mathsf{last}(h)$, $\mathsf{req}(q) = \langle \hat{t}, g, a, \hat{t}' \rangle$ and also consider a move vector $\vec{j}$ where q' is the successor of q for move vector $\vec{j}$. Clearly, q and q' exist with $\mathsf{sch}(q') = \mathsf{ind}$ as we know that behavior $\mathcal{B}_{\mathsf{ind}}$ can execute action a from $\mathsf{last}(h) = \mathsf{st}_{\mathcal{S}}(q)$. Moreover, $q' \notin [\tilde{\varphi}]$ else CG would have accounted for the behavior index ind. This is a contradiction: q' is indeed a winning state as from q' the target agent can move stop and since $[\tilde{\varphi}]$ is the maximal winning set, any winning state q' must be in $[\tilde{\varphi}]$. Therefore, CG is the composition generator for $\mathcal{T}_{[\tilde{\varphi}]}$ in $\mathcal{S}$.

□

In terms of computational complexity, the model checking task over ATL can be performed in polynomial time wrt the size of the game structure [Alur et al. 2002]. One can see, on the lines of Theorem 4.4, that $\mathcal{T}_{[\tilde{\varphi}]}$ can be computed in time polynomial in $|[\varphi]|$. Since the size of such space is exponential in the number of available behaviors, computing the SRTF via ATL model checking can be performed in exponential time with respect to the number of available behaviors (for deterministic systems).

To recap, in this section we provided two practically amenable techniques enjoying efficient state-of-the-art tools to obtain SRTFs, but only for deterministic systems. However, it remains to be seen if these approaches are sufficient for handling composition problems involving nondeterministic available behaviors.

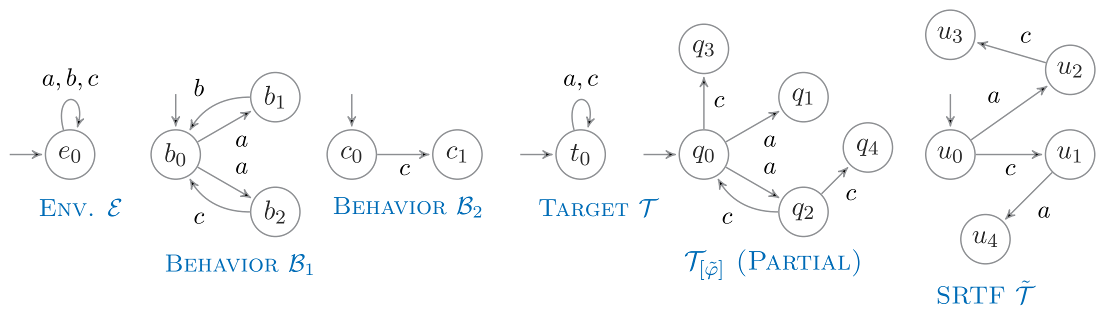


Figure 4.1: Instance where safety game synthesis and ATL model checking approaches are not sufficient.

4.2 Computing SRTFs for nondeterministic systems

Unfortunately, when we consider nondeterministic available behaviors, safety game and ATL model checking approaches are no longer sufficient. For systems with nondeterministic behaviors, the target specification $\mathcal{T}_{[\tilde{\varphi}]}$ extracted from the encoded ATL game structure $\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}$ (or $\mathsf{XtractBeh}(W)$ extracted from winning states W of safety game $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$) may not be realizable in the given system; that is, an exact composition may not exist for $\mathsf{XtractBeh}(W)$ in system $\mathcal{S}$. Recall that for nondeterministic behaviors, a target specification is realizable if the enacted target behavior is ND-simulated by the enacted system. To briefly recap, an ND-simulation requires the simulation property to be maintained across *all* nondeterministic behavior evolutions, whereas in plain simulation only one evolution needs to satisfy the simulation property (see Section 2.5.3 for details). Indeed, there are examples where $\mathcal{E}_{\mathcal{T}_{[\tilde{\varphi}]}} \preceq_{\text{ND}} \mathcal{E}_{\mathcal{S}}$ does *not* hold due to the nondeterminism present in $\mathcal{E}_{\mathcal{S}}$. In such cases, the computed target specification $\mathcal{T}_{[\tilde{\varphi}]}$ is a sort of target behavior in which agent transition requests are *conditional* on the nondeterministic execution of available behaviors. However, the agent using the target is not meant to have observability on such behaviors, and so it cannot decide its request upon such contingencies. To elaborate, in contrast to the controller, the user of the target specification cannot observe the internal state of the behaviors and its requests must be honored by an exact controller irrespective of the nondeterminism in the available behaviors. An exact controller does this by basing its delegation on which nondeterministic evolution ensued. Thus, the same target's request may be delegated by a composition to different available behaviors in different system histories.

Example 4.1. Figure 4.1 shows a simple behavior composition problem, with two behaviors $\mathcal{B}_1$ and $\mathcal{B}_2$, environment $\mathcal{E}$, and target specification $\mathcal{T} = \langle t_0, \{g\}, t_0, \varrho\rangle$, where $g(e_0) = \mathtt{true}$. Take $\mathcal{T}_{[\tilde{\varphi}]}$ (only a part of $\mathcal{T}_{[\tilde{\varphi}]}$ is shown in Figure 4.1) as a candidate for SRTF of $\mathcal{T}$ in system $\mathcal{S} = \langle \mathcal{B}_1, \mathcal{B}_2, \mathcal{E}\rangle$. The trace $\tau = \langle q_0, e_0\rangle \xrightarrow{g,a} \langle q_2, e_0\rangle \xrightarrow{g,c} \langle q_0, e_0\rangle \xrightarrow{g,a} \langle q_2, e_0\rangle$ of enacted target $\mathcal{E}_{\mathcal{T}_{[\tilde{\varphi}]}}$ cannot be realized in the enacted system $\mathcal{E}_{\mathcal{S}}$ because of nondeterminism present in behavior $\mathcal{B}_1$. See that behavior $\mathcal{B}_1$ may evolve to state b_1 or b_2 after the first action a is delegated. If $\mathcal{B}_1$ evolves to its state b_1, then the second action c can only be executed by behavior $\mathcal{B}_2$. Once that happens, none of the behaviors $\mathcal{B}_1$ or $\mathcal{B}_2$ can execute the third action a of trace τ. In terms of ND-simulation, note that $\langle e_0, q_0 \rangle \not\preceq_{\text{ND}} \langle e_0, b_0, c_0 \rangle$ because action a can be executed from enacted target state $\langle e_0, q_0 \rangle$ but not from enacted system state $\langle e_0, b_1, c_1 \rangle$. Lastly, note that the target specification $\tilde{\mathcal{T}}$ is actually the SRTF of $\mathcal{T}$ in $\mathcal{S}$. $\square$

The underlying reason for this conditionality in SRTFs lies in the semantics of game structures and ATL model checking. The agents/players in safety games and ATL model checking *have full observability on the game states*. Hence, the target agent can observe the nondeterministic behavior evolution and act optimistically on such an evolution. This issue does not arise in the classical case as there the complete target needs to be realized. Hence, irrespective of the behavior evolution, the target will still seek to request all the next actions. On the other hand, in our setting the target must request only those transitions which can actually be executed in the system.

In reality, what we need is for the target to be *conformant*, i.e., independent of conditions on the available behaviors states. That is, irrespective of how a behavior might evolve due to nondeterminism, the *same* sequence of subsequent target requests must be executable in the system. Hence inspired by the literature on planning under uncertainty we construct a type of *belief state* [Bonet and Geffner 2000, Palacios and Geffner 2006], and in turn, the *"belief-level" full enacted system*. The idea behind generating belief states is to track the states where the enacted system could be in from the target agent's perspective.

Our technique for synthesizing the SRTF relies on two simple operations on transition systems, namely, a specific *synchronous product* and a *conformance* enforcing procedure. Roughly speaking, the technique involves two steps:

1. We take the *synchronous product* of the enacted system $\mathcal{E}_\mathcal{S}$ and the target $\mathcal{T}$, yielding the structure $\mathcal{F}_{\langle \mathcal{S}, \mathcal{T} \rangle}$.

2. We modify $\mathcal{F}_{\langle \mathcal{S}, \mathcal{T} \rangle}$ to *enforce conformance* on its states that cannot be distinguished by the user of the target.

The full enacted system models the behavior that emerges from joint parallel execution of the enacted system and the target.

Definition 4.1 (Full enacted system). Given the enacted system $\mathcal{E}_\mathcal{S} = \langle S, A, s_0, \delta \rangle$ for a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$ and a target specification $\mathcal{T} = \langle T, G, t_0, \varrho \rangle$, the <u>*full enacted system*</u> of $\mathcal{T}$ and $\mathcal{S}$, denoted by $\mathcal{E}_\mathcal{S} \times \mathcal{T}$, is a tuple $\mathcal{F}_{\langle \mathcal{S}, \mathcal{T} \rangle} = \langle F, A, f_0, \gamma \rangle$, where:

- $F = S \times T$ is the finite set of $\mathcal{F}_{\langle \mathcal{S}, \mathcal{T} \rangle}$'s states. When $f = \langle s, t \rangle$, we denote s by $\mathsf{esys}(f)$, t by $\mathsf{tgt}(f)$, and we extend the env function to $\mathsf{env}(f) = \mathsf{env}(s)$;

- $f_0 = \langle s_0, t_0 \rangle \in F$, is $\mathcal{F}_{\langle \mathcal{S},\mathcal{T} \rangle}$'s initial state;
- A is the set of actions; and
- $\gamma \subseteq F \times G \times A \times \{1, \ldots, n\} \times F$ is $\mathcal{F}_{\langle \mathcal{S},\mathcal{T} \rangle}$'s transition relation, where $\langle f, g, a, k, f' \rangle \in \gamma$, or $f \xrightarrow{g,a,k} f'$ in $\mathcal{F}_{\langle \mathcal{S},\mathcal{T} \rangle}$ *iff*:
  - there exists guard $g \in G$ such that $g(\mathsf{env}(f)) = \mathtt{true}$, $\mathsf{tgt}(f) \xrightarrow{g,a} \mathsf{tgt}(f')$ in $\mathcal{T}$; and
  - $\mathsf{sys}(f) \xrightarrow{a,k} \mathsf{sys}(f')$ in $\mathcal{E}_{\mathcal{S}}$.

□

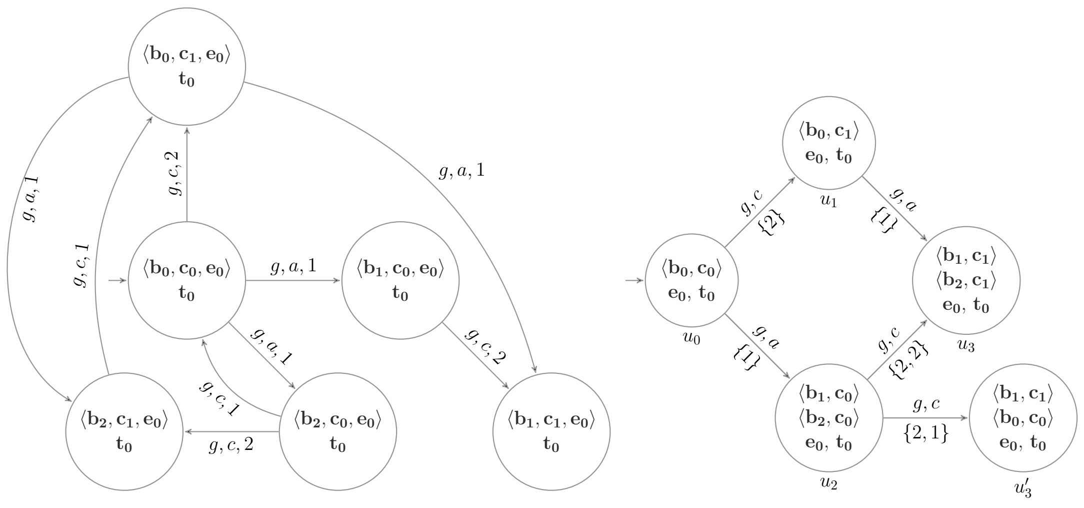
Conceptually, the full enacted system is the *synchronous product* of the enacted system and the target behavior. Each state of the full enacted system consists of two components, an enacted system state and a target state. Observe that the transition relation of the full enacted system requires both the enacted system and the target to evolve *jointly*. Structurally, this is made evident by including the target guard and behavior index in the transition relation.

Example 4.2. Figure 4.2 shows the full enacted system of target specification $\mathcal{T}$ in system $\mathcal{S} = \langle \mathcal{B}_1, \mathcal{B}_2, \mathcal{E} \rangle$. Note that from the initial state $\langle \langle b_0, c_0, e_0 \rangle, t_0 \rangle$ action a executed by behavior $\mathcal{B}_1$ may result in either $s_1 = \langle \langle b_1, c_0, e_0 \rangle, t_0 \rangle$ or $s_2 = \langle \langle b_2, c_0, e_0 \rangle, t_0 \rangle$. Clearly, the sequences of actions that can be requested from s_1 and s_2 are different. For instance, only the action c can be executed from s_1; however, from s_2 the sequence of actions $c \cdot c$ and $c \cdot a$ both are executable. □

Note the similarity between the full enacted system and the safety game structure built in Section 4.1.1. First, a state in the full enacted system is exactly like a game state, except for transition request and behavior delegation, which are used as transition labels in the full enacted system. Second, by construction, the transition relation in the full enacted system includes only target transitions which can be successfully delegated. In fact, both LTL synthesis and ATL model checking are efficient approaches to build the full enacted system. Also, from the construction of $\mathcal{F}_{\langle \mathcal{S},\mathcal{T} \rangle}$ we can conclude that it can be built in exponential time in the number of behaviors and polynomial in the number of states in each behavior.

Definition 4.2 (Belief-level full enacted system). Given a full enacted system $\mathcal{F}_{\langle \mathcal{S},\mathcal{T} \rangle} = \langle F, A, f_0, \gamma \rangle$ for a target $\mathcal{T} = \langle T, G, t_0, \varrho \rangle$ and a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ where $\mathcal{B}_i = \langle B_i, G_i, b_{i0}, \varrho_i \rangle$ for $i \leq n$ and $\mathcal{E} = \langle E, A, e_0, \rho \rangle$, the <u>*belief-level full enacted system*</u> is a tuple $\mathcal{K}_{\langle \mathcal{S},\mathcal{T} \rangle} = \langle Q, G, q_0, \delta_K \rangle$, where:

- $Q = 2^{(B_1 \times \cdots \times B_n)} \times E \times T$ is $\mathcal{K}_{\langle \mathcal{S},\mathcal{T} \rangle}$'s set of states. When $q = \langle \{s_1, \ldots, s_\ell\}, e, t \rangle \in Q$ we denote $\{s_1, \ldots, s_\ell, e\}$ by $\mathsf{ksys}(q)$ and t by $\mathsf{tgt}(q)$;



Full enacted system.

Belief-level system.

Figure 4.2: Full enacted system and belief level system for behavior composition problem shown in Figure 4.1.

- $q_0 = \langle \{s_0\}, e_0, t_0\rangle$ such that $f_0 = \langle (s_0, e_0), t_0\rangle$, is the initial state; and
- $\delta_K \subseteq Q \times G \times A \times Q$ is $\mathcal{K}_{\langle \mathcal{S},\mathcal{T}\rangle}$'s transition relation such that $\langle q, g, a, q'\rangle \in \delta_K$, where $q = \langle S, e, t\rangle$ and $q' = \langle S', e', t'\rangle$, iff :
  1. there exists a set $\mathsf{Indx} = \{\langle s_1 : k_1\rangle, \ldots, \langle s_\ell : k_\ell\rangle\}$ such that $\{s_1, \ldots, s_\ell\} = S$; and $\langle s_i \cup \{e\}, t\rangle \xrightarrow{g,a,k_i} \langle s' \cup \{e'\}, t'\rangle$ in $\mathcal{F}_{\langle \mathcal{S},\mathcal{T}\rangle}$ for all $i \leq \ell$. That is, the action a must be executable from all full enacted system states in q; and
  2. $S' = \bigcup_{\langle s:i\rangle \in \mathsf{Indx}} \{s' \mid \langle \langle s \cup \{e\}, t\rangle, g, a, i, \langle s' \cup \{e'\}, t'\rangle\rangle \in \gamma\}$. That is, S' should account for all (nondeterministic) evolutions from each full enacted system state in q resulting from action a. Let $f_{\mathcal{I}}(q \xrightarrow{g,a} q')$ be a function that returns the set Indx associated with a transition $q \xrightarrow{g,a} q' \in \delta_K$.

□

Intuitively, a belief state consists of all the states in which a nondeterministic full enacted system could be after executing a sequence of actions. Syntactically, a belief state consists of a set of enacted system states and a target state. To ensure that we maintain consistent environment behavior, the environment state is shared with the target state and the behavior (enacted system) states.

Observe that the transition relation of the belief-level system is the main ingredient in its construction. A belief level state q transitions to another state q' by executing action a if two key constraints hold. First, we require that all underlying enacted system states and the target state in q should be able to execute a. That is, there should be a set of behavior indexes, one per enacted system state in q, such that in each enacted system state the indexed behavior can execute a (condition 1 of δ_K). Second, the resulting belief state q' should be such that all possible (nondeterministic) successor states, as a result of execution a from enacted states in q, should be included in q', along with target evolution (with same environment evolution).

Example 4.3. Figure 4.2 shows the belief-level full enacted system for the target specification $\mathcal{T}$ and system $\mathcal{S} = \langle \mathcal{B}_1, \mathcal{B}_2, \mathcal{E} \rangle$. Note that the initial state u_0 encodes the initial state of the enacted system and the target behavior. Consider the evolution of the belief level system for deterministic and nondeterministic behavior actions. If a behavior is deterministic, such as $\mathcal{B}_2$, the number of enacted system states in the successor belief state remain the same. For example, from the initial state, if behavior $\mathcal{B}_2$ executes the action c, then the resulting belief state u_1 has only one enacted system state, which is $\langle b_0, c_2, e_0 \rangle$. On the other hand, if an action is nondeterministic, the number of enacted system states in the successor belief state may increase. For instance, if action a is executed from the initial state by $\mathcal{B}_1$, then the successor belief state u_2 encodes two enacted system states, which are $\langle b_1, c_0, e_0 \rangle$ and $\langle b_2, c_0, e_0 \rangle$. $\square$

Note that if all the behaviors are deterministic, then the belief-level full enacted system will be same as the full enacted system. In fact, and as expected, the belief system is needed only when the system includes *nondeterministic* available behaviors. Surprisingly, the belief level system $\mathcal{K}_{\langle \mathcal{S}, \mathcal{T} \rangle}$ is itself nondeterministic with respect to different behavior delegations. This is by design: conceptually different behavior delegations capture two aspects, different compositions and nondeterminism for transition selection in the resulting SRTF. First, observe that a realizable target specification may have more than one exact composition, hence the same request may be delegated to different behaviors. Second, the SRTF itself, generally, will be nondeterministic (to allow embedding of more information) and such a nondeterminism will be present in the belief-level system.

Example 4.4. The belief-level system (Figure 4.2) of target $\mathcal{T}$ and system $\mathcal{S} = \langle \mathcal{B}_1, \mathcal{B}_2, \mathcal{E} \rangle$ is nondeterministic for action c from state u_2. From the set of enacted system states $\{\langle b_1, c_0, e_0 \rangle, \langle b_2, c_0, e_0 \rangle\}$ two sets of delegations are possible. Since behavior $\mathcal{B}_2$ is capable of executing c from both the enacted system states, the controller may decide either to allocation c to $\mathcal{B}_2$ or to $\mathcal{B}_1$ for the enacted state $\langle b_2, c_0, e_0 \rangle$. $\square$

Note the difference in observability between the controller and the target. The user of the target does not have observability over the behavior states; hence the target specification should be such that irrespective of how a behavior nondeterministically evolves, all the subsequent target requests should be executable by some available behavior. In contrast, the controller can observe each behavior's current state, and therefore based on behavior evolution it may choose to allocate the same target request to different available behaviors. In a nutshell, the controller is allowed to be conditional on the behavior evolution; however the target has to be conformant. We note some similarities in the use of belief-level behaviors with the work in [De Giacomo et al. 2009] for composition under partial observability of the available behaviors. There the *controller* is required to be conformant; here instead the *target behavior* must be so.

Next, we show that $\mathcal{K}_{\langle \mathcal{S},\mathcal{T}\rangle}$ is indeed the SRTF of target $\mathcal{T}$ in system $\mathcal{S}$, which is the main result of this section.

Theorem 4.8. *Let $\mathcal{S}$ be an available system and $\mathcal{T}$ a target specification behavior. Then, $\mathcal{K}_{\langle \mathcal{S},\mathcal{T}\rangle}$ is the SRTF of $\mathcal{T}$ in $\mathcal{S}$.*

Proof. We first define a few technical notions required for the proof. Given a trace $\tau = s_0 \xrightarrow{a^1} \cdots \xrightarrow{a^n} s_n$, we denote the state s_i by $\tau[i]$, the label a^i by $\tau\langle i\rangle$, and prefix $s_0 \xrightarrow{a^1} \cdots \xrightarrow{a^i} s_i$ by $\tau[0,i]$, for $i \leq n$. Given a set of traces Γ, let $\mathsf{Pos}(\Gamma, i) = \{s \mid s = \tau[i], \tau \in \Gamma\}$ be the function that returns the set of i^{th} states from all traces in Γ.

We prove that $\mathcal{K}_{\langle \mathcal{S},\mathcal{T}\rangle}$ and the SRTB $\mathcal{T}^*$ of $\mathcal{T}$ in $\mathcal{S}$ are E-simulation equivalent.

- Proof for $\mathcal{T}^* \preceq_{\mathcal{E}} \mathcal{K}_{\langle \mathcal{S},\mathcal{T}\rangle}$: First, we show that all RTFs are E-simulated by $\mathcal{K}_{\langle \mathcal{S},\mathcal{T}\rangle}$. Let $\mathcal{T}' = \langle T', G', t'_0, \varrho'\rangle$ be an RTF of $\mathcal{T}$ in $\mathcal{S}$. Assume $\mathcal{T}' \not\preceq_{\mathcal{E}} \mathcal{K}_{\langle \mathcal{S},\mathcal{T}\rangle}$; that is, $\mathcal{T}'$ is not E-simulated by $\mathcal{K}_{\langle \mathcal{S},\mathcal{T}\rangle}$. Let $\tau_{\mathcal{T}'} = t'_0 \xrightarrow{g'^1,a^1} \cdots \xrightarrow{g'^n,a^n} t'_n$ be a trace of $\mathcal{T}'$ such that $\tau_{\mathcal{T}'}$ cannot be E-simulated state-wise by any trace of $\mathcal{K}_{\langle \mathcal{S},\mathcal{T}\rangle}$ and the simulation breaks at state t'_{n-1}. We show that this is impossible, since we can build a legal trace of $\mathcal{K}_{\langle \mathcal{S},\mathcal{T}\rangle}$ which can E-simulate the entire $\tau'_{\mathcal{T}}$.

  As $\mathcal{T}'$ is an RTF of $\mathcal{T}$ in $\mathcal{S}$, it holds that $\mathcal{T}' \preceq_{\mathcal{E}} \mathcal{T}$ ($\mathcal{T}'$ is E-simulated by $\mathcal{T}$) and $\mathcal{E}'_{\mathcal{T}} \preceq_{\text{ND}} \mathcal{E}_{\mathcal{S}}$ ($\mathcal{T}'$ has an exact solution in $\mathcal{S}$). Therefore, there exists a trace of $\mathcal{T}$

  $$\tau_{\mathcal{T}} = t_0 \xrightarrow{g^1,a^1} \cdots \xrightarrow{g^n,a^n} t_n$$

  such that $t'_i \preceq_{\mathcal{E}} t_i$ for all $i \leq n$;

  Let us define $\Gamma_{\mathcal{S}}$ as the maximal set of traces $s_0 \xrightarrow{a^1,k_1} \cdots \xrightarrow{a^n,k_n} s_n$ of enacted system $\mathcal{E}_{\mathcal{S}}$ of $\mathcal{S}$, such that:

  1. $t'_i \preceq_{\text{ND}} s_i, i \leq n$, i.e., the traces which copy the target trace $\tau_{\mathcal{T}'} = t'_0 \xrightarrow{g'^1,a^1} \cdots \xrightarrow{g'^n,a^n} t'_n$;
  2. they do so through transitions labelled by a_i, k_i for $i \leq n$ such that for any two traces $\tau_1, \tau_2 \in \Gamma_{\mathcal{S}}$ it is the case that:
     - if $\tau_1[i] = \tau_2[i]$, then $\tau_1\langle i\rangle = \tau_2\langle i\rangle$; that is, they are induced by the same controller; and

  - $\mathsf{env}(\tau_1[i]) = \mathsf{env}(\tau_2[i])$ for all $i \leq n$; that is, τ_1 and τ_2 contain same environment evolutions.

Since $\mathcal{T}'$ is realizable in $\mathcal{S}$ we know that at least one composition exists. Therefore, $\Gamma_\mathcal{S}$ will not be empty. Notice that, because of condition 2 above, there may be several such maximal sets. We nondeterministically select one.

Now, consider a trace $\tau_\mathcal{K} = q_0 \overset{g^1,a^1}{\longrightarrow} \cdots \overset{g^n,a^n}{\longrightarrow} q_n$ such that $q_i = \langle S_i, e_i, \tau_\mathcal{T}[i] \rangle$ for all $i \leq n$, where:

- $e_i = \mathsf{env}(\tau[i])$ for some $\tau \in \Gamma_\mathcal{S}$; and
- $s \in S_i$ iff $s \cup \{e_i\} \in \mathsf{Pos}(\Gamma_\mathcal{S}, i)$.

The idea behind Pos is to return all states that the enacted system could be in. We show $\tau_\mathcal{K}$ is a *legal trace of* $\mathcal{K}_{\langle \mathcal{S},\mathcal{T} \rangle}$; that is, it consists of legal states and transitions. We start by observing that:

- $\tau_\mathcal{K}[i] = \langle \{s_1, \ldots, s_\ell\}, e, t \rangle$, where $\{s_1 \cup \{e\}, \ldots, s_\ell \cup \{e\}\} = \mathsf{Pos}(\Gamma_\mathcal{S}, i)$ and $t = \tau_\mathcal{T}[i]$, is a legal state of $\mathcal{K}_{\langle \mathcal{S},\mathcal{T} \rangle}$ for all $i \leq n$;
- $\tau_\mathcal{K}[0]$ is the initial state of $\mathcal{K}_{\langle \mathcal{S},\mathcal{T} \rangle}$.

We then proceed by induction on n.

- For $n = 0$, we have that the trace $\tau_\mathcal{K}[0]$ consisting only of the initial state is trivially legal.
- By inductive hypothesis let us assume that $q_0 \overset{g^1,a^1}{\longrightarrow} \cdots \overset{g^i,a^i}{\longrightarrow} q_i$ (for $i < n$) is a legal trace of $\mathcal{K}_{\langle \mathcal{S},\mathcal{T} \rangle}$, and we show that $q_0 \overset{g^1,a^1}{\longrightarrow} \cdots \overset{g^{i+1},a^{i+1}}{\longrightarrow} q_{i+1}$ is also a legal trace of $\mathcal{K}_{\langle \mathcal{S},\mathcal{T} \rangle}$.

  Consider the transition $q_i \overset{g^{i+1},a^{i+1}}{\longrightarrow} q_{i+1}$ of $\tau_\mathcal{K}$. Let $\mathsf{Pos}(\Gamma_\mathcal{S}, i) = \{s_1, \ldots, s_\ell\}$. Since $\tau'_\mathcal{T}$ is realizable, there exists $s_j \overset{a^{i+1},k_j^{i+1}}{\longrightarrow} s'_j$ in $\mathcal{E}_\mathcal{S}$ for $j \leq \ell$ and $t_i \overset{g^{i+1},a^{i+1}}{\longrightarrow} t_{i+1}$ in $\mathcal{T}$. Hence, there exists exactly one set of indices (see definition of $\Gamma_\mathcal{S}$, condition 2), $\mathsf{Indx} = \{\langle s_1 : k_1 \rangle, \ldots, \langle s_\ell : k_\ell \rangle\}$, one per each element in $\mathsf{Pos}(\Gamma_\mathcal{S}, i)$, such that $\langle s \cup \{e\}, \tau_\mathcal{T}[i] \rangle \overset{g^{i+1},a^{i+1},k^{i+1}}{\longrightarrow} \langle s' \cup \{e'\}, \tau_\mathcal{T}[i+1] \rangle$ in $\mathcal{F}_{\langle \mathcal{S},\mathcal{T} \rangle}$ where $s \cup \{e\} \in \mathsf{Pos}(\Gamma_\mathcal{S}, i)$, $s' \cup \{e'\} \in \mathsf{Pos}(\Gamma_\mathcal{S}, i+1)$ and $\langle s : k^{i+1} \rangle \in \mathsf{Indx}$. That is, $q_i \overset{g^{i+1},a^{i+1}}{\longrightarrow} q_{i+1}$ is in $\mathcal{K}_{\langle \mathcal{S},\mathcal{T} \rangle}$.

So, RTF $\mathcal{T}'$ is E-simulated by $\mathcal{K}_{\langle \mathcal{S},\mathcal{T} \rangle}$; that is, $\mathcal{T}' \preceq_\mathcal{E} \mathcal{K}_{\langle \mathcal{S},\mathcal{T} \rangle}$. From Theorem 3.4 we know that the union of two RTFs is an RTF, therefore $\mathcal{T}^*$ is also a RTF. Consequently, $\mathcal{T}^* \preceq_\mathcal{E} \mathcal{K}_{\langle \mathcal{S},\mathcal{T} \rangle}$.

- Proof for $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle} \preceq_{\mathcal{E}} \mathcal{T}^*$: we simply observe that $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$ is an RTF, since by construction, we have $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle} \preceq_{\mathcal{E}} \mathcal{T}$ and $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle} \preceq_{\text{ND}} \mathcal{E}_{\mathcal{S}}$. Hence, by Theorem 3.4, $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$ is included in, and thus E-simulated by, $\mathcal{T}^*$.

□

The construction of the belief-level system $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$ is such that it serves as a witness for both the SRTF and its composition generator. Note that the transition relation in $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$ is such that it requires evidence in form of a set of behavior indexes, one per enacted system state, to ensure an action can be executed from each of these enacted states. These behavior indexes can then be extracted to compute the composition generator for $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$.

Theorem 4.9. *The composition generator for SRTF $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$ of a target specification $\mathcal{T}$ in system $\mathcal{S}$ is given by the function:*

$$\mathsf{CG}(h,q,g,a,q') = \{i \mid \langle s : i\rangle \in f_{\mathcal{I}}(q \xrightarrow{g,a} q'), s \cup \{\mathsf{env}(q)\} = \mathsf{last}(h)\}.$$

Proof. Let $\mathsf{ExactComp}(\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle},\mathcal{S})$ be the set of all exact compositions of $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$ in $\mathcal{S}$.

1. $\mathsf{ExactComp}(\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle},\mathcal{S}) \subseteq \{\mathsf{C} \mid \forall h,q,g,a,q'\ \mathsf{C}(h,q,g,a,q') \in \mathsf{CG}(h,q,g,a,q')\}$. Assume there exists a composition $\mathsf{C} \in \mathsf{ExactComp}(\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle},\mathcal{S})$ such that C cannot be generated from CG. Hence, there exists an enacted system history h and an h compatible transition $q \xrightarrow{g,a} q'$ of $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$ such that $\mathsf{C}(h,q,g,a,q') = \mathsf{ind}$ and $\mathsf{ind} \notin \mathsf{CG}(h,q,g,a,q')$. Let $\mathsf{last}(h) = s_h$ and $q = \langle\{s_1,\ldots,s_\ell\},e,t\rangle$. Since $q \xrightarrow{g,a} q'$ is an h compatible request, it follows that $s_h \in \{s_1,\ldots,s_\ell\}$. Since we know that $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$ has an exact composition in $\mathcal{S}$, therefore there exists $\mathsf{Indx} = \{\langle s_1 : i_1\rangle,\ldots,\langle s_\ell : i_\ell\rangle\}$ such that behavior $\mathcal{B}_{i_j}$ can execute action a from enacted state $s_{i_j} \cup \{e\}$, for all $j \leq \ell$. Since behavior $\mathcal{B}_{\mathsf{ind}}$ can also execute a from enacted state s_h, therefore $\mathsf{Indx'} = \{\langle s_1 : i_1\rangle,\ldots,\langle s'_h : \mathsf{ind}\rangle,\ldots,\langle s_\ell : i_\ell\rangle\}$ where $s'_h = s_h \setminus \{e\}$, is also a valid set of indexes for executing a from q. Consequently, $\langle s'_h : \mathsf{ind}\rangle \in f_{\mathcal{I}}(q \xrightarrow{g,a} q')$, and hence index ind will be in the set $\mathsf{CG}(h,q,g,a,q')$. Hence, our assumption is wrong and composition C is included in CG.

2. $\{\mathsf{C} \mid \forall q,t,g,a,q\ \mathsf{C}(h,q,g,a,q') \in \mathsf{CG}(h,q,g,a,q')\} \subseteq \mathsf{ExactComp}(\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle},\mathcal{S})$. Let C be a controller generated from CG such that C is not an exact composition of $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$ in $\mathcal{S}$. Hence, there exists an enacted system history h and an h compatible transition $q \xrightarrow{g,a} q'$ of $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$ such that $\mathsf{CG}(h,q,g,a,q')$ is undefined. This is a contradiction: *(i)* since $q \xrightarrow{g,a} q'$ is an h compatible transition, we know that $\mathsf{last}(h) \in \mathsf{ksys}(q)$; and *(ii)* $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$ has an exact composition in $\mathcal{S}$, so $f_{\mathcal{I}}(q \xrightarrow{g,a} q') \neq \emptyset$. Hence, C is an exact composition for $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$ in $\mathcal{S}$.

□

We now consider the complexity of computing SRTFs involving nondeterministic systems. The belief-level system $\mathcal{K}_{\langle \mathcal{S},\mathcal{T}\rangle}$ can be built in time $2^{O(|\mathcal{B}|^n)}$, where $|\mathcal{B}|$ is the number of states of the largest behavior in $\mathcal{S}$, and n is the number of available behaviors in $\mathcal{S}$. Observe, however, that $\mathcal{K}_{\langle \mathcal{S},\mathcal{T}\rangle}$ can be computed *on-the-fly* in a step-wise fashion: given the current belief state q, we can generate the next possible states without considering any other state in Q. This provides us with a *double exponential upper bound* on computing SRTFs where the available system has nondeterministic behaviors. We conjecture that the lower bound is EXPSPACE. Showing the lower bound for the general case remains an open question and important future work. Note that in the classical case, the problem of checking for existence of an exact composition is EXPTIME-complete for both deterministic and nondeterministic available behaviors. If the conjecture is proved to be true, it would be the first result showing that nondeterminism affects computation complexity in the context of behavior composition.

4.3 Summary

In this chapter, we showed the SRTF can be finitely represented and computed for both deterministic and nondeterministic available systems, and explored computational aspects of the optimisation framework. In composition problems involving only deterministic available behaviors, one can utilize synthesis and model checking tools to effectively compute the SRTF. For composition problems involving nondeterministic available behaviors we introduced a kind of belief space construction technique. All the approaches presented provide computable structures that facilitate the extraction of both the SRTF and its composition generator.

Our main results were as follows:

- The SRTF can be finitely represented and effectively computed.
- Synthesis via safety games and ATL model checking tools can be employed to compute SRTFs (and their composition generators) for deterministic systems.
- SRTFs for composition problems with nondeterministic behaviors can be computed via a particular belief space construction.
- Checking whether a target specification is the SRTF of a composition problem is EXPTIME-complete for deterministic behaviors.
- The SRTF for problems involving nondeterministic behaviors can be computed in time double exponential in the number of behaviors. We conjecture that the problem of checking whether a target specification is the SRTF for a problem with nondeterministic behaviors is at least EXPSPACE-hard.

CHAPTER 5

Composition with Exogenous events

"Mathematics rightly viewed possesses not only truth but supreme beauty."

–Bertrand Russell

In the previous chapter, we showed how to compute a supremal realizable target fragment (SRTF) along with its composition generator for an unsolvable behavior composition problem. We know that the alternate target specification obtained (SRTF) is the closest to the original target that one can obtain in the given system. In addition, we saw that techniques to compute the SRTFs for deterministic systems was insufficient when available behaviors were nondeterministic. The key purpose of allowing nondeterminism in the system is to model uncertainty in the behaviors and the environment. In this chapter, we study an orthogonal approach to handle uncertainty: *uncontrollable exogenous events.*

Inspired by *discrete event systems* [Cassandras and Lafortune 2006] and work on *reasoning about action* for dynamic systems [Reiter 2001], we show here how to accommodate *exogenous uncontrollable events* into the composition framework in a parsimonious manner. The overarching idea behind this is the fact that some of the uncertainty may actually be observable in real world devices. For instance, one might not know when the fuse of a light bulb will blow; however when it blows, one can observe it. In contrast, nondeterminism is used to model the internal hidden logic of behaviors; hence the nondeterministic evolutions are assumed to be unobservable to the target's user. In this chapter, we equip the optimisation framework of Chapter 3 to allow such uncontrollable, but potentially observable, exogenous events. In doing so, it will become clear how robust and elaboration tolerant are the definition of SRTFs and the technique to compute them.

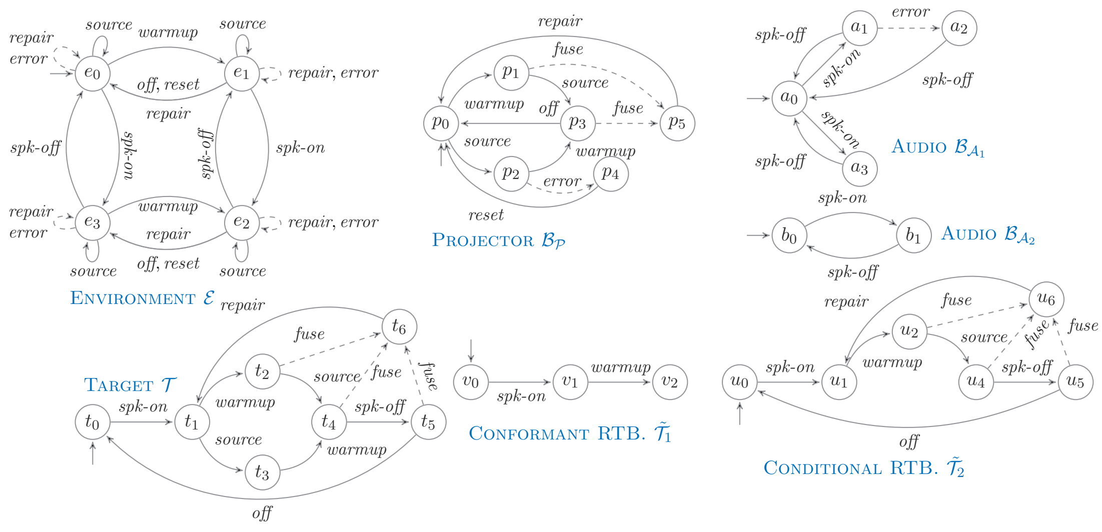

Environment $\mathcal{E}$
Projector $\mathcal{B}_{\mathcal{P}}$
Audio $\mathcal{B}_{\mathcal{A}_1}$
Audio $\mathcal{B}_{\mathcal{A}_2}$
Target $\mathcal{T}$
Conformant RTB. $\tilde{\mathcal{T}}_1$
Conditional RTB. $\tilde{\mathcal{T}}_2$
source
warmup
off, reset
repair
repair, error
spk-on
spk-off
fuse
error
reset
off

Figure 5.1: Media room scenario consisting of a projector, speaker and a target specification (see text for details). Dashed transitions denote uncontrollable exogenous events.

5.1 Framework with exogenous events

We extend the composition framework to model exogenous uncontrollable events. These are events that occur spontaneously in the available behaviors and are outside the behavior's control. To that end, we assume that the set of actions A in the system is partitioned into *controllable* domain actions (A^C) and *uncontrollable* exogenous events (A^U); that is, $A = A^C \cup A^U$ and $A^C \cap A^U = \emptyset$. Furthermore, as is standard in discrete event systems, we assume exogenous events to be deterministic.[1]

Example 5.1. Consider a presentation room with a projector and two audio devices as shown in Figure 5.1. The projector allows setting of the *source* and *warmup* of the device in any order, followed by turning it *off*. The project has two exogenous events, namely, *fuse* and *error*. Suppose that when the projector's light bulb is on—after *warmup* has been executed—it may *fuse* at any time and requires the device to be repaired. Similarly, if an unavailable source is selected before warming up the projector, an *error* may occur and the projector will need to be *reset*. The occurrence of exogenous events *fuse* and *error* may be observed by the user (a red light blinks on the projector when *error* occurs). The speakers of the audio device $\mathcal{B}_{\mathcal{A}_1}$ can go into an error state, after which it needs to be restarted. Audio device $\mathcal{B}_{\mathcal{A}_2}$ on the other hand can simply be toggled on/off. □

The occurrence of such exogenous events, for example *fuse* and *error* in the presentation room example above, is outside the control of the client or the controller, i.e., they occur *spontaneously*. Hence, they are akin to *exogenous events* in reasoning about action

[1]Should this not be the case, we can model the various outcomes with different uncontrollable exogenous events.

literature [Reiter 2001] and *uncontrollable events* in discrete event systems [Cassandras and Lafortune 2006].

We note that exogenous events play an inherently different role in available behaviors than nondeterminism. Exogenous (uncontrollable) events may happen *any time* from a relevant state (e.g., state p_1 in projector $\mathcal{B}_\mathcal{P}$), which allows modeling of concepts such as delayed uncertainty. More importantly, whereas nondeterminism is *not* observable to the target's user, exogenous events may be. In fact, the user agent is not even aware of the internal logic of available behaviors, so she cannot observe the internal evolutions of a device. However, since exogenous events occur in the behaviors, the user can now receive feedback from the behaviors in the form of observable exogenous events. Hence, the user of the projector room may be able to observe the light bulb fusing and respond to that accordingly.

When it comes to the target specification, exogenous event transitions represent those transitions that are accounted for, that is accepted, by the target but *outside the control of the user of the target*. Thus, when the target $\mathcal{T}$ is in state t_2, it only allows one exogenous event, namely, *fuse*, whose occurrence will cause the target to evolve to state t_6 where its user can only request repairing the projector. In that way, target $\mathcal{T}$ can adapt its behavior based on the occurrence of observable exogenous event *fuse*.

The formal definitions of an enacted system and a full enacted system remain the same, except that the action set is partitioned into controllable actions and uncontrollable exogenous events. In addition, given a full enacted system $\mathcal{F}_{\langle\mathcal{S},\mathcal{T}\rangle} = \langle F, A^C \cup A^U, f_0, \gamma\rangle$ (see Definition 4.1 on page 87) for an enacted system $\mathcal{E}_\mathcal{S} = \langle S, A^C \cup A^U, s_0, \delta\rangle$ and a target specification $\mathcal{T} = \langle T, G, t_0, \varrho\rangle$, we define the set $\Phi_{\langle\mathcal{S},\mathcal{T}\rangle}$ as those states in $\mathcal{F}_{\langle\mathcal{S},\mathcal{T}\rangle}$ from where prohibited exogenous events may originate. Formally,

$$\Phi_{\langle\mathcal{S},\mathcal{T}\rangle} = \{\langle s,t\rangle \mid \langle s,\alpha,k,s'\rangle \in \delta, \forall g t' \langle t,g,\alpha,t'\rangle \notin \varrho : g(\mathtt{env}(s)) = \mathtt{true}, \alpha \in A^U\}.$$

The set $\Phi_{\langle\mathcal{S},\mathcal{T}\rangle}$ contains full enacted system states $\langle s,t\rangle$ (s is an enacted system state and t is a target state) such that enacted system state s allows occurrence of an exogenous event that is not accounted for by target state t when the environment is in the state $\mathtt{env}(s)$.

Since the user may be able to observe exogenous events, we can now consider—unlike standard composition—two types of composition solutions. Following planning terminology [Ghallab et al. 2004], a *conditional* SRTF is one that assumes the user is able to observe exogenous events, whereas a *conformant* SRTF is one where such events are unobservable to the user.

5.2 Conditional SRTFs

As the exogenous events can be observed, the target's user can have action requests conditioned on their occurrence. Hence, inclusion of uncontrollable exogenous events in the targets should facilitate branching in the specifications conditioned on the observability of such events. Note that the user of such a conditional target specification will never be allowed to request an uncontrollable event, but she may merely be able to observe such an event and based on its occurrence possibly issue a different request. Introduction of such a capability in the target enables more flexible target specifications. First, the end user can now take advantage of observable contingencies which would have otherwise been hidden behind (unobservable) nondeterminism. And second, one can specify, in the target, uncontrollable events that one would *never* like to occur by excluding them from the specification.

Example 5.2. Consider the conditional target specification $\mathcal{T}$ shown in Figure 5.1. While in state t_2, if the *fuse* of the projector bulb is blown, then the end user can observe that and request a *repair*, as the target state is updated to t_6 after observing *fuse*. Otherwise, if the fuse remains intact, she can continue with the *source* request. Observe, that target specification $\mathcal{T}$ does not allow for any *error* events. Therefore, any composition of $\mathcal{T}$ should be such that it guarantees that *error* never occurs. $\square$

Interestingly, the existing definition of SRTFs from the optimisation framework (see Definition 3.10 on page 51) fits as is to formally capture the notion of conditional SRTFs. However, we need to define exact solution in the context of exogenous events. We do this by extending the ND-simulation [De Giacomo et al. 2013, Sardina et al. 2008] relation. Recall that, ND-simulation extends the plain simulation relation for nondeterministic systems. Briefly, there exists an exact composition for a target specification if the enacted system ND-simulates the enacted target behavior.

Informally in the context of exogenous events, we say an enacted target behavior is *conditionally* simulated by an enacted system iff the enacted system can match all the moves, controllable and uncontrollable, of the enacted target behavior, and only the permitted uncontrollable events as per the given target specification are allowed to occur.

Definition 5.1 (Conditional simulation). Let $\mathcal{E}_\mathcal{T} = \langle S_\mathcal{T}, A^C \cup C^U, s_{\mathcal{T}0}, \varrho_\mathcal{T} \rangle$ be an enacted target behavior of a target specification $\mathcal{T}$ in system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ and $\mathcal{E}_\mathcal{S} = \langle S, A^C \cup A^U, s_0, \delta \rangle$ be the enacted system behavior. Then, $\mathcal{C} \subseteq \mathcal{S}_\mathcal{T} \times \mathcal{S}$ is a *conditional simulation* relation of $\mathcal{E}_\mathcal{T}$ by $\mathcal{E}_\mathcal{S}$ such that $\langle s_\mathcal{T}, s \rangle \in \mathcal{C}$ iff:

1. $\mathsf{env}(s_\mathcal{T}) = \mathsf{env}(s)$;
2. for all transitions $s_\mathcal{T} \xrightarrow{a} s'_\mathcal{T}$ in $\mathcal{E}_\mathcal{T}$, where $a \in A^C \cup A^U$, there exists behavior $\mathcal{B}_i$ such that:
   a) there exists a transition $s \xrightarrow{a,i} s'$ in $\mathcal{E}_\mathcal{S}$ with $\mathsf{env}(s') = \mathsf{env}(s'_\mathcal{T})$; and
   b) for all transitions $s \xrightarrow{a,i} s'$ in $\mathcal{E}_\mathcal{S}$ with $\mathsf{env}(s') = \mathsf{env}(s'_\mathcal{T})$, it is the case that $\langle s'_\mathcal{T}, s' \rangle \in \mathcal{C}$;

3. for all transitions $s \xrightarrow{\alpha,i} s'$ in $\mathcal{E}_\mathcal{S}$, where $\alpha \in A^U$, there exists a transition $s_\mathcal{T} \xrightarrow{\alpha} s'_\mathcal{T}$ in $\mathcal{E}_\mathcal{T}$ such that $\langle s'_\mathcal{T}, s' \rangle \in \mathcal{C}$.

As standard, we say that the enacted system $\mathcal{E}_\mathcal{S}$ <u>*conditionally simulates*</u> the enacted target $\mathcal{E}_\mathcal{T}$, denoted by $\mathcal{E}_\mathcal{T} \preceq_C \mathcal{E}_\mathcal{S}$, iff $\langle s_{\mathcal{T}0}, s_0 \rangle \in \mathcal{C}$. □

Note that the initial two conditions are analogous to ND-simulation (Definition 2.16); that is, we require all actions of the RTF to be *feasible*. The third condition defines how uncontrollable exogenous events should be treated: since they are uncontrollable, their occurrences must be allowed in the target. If we want to prevent the occurrence of some exogenous event this can only be done by forbidding some controllable action ahead of exogenous event's possible occurrence. This is related to the notion of *controllability* in discrete event systems [Wonham and Ramadge 1987].

Example 5.3. Consider the enacted target behavior $\mathcal{E}_\mathcal{T}$ and enacted system $\mathcal{E}_\mathcal{S}$, of target specification $\mathcal{T}$ and system $\mathcal{S}$ in environment $\mathcal{E}$, shown in Figure 5.1, respectively. Clearly, $\mathcal{E}_\mathcal{T}$ is not conditionally simulated by $\mathcal{E}_\mathcal{S}$. After selecting the media *source*, an *error* can occur in state p_2 of projector. However, the target does not permit any uncontrollable event in state t_3. Thus, the third condition of Definition 5.1 is violated, and therefore, $\mathcal{E}_\mathcal{T} \not\preceq_C \mathcal{E}_\mathcal{S}$. □

A conditional RTF can then be defined simply as a target fragment whose enacted behavior is conditionally simulated by the enacted system.

Definition 5.2 (Conditional RTF). Formally, a target specification $\tilde{\mathcal{T}} = \langle \tilde{T}, \tilde{G}, \tilde{t_0}, \tilde{\varrho} \rangle$ is a <u>*conditional-RTF*</u> for a target $\mathcal{T} = \langle T, G, t_0, \varrho \rangle$ in a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ iff:

1. $\tilde{\mathcal{T}}$ is effective in $\mathcal{E}$ (see Definition 3.8 for effective target fragments);
2. $\tilde{\mathcal{T}} \preceq_\mathcal{E} \mathcal{T}$; that is, $\tilde{\mathcal{T}}$ is a fragment of $\mathcal{T}$; and
3. $\mathcal{E}_{\tilde{\mathcal{T}}} \preceq_C \mathcal{E}_\mathcal{S}$; that is, the enacted behavior of $\tilde{\mathcal{T}}$ is conditionally simulated by the enacted system $\mathcal{E}_\mathcal{S}$.

□

Example 5.4. Consider the conditional RTF $\tilde{\mathcal{T}}_2$ of target specification $\mathcal{T}$ in system $\mathcal{S}$ as shown in Figure 5.1. The action *repair* is conditional on the uncontrollable event *fuse* and it prohibits *error* from occurring in the available system. Check that $\tilde{\mathcal{T}}_2$ can be successfully realized in system $\mathcal{S}$. Since $\tilde{\mathcal{T}}_2$ does not require the setting of media *source* before *warmup*, an *error* event can always be prevented. □

As usual, a conditional RTF is *supremal* iff it is not strictly E-simulated by any other conditional RTF. Observe that $\tilde{\mathcal{T}}_2$, in our presentation room example, is actually the conditional SRTF of target specification $\mathcal{T}$ in system $\mathcal{S}$.

Computing conditional SRTFs

When it comes to computing conditional-SRTFs, we modify the belief level construction (see Section 4.2) to allow for exogenous events. Notice that exogenous events are considered to be observable in this case, so we can use their occurrence to refine the belief states in the belief-level full enacted system. This leads to the following definition:

Definition 5.3. Given a belief-level full enacted system $\mathcal{K}_{\langle \mathcal{S},\mathcal{T}\rangle} = \langle Q, G, q_0, \delta_K\rangle$ for full enacted system $\mathcal{F}_{\langle \mathcal{S},\mathcal{T}\rangle} = \langle F, A^C \cup A^U, f_0, \gamma\rangle$, the *conditional belief-level full enacted system* is a tuple $\mathcal{K}^{\mathcal{C}}_{\langle \mathcal{S},\mathcal{T}\rangle} = \langle Q^{\mathcal{C}}, G, q_0, \delta^{\mathcal{C}}_K\rangle$, where:

- $Q^{\mathcal{C}} = Q \setminus \{\langle S, e, t\rangle \mid \langle (s, e), t\rangle \in \Phi_{\langle \mathcal{S},\mathcal{T}\rangle}, s \in S\}$; that is, prohibited exogenous events should never occur;
- $\delta^{\mathcal{C}}_K \subseteq Q^{\mathcal{C}} \times G \times A \times Q^{\mathcal{C}}$ is $\mathcal{K}^{\mathcal{C}}_{\langle \mathcal{S},\mathcal{T}\rangle}$'s transition relation such that $\langle q, g, a, q'\rangle \in \delta^{\mathcal{C}}_K$, where $q = \langle S, e, t\rangle$ and $q' = \langle S', e', t'\rangle$, iff :
  - $a \in A^C$ and $\langle q, g, a, q'\rangle \in \delta_K$; that is, action a should be executable from all enacted states; and
  - $a \in A^U$ and $S' = \{s' \mid \langle (s, e), t\rangle, g, a, k, \langle (s', e'), t'\rangle \in \gamma, s \in S\}$; that is, we revise belief state if an exogenous event occurs.

□

Observe that the conditional belief-level system is similar to the belief-level full enacted system (Definition 4.2) except for two modifications. First, we exclude belief states containing enacted system system states from where target prohibited exogenous events may originate. Second, we suitably adapt the transition relation to account for the occurrence of exogenous events. Since these events are observable, when such an event happens we can refine the belief state by only considering the enacted system states which could have allowed it.

Example 5.5. Consider the belief state $\langle\langle (p_0, a_1, b_0), e_3, t_1\rangle, \langle (p_0, a_3, b_0), e_3, t_1\rangle\rangle$ and assume for the purpose of this example that *error* event is allowed by $\mathcal{T}$ when it is in state t_1. If exogenous event *error* occurs in the speaker $\mathcal{B}_{\mathcal{A}_1}$, then we know that, after the *spk-on* action $\mathcal{B}_{\mathcal{A}_1}$, had evolved to state a_1 (and not a_2). Hence, after observing the event *error* we can revise our belief state to only contain the enacted system state $\langle p_0, a_2, b_0, e_3\rangle$. □

As one would expect, the conditional belief level full enacted system represents the conditional SRTF.

Theorem 5.1. *Let $\mathcal{S}$ be an available system and $\mathcal{T}$ a target specification. Then, $\mathcal{K}^{\mathcal{C}}_{\langle \mathcal{S},\mathcal{T}\rangle}$ is the conditional-SRTB of $\mathcal{T}$ in $\mathcal{S}$.*

Proof. We prove that $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ and the conditional SRTB $\mathcal{T}^*$ of $\mathcal{T}$ in $\mathcal{S}$ are E-simulation equivalent.

Proof for $\mathcal{T}^* \preceq_{\mathcal{E}} \mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$: We first show that all conditional RTFs are E-simulated by $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$. Let $\mathcal{T}' = \langle T', G', t'_0, \varrho'\rangle$ be a conditional RTF of $\mathcal{T}$ in $\mathcal{S}$. Assume $\mathcal{T}' \not\preceq_{\mathcal{E}} \mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$; that is, $\mathcal{T}'$ is not E-simulated by $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$. Let $\tau_{\mathcal{T}'} = t'_0 \xrightarrow{g'^1,a^1} \cdots \xrightarrow{g'^n,a^n} t'_n$ be a trace of $\mathcal{T}'$ such that $\tau_{\mathcal{T}'}$ cannot be E-simulated state by state by any trace of $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$, and the simulation breaks at a state t'_{n-1}. We show that this is impossible, since we can build a legal trace of $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ which can E-simulate the entire $\tau'_{\mathcal{T}}$.

As $\mathcal{T}'$ is a conditional RTF of $\mathcal{T}$ in $\mathcal{S}$, it holds that $\mathcal{T}' \preceq_{\mathcal{E}} \mathcal{T}$ ($\mathcal{T}'$ is E-simulated by $\mathcal{T}$) and $\mathcal{E}'_{\mathcal{T}} \preceq_{\mathcal{C}} \mathcal{E}_{\mathcal{S}}$ ($\mathcal{T}'$ has an exact solution in $\mathcal{S}$). Therefore, there exists a trace of $\mathcal{T}$

$$\tau_{\mathcal{T}} = t_0 \xrightarrow{g^1,a^1} \cdots \xrightarrow{g^n,a^n} t_n$$

such that $t'_i \preceq_{\mathcal{E}} t_i$ for all $i \leq n$;

Let us define $\Gamma_{\mathcal{S}}$ as the maximal set of traces $s_0 \xrightarrow{a^1,k_1} \cdots \xrightarrow{a^\ell,k_\ell} s_\ell$, where $\ell \leq n$, of enacted system $\mathcal{E}_{\mathcal{S}}$ of $\mathcal{S}$, such that:

1. $t'_i \preceq_{\mathcal{C}} s_i, i \leq \ell$, i.e., traces in $\Gamma_{\mathcal{S}}$ copy the target trace $\tau_{\mathcal{T}'} = t'_0 \xrightarrow{g'^1,a^1} \cdots \xrightarrow{g'^\ell,a^\ell} t'_\ell$;
2. they do so through transitions labelled by a_i, k_i for $i \leq n$ such that for any two traces $\tau_1, \tau_2 \in \Gamma_{\mathcal{S}}$ it is the case that:
    - if $\tau_1[i] = \tau_2[i]$, then $\tau_1\langle i\rangle = \tau_2\langle i\rangle$ for controllable actions in τ_1 and τ_2; that is, they are induced by the same controller. Since exogenous events are uncontrollable, we cannot put any restrictions on them; and
    - $\mathsf{env}(\tau_1[i]) = \mathsf{env}(\tau_2[i])$ for all $i \leq n$; that is, τ_1 and τ_2 contain same environment evolutions.

Note, since exogenous events are uncontrollable, $\Gamma_{\mathcal{S}}$ may include system traces where the exogenous event may not occur as per τ'. That is, for every exogenous event at location i of τ', there will be a system trace of length exactly i. Since, $\mathcal{T}'$ is realizable in $\mathcal{S}$ we know that at least one composition exists. Therefore, $\Gamma_{\mathcal{S}}$ will not be empty. Notice that, because of condition 2 above, there may be several such maximal sets. We nondeterministically take one.

Now, consider a trace $\tau_{\mathcal{K}} = q_0 \xrightarrow{g^1,a^1} \cdots \xrightarrow{g^n,a^n} q_n$ such that $q_i = \langle S_i, e_i, \tau_{\mathcal{T}}[i]\rangle$ for all $i \leq n$, where:

- $e_i = \mathsf{env}(\tau[i])$ for some $\tau \in \Gamma_{\mathcal{S}}$; and
- $s \in S_i$ iff $s \cup \{e_i\} \in \mathsf{Pos}(\Gamma_{\mathcal{S}}, i)$ (see proof of Theorem 4.8 for definition of Pos).

The idea behind Pos is to return all states which the enacted system could be in. We show $\tau_{\mathcal{K}}$ is a *legal trace of* $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$; that is, it consists of legal states and transitions.

We start by observing that:

- $\tau_{\mathcal{K}}[i] = \langle\{s_1,\ldots,s_\ell\}, e, t\rangle$, where $\{s_1 \cup \{e\},\ldots,s_\ell \cup \{e\}\} = \mathsf{Pos}(\Gamma_{\mathcal{S}}, i)$ and $t = \tau_{\mathcal{T}}[i]$, is a legal state of $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ for all $i \leq n$;
- $\tau_{\mathcal{K}}[0]$ is the initial state of $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$.

We then proceed by induction on the length of trace $\tau_{\mathcal{K}}$.

- For $n = 0$, we have that the trace $\tau_{\mathcal{K}}[0]$ consisting only of the initial state is trivially legal.
- By inductive hypothesis let us assume that $q_0 \xrightarrow{g^1,a^1} \cdots \xrightarrow{g^i,a^i} q_i$ (for $i < n$) is a legal trace of $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$, and we show that $q_0 \xrightarrow{g^1,a^1} \cdots \xrightarrow{g^{i+1},a^{i+1}} q_{i+1}$ is also a legal trace of $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$.

  Let $\mathsf{Pos}(\Gamma_{\mathcal{S}}, i) = \{s_1,\ldots,s_\ell\}$. Consider the transition $q_i \xrightarrow{g^{i+1},a^{i+1}} q_{i+1}$ of $\tau_{\mathcal{K}}$. There are two possibilities: either a is a controllable action or an uncontrollable event.

  1. $a \in A^C$: Since $\tau'_{\mathcal{T}}$ is realizable, there exists $s_j \xrightarrow{a^{i+1},k_j^{i+1}} s'_j$ in $\mathcal{E}_{\mathcal{S}}$ for $j \leq \ell$ and $t_i \xrightarrow{g^{i+1},a^{i+1}} t_{i+1}$ in $\mathcal{T}$. Hence, there exists exactly one set of indices (see definition of Γ_S, condition 2), $\mathsf{Indx} = \{\langle s_1 : k_1\rangle,\ldots,\langle s_\ell : k_\ell\rangle\}$, one per each element in $\mathsf{Pos}(\Gamma_{\mathcal{S}}, i)$, such that $\langle s \cup \{e\}, \tau_{\mathcal{T}}[i]\rangle \xrightarrow{g^{i+1},a^{i+1},k^{i+1}} \langle s' \cup \{e'\}, \tau_{\mathcal{T}}[i+1]\rangle$ in $\mathcal{F}_{\langle\mathcal{S},\mathcal{T}\rangle}$ where $s \cup \{e\} \in \mathsf{Pos}(\Gamma_{\mathcal{S}}, i)$, $s' \cup \{e'\} \in \mathsf{Pos}(\Gamma_{\mathcal{S}}, i+1)$ and $\langle s : k^{i+1}\rangle \in \mathsf{Indx}$. That is, $q_i \xrightarrow{g^{i+1},a^{i+1}} q_{i+1}$ in $\mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$.
  2. $a \in A^U$: Since $\tau'_{\mathcal{T}}$ is realizable, uncontrollable event a must have occurred in at least one behavior state $s_m \in \{s_1,\ldots,s_\ell\}$. Let $S_a \subseteq \{s_1,\ldots,s_\ell\}$ be the behavior states where a could have potentially occurred. Hence, by definition of $\Gamma_{\mathcal{S}}$, $\mathsf{Pos}(\Gamma_{\mathcal{S}}, i+1)$ will contain $(i+1)^{\text{th}}$ states from only those traces that can be extended by uncontrollable event a. Therefore, $q_i \xrightarrow{g^{i+1},a^{i+1}} q_{i+1}$ in $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$.

So, conditional RTF $\mathcal{T}'$ is E-simulated by $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$; that is, $\mathcal{T}' \preceq_{\mathcal{E}} \mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$. From Theorem 3.4 we know that union of two RTFs is an RTF, therefore $\mathcal{T}^*$ is also a RTF. Consequently, $\mathcal{T}^* \preceq_{\mathcal{E}} \mathcal{K}_{\langle\mathcal{S},\mathcal{T}\rangle}$.

Proof for $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle} \preceq_{\mathcal{E}} \mathcal{T}^*$: we simply observe that $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ is a conditional RTF, since by construction, we have $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle} \preceq_{\mathcal{E}} \mathcal{T}$ and $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle} \preceq_{\mathcal{C}} \mathcal{E}_{\mathcal{S}}$. Hence, by Theorem 3.4, $\mathcal{K}^{\mathcal{C}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ is included in, and thus E-simulated by, $\mathcal{T}^*$. □

In terms of computational complexity, conditional SRTFs can be computed in time $2^{O(|\mathcal{B}|^n)}$ where $|\mathcal{B}|$ is the number of states of the largest behavior in $\mathcal{S}$, and n is the number of available behaviors in $\mathcal{S}$. This gives us a double exponential upper bound for the general case. However, note that the inherent complexity is due to nondeterminism in the available behaviors and not due to the uncontrollable exogenous events. If we consider exogenous events in a deterministic system setting; that is, where all available behaviors are deterministic, then the conditional SRTF can be computed in time exponential to the number of available behaviors. When restricted to deterministic available behaviors, one just needs to compute the full enacted system and exclude the states from where target prohibited exogenous events could occur. Since the user has observability on the exogenous events and the available behaviors are deterministic, the available system will always be in a single state with respect to the target. In other words, the belief-level system and the full enacted system will coincide.

5.3 Conformant SRTFs

In this section, we present the case where the user of the target *cannot* observe exogenous uncontrollable events. Though unobservable, the user is still allowed to define what uncontrollable event are permitted to occur and which are prohibited. Inspired by planning under incomplete information [Smith and Weld 1998], we call such target fragments *conformant*.

Example 5.6. The conformant RTF $\tilde{\mathcal{T}}_1$ of target $\mathcal{T}$ in system $\mathcal{S}$, as shown in Figure 5.1, contains a very restricted part of the target $\mathcal{T}$. Since target $\mathcal{T}$ prohibits *error*, conformant RTF $\tilde{\mathcal{T}}_1$, similar to conditional RTF $\tilde{\mathcal{T}}_2$, does not include the action sequence *spk-on·source*. Observe that after the *spk-on·warmup* action sequence, the enacted system reaches a state from where the uncontrollable event *fuse* can happen. Though *fuse* is permitted as per the specification $\mathcal{T}$, the only action that can be performed after *fuse* is the *repair* action. Since the user cannot observe *fuse*, neither *source* nor *repair* are guaranteed. Hence, unlike conditional RTF $\tilde{\mathcal{T}}_2$, conformant RTF $\tilde{\mathcal{T}}_1$ does not include the action *source* after *warmup*. □

Conformant RTFs are stricter than conditional since they promise execution of actions irrespective of which (permitted) uncontrollable exogenous events occur. This provides *robustness* in modelling as one can still prevent unacceptable conditions under non-observability at runtime. Technically, we say an RTF is conformant if it does not include any transitions with exogenous event. Note the target specification (problem input) is allowed to have exogenous events; however, a conformant RTF must have compiled them away.

In order to define conformant RTFs, we need to extend the ND-simulation to capture realizability under non-observability of uncontrollable exogenous events. Informally, we say that an enacted target behavior is under a *conformant simulation* with an enacted system if the simulation property is maintained invariant of exogenous events.

Definition 5.4 (Conformant simulation). Let $\mathcal{E}_{\mathcal{T}} = \langle S_{\mathcal{T}}, A^C, s_{\mathcal{T}_0}, \varrho_{\mathcal{T}} \rangle$ be an enacted target behavior of a target specification $\mathcal{T} = \langle T, G, t_0, \varrho \rangle$ in system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$

and $\mathcal{E}_{\mathcal{S}} = \langle S, A^C \cup A^U, s_0, \delta \rangle$ be the enacted system behavior. Then, $\mathcal{Z} \subseteq \mathcal{S}_{\mathcal{T}} \times \mathcal{S}$ is a <u>*conformant simulation*</u> relation of $\mathcal{E}_{\mathcal{T}}$ by $\mathcal{E}_{\mathcal{S}}$ such that $\langle s_{\mathcal{T}}, s \rangle \in \mathcal{Z}$ iff:

1. $\mathsf{env}(s_{\mathcal{T}}) = \mathsf{env}(s)$;
2. for all transitions $s_{\mathcal{T}} \xrightarrow{a} s'_{\mathcal{T}}$ in $\mathcal{E}_{\mathcal{T}}$, where $a \in A^C$, there exists behavior $\mathcal{B}_i$ such that:
   a) there exists a transition $s \xrightarrow{a,i} s'$ in $\mathcal{E}_{\mathcal{S}}$ with $\mathsf{env}(s') = \mathsf{env}(s'_{\mathcal{T}})$; and
   b) for all transitions $s \xrightarrow{a,i} s'$ in $\mathcal{E}_{\mathcal{S}}$ with $\mathsf{env}(s') = \mathsf{env}(s'_{\mathcal{T}})$, it is the case that $\langle s'_{\mathcal{T}}, s' \rangle \in \mathcal{Z}$
3. for all transitions $s \xrightarrow{\alpha,i} s'$ in $\mathcal{E}_{\mathcal{S}}$, where $\alpha \in A^U$, it is the case that $\langle s_{\mathcal{T}}, s' \rangle \in \mathcal{Z}$.

As standard, we say that there exists a <u>*conformant simulation*</u> of enacted target $\mathcal{E}_{\mathcal{T}}$ by enacted system $\mathcal{E}_{\mathcal{S}}$, denoted by $\mathcal{E}_{\mathcal{T}} \preceq_{\mathcal{Z}} \mathcal{E}_{\mathcal{S}}$, iff $\langle s_{\mathcal{T}0}, s_0 \rangle \in \mathcal{Z}$. □

Since a conformant RTF does not include any exogenous events, its enacted behavior is restricted only to controllable domain actions. The conformant simulation relation enforces two properties. First, it requires that ND-simulation (first two conditions) should be maintained for all controllable actions. Second, if any exogenous event occurs, it should retain the simulation property (third condition); that is, the simulation property should be an "invariant" of the exogenous events. What we mean by *invariant* is this: assume a target in state t and an enacted system state s are in conformant simulation, and a exogenous event α occurs such that the enacted system reaches a state s'. Then it should be the case that t and s' are also in conformant simulation; that is, we assume the target to stay still in state t when enacted system evolves to s'. Observe that the conformant simulation definition captures realizability in the absence of observability over *all* uncontrollable events (both prohibited and permitted). To define a conformant RTF $\mathcal{T}_1$ for a target $\mathcal{T}$ in system $\mathcal{S}$, one then needs to ensure that no $\mathcal{T}$ prohibited uncontrollable event occurs while realizing $\mathcal{T}_1$. Of course, such a constraint cannot be embedded in the RTF since conformant RTFs are free of any uncontrollable event. More precisely:

Definition 5.5 (Conformant RTF). A conformant target specification $\tilde{\mathcal{T}} = \langle \tilde{T}, \tilde{G}, \tilde{t_0}, \tilde{\varrho} \rangle$ is a <u>*conformant-RTF*</u> for a target $\mathcal{T} = \langle T, G, t_0, \varrho \rangle$ in a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ iff:

1. $\tilde{\mathcal{T}}$ is effective in $\mathcal{E}$;
2. $\tilde{\mathcal{T}} \preceq_{\mathcal{E}} \mathcal{T}$; that is, $\tilde{\mathcal{T}}$ is a fragment of $\mathcal{T}$;
3. $\mathcal{E}_{\tilde{\mathcal{T}}} \preceq_{\mathcal{Z}} \mathcal{E}_{\mathcal{S}}$; that is, there exists a conformant simulation relation between enacted target $\mathcal{E}_{\tilde{\mathcal{T}}}$ and enacted system $\mathcal{E}_{\mathcal{S}}$; and

4. for all tuples $\langle s_{\tilde{\mathcal{T}}}, s\rangle \in \mathcal{Z}$ and $\langle s_{\tilde{\mathcal{T}}}, s_{\mathcal{T}}\rangle \in \preceq$, where $\mathcal{Z}$ is a conformant simulation relation between $\mathcal{E}_{\tilde{\mathcal{T}}}$ and $\mathcal{E}_{\mathcal{S}}$ and $\preceq$ is the largest simulation relation of $\mathcal{E}_{\tilde{\mathcal{T}}}$ by $\mathcal{E}_{\mathcal{T}}$, it holds that: for all transitions $s \xrightarrow{\alpha,i} s'$ there exists a transition $s_{\mathcal{T}} \xrightarrow{\alpha} s'_{\mathcal{T}}$ in $\mathcal{T}$ such that $\langle s_{\tilde{\mathcal{T}}}, s_{\mathcal{T}}\rangle \in \preceq$, where $\alpha \in A^U$.

□

The first three conditions are in the standard scheme of defining a RTF. The last condition enforces only permitted exogenous events to ever occur in the system. A reachable enacted state should be such that all possible uncontrollable events from that state should be accounted by the enacted states corresponding original target state. As usual, a conformant RTF is *supremal* iff it is not strictly simulated by any other conformant RTF.

Computing conformant SRTFs

When it comes to computing conformant SRTFs one needs conformance over nondeterministic evolutions as well as uncontrollable exogenous events. However, the belief level full enacted system from Section 4.2 only ensures conformance over nondeterminism. In order to include exogenous events in it, inspired by λ-closure in automata theory [Hopcroft et al. 2007], we first define what we call the ε–closure of a state. Informally, the ε–closure of a state is a set of all states where the system could be as a result of an exogenous event from that state. Formally:

Definition 5.6. Given a full enacted system $\mathcal{F}_{\langle \mathcal{S},\mathcal{T}\rangle} = \langle F, A^C \cup A^U, f_0, \gamma\rangle$ and a state $f \in \mathcal{F}$, the *ε–closure* of f, denoted by $\varepsilon(f)$, is defined recursively as follows:

1. $f \in \varepsilon(f)$; that is, the state itself is in the closure;
2. for all transitions $f_1 \xrightarrow{g,\alpha,k} f_2$ in $\mathcal{F}_{\langle \mathcal{S},\mathcal{T}\rangle}$, where $\alpha \in A^U$ and $f_1 \in \varepsilon(f)$, it is the case that $f_2 \in \varepsilon(f)$; that is, all exogenous event reachable states are included; and
3. Nothing except for 1 and 2 should be in $\varepsilon(f)$.

□

We next re-define the belief-level full enacted system to accommodate exogenous events. Here, we consider the ε–closure in both the initial state and the transition relation.

Definition 5.7. Given a full enacted system $\mathcal{F}_{\langle \mathcal{S},\mathcal{T}\rangle} = \langle F, A^C \cup A^U, f_0, \gamma\rangle$ for a target $\mathcal{T} = \langle T, G, t_0, \varrho\rangle$ and a system $\mathcal{S} = \langle \mathcal{B}_1, \dots, \mathcal{B}_n, \mathcal{E}\rangle$ with environment $\mathcal{E} = \langle E, A^C \cup C^U, e_0, \rho\rangle$, the *conformant belief-level full enacted system* is a tuple $\mathcal{K}^{\mathcal{Z}}_{\langle \mathcal{S},\mathcal{T}\rangle} = \langle Q, G, q_0, \delta_K\rangle$, where:

- $Q = 2^{(B_1 \times \cdots \times B_n \times T \times E)} \setminus \{S \mid s \in S, s \in \Phi_{\langle \mathcal{S},\mathcal{T}\rangle}\}$;

- $q_0 = \varepsilon(f_0)$ is $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$'s initial state;
- $\delta_K \subseteq Q \times G \times A^C \times Q$, where $\langle S, g, a, S'\rangle \in \delta_K$ iff :
  - there exists a set $\mathsf{Indx} = \{\langle s_1 : k_1\rangle, \ldots, \langle s_\ell : k_\ell\rangle\}$ such that $\{s_1, \ldots, s_\ell\} = S$; $s_i \xrightarrow{g,a,k_i} s'_i$ in $\mathcal{F}_{\langle\mathcal{S},\mathcal{T}\rangle}$ for all $i \leq \ell$; and for all $i, j \leq \ell$ if $\mathsf{tgt}(s_i) = \mathsf{tgt}(s_j)$, then $\mathsf{tgt}(s'_i) = \mathsf{tgt}(s'_j)$; and
  - $S' = \bigcup_{\langle s:i\rangle \in \mathsf{Indx}} \{\varepsilon(s') | \langle s, g, a, i, s'\rangle \in \gamma\}$; that is, S' should contain the ε–closure of all successors of enacted system states in S resulting from action $a \in A^U$.

□

Since the original target specification is free to contain exogenous events, the ε–closure of an full enacted system state may include more than one target states. Hence, the belief level full enacted system is now exponential in size also on target state. For example, if we take target specification $\mathcal{T}$ shown in Figure 5.1, the ε–closure of any full enacted state with t_2 will include states with t_2 and t_6. Observe, if the target specification expresses exogenous events in the form of self-loops; that is, all exogenous event transitions are of the form $t \xrightarrow{g,\alpha} t$ where t is a target state and g is a guard, then the complexity with regard to the target will no longer be exponential. Note, the above construction is more expressive as it not only enables excluding prohibited exogenous events but also allows counting exogenous events, e.g., *error* should not occur more than 5 times. As one would expect, the above construction of $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ provides us with the conformant SRTF.

Theorem 5.2. *Let $\mathcal{S}$ be an available system and $\mathcal{T}$ a target specification. Then, $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ is a conformant SRTF of $\mathcal{T}$ in $\mathcal{S}$.*

Proof. We first define a number of additional technical notions required for the proof. The function $\omega(s \xrightarrow{a} s', A)$ takes a transition $s \xrightarrow{a} s'$ as input and returns the action a if $a \in A$, else it returns ϵ (empty). Let the function $\mathsf{act\text{-}seq}(\tau, A)$ return the action sequence of τ consisting only of actions included in A. Formally,

$$\mathsf{act\text{-}seq}(s_0 \xrightarrow{a^1} \cdots \xrightarrow{a^n} s_n, A) = \omega(s_0 \xrightarrow{a^1} s_1, A) \cdots \omega(s_{n-1} \xrightarrow{a^n} s_n, A).$$

Given a state $\tau[i]$ of trace τ let $\varepsilon(\tau, i)$ be the set of $\tau[i]$'s successor states in τ reachable from $\tau[i]$ by zero or more exogenous events in A^U. Formally,

$$\varepsilon(\tau, i) = \{s \mid \tau[i] \xrightarrow{\alpha_{i+1}} \cdots \xrightarrow{\alpha_{i+\ell}} s, \alpha_{i+j} \in A^U, 0 \leq j \leq \ell\}.$$

We now proceed with the proof and show that $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ and the conformant SRTF $\mathcal{T}^*$ of $\mathcal{T}$ in $\mathcal{S}$ are E-simulation equivalent.

Proof for $\mathcal{T}^* \preceq_{\mathcal{E}} \mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$: First, we show that all conformant RTFs are simulated by $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$. Let $\mathcal{T}' = \langle T', G', t'_0, \varrho'\rangle$ be a conformant RTF of $\mathcal{T}$ in $\mathcal{S}$. Assume $\mathcal{T}' \not\preceq_{\mathcal{E}} \mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$;

that is, $\mathcal{T}'$ is not E-simulated by $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$. Let $\tau_{\mathcal{T}'} = t'_0 \xrightarrow{g'^1,a^1} \cdots \xrightarrow{g'^n,a^n} t'_n$ be a trace of $\mathcal{T}'$ such that $\tau_{\mathcal{T}'}$ cannot be E-simulated state by state by any trace of $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ and the simulation breaks at a state t'_{n-1}. We show that this is impossible since, we can build a legal trace of $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ which can simulate the entire $\tau'_{\mathcal{T}}$. As $\mathcal{T}'$ is a conformant RTF of $\mathcal{T}$ in $\mathcal{S}$, it holds that $\mathcal{T}' \preceq_{\mathcal{E}} \mathcal{T}$ ($\mathcal{T}'$ is E-simulated by $\mathcal{T}$) and $\mathcal{E}'_{\mathcal{T}} \preceq_{\mathcal{Z}} \mathcal{E}_{\mathcal{S}}$ ($\mathcal{T}'$ is realizable in $\mathcal{S}$). Note, since $\mathcal{T}'$ is a conformant RTF, because of exogenous events, $\tau_{\mathcal{T}'}$ may be E-simulated by more than one trace of $\mathcal{T}$. Therefore, there exists a set of traces of $\mathcal{T}$ such that $\tau = t_0 \xrightarrow{g^1,a^1} \cdots \xrightarrow{g^\ell,a^\ell} t_\ell \in \Gamma_{\mathcal{T}}$, where $\ell \geq n$ iff:

1. $\mathsf{act\text{-}seq}(\tau'_{\mathcal{T}}, A^C) = \mathsf{act\text{-}seq}(\tau_{\mathcal{T}}, A^C)$, the sequence of controllable actions is same; and
2. if $t'_i \preceq_{\mathcal{E}} t_j$, where $i \leq j, i \leq n, j \leq \ell$, then either $t'_i \preceq_{\mathcal{E}} t_{j+1}$ or $t'_{i+1} \preceq_{\mathcal{E}} t_{j+1}$; the simulation relation is maintained across exogenous events in the target spec.

Let us define $\Gamma_{\mathcal{S}}$ as the maximal set of traces $\tau_{\mathcal{S}} = s_0 \xrightarrow{a^1,k_1} \cdots \xrightarrow{a^m,k_m} s_m$, where $m \geq n$, of enacted system $\mathcal{E}_{\mathcal{S}}$ of $\mathcal{S}$, such that:

1. if $t'_i \preceq_{\mathcal{Z}} s_j$, where $i \leq j, i \leq n, j \leq m$, then either $t'_i \preceq_{\mathcal{Z}} s_{j+1}$ or $t'_{i+1} \preceq_{\mathcal{Z}} s_{j+1}$;
2. $\mathsf{act\text{-}seq}(\tau'_{\mathcal{T}}, A^C) = \mathsf{act\text{-}seq}(\tau_{\mathcal{S}}, A^C)$, the enacted system traces can copy the RTF trace $\tau'_{\mathcal{T}}$;
3. they do so through transitions labelled by a_i, k_i for $i \leq n$ such that for any two traces $\tau_1, \tau_2 \in \Gamma_{\mathcal{S}}$ it is the case that if $\tau_1[i] = \tau_2[i]$, then $\tau_1\langle i\rangle = \tau_2\langle i\rangle$.

Note, since only allowed exogenous events can occur, the induced system traces will correspond to the target traces in $\Gamma_{\mathcal{T}}$. Since, $\mathcal{T}'$ is realizable in $\mathcal{S}$ we know that at least one composition exists. Therefore, $\Gamma_{\mathcal{S}}$ will not be empty. Notice that, because of condition 3 above, there may be several such maximal sets. We nondeterministically take one.

We observe that due to exogenous events the enacted system traces may be longer than the trace $\tau_{\mathcal{T}'}$. Given an action sequence $\vec{a} = a_1 \ldots a_n$ and a trace τ_1, let $\tau_1^{\vec{a}}$ denote the shortest prefix of τ_1 such that $\mathsf{act\text{-}seq}(\tau_1^{\vec{a}}, A^C) = \vec{a}$.

Now, consider a trace $\tau_{\mathcal{K}} = q_0 \xrightarrow{g^1,a^1} \cdots \xrightarrow{g^n,a^n} q_n$ such that $q_i = \langle \mathsf{Pos}^{\mathcal{Z}}(\Gamma_{\mathcal{S}}, \Gamma_{\mathcal{T}}, i)\rangle$ for all $i \leq n$ where:

$$\mathsf{Pos}^{Z}(\Gamma_{\mathcal{S}}, \Gamma_{\mathcal{T}}, i) = \textstyle\bigcup_{\tau_1 \in \Gamma_{\mathcal{F}}} \{\varepsilon(\tau_1[j]) \mid j = |\tau_1^{\vec{a}}|, \vec{a} = \mathsf{act\text{-}seq}(\tau_{\mathcal{T}'}[0,i], A^C)\}$$

and,

$$\Gamma_{\mathcal{F}} = \{\langle s,t\rangle \xrightarrow{g^1,a^1,k_1} \cdots \xrightarrow{g^m,a^m,k_m} \langle s',t'\rangle \,|\, s \xrightarrow{a^1,k_1} \cdots \xrightarrow{a^m,k_m} s' \in \Gamma_{\mathcal{S}}, t \xrightarrow{g^1,a^1} \cdots \xrightarrow{g^m,a^m} t' \in \Gamma_{\mathcal{T}}\}.$$

Note, since the system evolutions must match the original target specification, $\Gamma_{\mathcal{F}}$ is well defined. The idea behind Pos^{Z} is to return all states where the enacted system could be due to nondeterminism or exogenous events, after realizing a sequence of controllable actions.

We show $\tau_{\mathcal{K}}$ is a *legal trace of* $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$; that is, it consists of legal states and transitions. We start by observing that:

- $\tau_{\mathcal{K}}[i] = \langle\{s_1, \ldots, s_\ell\}\rangle$, where $\{s_1, \ldots, s_\ell\} = \mathsf{Pos}^{\mathcal{Z}}(\Gamma_{\mathcal{S}}, \Gamma_{\mathcal{T}}, i)$, is a legal state of $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ for all $i \leq n$;
- $\tau_{\mathcal{K}}[0]$ is the initial state of $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$.

Then we proceed by induction of $\tau_{\mathcal{K}}$.

- For $n = 0$, we have that the trace $\tau_{\mathcal{K}}[0]$ consisting only of the initial state is trivially legal.
- By inductive hypothesis let us assume that $q_0 \xrightarrow{g^1, a^1} \cdots \xrightarrow{g^i, a^i} q_i$ (for $i < n$) is a legal trace of $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$. We show that $q_0 \xrightarrow{g^1, a^1} \cdots \xrightarrow{g^{i+1}, a^{i+1}} q_{i+1}$ is also a legal trace of $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$. Consider the transition $q_i \xrightarrow{g^{i+1}, a^{i+1}} q_{i+1}$ of $\tau_{\mathcal{K}}$. Let $\mathsf{Pos}^{\mathcal{Z}}(\Gamma_{\mathcal{S}}, \Gamma_{\mathcal{T}}, i) = \{s_1, \ldots, s_\ell\}$. Since $\tau_{\mathcal{T}'}$ is realizable, there exists $\mathsf{esys}(s_j) \xrightarrow{a^{p+1}, k_j^{p+1}} \mathsf{esys}(s'_j)$ in $\mathcal{E}_{\mathcal{S}}$ for $j \leq \ell, p \geq i$ and $t_i \xrightarrow{g^{p+1}, a^{p+1}} t_{p+1}$ in $\mathcal{T}$. Hence, there exists exactly one set of indices (see definition of $\Gamma_{\mathcal{S}}$, condition 3), $\mathsf{Indx} = \{\langle s_1 : k_1\rangle, \ldots, \langle s_\ell : k_\ell\rangle\}$, one per each element in $\mathsf{Pos}^{\mathcal{Z}}(\Gamma_{\mathcal{S}}, \Gamma_{\mathcal{T}}, i)$, such that $s \xrightarrow{g^{p+1}, a^{p+1}, k^{p+1}} s'$ in $\mathcal{F}_{\langle\mathcal{S},\mathcal{T}\rangle}$ where $s \in \mathsf{Pos}^{\mathcal{Z}}(\Gamma_{\mathcal{S}}, \Gamma_{\mathcal{T}}, i)$, $s' \in \mathsf{Pos}^{\mathcal{Z}}(\Gamma_{\mathcal{S}}, \Gamma_{\mathcal{T}}, i+1)$ and $\langle s : k^{p+1}\rangle \in \mathsf{Indx}$. Note, we consider $\varepsilon{-}\mathsf{closure}$ when evolving to successor belief state, in align with the definition of $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$. That is, $q_i \xrightarrow{g^{i+1}, a^{i+1}} q_{i+1}$ in $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$.

Note that by construction of $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$, the last condition of the conformant simulation definition is automatically satisfied.

So, RTF $\mathcal{T}'$ is E-simulated by $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$; that is, $\mathcal{T}' \preceq_{\mathcal{E}} \mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$. From Theorem 3.4 we know that union of two RTFs is an RTF; therefore $\mathcal{T}^*$ is also a RTF. Consequently, $\mathcal{T}^* \preceq_{\mathcal{E}} \mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$.

Proof for $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle} \preceq_{\mathcal{E}} \mathcal{T}^*$: We simply observe that since $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ is a conformant RTF by construction, we have $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle} \preceq_{\mathcal{E}} \mathcal{T}$ and $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle} \preceq_{\mathcal{Z}} \mathcal{E}_{\mathcal{S}}$. Hence $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ by Theorem 3.4 $\mathcal{K}^{\mathcal{Z}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ is included in, and thus E-simulated by, $\mathcal{T}^*$. $\square$

In terms of computational complexity, conformant SRTFs can be computed in time $2^{O(|\mathcal{B}|^n \times |\mathcal{T}|)}$, where $|\mathcal{B}|$ is the number of states of the largest behavior in $\mathcal{S}$, n is the number of available behaviors in $\mathcal{S}$, and $|\mathcal{T}|$ is the size of the target specification. Note, in the conformant case, the inherent complexity is due to not only the nondeterminism in the available behaviors but also the uncontrollable exogenous events. Indeed, even if

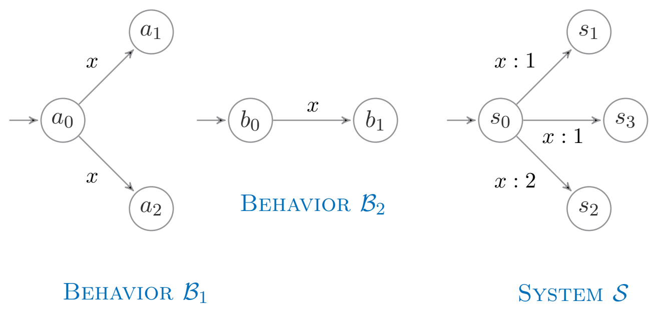


Figure 5.2: Relating DES and behavior composition.

we consider exogenous events in a deterministic system setting, the belief set construction technique for conformant SRTFs will retain its double exponential upper bound.

5.4 Discrete event systems

A related, and seemingly similar, area to behavior composition is the engineering field of Discrete Event Systems (DES) [Wonham and Ramadge 1987, Cassandras and Lafortune 2006]. Briefly, in a DES problem, given a *plant* (system) and a *specification*, the task is to synthesize a supervisor that can control the plant such that the working of the plant is as per the specification. The plant itself consists of controllable and uncontrollable actions, and the supervisor is allowed to disable only the controllable actions. One then *constructs the most flexible supervisor which only disables the minimal number of controllable actions such that the restricted behavior of the plant matches the given specification.* If the specification cannot be met, then one can synthesize a supervisor which guarantees a *supremal* sub-specification (maximal fragment of the specification). Interestingly, both DES and behavior composition are concerned with *controllability* of a given plant or an available system in order to meet a given language or a target specification.

From the outset, it may seem that behavior composition and DES are tackling the same problem, maybe from different perspectives: DES from an Engineering perspective and composition from a Computer Science perspective. Nonetheless, the inherent control problem in SCT and behavior composition are different in nature. In the latter, one seeks to control the available behaviors, whereas in the former one can prevent (some of) the actions. Consider the simple example shown in Figure 5.2 with a nondeterministic behavior $\mathcal{B}_1$ and a deterministic behavior $\mathcal{B}_2$. Note that both behaviors share the action x; hence, in the enacted system $\mathcal{S}$, x will be nondeterministic for $\mathcal{B}_1$ but not for $\mathcal{B}_2$ (as shown by the indexes used in $\mathcal{S}$). The input in DES is the whole plant, and it does not have a notion analogous to available behaviors. Therefore, a component-based nondeterminism cannot be captured (directly) in a plant, and one has to make events (i.e., indexes) explicit in the plant [Balbiani et al. 2008].

Another important mismatch relates to the semantics of nondeterminism: the nondeterminism of controllable actions in a plant is *angelic*, in the sense that the supervisor can control its evolution. On the other hand, nondeterminism of available behaviors is *devilish*, as it cannot be controlled. This is one of the reasons why, as far as we know, DES frameworks do not have a notion similar to ND-simulation [Sardina et al. 2008]. Indeed, uncertainty in DES is modelled via (deterministic) uncontrollable events [Wonham and Ramadge 1987], whereas nondeterminism [De Giacomo et al. 2013] is used in behavior composition.

In terms of terminology, the word "composition" differs considerably in DES: it refers to synchronous product between automaton of sub-systems, instead of an asynchronous construction. In addition, DES does not differentiate between the plant itself and the environment in which it operates.

Lastly, in the context of this work, a closely related line of work from DES is the computation of controllable sub-automata [Sun et al. 2010; 2012] for a given plant and specification. Briefly, an automaton $\mathcal{T}_1 = \langle T_1, A_1, t_0, \varrho_1 \rangle$ is a sub-automata of $\mathcal{T} = \langle T, A, t_0, \varrho \rangle$ if $T_1 \subseteq T$, $A_1 \subseteq A$, and ϱ_1 is same as ϱ but restricted to the states in T_1 [Sun et al. 2010]. A sub-automaton is then considered a partial specification; that is, an RTF in our context. Observe that besides being a very syntactic notion, the definition of sub-automaton implies simulation between the automaton and its sub-automaton, but the reverse is not true. In other words, if $\mathcal{T}$ simulates $\mathcal{T}_1$, then $\mathcal{T}_1$ may not be $\mathcal{T}$'s sub-automaton. Our definition of target fragments subsumes the syntactic sub-automatons definition. Observe that new (nondeterministic) branching cannot be introduced in sub-automatons, hence they cannot embed more information regarding future requests as one can do in target fragments.

5.5 Summary

Incorporating uncontrollable exogenous events provides us with a *richer* language to capture target specifications. Indeed, it provides the flexibility for defining uncertainties which are out of control of the user but may still be observable and spontaneous. Arguably, it provides additional semantics to the composition problem which is closer to real life settings by only minimally modifying the composition framework from Chapter 3. Based on the observability on such events, inspired by automated planning, we provided two kinds of solutions - *conditional* SRTFs and *conformant* SRTFs. If the user of the target is able to observe exogenous events, she can then condition her requests based on their occurrence. On the other hand, if the user has no observability on such events, then she can only define which exogenous events are allowed to occur and which are prohibited, but her action requests ought to be conformant with respect to them.

To conclude:

- Uncontrollable exogenous events can be used to model uncertainties that are *delayed* and *potentially observable*.
- Based on the user's ability to observe such events, two kinds of solutions are defined: *conditional* SRTFs and *conformant* SRTFs.
- Conditional and conformant SRTFs can be computed by suitably modifying the belief-level full enacted system.
- The current technique to compute conditional and conformant SRTFs, for the general case, is in complexity class 2-EXPTIME.

CHAPTER 6

Decision theoretic composition

"Not to be absolutely certain is, I think, one of the essential things in rationality."

–Bertrand Russell

So far we have seen how to deal with unsolvable behavior composition instances in a qualitative manner without requiring any further user input. In this chapter, we propose an extension to the classical composition problem that goes beyond strict uncertainty, by accommodating ways of quantifying different uncertainties in the model. In the classical composition setting, each available behavior is modelled as a nondeterministic transition system to represent partial controllability; the target behavior is modelled as a deterministic transition system to represent full controllability; and the environment, which is fully accessible by all behaviors, is modelled as a nondeterministic transition system to represent partial predictability. As one can recognize, in the classical framework, there are three potential sources of uncertainty: the potential nondeterminism in both the shared environment and the available behaviors; and the next target request arising from possible different transitions from a target's state. In the probabilistic behavior composition framework to be developed here, all three uncertainties are quantified. Note this is a reasonable assumption in many realistic settings, in which such information is readily available to the modeller.

Consider a domain in which different bots are meant to maintain a garden by performing various activities such as *cleaning*, *watering*, and *picking* flowers. Some of these bots may be equipped with buckets that may nondeterministically get full after using them. This nondeterminism can be quantified depending on various aspects of the domain (e.g., size of the bucket, average amount of dirt collected in a single *clean* action, etc.). Similarly, execution of actions in the shared garden environment can also be represented stochastically. For instance, a single *clean* operation may not always successfully clean the whole garden; factors affecting the probability of a successful clean include the size of the garden and the current season. More interestingly, given that the desired target specification for maintaining the garden may involve more than one action from a given state, probabilities can be assigned to these depending on their expected frequency. For instance, in some states, the gardening target system is expected to request the *picking* action only 30% of the time; the rest of the time it will just request *watering* the garden.

Here, we present a decision theoretic framework for behavior composition. In doing so, we define the notion of *optimal composition controllers* using the "expected realizability" of the target, as well as the notion of *maximal* compositions; that is, controllers that will solve the composition problem robustly. Then, we provide a translation of a decision theoretic behavior composition problem into a Markov decision process (MDP) [Puterman 2005, French 1986], and show that finding an optimal policy for such MDP amounts to finding an optimal composition. This problem reduction provides a readily available technique for solving the new composition framework using the established MDP paradigm.

6.1 Markov decision processes

Markov decision processes provide a formal framework for decision making in stochastic (probabilistic) domains. They are widely used in Artificial Intelligence as well as in areas beyond computer science such as Operations Research. A Markov decision process (MDP) is a *discrete time stochastic control process* [Puterman 2005, Boutilier et al. 1999]. At each step, a process is in a state q, the decision maker chooses an action a, the process evolves to a successor state q' with some probability, and the decision maker receives a reward r. The more preferred decision maker is one that collects maximum (potential) rewards over time.

Definition 6.1 (Markov decision process (MDP) [Puterman 2005])**.** Formally, a *Markov decision process* (MDP) is a tuple $\mathcal{M} = \langle Q, A, p, r\rangle$, where:

- Q is a finite set of states;
- A is a finite set of domain actions;
- $p : Q \times A \times Q \mapsto [0,1]$ is the transition probability function function: $p(q,a,q')$ denotes probability of the process evolving to state q' when action a is executed in state q. We required that for all states $q \in Q$ and actions $a \in A$ it is the case that $\sum_{q'\in Q} p(q,a,q') = 1$;
- $r : Q \times A \mapsto \mathbb{R}$ is the reward function: $r(q,a)$ denotes the immediate reward obtained when action a is executed in state q.

$\square$

The intuition is that the decision maker partially controls the evolution of the MDP by selecting an action for execution to obtain maximum reward. Usually, in the MDP terminology, decision epochs are associated with an MDP. Decision epochs are discrete time steps at which decisions are taken. For instance, the first action is chosen at time t_0, next at time t_1, and so on. A *policy* is a collection of state-action mappings stating which action the decision maker should take at each time step of the process. A policy is called *stationary* if the decision only depends on the state and not on the time step; that is, if the process reaches the same state at two different time steps, then the policy prescribes the same action. Additionally, a *Markovian* (memoryless) policy is one that depends only on the current state of the system; that is, it is independent of the history. Formally, a Markovian stationary policy is a function $\pi : Q \mapsto A$; where $\pi(q)$ denotes the action to be taken in state q. Solving an MDP then involves computing a policy that accumulates maximum reward over time. In doing so, one can be interested in *finite horizon* problems, where the decision maker is meant to perform a fixed number of sequential decisions, or *infinite horizon* problems, where rewards over infinite runs of the MDP are considered.

The *expected value* of a given policy π from a state q for a horizon t of an MDP $\mathcal{M} = \langle Q, A, p, r\rangle$ can be inductively calculated by the following value function [Boutilier et al. 1999]:

$$V_t^\pi(q) = r(q, \pi(q)) + \sum_{q' \in Q} p(q, \pi(q), q') \times V_{t-1}^\pi(q'),$$

where $V_0^\pi(q) = 0$. In words, the expected value of a policy is the sum of expected rewards obtained by acting as per the policy. A policy π is *optimal* for an MDP $\mathcal{M}$ and a horizon t if for all other policies π' of $\mathcal{M}$ it is the case that $V_t^\pi(q) \geq V_t^{\pi'}(q)$ for all states q of $\mathcal{M}$. That is, an optimal policy has the highest expected value from all MDP states. Optimal policies can be computed via Dynamic Programming approaches [Boutilier et al. 1999] following Bellman's *principle of optimality* [Bellman 1957], which states that an optimal policy has optimal sub-policies. The value of an optimal policy in a state q for a *finite* horizon k is given by:

$$V_k^*(q) = \max_{a \in A}\{r(q, a) + \sum_{q' \in Q} p(q, a, q') \times V_{k-1}^*(q')\}.$$

The above equation forms the basis of the well know *value iteration algorithm* [Puterman 2005] for finite horizon problems. Briefly, the value iteration begins by computing V_0^*, and uses the above equation to compute V_t^* (till the given horizon t) by successively computing the action at each step which will maximize the value function.

Obviously, if the horizon is infinite, the cumulative rewards obtained will also be infinite. In such cases, a discounting factor is often used to provide convergence. Similar to the finite horizon case, the value of an optimal policy in a state q for *infinite* horizon relative to a discount factor of $0 \leq \gamma < 1$ is as follows [Howard 1960]:

$$V^*(q) = \max_{a \in A}\{r(q,a) + \gamma \sum_{q' \in Q} p(q,a,q') \times V^*(q')\}.$$

In order to ensure termination for computing the optimal policy for infinite horizon problems, one defines a stopping criteria ϵ and stops when $V^*_t - V^*_{t-1} \leq \epsilon$. Howard [1960] showed that there always exists an optimal *stationary* policy for infinite horizon problems; that is, one that does not depend on which stage a decision is taken. Since, in the general case, a target specification is meant to be executed as a continuous (infinite) process, we are interested in stationary policies over an infinite horizon.

Note, we do not differentiate cost and reward functions as classically described. In our setting we do not model the cost of executing the action; hence only reward functions are considered.

6.2 Probabilistic framework for behavior composition

In a classical composition problem, incomplete information on any component is modeled by means of nondeterminism in the transition systems (in the available behaviors or in the environment) or by different transitions per state (in the target). As such, all the work so far on the problem of behavior composition has assumed a setting of *strict uncertainty* [French 1986]. To elaborate, the space of possibilities; that is the possible effects of actions, evolution of behaviors, and future action requests, is known, but the probabilities of these potential alternatives are not quantified.

In this section, we extend the classical composition framework (see Section 2.3.1) to accommodate stochastic measures in the different components, thus yielding a framework for behavior composition under *quantified uncertainty*. In particular, we use probabilities to represent uncertainty of the dynamics of the environment and of the available behaviors, as well as of the preferences on actions in the target module. Such probabilities are provided by a domain expert who is able to state how often a device is expected to fail, an action brings about its expected effects, or certain requests arrive at the system. In addition, we assume such probabilities to be stationary; that is, they do not change with time.

Environment As is standard in behavior composition, we assume to have a *fully observable shared environment*, which provides an abstract account of actions' preconditions and effects, and a mean of communication among modules. Since, in general, we have incomplete information about the actual preconditions and effects of actions, we shall use a stochastic model of the environment. Thus, given a state and an action to be executed in such state, different successor states may ensue with different probabilities.

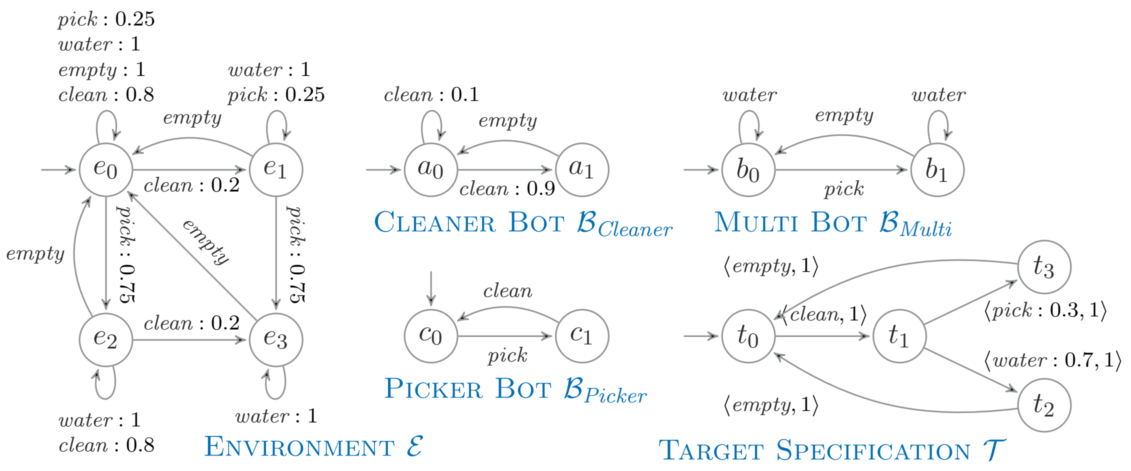


Figure 6.1: The garden bots system $\mathcal{S}_{Garden} = \langle \mathcal{B}_{Cleaner}, \mathcal{B}_{Multi}, \mathcal{B}_{Picker}, \mathcal{E} \rangle$ and the target specification $\mathcal{T}$. The transition $t_1 \xrightarrow{\langle water:0.7,1 \rangle} t_2$ in target $\mathcal{T}$ means that action *water* has a reward of 1 and it is requested 70% of the time from state t_1.

Definition 6.2 (**Stochastic environment**)**.** A *stochastic environment* is a tuple $\mathcal{E} = \langle E, A, e_0, \mathcal{P}^{\mathcal{E}} \rangle$, where:

- E is the finite set of environment's states;
- A is a finite set of shared actions;
- $e_0 \in E$ is the initial state of the environment; and
- $\mathcal{P}^{\mathcal{E}} : E \times \mathcal{A} \times E \mapsto [0, 1]$ is the probabilistic transition function among states: $\mathcal{P}^{\mathcal{E}}(e, a, e') = p$, or just $e \xrightarrow{a:p} e'$ in $\mathcal{E}$, states that action a when performed in state e leads the environment to a successor state e' with probability p.

□

Furthermore, to ensure soundness, we require that for every $e \in E$ and $a \in A$, it is the case that $\sum_{e' \in E} \mathcal{P}^{\mathcal{E}}(e, a, e') \in \{0, 1\}$. That is, either an action is not executable (the sum is 0) or all possible evolutions of the environment are accounted for (the sum is 1).

Example 6.1. A scenario wherein a garden is maintained by several bots is depicted in Figure 6.1. To keep the garden healthy one needs to regularly *water* the plants, *pick* the ripe fruits and flowers, *clean* the garden by collecting fallen leaves and removing dirt, and *emptying* the various waste bins. Whereas *cleaning* and *emptying* the bins is a regular activity, *picking* and *watering* are done as required. The environment $\mathcal{E}$ models the states the garden can be in. The environment allows *picking* and *cleaning* activities to be done in any order, and plants can be watered in any state. The *pick* action results in the flowers and fruits been fully picked 75% of the time (i.e., 25% of the time the garden still remains to be picked), whereas the *clean* action results in the garden being totally cleaned 20% of the time (i.e., dirt still remains 80% of the time). A *pick* action from the initial state (e_0) results in the garden being picked but dirty (e_2) with a probability of 0.75; a subsequent *clean* action results in the garden being both picked and clean (e_3), with a probability of 0.2. Similarly, a *clean* action from the initial state results in the garden being fully clean but not picked (e_1) 20% of the time, and a subsequent *pick* action causes the garden being cleaned and picked (e_3) 75% of the time. For simplicity, we assume that emptying the bins always results in the environment evolving to its initial state. □

Available behaviors Recall that a behavior essentially stands for the logic of some available component (e.g., device, agent, plan, workflow), which provides its user with a set of actions that can be performed step by step. The source of uncertainty in the available behaviors is the nondeterminism that is used to represent their hidden logic. We quantify this nondeterminism in the stochastic framework.

Definition 6.3 (Stochastic behavior). A *stochastic behavior* over an environment $\mathcal{E} = \langle E, A, e_0, \mathcal{P}^{\mathcal{E}} \rangle$ is a tuple $\mathcal{B} = \langle B, b_0, \mathcal{P}^{\mathcal{B}} \rangle$, where:

- B is the finite set of behavior's states;
- $b_0 \in B$ is the initial state of the behavior;
- $\mathcal{P}^{\mathcal{B}} : B \times E \times A \times B \mapsto [0, 1]$ is the probabilistic transition function of the behavior: $\mathcal{P}^{\mathcal{B}}(b, a, e, b') = p$, or $e \xrightarrow{a,e:p} e'$ in $\mathcal{B}$, denotes that action a executed in behavior state b when the environment is in state e will result in the behavior evolving to state b' with probability p.

□

Since *all* potential transitions are accounted for in the model, we require that for every $b \in B$, $a \in A$, and $e \in E$, $\sum_{b' \in B} \mathcal{P}^{\mathcal{B}}(b, a, e, b') \in \{0, 1\}$. Observe that we omit the notion of guards in the definition of stochastic behaviors. In fact, guards are implicitly compiled into the probabilistic transition function of available behaviors. This is evident by the inclusion of the environment states in the definition of $\mathcal{P}^{\mathcal{B}}$. For example, if an action a cannot be executed from a behavior state b when the environment is in state e, then $\mathcal{P}^{\mathcal{B}}(b, a, e, b')$ will be 0 for all behavior states b'.

Example 6.2. In the gardening scenario, we assume there are three available garden bots; see Figure 6.1. The cleaner bot $\mathcal{B}_{Cleaner}$ cleans the garden by collecting the fallen leaves, dirt, waste, etc., into its own bucket. Most generally—90% of the time—its bucket becomes full with a single cleaning session, and the bot has to empty it before cleaning again. We assume the *empty* action involves emptying all garden bins as well as the bots' buckets. The picker bot $\mathcal{B}_{Picker}$ can pick and clean the garden; since it is not equipped with a bucket, it picks and collects from the ground directly. Finally, the multi-bot $\mathcal{B}_{Multi}$ has the capability to *water* the plants and *pick* fruits. It has a small bucket, and so it needs to *empty* it after every picking action. $\square$

A behavior is deterministic if, given a state and a legal action in that state, *the* next behavior state is unique—the behavior is *fully controllable* through the selection of the next action to perform. Formally, a behavior $\mathcal{B} = \langle B, b_0, \mathcal{P}^{\mathcal{B}} \rangle$ over an environment $\mathcal{E} = \langle E, A, e_0, \mathcal{P}^{\mathcal{E}} \rangle$ is *deterministic* iff for every $b, b' \in B$, $e \in E$, and $a \in A$, it is the case that $\mathcal{P}^{\mathcal{B}}(b, e, a, b') \in \{0, 1\}$. In such a case, the dynamics of the behavior can be represented using a transition relation $\delta_{\mathcal{B}} \subseteq B \times E \times A \times B$, where $\delta_{\mathcal{B}}(b, e, a, b')$ holds iff $\mathcal{P}^{\mathcal{B}}(b, e, a, b') = 1$.

Target specification Similarly to the classical composition framework, a target behavior is basically a *deterministic* behavior over $\mathcal{E}$ that represents the fully controllable desired behavior.

Definition 6.4 (Stochastic target specification). A <u>*stochastic target specification*</u> over an environment $\mathcal{E} = \langle E, A, e_0, \mathcal{P}^{\mathcal{E}} \rangle$ is a tuple $\mathcal{T} = \langle T, t_0, \delta, R, \mathcal{P}_{\mathsf{req}} \rangle$, where:

- T is the finite set of the target's states;
- $t_0 \in T$ is the initial state of the target;
- $\delta \subseteq T \times E \times A \times T$ is the target's deterministic transition relation: $\langle t, e, a, t' \rangle \in \delta$, or $t \xrightarrow{e:a} t'$ in $\mathcal{T}$, states that action a executed in the target state t, when the environment is in a state e, results in the target evolving to (unique) state t';
- $R : T \times A \mapsto \mathbb{R}^+$ is the *reward function* of the target: $R(t, a)$ denotes the reward obtained when the action a is successfully executed in target state t;
- $\mathcal{P}_{\mathsf{req}} : T \times E \times A \mapsto [0, 1]$ is the *probabilistic action request function*: $\mathcal{P}_{\mathsf{req}}(t, e, a)$ denotes the probability of the target requesting the execution of action a when it is in state t and the environment is in state e.

$\square$

For consistency, we require that $\sum_{a \in A} \mathcal{P}_{\mathsf{req}}(t, e, a) \in \{0, 1\}$, for every $t \in T$, $e \in E$ (i.e., all possible requests are accounted for), and moreover, for all $a \in A$, we have $\mathcal{P}_{\mathsf{req}}(t, e, a) = 0$ whenever there is no state $t' \in T$ such that $\langle t, e, a, t' \rangle \in \delta$. Observe that a target specification is accompanied by two additional functions: a reward function R and an action request frequency function $\mathcal{P}_{\mathsf{req}}$. The reward function R denotes the importance of an action in a target state; generally speaking, more important actions will have a higher reward. On the other hand, the function $\mathcal{P}_{\mathsf{req}}$ denotes how frequently an action will be requested from a given target and environment state. A *uniform-reward* target behavior is one where all actions have the same reward; that is, there exists $\alpha \in \mathbb{R}^+$ such that for all $a \in A$ and $t \in T$, we have $R(t, a) = \alpha$.

Example 6.3. The desired behavior required to maintain the garden in a particular season is not directly represented by any of the existing bots in the garden, and is modeled by the deterministic uniform-reward target bot $\mathcal{T}$ shown in Figure 6.1. Intuitively, the garden should always be cleaned first to remove any fallen leaves and dirt, followed by either picking or watering the garden. Since flowers and fruits do not grow every day, picking is required only 30% of the time; 70% of the time a request for watering the garden will be issued. Finally, the bins are to be emptied, and the whole process can repeat again. All requests are of equal value: 1 unit (the second component in each transition label). □

This concludes the definition of the core components for a decision-theoretic behavior composition problem. As the reader can see, this framework is similar to the classical composition framework described in Section 2.3.1, except that *stochastic probabilistic transitions* are used instead of transition relations; a probability distribution over the potential action requests is used in the specification of the target; and a reward function is used in the target to state how "important" a particular request is. Note also that the probability function $\mathcal{P}_{\mathsf{req}}$ in the target is very different to the ones used in the available behaviors and the environment. In the former, it denotes the probability of the target executing (i.e., requesting) an action from a given state, whereas in the latter the corresponding function simply denotes the stochastic evolutions of the entity.

Enacted system Similarly to the classic composition and the qualitative optimisation framework, we compute the *synchronous product* of the environment with the *asynchronous product* of all available behaviors to define a *stochastic enacted system*.

Definition 6.5 (Stochastic enacted system). Let $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ be a system, where $\mathcal{E} = \langle E, A, e_0, \mathcal{P}^{\mathcal{E}} \rangle$ and $\mathcal{B}_i = \langle B_i, b_{i0}, \mathcal{P}^{\mathcal{B}_i} \rangle$, for $i \in \{1, \ldots, n\}$. The *stochastic enacted system* behavior of $\mathcal{S}$ is the tuple $\mathcal{E}_{\mathcal{S}} = \langle S, A, \{1, \ldots, n\}, s_0, \mathcal{P}_{\mathcal{S}} \rangle$, where:

- $S = B_1 \times \cdots \times B_n \times E$ is the finite set of $\mathcal{E}_{\mathcal{S}}$'s states; when $s = \langle b_1, \ldots, b_n, e \rangle$, we denote b_i by $\mathsf{beh}_i(s)$, for $i \in \{1, \ldots, n\}$, and e by $\mathsf{env}(s)$;
- $s_0 \in S$, with $\mathsf{env}(s_0) = e_0$ and $\mathsf{beh}_i(s_0) = b_{i0}$, for each $i \in \{1, \ldots, n\}$, is $\mathcal{E}_{\mathcal{S}}$'s initial state;

- $\mathcal{P}_\mathcal{S} : S \times A \times \{1, \ldots, n\} \times S \mapsto [0, 1]$ is $\mathcal{E}_\mathcal{S}$'s probabilistic transition function, defined as follows:

$$\mathcal{P}_\mathcal{S}(s, a, k, s') = \mathcal{P}^\mathcal{E}(\mathsf{env}(s), a, \mathsf{env}(s')) \times \mathcal{P}^{\mathcal{B}_k}(\mathsf{beh}_k(s), a, \mathsf{env}(s), \mathsf{beh}_k(s')),$$

$$\text{if } \mathsf{beh}_i(s) = \mathsf{beh}_i(s'), \text{ for each } i \in \{1, \ldots, n\} \setminus \{k\};$$
$$\text{and } \mathcal{P}_\mathcal{S}(s, a, k, s') = 0, \text{ otherwise.}$$

□

Observe that in the stochastic enacted system all the sources of nondeterminism in the system, namely, nondeterminism in the available behavior and the environment, are quantified. So, informally, the decision-theoretic (DT) behavior composition task is stated as follows:

> *Given a system $\mathcal{S}$ and a target behavior $\mathcal{T}$, find the "optimal" way of (partially) controlling the available behaviors in $\mathcal{S}$ in a step-by-step manner so as to "best realize" a specific deterministic target behavior.*

6.3 Decision theoretic controllers

In this section, we make the problem of finding an optimal controller precise. In order to bring about a desired virtual target behavior in an available system, we assume the existence of a (central) *controller* module that is able to control the available behaviors, in the sense that, at each step, it can observe all behaviors, instruct them to execute an action (within their capabilities), stop, and resume them. In classical behavior composition, one then looks for a controller that *guarantees* that the target will *always* be implemented in the system; that is, no matter how the target happens to request actions within its logic or how the available behaviors and the environment happen to evolve as a result of action delegations. Such a controller is then deemed to be an (exact) solution to the problem.

However, when it comes to realizing a target module in a composition framework as the one described above, one should not look just for *exact* solutions, as in general there may be none. Instead, one should look for *optimal* ways of maximizing the "expected realizability" of the target in the available system.

The definition of a controller remains unchanged (see Section 2.4), namely, a *controller* for an available system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$ is a partial function $C : \mathcal{H} \times A \mapsto \{1, \ldots, n\}$ such that, given a system history $h \in \mathcal{H}$ and an action $a \in A$ that ought to be performed, returns the index of the behavior to which the action a is to be delegated for execution. Informally, a "dead-end" is reached in a history if the controller selects a behavior which is not capable of executing the delegated action. Then, given two controllers, one should prefer the one that reaches a dead-end with lower probability, or put differently, the one that has the highest probability of honoring the target's requests. In particular, a controller that is guaranteed to never reach a dead-end will be an exact, and thus optimal, solution.

We say that a history is reachable by a controller, if starting from the initial state of the enacted system, the behavior executing the action at each state of the history is indeed the one selected by the controller. More formally, a history $h = s^0 \xrightarrow{a^1,k^1} \cdots \xrightarrow{a^\ell,k^\ell} s^\ell$ is _reachable_ by a controller C (in a system $\mathcal{S}$) iff $k^i = \mathsf{C}(s^0 \xrightarrow{a^1,k^1} \cdots \xrightarrow{a^{i-1},k^{i-1}} s^{i-1}, a^i)$, for each $i \in \{1, \ldots, \ell\}$. We denote by $\mathcal{H}^\ell_{\mathsf{C}}$ the set of all reachable histories of length ℓ and $\mathcal{H}_{\mathsf{C}} = \bigcup_{i \geq 0} \mathcal{H}^i_{\mathsf{C}}$ the set of all histories reachable by C.

6.3.1 Value of a controller and compositions

In order to evaluate and compare controllers, we define the value of a controller for a given target and system. Roughly speaking, a controller is "rewarded" for every target request that it fulfills by a successful delegation to an available behavior. More specifically, at every point, a controller gets a reward that depends both on the frequency of such a request and the value of (fulfilling) it.

Let $\mathcal{T} = \langle T, t_0, \delta, R, \mathcal{P}_{\mathsf{req}} \rangle$ be a target specification to be realized in a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$. Let C be a controller for target $\mathcal{T}$ in system $\mathcal{S}$, and $\mathcal{E}_{\mathcal{S}}$ be the enacted system behavior as defined in the previous section. First, consider the case of evaluating the performance of a controller over a finite number of requests. Intuitively, the value of a controller is the total reward it can collect by successfully delegating the target's actions to the available behaviors.

Definition 6.6 (**Value of a controller: k-steps**)**.** The _value of a controller for k steps_ of target $\mathcal{T}$ in a system $\mathcal{S}$ is defined as $\mathcal{V}^{\mathsf{C}}_k = \mathcal{V}^{\mathsf{C}}_k(s_0, t_0)$, where the value of controller C for $k \geq 1$ requests, denoted by $\mathcal{V}^{\mathsf{C}}_k(h, t)$, at system history $h \in \mathcal{H}$ when the target is in state $t \in T$ is given by $\mathcal{V}^{\mathsf{C}}_0(h, t) = 0$, for $k = 0$ and all $h \in \mathcal{H}$, and for $k > 0$ we have:

$$\begin{aligned}\mathcal{V}^{\mathsf{C}}_{k>0}(h,t) = \sum_{a \in A} [&\mathcal{P}_{\mathsf{req}}(t, \mathsf{env}(\mathsf{last}(h)), a) \times IR^{\mathsf{C}}(h,t,a) \ + \\ &\sum_{\substack{s' \in S \\ \langle t,e,a,t' \rangle \in \delta}} \mathcal{P}_{\mathsf{req}}(t, \mathsf{env}(\mathsf{last}(h)), a) \times \mathcal{P}_{\mathcal{S}}(\mathsf{last}(h), a, \mathsf{C}(h,a), s') \times \mathcal{V}^{\mathsf{C}}_{k-1}(h \xrightarrow{a,\mathsf{C}(h,a)} s', t')],\end{aligned}$$

where $IR^{\mathsf{C}}(h,t,a)$ stands for the *immediate* reward collected by the controller C when requested to delegate action a at history h:

$$IR^{\mathsf{C}}(h,t,a) = \begin{cases} R(t,a) & \text{if } \exists s'. \mathcal{P}_{\mathcal{S}}(\mathsf{last}(h), a, \mathsf{C}(h,a), s') > 0; \\ 0 & \text{if } \mathsf{C}(h,a) \text{ is undefined;} \\ -R(t,a) & \text{otherwise.} \end{cases}$$

□

We say that a controller C^* is <u>*k-maximal*</u> if for all other controllers C, $\mathcal{V}_k^{\mathsf{C}^*} \geq \mathcal{V}_k^{\mathsf{C}}$. In order to calculate a controller's value, one then needs to sum the expected immediate reward; that is, the probability of requesting a next action multiplied by the reward gained for its execution, and all the expected future rewards. Observe that we include the probability of a current action request when calculating the expected reward for the next step; the *expected* probability of an enacted system evolving to a state includes the probability of its own transition as well as the probability of the target actually requesting that action.

Since a target may include infinite traces, we are in general interested in controllers that are optimal for any number of potential requests; that is, for infinite executions of the target behavior. To cope with unbounded executions of a target, we appeal to the use of a *discount factor*, as is customary in sequential decision making over infinite episodes [French 1986, Boutilier et al. 1999]. The idea is that the satisfaction of later target-compatible requests is less important than those issued earlier.

Definition 6.7 (Value of a controller: infinite steps). The value of a controller C relative to a discount factor $0 \leq \gamma < 1$ for a system $\mathcal{S}$ and target specification $\mathcal{T}$, is defined as $\mathcal{V}_\gamma^{\mathsf{C}} = \mathcal{V}_\gamma^{\mathsf{C}}(s_0, t_0)$ where:

$$\begin{aligned}\mathcal{V}_\gamma^{\mathsf{C}}(h,t) = \sum_{a \in A} \big[& \mathcal{P}_{\mathsf{req}}(t, \mathsf{env}(\mathsf{last}(h)), a) \times IR^{\mathsf{C}}(h,t,a) + \\ & \gamma \sum_{\substack{s' \in S \\ \langle t,e,a,t' \rangle \in \delta}} \mathcal{P}_{\mathsf{req}}(t, \mathsf{env}(\mathsf{last}(h)), a) \times \mathcal{P}_{\mathcal{S}}(\mathsf{last}(h), a, \mathsf{C}(h,a), s') \times \mathcal{V}_\gamma^{\mathsf{C}}(h \xrightarrow{a, \mathsf{C}(h,a)} s', t')\big].\end{aligned}$$

□

The use of a discount factor plays the same role as for infinite horizon Markov decision processes; namely, it allows convergence of the value of a controller [Boutilier et al. 1999, Puterman 2005]. Note that the assumption that temporally closer rewards are more important than distant ones is particularly suitable in the context of composition problems, where behaviors may fail, the target and available system may be reset, or the problem may not be fully solvable. Finally, we say that a controller C^* is <u>*γ-maximal*</u> (of target $\mathcal{T}$ in system $\mathcal{S}$) if for all other controllers C, it is the case that $\mathcal{V}_\gamma^{\mathsf{C}^*} \geq \mathcal{V}_\gamma^{\mathsf{C}}$.

Put together, the decision theoretic behavior composition problem, or simply *DT-composition problem*, amounts to synthesizing a γ-maximal controller for a given target specification $\mathcal{T}$, given system $\mathcal{S}$, and discount factor γ.

6.3.2 Exact compositions

A behavior composition problem has an exact solution when there exists a controller that can *fully* realize the target; that is, a controller that can *always* honor the target's requests, no matter what happens. There have recently been various approaches in the literature to synthesize such a controller, called a *composition*, if any exists (see Section 2.4). Within our decision theoretic setting, it is important to clearly define what an exact solution is and its relationship with "optimal" controllers.

Since the target behavior is deterministic, its specification can be seen as the set of all possible sequences of actions that can be requested, starting from the initial state. Thus, given any finite run of the target, the most one could expect is that every single action is successfully realized in the system. This would imply that all possible rewards in the run have indeed been collected. Since one does not know a priori which actual run will ensue, we consider the maximum expected reward when running the target. To make this precise, we define the notion of *maximum expected reward.*

Definition 6.8 (**Maximum expected reward**). The *maximum expected reward* gained for $k \geq 0$ steps when executing a target specification from its state t when environment is in a state e, denoted by $\mathcal{R}_k^{\max}(t, e)$, is given by:

$$\mathcal{R}_{k\geq 1}^{\max}(t,e) = \sum_{a\in A}[\mathcal{P}_{\mathsf{req}}(t,e,a) \times R(t,a) + \sum_{\substack{e'\in E \\ \langle t,e,a,t'\rangle\in\delta}} \mathcal{P}_{\mathsf{req}}(t,e,a) \times \mathcal{P}_{\mathcal{E}}(e,a,e') \times \mathcal{R}_{k-1}^{\max}(t',e')],$$

where $\mathcal{R}_0^{max}(t,e) = 0$ for $k = 0$. □

As above, we take $\mathcal{R}_k^{\max} = \mathcal{R}_k^{\max}(t_0, e_0)$, for any $k \geq 0$. Note that this definition is well defined for both cyclic and acyclic targets. Of course, for a cyclic target $\mathcal{R}_k^{\max}$ will increase with the number of steps k. For an acyclic target with a longest path of length ℓ, it is easy to check that $\mathcal{R}_k^{\max} = \mathcal{R}_\ell^{\max}$, for every $k \geq \ell$.

Thus, a controller C is an *exact composition* if $\mathcal{V}_k^{\mathsf{C}} = \mathcal{R}_k^{\max}$, for all $k \geq 1$, that is, C can *fully* and *always* realize a target behavior in the available system. Note that controllers are meant to have full observability of the current history. A *Markovian* (i.e., memoryless) controller C is one that only looks at the current state of the system to decide the delegation; formally, for all histories $h, h' \in \mathcal{H}$ such that $\mathsf{last}(h) = \mathsf{last}(h')$ and action $a \in A$, $\mathsf{C}(h,a) = \mathsf{C}(h',a)$ applies. When it comes to exact solutions, Markovian controllers are enough under full observability.

Theorem 6.1. *Let $\mathcal{S}$ be a system and $\mathcal{T}$ be a target behavior. Then, if there exists an exact solution for realizing $\mathcal{T}$ in $\mathcal{S}$, then there exists a Markovian controller which is also an exact solution.*

Proof. Let C^* be an exact solution for realizing $\mathcal{T}$ in $\mathcal{S}$. For any $h \in \mathcal{H}$ and $a \in A$, we define a new controller $\widehat{\mathsf{C}}(h,a) = \mathsf{C}^*(h',a)$ if $h' \in \mathcal{H}_{\mathsf{C}^*}$ is such that $\mathsf{last}(h) = \mathsf{last}(h')$ and for all $h'' \in \mathcal{H}_{\mathsf{C}^*}$ such that $\mathsf{last}(h'') = \mathsf{last}(h)$, it is the case that $\mathsf{C}^*(h',a) \leq \mathsf{C}^*(h'',a)$. Otherwise, if such a history h' does not exist, we leave $\widehat{\mathsf{C}}(h,a)$ undefined.

Note that $\widehat{\mathsf{C}}$ is not only well-defined but also Markovian. Consider two histories $h_1, h_2 \in \mathcal{H}$ such that $\mathsf{last}(h_1) = \mathsf{last}(h_2)$, and suppose that $\widehat{\mathsf{C}}(h_1,a) = k_1$. Then, $\mathsf{C}^*(h_1',a) =$

k_1, for some $h'_1 \in \mathcal{H}_{\mathsf{C}^*}$ and $\widehat{\mathsf{C}}(h_2, a) = \mathsf{C}^*(h'_1, a) = k_1$ as well; the same witness history h'_1 can be used for h_2 too. Furthermore, because h'_1 is reachable by C^*, together with the fact that C^* is indeed an exact solution, this implies that $k_1 \in \{1, \ldots, n\}$ is a correct delegation, in the sense that behavior $\mathcal{B}_{k_1}$ is able to perform a legal step on action a when the environment is in state $\mathsf{env}(\mathsf{last}(h))$, and since $\mathsf{last}(h) = \mathsf{last}(h'_1)$, such a delegation is also legal at history h_2 and $\widehat{\mathsf{C}}$ is also exact. □

Theorem 6.1 shows that memoryless controllers are sufficient when it comes to exact compositions. More importantly, exact solutions are guaranteed to be always optimal controllers under unbounded runs, independently of the discount factor chosen.

Theorem 6.2. *If a controller is an exact composition for a decision-theoretic behavior composition problem, then such a controller is a γ-composition, for any $0 \leq \gamma < 1$.*

Proof. Let C^* be an exact solution to a DT-composition problem, and assume, wlog, a target with a uniform reward α. Then, at each step, C^* collects the maximum possible reward of α. If a discount factor γ is used, then C^* will collect a reward of $\alpha \times \sum_{n=1}^{\ell} \gamma^{n-1}$ over ℓ steps, which is indeed the maximum possible reward for a γ-composition after ℓ steps. Hence, C^* is also a γ-composition for the given composition problem. □

Theorem 6.2 shows that exact compositions are invariant on the value of the discount factor. To elaborate, if a controller is a γ_1-composition for a composition problem accommodating an exact controller, then that controller will also be a γ_2-composition for any $\gamma_2 > 0$.

6.4 Computing optimal controllers via MDP reduction

Various techniques have been used to actually solve the (qualitative) classical behavior composition problems, including PDL satisfiability [De Giacomo and Sardina 2007], search-based approaches [Stroeder and Pagnucco 2009], ATL synthesis [De Giacomo and Felli 2010], and computation of special kinds of simulation relations [Sardina et al. 2008]. Unfortunately, in the context of the decision theoretic framework of Section 6.2, none of these techniques can be applied. In this section, we show how to solve a decision theoretic composition problem, by reducing it to a Markov decision problem in a natural manner. We also demonstrate the reduction with a proof-of-concept implementation using an existing off-the-shelf MDP solver.

6.4.1 From behavior composition to MDPs

Informally, in our setting, the decision maker is the controller, and thus, the possible actions that can be taken are those of behavior delegation. Consider then a system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E}\rangle$, with $\mathcal{E}_{\mathcal{S}} = \langle S, A, \{1, \ldots, n\}, s_0, \mathcal{P}_{\mathcal{S}}\rangle$ denoting the corresponding stochastic enacted system behavior, and a target specification $\mathcal{T} = \langle T, t_0, \delta, R, \mathcal{P}_{\mathsf{req}}\rangle$. We define the corresponding encoded MDP $\mathcal{M}_{\langle \mathcal{S}, \mathcal{T}\rangle} = \langle Q, \mathsf{ind}, p, r\rangle$ as follows:

- $Q = S \times T \times A \ \cup \ \{q_\sharp\}$, where for all $\langle s, t, a\rangle \in Q$, $\mathcal{P}_{\mathsf{req}}(t, \mathsf{env}(s), a) > 0$. Given an MDP state $q = \langle s, t, a\rangle \in Q$, we define $\mathsf{sys}(q) = s$, $\mathsf{tgt}(q) = t$, and $\mathsf{req}(q) = a$. A special, domain independent, state $q_\sharp$ is used as a "dummy" initial state of the process.

- $\mathsf{ind} = \{1, \ldots, n, u\}$; that is, an action in the encoded MDP stands for a behavior selection, we include a special action u denoting no selection.

- The state transition function is defined as follows:

$$p(q, i, q') = \begin{cases} \mathcal{P}_{\mathsf{req}}(\mathsf{tgt}(q'), \mathsf{env}(\mathsf{sys}(q')), \mathsf{req}(q')), \text{ if} \\ \qquad q = q_\sharp, \mathsf{sys}(q') = s_0, \mathsf{tgt}(q') = t_0; \\ \\ \mathcal{P}_{\mathcal{S}}(\mathsf{sys}(q), \mathsf{req}(q), i, \mathsf{sys}(q')) \times \\ \quad \mathcal{P}_{\mathsf{req}}(\mathsf{tgt}(q'), \mathsf{env}(\mathsf{sys}(q')), \mathsf{req}(q')), \text{ if} \\ \qquad q \neq q_\sharp; \\ \\ 0, \text{otherwise.} \end{cases}$$

- The reward function is defined as:

$$r(q, i) = \begin{cases} R(\mathsf{tgt}(q), \mathsf{req}(q)) & \text{if } \mathcal{P}_{\mathcal{S}}(\mathsf{sys}(q), \mathsf{req}(q), i, \mathsf{sys}(q')) > 0 \\ & \qquad \text{for some } q' \in Q \text{ and } q \neq q_\sharp; \\ \\ 0 & \text{if } i = u \text{ or } q = q_\sharp; \\ \\ -R(\mathsf{tgt}(q), \mathsf{req}(q)) & \text{otherwise.} \end{cases}$$

In the resulting MDP, a state is built from the state of the enacted system behavior (which includes the states of the environment and those of all available behaviors), the state of target behavior, and an action being requested; in other words, a "snapshot" of the whole composition problem. Each transition in the MDP represents the behavior—through its index—to which the current request is delegated for execution. The dynamics of the MDP encodes both the dynamics of the enacted system behavior and the target behavior, as well as that of the stochastic process (i.e., the user of the target) that is requesting actions. Finally, the reward function in the MDP merely mimics that of the encoded behavior composition problem; no reward is given from the initial dummy state, and an unfeasible delegation (i.e., one where the chosen behavior may not perform the action) receives a penalty (i.e., it is better to prescribe "u").

Given a policy for the encoded MDP, one can then extract the controller *induced* by it. Informally, an induced controller will mimic the decisions prescribed by its policy.

Definition 6.9 (Induced controller). Given a policy $\pi : Q \mapsto \mathsf{ind}$ for the MDP $\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}$, we define the <u>*$\pi$ induced controller*</u> $\mathsf{C}_\pi(h,a)$, where $h = s^0 \xrightarrow{a^1,k^1} \cdots \xrightarrow{a^\ell,k^\ell} s^\ell$ is an enacted system history, with $\ell \geq 0$ and $a \in A$, as $\mathsf{C}_\pi(h,a) = \pi(q)$ if:

- $\mathsf{sys}(q) = \mathsf{last}(h)$;
- $a = \mathsf{req}(q)$; and
- $t_0 \xrightarrow{\mathsf{env}(s^0):a^1} \cdots \xrightarrow{\mathsf{env}(s^{\ell-1}):a^\ell} \mathsf{tgt}(q)$ in $\mathcal{T}$.

□

Note that the output for histories that do not yield any legal evolution of the target is irrelevant, and hence, for these cases the induced controller will be undefined. The encoded MDP $\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}$ has a dummy start state $q_\sharp$ used to initialize the enacted system and the target behavior. As one would expect, the reward function of the encoded MDP and the step-wise reward for calculating the value of the controller are strongly linked together. Indeed, successfully delegating an action from a target and enacted system state will lead to same reward for a policy and its induced controller. It is natural then to expect that the value of an γ-optimal policy should equal the value of its induced controller oven an infinite horizon.

Lemma 6.3. *Let π be an γ-optimal policy for the MDP $\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}$ for a given target specification $\mathcal{T}$ and system $\mathcal{S}$. Then, $\gamma\mathcal{V}_\ell^{\mathsf{C}_\pi} = V_{\ell+1}^\pi(q_\sharp)$, where $\mathcal{V}_\ell^{\mathsf{C}_\pi}$ is the value of policy π induced controller C_π for ℓ steps, $V_{\ell+1}^\pi(q_\sharp)$ is the value of policy π from state $q_\sharp$ for $\ell+1$ steps, $0 \leq \gamma < 1$, and $l > 0$.*

Proof. We use induction on ℓ to prove our claim. Let $\mathcal{T} = \langle T, t_0, \delta, R, \mathcal{P}_{\mathsf{req}}\rangle$ be a target specification and $\mathcal{E}_\mathcal{S} = \langle S, A, \{1,\ldots,n\}, s_0, \mathcal{P}_\mathcal{S}\rangle$ the stochastic enacted system of $\mathcal{S}$. For $\ell = 1$ we obtain the following equations.

$$\begin{aligned} V_2^\pi(q_\sharp) &= r(q_\sharp,\pi(q_\sharp)) + \gamma \sum_{q\in Q} p(q_\sharp,\pi(q_\sharp),q) \times V_1^\pi(q) \\ &= r(q_\sharp,\pi(q_\sharp)) + \gamma \sum_{q\in Q} p(q_\sharp,\pi(q_\sharp),q) \times r(q,\pi(q)) \end{aligned}$$

Now, as per the encoded MDP definition, $r(q_\sharp, \pi(q_\sharp)) = 0$ and $\sum\limits_{q \in Q} p(q_\sharp, \pi(q_\sharp), q) \equiv \sum\limits_{a \in A} \mathcal{P}_{\mathsf{req}}(t_0, \mathsf{env}(s_0), a)$.

Note, the reward functions for both the controller and the encoded MDP are dependent only on the system state, chosen behavior index and the requested action, and both will return the same reward[1] ($R(\mathsf{tgt}(q), \mathsf{req}(q))$ or 0) for same set of inputs. Hence,

$$\begin{aligned} V_2^\pi(q_\sharp) &= 0 + \gamma \sum_{a \in A} [\mathcal{P}_{\mathsf{req}}(t_0, \mathsf{env}(s_0), a) \times IR^{C_\pi}(s_0, t_0, a)] \\ &= \gamma \mathcal{V}_1^{\mathsf{C}_\pi}(s_0, t_0) \\ &= \gamma \mathcal{V}_1^{\mathsf{C}_\pi} \end{aligned}$$

For the inductive step, assume that $\gamma \mathcal{V}_k^{\mathsf{C}_\pi} = V_{k+1}^\pi(q_\sharp)$ holds for some $k > 0$. We will show that $\gamma \mathcal{V}_{k+1}^{\mathsf{C}_\pi} = V_{k+2}^\pi(q_\sharp)$ also holds. First, let us calculate the difference $\mathcal{V}_{k+1}^{\mathsf{C}_\pi} - \mathcal{V}_k^{\mathsf{C}_\pi}$ by expanding the general recursive definition of $\mathcal{V}^{\mathsf{C}_\pi}$. Let $\mathsf{C} = \mathsf{C}_\pi$, $e_i = \mathsf{env}(s_i)$ for all $i \geq 0$, and $h_i = s_0 \xrightarrow{a_1, \mathsf{C}(s_0, a_1)} s_1 \cdots \xrightarrow{a_i, \mathsf{C}(s_{i-1}, a_i)} s_i$ for all $i \geq 0$ in the equations below.

$$\begin{aligned} \mathcal{V}_k^{\mathsf{C}} &= \sum_{a_1 \in A} [\mathcal{P}_{\mathsf{req}}(t_0, e_0, a_1) \times IR^{\mathsf{C}}(h_0, t_0, a_1) + \\ &\qquad \gamma \sum_{\substack{s_1 \in S \\ \langle t_0, e_0, a_1, t_1 \rangle \in \delta}} \mathcal{P}_{\mathsf{req}}(t_0, e_0, a_1) \times \mathcal{P}_{\mathcal{S}}(s_0, a_1, \mathsf{C}(s_0, a_1), s_1) \times \mathcal{V}_{k-1}^{\mathsf{C}}(h_1, t_1)] \\ &= \sum_{a_1 \in A} \mathcal{P}_{\mathsf{req}}(t_0, e_0, a_1) \times IR^{\mathsf{C}}(h_0, t_0, a_1) + \\ &\qquad \gamma \sum_{\substack{a_1 \in A \\ s_1 \in S \\ \langle t_0, e_0, a_1, t_1 \rangle \in \delta}} \mathcal{P}_{\mathsf{req}}(t_0, e_0, a_1) \times \mathcal{P}_{\mathcal{S}}(s_0, a_1, \mathsf{C}(s_0, a_1), s_1) \times \mathcal{V}_{k-1}^{\mathsf{C}}(h_1, t_1) \\ &= \sum_{a_1 \in A} \mathcal{P}_{\mathsf{req}}(t_0, e_0, a_1) \times IR^{\mathsf{C}}(h_0, t_0, a_1) + \cdots + \\ &\quad \gamma^{i-1} \sum_{\substack{a_j \in A \\ s_j \in S \\ \langle t_{j-1}, e_{j-1}, a_j, t_j \rangle \in \delta \\ j \leq i}} \mathcal{P}_{\mathsf{req}}(t_0, e_0, a_1) \times \mathcal{P}_{\mathcal{S}}(s_0, a_1, \mathsf{C}(s_0, a_1), s_1) \times \cdots \times \\ &\qquad \mathcal{P}_{\mathcal{S}}(s_{i-2}, a_{i-1}, \mathsf{C}(s_{i-2}, a_{i-1}), s_{i-1}) \times \mathcal{P}_{\mathsf{req}}(t_{i-1}, e_{i-1}, a_i) \times IR^{\mathsf{C}}(h_{i-1}, t_{i-1}, a_i) + \\ &\quad \gamma^{i} \sum_{\substack{a_j \in A \\ s_j \in S \\ \langle t_{j-1}, e_{j-1}, a_j, t_j \rangle \in \delta \\ j \leq i}} \mathcal{P}_{\mathsf{req}}(t_0, e_0, a_1) \times \mathcal{P}_{\mathcal{S}}(s_0, a_1, \mathsf{C}(s_0, a_1), s_1) \times \cdots \times \\ &\qquad \mathcal{P}_{\mathsf{req}}(t_{i-1}, e_{i-1}, a_i) \times \mathcal{P}_{\mathcal{S}}(s_{i-1}, a_i, \mathsf{C}(s_{i-1}, a_i), s_i) \times \mathcal{V}_{k-i}^{\mathsf{C}}(h_i, t_i) \\ &\qquad \text{where } i \leq k. \end{aligned}$$

We know that $\mathcal{V}_0^{\mathsf{C}} = 0$; hence using the above expansion we get:

$$\begin{aligned} &\mathcal{V}_{k+1}^{\mathsf{C}} - \mathcal{V}_k^{\mathsf{C}} = \\ &\quad \gamma^{k+1} \sum_{a_j \in A, s_j \in S, j \leq k+1} [\mathcal{P}_{\mathsf{req}}(t_0, e_0, a_1) \times \mathcal{P}_{\mathcal{S}}(s_0, a_1, \mathsf{C}(s_0, a_1), s_1) \times \cdots \times \\ &\qquad \mathcal{P}_{\mathsf{req}}(t_k, e_k, a_{k+1}) \times \mathcal{P}_{\mathcal{S}}(s_{k-1}, a_k, \mathsf{C}(s_{k-1}, a_k), s_k) \times IR^{\mathsf{C}}(h_k, t_k, a_{k+1})] \end{aligned} \tag{6.1}$$

[1]Though the reward function for the MDP can return $-R(\mathsf{tgt}(q), \mathsf{req}(q))$, since the index u is always applicable with a reward of 0, the reward function for an optimal policy will never return $-R(\mathsf{tgt}(q), \mathsf{req}(q))$.

Similarly, we calculate the difference $V^\pi_{k+2}(q_\sharp) - V^\pi_{k+1}(q_\sharp)$ by expanding the general recursive definition of $V^\pi_k(q_0)$, where $q_0 = q_\sharp$ and hence, $r(q_0, \pi(q_0) = 0)$.

$$\begin{aligned}
V^\pi_k(q_0) &= r(q_0, \pi(q_0)) + \gamma \sum_{q_1 \in Q} p(q_0, \pi(q_0), q_1) \times V^\pi_{k-1}(q_1) \\
&= \gamma \sum_{q_1 \in Q} p(q_0, \pi(q_0), q_1) \times [r(q_1, \pi(q_1)) + \gamma \sum_{q_2 \in Q} p(q_1, \pi(q_1), q_2) \times V^\pi_{k-2}(q_2)] \\
&= \gamma \sum_{q_1 \in Q} p(q_0, \pi(q_0), q_1) \times r(q_1, \pi(q_1)) + \\
&\qquad \gamma^2 \sum_{q_1, q_2 \in Q} p(q_0, \pi(q_0), q_1) \times p(q_1, \pi(q_1), q_2) \times V^\pi_{k-2}(q_2) \\
&= \gamma \sum_{q_1 \in Q} p(q_0, \pi(q_0), q_1) \times r(q_1, \pi(q_1)) + \cdots + \\
&\quad \gamma^i \sum_{\substack{q_j \in Q \\ j \le i}} p(q_0, \pi(q_0), q_1) \times \cdots \times p(q_{i-2}, \pi(q_{i-2}), q_{i-1}) \times r(q_{i-1}, \pi(q_{i-1})) + \\
&\qquad \gamma^i \sum_{\substack{q_j \in Q \\ j \le i}} p(q_0, \pi(q_0), q_1) \times \cdots \times p(q_{i-1}, \pi(q_{i-1}), q_i) \times V^\pi_{k-i}(q_i) \\
&\qquad\qquad \text{where } i \le k.
\end{aligned}$$

We know that $V^\pi_0(q) = 0$ for any state q in $\mathcal{M}_{\langle \mathcal{S}, \mathcal{T} \rangle}$; hence using the above expansion we get ($q_0 = q_\sharp$):

$$V^\pi_{k+2}(q_0) - V^\pi_{k+1}(q_0) = \gamma^{k+2} \sum_{\substack{q_j \in Q \\ j \le k+2}} p(q_0, \pi(q_0), q_1) \times \cdots \times p(q_k, \pi(q_k), q_{k+1}) \times r(q_{k+1}, \pi(q_{k+1})) \quad (6.2)$$

From the definition of $\mathcal{M}_{\langle \mathcal{S}, \mathcal{T} \rangle}$ we know that $p(q_0, \pi(q_0), q_1) = \mathcal{P}_{\mathsf{req}}(t_0, e_0, a_1)$ and $p(q_i, \pi(q_i), q_{i+1}) = \mathcal{P}_{\mathcal{S}}(s_i, a_{i+1}, \pi(q_i), s_{i+1}) \times \mathcal{P}_{\mathsf{req}}(t_{i+1}, e_{i+1}, a_{i+2})$ where $t_i = \mathsf{tgt}(q_i)$, $s_i = \mathsf{sys}(q_i)$, $e_i = \mathsf{env}(s_i)$ and $a_{i+1} = \mathsf{req}(q_i)$ for all $i > 0$. In order to substitute these in Equation 6.2, note that $\mathsf{sys}(q_1) = s_0$, $\mathsf{sys}(q_2) = s_1$, and so on. Substituting these and replacing π by induced controller C in Equation 6.2 we get:

$$\begin{aligned}
V^\pi_{k+2}(q_0) - V^\pi_{k+1}(q_0) = \\
\gamma^{k+2} \sum_{\substack{q_j \in Q \\ j \le k+1}} \mathcal{P}_{\mathsf{req}}(t_0, e_0, a_1) \times \mathcal{P}_{\mathcal{S}}(s_0, a_1, \mathsf{C}(s_0, a_1), s_1) \times \mathcal{P}_{\mathsf{req}}(t_1, e_1, a_2) \times \cdots \times \\
\mathcal{P}_{\mathcal{S}}(s_{k-1}, a_k, \mathsf{C}(s_{k-1}, a_k), s_k) \times \mathcal{P}_{\mathsf{req}}(t_k, e_k, a_{k+1}) \times R'(t_k, a_{k+1}) \quad (6.3)
\end{aligned}$$

where $R'(t_k, a_{k+1})$ is equal to $R(t_k, a_{k+1})$ if action a is executable from s_k, otherwise the value of R' is 0. We do not need to consider negative reward since π is optimal.

Similarly, from the definition of IR^{C} we get $IR^{\mathsf{C}}(h_k, t_k, a_{k+1}) = R'(t_k, a_{k+1})$. Combining this with Equations 6.1 and 6.3 we get:

$$V^{\pi}_{k+2}(q_\sharp) - V^{\pi}_{k+1}(q_\sharp) = \gamma(\mathcal{V}^{\mathsf{C}}_{k+1} - \mathcal{V}^{\mathsf{C}}_{k}).$$

Since $V^{\pi}_{k+1}(q_\sharp) = \gamma\mathcal{V}^{\mathsf{C}}_{k}$ we can conclude that $V^{\pi}_{k+2}(q_\sharp) = \gamma\mathcal{V}^{\mathsf{C}}_{k+1}$. □

Using Lemma 6.3 we show that optimising a policy is similar to optimising its induced controller. Hence, a γ-optimal policy will induce a γ-maximal controller.

Theorem 6.4. *Let $\mathcal{S}$ be an available system and $\mathcal{T}$ a target behavior. Let $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}$ be the corresponding* MDP *encoding as described above. If π is an γ-optimal policy for $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}$, then its induced controller C_π is a γ-maximal controller for realizing $\mathcal{T}$ in $\mathcal{S}$.*

Proof. Assume C^π is not γ-maximal and there exists a controller C^* such that $\mathcal{V}^{\mathsf{C}^*}_\gamma > \mathcal{V}^{\mathsf{C}_\pi}_\gamma$. Let π^* be the policy that induces C^*. Applying Lemma 6.3 we get $V^{\pi^*}_\gamma(q_\sharp) > V^{\pi}_\gamma(q_\sharp)$, a clear contradiction since we know that π is an optimal policy. □

Theorem 6.4 shows the correctness of the encoding, and provides us with a technique for solving DT-composition problems, by using, for instance, policy-iteration implementations [Howard 1960].

Example 6.4. We generated the optimal policy for the garden scenario from Figure 6.1 by using a simple existing MDP solver, called jMarkov.[2] The problem does *not* actually have an exact solution. To see that, consider the sequence of action requests *clean*· *water*· *empty* compatible with the target $\mathcal{T}_{Garden}$. It is not hard to verify that the first and last actions need to be delegated to bot $\mathcal{B}_{Cleaner}$, whereas the second action *water* ought to be delegated to bot $\mathcal{B}_{Multi}$. However, bot $\mathcal{B}_{Cleaner}$ will be able to perform the last action *empty* only if it has evolved to state a_1 after *clean*'s execution. Otherwise, if $\mathcal{B}_{Cleaner}$ happens to stay in state a_0 instead, the last action *empty* will not be realized in the system $\mathcal{S}_{Garden}$ and a dead-end will be reached.

Note, however, that the chances of $\mathcal{B}_{Cleaner}$ evolving to state a_0 are indeed low (precisely, a probability of 0.1). Hence, an optimal controller—a decision theoretic composition—should still choose $\mathcal{B}_{Cleaner}$ to execute the first *clean* action. This is indeed the controller induced by the optimal policy found when solving the corresponding MDP. This is partially listed below as output from jMarkov (`BEH0`, `BEH1`, and `BEH2` stand for behaviors $\mathcal{B}_{Cleaner}$, $\mathcal{B}_{Multi}$, and $\mathcal{B}_{Picker}$, respectively):

```
Beh:0 0 0 | Tgt:0| Env:0|Act:CLEAN ------> BEH0
Beh:0 0 0 | Tgt:1| Env:0|Act:WATER ------> BEH1
Beh:1 0 0 | Tgt:1| Env:0|Act:WATER ------> BEH1
```

[2]http://copa.uniandes.edu.co/software/jmarkov/

```
Beh:1 0 0 | Tgt:2| Env:0|Act:EMPTY ------> BEH0
Beh:0 0 0 | Tgt:2| Env:0|Act:EMPTY ------> U
...
```

Observe that if after performing a *clean* action, behavior $\mathcal{B}_{Cleaner}$ (BEH0) stays in its state a_0, the policy prescribes U, thus signaling a dead-end in the composition.

In turn, the following rules in the policy will successfully realize the request sequence $clean \cdot pick \cdot empty$:

```
Beh:0 0 0 | Tgt:0| Env:0|Act:CLEAN ------> BEH0
Beh:0 0 0 | Tgt:1| Env:0|Act:PICK ------> BEH1
Beh:0 1 0 | Tgt:1| Env:0|Act:WATER ------> BEH1
Beh:0 1 0 | Tgt:3| Env:0|Act:EMPTY ------> BEH1
```

Finally, bot $\mathcal{B}_{Picker}$ (BEH2) will be used by the induced controller in cases such as the following ones:

```
Beh:0 1 0 | Tgt:1| Env:0|Act:PICK ------> BEH2
Beh:0 1 1 | Tgt:0| Env:0|Act:CLEAN ------> BEH2
```

Observe that in the configuration of the second rule, behavior $\mathcal{B}_{Cleaner}$ is also able to perform the cleaning action; however, it is best to use the picker bot as this will bring it to state c_0, from where it is able to pick again if needed (note that bot $\mathcal{B}_{Multi}$ is in state b_1 from where it cannot *pick*).

All the above rules are only for the cases in which the environment remains in its state e_0; other (similar) rules exist in the policy/controller for other environment states.
□

6.4.2 Exact compositions

As discussed, in a decision theoretic composition problem, one looks, in general, for the "optimal" controller, since exact compositions may not exist. Nonetheless, the following result states that if an exact controller does exist, it is enough to restrict to the finite horizon case in the corresponding MDP (without losing optimality).

Theorem 6.5. *If there exists an exact composition for realizing a given target specification $\mathcal{T}$ in a system $\mathcal{S}$, then the controller induced by any $(|Q|+1)$-optimal policy for MDP $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle} = \langle Q, \mathit{ind}, p, r\rangle$ is an exact composition.*

Proof. This follows from the fact that there exists an optimal policy for $\mathcal{M}_{\langle\mathcal{S},\mathcal{T}\rangle}$ that is stationary (there exists a Markovian exact composition due to Theorem 6.1), and the fact that by optimizing the MDP up to $Q+1$ steps, it is guaranteed that *all* possible configurations of the whole composition framework—which includes both available system and target—are taken into account. □

This result is important in that it provides a way of verifying whether a DT-composition problem accepts an exact solution; namely, find an optimal policy π for horizon $|Q|+1$ and check whether $\mathcal{V}^{C_\pi}_{|Q|} = \mathcal{R}^{\max}_{|Q|}$ (recall the first step in the MDP involves no action request and attracts no reward). Of course, it is possible to restate the above theorem in terms of an infinite horizon problem:

Corollary 6.6. *If there exists an exact composition for realizing a given target $\mathcal{T}$ in a system $\mathcal{S}$, then there exists a discount factor $\hat{\gamma}$ such that for any γ-optimal policy π for MDP $\mathcal{M}_{\langle \mathcal{S},\mathcal{T}\rangle}$, with $\gamma \geq \hat{\gamma}$, the induced controller C_π is an exact composition of $\mathcal{T}$ in $\mathcal{S}$.*

When no exact composition exists, however, all one can do is to settle for the (optimal) controller induced by an optimal policy in the encoded MDP. Since non-exact compositions will include dead-ends; that is, possible histories where some target-compatible request may not be fulfilled, other mechanisms will be required to bring the overall system to a "healthy" configuration, such as resetting the whole system or even some parts of it.

We close this section by relating our approach to behavior composition to the "classical" approaches to the problem from the literature (e.g., [De Giacomo et al. 2013, De Giacomo and Felli 2010]). In such approaches, the task amounts to deciding whether an *exact* composition controller exists (and to synthesize one if any) in settings under strict uncertainty. The dynamics of behaviors and that of the environment are represented by means of *transition relations*, rather than probabilistic transition functions. As a result, the designer can only model whether a transition is possible or not. In addition, the target behavior does not include a probabilistic request function $\mathcal{P}_{\mathsf{req}}$, but simply a transition relation stating what actions can be legally requested.

As expected, the following result states that our DT-composition framework is *at least as expressive* as the classical one.

Theorem 6.7. *For any instance of a classical behavior composition, there is a decision-theoretic behavior composition instance such that there exists a composition solution for the former iff there exists an exact composition for the latter.*

Proof. Let $\langle \mathcal{S}, \mathcal{T}\rangle$ be a classical behavior composition problem where $\mathcal{T} = \langle T, G, t_0, \varrho\rangle$ is the target specification and $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E}\rangle$ be the system with behaviors $\mathcal{B}_i = \langle B_i, G_i, b_{i0}, \varrho_i\rangle$, for $i \leq n$, and environment $\mathcal{E} = \langle E, A, e_0, \rho\rangle$. We build a DT-composition problem instance with target $\mathcal{T}^{\mathsf{d}} = \langle T, t_0, \delta, R, \mathcal{P}_{\mathsf{req}}\rangle$, behaviors $\mathcal{B}^{\mathsf{d}}_i = \langle B_i, b_0, \mathcal{P}^{\mathcal{B}_i}\rangle$ for $i \leq n$, and environment $\mathcal{E}^{\mathsf{d}} = \langle E, A, e_0, \mathcal{P}_{\mathcal{E}}\rangle$ as follows:

- The environment probabilistic transition function is defined such that $\mathcal{P}^{\mathcal{E}}(e, a, e') = 1/|\Delta(e, a)|$, whenever $\langle e, a, e'\rangle \in \rho$, where ρ is the transition relation of the original classical environment and $\Delta(e, a) = \{e' \mid \langle e, a, e'\rangle \in \rho\}$.

- The probabilistic transition function for each available behavior $\mathcal{B}_i^{\mathsf{d}}$ is defined as $\mathcal{P}^{\mathcal{B}_i}(b, e, a, b') = 1/|\Delta(b, e, a)|$, whenever $\langle b, g, a, b'\rangle \in \varrho_i$ such that $g(e) = \mathtt{true}$, where ϱ_i is the transition relation of the original classical available behavior $\mathcal{B}_i$ and $\Delta(b, e, a) = \{b' \mid \langle b, g, a, b'\rangle \in \varrho_i, g(e) = \mathtt{true}\}$.

- The probabilistic action request function of the target behavior is defined as $\mathcal{P}_{\mathsf{req}}(t, e, a) = 1/|\Delta(t, e)|$, whenever $\langle t, g, a, t'\rangle \in \varrho$ such that $g(e) = \mathtt{true}$, where ϱ is the transition relation of the original target and $\Delta(t, e) = \{a \mid \langle t, g, a, t'\rangle \in \varrho, g(e) = \mathtt{true}\}$.

- The target reward function is defined as $R(t, a) = 1$ for all $a \in A$ and $t \in T$ such that $\mathcal{P}_{\mathsf{req}}(t, e, a) > 0$ for some $e \in E$.

In all other cases, the probabilities are assumed to be zero. Assume the target specification is realizable in system $\mathcal{S}$ and C is an exact composition for the classical composition problem $\langle \mathcal{S}, \mathcal{T}\rangle$. We show that C is also an exact composition for stochastic behavior composition problem $\langle \mathcal{S}^{\mathsf{d}}, \mathcal{T}^{\mathsf{d}}\rangle$, where $\mathcal{S}^{\mathsf{d}} = \langle \mathcal{B}_1^{\mathsf{d}}, \ldots, \mathcal{B}_n^{\mathsf{d}}, \mathcal{E}^{\mathsf{d}}\rangle$. Since C is an exact composition, it will realize all traces of enacted target $\mathcal{E}_{\mathcal{T}}$ in system $\mathcal{S}$. Using C on $\mathcal{T}^{\mathsf{d}}$ in $\mathcal{S}^{\mathsf{d}}$, after k steps, it will gain a reward exactly equal to k ($R(t, a)$=1 in our reduction). Observe that, as per the encoding above, $\mathcal{R}_k^{max} = k$. Hence, C is also an exact composition for $\mathcal{T}^{\mathsf{d}}$ in $\mathcal{S}^{\mathsf{d}}$.

For the opposite direction, assume that C is an exact composition for $\mathcal{T}^{\mathsf{d}}$ in $\mathcal{S}^{\mathsf{d}}$ but not for $\mathcal{T}$ in $\mathcal{S}$. Hence, there exists a trace τ of enacted system $\mathcal{E}_{\mathcal{T}}$ which C cannot realize. Let $\tau = s_{\mathcal{T}}^0 \xrightarrow{a^1, g^1} \cdots \xrightarrow{a^\ell, g^\ell} s_{\mathcal{T}}^{\ell+1}$, and h be a τ compatible system history such that $\mathsf{C}(h, a^\ell)$ is undefined; that is C cannot honor the action request a^ℓ when the system is in $\mathsf{last}(h)$. Consequently, if C was used to realize $\mathcal{T}^{\mathsf{d}}$ in $\mathcal{S}^{\mathsf{d}}$ then it will not gain the immediate expected reward $R(t^{\ell-1}, a^\ell)$. Therefore, after ℓ steps, $V_\ell^{\mathsf{C}} < \mathcal{R}_\ell^{max}$: a contradiction since C is an exact composition for $\mathcal{T}^{\mathsf{d}}$ in $\mathcal{S}^{\mathsf{d}}$, and so $\mathcal{V}_k^{\mathsf{C}} = \mathcal{R}_k^{max}$ should hold for all $k > 1$. □

Clearly, not every DT-composition problem can be mapped to the classical setting, as is the case with our gardening scenario. It follows then that the framework developed here, not surprisingly, is a strict extension of the classical one for behavior composition.

In terms of computational complexity, fully observable MDPs can be solved in time polynomial in the size of state space and actions [Puterman 2005]. Since the size of $\mathcal{M}_{\langle \mathcal{S}, \mathcal{T}\rangle}$'s state space is indeed exponential in the number of behaviors, checking if a controller is maximal is exponential in the number of available behaviors. Observe that this complexity bound is tight since checking the existence of an exact controller can be reduced to a decision theoretic problem (see Theorem 6.7).

Note that the upper bound in the quantitative setting is in EXPTIME even for the general case, which is less than the current upper bound for computing SRTFs in the qualitative optimisation setting involving nondeterministic available behaviors. However, the decision theoretic framework suffers from two key limitations. First, qualitative properties such as dead-end avoidance cannot be guaranteed. A decision theoretic composition will maximize the *expected realizability* of a target specification in a system. Hence, a system may evolve nondeterministically to a low probability state from where subsequent target requests will not be honored. Alternatively, a target specification may associate high reward with an action that may be nondeterministic in the given system such that one of its nondeterministic evolution causes the system to reach a dead-end. Second, the decision theoretic framework does not provide insights into any extra information that could render an unsolvable problem solvable. For instance, in the qualitative framework, SRTFs were allowed to introduce extra branching to capture information such as advanced selection of action requests. In the quantitative framework developed here, one seeks to maximize the rewards based on the given target specification instead of proposing the best alternate target specification.

6.5 Summary

In this chapter, we generalized the classical behavior composition problem to one that is able to account for *quantified uncertainties* in the domain, both in the dynamics of the behaviors and environment, as well as in the preferences over requests from the target user. The task then is to find an optimal controller—a decision theoretic composition—that maximizes the *expected realizability* of the target. In order to solve a DT-composition problem, we showed how to reduce it to the problem of finding an optimal policy in a Markov decision process, an established framework for sequential stochastic decision making.

To summarise:

- We proposed a decision theoretic account of behavior composition based on quantification of uncertainties.
- The probabilistic framework quantifies the nondeterminism in the available behaviors, environment, and the frequency of requests along with importance of actions in target specification.
- We defined the notion of *maximal* compositions based on "expected realizability" of the target specifications.
- We provided a technique to compute maximal compositions via reduction to Markov decision processes.

CHAPTER 7

Conclusion

"We know very little, and yet it is astonishing that we know so much, and still more astonishing that so little knowledge can give us so much power."
–*Bertrand Russell*

Behavior composition involves automatically synthesising a controller to implement a desired target specification by utilising available behaviors that operate in a shared environment. In the classical setting, one is only interested in *exact controllers* that are able to completely realize a given target specification. A shortcoming of the classical setting is its lack of capability of dealing with composition instances where an exact controller does not exist; that is, where the target specification cannot be realized by using the available behaviors. The focus of this thesis has been to develop behavior composition frameworks and techniques that will cater for unsolvable problem instances, thereby making behavior composition applicable to broader range of cases. With the overarching idea to look for *optimal* solutions in unsolvable problem instances, we defined what such optimal solutions are, and provided techniques to compute them, in *qualitative* and *quantitative* settings.

In the qualitative approach, developed in Chapter 3, we focused on the target's perspective and characterized *supremal realizable target fragments* (SRTFs)–optimal target fragments that accommodate exact controllers in the given system. We proved that the notion of SRTFs is both *sound* and *complete* with respect to controllers (Theorems 3.9 and 3.10). More importantly, we showed that SRTFs are unique up to simulation equivalence (Theorem 3.8). In order to compute the SRTF for problem instances containing only deterministic available behaviors (Section 4.1), we reduced the qualitative optimisation problem to a particular safety game [Bloem et al. 2011]. For the general case involving nondeterministic behaviors, we relied on sort of belief-space [Bonet and Geffner 2000] construction technique (Section 4.2). In Chapter 5, we introduced *uncontrollable exogenous* events in our qualitative optimisation framework to represent observable uncertainties and suitably adapted the solution concepts, along with the belief-space construction technique. Throughout the qualitative approach, we formulated the notion of optimal solutions along with their properties based on the formal notion of simulation [Milner 1971].

In comparison to the qualitative approach that assumed *strict uncertainty* [French 1986], the quantitative approach required additional domain knowledge. The outcome of the quantitative approach, developed in Chapter 6, is a *decision-theoretic* behavior composition framework, in which the task is to maximize the so-called *expected realizability* of a target behavior in a given available system. To facilitate this, we quantified all sources of uncertainty in the classical composition framework. First, the uncertainty on the nondeterminism in available behaviors and the environment needs to be measured. Second, the relative importance of each potential target request at a given state is specified. With the uncertainty quantified into the model, we proposed the notion of *maximal composition controllers* as those which maximize the expected target realizability. We constructed a technique to compute maximal composition controllers by encoding the problem into a particular kind of Markov decision process [Puterman 2005]. Importantly, we showed how to extract a maximal controller from an optimal policy of the encoded MDP (Theorem 6.4).

As one would expect, an optimal solution for solvable composition instances will coincide with the exact one. In particular, for such cases, the SRTF of the qualitative framework will be simulation equivalent to its target specification (Theorem 3.11); in the quantitative framework a maximal controller will also be an exact controller (Theorem 6.5). More importantly, we proved that the qualitative and quantitative optimisation frameworks developed in this thesis strictly subsume the classic behavior composition problem (Theorems 3.1 and 6.7).

The choice between using a qualitative or quantitative approach to behavior composition optimisation depends on user requirements and available domain information. The qualitative approach aims at providing an optimal solution; that is, the SRTF, without requiring any further domain knowledge or additional behaviors. In fact, the input to the qualitative optimisation problem is the same as the classical behavior composition problem (except for allowing nondeterministic targets). SRTFs have an advantage of being in the same language as the problem specification and they come with the guarantee of full realizability; that is, there will exist a controller that will never get stuck while honoring requests as per the SRTF. However, the current technique to compute SRTFs for the general case has a time complexity of 2-EXPTIME, which is higher than the time complexity of computing exact compositions for the classical behavior composition. In contrast, the quantitative approach relies on the availability of additional domain knowledge to quantify the sources of uncertainty in the domain. Though, in the quantitative setting, one obtains a maximal composition that provides maximum realizability of the target specification, a maximal controller may get stuck; that is, at a certain step the next legal request may not be realizable. However, one can compute maximal compositions in the same time complexity as computing exact compositions.

Behavior composition and related fields

From an AI perspective, automated planning [Ghallab et al. 2004], supervisory control theory (SCT) [Wonham and Ramadge 1987, Cassandras and Lafortune 2006] and behavior composition are all synthesis problems. The aim in planning is to build a plan, in SCT to build a supervisor, and in behavior composition to generate a controller. Observe that, at the core, these problems are concerned with qualitative temporal decision making in dynamic domains and exhibit strong resemblance in how their problem components are modeled (e.g., using transition system like models) and the techniques used to solve the problem (e.g., model checking, search, etc). In fact, exploration of the relationship between these three synthesis tasks has already gained attention [Balbiani et al. 2008, Bertoli et al. 2010, Barbeau et al. 1995].

The task in classical planning [Ghallab et al. 2004] is to generate a *plan* for a given goal that is meant to be achieved in a deterministic fully observable domain starting from a known initial state. Advanced forms of planning formalisms include relaxing the deterministic and fully observable assumptions on the domain [Bertoli et al. 2001a, Ghallab et al. 2004] and synthesising plans for temporally extended goals [De Giacomo and Vardi 2000]. Classical behavior composition problem can be considered to be an advanced form of planning with a *maintenance goal*, namely, to always satisfy the target's request, in a nondeterministic domain. From a planning perspective, a plan (i.e., a controller) prescribes behavior delegations rather than domain actions [De Giacomo and Sardina 2007]. In particular, classical behavior composition problems have been considered under various planning frameworks, including planning as model checking [Pistore et al. 2004], planning in asynchronous domains [Bertoli et al. 2010], and nondeterministic planning [Ramirez et al. 2013].

Supervisory control theory [Wonham and Ramadge 1987, Cassandras and Lafortune 2006] concerns itself with restriction of a *plant* (modelled by an automaton) such that the restricted plant's language equals a given specification. As argued in section 5.4 with respect to behavior composition and SCT, it may seem that both theses areas tackle the same problem, possibly from different perspectives: SCT from an Engineering perspective and composition from a Computer Science one. However, the core control problem in SCT and behavior composition are different. In SCT one seeks to control the whole plant, hence it does not have a notion analogous to available behaviors. Therefore, component-based nondeterminism cannot be captured (directly) in a plant. Another important mismatch relates to the semantics of nondeterminism: the nondeterminism of controllable actions in a plant is *angelic*, in the sense that the supervisor can control its evolution. On the other hand, nondeterminism of available behaviors is *devilish*, as it cannot be controlled. This is one of the reasons why, as far as we know, SCT frameworks such as DES do not have a notion similar to ND-simulation [Sardina et al. 2008]. In fact, uncertainty is modelled in SCT via (deterministic) uncontrollable events [Wonham and Ramadge 1987], whereas nondeterminism [De Giacomo et al. 2013] is used in behavior composition.

From an automata theoretic viewpoint, behavior composition is related to *shuffled languages* [Berglund et al. 2011]. Briefly, the shuffle of two strings $u = ab$ and $v = cd$ is the set of all possible interleavings of symbols in u and v; that is, $\{abcd, acbd, cabd, cadb, cdab\}$. The shuffle between two languages is then simply the shuffle between the words contained in them [Berglund et al. 2011]. If we consider the domain actions as symbols, then the language of the enacted system is the shuffle between the languages of the enacted behaviors. Since the available behaviors act in an *interleaved* fashion, the enacted system encodes *all* the possible interleavings that arise due to activating behaviors in different order. Conceptually, a target specification can be realized in a system if the language of its enacted target is a subset of the language of the enacted system (i.e., a shuffle of the languages of the enacted behaviors). This will hold true only for deterministic available behaviors due to different treatment of nondeterminism in automata and behavior composition. In automata theory, a word is accepted if *any* of the nondeterministic evolutions reaches a final state. However, in behavior composition (from an automata theoretic perspective) *all* nondeterministic (system) evolutions should be accepted.

Future work

There are multiple lines of possible future work in the area of behavior composition optimisation. First and foremost, we aim to confirm the conjecture that our technique builds SRTFs that are optimal with respect to worst-case complexity, thus implying that synthesis of SRTFs is, in general, more difficult than synthesis of exact composition controllers. Second, one may design *anytime* algorithms to compute SRTFs. The central idea will be to begin with the universally smallest RTF (one with single state and no transitions) and incrementally build better RTFs successively. Orthogonal to this anytime approach, since RTFs are closed under union, one could compute various RTFs *in parallel* before taking their union.

Third, one may propose approaches that trade optimality for faster computation, such as restricting realizable target fragments to merely removing transitions from the original target specification, or bounding its number of states. A *resource bounded* account of behavior composition will be beneficial in situations such as embedded systems where memory or computation might be limited. Fourth, it would be interesting to develop a *hybrid* framework combining the qualitative and quantitative optimisation approaches. The central idea in such a framework would be to use the qualitative approach to compute the SRTF and the quantitative technique for maximising the realizability of the remaining target specification. Such a hybrid framework would also highlight fragments of the target specification that need extra domain knowledge (parts of the target that are included in its SRTF will not require extra domain information). Finally, it will be of practical use to go beyond the theoretical foundations of frameworks, as presented in this thesis, and perform an empirical analysis on computing optimal solutions as attempted for classical behavior composition [Ramirez et al. 2013].

One could encourage other approaches orthogonal to the optimisation techniques when confronted with a behavior composition problem instance admitting no complete solution. For example, one could look for additional available behavior modules or enhancement of existing ones with new capabilities to recover exact solvability. In some cases, simply adding extra “copies” of existing modules would be enough. A related (open) issue in behavior composition is with regard to *redundancy* of behaviors. All the current techniques to compute an exact (or optimal) solution utilize *all* the available behaviors. Presented with a problem having multiple copies of a behavior there is no technique that can *intelligently* recognize if the target specification can be realized by a subset of those copies. Of course, one would like to avoid naively calculating the required number of copies by doing an iterative deepening style of computation by increasing the number of copies used in each iteration. Since the computational complexity of the classical behavior composition problem is exponential in the number of behaviors, an intelligent technique able to include behaviors in the enacted system in an on-the-fly as required manner will have a direct impact on the practical efficiency.

In all the approaches and suggestions discussed so far a target specification is assumed to be final. One could relax this assumption and treat a target specification as partial. Intuitively, a *partial* target specification would imply that the controller is allowed to “fill-in” some actions in order to realize the next target request. This could be essential in situations where the modeller may have forgotten to incorporate certain conditions. For example, if the modeller misses the turn-off action, the controller will automatically execute it on her behalf. In addition, this may provide an *interactive* approach to synthesising practically meaningful target specifications by allowing the modeller and controller to collaborate with each other.

We conclude by noting that the area of behavior composition poses interesting, and important, research challenges (some of which are outlined above). In particular, this thesis provides a first example of behavior composition frameworks that cater for target specifications that do not have exact controllers in a given system. We proposed qualitative and quantitative approaches to formally define and construct *optimal* solutions in these cases. We borrowed the idea of *exogenous events* from the fields of reasoning about action [Reiter 2001] and supervisory control theory [Wonham and Ramadge 1987, Cassandras and Lafortune 2006] to provide an interesting extension to the classical behavior composition setting. Finally, we provided sound effective techniques to synthesize optimal solutions for the various proposed frameworks.

Bibliography

R. Alur, T. A. Henzinger, F. Y. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. Mocha: Modularity in model checking. In *Computer Aided Verification*, pages 521–525. Springer, 1998.

R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM (JACM)*, 49(5):672–713, 2002.

E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems II*, volume 999 of *Lecture Notes in Computer Science (LNCS)*, pages 1–20. Springer, 1995.

C. Baier, J.-P. Katoen, et al. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.

P. Balbiani, F. Cheikh, and G. Feuillade. Composition of interactive web services based on controller synthesis. In *Proceedings of the IEEE Congress on Services (SERVICES)*, pages 521–528, 2008.

J. Balcázar, J. Gabarro, and M. Santha. Deciding bisimilarity isp-complete. *Formal aspects of computing*, 4(1):638–648, 1992.

M. Barbeau, F. Kabanza, and R. St-Denis. Synthesizing plant controllers using real-time goals. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 791–800, 1995.

R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

D. Berardi. *Automatic composition services: models, techniques and tools.* PhD thesis, PhD thesis, SAPIENZA–Universita di Roma, Dipartimento di Informatica e Sistemistica, 2005.

D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Service-Oriented Computing-ICSOC 2003*, pages 43–58. Springer, 2003a.

D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. E-service composition by description logics based reasoning. In *Proceedings of the Int. Workshop on Description Logics (DL03)*, 2003b.

D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioural descriptions. *International Journal of Cooperative Information Systems*, 14(4):333–376, 2005.

D. Berardi, F. Cheikh, G. De Giacomo, and F. Patrizi. Automatic service composition via simulation. *International Journal of Foundations of Computer Science*, 19(2):429–452, 2008.

M. Berglund, H. Björklund, and J. Högberg. Recognizing shuffled languages. In *Language and Automata Theory and Applications*, pages 142–154. Springer, 2011.

P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2001, pages 473–478, 2001a.

P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 17, pages 473–478. Citeseer, 2001b.

P. Bertoli, M. Pistore, and P. Traverso. Automated composition of web services via planning in asynchronous domains. *Artificial Intelligence*, 174(3):316–361, 2010.

R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, pages 1–28, 2011.

B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *Proceedings of Artificial Intelligence Planning Systems*, pages 52–61. AAAI, 2000.

L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are two web services compatible? In *Technologies for E-Services*, pages 15–28. Springer, 2005.

C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11(1):94, 1999.

A. Bredenfeld. *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020. Springer, 2006.

A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Oreback. Towards component-based robotics. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 163–168, 2005.

D. Brugali, A. Brooks, A. Cowley, C. Côté, A. C. Domínguez-Brito, D. Létourneau, F. Michaud, and C. Schlegel. Trends in component-based robotics. In *Software Engineering for Experimental Robotics*, pages 135–142. Springer, 2007.

J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L.-J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and computation*, 98(2):142–170, 1992.

D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella, and F. Patrizi. Automatic service composition and synthesis: the roman model. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 31(3):18–22, 2008.

C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Springer, Secaucus, NJ, USA, 2006. ISBN 0387333320.

A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Computer Aided Verification*, pages 359–364. Springer, 2002.

G. De Giacomo and P. Felli. Agent composition synthesis based on ATL. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 499–506, 2010.

G. De Giacomo and F. Patrizi. Automated composition of nondeterministic stateful services. In *Proceedings of the International Workshop on Web Services and Formal Methods (WSFM)*, volume 6194 of *Lecture Notes in Computer Science (LNCS)*, pages 147–160. Springer, 2010.

G. De Giacomo and S. Sardina. Automatic synthesis of new behaviors from a library of available behaviors. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1866–1871, 2007.

G. De Giacomo and M. Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *Recent Advances in AI Planning*, pages 226–238. Springer, 2000.

G. De Giacomo, R. De Masellis, and F. Patrizi. Composition of partially observable services exporting their behaviour. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 9, 2009.

G. De Giacomo, F. Patrizi, and S. Sardina. Agent programming via planning programs. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 491–498. International Foundation for Autonomous Agents and Multiagent Systems, 2010a.

G. De Giacomo, F. Patrizi, and S. Sardina. Generalized planning with loops under strong fairness constraints. In *Proceedings of Principles of Knowledge Representation and Reasoning (KR)*, pages 351–361, Toronto, Canada, May 2010b. Proceedings of Principles of Knowledge Representation and Reasoning (KR).

G. De Giacomo, F. Patrizi, and S. Sardina. Automatic behavior composition synthesis. *Artificial Intelligence Journal*, 196:106–142, 2013.

S. French. *Decision Theory: An Introduction to the Mathematics of Rationality.* Ellis Horwood, 1986.

M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice.* Morgan Kaufmann, 2004.

K. Gill, S.-H. Yang, F. Yao, and X. Lu. A ZigBee-based home automation system. *Consumer Electronics, IEEE Transactions on*, 55(2):422–430, 2009.

A. Girard and G. J. Pappas. Approximation metrics for discrete and continuous systems. *Automatic Control, IEEE Transactions on*, 52(5):782–798, 2007.

D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* The MIT Press, 2000.

P. Haslum and P. Jonsson. Some results on the complexity of planning with incomplete information. In *Recent Advances in AI Planning*, pages 308–318. Springer, 2000.

G. T. Heineman, I. Crnkovic, and H. W. Schmidt. *Component-based software engineering.* Springer, 2005.

J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, And Computation.* Addison-Wesley, 2007.

R. Howard. *Dynamic Programming and Markov Process.* MIT Press, 1960.

L. S. Humphries, G. Rasmussen, D. L. Voita, and J. D. Pritchett. Home automation system, Apr. 15 1997. US Patent 5,621,662.

B. Jobstmann and R. Bloem. Optimizations for ltl synthesis. In *Formal Methods in Computer Aided Design, 2006. FMCAD'06*, pages 117–124. IEEE, 2006.

B. Jobstmann, S. Galler, M. Weiglhofer, and R. Bloem. Anzu: A tool for property synthesis. In *Proceedings of the International Conference on Computer Aided Verification (CAV)*, pages 258–262, 2007.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

P. C. Kanellakis and S. A. Smolka. Ccs expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

J.-H. Kim, S.-J. Han, and J.-B. Lee. Vacuum cleaner and control method thereof, Nov. 24 1998. US Patent 5,841,259.

J.-H. Kim, Y.-D. Kim, and K.-H. Lee. The third generation of robotics: Ubiquitous robot. In *Proceedings of the 2nd Int Conf on Autonomous Robots and Agents*, 2004.

A. Lomuscio and F. Raimondi. Mcmas: A model checker for multi-agent systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 450–454. Springer, 2006.

A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Proceedings of the International Conference on Computer Aided Verification (CAV)*, pages 682–688, 2009.

R. Lundh, L. Karlsson, and A. Saffiotti. Automatic configuration of multi-robot systems: Planning for multiple steps. In *Proceedings of the European Conference in Artificial Intelligence (ECAI)*, volume 8, pages 616–620, 2008.

Y. Lustig and M. Y. Vardi. Synthesis from component libraries. In *Foundations of Software Science and Computational Structures*, pages 395–409. Springer, 2009.

Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 6(1):68–93, 1984.

C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design: OBDD-foundations and applications.* Springer, 1998.

R. Milner. An algebraic definition of simulation between programs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 481–489, 1971.

R. Milner. *Communication and concurrency.* Prentice-Hall, Inc., 1989.

G. E. Monahan. State of the arta survey of partially observable markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

A. Muscholl and I. Walukiewicz. A lower bound on web services composition. In *Proceedings of the 10th Int. Conf. on Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 4423 of *Lecture Notes in Computer Science (LNCS).* Springer, 2007.

H. Palacios and H. Geffner. Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 900–905, 2006.

F. Patrizi. *Simulation-based Techniques for Automated Service Composition.* PhD thesis, PhD thesis, SAPIENZA–Universita di Roma, Dipartimento di Informatica e Sistemistica, 2009.

M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and monitoring web service composition. In *Artificial Intelligence: Methodology, Systems, and Applications*, pages 106–115. Springer, 2004.

N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive (1) designs. In *Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.

A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Automata, Languages and Programming*, pages 652–671. Springer, 1989a.

A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190. ACM, 1989b.

A. Pnueli and E. Shahar. The tlv system and its applications. Technical report, Weizmann Institute, 1996.

A. Pnueli, Y. Saar, and L. D. Zuck. JTLV: A framework for developing verification algorithms. In *Computer Aided Verification*, pages 171–174. Springer, 2010.

M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*, volume 414. John Wiley & Sons, Inc., 2005.

M. Ramirez, N. Yadav, and S. Sardiña. Behavior composition as fully observable non-deterministic planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS).* AAAI Press, 2013.

J. Rao and X. Su. A survey of automated web service composition methods. In *Semantic Web Services and Web Process Composition*, pages 43–54. Springer, 2005.

R. Reiter. *Knowledge in action: logical foundations for specifying and implementing dynamical systems.* Cambridge University Press, 2001.

J. Rintanen. Complexity of planning with partial observability. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 4, pages 345–354, 2004.

A. Saffiotti and M. Broxvall. Peis ecologies: Ambient intelligence meets autonomous robotics. In *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, pages 277–281. ACM, 2005.

J. Sametinger. *Software engineering with reusable components.* Springer, 1997.

S. Sardina and G. De Giacomo. Realizing multiple autonomous agents through scheduling of shared devices. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 304–312, 2008.

S. Sardina and G. De Giacomo. Composition of congolog programs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 9, pages 904–910, 2009.

S. Sardina, F. Patrizi, and G. De Giacomo. Automatic synthesis of a global behavior from multiple distributed behaviors. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1063–1069, 2007.

S. Sardina, F. Patrizi, and G. De Giacomo. Behavior composition in the presence of failure. In *Proceedings of Principles of Knowledge Representation and Reasoning (KR)*, pages 640–650, 2008.

E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. Htn planning for web service composition using shop2. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(4):377–396, 2004.

D. E. Smith and D. S. Weld. Conformant graphplan. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 889–896, 1998.

T. Stroeder and M. Pagnucco. Realising deterministic behaviour from multiple non-deterministic behaviours. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 936–941, 2009.

Y. Sun, H. Lin, F. Liu, and B. M. Chen. Computation for supremal simulation-based controllable subautomata. In *Control and Automation (ICCA), 2010 8th IEEE International Conference on*, pages 1450–1455, 2010.

Y. Sun, H. Lin, and F. Liu. Computation for supremal simulation-based controllable and strong observable subautomata. In *Control Conference (CCC), 2012 31st Chinese*, pages 2128–2133, 2012.

M. W. Tilden. The evolution of functional robo-ecologies. *Ars Electronica*, 93:195–200, 1993.

C.-C. Tsai, Y.-S. Wang, Y.-Y. Li, and F.-C. Tai. Cooperation and task execution of an anthropomorphous two-armed robot: An application to coffee making. In *System Science and Engineering (ICSSE), 2010 International Conference on*, pages 239–244. IEEE, 2010.

R. Van Ommering, F. Van Der Linden, J. Kramer, and J. Magee. The Koala component model for consumer electronics software. *Computer*, 33(3):78–85, 2000.

M. B. van Riemsdijk, M. Dastani, and J.-J. C. Meyer. Semantics of declarative goals in agent programming. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 133–140. ACM Press, 2005.

W. M. Wonham and P. J. Ramadge. On the supremal controllable sub-language of a given language. *SIAM Journal on Control and Optimization*, 25(3):637–659, 1987.

N. Yadav and S. Sardiña. Decision theoretic behavior composition. In L. Sonenberg, P. Stone, K. Tumer, and P. Yolum, editors, *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 575–582. IFAAMAS, 2011.

N. Yadav and S. Sardiña. Qualitative approximate behavior composition. In L. F. del Cerro, A. Herzig, and J. Mengin, editors, *Proceedings of the European Conference on Logics in Artificial Intelligence (JELIA)*, volume 7519 of *Lecture Notes in Computer Science*, pages 450–462. Springer, 2012. ISBN 978-3-642-33352-1.

N. Yadav, P. Felli, G. DeGiacomo, and S. Sardiña. On the supremal realizability of behaviors with uncontrollable exogenous events. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).* AAAI Press, 2013.