Synthesis for parameterized systems

Mathieu Lehaut

Synthesis for Parameterized Systems

Mathieu Lehaut

Contents

Chapter 1

Introduction

Parameterized Systems

The first computers were machines used to solve problems in a simple way: a single task was solved by a single machine. However, this view certainly does not hold today. Progress in communication networks made it possible for multiple computers to cooperate to solve a single task, whether located next to each other or halfway across the world. Even with a single computer, a single task can be performed by multiple processes each supported by different cores.

Indeed, nowadays many systems are composed of a number of distinct but interacting participants. For instance, a fleet of drones may cooperate in order to explore some unknown terrain by communicating between them the data they gathered. Or, imagine some kind of distributed computing where computing nodes are located in several different data centers. They are sent parts of a problem to be solved, and then those solutions are assembled to compute the final answer to the initial problem. Another example would be communication protocols, where routers collaborate in order to create efficient paths to relay data from point A to point B. Such systems are usually called *distributed systems*.

One important parameter of a distributed system is the number of entities taking part in this system. Indeed, a programmer may write different programs depending on whether there are two, ten, or more participants. However, it is not always possible to know this number in advance. Sometimes, the number of active participants is only known at the beginning of the execution. Sometimes, new entities can dynamically join the system in the middle of an execution. In any case, the system should always be able to complete the task it has been given, whatever the number of participants. Those distributed systems called *parameterized systems*. In this work, participants in a parameterized system will generally be called *processes*.

The main difficulty when reasoning about parameterized systems is that there is no bound on the number of processes involved in that system. We can further distinguish two cases of parameterized systems: when for each execution of the system the number of processes is fixed throughout the execution (although not known in advance), and when new processes can be added during an execution. The former case is the *parameterized* case, while the latter will be called the *dynamic* case. See for instance [BJK+15] for results on verification for the parameterized case, and [BMOT05] for an example of model using dynamic creation of threads.

Moreover, we consider systems where processes may interact with an external

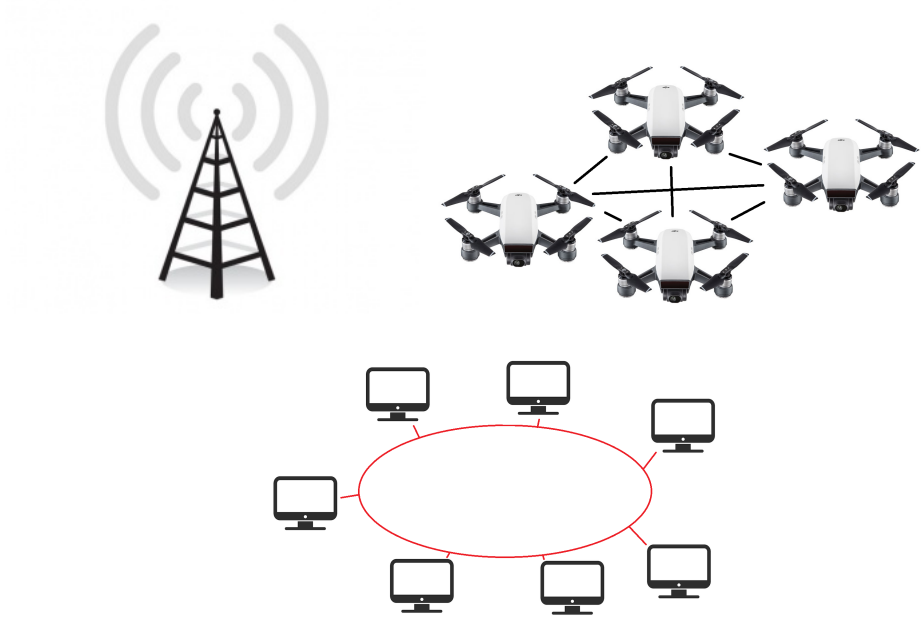

Figure 1.1: Illustrations of various distributed systems

environment that is not under control of the system. This uncontrollable environment can model various real-world phenomena. In the case of a drone fleet exploring some unknown terrain, drones may be equipped with sensors that gather data about that terrain, but obviously they cannot control whether, e.g., there is a wall right in front of them, or if the terrain is flat and has no obstacle. For distributed computing, an operator may input some data during the execution to alter the flow of the computation.

Whatever the case is, the system needs to react to these perturbations and still manage to fulfill its task. Systems interacting with an uncontrollable environment are called *open systems*, or also *reactive systems*, as opposed to closed systems where everything is controllable. The environment adds another layer of complexity when dealing with such systems. Since it cannot be controlled, it is often modeled as an hostile antagonist, that tries to make the system fail its task. If the system can still fulfill its goal against an environment trying to prevent it, then we know that the system will always accomplish its objective. Conversely, if the environment can prevent the system from doing that, then we know that in at least one case the system cannot fulfill its task and therefore is not correct.

Specifications, Control, and Synthesis

The different systems we described have a common point in that they have a task that needs to be accomplished. More precisely, among all executions of a system, only a subset of those are desirable executions, in other words, only some of the possible behaviors of the system are deemed correct. A *specification* is a way to describe such a set of correct behaviors. We say that a behavior satisfies the specification if that behavior belongs to the set of correct behaviors, with respect to this specification.

Let us go back to the previous examples of parameterized systems. If we consider

the drone fleet that needs to investigate some terrain, then a possible specification could say that the whole terrain needs to have been explored and that the terrain information needs to have been passed to every drone in the fleet. If we consider a distributed computing system, then maybe the specification ensures that the computing eventually ends and that all participants agree on the correct value.

As the programmers behind every program are human (at least usually), mistakes in the code are unavoidable, and the program written may not work as its authors intended. Such mistakes can have many repercussions, ranging from benign bugs that only make the user lose some time to catastrophic machine failures costing billions in damage cost and potentially endangering human lives. Many engineering methods have been created to mitigate those risks, ranging from code review to unit tests, but in the end relying on the human mind always leave the possibility of missing a mistake. Therefore, the field of *automatic verification* is a thriving field of research whose main goal is to automatically ensure that programs actually do what their authors think they do. See for instance [Esp14] for an overview focusing on closed systems.

Among the many methods used in automatic verification, *model-checking* is a popular method (see e.g., [CJGK+18, CHVB18, BJNT00]) generally defined as follows: given an abstract model M of the system and a formal specification φ, do all executions of M satisfy φ? Usually, the model M is an abstraction of the real system that can be checked exhaustively, which is usually not possible to do on the real system, while the specification φ is often given in some kind of formal logic describing the set of correct behaviors of M.

There are however some drawbacks to model-checking as a tool for verification. First, the users must provide both the model and the specification given as input. While designing specifications is usually not too complex if the formalism is easy to use, creating a model representing a system is usually a hard task. But let us assume that the users have both a model and a specification, and feed them to some model-checking tool. If the tool answers that the model is correct, we are done. However, if the tool answers negatively, then usually a counter-example is given to the users. Then they need to understand why there is an error, tweak the system and the model, then try again with the model-checking tool, and iterate like this until a positive answer is received. The drawbacks are twofold: first this procedure takes a lot of human working time in order to create and then modify the models, and secondly the users are not even sure that they will eventually reach a correct model. Indeed, imagine that after n iterations the model is still not correct. The users have invested a lot of time and resources, but they are still unsure. How can the user know whether he simply needs a few more iterations, or whether the specification cannot actually be satisfied at all whatever the model?

To solve those problems, we turn to *automatic synthesis* of programs. Here, the goal is to automatically generate a program that is correct by construction, with respect to some specification given as input. Then there are two possible outcomes: either a correct program exists and is output, or the specification can not be satisfied and a negative answer is output. This solves both problems of model-checking in that the user only has to provide the specification, and not models of the system, and that there is no need to iterate because either there is a correct program and the synthesis tool will output such a program, or there are none and the user will know that their specification is simply not feasible. The drawback of this approach

is of course that generating a program from scratch is usually harder than simply checking an existing one. To partially alleviate this hard part, we also study some kind of intermediate method called *automatic control*, where a partially defined system is also given as input. This partially defined system is such that at every step of an execution, multiple choices for the next action are available. The goal of this method is to generate a *controller* which dictates what action should be taken in order to satisfy the specification, if such a controller exists.

The synthesis problem was first defined by Church in [Chu57] in the context of circuit synthesis. Remember that we consider open parameterized systems that interact with an uncontrollable environment. The synthesis and control problems are then better described using game theory formalism: two players that we call System and Environment play actions and build an execution while trying to respectively satisfy and falsify the specification. Then the goal is to find a *strategy* for System such that whatever Environment does, all resulting executions satisfy the specification. This strategy, if it exists, can then be translated to a correct program for the system. And if there is no such strategy, then we can negatively answer the problem. This view actually comes from [BL69], which was the first work framing Church's problem into a game formalism. This led to a better understanding of the problem, and was followed by Rabin's works in [Rab72] using tree automata to solve this problem. Since then, the synthesis problem has been extended in many different directions. For instance, [HTWZ15] measures the time between a request and its response to define optimal strategies for reactive systems. Synthesis with a non-discrete amount of time is investigated in [JORW11]. Then many different ways to express specifications have been studied. For example, the synthesis problem has been studied for an extension of LTL with parameterized temporal operators [JTZ18], for register automata and transducers [KK19, EFR19, KMB18], for N-memory automata [BT16], and for the Logic of Repeating Values [FP18a]. Moreover, distributed synthesis for specific communication architectures were studied in, e.g., [PR90], [KV01], and [FS05]. In order to lower the inherent complexity of this problem, underapproximation techniques were developped such as in [QR05] and [LTKR08] in which the number of context switching is bounded, thus limiting the space of possible executions.

To study the synthesis problem on systems with an unbounded number of processes, we naturally turn to words on infinite alphabets, also known as data words. Data words can be used to model executions of such systems, as the infinite alphabet lets us write process identities for any number of different processes. Unlike classical words however, there is no canonical automaton or notion of regular languages. Various approaches have been studied to specify data words languages. Finite-memory automata (also known as register automata) are defined in [KF94]. A register automaton can store the data read in one of its registers (which come in a fixed number), and then later check for equality between the new currently read data value and one of the data stored in a register. A similar view is studied in [DL09] where a freeze quantifier is added to LTL formulas which also allows to check equality between a data value and another deeper in the formula, followed by [DDG12] where a restricted use of the freeze quantifier is suggested which only allows to state that a data value is repeated. In [BDM+11], class automata are defined to deal with set of data words represented by first-order formulas with two variables. Regular expressions for data words are defined in [LTV15], called regular expressions with

memory, which are equivalent to register automata. Two other automata types called alternating variable Büchi word automata and nominal automata are studied in [FGS19] and [SKMW17] respectively.

The scope of this thesis is to study the synthesis and control problems for parameterized systems.

Contributions

To study the control problem, we first need a model for the partially defined systems given as input to the problem. To that end, we define Dynamic Pushdown Systems (shortened as DPS), a model for parameterized systems where each process comes equipped with an unbounded stack structure. This stack structure allows us to model various useful real-life mechanisms, such as unbounded data storing for each process, a recursion stack for recursive function calls, or simply any *last-in-first-out* data structure. Moreover, processes in a DPS share a common finite state called global state. This global state is a way to represent communication between processes via a shared (finite) resource, which can be found for instance in lock mechanisms for parallel computing. Then each process can asynchronously perform transitions depending on its local state and on the shared global state, and update both of them. Finally, new processes can join dynamically during an execution, without any limit on the total number of participants. An acceptance condition is given as target states for both the global state and the state of each process, and serves as a kind of specification for the system. Indeed, these target states represent either a "good" state of the system that we want to reach in acceptable executions, or reversely can be seen as sets of "bad" states that need to be avoided for an execution to be acceptable.

Such systems are highly complex: even the problem of knowing whether there is at least one acceptable execution is undecidable, so the control problem is even more out of question. Therefore, we study a natural restriction on the set of possible executions by only focusing on round-bounded executions, as defined in [LMP10a], which imposes a fixed order on processes performing transitions during the executions. This restriction comes naturally in the context of ring architectures with token passing for instance, where processes are organized in a round-robin fashion and can only execute actions when they have a token, which they then pass to the next process when they are done and so on. When considering this restriction, we prove that the existence of an acceptable execution is PSPACE-complete. Finally, we consider the control problem by splitting global states into System states and Environment states, so that the owner of the current global state decides which process performs which transition in the next step. This allows us to model the uncontrollable part in the system. We show that the control problem is decidable, albeit non-elementary, and that having stacks does not actually make this problem harder (but still let us model more expressive systems).

This forms the content of Chapter 3. Most of these results were published in our ATVA 2018 article [BLS18].

For the synthesis problem, we consider a finite alphabet of actions split into System actions and Environment actions in order to distinguish controllable and uncontrollable events. Unlike in the previous chapter, we consider the parameterized

case for executions: a finite set of processes is fixed before the execution and no other processes can be involved during the execution. To allow for more precise models, we also split that set of processes into System processes, Environment processes, and mixed processes, with the intention that System processes can only perform System actions, Environment processes can only perform Environment actions, and mixed processes can do both types. The motivation behind this distinction is that both players are not necessarily able to act with all processes. For instance, with our drone fleet example from before, maybe only a small part of the drones are equipped with sensors so it would not make sense to let Environment actions be performed on processes without those.

To represent executions of the system, we use data words to model the finite but not bounded in advance number of processes. We then consider specifications given by formulas of first-order logic. We formalize the synthesis game as an asynchronous game where System and Environment do not necessarily strictly alternate, instead Environment may perform actions at any time while System must wait for Environment to let him play. Indeed, if Environment represents an uncontrollable part in the model, then it would be slightly unnatural to enforce a System-Environment-System-... strict alternation. To avoid pathological cases, we add a simple fairness condition. The synthesis problem is about deciding whether there is a winning strategy for System given a finite alphabet and a formula for the specification. This problem is parameterized by the class of formulas used and by the cardinality of the sets of processes. We study this problem for two natural subclasses of first-order logic, and prove for one of these subclasses that the problem is decidable if and only if Environment only controls a bounded number of processes (whereas the set of System processes is still unbounded).

This is the content of Chapter 4, and the results were published in our FoSSaCS 2020 article [BBLS20].

This work is organized as follows. In Chapter 2, a few important notions for control and synthesis are defined. Chapter 3 focuses on the control problem, with the first section introducing the model for parameterized systems, the second introducing the round-bounded restriction, the third section is for results for the control problem, and the fourth studies an alternative restriction. The synthesis problem is studied in Chapter 4, which is defined in the first section, with the following two sections dedicated to two subclasses of first-order logic. Finally, we conclude in Chapter 5.

Chapter 2

Preliminaries

Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{N}_> = \mathbb{N} \setminus \{0\}$. If A is an alphabet (finite or not), then A^* denotes the set of finite words on A, A^ω the set of infinite words on A, and $A^\infty = A^* \cup A^\omega$ is the union of both. If $w = a_0 a_1 \dots a_{n-1}$ is an element of A^* then its size, denoted by $|w|$, is n. If $w \in A^\omega$ then let $|w| = +\infty$. We denote by ε the empty word, that is the only word of size 0. Let $Pos(w)$ be the set of positions of w, i.e. $Pos(w) = \{0, 1, \dots, n\}$ if w is a finite word of size n, and $Pos(w) = \mathbb{N}$ if w is infinite. If $w = a_0 a_1 \cdots \in A^\infty$ is a word and $i \in Pos(w)$, then let $w[i] = a_i$. Finally, we say that $a \in w$ if there is an $i \in Pos(w)$ such that $w[i] = a$.

2.1 Transition Systems

As said in the introduction, we want to study open parameterized systems, that is, systems with an unknown number of processes interacting with an external environment. Naturally, we need a way to model those systems in order to be able to define and solve problems about them. And first, before modeling the system as a whole, we need a way to model what a single process is capable of. Stated in the most general way, a process has an *internal state*, and depending on this state it can execute *actions* which will then update its internal state.

Let us formalize this idea by the notion of *transition systems*. Let Σ be a finite set of action *labels*.

Definition 1. A Σ-*labeled transition system* (short: Σ-LTS) is a tuple $\mathcal{T} = (V, E, v^{init})$ where:

- V is a (finite or infinite) set of *nodes*,
- $E \subseteq V \times \Sigma \times V$ is the *transition relation*, and
- $v^{init} \in V$ is the *initial node*.

Now let $\mathcal{T} = (V, E, v^{init})$ be a transition system, we say that a node v is an a-*successor* of u if $(u, a, v) \in E$, and that v is a successor of u if there is $a \in \Sigma$ such that v is an a-successor of u. We say that a transition system is deterministic if for all $u \in V$ and $a \in \Sigma$, there is at most one $v \in V$ such that v is an a-successor of u. Otherwise, we call the transition system non-deterministic.

A *finite partial run* is a non-empty, finite sequence of nodes $\rho = v_0 v_1 \cdots v_n$ such that for all $0 < i \leq n$, v_i is a successor of v_{i-1}. Infinite partial runs are defined

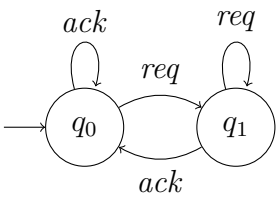

Figure 2.1: Illustration of a request-acknowledgment transition system for a single process.

similarly, except that the sequence of nodes is infinite. A *finite* (resp. *infinite*) *run* is a finite (resp. infinite) partial run starting from v^{init}. Note that a run can be seen as a word on the alphabet V. Let $\mathsf{Runs}(\mathcal{T})$ be the set of all runs of $\mathcal{T}$.

A (labeled) transition system is effectively a way to model the computation of a single process, in the following sense: for a given (finite or not) run $\rho = v_0 v_1 \cdots$, we say that a word $w = a_0 a_1 \dots$ over Σ is *compatible* with ρ if for all $0 \leq i < |w|$, v_{i+1} is an a_i-successor of v_i. Intuitively, a run is the sequence of states a process visits during an execution, while a word compatible with that run can be viewed as a trace of the actions made by the process that ended up giving the run in question. Note that a single run can have multiple compatible words, as a node v can be both an a-successor and an a'-successor of the same node u for two different labels $a \neq a'$.

Transitions systems are used to define semantics of different models. Let us introduce two kinds of such models: finite state machines and pushdown machines.

Definition 2. A *finite state machine* is a tuple $M = (\Sigma, Q, T, q_0)$ where Σ is a finite alphabet, Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, and $T \subseteq Q \times \Sigma \times Q$ is the *transition relation*.

The semantics of a finite state machine is the Σ-LTS $\mathcal{T}_M = (Q, T, q_0)$, so we call a run of M a run of the underlying transition system $\mathcal{T}_M$, and say M is deterministic if $\mathcal{T}_M$ is deterministic.

Finite state machines are a natural way to model simple processes with a bounded memory.

Example 1. Imagine a simple distributed system where an unbounded pool of processes can receive *requests* and later *acknowledge* them. To model this kind of actions, we introduce the alphabet $\Sigma = \{req, ack\}$ where *req* is a label for a request action and *ack* a label for an acknowledgement action.

We define a finite state machine M as follows, illustrated in Figure 2.1: $M = (\Sigma, Q, q_0, T)$ with $Q = \{q_0, q_1\}$, and $T = \{(q_0, req, q_1), (q_1, req, q_1), (q_1, ack, q_0), (q_0, ack, q_0)\}$. Here q_0 is the state of the process when there is no pending request, while q_1 is the state reached when there is one or multiple pending requests. An example of run is $\rho = q_0 q_1 q_0 q_1 q_1 q_0$, with $w = req\ ack\ req\ req\ ack$ being a compatible word.

However, sometimes having only a bounded memory is too restrictive for modeling purposes. For instance in the previous example, one acknowledgment is enough to satisfy any number of requests, whereas one may want to simulate a process that needs to acknowledge each request received individually.

Let us then define another model where this time the semantics is an infinite transition system. This model has access to an unbounded memory, represented

as a *stack*. This is a data structure where symbols from a stack alphabet can be *pushed* to add them into the stack, and later *popped* to be retrieved. However, only the most recently pushed symbol can be popped. This is often referred to as a *Last-In-First-Out* structure.

Let $\mathsf{Act} = \{\mathsf{push}, \mathsf{pop}, \mathsf{int}\}$ be the set of *stack actions*: push when a symbol is pushed onto the stack, pop when a symbol is popped from the stack, and int (for *internal* action) when the stack is not modified.

Definition 3. A *pushdown machine* is a tuple $P = (\Sigma, \Gamma, Q, T, q_0)$ where Σ is a finite alphabet, Γ is a finite stack alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, and $T \subseteq Q \times (\Sigma \times \mathsf{Act} \times \Gamma) \times Q$ is the transition relation.

A configuration of a pushdown machine is a pair $(q, \gamma) \in Q \times \Gamma^*$: q is the current state and γ, a word on Γ, is the current content of the stack with the first (leftmost) letter being the most recently pushed stack symbol.

The semantics of a pushdown machine is a Σ-labeled transition system $\mathcal{T}_P = (V, E, v^{init})$ where $V = Q \times \Gamma^*$ is the set of configurations of P, $v^{init} = (q_0, \varepsilon)$ is the initial configuration, and the transition function E is defined as follows, where $q, q' \in V$, $a \in \Sigma$, $A \in \Gamma$, and $\gamma \in \Gamma^*$:

- $((q, \gamma), a, (q', A\gamma)) \in E$ if there is a transition $(q, (a, \mathsf{push}, A), q') \in T$,
- $((q, A\gamma), a, (q', \gamma)) \in E$ if there is a transition $(q, (a, \mathsf{pop}, A), q') \in T$,
- $((q, \gamma), a, (q', \gamma)) \in E$ if there is a transition $(q, (a, \mathsf{int}, A), q') \in T$.

Note that the stack symbol in the third case (for int transition) does not matter, so we may write $(q, (a, \mathsf{int}, -), q')$ instead for such a transition.

Let us give an example of such a pushdown machine.

Example 2. Let us imagine a slightly more complex $req - ack$ system. Each process first receive a *start* action, followed by any number of *req* actions, and then a *stop* action to indicate that no more request is coming. Then it must do as many *ack* actions as the number of *req* received, and finally execute a *ready* action to indicate that the process is ready to receive new requests.

Let $\Sigma = \{start, req, stop, ack, ready\}$ be the alphabet of actions as described above. We define the pushdown machine $P = (\Sigma, \{Z, R\}, \{q_0, q_1, q_2\}, T, q_0)$ with T illustrated in Figure 2.2. Here Z is a stack symbol used to denote the bottom of the stack, and R is used to count the number of requests. The state q_0 represents a state of the process before a *start* when there are no pending requests, state q_1 means that the process is currently receiving requests, and state q_2 means that the requests are being acknowledged.

An example of run is

$$\rho = (q_0, \varepsilon)(q_1, Z)(q_1, RZ)(q_1, RRZ)(q_2, RRZ)(q_2, RZ)(q_2, Z)(q_0, \varepsilon)(q_1, Z)(q_1, RZ)$$

ending in state q_1 with one pending request in the stack. A compatible word for this run is $w = start\ req\ req\ stop\ ack\ ack\ ready\ start\ req \in \Sigma^*$.

Recall that our overall goal is to automatically generate open parameterized systems that always produce correct behaviors with respect to a given objective, the *specification*. Thus we need to model how the system operates as a whole. In the next two sections, we study two different approaches:

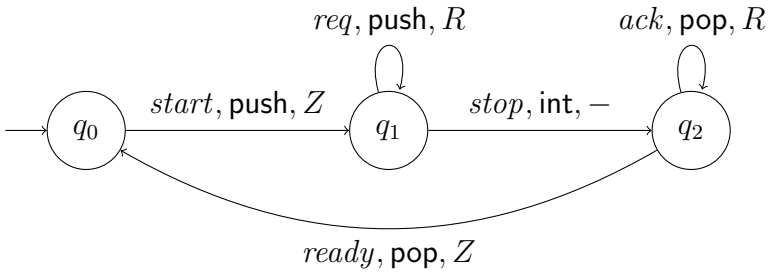

Figure 2.2: Req-ack system with a pushdown machine.

(i) when the system is already partially defined, and the goal is to generate a controller that chooses at each point what action to make among those available, or

(ii) when the system is not defined at all, and all actions are possible at every moment.

In both cases, the goal is to know whether it is possible to find and if it is, to generate a program that dictates what actions the system should do depending on what actions have been executed so far. In the first case we talk about *control*, while in the second case we call this *synthesis*.

2.2 Control

In this section we consider the first case, where a system that is already partially defined is given as an input and we want to control it in order to get correct executions. First, let us define how we model such a system.

We introduce dynamic concurrent transition systems (DCTS), which model concurrent systems with an unbounded number of identical processes. Each process is represented as a *process* transition system, and can execute actions. Those actions are synchronized with a *global* transition system, which models a ressource shared by all processes. Finally, the system can spawn new processes dynamically during the execution without any bound. While all processes execute a copy of a transition system, they evolve independently after being spawned.

2.2.1 Dynamic Concurrent Transition Systems

Let us now formalize this definition.

Definition 4. A *dynamic concurrent transition system* (short: DCTS) is a tuple $\mathcal{C} = (\mathcal{T}_{\mathsf{glob}}, \mathcal{T}_{\mathsf{loc}})$ where

- $\mathcal{T}_{\mathsf{glob}} = (V_{\mathsf{glob}}, E_{\mathsf{glob}}, v_{\mathsf{glob}}^{init})$ is the *global* Σ-LTS, and
- $\mathcal{T}_{\mathsf{loc}} = (V_{\mathsf{loc}}, E_{\mathsf{loc}}, v_{\mathsf{loc}}^{init})$ is the *local* (or *process*) Σ-LTS.

The semantics of this DCTS is a Σ-labeled transition system $\mathcal{T}_{\mathcal{C}} = (V, E, v^{init})$ defined hereafter.

First, V is the set of *configurations*, which will also be denoted by $\mathsf{Conf}(\mathcal{C})$: a configuration is a tuple $c = (v_g, v_1, \ldots, v_n)$ with $n \in \mathbb{N}$ such that $v_g \in V_{\mathsf{glob}}$ and for

all $i \in \mathbb{N}_>$ we have $v_i \in V_{\mathsf{loc}}$. Let the size of a configuration, denoted by $|c|$, be the number of process nodes, which is n for the configuration above. The *initial configuration* is $c^{init} = (v_{\mathsf{glob}}^{init})$, a configuration of size 0.

If a node of a transition system represents the state of a process, then one can see a configuration as the state of the whole concurrent system, with v_g being a global state shared by all processes while each v_i represents the state of process i. Note that while the number of processes that can appear in a configuration (that is, the size of the configuration) is not bounded, it is always finite for a given configuration. We refer to processes that are in the configuration as *active* processes.

Then, the transition relation between configurations E is defined as follows. Let c and c' be two configurations, $a \in \Sigma$ and $i \in \mathbb{N}_>$. We say that c' is an (a, i)-successor of c if one of the following holds:

1.
   - $c = (u_g, v_1, \ldots, v_{i-1}, u_i, v_{i+1}, \ldots, v_n)$,
   - $c' = (v_g, v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_n)$,
   - v_g and v_i are a-successors of u_g and u_i in $\mathcal{T}_{\mathsf{glob}}$ and $\mathcal{T}_{\mathsf{loc}}$ respectively.
2.
   - $c = (u_g, v_1, \ldots, v_{i-1})$,
   - $c' = (v_g, v_1, \ldots, v_{i-1}, v_i)$,
   - v_g and v_i are a-successors of u_g and v_{loc}^{init} in $\mathcal{T}_{\mathsf{glob}}$ and $\mathcal{T}_{\mathsf{loc}}$ respectively.

The first kind of successors indicates that the (already active) process i made a transition with label a, and changed the global state of the system accordingly. The second kind instead shows how a process is activated, starting with a transition of label a from its initial node, while updating the global state as well.

We call runs of $\mathcal{C}$ runs of the underlying transition system $\mathcal{T}_{\mathcal{C}}$, which are sequences of configurations. By abuse of notation we will denote by $\mathsf{Runs}(\mathcal{C})$ the set of runs of $\mathcal{C}$ instead of $\mathsf{Runs}(\mathcal{T}_{\mathcal{C}})$. For any run $\rho = c_0 c_1 \cdots \in \mathsf{Runs}(\mathcal{C})$, we let $\mathsf{Act}(\rho) = \{p \in \mathbb{N}_> \mid \exists i < |\rho| \text{ such that } c_i \text{ is of size } p\}$ be the set of active processes of ρ.

As said earlier, DCTS are models for parameterized systems, and therefore $\mathsf{Runs}(\mathcal{C})$ is the set of all possible executions of such a system. But maybe not all of those executions are *acceptable* in the sense of what the system is supposed to do. Remember that we want to obtain only executions that are correct with respect to a given objective, the specification. Therefore we have to further restrict runs so that only those correct runs can be generated.

Here a specification will be given as an *acceptance condition*: let $\mathcal{C}$ be a DCTS, then an acceptance condition for $\mathcal{C}$ is a set $\mathsf{Acc} \subseteq \mathsf{Conf}(\mathcal{C})^\infty$. A run is said to be *accepting* if it is in Acc. In other words, accepting runs are the runs that satisfy the specification.

An acceptance condition can be as general as wanted, but let us define a few specific conditions that will be used in this thesis. First, let $\mathcal{F} = (\mathcal{F}_{\mathsf{glob}}, \mathcal{F}_{\mathsf{loc}})$ where $\mathcal{F}_{\mathsf{glob}} \subseteq V_{\mathsf{glob}}$ and $\mathcal{F}_{\mathsf{loc}} \subseteq V_{\mathsf{loc}}$ are two sets of accepting nodes for the global and process transition systems. We call $\mathcal{F}$ an *accepting profile*. Intuitively, an accepting node represents a state of a process that must be reached.

Now we say that a configuration is accepting if all its component nodes are accepting, i.e., $c = (v_g, v_1, \ldots, v_n)$ is accepting if and only if $v_g \in \mathcal{F}_{\mathsf{glob}}$ and $v_i \in \mathcal{F}_{\mathsf{loc}}$ for all $i \in \{1, \ldots, n\}$. We can then define three different acceptance conditions as follows:

- $\mathsf{Reach}(\mathcal{F}) = \{c_0c_1\cdots \mid \exists i \in \mathbb{N} \text{ such that } c_i \text{ is accepting}\}$
- $\mathsf{Büchi}(\mathcal{F}) = \{c_0c_1\cdots \mid \text{there are infinitely many } i \in \mathbb{N} \text{ such that } c_i \text{ is accepting}\}$
- $\mathsf{coBüchi}(\mathcal{F}) = \{c_0c_1\cdots \mid \text{there are finitely many } i \in \mathbb{N} \text{ such that } c_i \text{ is accepting}\}$

Obviously the latter two conditions only make sense for infinite runs, otherwise they are trivially false and true respectively.

Intuitively, the Reach condition simply states that at some point during the run all processes were in an accepting state, and we do not really care what happens afterwards, while the Büchi condition asks for this to happen infinitely often. The third condition might seem strange as it requires to completely avoid accepting states after some point, but it may be easier to understand if we instead consider accepting nodes as states that the system needs to avoid, i.e., "bad" states for the processes. Then this condition simply states that after a finite prefix we never encounter such a bad configuration, which is often referred to as a *safety* condition.

Sometimes it makes more sense to look at each process individually. For instance the specification could be that each process eventually reaches an accepting state, but not necessarily that *all* processes reach it at the same time as is required by Reach. Let us define this variation. We say that a configuration $c = (v_g, v_1, \ldots, v_n)$ is accepting for process $p \in \{1, \ldots, n\}$ if $v_p \in \mathcal{F}_{\mathsf{loc}}$. Then we define the *independent* variations for the three acceptance conditions given above:

- $\mathsf{IndReach}(\mathcal{F}) = \{\rho = c_0c_1\cdots \mid \forall p \in \mathsf{Act}(\rho), \text{ there is } i < |\rho| \text{ such that } c_i \text{ is accepting for } p\}$
- $\mathsf{IndBüchi}(\mathcal{F}) = \{\rho = c_0c_1\cdots \mid \forall p \in \mathsf{Act}(\rho), \text{ there are infinitely many } i \in \mathbb{N} \text{ such that } c_i \text{ is accepting for } p\}$
- $\mathsf{IndCoBüchi}(\mathcal{F}) = \{\rho = c_0c_1\cdots \mid \forall p \in \mathsf{Act}(\rho) \text{ there are finitely many } i \in \mathbb{N} \text{ such that } c_i \text{ is accepting for } p\}$

Example 3. Let us revisit again the $req - ack$ example mentioned in Example 1, where processes receive requests and later acknowledge them. We have seen in that example how to model a single process with a transition system $\mathcal{T}_M$ derived from a finite state machine M. Now using a DCTS we can model a parameterized system where an unbounded number of processes can receive requests. For instance, we can define $\mathcal{C} = (\mathcal{T}_{\mathsf{glob}}, \mathcal{T}_M)$ with $\mathcal{T}_M$ as defined in Example 1 and $\mathcal{T}_{\mathsf{glob}}$ a global transition system modeling that *req* and *ack* actions alternate which is defined as

$$\mathcal{T}_{\mathsf{glob}} = (\{v_{ack}, v_{req}\}, \{(v_{ack}, req, v_{req}), (v_{req}, ack, v_{ack})\}, v_{ack})$$

Then we let $\mathcal{F} = (\{v_{ack}, v_{req}\}, \{v_0\})$ and we take $\mathsf{Acc} = \mathsf{IndBüchi}(\mathcal{F})$ as an acceptance condition, which in other words means that every process has no pending request infinitely often. This ensures that in any accepting run, all requests are eventually acknowledged. See Figure 2.3 for an illustration.

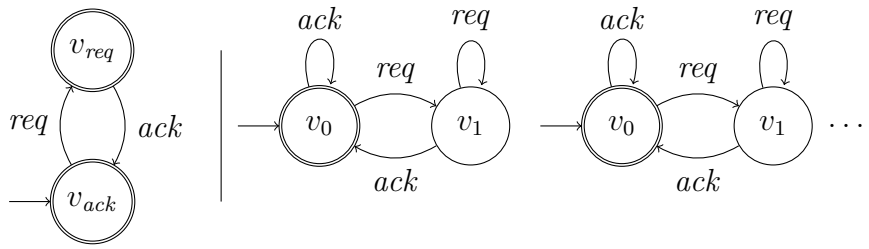

Figure 2.3: Illustration of a DCTS for a $req - ack$ specification.

2.2.2 Control Problem

With a parameterized system given as a DCTS, what does "generating a program that produces correct behaviors" means? Essentially, it means producing a function that takes as input a configuration of the DCTS and outputs a successor configuration (or equivalently, an action a and the identifier i of a process such that there is an (a, i)-successor configuration). Furthermore, following the recommendations of this function always produces an accepting run with respect to a given acceptance condition.

Such a function is called a *controller*, as it describes how to control actions of the system in order to obtain good behaviors. For instance, with the DCTS given in the example above, a simple way to produce an accepting run is by a controller that forces the first process to do a *req* then an *ack* action and then does the same again forever.

However, we have not yet taken into account that the systems we model are open, and that processes interact with an external actor, which we will call *Environment*, that cannot be controlled by the system. This means that there must be a part inside our model in which a controller cannot dictate actions, and where Environment can choose what actions to do. But then we cannot expect Environment to follow a controller as described above that always produce correct behaviors. Actually, it is conceptually easier to assume that Environment is an antagonistic actor that will try to make the system produce bad behaviors. Then the problem becomes finding a controller that will choose *controllable* actions and that will always produce accepting runs *whatever* actions are made by Environment.

Example 4. Using the $req - ack$ system from the DCTS of Example 3, suppose that first Environment may do a *req* action on some process, then the system can execute an *ack* action on a process, and so on.

Then there is an easy way to build a controller that always gets an accepting run: the controller only needs to follow each *req* by Environment by doing an *ack* immediately on the *same* process. Formally, the controller is a function f such that for any configuration $c = (v_{req}, u_1, \ldots, u_n)$ where exactly one of the u_i is v_1 and all others are v_0, then $f(c) = (ack, i)$.

Note that as we do not control actions by Environment, we cannot predict which process will get a request and therefore the controller actually reacts to what Environment does, instead of blindly following a single run.

It is important to note that the existence of an accepting behavior does not imply the existence of a controller, as shown in the following example:

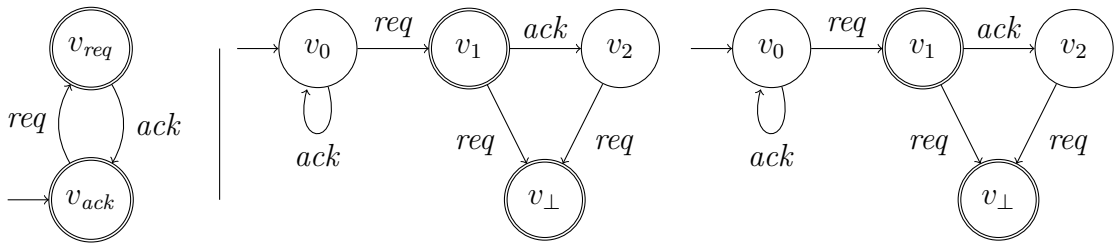

Figure 2.4: Another $req - ack$ specification.

Example 5. Let us imagine another $req - ack$ specification where this time, a process can only receive at most one request, and this request must still be acknowledged, otherwise the system fails. As before, Environment does the requests and the controller decides the acknowledgements alternatively. See Figure 2.4 for an illustration of this new system. Here $v_\perp$ represents a failing state. The acceptance condition is $\mathsf{coB\ddot{u}chi}(\{v_{ack}, v_{req}\}, \{v_1, v_\perp\})$. In other words, a run is accepting iff no process ever reaches $v_\perp$, as it is not possible to exit this state, and if all *req* are eventually followed by an *ack*.

Accepting runs for this DCTS obviously exist: for instance, the infinite run of the form $req - ack$ on process 1, then $req - ack$ on process 2, and so on is accepting. However, the Environment could also perform two *req* actions on the same process, and then whatever the continuation of the execution is then the run will not be accepting. Since the Environment is by definition uncontrollable, no controller can prevent this from happening, and therefore there is no controller satisfying this specification.

With these informal examples in mind, we now formalize what we mean by the interactions between Environment and the system, controllers, and the control problem. To that end, we introduce the notion of two-player games on transition systems. From now on, we shall call System the entity that can execute controllable actions, as opposed to Environment. Those two entities will be the *players* in the games.

Definition 5. A *game* is a tuple $\mathcal{G} = (\mathcal{T}, V_\mathsf{s}, V_\mathsf{e}, \mathsf{Acc})$ where $\mathcal{T} = (V, E, v^{init})$ is a transition system, $V_\mathsf{s} \uplus V_\mathsf{e} = V$ is a partition of the nodes of $\mathcal{T}$ into *system nodes* and *environment nodes* respectively, and $\mathsf{Acc} \subseteq V^\infty$ is an acceptance condition.

A *play* ρ of $\mathcal{G}$ is simply a run of the underlying TS $\mathcal{T}$. Intuitively the partition of V is used to determine whether it is System's or Environment's turn to play: if the current node belongs to System then the next successor is chosen by System, otherwise it is chosen by Environment. Note that the two players are not necessarily alternating turns, e.g., a successor of a system node may still be a system node which would allow System to have multiple consecutive turns. We say that a play is *maximal* if either it is infinite, or it ends in a node with no successor.

A *strategy* for a player $\mathsf{pl} \in \{\mathsf{s}, \mathsf{e}\}$ is a partial mapping $\sigma_\mathsf{pl} : V^* V_\mathsf{pl} \to V$ such that for all $\rho = v_0 \dots v_n$ with $v_n \in V_\mathsf{pl}$ we have that $\sigma_\mathsf{pl}(\rho)$, if defined, is a successor of v_n. Intuitively, a strategy is simply a function that represents what a player will choose whenever a choice is needed. Furthermore, a strategy must be *non-blocking*: if σ_pl is a strategy for pl, ρ is a run ending in $v \in V_\mathsf{pl}$, and v has at least one successor, then $\sigma_\mathsf{pl}(\rho)$ must necessarily be defined. We say that a play is compatible with a

strategy of player pl if the choices made by pl during the play are consistent with the strategy. Formally, a play $\rho = v_0 v_1 \dots$ is compatible with strategy σ_{pl} of player pl if the following condition is satisfied: for all $0 < i < |\rho|$ such that $v_{i-1} \in V_{\mathsf{pl}}$, we have that $v_i = \sigma_{\mathsf{pl}}(v_0 \dots v_{i-1})$. Note that if we have a strategy for both System and Environment, then there is only one play that is compatible with both strategies.

A play is *winning* for System if it belongs to the acceptance condition Acc. We say that a strategy σ_{s} for System is *winning* if all plays compatible with this strategy are winning. In other words, σ_{s} is winning if for all strategies σ_{e} for Environment, the play compatible with σ_{s} and σ_{e} is winning. Finally, we say the game is winning for System if there is a winning strategy for System, otherwise Environment is winning.

A game is said to be *determined* if either System has a winning strategy, or Environment has a winning strategy. Furthermore, we say that σ_{pl} is *memoryless* if, for all $w, w' \in V^*$ and $v \in V_{\mathsf{pl}}$, we have $\sigma_{\mathsf{pl}}(wv) = \sigma_{\mathsf{pl}}(w'v)$, i.e., the strategy only depends on the last node. It is known that games with a reachability, Büchi, or coBüchi acceptance condition are determined and that if there is a winning strategy for a player, then there is one that is winning and memoryless [EJ91, Zie98].

A strategy for System in a game can be viewed as a controller, as it describes exactly what action to take (i.e., which successor to choose) depending on the current situation and the past of the execution (only the current situation for a memoryless strategy). Therefore, our goal of generating a controller for a system that always produces good behaviors can be translated as generating a winning strategy for System in the corresponding game. We can thus define the control problem in general as follows:

CONTROL	
Input:	A DCTS $\mathcal{C}$, a partition $C_{\mathsf{s}} \uplus C_{\mathsf{e}}$ of $\mathsf{Conf}(\mathcal{C})$, $\mathsf{Acc} \subseteq \mathsf{Conf}(\mathcal{C})^\infty$
Question:	Is there a winning strategy for System in $\mathcal{G} = (\mathcal{T}_\mathcal{C}, C_{\mathsf{s}}, C_{\mathsf{e}}, \mathsf{Acc})$?

Note that we need a finite representation for all inputs, as they can be objects with an infinite size. Therefore we will restrict ourselves to transitions systems defined by finite state and pushdown machines, and acceptance conditions that can be described in a finite manner, e.g., Reach, $\mathsf{coBüchi}$, $\mathsf{Büchi}$, or their independent variations.

Example 6. Let us formalize the simple $req - ack$ game defined in Example 4, where alternatively Environment does req actions and System must do corresponding ack actions. The DCTS $\mathcal{C}$ is the one used in Example 3 and illustrated in Figure 2.3. The partition of configurations into C_{s} and C_{e} is as follows: a configuration $c = (v_g, v_1, \dots)$ belongs to System if $v_g = v_{req}$, and to Environment if $v_g = v_{ack}$. The acceptance condition is, as in Example 3, $\mathsf{Acc} = \mathsf{IndBüchi}(\mathcal{F})$ with $\mathcal{F} = (\{v_{ack}, v_{req}\}, \{v_0\})$, which essentially states: for each process each req must eventually be followed by an ack. The game is then defined as $\mathcal{G} = (\mathcal{T}_\mathcal{C}, C_{\mathsf{s}}, C_{\mathsf{e}}, \mathsf{Acc})$.

The controller described in Example 4, which answers immediately to every req by performing an ack on the same process, is equivalent to the strategy σ_{s} defined as follows:

$$\sigma_{\mathsf{s}}(c_0 \cdots c_n \cdot (v_{req}, u_1, \dots, u_k)) = (v_{ack}, v_1, \dots, v_k)$$

such that, with $i = min\{j \mid u_j = v_1\}$, we have that $v_i = v_0$ and $v_j = u_j$ for all $j \neq i$. In other words, this strategy performs an ack on the first process with a pending request.

An example of winning play that is compatible with σ_s is the following play:

$$\rho = (v_{ack}) \cdot (v_{req}, v_1) \cdot (v_{ack}, v_0) \cdot (v_{req}, v_0, v_1) \cdot (v_{ack}, v_0, v_0) \cdot (v_{req}, v_0, v_0, v_1) \cdot \dots$$

where Environment plays a *req* on a fresh process each time. Another example is the play $\rho' = (v_{ack}) \cdot ((v_{req}, v_1) \cdot (v_{ack}, v_0))^\omega$ where Environment only plays *req* on the first process. It is easy to see that the strategy σ_s is winning, and therefore the answer to the control problem for this game is positive.

2.3 Synthesis

We have seen in the last section how to define the control problem. In that section, a partial system was given as an input to restrain the possible choices available at each step of the execution. Then the control problem was about generating a controller that will choose which available action to make at every moment.

But sometimes the user does not have such a system at hands. Another goal in this context is to try to automatically generate a correct system from scratch, with just the possible actions and the specification as an input. Then there are two possible cases: either this attempt fails, and then the user knows that it is pointless to even attempt to build a system with that specification. Or, it succeeds and then the user gets a system that is by definition correct. We call this the synthesis problem. Actually, the synthesis problem can be seen as a control problem where the input system allows any action to be executed at any moment.

In this context, there is no proper state of the system as we had with the control problem. Therefore, we only care about the trace of the execution that has been performed in order to see if the specification is satisfied. To represent this execution trace, we will use data words.

2.3.1 Data Words

Data words are a very useful model to reason with sequences of objects that cannot be captured with only a finite set [Seg06]. Let us fix a finite alphabet Σ of *actions*, and an infinite alphabet $\mathcal{D}$ of *data values*.

Definition 6. A *data word* is an element of $(\Sigma \times \mathcal{D})^\infty$, i.e., a word $w = (a_0, d_0)(a_1, d_1) \cdots$ where $a_0, a_1, \dots$ are actions in Σ and $d_0, d_1, \dots$ are data values in $\mathcal{D}$. Data words are finite if they belong to $(\Sigma \times \mathcal{D})^*$, or infinite if they are in $(\Sigma \times \mathcal{D})^\omega$.

Note that a data value could represent an integer, a string of characters, any measurement that needs real numbers, and so on. Here in the context of this thesis, a pair $(a, d) \in \Sigma \times \mathcal{D}$ models that action a was executed by the process with identifier d. Usually, process identifiers are integers, and as such the set $\mathcal{D}$ will usually be $\mathbb{N}$ or $\mathbb{N}_>$.

Example 7. Using our already presented example with requests and acknowledgments, let us take $\Sigma = \{req, ack\}$ and $\mathcal{D} = \mathbb{N}$, and let w be the data word $(req, 0)(req, 1)(ack, 1)(req, 2)(ack, 0)(ack, 2)$. The data word w models an execution where the process with id 0 receives a request, then process 1 also receives a request, which it acknowledges immediately, then process 2 receives a request, process 0 acknowledges the earlier request, and finally process 2 also acknowledges its own request.

Example 8. Now let us imagine a system where a process *receives* a token, then *sends* it to the next process, which then sends it to the next one and so on. Let $\Sigma = \{rcv, send\}$ and $\mathcal{D} = \mathbb{N}$, then a data word modeling such an execution would be the infinite data word $w = (rcv, 0)(send, 0)(rcv, 1)(send, 1)(rcv, 2)(send, 2)\dots$

If data words are used to model executions of concurrent systems, then a specification denotes which one are *correct* executions. Therefore, a specification can be seen as the set of data words that correspond to correct executions, hence a specification S is a set of data words, i.e., $S \subseteq (\Sigma \times \mathcal{D})^\infty$. We say that a data word w *satisfies* a specification S if $w \in S$.

Example 9. Going back to the system described in Example 7, one could want that all requests are eventually acknowledged. In terms of data words, this translates as the set $S_1 = \{w \mid \text{every } (req, i) \text{ in } w \text{ is eventually followed by a } (ack, i)\}$. Then the data word w given in Example 7 satisfies S_1, while $w' = (req, 0)(req, 1)(ack, 0)$ does not.

Example 10. With the system from Example 8, one possible specification would be that every process eventually receives the token, i.e., $S_2 = \{w \mid \forall i \in \mathbb{N}, (rcv, i) \in w\}$, which is satisfied by the data word given in the example. Note that this specification alone does not ensure that the informal description of the system that we gave, i.e., that a process can only send the token to the next one, is actually followed: the data word $w' = (rcv, 1)(rcv, 0)(rcv, 3)(rcv, 2)(rcv, 5)(rcv, 4)\dots$ also satisfies S_2, but there are no *send* actions and the order in which processes receive the token is not correct, which should not be possible within the system described earlier.

Now a specification can be a finite or infinite set, and so one needs a finite way of describing such a set. To that end, we introduce first-order logic in the next section.

2.3.2 First-Order Logic

Let us first introduce first-order logic on words (over a finite alphabet) before discussing its extension to data words.

First-order logic on words. First-order (FO) logic is a powerful logic that has been extensively studied on finite words over a finite alphabet [MP71]. In that logic, variables are used to quantify on positions of a word. Moreover, different predicates can be used to check various properties of the positions represented by variables. First, there is a unary predicate for each letter of the (finite) alphabet to check the letter at a position. Then there are binary predicates to compare two positions: one can check that two positions are the same, that a position occurs later in the word than another, or that a position is the direct successor of another. Then formulas of FO logic are built from those predicates as well as the usual logic connectors: negation, disjunction, conjunction, and existential and universal quantifiers for variables.

Fix Σ an alphabet and $\mathcal{V}$ a set of *variables*. The formal grammar is given as follows:

$$\varphi ::= a(x) \mid x = y \mid \mathrm{succ}(x, y) \mid x < y \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi$$

with $x, y \in \mathcal{V}$ and $a \in \Sigma$. To evaluate a formula, variables will be interpreted as positions of a data word. A *valuation* is a partial function $\nu : \mathcal{V} \to \mathbb{N}$. If ν is a valuation, $x \in \mathcal{V}$ and $n \in \mathbb{N}$, then $\nu[x \leftarrow n]$ is a valuation defined as:

$$\nu[x \leftarrow n](y) = \begin{cases} n & \text{if } y = x, \\ \nu(y) & \text{otherwise.} \end{cases}$$

Given a word w of size n and a valuation ν such that $\forall x \in \mathcal{V}, \nu(x) \in \{0, \dots, n-1\}$, we say that w and ν satisfy a formula φ, denoted as $(w, \nu) \models \varphi$, if $((w, \nu), \varphi)$ belongs to the satisfaction relation defined inductively as follows:

- $(w, \nu) \models a(x)$ if $w[\nu(x)] = a$,
- $(w, \nu) \models x = y$ if $\nu(x) = \nu(y)$,
- $(w, \nu) \models \mathrm{succ}(x, y)$ if $\nu(y) = \nu(x) + 1$,
- $(w, \nu) \models x < y$ if $\nu(x) < \nu(y)$,
- $(w, \nu) \models \neg\varphi$ if (w, ν) does not satisfy φ,
- $(w, \nu) \models \varphi \vee \varphi'$ if $(w, \nu) \models \varphi$ or $(w, \nu) \models \varphi'$,
- $(w, \nu) \models \exists x.\varphi$ if there is $n \in \mathbb{N}$ such that $(w, \nu[x \leftarrow n]) \models \varphi$.

Finally we say that w satisfies φ, noted $w \models \varphi$, if $(w, \{\}) \models \varphi$ where $\{\}$ denotes the empty valuation.

More intuitively, $a(x)$ is satisfied if there is an a at position x in w, $x = y$ means that x and y both refer to the same position, $\mathrm{succ}(x, y)$ means that y is the position immediately following x in w, $x < y$ means that y is a position somewhere after x in w, and negation, disjunction, and existential quantification act as expected.

Usual notations are defined by: $\varphi \wedge \varphi' \equiv \neg(\neg\varphi \vee \neg\varphi')$, $\varphi \Rightarrow \varphi' \equiv \neg\varphi \vee \varphi'$, $\forall x.\varphi \equiv \neg\exists x.\neg\varphi$, $\top \equiv a(x) \vee \neg a(x)$, $\bot = \neg\top$. Moreover, given a formula $\varphi(x_1, \dots, x_n, y)$, we use $\exists^{\geq m} y.\varphi(x_1, \dots, x_n, y)$ as an abbreviation for

$$\exists y_1 \dots \exists y_m. \left(\bigwedge_{1 \leq i < j \leq m} \neg(y_i = y_j) \wedge \bigwedge_{1 \leq i \leq m} \varphi(x_1, \dots, x_n, y_i) \right)$$

if $m > 0$, and $\exists^{\geq 0} y.\phi(x_1, \dots, x_n, y) = \top$. This abbreviation expresses that there are *at least* m different positions $y_1, \dots, y_m$ that verify φ. Similarly we also use $\exists^{=m} y.\varphi$ as an abbreviation for $(\exists^{\geq m} y.\varphi) \wedge (\neg\exists^{\geq m+1} y.\varphi)$ to express that there are *exactly* m such positions.

A FO formula induces a language, which is the set of words that satisfy the formula.

Example 11. For instance, the formula:

$$\varphi = \forall x.\, [a(x) \Rightarrow (\exists y.b(y) \wedge \mathrm{succ}(x, y) \wedge (\exists z.c(z) \wedge y < z))]$$

expresses that every a in the word is immediately followed by a b, which is followed by a c later in the word.

Notably, the set of languages that can be defined by FO formulas is exactly the set of *star-free* languages, that is languages which can be defined with a regular expression with complement and without the Kleene star $*$ [PP86]. Computing whether there is a word satisfying a given first-order formula, which is known as the satisfiability problem, has been shown to be non-elementary [Sto74]. Therefore, various restrictions for FO logic have been studied, most notably by restricting the set of variables to a finite amount. See e.g. [DGK08] for a survey. Extensions of the logic have also been proposed, for instance to reason about infinite words [PP04]. Let us now focus on an extension to data words.

Extension to data words. First-order logic has been extended to data words in [NSV04] by introducing a binary predicate $\sim$ that checks whether two positions have the same data value.

Formally, the new grammar is given as follows:

$$\varphi ::= a(x) \mid x = y \mid x \sim y \mid \mathrm{succ}(x, y) \mid x < y \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi$$

The satisfaction relation is modified as follows:

- The rule $(w, \nu) \models a(x)$ if $w[\nu(x)] = a$ now becomes $(w, \nu) \models a(x)$ if $w[\nu(x)] = (a, d)$ for some $d \in \mathcal{D}$, and
- A new rule is added: $(w, \nu) \models x \sim y$ if $w[\nu(x)] = (a, d)$ and $w[\nu(y)] = (a', d)$ for some $a, a' \in \Sigma$ and $d \in \mathcal{D}$.

The first change simply reflects that we now evaluate formulas on data words and not words, while the new rule describes the semantics of predicate $\sim$, which is exactly what was explained earlier. The rest is defined as before, and we keep the usual notations such as $\wedge$, $\Rightarrow$ and so on that were already defined.

The specification induced by a formula φ is the set $S_\varphi = \{w \mid w \models \varphi\}$ of data words that satisfy the formula.

Example 12. The specification described in Example 9, which is that each *req* on a process is later acknowledged by the same process, is naturally given by the formula using the shortcuts defined above: $\varphi_1 = \forall x.\,(req(x) \Rightarrow \exists y.(y > x \wedge y \sim x \wedge ack(y)))$, i.e., in plain English "*for every position x in the data word, if the action is a req, then there must be a position y which is later in the word, has the same data value, and has action ack.*"

Note that with the logic defined as it is, one can only compare two data values with respect to equality: either two positions have the same data value or not. So even if the set $\mathcal{D}$ of data values comes equipped with, say, a total order as is the case with $\mathbb{N}$, it is not possible to use it in formulas. Therefore properties such as "*the data word has its data values sorted*" cannot be expressed.

Let us now define a few useful subclasses for this logic.

Subclasses of FO. One notable subclass of FO formulas are two-variable formulas: formulas that use at most two different variable symbols, although they can be used (and reused) any number of times. Formally, the set of variables is now the finite set $\mathcal{V} = \{x, y\}$, and the rest of the definition is unchanged. For instance,

the specification "*there are three different data values in the word*" can easily be expressed with a formula using three variables:

$$\exists x \exists y \exists z. \, (\neg(x \sim y) \wedge \neg(y \sim z) \wedge \neg(x \sim z))$$

but cannot be expressed with two or less variable symbols. However, two-variable logic is not only limited to counting up to two, as two variables are enough to express that "*there are at least three a"s*" by using predicate $<$ and reusing a variable name:

$$\exists x. \, [a(x) \wedge (\exists y. \, [x < y \wedge a(y) \wedge (\exists x. y < x \wedge a(x))])]$$

Let us denote this two-variable logic by FO^2.

Moreover, one can restrict the expressive power of FO by allowing only some of the predicates between $\sim$, succ, and $<$. Let us denote by $\mathrm{FO} = \mathrm{FO}[\sim, \mathrm{succ}, <]$ the full logic with all three predicates allowed, $\mathrm{FO}[\sim]$ the logic where $\sim$ is allowed but succ and $<$ cannot be used, and so on. This can be combined with the two variable restrictions, e.g. $\mathrm{FO}^2[\sim]$ formulas can use $\sim$ and at most two variables. Note that $\mathrm{FO}[\sim, <]$ is equivalent to FO as succ can be simulated using $<$:

$$\mathrm{succ}(x, y) \equiv x < y \wedge \neg \exists z. (x < z \wedge z < y)$$

but this is not true anymore when restricted to two variables. Moreover, not including $\sim$ means that the data value part of the data word cannot be accessed, in other words the formula can only state properties over the finite part of the data word.

2.3.3 Synthesis Problem

Recall that our overall goal is to automatically generate correct programs for open parameterized systems, in other words programs that will respect the given specification. Therefore, at the very least one needs to ask if there is *at least one* execution that satisfies this specification. This is called the non-emptiness or satisfiability problem, which is defined as follows:

SATISFIABILITY	
Input:	A first-order formula φ of $\mathrm{FO}[\sim, \mathrm{succ}, <]$
Question:	$S_\varphi \neq \emptyset$?

For example, the specification φ_1 given in Example 12 is satisfiable, with the empty word being a satisfying word. The satisfiability problem can also be defined for subclasses of FO as defined earlier. For instance, it has been shown that the satifisfiablity problem for the full logic is undecidable (even restricted to three variable names), whereas FO^2 is actually decidable (but with a non-elementary complexity). Refer to [BMS+06] for results on this question.

If there is no data word satisfying φ, then we are certain that no program can satisfy the specification. This happens when the specification is too constrained and asks for something impossible, e.g., a FO formula expressing that "a data word must contain at least three a's but no more than one a."

However, as we want to model open systems, satisfiability of a specification is not enough for our purposes. To take the environment into account, we will

fix the following assumptions. First, the alphabet of actions Σ is partitioned into *system actions* and *environment actions*. We call the set of those actions Σ_{s} and Σ_{e} respectively. Second, the environment may execute any of its actions at any time, whereas the system can execute actions only when the environment allows it. However if the environment never lets the system execute actions, then almost no specification could be satisfied. Therefore we only consider *fair* runs, where the environment can never totally prevent the system from playing.

Then the synthesis problem is about deciding the existence of a program that dictates actions the system should play such that whatever the environment does, the execution will satisfy the specification. The input of this problem is the specification as well as the sets of system and environment actions, and the output is a program as described above if one exists.

Example 13. Using the $req - ack$ specification from Example 12, let $\Sigma_{\mathsf{s}} = \{ack\}$ and $\Sigma_{\mathsf{e}} = \{req\}$, that is requests are sent by the environment and the system must acknowledge them. Then there is a way for the system to satisfy the specification φ_1: starting with an empty queue, whenever the environment does a *req* action on a process i, add i to the queue. Then when it is possible to execute an action, dequeue the first process in the queue and do an *ack* on this process. This program ensures that every *req* is eventually followed by an *ack* on the same process, as long as the fairness condition ensures that the system can execute one of its actions infinitely often. Note that there are other ways to satisfy the specification.

Example 14. Now let us slightly modify φ_1 to get another specification: let

$$\varphi_2 = \forall x.\, [req(x) \Rightarrow \exists y.(\mathrm{succ}(x, y) \wedge y \sim x \wedge ack(y))]$$

This specification requires that every *req* is *immediately* followed by an *ack* on the same process. Then φ_2 is trivially satisfied by the empty data word, or any data word of the form $(req, i_1)(ack, i_1)(req, i_2)(ack, i_2)\ldots$, but there is no program that will satisfy the specification because the environment can do two *req* consecutively.

The last example shows that a positive answer for the satisfiability problem does not imply a positive answer for the synthesis problem, as one cannot force the environment to do what is necessary to satisfy the specification. Another kind of counter-example would be a specification where System has to "guess the future", such as "There must be an a if and only if there is a b after" where a is an action controllable by the system and b is an action made by the environment.

Now let us formally define the synthesis problem. Let φ be an FO formula over $\Sigma = \Sigma_{\mathsf{s}} \uplus \Sigma_{\mathsf{e}}$ a finite alphabet of actions and $\mathcal{D}$ an infinite set of data values.

Definition 7. A *strategy* for the system is a mapping $\sigma : (\Sigma \times \mathcal{D})^* \to (\Sigma_{\mathsf{s}} \times \mathcal{D}) \cup \{\varepsilon\}$. A data word $w = (a_0, d_0)\ldots$ is *compatible* with σ if the following two conditions hold:

1. for all $i < |w|$ such that $a_i \in \Sigma_{\mathsf{s}}$, we have that $(a_i, d_i) = \sigma((a_0, d_0)\ldots(a_{i-1}, d_{i-1}))$.
2. if w is finite, then $\sigma(w) = \varepsilon$.

Furthermore, let us define fair data words with respect to a strategy. A data word $w = (a_0, d_0)\ldots$ is NOT fair with respect to a strategy σ if the following three conditions are satisfied:

1. w is infinite,
2. there are infinitely many $i \in \mathbb{N}$ such that $\sigma((a_0, d_0) \dots (a_i, d_i)) \neq \varepsilon$, and
3. there are only finitely many $i \in \mathbb{N}$ such that $a_i \in \Sigma_{\mathsf{s}}$.

In all other cases, w is said to be σ-fair. Finally, a strategy σ is φ-winning if all data words that are σ-compatible and σ-fair satisfy φ.

The general synthesis problem is defined as follows:

SYNTHESIS	
Input:	$\Sigma = \Sigma_{\mathsf{s}} \uplus \Sigma_{\mathsf{e}}$, $\mathcal{D}$, a first-order formula φ over Σ
Question:	Is there a φ-winning strategy for the system?

Chapter 3

Controller Synthesis for Automata Specifications

In this chapter, we study the control problem defined in Section 2.2. Recall that for the control problem, a system that is already partially defined is given as an input in addition to the specification. The goal is to then generate a controller that decides which actions to execute in order to satisfy the specification.

Here, we study open distributed systems where each process has a stack that can contain some data. Having a stack for each process in the model allows us to model various useful real-life mechanisms, such as unbounded data storing for each process, a recursion stack for recursive function calls, or simply any *last-in-first-out* data structure. Obviously, having a stack also raises the complexity of the model, as a process can no longer be defined as a finite state machine. This means that we have two kinds of infinities to deal with: the number of processes is unbounded, and each process has an unbounded memory. With such features, the general case is undecidable. However, with some natural restrictions the control problem becomes decidable again.

In Section 3.1, we define *dynamic pushdown systems*, a model to represent such systems and whose semantics is a DCTS as defined in the previous chapter. In Section 3.2, we discuss a restriction on the behaviors of such systems called *round-bounded behaviors*. In Section 3.3, we study the *control problem* for those dynamic pushdown systems. Finally in Section 3.4, we focus on another kind of restriction called *context-bounded behaviors*.

3.1 Dynamic Pushdown Systems

3.1.1 Definition

A dynamic pushdown system works as follows: there is a *global state* shared by all processes, that represents the state of the system. Then each active process has its own local state and local stack. A transition by a process may access the global state, local state and local stack of this process, but not local states or stacks of other processes. It will then change the global state, as well as the local state and local stack of the process making the transition. New processes can be spawned with an empty stack during the execution. There is no bound on the number of processes spawned during an execution, and each process acts independently, although they

share the same set of local states and transitions. Finally, accepting configurations are based on a set of accepting global states and accepting local states, with the stack contents of each process being ignored for that purpose. Then depending on the acceptance condition, the goal is to either reach, repeatedly reach, or ultimately avoid accepting configurations. Here is the formal definition.

Definition 8. A *dynamic pushdown system* (short: DPS) is a tuple

$$\mathcal{P} = (M, P, F_{\mathsf{glob}}, F_{\mathsf{loc}}, \mathfrak{F})$$

where $M = (\Sigma, S, T_{\mathsf{glob}}, s^{init})$ is a finite state machine, $P = (\Sigma, \Gamma, L, T_{\mathsf{loc}}, \ell^{init})$ is a pushdown machine over the same alphabet, $F_{\mathsf{glob}} \subseteq S$, $F_{\mathsf{loc}} \subseteq L$, and $\mathfrak{F} \in \{\mathsf{Reach}, \mathsf{Büchi}, \mathsf{coBüchi}\}$.

The semantics of a DPS is the DCTS (as defined in Section 2.2) $\mathcal{C}_{\mathcal{P}} = (M, P)$. In other words, DPS are a special case of DCTS where the global transition system is given by a finite state machine and the process transition system by a pushdown machine. Then by abuse of notation, we will say configurations of a DPS $\mathcal{P}$ instead of configurations of the underlying DCTS $\mathcal{C}_{\mathcal{P}}$. Those configurations are of the form $c = (s, (\ell_1, \gamma_1), (\ell_2, \gamma_2), \ldots, (\ell_n, \gamma_n))$ with $s \in S$, $\ell_1, \ldots, \ell_n \in L$, and $\gamma_1, \ldots, \gamma_n \in \Gamma^*$. Elements of S are called *global states*, and those from L are called *process* or *local states*. The initial configuration is simply (s^{init}).

Finally, the acceptance condition of this DCTS is derived from the last three elements of $\mathcal{P}$: the acceptance condition is $\mathsf{Acc}(F_{\mathsf{glob}}, F_{\mathsf{loc}}, \mathfrak{F}) = \mathfrak{F}(F_{\mathsf{glob}}, \{(\ell, \gamma) \mid \ell \in F_{\mathsf{loc}}\})$. In other words, a configuration $c = (s, (\ell_1, \gamma_1), (\ell_2, \gamma_2), \ldots, (\ell_n, \gamma_n))$ is accepting if $s \in F_{\mathsf{glob}}$ and $\ell_i \in F_{\mathsf{loc}}$ for all $i \in \{1, \ldots, n\}$. Put simply, depending on whether $\mathfrak{F}$ is Reach, $\mathsf{Büchi}$, or $\mathsf{coBüchi}$, the acceptance condition is to either reach an accepting configuration, repeatedly reach an accepting configuration, or eventually always avoid accepting configurations respectively. Again by abuse of notation, what we call accepting runs of $\mathcal{P}$ are runs for the underlying DCTS that are accepting with respect to Acc as defined above.

Example 15. Let us study an example for a model of a lock system for a resource shared by multiple processes. This resource can be read by any process, but must be locked before being written on so that only one process can modify it at a time. We give the following implementation of this system, and then we will check whether it is a correct one.

Let $\mathcal{P} = (M, P, \{s_\perp\}, \{\ell^{init}, \ell_1\}, \mathsf{Reach})$ with M and P defined as pictured in Figure 3.1. The idea is that the global state $s_\perp$ can only be reached if two *write* actions are made sequentially, which should not happen as only one process at a time should be able to modify the shared resource. Therefore, there is a bug in the implementation if one can find an accepting run of this DPS. An example of such a run is given in Figure 3.2.

Sometimes stacks are not needed for the model, so let us also define *dynamic finite state systems*, where the process transition system is given by a finite state machine instead of a pushdown machine.

Definition 9. A *dynamic finite state system* (short: DFS) is a tuple

$$\mathcal{F} = (M_{\mathsf{glob}}, M_{\mathsf{loc}}, F_{\mathsf{glob}}, F_{\mathsf{loc}}, \mathfrak{F})$$

where $M_{\mathsf{glob}} = (\Sigma, S, T_{\mathsf{glob}}, s^{init})$ and $M_{\mathsf{loc}} = (\Sigma, L, T_{\mathsf{loc}}, \ell^{init})$ are two finite state machines over the same alphabet, $F_{\mathsf{glob}} \subseteq S$, $F_{\mathsf{loc}} \subseteq L$, and $\mathfrak{F} \in \{\mathsf{Reach}, \mathsf{Büchi}, \mathsf{coBüchi}\}$.

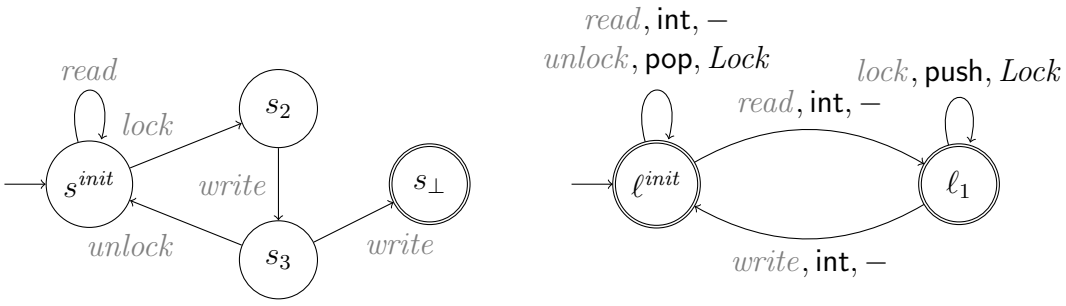

Figure 3.1: A DPS represented as its global finite state machine (left) and its process pushdown machine (right). Labels are in light gray.

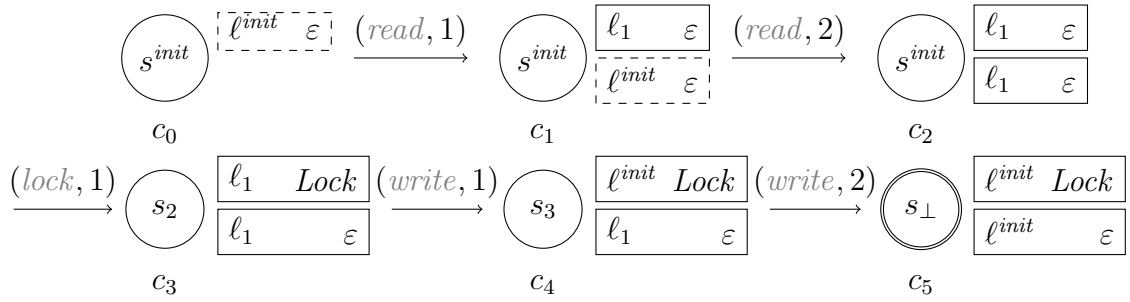

Figure 3.2: An example of an accepting finite run involving two processes

Its semantics is defined as the DCTS $\mathcal{C}_{\mathcal{F}} = (M_{\mathsf{glob}}, M_{\mathsf{loc}})$, and its acceptance condition $\mathfrak{F}(F_{\mathsf{glob}}, F_{\mathsf{loc}})$, similarly to DPS. A configuration of a DFS is of the form $c = (s, \ell_1, \ldots, \ell_n)$.

DPS are very similar to data automata introduced in [BDM+11] (or equivalently to class memory automata from [BS10]) due to the global/local distinction, if we forget about stacks. In fact, the only difference between DFS and data automata is that the *base automaton* in data automata (corresponding to the global FSM of a DFS) is given by a transducer. Also we do not really study the language produced by DPS, as we are only interested in whether accepting runs exist and not in the traces of actions that they produce. DPS are also equivalent to asynchronous dynamic pushdown networks (ADPN) from [BESS05] in which all transitions are global, that is, all transitions depend on both the local and global states. However, the underapproximation they study (context-bounded executions) is stricter than the bound on rounds from [LMP10a] that we will use in the following sections.

3.1.2 Emptiness Problem

The natural first step that comes to mind is to decide whether or not a given DPS (or DFS) has at least one accepting run starting from the initial configuration. This is called the emptiness problem, formalized as follows:

DPS-EMPTINESS	
Input:	A DPS $\mathcal{P}$
Question:	Is there an accepting run for $\mathcal{P}$?

The variant for DFS is defined analogously.

DFS-EMPTINESS	
Input:	A DFS $\mathcal{F}$
Question:	Is there an accepting run for $\mathcal{F}$?

Indeed, as we have seen in the previous example, even being able to find an accepting run is sometimes enough to prove that the implementation is not correct. Furthermore, if even the emptiness problem is undecidable or too hard, then it will naturally be futile to even try to control such systems because they are too complex. And actually, we show that this is the case.

Theorem 1. *The problem DPS-EMPTINESS is undecidable, and the problem DFS-EMPTINESS is inherently non-elementary.*

Proof. For the first part of the theorem, the undecidability result is widely known as two processes with a stack each are enough to simulate a Turing Machine even simply with a reachability condition. See e.g. [Ram00].

For the second part, we can relate DFS to Vector Addition Systems with States (VASS). A k-dimensional VASS is a tuple (Q, T) where Q is a finite set of control states and $T \subseteq Q \times \{-1, 0, +1\}^k \times Q$ is the transition relation. A configuration of a VASS is a pair (q, ν) where $q \in Q$ is the current control state and $\nu : \mathbb{N}^k$ is a valuation for the k counters. Applying a transition changes the current control state and change every counter individually by either adding 1, subtracting 1, or leaving the counter unmodified, with the condition that each counter must remain non-negative (otherwise the transition is not enabled). For instance, a transition $(q, (+1, -1, 0, 0), q')$ can be applied from configuration $(q, (3, 2, 1, 0))$ which results in configuration $(q', (4, 1, 1, 0))$.

Now notice that a configuration $c = (s, \ell_1, \ldots, \ell_n)$ of a DFS essentially counts how many processes are in each local state, of which there are a finite number. Thus from a given DFS we can build an $|L|$-dimensional VASS where each counter corresponds to a local state, and control states are the global states of the DFS. A configuration c of a DFS as above is equivalent to the configuration (s, ν) such that $\nu(k) = |\{i \mid \ell_i \text{ is the } k\text{-th local state}\}|$. A transition of the DFS that makes a process go from state ℓ to ℓ' and modify the global state from s to s' is simulated by a transition that subtracts 1 from the counter representing ℓ, adds 1 to the counter for ℓ', leaves the other counters unmodified, and change the control state from s to s'. A transition that activates a new process is simulated by a $+1$ on the adequate counter.

Similarly, a k-dimensional VASS can be simulated by a DFS with $k+2$ different local states with a simple idea. Let $L = \{\ell_1, \ldots, \ell_k\} \cup \{\ell^{init}, \ell_\perp\}$ where $\ell_1, \ldots, \ell_k$ are used to simulate the k counters of the VASS, and $\ell_\perp$ is a sink state used to simulate subtractions. A transition of the VASS is then simulated as follows:

- for each counter i with a $+1$, spawn a new process in state ℓ_i,

- for each counter i with a -1, move a process from state ℓ_i to $\ell_\perp$, and
- change the global state according to the transition.

This may require intermediate global states so that one VASS transition is simulated by multiple (at most k) transitions of the DFS, but this causes no problem.

Therefore, VASS and DFS are equivalent. The reachability problem for VASS (or equivalently, for Petri nets) is known to be decidable and non-elementary [May84, LS15], so the same can be said for emptiness of DFS. □

Since dynamic pushdown systems are too expressive, the next step is to try to restrict their power in order to get decidable subclasses. This is the object of the next sections.

3.2 Emptiness for Round-bounded Behaviors

We now study the case where we restrict behaviors to *round-bounded* runs, which were first introduced in [LMP10a]. Intuitively, during a *round*, the first process will do any number of transitions (possibly 0), then the second process will do any number of transitions, and so on. Once process $p+1$ has started performing transitions, process p cannot act again in this round. A run is then said to be *B-round bounded* if it can be split into at most B rounds.

This restriction comes naturally in the context of ring architectures with token passing for instance, where processes are organized in a round-robin fashion and can only execute actions when they have a *token*, which they then pass to the next process when they are done. Moreover, the idea is that most of the time an accepting run can be found with a low number of rounds, which can give examples of bad behaviors as in Example 15.

For instance, the run given in Figure 3.2 is a 2-bounded run (or n-bounded for any $n > 2$): there is one round from c_0 to c_2 with one transition from process 1 followed by one transition from process 2, and then when process 1 does another action it means another round begins. This second round goes from c_2 to the final configuration c_5, with process 1 making two transitions followed by a single transition from process 2.

Let us first give a formal definition, and later we shall prove that the emptiness problem for DPS is decidable under this restriction.

3.2.1 Round-bounded semantics

Formally, given a DPS $\mathcal{P}$ (it is defined similarly for DFS), we define *extended configurations* that contain information for counting rounds. An extended configuration is of the form (c, p, r) where c is a configuration of the DPS of size n, $p \in \{0, \dots, n\}$ represents the last process that made an action (or 0 if it is not yet defined), and $r \in \mathbb{N}_>$ is the round number. The initial extended configuration is $(c^{init}, 0, 1)$. If (c, p, r) is an extended configuration and c' is an (a, i)-successor of c, then the corresponding (a, i)-successor extended configuration is (c', i, r') with $r' = r$ if $i \geq p$, $r' = r + 1$ otherwise.

Let $B \in \mathbb{N}_>$ be a *bound*, and $\mathcal{P}$ a DPS. The *B-round bounded semantics* of $\mathcal{P}$, denoted by $\mathcal{P}^B$, is the transition system where nodes are extended configurations

with a number of rounds r up to B included, and transitions are as described above. Then, we say that a run ρ of $\mathcal{P}$ is B-round bounded if it is a run of $\mathcal{P}^B$, i.e., the maximum r that appears in ρ is less than or equal to B.

Example 16. For instance, the sequence of extended configurations corresponding to the run given in Figure 3.2 is:

$$\rho = (c_0, 0, 1)(c_1, 1, 1)(c_2, 2, 1)(c_3, 1, 2)(c_4, 1, 2)(c_5, 2, 2)$$

with the change of round occurring between c_2 and c_3. This run is indeed B-round bounded for any $B \geq 2$, as the maximum round that appears is 2.

Remark that a bound on the number of rounds does not imply a bound on the number of processes that can appear during an execution (and vice-versa), since new processes being spawned do not increase the round number. Moreover, even with a fixed number of processes, the length of a run is still not bounded as a process can execute as many actions as wanted consecutively during the same round.

The round-bounded emptiness problem is about deciding whether a given DPS has an accepting B-round bounded run for a given B.

DPS-EMPTINESS$_{rb}$	
Input:	A DPS $\mathcal{P}$, a bound $B \in \mathbb{N}_>$ (given in unary)
Question:	Is there an accepting B-round bounded run for $\mathcal{P}$?

Again, the variant for DFS is defined analogously:

DFS-EMPTINESS$_{rb}$	
Input:	A DFS $\mathcal{F}$, a bound $B \in \mathbb{N}_>$ (given in unary)
Question:	Is there an accepting B-round bounded run for $\mathcal{F}$?

Let us prove that both problems are decidable, and are PSPACE-complete.

Theorem 2. *DPS-EMPTINESS$_{rb}$ and DFS-EMPTINESS$_{rb}$ are PSPACE-complete.*

The next two sections are devoted to the proof of this theorem. In the first one, we give a PSPACE algorithm for DPS-EMPTINESS$_{rb}$, and in the second one we give a proof of PSPACE-hardness for DFS-EMPTINESS$_{rb}$. Since DFS are a special case of DPS, the theorem will then be proven. Note that [LMP10b] already proves the lower bound for their model by reduction from the membership problem for linear-bounded automata. For the sake of completeness, we will give another proof tailored to our definitions.

3.2.2 Decidability of DPS-EMPTINESS$_{rb}$

Let $B \in \mathbb{N}_>$ be a bound and $\mathcal{P} = (M, P, F_{\mathsf{glob}}, F_{\mathsf{loc}}, \mathfrak{F})$. We first give an algorithm for the simpler case where $\mathfrak{F} = \mathsf{Reach}$. The other two cases of acceptance condition will use a similar but slightly more involved idea.

Reach **acceptance.**

The proof will be in three parts. In the first part, we present the notion of interface. Intuitively, an interface represents the part of a run pertaining to a single process, while only keeping necessary information needed to coordinate with other processes involved in the run. More precisely, nothing pertaining to local state or stack is stored, as this information cannot be used by other processes. We show that a collection of interfaces can represent an accepting run of the DPS if some conditions are satisfied.

In the second part, we show that checking whether a given tuple is an interface can be done in polynomial time. This is done by reducing the problem to the emptiness problem for pushdown automata.

In the final part, we give the algorithm to solve DPS-EMPTINESS$_{rb}$ and describe its complexity. This part uses results from the two previous parts.

Interfaces. As said just above, an interface is a collection of information used to represent the part of a run involving a single process. More specifically, for each round, an interface stores the global state that occured when the process first started playing during this round, and the global state after the process performed its last action of the round (which are the same if the process did not perform any action and/or was not already active during this round). Moreover, an interface also stores the starting round of the process, i.e., the first round where the process did an action.

An interface does not store the local states and stack contents of the process, as these cannot be accessed by other processes anyway. We simply require that there exists such local states and stack contents such that a run can be built using the global states stored in the interface.

Definition 10. An *interface* is a tuple

$$\mathcal{I} = [r^{\mathrm{s}}, (s_1, \ldots, s_B), (s'_1, \ldots, s'_B)] \in \{1, \ldots, B\} \times S^B \times S^B$$

satisfying the following conditions:

1. For all $1 \leq r < r^{\mathrm{s}}$, we have $s_r = s'_r$, i.e., before the starting round r^{s} the process does not change the global state.
2. There are local states $\ell_{r^{\mathrm{s}}-1}, \ldots, \ell_B \in L$ and stack contents $\gamma_{r^{\mathrm{s}}-1}, \ldots, \gamma_B \in \Gamma^*$ such that
   (i) for all $r^{\mathrm{s}} \leq r \leq B$ there is a finite partial run of $\mathcal{P}$ from $c_r = (s_r, (\ell_{r-1}, \gamma_{r-1}))$ to $c'_r = (s'_r, (\ell_r, \gamma_r))$,
   (ii) this run has length at least two (i.e., it performs at least one transition) if $r = r^{\mathrm{s}}$,
   (iii) $\ell_{r^{\mathrm{s}}-1}$ is the initial local state ℓ^{init}, $\gamma_{r^{\mathrm{s}}-1}$ is the empty stack ε, and
   (iv) $\ell_B \in F_{\mathsf{loc}}$ is an accepting local state.

We refer to the first B-tuple of $\mathcal{I}$ as the *left interface*, noted $\mathcal{I}^{left}$, and to the second B-tuple as the *right interface*, noted $\mathcal{I}^{right}$. The starting round r^{s} of $\mathcal{I}$ is referred to as $r^{\mathrm{s}}_{\mathcal{I}}$.

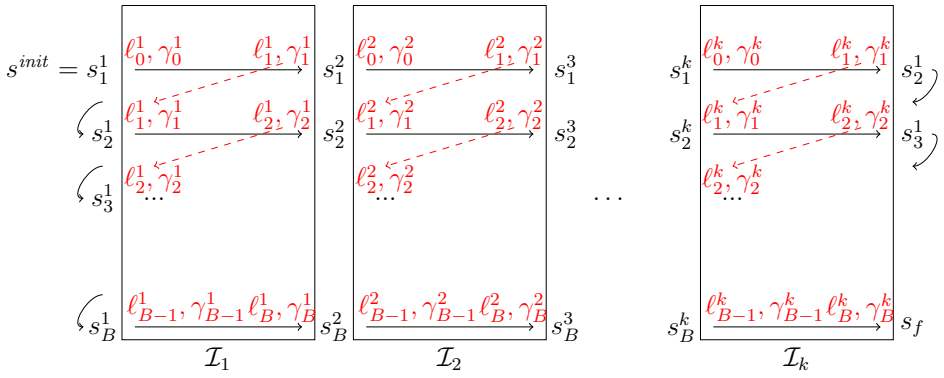

Figure 3.3: A run as the composition of compatible interfaces; for simplicity all starting rounds here are 1

Interfaces are used to decompose a run of a DPS involving k processes into k separate parts. Conversely, using k interfaces, we can build a run of a DPS as long as those interfaces are compatible, in the sense that the starting rounds are never decreasing (since a process cannot become active before an earlier process), and that the global states coincide between interfaces.

Formally, we say that an interface $\mathcal{I}_1$ is *compatible* with an interface $\mathcal{I}_2$ if $r^{\mathrm{s}}_{\mathcal{I}_1} \leq r^{\mathrm{s}}_{\mathcal{I}_2}$ and ${\mathcal{I}_1}^{right} = {\mathcal{I}_2}^{left}$. Then, we say that a sequence $\mathcal{I}_1, \ldots, \mathcal{I}_k$ of interfaces is *valid* if the following conditions are satisfied:

- For all $1 < p \leq k$, $\mathcal{I}_{p-1}$ is compatible with $\mathcal{I}_p$.
- Let ${\mathcal{I}_1}^{left} = (s_1, \ldots, s_B)$ and ${\mathcal{I}_k}^{right} = (s'_1, \ldots, s'_B)$. Then:
  - s_1 is the initial global state s^{init},
  - s'_B is an accepting global state, and
  - $s_r = s'_{r-1}$ for all $1 < r \leq B$.

A valid sequence of k interfaces is an abstract way to represent an accepting run with k processes, as illustrated in Figure 3.3. This is formalized in the following lemma.

Lemma 3. *There is a (finite) accepting B-bounded run of $\mathcal{P}$ if and only if there is a valid sequence of k interfaces $\mathcal{I}_1, \ldots, \mathcal{I}_k$ for some $k \geq 1$.*

Proof. Given a configuration $c = (s, (\ell_1, \gamma_1), \ldots, (\ell_k, \gamma_k))$, we let $proj_S(c) = s$, and $proj_{L,p}(c) = (\ell_p, \gamma_p)$ if $p \leq k$ and $proj_{L,p}(c) = (\ell^{init}, \varepsilon)$ if $k < p$.

Let us first remark that given a run of $\mathcal{P}$, the size of a successor configuration is greater than the size of the previous one if and only if the process performing an action is different than the previous one and it is the first time that this process performs an action in the run. If this new process is the k-th one, then the size of the configuration goes from $k-1$ to k. In terms of interfaces, this means that $r^{\mathrm{s}}_{\mathcal{I}_k}$ is the current round. Similarly, if a process p performs an action and the configuration reached has size $k > p$, then necessarily p was already active before this round.

$\boxed{\Rightarrow}$ Let ρ be an accepting B-bounded run of $\mathcal{P}$ using k processes. Note that ρ can be divided into $\rho = \rho^0\rho^1 \ldots \rho^{B'}$, with $B' \leq B$, where each ρ^r is the part of the run corresponding to round r, and $\rho^0 = ((s^{init}), 0, 1)$. For every $1 \leq p \leq k$ and

$1 \leq r \leq B'$, let $n(p,r) \in \mathbb{N}$ and $(c_\alpha^{(p,r)})_{\alpha \leq n(p,r)}$ be the (possibly empty) finite sequence of configurations visited by process p during round r in the run ρ. Formally, for each $1 \leq r \leq B'$, $\rho^r = \rho^{1r} \cdots \rho^{kr}$, with $\rho^{pr} = (c_1^{(p,r)}, p, r) \cdots (c_{n(p,r)}^{(p,r)}, p, r)$. Note that ρ^{pr} may be empty.

Fix a round $1 \leq r \leq B'$, and a process $1 \leq p \leq k$. Let p', r' be such that $\rho^{p'r'}$ is the last non-empty part of ρ before ρ^{pr}. Let $s_0^{(p,r)} = proj_S(c_{n(p',r')}^{(p',r')})$ and $(\ell_0^{(p,r)}, \gamma_0^{(p,r)}) = proj_{L,p}(c_{n(p',r')}^{(p',r')})$ be respectively the global state and local state reached by process p just before process p starts in round r. Let $s_m^{(p,r)} = proj_S(c_m^{(p,r)})$ and $(\ell_m^{(p,r)}, \gamma_m^{(p,r)}) = proj_{L,p}(c_m^{(p,r)})$, for $1 \leq m \leq n_{(p,r)}$. Since there is a transition from $(c_{n(p',r')}^{(p',r')}, p', r')$ to $(c_1^{(p,r)}, p, r)$ and, for all $1 \leq m < n(p,r)$, there is a transition from $(c_m^{(p,r)}, p, r)$ to $(c_{m+1}^{(p,r)}, p, r)$, then by definition, we have that $c_1^{(p,r)}$ is an (a,p)-successor of $c_{n(p',r')}^{(p',r')}$ for some $a \in \Sigma$, and, for all $1 \leq m < n(p,r)$, $c_{m+1}^{(p,r)}$ is an (a,p)-successor of $c_m^{(p,r)}$ for some $a \in \Sigma$. Observe that this implies that $(s_{m+1}^{(p,r)}, (\ell_{m+1}^{(p,r)}, \gamma_{m+1}^{(p,r)}))$ is an $(a,1)$-successor of $(s_m^{(p,r)}, (\ell_m^{(p,r)}, \gamma_m^{(p,r)}))$, for all $0 \leq m < n(p,r)$. Hence there is a finite run of $\mathcal{P}$ from $(s_0^{(p,r)}, (\ell_0^{(p,r)}, \gamma_0^{(p,r)}))$ to $(s_{n(p,r)}^{(p,r)}, (\ell_{n(p,r)}^{(p,r)}, \gamma_{n(p,r)}^{(p,r)}))$. If ρ^{pr} is empty we let $s_{n(p,r)}^{(p,r)} = s_0^{(p,r)}$ and $(\ell_{n(p,r)}^{(p,r)}, \gamma_{n(p,r)}^{(p,r)})) = ((\ell_0^{(p,r)}, \gamma_0^{(p,r)}))$. For all $B' < r \leq B$, for all $1 \leq p \leq k$, let $n(p,r) = 0$, and $s_0^{(p,r)} = s_{n(k,B')}^{(k,B')}$, and $(\ell_0^{(p,r)}, \gamma_0^{(p,r)}) = (\ell_{n(p,B')}^{(p,B')}, \gamma_{n(p,B')}^{(p,B')})$. In other words, the global state stays unchanged since the end of the run ρ, and the local state and stack of each process stays as it was at the end of their local run in the last round B'.

For each process p, let r_p^{s} be the smallest round $1 \leq r \leq B'$ such that ρ^{pr} is not empty. Thus, for all $1 \leq p \leq k$, we define the interface

$$\mathcal{I}_p = [r_p^{\mathrm{s}}, (s_0^{(p,1)}, \ldots, s_0^{(p,B)}), (s_{n(p,1)}^{(p,1)}, \ldots, s_{n(p,B)}^{(p,B)})]$$

From the above, we know that there exists a finite partial run of $\mathcal{P}$ from $(s_0^{(p,r)}, (\ell_0^{(p,r)}, \gamma_0^{(p,r)}))$ to $(s_{n(p,r)}^{(p,r)}, (\ell_{n(p,r)}^{(p,r)}, \gamma_{n(p,r)}^{(p,r)}))$, for all $1 \leq r \leq B$. Moreover, in the first round, if a process plays, it is for the first time. By definition, $(\ell_0^{(p,1)}, \gamma_0^{(p,1)}) = proj_{L,p}(c_{n(p',r')}^{(p',r')})$ with $(p', r') = (p-1, 1)$ if $p > 1$, $(0,0)$ otherwise. This configuration $c_{n(p',r')}^{(p',r')}$ is of size $< p$ since process p has not already played in this run, thus $(\ell_0^{(p,1)}, \gamma_0^{(p,1)}) = (\ell^{init}, \varepsilon)$. Since ρ is winning, every process ends in a final local state, i.e., $\ell_{n(p,B)}^{(p,B)}$ is accepting, for all p. Finally, for all $r < r_p^{\mathrm{s}}$, ρ^{pr} is empty. By construction then, $s_0^{(p,r)} = s_{n(p,r)}^{(p,r)}$. All of this ensures that $\mathcal{I}_p$ is indeed an interface. By construction, for all process $1 < p \leq k$ and for all rounds $r \geq 1$ we have $s_0^{(p,r)} = s_{n(p-1,r)}^{(p-1,r)}$. Moreover, by definition of a run of $\mathcal{P}$, if a process p appears in round r, then necessarily, process $p+1$ appears for the first time in round $r' \geq r$. Hence for all $1 \leq p < k$, $r_p^{\mathrm{s}} \leq r_{p+1}^{\mathrm{s}}$ and interface $\mathcal{I}_p$ is compatible with interface $\mathcal{I}_{p+1}$. By construction, for all $1 \leq r \leq B$, $s_0^{(1,r)} = s_{n(k,r-1)}^{(k,r-1)}$, and because ρ is winning we also have $s_0^{(1,1)} = s^{init}$ and $s_{n_{(k,B)}}^{(k,B)}$ is accepting, then all the conditions of the lemma are fulfilled.

$\boxed{\Leftarrow}$ Conversely, let $\mathcal{I}_1, \ldots, \mathcal{I}_k$ be interfaces verifying the conditions. Let $s^{(p,r)}$ be the r-th component of $\mathcal{I}_p{}^{left}$ and $s'^{(p,r)}$ be the r-th component of $\mathcal{I}_p{}^{right}$. Consider

the following partial run of $\mathcal{P}$:

$$\rho_{pr} = \begin{cases} (s_1^{(p,r)}, (\ell_1^{(p,r)}, \gamma_1^{(p,r)})) \dots (s_{n(p,r)}^{(p,r)}, (\ell_{n(p,r)}^{(p,r)}, \gamma_{n(p,r)}^{(p,r)})) \text{ if } r \geq r^{\mathrm{s}}_{\mathcal{I}_p}, \\ (s^{(p,r)}, (\ell^{init}, \varepsilon)) \text{ otherwise.} \end{cases}$$

with $s_1^{(p,r)} = s^{(p,r)}$ and $s_{n(p,r)}^{p,r} = s'^{(p,r)}$ in the first case. The existence of such a run, as well as the local states and stack contents, is ensured by the definition of an interface. Moreover, we have that

$$(\ell_1^{(p,r)}, \gamma_1^{(p,r)}) = \begin{cases} (\ell_{n(p,r-1)}^{(p,r-1)}, \gamma_{n(p,r-1)}^{(p,r-1)}) & \text{if } r > r^{\mathrm{s}}_{\mathcal{I}_p}, \\ (\ell^{init}, \varepsilon) & \text{if } r = r^{\mathrm{s}}_{\mathcal{I}_p} \end{cases} \tag{3.1}$$

We build a B-bounded run ρ of $\mathcal{P}$ as follows. The idea is simply to rearrange the runs ρ_{pr} defined above in order to get a run of $\mathcal{P}$, as pictured in Figure 3.3. For all $1 \leq p \leq k$ and $r^{\mathrm{s}}_{\mathcal{I}_p} \leq r \leq B$, let $\rho^{(p,r)}$ be the (possibly empty) sequence of nodes

$$\rho^{(p,r)} = (c_2^{(p,r)}, p, r), \dots, (c_{n(p,r)}^{(p,r)}, p, r)$$

where $c_m^{(p,r)} = (s_m^{(p,r)}, (\ell_1, \gamma_1), \dots, (\ell_{k'}, \gamma_{k'}))$ with:

- $\ell_p = \ell_m^{(p,r)}$ and $\gamma_p = \gamma_m^{(p,r)}$,
- for $q < p$, $\ell_q = \ell_{n(q,r)}^{(q,r)}$ (i.e. the last local state of process q) and $\gamma_q = \gamma_{n(q,r)}^{(q,r)}$,
- for $p < q \leq k'$, $\ell_q = \ell_{n(q,r-1)}^{(q,r-1)}$ and $\gamma_q = \gamma_{n(q,r-1)}^{(q,r-1)}$ (in that case, the last local state of process q has been reached in the previous round)

For p such that $r^{\mathrm{s}}_{\mathcal{I}_p} > r$, we let $\rho^{(p,r)} = \varepsilon$ instead, and we let $\rho^{(0,0)} = ((s^{init}), 0, 1)$. By definition of an interface, $\rho^{(p,r)}$ is a partial run of $\mathcal{P}$.

We show that the sequence $\rho = \rho^{(0,0)}\rho^{(1,1)}\rho^{(2,1)} \dots \rho^{(k,1)}\rho^{(1,2)} \dots \rho^{(k,B)}$ is a run of $\mathcal{P}$ by induction on the pair (p, r). The base case where $p = r = 0$ is trivial. Let $1 \leq p < k$ and $1 \leq r \leq B$ and assume that $\rho^{(0,0)}\rho^{(1,1)} \dots \rho^{(p,r)}$ is a run. Let (p', r') be the successor of (p, r).

- Suppose that $r = r'$ and $p' = p + 1$ (same round, next process). Without loss of generality, assume that $\rho^{(p,r)}$ and $\rho^{(p+1,r)}$ are not empty. Let us denote by $c = c_{n(p,r)}^{(p,r)}$ the last configuration of $\rho^{(p,r)}$ and by $c' = c_2^{(p+1,r)}$ the first configuration of $\rho^{(p+1,r)}$, we show that $(c', p+1, r)$ is a successor of (c, p, r).

  If $r > r^{\mathrm{s}}_{\mathcal{I}_{p+1}}$, then necessarily $r > 1$ and $|c'| = |c|$. Then

  $$c = (s_{n(p,r)}^{(p,r)}, (\ell_1, \gamma_1) \dots (\ell_{k'}, \gamma_{k'}))$$

  with

  $$(\ell_{p+1}, \gamma_{p+1}) = (\ell_{n(p+1,r-1)}^{(p+1,r-1)}, \gamma_{n(p+1,r-1)}^{(p+1,r-1)}) = (\ell_1^{(p+1,r)}, \gamma_1^{(p+1,r)}))$$

  and

  $$c' = (s_2^{(p+1,r)}, (\ell'_1, \gamma'_1) \dots (\ell'_{k'}, \gamma'_{k'}))$$

  with

  $$(\ell'_{p+1}, \gamma'_{p+1}) = (\ell_2^{(p+1,r)}, \gamma_2^{(p+1,r)})).$$

such that

$$(\ell_q, \gamma_q) = (\ell'_q, \gamma'_q) = \begin{cases} (\ell^{(q,r)}_{n(q,r)}, \gamma^{(q,r)}_{n(q,r)}) & \text{if } q \leq p, \\ (\ell^{(q,r-1)}_{n(q,r-1)}, \gamma^{(q,r-1)}_{n(q,r-1)}) & \text{if } q > p \end{cases}$$

Since $\mathcal{I}_p$ is compatible with $\mathcal{I}_{p+1}$, we have that $s^{(p+1,r)} = s'^{(p,r)}$. Thus $s_1^{(p+1,r)} = s^{(p+1,r)} = s'^{(p,r)} = s^{(p,r)}_{n(p,r)}$, and by definition of an interface, $(s_2^{(p+1,r)}, (\ell'_{p+1}, \gamma'_{p+1}))$ is a successor of $(s^{(p,r)}_{n(p,r)}, (\ell_{p+1}, \gamma_{p+1}))$. Hence from the above equalities, the extended configuration $(c', p+1, r)$ is indeed a successor of (c, p, r).

If $r = r^{\mathrm{s}}_{\mathcal{I}_{p+1}}$, then $|c'| = p + 1 = |c| + 1$ since process $p + 1$ performs an action for the first time. Then

$$c = (s^{(p,r)}_{n(p,r)}, (\ell_1, \gamma_1) \dots (\ell_p, \gamma_p))$$

and

$$c' = (s_2^{(p+1,r)}, (\ell'_1, \gamma'_1) \dots (\ell'_{p+1}, \gamma'_{p+1})$$

with

$$(\ell'_{p+1}, \gamma'_{p+1}) = (\ell_2^{(p+1,r)}, \gamma_2^{(p+1,r)})$$

Since $r^{\mathrm{s}}_{\mathcal{I}_{p+1}} = r$, by definition of the interface, we know that $(\ell_1^{(p+1,r)}, \gamma_1^{(p+1,r)}) = (\ell^{init}, \varepsilon)$ and that $(s_2^{(p+1,r)}, (\ell_2^{(p+1,r)}, \gamma_2^{(p+1,r)}))$ is a successor of $(s^{(p+1,r)}, (\ell^{init}, \varepsilon))$. As with the other case, $s_1^{(p+1,r)} = s^{(p,r)}_{n(p,r)}$, thus $(c', p+1, r)$ is indeed a successor of (c, p, r).

- Suppose now that $r' = r + 1$, $p = k$, and $p' = 1$ (start of a new round). Again, without loss of generality, suppose that $\rho^{(k,r)}$ and $\rho^{(1,r+1)}$ are not empty, and let us denote by $c = c^{(k,r)}_{n(k,r)}$ the last configuration of $\rho^{(k,r)}$ and by $c' = c_2^{(1,r+1)}$ the first configuration of $\rho^{(1,r+1)}$. We show that $(c', 1, r+1)$ is a successor of (c, k, r).

  Necessarily the starting round of the first process is 1, so $r^{\mathrm{s}}_{\mathcal{I}_1} < r + 1$ and have that $|c'| = |c|$. Then

$$\begin{aligned} c' &= (s_2^{(1,r+1)}, (\ell'_1, \gamma'_1), \dots, (\ell'_k, \gamma'_k)) \\ &= (s_2^{(1,r+1)}, (\ell_2^{(1,r+1)}, \gamma_2^{(1,r+1)}), \dots, (\ell^{(k,r)}_{n(k,r)}, \gamma^{(k,r)}_{n(k,r)})), \end{aligned}$$

  and

$$\begin{aligned} c &= (s^{(k,r)}_{n(k,r)}, (\ell_1, \gamma_1), \dots, (\ell_k, \gamma_k)) \\ &= (s^{(k,r)}_{n(k,r)}, (\ell^{(1,r)}_{n(1,r)}, \gamma^{(1,r)}_{n(1,r)}), \dots, (\ell^{(k,r)}_{n(k,r)}, \gamma^{(k,r)}_{n(k,r)})). \end{aligned}$$

  By definition, we have that $s^{(k,r)}_{n(k,r)} = s'^{(k,r)}$ and $s^{(1,r+1)} = s_1^{(1,r+1)}$. Moreover, since $\mathcal{I}_1, \dots, \mathcal{I}_k$ is a valid sequence of interfaces, then $s'^{(k,r)} = s^{(1,r+1)}$. The existence of the run ρ_{1r+1} and equalities 3.1 allow to conclude that $(c', 1, r+1)$ is indeed a successor of (c, k, r).

The run ρ we built is by definition B-bounded. As $\mathcal{I}_1, \ldots, \mathcal{I}_k$ are interfaces, their last local state is accepting. Moreover, since $\mathcal{I}_1, \ldots, \mathcal{I}_k$ is a *valid* sequence of interfaces we know that the last global state is accepting. Thus ρ is accepting, which concludes the proof of the lemma.

Observe that we have supposed in this proof that no $\rho^{(p,r)}$ was empty. If this was not the case, the global state will remain the same in the different interfaces, and the same proof applies, with tedious modifications of the indices. Again, if a whole round is missing, or if the run does not start in round 1, then one can rewrite the numbers of the round in a correct way, with no other modification, and with the number of rounds even smaller than before. □

Note that if there is a valid sequence of interfaces $\mathcal{I}_1, \ldots, \mathcal{I}_k$ such that two interfaces $\mathcal{I}_i$ and $\mathcal{I}_j$ are equal for some $i \neq j$, then $\mathcal{I}_1, \ldots, \mathcal{I}_i, \mathcal{I}_{j+1}, \ldots, \mathcal{I}_k$ is also a valid sequence of interfaces. Therefore, if there is a valid sequence of interfaces, then there is one where all interfaces are pairwise distinct. With $N = B \times |S|^B \times |S|^B$ being the total number of interfaces, we deduce the following fact.

Corollary 4. *There is a (finite) accepting B-bounded run of $\mathcal{P}$ if and only if there is a valid sequence of k interfaces $\mathcal{I}_1, \ldots, \mathcal{I}_k$ for some $1 \leq k \leq N$.*

Guessing an interface. The algorithm we will describe needs to be able to guess interfaces. To that end, it will actually guess a tuple of the form $\mathcal{I} = [r^{\mathrm{s}}, (s_1, \ldots, s_B), (s'_1, \ldots, s'_B)]$, and then check (in polynomial time) whether $\mathcal{I}$ is an interface. To do that, we simply need to guess the behavior of a single process whose global state is changed at some points, which simulates other processes doing transitions before letting that process play again in a later round. In other words, we check the non-emptiness of a pushdown automaton (a pushdown machine with a set of accepting states) that simulates the actions of $\mathcal{P}$ on a single process and has special transitions to change the global state from s'_r to s_{r+1}. As non-emptiness of a pushdown automaton can be checked in polynomial time [HMRU00], whether a given tuple is an interface can also be checked in polynomial time.

We define the pushdown automaton $A_{\mathcal{I}} = (\Sigma', \Gamma, Q, T, q_0, \{\mathrm{Win}\})$ with $Q = \{q_0, \mathrm{Win}\} \cup (S \times L \times \{r^{\mathrm{s}}, \ldots, B\})$, Σ' is an arbitrary alphabet of size 1 which will be omitted, $F = \{\mathrm{Win}\}$, and T defined as follows.

For every $A \in \Gamma$, for every $s, s' \in S$, for every $\ell, \ell' \in L$, for every $r \in \{r^{\mathrm{s}}, \ldots, B\}$, for all $a \in \Sigma$ and $\mathsf{act} \in \{\mathsf{push}, \mathsf{pop}, \mathsf{int}\}$,

1. $((s, \ell, r), (\mathsf{act}, A), (s', \ell', r)) \in T$ if $(s, a, s') \in T_{\mathsf{glob}}$ and $(\ell, a, \mathsf{act}, A, \ell') \in T_{\mathsf{loc}}$,
2. $(q_0, (\mathsf{act}, A), (s', \ell', r^{\mathrm{s}})) \in T$ if $(s_{r^{\mathrm{s}}}, a, s') \in T_{\mathsf{glob}}$ and $(\ell^{init}, a, \mathsf{act}, A, \ell') \in T_{\mathsf{loc}}$,
3. $((s'_r, \ell, r), (\mathsf{int}, -), (s_{r+1}, \ell, r+1) \in T$ if $r < B$,
4. $((s'_B, \ell, B), (\mathsf{int}, -), \mathrm{Win})$ if $\ell \in F_{\mathsf{loc}}$.

Intuitively, the two first kind of transitions corresponds to the transitions of $\mathcal{P}$ for a single process, while the third kind non-deterministically changes the global state from s'_r to s_{r+1} (which, in $\mathcal{P}$, corresponds to other processes acting and modifying the global state). The third component of Q allows us to track the index of the component of the interface we are following, which corresponds to the round being simulated, and is increased only when performing a transition of the third kind. $A_{\mathcal{I}}$

accepts only after taking transitions of the third kind exactly $B - r^{\mathrm{s}}$ times, and after that reaching global state s'_B with an accepting local state, which is the only way to reach the final state Win through the last kind of transition. The second kind of transition is there to ensure that the run at round r^{s} is not empty, as this is the only kind of transition that exits the initial state q_0 which is essentially equivalent to the state $(s_{r^{\mathrm{s}}}, \ell^{init}, r^{\mathrm{s}})$. Note that this automaton does not check that $s_r = s'_r$ for all $r < r^{\mathrm{s}}$, which is the last condition needed to ensure that $\mathcal{I}$ is an interface.

Lemma 5. *$\mathcal{I}$ is an interface if and only if there is an accepting run of $A_{\mathcal{I}}$ and $s_r = s'_r$ for all $r < r^s$.*

Proof. $\boxed{\Leftarrow}$ Suppose there is an accepting run of $A_{\mathcal{I}}$ and that $s_r = s'_r$ for all $r < r^{\mathrm{s}}$. By construction, the accepting run is necessarily of the form $(q_0, \varepsilon) \cdot \rho^{r^{\mathrm{s}}} \cdots \rho^B \cdot (Win, \gamma)$, where, for all $r^{\mathrm{s}} \leq r \leq B$,

$$\rho^r = ((s^1_r, \ell^1_r, r), \gamma^1_r) \cdots ((s^{i_r}_r, \ell^{i_r}_r, r), \gamma^{i_r}_r)$$

with

- $s^1_r = s_r$, the r-th component of $\mathcal{I}^{left}$, if $r \neq r^{\mathrm{s}}$, and
- $s^{i_r}_r = s'_r$, the r-th component of $\mathcal{I}^{right}$.

Moreover, for all $r^{\mathrm{s}} \leq r \leq B - 1$, $\ell^{i_r}_r = \ell^1_{r+1}$. Finally, for all $1 \leq i < i_r$, $(s^{i+1}_r, (\ell^{i+1}_r, \gamma^{i+1}_r))$ is a successor of $(s^i_r, (\ell^i_r, \gamma'^i_r))$ in $\mathcal{P}$, and $(s^1_{r^{\mathrm{s}}}, (\ell^1_{r^{\mathrm{s}}}, \gamma^1_{r^{\mathrm{s}}}))$ is a successor of $(s_{r^{\mathrm{s}}}, \ell^{init}, \varepsilon)$.

To show that $\mathcal{I}$ fulfills the conditions for being an interface, let $\ell_r = \ell^1_{r+1}$ and $\gamma_r = \gamma^1_{r+1}$ for all $r^{\mathrm{s}} - 1 \leq r < B$. Moreover, let $\ell_B = \ell^{i_B}_B$ and $\gamma_B = \gamma^{i_B}_B$. Then, for all $r > r^{\mathrm{s}}$, there is a partial run of $\mathcal{P}$ starting from $c_r = (s^1_r, (\ell^1_r, \gamma^1_r)) = (s_r, (\ell_{r-1}, \gamma_{r-1}))$ to $c'_r = (s^{i_r}_r, (\ell^{i_r}_r, \gamma^{i_r}_r) = (s'_r, (\ell^1_{r+1}, \gamma^1_{r+1})) = (s'_r, (\ell_r, \gamma_r))$, and for $r = r^{\mathrm{s}}$, there is a partial run of $\mathcal{P}$ starting from $c_{r^{\mathrm{s}}} = (s_{r^{\mathrm{s}}}, \ell^{init}, \varepsilon)$ to $c'_{r^{\mathrm{s}}} = (s'_{r^{\mathrm{s}}}, (\ell_{r^{\mathrm{s}}}, \gamma_{r^{\mathrm{s}}}))$.

Finally, by construction of $A_{\mathcal{I}}$ we have that $\ell_B = \ell^{i_B}_B \in F_{\mathsf{loc}}$, and by hypothesis we know that $s_r = s'_r$ for all $r < r^{\mathrm{s}}$. So, $\mathcal{I}$ is indeed an interface.

$\boxed{\Rightarrow}$ Conversely, suppose $\mathcal{I}$ is an interface. One can build an accepting run of $A_{\mathcal{I}}$: $(q_0, \varepsilon) \cdot \rho^{r^{\mathrm{s}}} \cdots \rho^B \cdot (\mathrm{Win}, \gamma_B)$ as follows. For $r^{\mathrm{s}} \leq r \leq B$, let

$$(s_r, (\ell_{r-1}, \gamma_{r-1}))(s^r_1, (\ell^r_1, \gamma^r_1)) \ldots (s^r_{i_r-1}, (\ell^r_{i_r-1}, \gamma^r_{i_r-1}))(s'_r, (\ell_r, \gamma_r))$$

be the run of $\mathcal{P}$ ensured by $\mathcal{I}$. Then, when $r = r^{\mathrm{s}}$, we let

$$\rho^{r^{\mathrm{s}}} = ((s^{r^{\mathrm{s}}}_1, \ell^{r^{\mathrm{s}}}_1, r^{\mathrm{s}}), \gamma^{r^{\mathrm{s}}}_1) \ldots ((s^{r^{\mathrm{s}}}_{i_{r^{\mathrm{s}}}-1}, \ell^{r^{\mathrm{s}}}_{i_{r^{\mathrm{s}}}-1}, r^{\mathrm{s}}), \gamma^{r^{\mathrm{s}}}_{i_{r^{\mathrm{s}}}-1})((s'_{r^{\mathrm{s}}}, \ell_{r^{\mathrm{s}}}, r^{\mathrm{s}}), \gamma_{r^{\mathrm{s}}})$$

and for $r^{\mathrm{s}} < r \leq B$, we let

$$\rho^r = ((s_r, \ell_{r-1}, r), \gamma_{r-1})((s^r_1, \ell^r_1, r), \gamma^r_1) \ldots ((s^r_{i_r-1}, \ell^r_{i_r-1}, r), \gamma^r_{i_r-1})((s'_r, \ell_r, r)\gamma_r)$$

which are runs of $A_{\mathcal{I}}$ using only transitions of the first type.

The transition from (q_0, ε) to $\rho^{r^{\mathrm{s}}}$ is a transition of the second type, transitions from ρ^r to ρ^{r+1} are transitions of the third type, and the transition from ρ^B to (Win, γ_B) is a transition of the fourth type which is possible because ℓ_B is accepting by definition of an interface. □

Description of the algorithm. The algorithm to solve DPS-EMPTINESS$_{rb}$ first non-deterministically guesses a tuple $\mathcal{I}_1$ for the first process and checks that it is actually an interface as described in the previous part. It then stores $r^{\mathrm{s}}_{\mathcal{I}_1}$, ${\mathcal{I}_1}^{left}$, and ${\mathcal{I}_1}^{right}$, which takes a polynomial amout of space.

Then, it guesses an interface $\mathcal{I}_2$ for the second process, checks that it is compatible by comparing $r^{\mathrm{s}}_{\mathcal{I}_2}$ and ${\mathcal{I}_2}^{left}$ with the previously stored $r^{\mathrm{s}}_{\mathcal{I}_1}$ and ${\mathcal{I}_1}^{right}$, and then replaces ${\mathcal{I}_1}^{right}$ by ${\mathcal{I}_2}^{right}$ and $r^{\mathrm{s}}_{\mathcal{I}_1}$ by $r^{\mathrm{s}}_{\mathcal{I}_2}$. That way, only ${\mathcal{I}_1}^{left}$, $r^{\mathrm{s}}_{\mathcal{I}_2}$, and ${\mathcal{I}_2}^{right}$ are stored, and the amout of space taken is unchanged. We continue guessing compatible interfaces, storing at each step i the values of ${\mathcal{I}_1}^{left}$, $r^{\mathrm{s}}_{\mathcal{I}_i}$, ${\mathcal{I}_i}^{right}$.

Eventually, the algorithm guesses that the last process has been reached (remember that we need only up to an exponential number of interfaces by Corollary 4). At that point, there are two halves of interfaces stored in memory: the left interface ${\mathcal{I}_1}^{left} = (s_1, \ldots, s_B)$ of the first process, and the right interface ${\mathcal{I}_k}^{right} = (s'_1, \ldots, s'_B)$ of the last process.

We accept if, for all $i \in \{1, \ldots, B-1\}$, we have that $s'_i = s_{i+1}$, $s_1 = s^{init}$, and $s'_B \in F_{\mathsf{glob}}$. By Lemma 3, there is an accepting B-run of $\mathcal{P}$. This algorithm is non deterministic and takes a polynomial amount of space, therefore the problem is in NPSPACE, which is equivalent to PSPACE due to Savitch's theorem [Sav70].

Büchi and coBüchi acceptance condition.

The case where the acceptance condition $\mathfrak{F}$ is coBüchi is trivial and not interesting (at least with respect to the emptiness problem), since every finite run is accepting. Let us fix a DPS $\mathcal{P}$ where $\mathfrak{F} =$ Büchi. We first explain what form accepting runs have, then give a slightly modified definition of interfaces in order to caracterize the existence of an accepting run as we did in Lemma 3.

Necessarily, we need to handle infinite runs, as finite runs cannot be accepting. Generally, we can distinguish three non-intersecting types of infinite runs, depending on how many processes perform infinitely many actions:

1. No process performs infinitely many actions, so there are infinitely many processes that perform finitely many actions each, or
2. Only one process performs infinitely many actions, or
3. At least two processes perform infinitely many actions each.

However, we can directly rule out case 3, as this case is incompatible with a round-bounded definition. Indeed, two (or more) processes performing infinitely many actions each necessarily means that they alternate an infinite number of times. Therefore the number of rounds cannot be bounded.

In case 2, as with the case of Reach only a finite amount of processes are involved in the run. The only difference is now that at some point after a finite part of the run, one of those processes keeps performing actions forever. For the Büchi condition to be satisfied, it means that this process must be able to repeatedly reach an accepting global state and local state.

In case 1, there is also a finite part of the run that is played with a finite amount of processes. However, during the last round of the run, after the last action of those processes, a new "phase" starts where a new process is created, performs some finite amount of actions, then another process is created, and so on. For this run to be

accepting, every process created must end in an accepting local state, and the global state must infinitely often be accepting.

Let us formalize these two cases.

Lemma 6. *Let ρ be a (necessarily infinite) accepting run of $\mathcal{P}$. Then there is an accepting extended configuration (c, p, r), a finite run ρ_{fin} ending in (c, p, r), and an infinite partial run ρ_{inf} such that either:*

1. *$\rho_{inf} = \prod_{i\geq 1}(c_i, p, r)$ for some $(c_i)_{i\geq 1}$, or*
2. *$\rho_{inf} = \prod_{i\geq 1} \rho_i$ where $\rho_i = \prod_{j=1}^{n(i)}(c_i^j, p_i, r)$ are finite runs such that all p_i are pairwise distinct processes that do not appear in ρ_{fin}.*

We say that a run is of type 1 if the first condition is satisfied (one process performs infinitely many actions), and type 2 if the second is satisfied (infinitely many processes perform finitely many actions each).

Proof. As sketched above, there are only two possible cases for ρ: either there is some point in the run after which only one process performs actions, or during the final round infinitely many processes join the execution.

In the first case, let $\rho = \rho' \cdot \rho''$ where ρ'' is the part of ρ forming the infinite partial run where only the last process performs actions. Since ρ is winning, it visits an accepting configuration infinitely often, therefore there are also an infinite amount of accepting configurations visited in ρ''. Then $\rho'' = \rho_1 \cdot \rho_2$ where ρ_1 ends in the first accepting configuration visited in ρ'' and ρ_2 is the rest of the run. We then let $\rho_{fin} = \rho' \cdot \rho_1$ and $\rho_{inf} = \rho_2$, which satisfy the condition of the lemma.

In the second case, we distinguish two cases. In the case where ρ forms only one round, since it is winning it visits accepting configurations infinitely often. We then take ρ_{fin} to be the prefix of ρ ending in the first accepting configuration visited, and ρ_{inf} is the rest of ρ. In the case where ρ spans over $r > 1$ rounds, let $P = \{p_1, \ldots, p_n\}$ be the set of processes involved in rounds 1 to $r-1$, and let $\rho = \rho' \cdot \rho''$ where ρ' ends in the last configuration (c, p, r) with $p \in P$. Now it is not necessarily true that c is accepting, but at least all local states of processes of P must be accepting, otherwise ρ could not be winning since those local states will not change anymore. Similarly, all processes created in ρ'' must also end in an accepting local state for the same reason. Moreover, as ρ is winning, then there are infinitely many accepting configurations visited in ρ''. With $\rho'' = \rho_1 \cdot \rho_2$ where ρ_1 ends in the first accepting configuration of ρ'', we then let $\rho_{fin} = \rho' \cdot \rho_1$ and $\rho_{inf} = \rho_2$, which satisfy the lemma. □

For runs of type 1, we need to know the *ending round* of each process, that is the round where the process performs its last action, in order to ensure that every process has stopped before the one that has to perform infinitely many actions starts doing it. Moreover, we need to know if that process can actually reach an accepting state infinitely often. Therefore we slightly modify the definition of an interface to take this into account:

Definition 11. A *Büchi-interface* is a tuple

$$\mathcal{I} = [r^{\mathrm{s}}, r^{\mathrm{e}}, (s_1, \ldots, s_B), (s'_1, \ldots, s'_B)] \in \{1, \ldots, B\}^2 \times S^B \times S^B$$

satisfying the following conditions:

1. For all $1 \leq r < r^{\mathrm{s}}$ and $r^{\mathrm{e}} < r \leq B$, we have $s_r = s'_r$.
2. There are local states $\ell_{r^{\mathrm{s}}-1}, \ldots, \ell_{r^{\mathrm{e}}} \in L$ and stack contents $\gamma_{r^{\mathrm{s}}-1}, \ldots, \gamma_{r^{\mathrm{e}}} \in \Gamma^*$ such that
   (i) for all $r^{\mathrm{s}} \leq r \leq r^{\mathrm{e}}$ there is a finite partial run of $\mathcal{P}$ from $c_r = (s_r, (\ell_{r-1}, \gamma_{r-1}))$ to $c'_r = (s'_r, (\ell_r, \gamma_r))$,
   (ii) this run has length at least two (i.e., it performs at least one transition) if $r = r^{\mathrm{s}}$,
   (iii) $\ell_{r^{\mathrm{s}}-1}$ is the initial local state ℓ^{init}, $\gamma_{r^{\mathrm{s}}-1}$ is the empty stack ε, and
   (iv) $\ell_{r^{\mathrm{e}}} \in F_{\mathsf{loc}}$ is an accepting local state.

Definition 12. We say that a run is a 1-process run if all configurations of that run are of size ≤ 1. Then, an *accepting interface* is a Büchi-interface with the following condition added: "(v) there is an accepting 1-process partial run of $\mathcal{P}$ starting from $c = (s'_{r^{\mathrm{e}}}, (\ell_{r^{\mathrm{e}}}, \gamma_{r^{\mathrm{e}}}))$."

Compatibility and valid sequences of Büchi-interfaces is defined analogously to interfaces. We now give two lemmas that characterize the two different types of accepting runs in terms of interfaces, as with Lemma 3:

Lemma 7. *There is an accepting B-bounded run of $\mathcal{P}$ of type 1 if and only if there is a valid sequence of k Büchi-interfaces $\mathcal{I}_1, \ldots, \mathcal{I}_k$ for $k \geq 1$ and there is $1 \leq p \leq k$ such that:*

- *for all $p' < p$, $r^e_{\mathcal{I}_{p'}} \leq r^e_{\mathcal{I}_p}$,*
- *for all $p' > p$, $r^e_{\mathcal{I}_{p'}} < r^e_{\mathcal{I}_p}$, and*
- *$\mathcal{I}_p$ is an accepting interface.*

Lemma 8. *There is an accepting B-bounded run of $\mathcal{P}$ of type 2 if and only if there is a valid sequence of k Büchi-interfaces $\mathcal{I}_1, \ldots, \mathcal{I}_k$ for $k \geq 1$ and there are k' Büchi-interfaces $\mathcal{I}'_1, \ldots, \mathcal{I}'_{k'}$ such that:*

- *for all $1 \leq p \leq k'$, $r^s_{\mathcal{I}'_p} = r^e_{\mathcal{I}'_p} = \max\{r^e_{\mathcal{I}_{p'}} \mid 1 \leq p' \leq k\}$,*
- *$\mathcal{I}_k$ is compatible with $\mathcal{I}'_1$,*
- *for all $1 < p \leq k'$, $\mathcal{I}_{p-1}$ is compatible with $\mathcal{I}_p$, and*
- *there are $p \neq p' \in \{1, \ldots, k'\}$ such that ${\mathcal{I}'_p}^{right} = {\mathcal{I}'_{p'}}^{right}$.*

These two lemmas are illustrated in Figure 3.4 and Figure 3.5 respectively. Both of their proofs rely on the previous Lemma 6 to decompose an accepting run into a finite and infinite part, with Lemma 3 taking care of the finite part.

Proof of Lemma 7 (sketch). Suppose ρ is an accepting B-bounded run of $\mathcal{P}$ of type 1. Let $\rho = \rho_{fin} \cdot \rho_{inf}$ given by Lemma 6 with ρ_{fin} ending in (c, p, r) which is accepting. From Lemma 3, we have that there is a valid sequence of k interfaces $\mathcal{I}_1, \ldots, \mathcal{I}_k$ corresponding to ρ_{fin}. Specifically, by construction, we have that for all $p' \leq p$ the ending round $r^{\mathrm{e}}_{\mathcal{I}_{p'}}$ of interface p' is lower or equal than r (and equal for $\mathcal{I}_p$), and that $r^{\mathrm{e}}_{\mathcal{I}_{p'}}$ is

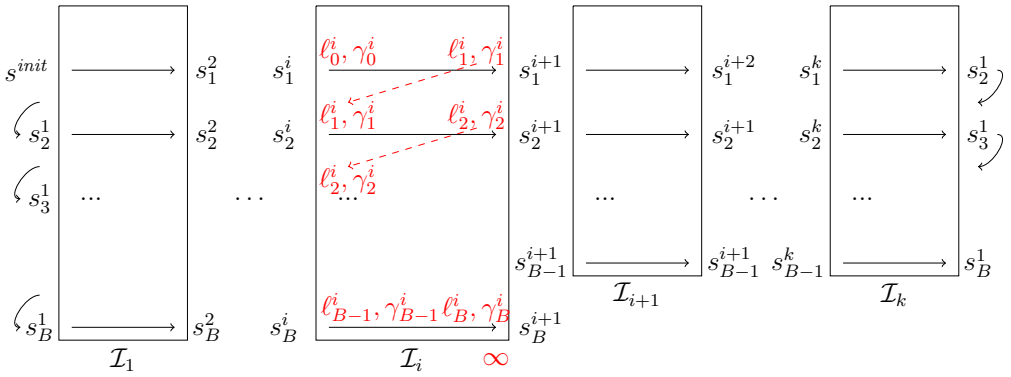

Figure 3.4: A run of type 1 as the composition of interfaces, here process i is the one that plays infinitely

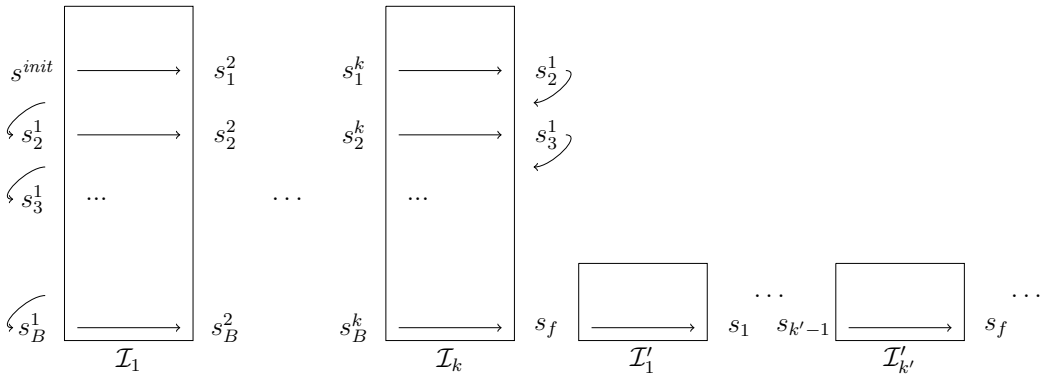

Figure 3.5: A run of type 2 as a composition of interfaces

strictly lower than r if $p' > p$. Moreover, we have that $c = (s'_{r^{\mathrm{e}}_{\mathcal{I}_p}}, \dots, (\ell_{r^{\mathrm{e}}_{\mathcal{I}_p}}, \gamma_{r^{\mathrm{e}}_{\mathcal{I}_p}}), \dots)$. Furthermore, since ρ_{inf} is a run starting from configuration c involving only process p, then it can be seen as a 1-process run from configuration $c' = (s'_{r^{\mathrm{e}}_{\mathcal{I}_p}}, (\ell_{r^{\mathrm{e}}_{\mathcal{I}_p}}, \gamma_{r^{\mathrm{e}}_{\mathcal{I}_p}}))$. Thus $\mathcal{I}_p$ is an accepting interface.

Now suppose that $\mathcal{I}_1, \dots, \mathcal{I}_k$ is a valid sequence of interfaces and let $p \leq k$ such that $\mathcal{I}_p$ is also an accepting interface. Again by Lemma 3, we can derive from $\mathcal{I}_1, \dots, \mathcal{I}_k$ a finite run ρ_{fin} ending in an accepting configuration (c, p', r) for some $p' \in \{1, \dots, k\}$ and $r \leq B$. Since p is the last process with the maximal ending round by hypothesis, then we can deduce that $p' = p$. Moreover, by construction of ρ_{fin} we have that $c = (s'_{r^{\mathrm{e}}_{\mathcal{I}_p}}, \dots, (\ell_{r^{\mathrm{e}}_{\mathcal{I}_p}}, \gamma_{r^{\mathrm{e}}_{\mathcal{I}_p}}), \dots)$. Using the fact that $\mathcal{I}_p$ is an accepting interface, it is then easy to build an infinite run ρ_{inf} starting from (c, p, r) that mimics the run given by $\mathcal{I}_p$. We then let $\rho = \rho_{fin} \cdot \rho_{inf}$ which is an accepting run. □

Proof of Lemma 8 (sketch). Suppose ρ is an accepting B-bounded run of $\mathcal{P}$ of type 2, and let $\rho = \rho_{fin} \cdot \rho_{inf}$ given by Lemma 6 with ρ_{fin} ending in (c, p, r). Then like in the previous proof, Lemma 3 gives us existence of a valid sequence of interfaces $\mathcal{I}_1, \dots, \mathcal{I}_k$ where k is the number of processes involved in ρ_{fin}. Now $\rho_{inf} = \prod_{i \geq 1} \rho_i$ where each ρ_i involve only one process that does not appear anywhere outside of ρ_i. Moreover, since ρ is accepting, then accepting configurations are visited infinitely often. In particular, since there are a finite number of global states, at least one must be visited infinitely often. Let s be one such state, and let $p < p'$ be two indices such that ρ_p and $\rho_{p'}$ end in s. We then let $\mathcal{I}'_i = [r, r, (s_{i-1}), (s_i)]$ for all $1 \leq i \leq p'$ where s_0 is the global state of c and s_i is the last global state of ρ_i. Then all conditions of the lemma are satisfied.

Conversely, suppose we have interfaces $\mathcal{I}_1, \dots, \mathcal{I}_k$ and $\mathcal{I}'_1, \dots, \mathcal{I}'_{k'}$ and that the conditions of the lemma are satisfied. With Lemma 3, we obtain from $\mathcal{I}_1$ to $\mathcal{I}_k$ a run ρ_{inf} ending in some configuration (c, p, r) such that by construction $r = \max\{r^{\mathrm{e}}_{\mathcal{I}_{p'}} \mid 1 \leq p' \leq k\}$. Let s_i for $i \leq k'$ such that $\mathcal{I}_i'^{right} = (s_i)$, and s_0 be the global state of c. From interfaces $\mathcal{I}'_i$, we can extract a run $\rho_i = c_i^1 \dots c_i^{j_i}$ from $c_i^1 = (s_{i-1}, (\ell^{init}, \varepsilon))$ to $c_i^{j_i} = (s_i, (\ell'_i, \gamma'_i))$ for some accepting local state ℓ'_i and some stack γ'_i. More precisely, for $i \leq k'$ and $j < j_i$, let $c_i^j = (s_i^j, (\ell_i^j, \gamma_i^j))$. With $c = (s_0, ((\ell_1, \gamma_1), \dots, (\ell_k, \gamma_k)))$, we then define

$$\bar{c}_i^j = (s_i^j, ((\ell_1, \gamma_1), \dots, (\ell_k, \gamma_k), (\ell'_1, \gamma'_1), \dots, (\ell'_{i-1}, \gamma'_{i-1}), (\ell_i^j, \gamma_i^j)))$$

and then $\bar{\rho}_i = (\bar{c}_i^2, k+i, r) \dots (\bar{c}_i^{j_i}, k+i, r)$. We then let

$$\rho_{inf} = \bar{\rho}_1 \cdot \dots \cdot \bar{\rho}_p \cdot (\bar{\rho}_{p+1} \cdot \dots \cdot \bar{\rho}_{p'})^\omega$$

which is a valid run that is also accepting. □

Checking that a tuple is a Büchi-interface can be reduced to the emptiness of a pushdown automaton exactly like Lemma 5, and checking for an accepting interface can be done similarly by reducing to the emptiness of a Büchi pushdown automaton. Then using the characterization of accepting runs with Büchi-interfaces given by the two previous lemmas, one can build an algorithm similar to the one given for Reach acceptance that works as follows.

First, the algorithm guesses whether to look for a type 1 or a type 2 run. In the first case, it starts guessing consecutive Büchi interfaces while only storing the left interface of the first one and the right interface of the last one as with the previous algorithm, but also stores the maximal ending round of these interfaces. At some point, it then guesses an accepting interface, and checks that the maximal ending round stored is lower than or equal to the ending round of the accepting interface. Then it stores that ending round, and continues guessing Büchi interfaces while checking that their ending round is strictly lower than the ending round of the accepting interface stored earlier. Finally, it decides to stop at some point, and checks that the right interface of the last one is compatible with the left interface of the first one to ensure that the sequence of interfaces guessed is valid.

In the second case, it guesses a valid sequence of Büchi interfaces as described in the Reach algorithm while also storing the maximal ending round encountered. Once this is done, it starts guessing consecutive compatible Büchi interfaces whose starting rounds and ending rounds are the maximum ending round, while at some point non-deterministically storing a right interface and checking that it appears again in another interface later.

Lemmas 7 and 8 prove the correctness of this algorithm, while the complexity is still overall in (N)PSPACE.

3.2.3 PSPACE-hardness of DFS-EMPTINESS$_{rb}$

We reduce from the non-emptiness of the intersection of a collection of finite automata $A_1, \ldots, A_n$, which is also known to be PSPACE-complete [Koz77].

Let $A_1, \ldots, A_n$ be n finite automata over a finite alphabet Σ. That is, $A_i = (Q^i, T^i, q_0^i, F^i)$ where $T^i \subseteq Q^i \times \Sigma \times Q^i$ is the transition relation. We denote by $\mathcal{L}(A_i) \subseteq \Sigma^*$ the language of A_i, which is defined as usual. The intersection problem asks whether there is a (nonempty) word $w \in \Sigma^+$ such that $w \in \mathcal{L}(A_i)$ for all $i \in \{1, \ldots, n\}$. Without loss of generality, let us assume that $q_0^i \notin F^i$ for all A_i.

The bound B on the number of rounds will be n, the number of automata. We construct a DFS that non-deterministically guesses a word w in the first round. Moreover, in round i, it will check that w is accepted by A_i. To do this, each process simulates exactly one transition of A_i on one letter of w. Each process performs one action each round, and, to ensure that the word w is the same for each A_i, stores the corresponding letter in its local state. The global state stores the state of the currently simulated automaton. If we get a final state at the end of each round, it means that each A_i accepts the same word w.

We use n copies of Σ plus two additional letters $\#$ and $\triangleright$: let $\Sigma' = \{a^i \mid a \in \Sigma, 1 \leq i \leq n\} \uplus \{\#, \triangleright\}$. Intuitively, the superscript i of a letter a^i denotes in which round this letter can be played, while $\#$ and $\triangleright$ are used to start the first round and go to the next round respectively. Formally, we define $\mathcal{F} = (M_{\mathsf{glob}}, M_{\mathsf{loc}}, F_{\mathsf{glob}}, F_{\mathsf{loc}}, \mathsf{Reach})$ with $M_{\mathsf{glob}} = (\Sigma', S, T_{\mathsf{glob}}, s^{init})$ and $M_{\mathsf{loc}} = (\Sigma', L, T_{\mathsf{loc}}, \ell^{init})$ as follows:

- $S = \{s^{init}\} \cup \Big(\big(\bigcup_{i \in \{1,\ldots,n\}} Q_i\big) \times \{1, \ldots, n\}\Big)$,
- $F_{\mathsf{glob}} = F^n \times \{n\}$,
- $L = \{\ell^{init}, \ell_\#\} \cup (\Sigma \times \{1, \ldots, n\})$,
- $F_{\mathsf{loc}} = \{\ell_\#\} \cup (\Sigma \times \{n\})$.

- Transitions are as follows:

T_{glob}	T_{loc}
$(s^{init}, \#, (q_0^1, 1))$	$(\ell^{init}, \#, \ell_\#)$
$((q,i), a^i, (q',i))$ if $(q,a,q') \in T^i$	$(\ell^{init}, a^1, (a,1))$ for $a \in \Sigma$
	$((a,i), a^{i+1}, (a, i+1))$ for $a \in \Sigma$
$((q_f,i), \triangleright, (q_0^{i+1}, i+1))$ if $q_f \in F^i$	$(\ell_\#, \triangleright, \ell_\#)$

The only technical point here is that the first process is only used to start the next round, and does not actually take part in simulating runs of the A_i. We also store the number of the round in the global state and each local state to ensure that every process plays exactly once in each round.

Starting from the initial configuration, the first transition necessarily has a label $\#$ as this is the only transition leaving global initial state s^{init}. The first process then has $\ell_\#$ for local state (and will be the only one with that state as no other $\#$ is ever played again). Then some number of a^1 transitions follow, spawning new processes for each such transition, each keeping a in their respective local states as well as the round number (which is 1). Since only letters of the form a_i can be played during round i, this prevents processes from playing more than once in the same round. Meanwhile the global state stores the current state of automaton A_1, as well as the current round so that only letters of this round can be played. At some point when it reaches a final state q_f, a new transition becomes available that starts the next round with a letter $\triangleright$, which only the first process can perform. This signals the end of the simulation of a run of A_1 and starts the second round of $\mathcal{F}$ and the simulation of a run of A_2, and this repeats again until A_n. Each time a process performs a transition in round i, its local state update the round from $i-1$ to i, and in order to be accepted, every process in a run must reach round n. This ensures that every process performs one transition each round. Finally, when a final state is reached in round n and all processes have played in that round, the run is accepted.

Lemma 9. *There is an accepting n-bounded run of $\mathcal{F}$ iff there is a non-empty word in $\bigcap_{1 \leq i \leq n} \mathcal{L}(A_i)$.*

Proof. $\boxed{\Leftarrow}$ Suppose that there is some word $w = a_1 \dots a_k$ of size $k > 0$ belonging to each $\mathcal{L}(A_i)$. For all $1 \leq i \leq n$, let $q_0^i \xrightarrow{a_1} \dots \xrightarrow{a_k} q_k^i$ be an accepting run of A_i. Then we define the following n-bounded run of $\mathcal{F}$:

$$\rho = ((s^{init}), 0, 1) \cdot \rho^1 \cdots \rho^n$$

with $\rho^i = (c_0^i, 1, i) \dots (c_k^i, k+1, i)$ where

$$c_j^1 = ((q_j^1, 1), \ell_\#, (a_1, 1), \dots (a_j, 1))$$

for all $0 \leq j \leq k$ and

$$c_j^i = ((q_j^i, i), \ell_\#, (a_1, i), \dots, (a_j, i), (a_{j+1}, i-1), \dots, (a_k, i-1))$$

for all $1 < i \leq n$ and $0 \leq j \leq k$. It is easily verifiable from the definitions that each c_j^i is a $(a_j^i, j+1)$-successor of c_{j-1}^i if $j > 0$ and that each c_0^i is a $(\triangleright, 1)$-successor of c_k^{i-1} for $i > 1$ using the fact that $q_k^i \in F_i$ for all $1 \leq i < n$. Therefore ρ is a valid run

of $\mathcal{F}$. It is accepting because $c_k^n = ((q_k^n, n), \ell_\#, (a_1, n), \ldots, (a_k, n))$ with $q_k^n \in F_n$ so $(q_k^n, n) \in F_{\mathsf{glob}}$, and $\ell_\#, (a_1, n), \ldots, (a_k, n) \in F_{\mathsf{loc}}$. It is also n-bounded because each ρ^i is one round.

$\boxed{\Rightarrow}$ Let ρ be an accepting n-bounded run of $\mathcal{F}$. By definition it can be split in $\rho = \rho^1 \cdots \rho^n$ where each ρ^i is the part of ρ in round i. Let us first focus on

$$\rho^1 = ((s^{init}), 0, 1)(c_0^1, p_0^1, 1)(c_1^1, p_1^1, 1) \ldots (c_k^1, p_k^1, 1)$$

We prove by induction on j that for all $0 \leq j \leq k$:

1. $p_j^1 = j + 1$, and
2. c_j^1 is of the form $((q_j^1, 1), \ell_\#, (a_1, 1), \ldots, (a_j, 1))$ with $q_j^1 \in Q^1$ and $a_1, \ldots, a_j \in \Sigma$ such that $q_0^1 \xrightarrow{a_1 \ldots a_j} q_j^1$ in A_1.

- For $j = 0$, the only transition exiting the initial global state s^{init} has label $\#$, therefore the only local transition possible for the first process is to go to local state $\ell_\#$, with global state going to $(q_0^1, 1)$. Hence $p_0^1 = 1$ and $c_0^1 = ((q_0^1, 1), \ell_\#)$ as expected.

- Suppose properties 1 and 2 are true for some $0 \leq j < k$, so $p_j^1 = j + 1$ and $c_j^1 = ((q_j^1, 1), \ell_\#, (a_1, 1), \ldots, (a_j, 1))$ with $q_0^1 \xrightarrow{a_1 \ldots a_j} q_j^1$ in A_1. For the next transition, since the round does not change, it means either the $j+1$-th process performs another transition, or a new process is spawned. The first case is not possible by construction, as either
  - $j = 0$, $c_0^1 = ((q_0^1, 1), \ell_\#)$, and from local state $\ell_\#$ the only transition is labeled by $\triangleright$, but there is no transition labeled by $\triangleright$ from global state $(q_0^1, 1)$ since q_0^1 is not final, or
  - $j > 0$, $c_j^1 = ((q_j^1, 1), \ldots, (a_j, 1))$, and only transitions labeled by a_j^2 are available from local state $(a_j, 1)$ but no transition labeled as such exit global state $(q_j^1, 1)$.

  Therefore there is some letter $a_{j+1} \in \Sigma$ and a state $q_{j+1}^1 \in Q^1$ such that $(q_j^1, a_{j+1}, q_{j+1}^1) \in T^1$, $c_{j+1}^1 = ((q_{j+1}^1, 1), \ell_\#, (a_1, 1), \ldots, (a_j, 1), (a_{j+1}, 1))$ and $p_{j+1} = j + 2$. Since $q_0^1 \xrightarrow{a_1 \ldots a_j} q_j^1$ and $(q_j^1, a_{j+1}, q_{j+1}^1) \in T^1$, then we obtain that $q_0^1 \xrightarrow{a_1 \ldots a_{j+1}} q_{j+1}^1$ as expected.

Finally, we have that $q_k^1 \in F^1$, because either

- $n = 1$ and since ρ is accepting then necessarily $(q_k^1, 1) \in F^1 \times \{1\}$, or
- $n > 1$ and there is transition from $(c_k^1, k + 1, 1)$ the last configuration of ρ^1 to the first configuration of ρ^2 which is of the form $(c_0^2, p_0^2, 2)$, and the only way to start a new round is with a $\triangleright$-labeled transition which can only happen if the global state is of the form $(q_k^1, 1)$ with q_k^1 an accepting state of A_1.

Therefore we have that $q_0^1 \xrightarrow{a_1 \ldots a_k} q_k^1 \in F^1$, so $w = a_1 \ldots a_k \in \mathcal{L}(A_1)$.

Using a similar reasoning, we can prove that $w \in \mathcal{L}(A_i)$ for all $1 < i < n$. To prove that in a given round $i > 1$ each process plays exactly once and in the same order as round 1, the following points are needed:

- New processes cannot be spawned, because that would require a transition with a label of the form a^1 which is the only kind of transition available from the initial local state ℓ^{init}, but a global state of the form (q, i) for $i > 1$ has no available transition with such a label.

- A process cannot play more that once with the same arguments than in round 1.

- Suppose that process $j + 1$ is does not perform any transition in round i. Its local state at the beginning of the round is of the form $(a_j, i-1)$, and the only transition available from this local state has label a^i. Transitions with labels of the form a^i are unavailable outside of round i, because the global state needs to be of the form (q, i). Therefore, process $j + 1$ stays in local state $(a_j, i - 1)$ for the rest of the run. Since this local state is not accepting, then ρ is not accepting too, contradicting the initial hypothesis.

Moreover, as each process stored in round 1 the letter that was used, only the same letter can be used again, ensuring that the same word is simulated in each round. Therefore $w \in \bigcap_{1 \leq i \leq n} \mathcal{L}(A_i)$. □

This concludes the proof of Theorem 2. Now that we established the complexity for the emptiness problem, the next section is dedicated to study the control problem.

3.3 Control of Dynamic Pushdown Games

The emptiness problem is about finding an accepting run of a DPS when one is in control in every step of the run. The natural question is now: what happens if there is an uncontrollable part in the system? Remember that our goal is to build a controller for systems embedded into an uncontrollable environment.

Therefore, we extend dynamic pushdown systems to a game-based setting in order to better model interactions between the controller and the environment. Since the emptiness problem is already too hard for unbounded DPS, in this section we will only study round-bounded DPS.

3.3.1 Dynamic Pushdown Games

Here we define dynamic pushdown games, which are games played on dynamic pushdown systems. A partition of the global states is given into System global states and Environment global states, which leads us to distinguish System configurations from Environment configurations. Let $\mathcal{P} = (M, P, F_{\mathsf{glob}}, F_{\mathsf{loc}}, \mathfrak{F})$ be a DPS with $M = (\Sigma, S, T_{\mathsf{glob}}, s^{init})$ and $P = (\Sigma, \Gamma, L, T_{\mathsf{loc}}, \ell^{init})$.

Definition 13. A *dynamic pushdown game* (DPG) is a tuple $\mathcal{G} = (\mathcal{P}, S^{\mathsf{s}}, S^{\mathsf{e}})$ where $S = S^{\mathsf{s}} \uplus S^{\mathsf{e}}$ is a partition of global states of $\mathcal{P}$ into *System* global states and *Environment* global states.

The semantics of a DPG $\mathcal{G}$ is a game, as defined in Definition 5, which is the tuple $(\mathcal{T}_{\mathcal{P}}, \mathsf{Conf}^{\mathsf{s}}(\mathcal{P}), \mathsf{Conf}^{\mathsf{e}}(\mathcal{P}), \mathsf{Acc}(F_{\mathsf{glob}}, F_{\mathsf{loc}}, \mathfrak{F}))$ such that $\mathcal{T}_{\mathcal{P}}$ is the transition system associated with $\mathcal{P}$, where a configuration $c = (s, (\ell_1, \gamma_1), \ldots, (\ell_n, \gamma_n))$ is in $\mathsf{Conf}^{\mathsf{s}}(\mathcal{P})$

if $s \in S^{\mathsf{s}}$ and in $\mathsf{Conf}^{\mathsf{e}}(\mathcal{P})$ otherwise, and $\mathsf{Acc}(F_{\mathsf{glob}}, F_{\mathsf{loc}}, \mathfrak{F})$ is the acceptance condition of $\mathcal{P}$. Dynamic finite games (DFG) are defined similarly with a DFS instead of a DPS.

We can then define the control problem for this specific kind of games:

CONTROL-DPG	
Input:	A DPG $\mathcal{G}$
Question:	Is there a winning strategy for System in $\mathcal{G}$?

CONTROL-DFG	
Input:	A DFG $\mathcal{G}$
Question:	Is there a winning strategy for System in $\mathcal{G}$?

Since the emptiness problem for DPS is already undecidable, this implies that the control problem is also undecidable, as emptiness can be seen as a special kind of control where System controls everything. The situation is not much better for control of DFG, which is also undecidable in the absence of bounds. Indeed, DFS are equivalent to VASS (see proof of Theorem 1), and it has been shown that games on VASS are undecidable. [ABd03]

Therefore, as we did for emptiness, we now define the round-bounded alternative for these games.

Definition 14. Let $\mathcal{G} = (\mathcal{P}, S^{\mathsf{s}}, S^{\mathsf{e}})$ be a DPG as defined above, and $B \in \mathbb{N}_{>}$ be a bound. Then by $\mathcal{G}^B$ we denote the game $(\mathcal{P}^B, \mathsf{Conf}^{\mathsf{s}}(\mathcal{P}), \mathsf{Conf}^{\mathsf{e}}(\mathcal{P}), \mathsf{Acc})$ which is played on the B-round bounded semantics of $\mathcal{P}$, and where the definition of $\mathsf{Conf}^{\mathsf{s}}(\mathcal{P})$, $\mathsf{Conf}^{\mathsf{e}}(\mathcal{P})$, and Acc are lifted to extended configurations.

Round-bounded DFG are defined similarly as well.

Example 17. We give an example illustrated in Figure 3.6. The idea is as follows: there is a finite set of different task kinds, and processes may receive tasks from this set to be executed. Once all tasks have been received, the processes must then execute them, in reverse order. Finally, when all processes have executed all of their tasks, they must all shutdown.

In the game given in the illustration, in the first round Environment plays and can *start* new processes, distribute any number of tasks with the associated $task_i$ action (and possibly give multiple copies of the same task kind to the same process), and then do that again to a fresh process, and so on, until she decides to stop (by doing a *nop* action on a fresh process). Then the second round starts, where System must do an *exec* action on every process. If she fails to do so, then Environment will be able to do a *fail* action and the run stops immediately. Otherwise, the third round starts and System must execute the tasks each process received in the first round with the corresponding $exec_i$. Only when all tasks have been removed for all processes can the fourth and last round start, in which all processes do a *halt* action and then stop, and then System only does *nop* actions on fresh processes to get an infinite winning run. Note that if Environment never stops creating new processes and distributing tasks in the first round, then the infinite run is also winning for System.

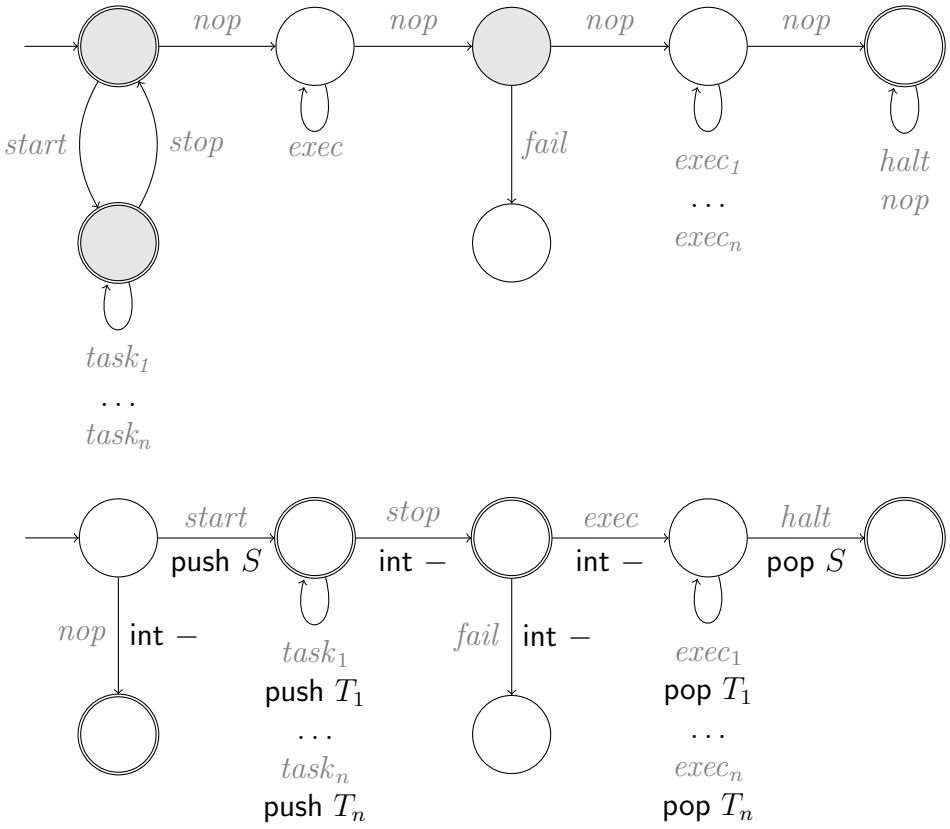

Figure 3.6: Example of a DPG, with the global finite state machine (above) and the local pushdown machine (below). Global states of Environment have a gray background, the others belong to System. The winning condition is Büchi.

Then as long as $B \geq 4$, there is a winning strategy for System, which simply consists in doing all *exec* in the second round in the correct order, executing multiple $task_i$ in the third round, and only doing a single *halt* when every process has no tasks remaining. However, it is easy to see that there is no winning strategy if $B \leq 3$ in this example.

We again define the control problem for the bounded version of games:

CONTROL-DPG$_{rb}$	
Input:	A DPG $\mathcal{G}$, a bound $B \in \mathbb{N}_>$ (given in unary)
Question:	Is there a winning strategy for System in $\mathcal{G}^B$?

CONTROL-DFG$_{rb}$	
Input:	A DFG $\mathcal{G}$, a bound $B \in \mathbb{N}_>$ (given in unary)
Question:	Is there a winning strategy for System in $\mathcal{G}^B$?

Remember that for such games, if there is a winning strategy then there is one that is memoryless. We prove that the problem CONTROL-DPG$_{rb}$ is decidable, and that CONTROL-DFG$_{rb}$ has an inherent non-elementary complexity.

Theorem 10. *CONTROL-DPG$_{rb}$ and CONTROL-DFG$_{rb}$ are decidable and inherently non-elementary.*

The proof of this theorem is the subject of the next two sections. Section 3.3.2 presents an algorithm for CONTROL-DPG$_{rb}$ via a reduction to phase-bounded multi-pushdown games [Set09] to show decidability. Section 3.3.3 gives a hardness proof with a reduction from the satisfiability problem for first-order formulas on finite words.

3.3.2 Upper Bound

Decidability of CONTROL-DPG$_{rb}$ comes from decidability of games on phase-bounded multi-pushdown systems (short: multi-pushdown games), which were first studied in [Set09] and rely on the phase-bounded multi-pushdown automata from [LMP07].

Multi-Pushdown Games. A multi-pushdown system is a collection of a *fixed* number of stacks, on which with the usual pop, push, int as well a a zero-test zero action can be performed, with a state from a finite set which is shared by all stacks and can be updated when performing actions.

Intuitively, a *phase* is a sequence of actions in a run during which only one fixed "active" stack can be read (i.e., either make a pop transition or a zero-test transition), but push and internal transitions are unrestricted. There are no other constraints on the number of transitions or the order of the transitions done during a phase.

Definition 15. A *multi-pushdown system (MPS)* is a tuple

$$\mathcal{M} = (\kappa, N, S, \Gamma, \Delta, s^{init})$$

where the natural number $\kappa \geq 1$ is the *phase bound*, $N \in \mathbb{N}$ is the number of stacks, S is the finite set of *states*, Γ is the finite *stack alphabet*, $\Delta \subseteq S \times \mathsf{Act}_{\mathsf{zero}} \times \{1, \ldots, N\} \times \Gamma \times S$ is the *transition relation* where $\mathsf{Act}_{\mathsf{zero}} = \{\mathsf{push}, \mathsf{pop}, \mathsf{int}, \mathsf{zero}\}$, and $s^{init} \in S$ is the initial state.

The semantics of a MPS is the transition system $\mathcal{T}_{\mathcal{M}} = (V, E, v^{init})$ defined as follows. A node $v \in V$ is of the form

$$v = (s, \gamma_1, \ldots, \gamma_N, st, ph)$$

where

- $s \in S$ is the state,
- $\gamma_\sigma \in \Gamma^*$ is the content of stack σ,
- $st \in \{0, \ldots, N\}$ and $ph \in \{1, \ldots, \kappa\}$ are used to keep track of the current active stack (0 when it is undefined) and the current phase, respectively.

Given $v = (s, \gamma_1, \ldots, \gamma_N, st, ph) \in V$ and $v' = (s', \gamma'_1, \ldots, \gamma'_N, st', ph') \in V$, we have an edge $(v, v') \in E$ if and only if there exist $op \in \mathsf{Act}_{\mathsf{zero}}$, $\sigma \in \{1, \ldots, N\}$, and $A \in \Gamma$ such that $(s, op, \sigma, A, s') \in \Delta$ and the following hold:

- $\gamma_\tau = \gamma'_\tau$ for all $\tau \neq \sigma$,
- $\gamma_\sigma = \gamma'_\sigma$ if $op = \mathsf{int}$, $\gamma'_\sigma = A \cdot \gamma_\sigma$ if $op = \mathsf{push}$, $\gamma_\sigma = A \cdot \gamma'_\sigma$ if $op = \mathsf{pop}$, and $\gamma_\sigma = \gamma'_\sigma = \varepsilon$ if $op = \mathsf{zero}$,
- if $op \in \{\mathsf{int}, \mathsf{push}\}$, then $st = st'$ and $ph = ph'$ (the active stack and, hence, the phase do not change),
- if $op \in \{\mathsf{pop}, \mathsf{zero}\}$, then,
  - either $\sigma = st$ (σ is the current active stack), and $st = st'$ and $ph = ph'$,
  - or $st = 0$, and $st' = \sigma$ and $ph' = ph = 1$,
  - or $st \notin \{0, \sigma\}$ and $ph < \kappa$, and $st' = \sigma$ and $ph' = ph + 1$.

  Observe that, if $st = 0$, by definition, $ph = 1$, and σ is the first active stack to be defined. Moreover, if σ was not the current active stack, then a new phase starts (if possible).

We also note $s \xrightarrow{op,\sigma,A} s'$ for a transition (s, op, σ, A, s'), and we may write $-$ instead of A if the stack symbol does not matter, i.e. for int and zero transitions.

Finally, the initial node is $v^{init} = (s^{init}, \varepsilon, \dots, \varepsilon, 0, 1)$, i.e. we start from the initial state with all stacks empty.

Now we define multi-pushdown games which are simply games played on MPS. Let $\mathcal{M} = (\kappa, N, S, \Gamma, \Delta, s^{init})$ be a MPS as defined above.

Definition 16. A *multi-pushdown game* (MPG) over $\mathcal{M}$ is a tuple $\mathfrak{G}_\mathcal{M} = (S_0, S_1, \alpha)$ where $S = S_0 \uplus S_1$ is a partition of the set of states, and $\alpha : S \to Col$ with $Col \subseteq \mathbb{N}$ a finite set is the *ranking function*.

Its semantics is the game $(\mathcal{T}_\mathcal{M}, V_0, V_1, \mathsf{Acc})$ where $V_j = \{(s, \gamma_1, \dots, \gamma_N, st, ph) \in V \mid s \in S_j\}$ for $j \in \{0, 1\}$ is the partition of nodes induced by the partition of states, and the acceptance condition is a *parity condition* given by α:

$$\mathsf{Acc} = \{\rho \in V^\omega \mid \mathsf{min}(\mathsf{Inf}_\alpha(\rho)) \text{ is even}\}$$

where $\mathsf{Inf}_\alpha((s_0, \dots) \cdot (s_1, \dots) \cdot \dots) = \{c \in Col \mid c \text{ appears infinitely often in the sequence } \alpha(s_0)\alpha(s_1)\dots\}$. In other words, we look at the sequence of *colors* (as given by the ranking function α) encountered during the run, and if the smallest color that is encountered infinitely often is even then the run is winning.

We note CONTROL-MPG$_{pb}$ the phase-bounded control problem for MPG:

CONTROL-MPG$_{pb}$	
Input:	A MPG $\mathfrak{G}_\mathcal{M}$
Question:	Is there a winning strategy in $\mathfrak{G}_\mathcal{M}$?

Theorem 11 ([Set09, ABKS17]). *CONTROL-MPG$_{pb}$ is decidable, and is non-elementary in the number of phases.*

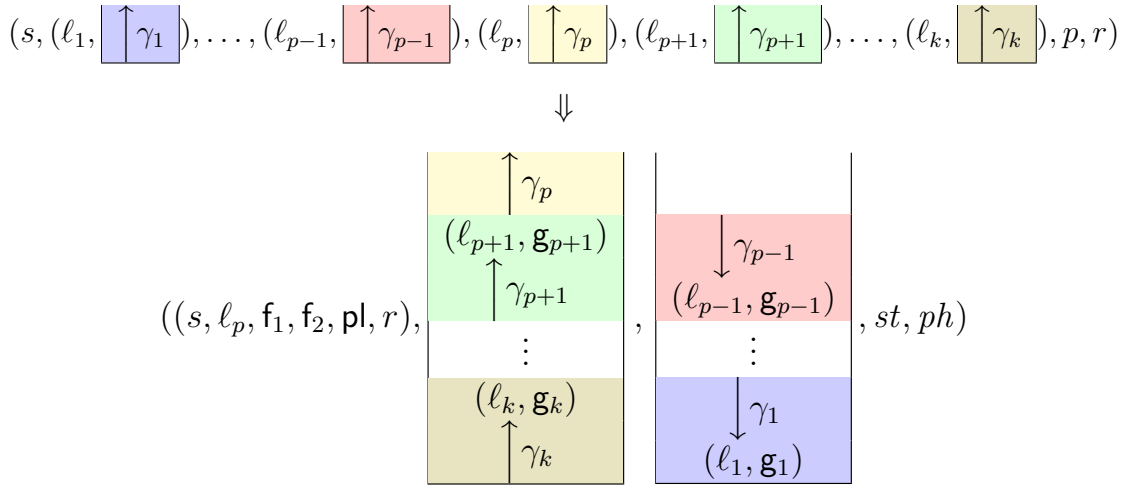

Figure 3.7: Encoding of a configuration in $\mathcal{G}^B$ by a configuration in $\mathfrak{G}_\mathcal{M}$

The upper bound was first shown in [Set09] by adopting the technique from [Wal01], which reduces pushdown games to games played on finite-state arenas. On the other hand, [ABKS17] proceeds by induction on the number of phases, reducing a $(\kappa+1)$-phase game to a κ-phase game.

Similarly, we could try a direct proof of our Theorem 10 by induction on the number of rounds. However, this proof would be very technical and essentially reduce round-bounded parameterized systems to multi-pushdown systems. Therefore, we proceed by reduction to multi-pushdown games, providing a modular proof with clearly separated parts.

From Parameterized Pushdown Games to Multi-Pushdown Games. We reduce the problem CONTROL-DPG$_{rb}$ to CONTROL-MPG$_{pb}$. Let $B \in \mathbb{N}_>$ be a bound and $\mathcal{G} = (\mathcal{P}, S^{\mathsf{s}}, S^{\mathsf{e}})$ be a DPG where $\mathcal{P} = (M, P, F_{\mathsf{glob}}, F_{\mathsf{loc}}, \mathfrak{F})$ with $M = (\Sigma, S, T_{\mathsf{glob}}, s^{init})$ and $P = (\Sigma, \Gamma, L, T_{\mathsf{loc}}, \ell^{init})$.

We build an MPG $\mathfrak{G}_\mathcal{M}$ over an MPS $\mathcal{M}$ with two stacks such that System has a winning strategy in $\mathcal{G}^B$ if and only if System has a winning strategy in $\mathfrak{G}_\mathcal{M}$.

To alleviate possible confusions between the two kinds of games, players in the multi-pushdown game $\mathfrak{G}_\mathcal{M}$ will be refered to as Player 0 and Player 1 instead of System and Environment respectively. In the following, given a global state $s \in S$ of the DPG, we let $pl(s) \in \{0,1\}$ denote the player associated with s, i.e., $pl(s) = 0$ if and only if $s \in S^{\mathsf{s}}$. Furthermore, if $\mathsf{pl} \in \{0,1\}$ stands for a player, then $\overline{\mathsf{pl}} = 1 - \mathsf{pl}$ stands for the other player.

The main idea of the reduction is to represent a configuration of $\mathcal{G}^B$ as a node in $\mathfrak{G}_\mathcal{M}$ as depicted in Figure 3.7.

Stack contents of process p and of all processes $p' > p$ are stored in the first stack of the MPG, while the stack contents of processes $p' < p$ are stored *in reverse order* on the second stack. Component $\mathsf{pl} \in \{0,1\}$ of the node's state denotes the current player. By default, it is $pl(s)$, that is, Player 0 controls the node of $\mathfrak{G}_\mathcal{M}$ if it simulates a configuration controlled by System in $\mathcal{G}^B$, and vice-versa (we will describe an exception to this rule later). We explain f_1 and f_2 further below.

The process p that has moved last is considered as the *active* process whose

local state ℓ_p is kept in the state of $\mathfrak{G}_{\mathcal{M}}$ along with s, and whose stack content γ_p is accessible on stack 1 (in the correct order). This allows the multi-pushdown game to simulate transitions of process p, modifying its local state and stack contents accordingly (see *Basic Transitions* in the formalization below).

If a player decides to take a transition for some process $p' > p$, she will store ℓ_p on stack 2 and shift the contents of stack 1 onto stack 2 until she retrieves the local state $\ell_{p'}$ of p' along with its stack contents $\gamma_{p'}$ (see Figure 3.8 and *Transitions for Process Change* in the formalization of $\mathcal{M}$).

If, on the other hand, the player decides to take a transition for some process $p' < p$, then she stores ℓ_p on stack 1 and shifts the contents of stack 2 onto stack 1 to recover the local state $\ell_{p'}$ and stack contents $\gamma_{p'}$ (see Figure 3.9 and *Transitions for Round Change*). This may imply two phase switches, one to shift stack symbols from 2 to 1, and another one to continue simulating the current process on stack 1. However, $2B - 1$ phases are sufficient to simulate B rounds.

There are a few subtleties: First, at any time, we need to know whether the current node of $\mathfrak{G}_{\mathcal{M}}$ corresponds to an accepting configuration in $\mathcal{G}^B$. To this aim, the state component $(s, \ell_p, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r)$ of $\mathcal{M}$ contains the flags $\mathsf{f}_1, \mathsf{f}_2 \in \{\checkmark, \times\}$ where, as an invariant, we maintain $\mathsf{f}_1 = \checkmark$ if and only if $\{\ell_{p+1}, \dots, \ell_k\} \subseteq F_{\mathsf{loc}}$ and $\mathsf{f}_2 = \checkmark$ if and only if $\{\ell_1, \dots, \ell_{p-1}\} \subseteq F_{\mathsf{loc}}$. Thus, a node of $\mathfrak{G}_{\mathcal{M}}$ corresponds to an accepting configuration of $\mathcal{G}^B$ if its state is of the form $(s, \ell, \checkmark, \checkmark, \mathsf{pl}, r)$ with $s \in F_{\mathsf{glob}}$ and $\ell \in F_{\mathsf{loc}}$. To faithfully maintain the invariant, every local state ℓ_q that is pushed on one of the two stacks, comes with an additional flag $\mathsf{g}_q \in \{\checkmark, \times\}$, which is $\checkmark$ if and only if all local states *strictly below* on the stack are contained in F_{loc}. It is then possible to keep track of a property of *all* local states on a given stack simply by inspecting and locally updating the topmost stack symbols.

Second, one single transition in $\mathcal{G}$ is potentially simulated by several transitions in $\mathcal{M}$ in terms of the gadgets given in Figures 3.8 and 3.9. The problem here is that once Player pl commits to taking a transition by entering a gadget, she is not allowed to get stuck. Otherwise, the simulation would end abruptly and Player 1 would win the game (because the play is finite), while it does not necessarily means that Environment wins in $\mathcal{G}$. Threfore to ensure progress, there are transitions from inside a gadget to a sink state $\mathrm{win}_{\bar{\mathsf{pl}}}$ that is winning for Player $\bar{\mathsf{pl}}$.

Third, suppose that, in a non-final configuration of $\mathcal{G}^B$, it is Environment's turn, but no transition is available. Then, Environment wins the play. But how can Player 1 prove in $\mathfrak{G}_{\mathcal{M}}$ that no transition is available in the original game $\mathcal{G}^B$? Actually, she will give the control to Player 0, who will eventually get stuck and, therefore, lose (cf. transitions for *Change of Player* below).

Let us define the MPS $\mathcal{M} = (\kappa, N, S_M, \Gamma_M, \Delta, s_M^{init})$ and the MPG $\mathfrak{G}_{\mathcal{M}} = (S_M^0, S_M^1, \alpha)$ formally. We let $\kappa = 2B - 1$ (the maximal number of phases needed), $N = 2$ (the number of stacks), and $\Gamma_M = \Gamma \uplus (L \times \{\checkmark, \times\})$.

States. The set of states is $S_M = \{s_M^{init}\} \uplus S_{\mathrm{sim}} \uplus \{\mathrm{win}_0, \mathrm{win}_1\} \uplus \mathfrak{I}$ where s_M^{init} is the initial state. Moreover, $S_{\mathrm{sim}} = S \times L \times \{\checkmark, \times\}^2 \times \{0, 1\} \times \{1, \dots, B\}$. A state $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \in S_{\mathrm{sim}}$ stores the global state s and the local state ℓ of the last process p that executed a transition. The third and forth component f_1 and f_2 tell us whether all processes $p' > p$ and, respectively, $p' < p$ of the current configuration are in a local final state (indicated by $\checkmark$). Then, pl denotes the player that is about to play

(usually, we have $\mathsf{pl} = pl(s)$, but there will be deviations as we said earlier). Finally, r is the current round that is simulated. Recall that $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r)$ represents an accepting configuration of $\mathcal{G}$ if and only if $s \in F_{\mathsf{glob}}$, $\ell \in F_{\mathsf{loc}}$, and $\mathsf{f}_1 = \mathsf{f}_2 = \checkmark$. Let $\mathfrak{W} \subseteq S_{\mathrm{sim}}$ be the set of such states. The states win_0 and win_1 are self-explanatory. Finally, we use several intermediate states, contained in $\mathfrak{I}$, which will be determined below along with the transitions.

The partition $S_M = S_M^0 \uplus S_M^1$ is defined as follows: First, we have $s_M^{init} \in S_M^0$ iff s^{init} belongs to System. Concerning states from S_{sim}, we let $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \in S_M^{\mathsf{pl}}$. The sink states win_0 and win_1 both belong to Player 0 (but this does not really matter). Membership of intermediate states is defined below as they are introduced.

Ranking function and end of the game. Depending on whether the acceptance type $\mathfrak{F}$ of the DPG is Reach, Büchi, or coBüchi, there will be slight changes to the game $\mathfrak{G}_{\mathcal{M}}$. In all three cases, win_i is a sink state that is winning for Player i for $i \in \{0, 1\}$.

- If $\mathfrak{F} = \mathsf{Reach}$:
  For all states $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \in \mathfrak{W}$, we will have a transition $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{int},1,-} \mathrm{win}_0$, which will be the only transition outgoing from $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r)$. Then the ranking function α maps win_0 to 0 and everything else to 1.

- If $\mathfrak{F} = \mathsf{Büchi}$:
  We simply define α as the function that maps every state in $\mathfrak{W}$ and win_0 to 0, with everything else mapped to 1. In this case, we do not add any special transition from a node with state in $\mathfrak{W}$ to win_0.

- If $\mathfrak{F} = \mathsf{coBüchi}$:
  Here, α maps every state in $\mathfrak{W}$ and win_1 to 1, and every other state to 0. Similarly, no transition is added in this case.

If $\mathfrak{F} = \mathsf{Reach}$, we do not want any transition other than the one we added exiting a state in $\mathfrak{W}$. So we need to be careful not to add any other transition that we would otherwise add for states outside $\mathfrak{W}$ or for the two other acceptance conditions. To that end, let us define

$$S^*_{\mathrm{sim}} = \begin{cases} S_{\mathrm{sim}} \setminus \mathfrak{W} & \text{if } \mathfrak{F} = \mathsf{Reach}, \\ S_{\mathrm{sim}} & \text{otherwise.} \end{cases}$$

Initial Transitions. For all $a \in \Sigma$, for all transitions of the form $(s^{init}, a, s') \in T_{\mathsf{glob}}$ and $(\ell^{init}, (a, op, A), \ell') \in T_{\mathsf{loc}}$ in $\mathcal{G}^B$, we introduce in $\mathcal{M}$ a transition $s_M^{init} \xrightarrow{op,1,A} (s', \ell', \checkmark, \checkmark, pl(s'), 1) \in \Delta$.

Basic Transitions. We now define the transitions of $\mathcal{M}$ simulating transitions of $\mathcal{P}$ that do not change the process. For all states $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \in S^*_{\mathrm{sim}}$ in $\mathcal{M}$, and transitions $(s, a, s') \in T_{\mathsf{glob}}$ and $(\ell, (a, op, A), \ell') \in T_{\mathsf{loc}}$ in $\mathcal{P}$, there is a transition $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{op,1,A} (s', \ell', \mathsf{f}_1, \mathsf{f}_2, pl(s'), r) \in \Delta$ in $\mathcal{M}$.

Change of Player. As we have said, when a player enters a gadget to simulate a change of round or player, she is committed to complete the change. If no transition in the original game is available from a configuration belonging to Player 1, in the multi-pushdown game, that same player will have no choice but eventually taking a transition leading to win_0, allowing Player 0 to win the game $\mathfrak{G}_{\mathcal{M}}$. However, if the blocking configuration was not winning in $\mathcal{G}^B$, Player 1 should win the game. To get around this discrepancy, when Player 1 thinks she does not have an outgoing transition (in $\mathcal{P}$), she can give the token to Player 0. That is, for all $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, 1, r) \in S^*_{\mathrm{sim}}$, we introduce the transition $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, 1, r) \xrightarrow{\mathsf{int},1,-} (s, \ell, \mathsf{f}_1, \mathsf{f}_2, 0, r) \in \Delta$.

Transitions for Process Change. We define the sets $\mathfrak{I}_?, \mathfrak{I}_{\mathrm{NP}}, \mathfrak{I}_{\mathrm{NR}}$ used to change the active process.

Within the same round:

For all $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \in S^*_{\mathrm{sim}}$, we introduce, in $\mathcal{M}$, the gadget given in Figure 3.8. As we move to another process, the current local state ℓ is pushed on stack 2, along with flag f_2, which tells us whether, henceforth, all states on stack 2 *below* the new stack symbol are local accepting states. Afterwards, the value of f_2 kept in the global state has to be updated, depending on whether $\ell \in F_{\mathsf{loc}}$ or not. Actually, maintaining the value of f_2 is done in terms of additional (but finitely many) states. For the sake of readability, however, we rather consider that f_2 is a variable and use $\mathsf{upd}(\mathsf{f}_2, \ell)$ to update its value. We continue shifting the contents of stack 1 onto stack 2 (updating f_2 when retrieving a local state). Now, there are two possibilities. We may eventually pop a new current local state $\hat{\ell}$ and then simulate the transition of the corresponding existing process. Or, when there are no more symbols on stack 1, we create a new process.

Formally, we have the set $\mathfrak{I}_{\mathrm{NP}}$ of intermediate states for a process change. For every $A \in \Gamma$, $s \in S$, $\ell \in L$, $\mathsf{f}_1, \mathsf{f}_2, \mathsf{g} \in \{\checkmark, \times\}$, $\mathsf{pl} \in \{0, 1\}$, and $r \leq B$ such that $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \in S^*_{\mathrm{sim}}$, we include $\mathrm{np}(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r)$, $\mathrm{np}^A(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r)$, and $\mathrm{np}^\ell(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r)$ in $\mathfrak{I}_{\mathrm{NP}}$, and we add the transitions

1. $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{push},2,(\ell,\mathsf{f}_2)} \mathrm{np}(s, \mathsf{f}_1, \mathsf{upd}(\mathsf{f}_2, \ell), \mathsf{pl}, r)$,
2. $\mathrm{np}(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{pop},1,A} \mathrm{np}^A(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r)$,
3. $\mathrm{np}^A(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{push},2,A} \mathrm{np}(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r)$,
4. $\mathrm{np}(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{pop},1,(\ell,\mathsf{g})} \mathrm{np}^\ell(s, \mathsf{g}, \mathsf{f}_2, \mathsf{pl}, r)$,
5. $\mathrm{np}^\ell(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{push},2,(\ell,\mathsf{f}_2)} \mathrm{np}(s, \mathsf{f}_1, \mathsf{upd}(\mathsf{f}_2, \ell), \mathsf{pl}, r)$,
6. $\mathrm{np}(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{pop},1,(\ell,\mathsf{g})} ?(s, \ell, \mathsf{g}, \mathsf{f}_2, \mathsf{pl}, r)$, and
7. $\mathrm{np}(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{zero},1,-} ?(s, \ell^{init}, \checkmark, \mathsf{f}_2, \mathsf{pl}, r)$

where $\mathsf{upd}(\mathsf{f}_2, \ell) = \checkmark$ iff $\mathsf{f}_2 = \checkmark \wedge \ell \in F_{\mathsf{loc}}$. States of the form $?(s, \ell, \mathsf{g}, \mathsf{f}_2, \mathsf{pl}, r)$ are used to exit this gadget, and will be defined shortly after.

In the next round:

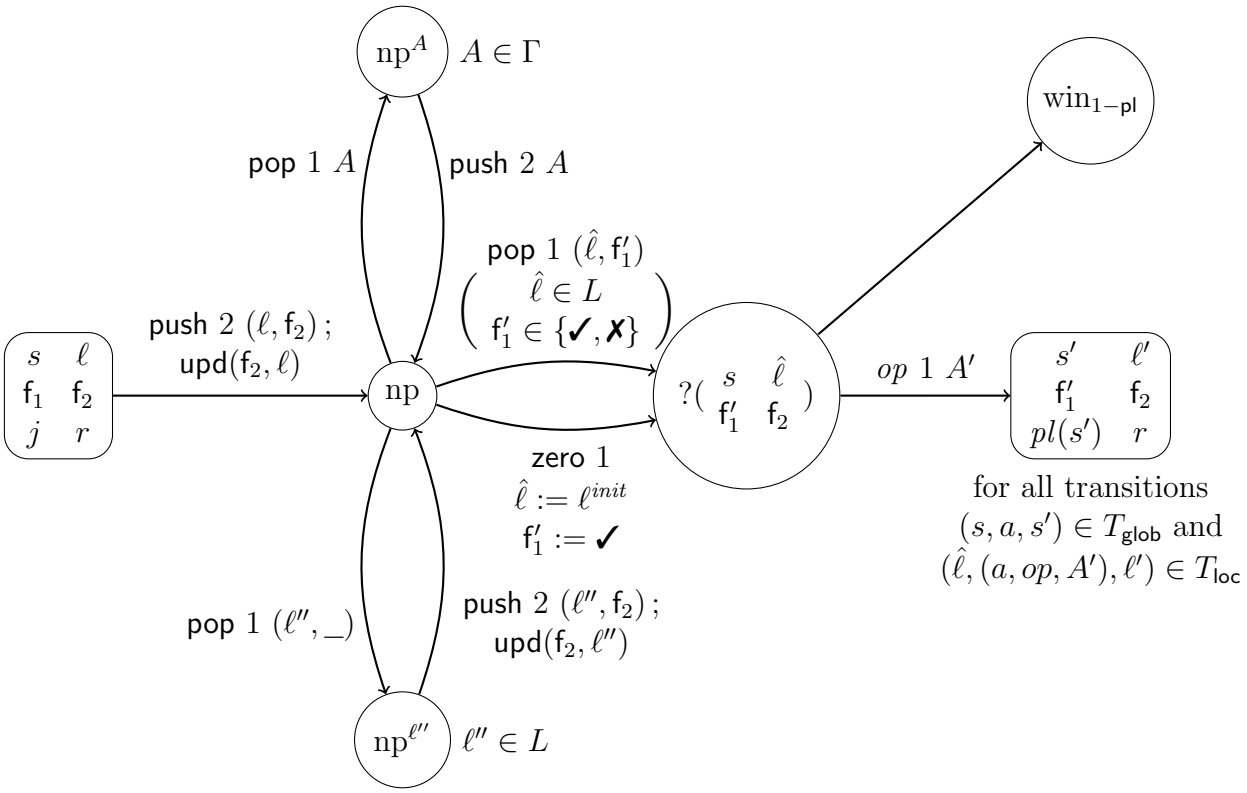

Figure 3.8: Change from process p to some process $p' > p$ (staying in the same round). All intermediate states belong to Player j; from every intermediate state, there is an outgoing internal transition to $\mathrm{win}_{1-\mathsf{pl}}$. Moreover, $\mathsf{upd}(\mathsf{f}_2, \bar{\ell})$ stands for the update rule IF $(f_2 = \checkmark \wedge \bar{\ell} \in F_{\mathsf{loc}})$ THEN $\mathsf{f}_2 := \checkmark$ ELSE $\mathsf{f}_2 := \times$.

For all $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \in S^*_{\mathrm{sim}}$ such that $r < B$, we introduce, in $\mathcal{M}$, the gadget given in Figure 3.9. It is similar to the previous gadget. However, we now shift symbols from stack 2 onto stack 1 and have to update f_1 accordingly.

Formally, let $\mathfrak{I}_{\mathrm{NR}}$ the set of intermediate states for a round change. For every $A \in \Gamma$, $s \in S$, $\ell \in L$, $\mathsf{f}_1, \mathsf{f}_2, \mathsf{g} \in \{\checkmark, \times\}$, $\mathsf{pl} \in \{0, 1\}$, and $r \leq B$ such that $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \in S^*_{\mathrm{sim}}$, we include $\mathrm{nr}(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r)$, $\mathrm{nr}^A(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r)$, and $\mathrm{nr}^\ell(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r)$ in $\mathfrak{I}_{\mathrm{NR}}$, together with the following transitions:

1. $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{push},1,(\ell,\mathsf{f}_1)} \mathrm{nr}(s, \mathsf{upd}(\mathsf{f}_1, \ell), \mathsf{f}_2, \mathsf{pl}, r)$,
2. $\mathrm{nr}(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{pop},2,A} \mathrm{nr}^A(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r)$,
3. $\mathrm{nr}^A(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{push},1,A} \mathrm{nr}(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r)$,
4. $\mathrm{nr}(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{pop},2,(\ell,\mathsf{g})} \mathrm{nr}^\ell(s, \mathsf{f}_1, \mathsf{g}, \mathsf{pl}, r)$,
5. $\mathrm{nr}^\ell(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{push},1,(\ell,\mathsf{f}_1)} \mathrm{nr}(s, \mathsf{upd}(\mathsf{f}_1, \ell), \mathsf{f}_2, \mathsf{pl}, r)$, and
6. $\mathrm{nr}(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{pop},2,(\ell,\mathsf{g})} ?(s, \ell, \mathsf{f}_1, \mathsf{g}, \mathsf{pl}, r + 1)$.

To simplify the proof of correctness, we assume that, after a transition of type 6., the first stack becomes the active stack, forcing a phase change so that the phase

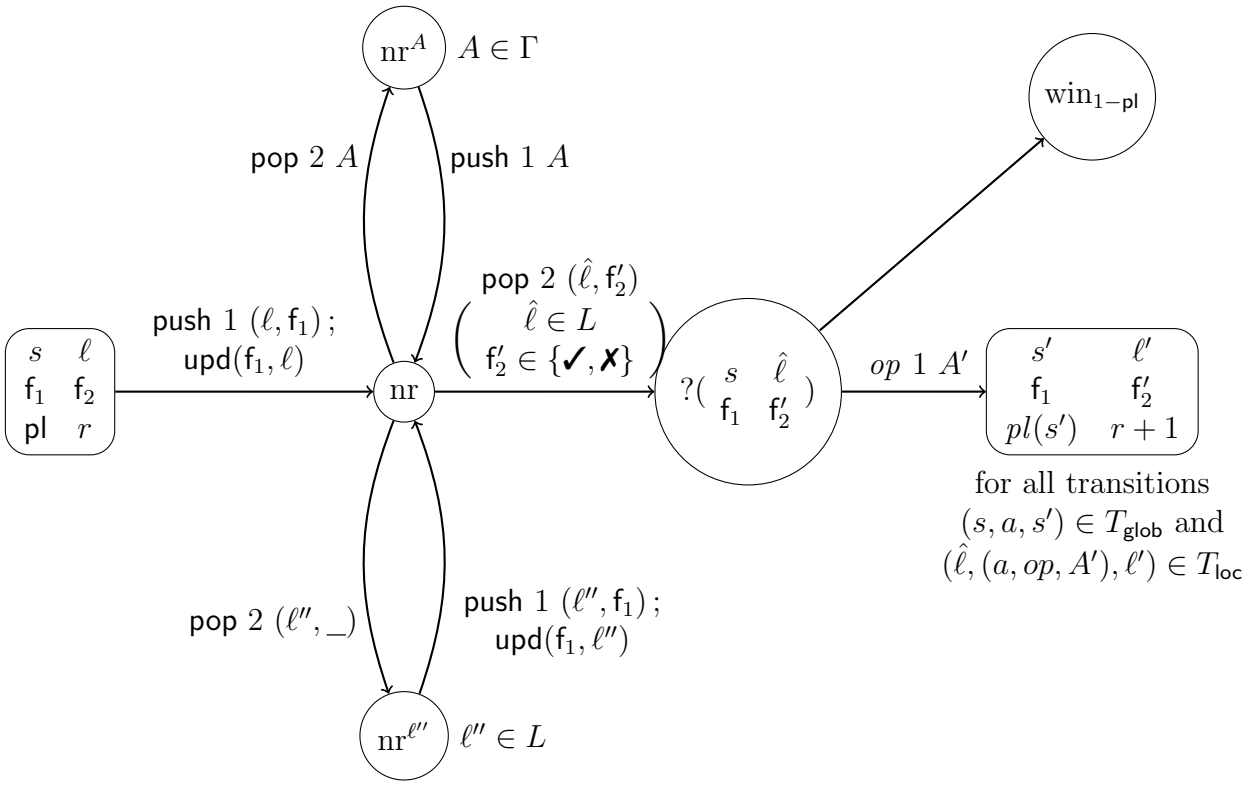

Figure 3.9: Go from a process p to some process $p' < p$ (involving a round change). All intermediate states belong to Player j; from every intermediate state, there is an outgoing internal transition to $\mathrm{win}_{1-\mathsf{pl}}$. Moreover, $\mathsf{upd}(\mathsf{f}_1, \bar{\ell})$ stands for the update rule IF $(f_1 = \checkmark \wedge \bar{\ell} \in F_{\mathsf{loc}})$ THEN $\mathsf{f}_1 := \checkmark$ ELSE $\mathsf{f}_1 := \times$.

number is always incremented by 2 after going in a round change gadget. This can be done for instance by using intermediate states and doing dummy push then pop transitions on stack 1.

Exiting the gadgets:

First, for all $s \in S$, $\ell \in L$, $\mathsf{f}_1, \mathsf{f}_2 \in \{\checkmark, \times\}$, $\mathsf{pl} \in \{0, 1\}$, and $r \leq B$, we have $?(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \in \mathfrak{I}_?$. For all such states, there is a transition

1. $?(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{int},1,-} \mathrm{win}_{1-\mathsf{pl}}$.

We also add

2. $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{\mathsf{int},1,-} \mathrm{win}_{1-\mathsf{pl}}$ for all $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \in S^*_{\mathrm{sim}}$.

These two transitions force a player to lose if there is no other transition available.

Moreover, for all $(s, a, s') \in T_{\mathsf{glob}}$ and $(\ell, (a, op, A), \ell') \in T_{\mathsf{loc}}$ in $\mathcal{P}$, there is

3. $?(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{op,1,A} (s', \ell', \mathsf{f}_1, \mathsf{f}_2, pl(s'), r)$

which completes the simulation of a transition from $\mathcal{P}$.

Finally, when $\mathsf{pl} = 0$ and $pl(s) = 1$, for all $(s, a, s') \in T_{\mathsf{glob}}$ and $(\ell, (a, op, A), \ell') \in T_{\mathsf{loc}}$ in $\mathcal{P}$, there are two additional transitions

4. $?(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{(op,1,A)} \mathrm{win}_0$, and

5. $(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \xrightarrow{(op,1,A)} \mathrm{win}_0$.

These last transitions allow Player 0 to win in the case of a change of player, if a transition was indeed available to Player 1. Otherwise, Player 0 will have no other choice but to take the transition leading to win_1.

We can now state correctness of our construction.

Lemma 12. *Player 0 has a winning strategy in $\mathfrak{G}_{\mathcal{M}}$ if and only if System has a winning strategy in $\mathcal{G}^B$.*

The rest of this section is dedicated to the proof of this lemma.

By construction, every play of $\mathcal{G}^B$ is closely mirrored by a play of the game $\mathfrak{G}_{\mathcal{M}}$ we built (and vice-versa). Despite having more intermediate states in the gadgets, the possible plays in $\mathfrak{G}_{\mathcal{M}}$ are restricted in a way such that basically the only thing a player can choose is a process and a transition to be executed by that process, which corresponds to what a player can do in $\mathcal{G}^B$. Let us formalize this intuition by giving a mapping π between plays of $\mathcal{G}^B$ and plays of $\mathfrak{G}_{\mathcal{M}}$.

In the base game $\mathcal{G}^B$, for all configurations c, c', round r and processes $p' < p$, there is a transition $(c, p, r) \to (c', p', r+1)$ iff there is a transition $(c, p', r+1) \to (c', p', r+1)$. Similarly, for all $p' > p$, there is a transition $(c, p, r) \to (c', p', r)$ iff there is a transition $(c, p', r) \to (c', p', r)$. In $\mathfrak{G}_{\mathcal{M}}$, a transition from "$(c, p, r)$" to "$(c', p', r+1)$" will be simulated by a sequence of transitions corresponding to a "dummy transition" from "(c, p, r)" to "$(c, p', r+1)$" followed by an actual transition to "$(c', p', r+1)$". This will be similar for $p' > p$.

By abuse of notation, we say that a node $v \in V$ of $\mathfrak{G}_{\mathcal{M}}$ is in S_{sim} if $v = (s_{\mathcal{M}}, \overline{\gamma_1}, \overline{\gamma_2}, st, ph)$ with $s_{\mathcal{M}} \in S_{\mathrm{sim}}$.

Let $v = ((s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r), \overline{\gamma_1}, \overline{\gamma_2}, 1, ph) \in S_{\mathrm{sim}}$ be a node of $\mathfrak{G}_{\mathcal{M}}$, with

$$\overline{\gamma_1} = \gamma_p \cdot (\ell_{p+1}, \mathsf{g}_{p+1}) \cdot \gamma_{p+1} \cdots (\ell_k, \mathsf{g}_k) \cdot \gamma_k$$

$$\overline{\gamma_2} = \tilde{\gamma}_{p-1} \cdot (\ell_{p-1}, \mathsf{g}_{p-1}) \cdots \tilde{\gamma}_1 \cdot (\ell_1, \mathsf{g}_1)$$

for some $\ell_1, \ldots, \ell_k \in L$, $\gamma_1, \ldots, \gamma_k \in \Gamma^*$, and $\mathsf{g}_1, \ldots, \mathsf{g}_k \in \{\checkmark, \text{✗}\}$, where $\tilde{\gamma}$ denotes the mirror of γ.

We let $\mathit{Player}(v) = j$ denote the actual player associated with v (i.e., the pl component of the state), and let $\mathrm{size}(\overline{\gamma_1}) = k - p$ and $\mathrm{size}(\overline{\gamma_2}) = p - 1$ be the number of elements from $L \times \{\checkmark, \text{✗}\}$ in each stack. Moreover, if $\mathit{Player}(v) = 1$, we denote the configuration in which Player 1 has chosen to give the token to Player 0 by $\mathit{ChangePlayer}(v) = ((s, \ell, \mathsf{f}_1, \mathsf{f}_2, 0, r), \overline{\gamma_1}, \overline{\gamma_2}, 1, ph)$.

We say that v is *well-defined* if the following conditions are satisfied:

1. For all $1 \leq i \leq p-1$, $\mathsf{g}_i = \checkmark$ if and only if for all $1 \leq p' < i$, $\ell_{p'} \in F_{\mathsf{loc}}$,

2. For all $p+1 \leq i \leq k$, $\mathsf{g}_i = \checkmark$ if and only if for all $i < p' \leq k$, $\ell_{p'} \in F_{\mathsf{loc}}$,

3. $\mathsf{f}_1 = \checkmark$ if and only if for all $p < p' \leq k$, $\ell_{p'} \in F_{\mathsf{loc}}$,

4. $\mathsf{f}_2 = \checkmark$ if and only if for all $1 \leq p' < p$, $\ell_{p'} \in F_{\mathsf{loc}}$,

5. $\mathsf{pl} = pl(s)$, and

6. $ph = 2r$.

We extend this definition to nodes of $\mathfrak{I}_?$ in the following way: $(?(s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r), \overline{\gamma_1}, \overline{\gamma_2}, 1, ph)$ is well-defined if $((s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r), \overline{\gamma_1}, \overline{\gamma_2}, 1, ph)$ is well-defined.

First, we define the mapping for plays π on individual nodes. We will extend it to plays after that. For a configuration $c = (s, (\ell_1, \gamma_1) \dots (\ell_k, \gamma_k))$ and a process $p \in \{1, \dots, k\}$, we define the following flags

$$\mathsf{g}^<(c, p) = \checkmark \text{ iff } \ell_1, \dots, \ell_{p-1} \in F_{\mathsf{loc}}$$
$$\mathsf{g}^>(c, p) = \checkmark \text{ iff } \ell_{p+1}, \dots, \ell_k \in F_{\mathsf{loc}}$$

and the following two stacks

$$\tau_1(c, p) = \gamma_p \cdot (\ell_{p+1}, \mathsf{g}^>(c, p+1)) \cdot \gamma_{p+1} \cdots (\ell_k, \mathsf{g}^>(c, k)) \cdot \gamma_k$$
$$\tau_2(c, p) = \quad \tilde{\gamma}_{p-1} \cdot (\ell_{p-1}, \mathsf{g}^<(c, p-1)) \cdots \tilde{\gamma}_1 \cdot (\ell_1, \mathsf{g}^<(c, 1))$$

Let $c = (s, (\ell_1, \gamma_1) \dots (\ell_k, \gamma_k))$ and $u = (c, p, r)$ an extended configuration. Its image is defined by

$$\pi(u) = ((s, \ell_p, \mathsf{g}^>(c, p), \mathsf{g}^<(c, p), pl(s), r), \tau_1(c, p), \tau_2(c, p), 1, 2r)$$

Observe that $\pi(u)$ is well-defined and is in S_{sim}.

Conversely, if $v \in S_{\text{sim}}$ is well-defined, then $v = ((s, \ell_p, \mathsf{f}_1, \mathsf{f}_2, pl(s), r), \overline{\gamma_1}, \overline{\gamma_2}, 1, 2r)$ with

$$\overline{\gamma_1} = \gamma_p \cdot (\ell_{p+1}, \mathsf{g}_{p+1}) \cdot \gamma_{p+1} \cdots (\ell_k, \mathsf{g}_k) \cdot \gamma_k$$
$$\overline{\gamma_2} = \tilde{\gamma}_{p-1} \cdot (\ell_{p-1}, \mathsf{g}_{p-1}) \cdots \tilde{\gamma}_1 \cdot (\ell_1, \mathsf{g}_1)$$

We define $\hat{\pi}(v) = ((s, (\ell_1, \gamma_1) \dots (\ell_k, \gamma_k)), p, r)$. $\hat{\pi}$ is similarly defined on well-defined nodes in $\mathfrak{I}_?$. If $v \in S_{\text{sim}}$, then $\pi(\hat{\pi}(v)) = v$.

Note that every reachable $v \in S_{\text{sim}}$ in $\mathfrak{G}_{\mathcal{M}}$ is either well-defined, or only fails item 5 because $\mathsf{pl} = 0$ and $pl(s) = 1$ (in case of a "Change of Player" initiated by Player 1) (see Corollary 16 hereafter).

When $u' = (c', p', r')$ is a successor of $u = (c, p, r)$ in $\mathcal{G}^B$, we do not necessarily have $\pi(u')$ successor of $\pi(u)$ in $\mathfrak{G}_{\mathcal{M}}$, because of the mechanism of process or round change. We introduce a notation to describe the (unique) part of run that allows to go from $\pi(u)$ to $\pi(u')$.

- If $p < p'$, we define recursively the functions $\text{next}_i^{\text{np}} : \mathfrak{I}_{\text{NP}} \times (\Gamma'^*)^2 \times \{1\} \times \mathbb{N} \to V^*$, for $i \in \mathbb{N}^*$.

  Let $\text{np}(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r) \in \mathfrak{I}_{\text{NP}}$, $A \in \Gamma$, $\ell \in L, \mathsf{g} \in \{\checkmark, \text{✗}\}$,

$$\mathrm{next}_i^{\mathrm{np}}(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),A\cdot\overline{\gamma_1},\overline{\gamma_2},1,2r) =$$
$$(\mathrm{np}^A(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),\overline{\gamma_1},\overline{\gamma_2},1,2r)\cdot(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),\overline{\gamma_1},A\cdot\overline{\gamma_2},1,2r)$$
$$\cdot\,\mathrm{next}_i^{\mathrm{np}}(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),\overline{\gamma_1},A\cdot\overline{\gamma_2},1,2r)$$

$$\mathrm{next}_i^{\mathrm{np}}(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),(\ell,\mathsf{g})\cdot\overline{\gamma_1},\overline{\gamma_2},1,2r) =$$
$$(\mathrm{np}^\ell(s,\mathsf{g},\mathsf{f}_2,\mathsf{pl},r)),\overline{\gamma_1},\overline{\gamma_2},1,2r)\cdot(\mathrm{np}(s,\mathsf{g},\mathsf{upd}(\mathsf{f}_2,\ell),\mathsf{pl},r),\overline{\gamma_1},(\ell,\mathsf{f}_2)\cdot\overline{\gamma_2},1,2r)$$
$$\cdot\,\mathrm{next}_{i-1}^{\mathrm{np}}(\mathrm{np}(s,\mathsf{g},\mathsf{upd}(\mathsf{f}_2,\ell),\mathsf{pl},r),\overline{\gamma_1},(\ell,\mathsf{f}_2)\cdot\overline{\gamma_2},1,2r)\text{ if } i>1$$

$$\mathrm{next}_1^{\mathrm{np}}(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),(\ell,\mathsf{g})\cdot\overline{\gamma_1},\overline{\gamma_2},1,2r) = (?(s,\ell,\mathsf{g},\mathsf{f}_2,\mathsf{pl},r),\overline{\gamma_1},\overline{\gamma_2},1,2r)$$

$$\mathrm{next}_1^{\mathrm{np}}(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),\varepsilon,\overline{\gamma_2},1,2r) = (?(s,\ell_0,\checkmark,\mathsf{f}_2,\mathsf{pl},r),\varepsilon,\overline{\gamma_2},1,2r)$$

If $1\leq i\leq \mathrm{size}(\overline{\gamma_1})+1$, it is easy to see that $\mathrm{next}_i^{\mathrm{np}}(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),\overline{\gamma_1},\overline{\gamma_2},1,2r)$ is well defined, since the number of elements of $\overline{\gamma_1}$ strictly decreases at each iteration, and since the index i decreases at each popping of an element of $L\times\{\checkmark,\text{✗}\}$.

- If $p>p'$, we define recursively $\mathrm{next}_i^{\mathrm{nr}}:\mathfrak{I}_{\mathrm{NP}}\times(\Gamma'^*)^2\times\{2\}\times\mathbb{N}\to V^*$, for $i\in\mathbb{N}^*$. Let $\mathrm{nr}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r)\in\mathfrak{I}_{\mathrm{NP}}$, $A\in\Gamma$, $\ell\in L,\mathsf{g}\in\{\checkmark,\text{✗}\}$:

$$\mathrm{next}_i^{\mathrm{nr}}(\mathrm{nr}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),\overline{\gamma_1},A\cdot\overline{\gamma_2},2,2r+1) =$$
$$(\mathrm{nr}^A(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),\overline{\gamma_1},\overline{\gamma_2},2,2r+1)\cdot(\mathrm{nr}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),A\cdot\overline{\gamma_1},\overline{\gamma_2},2,2r+1)$$
$$\cdot\,\mathrm{next}_i^{\mathrm{nr}}(\mathrm{nr}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),A\cdot\overline{\gamma_1},\overline{\gamma_2},2,2r+1)$$

$$\mathrm{next}_i^{\mathrm{nr}}(\mathrm{nr}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),\overline{\gamma_1},(\ell,\mathsf{g})\cdot\overline{\gamma_2},2,2r+1) =$$
$$(\mathrm{nr}^\ell(s,\mathsf{f}_1,\mathsf{g},\mathsf{pl},r)),\overline{\gamma_1},\overline{\gamma_2},2,2r+1)\cdot(\mathrm{nr}(s,\mathsf{upd}(\mathsf{f}_1,\ell),\mathsf{g},\mathsf{pl},r),(\ell,\mathsf{f}_1)\cdot\overline{\gamma_1},\overline{\gamma_2},2,2r+1)$$
$$\cdot\,\mathrm{next}_{i-1}^{\mathrm{nr}}(\mathrm{nr}(s,\mathsf{upd}(\mathsf{f}_1,\ell),\mathsf{g},\mathsf{pl},r),(\ell,\mathsf{f}_1)\cdot\overline{\gamma_1},\overline{\gamma_2},2,2r+1)\text{ if } i>1$$

$$\mathrm{next}_1^{\mathrm{nr}}(\mathrm{nr}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),\overline{\gamma_1},(\ell,\mathsf{g})\cdot\overline{\gamma_2},2,2r+1) =$$
$$(?(s,\ell,\mathsf{f}_1,\mathsf{g},\mathsf{pl},r+1),\overline{\gamma_1},\overline{\gamma_2},1,2r+2)^{1}$$

Similarly, if $1\leq i\leq\mathrm{size}(\overline{\gamma_2})$, one can see that $\mathrm{next}_i^{\mathrm{nr}}(\mathrm{nr}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),\overline{\gamma_1},\overline{\gamma_2},2,2r+1)$ is well defined as the number of elements of $\overline{\gamma_2}$ strictly decrease at each iteration and the index i decrease when an element from $L\times\{\checkmark,\text{✗}\}$ is popped.

[1] Normally, the last two elements of the tuple (active stack and current phase) should be 2 and $2r+1$. But as mentioned above, we assume that after the round change, the active stack is again stack 1, which can be achieved by pushing some element on stack 1 and popping it back immediately. We omit the necessary intermediate states to alleviate notations.

Now we define

$$\mathrm{next}_p^{p'}((s,\ell,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),\overline{\gamma_1},\overline{\gamma_2},1,ph)) =$$
$$\begin{cases}\varepsilon & \text{if } p=p'\\ (\mathrm{np}(s,\mathsf{f}_1,\mathsf{upd}(\mathsf{f}_2,\ell),\mathsf{pl},r),\overline{\gamma_1},(\ell,\mathsf{f}_2)\cdot\overline{\gamma_2},1,ph)\cdot & \\ \quad \mathrm{next}^{\mathrm{np}}_{p'-p}(\mathrm{np}(s,\mathsf{f}_1,\mathsf{upd}(\mathsf{f}_2,\ell),\mathsf{pl},r),\overline{\gamma_1},(\ell,\mathsf{f}_2)\cdot\overline{\gamma_2},1,ph) & \text{if } p<p'\\ (\mathrm{nr}(s,\mathsf{upd}(\mathsf{f}_1,\ell),\mathsf{f}_2,\mathsf{pl},r),(\ell,\mathsf{f}_1)\cdot\overline{\gamma_1},\overline{\gamma_2},2,ph+1)^{2}\cdot & \\ \quad \mathrm{next}^{\mathrm{nr}}_{p-p'}(\mathrm{nr}(s,\mathsf{upd}(\mathsf{f}_1,\ell),\mathsf{f}_2,\mathsf{pl},r),(\ell,\mathsf{f}_1)\cdot\overline{\gamma_1},\overline{\gamma_2},2,ph+1) & \text{if } p'<p\end{cases}$$

Lemma 13. *Given a configuration $c=(s,(\ell_1,\gamma_1)\dots(\ell_k,\gamma_k))$ and two processes $p\in\{1,\dots,k\}, p'\in\{1,\dots,k+1\}$ such that $p\neq p'$, $\mathrm{next}_p^{p'}(\pi(c,p,r))$ is a run of $\mathfrak{G}_{\mathcal{M}}$ ending in the state $(?(s,\ell_{p'},\mathsf{g}^{>}(c,p'),\mathsf{g}^{<}(c,p'),pl(s),r'),\tau_1(c,p'),\tau_2(c,p'),1,2r')$. Moreover, if $p<p'$, then $r'=r$, if $p>p'$ then $r'=r+1$.*

Proof. First we show some general properties of $\mathrm{next}_i^{\mathrm{np}}$.

Let $u=(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),\overline{\gamma_1},\overline{\gamma_2},1,2r)$ be a node such that $\mathrm{size}(\overline{\gamma_1})\geq 1$. Then by definition there exists $n\in\mathbb{N}$ and $A_1,\dots,A_n$ and (ℓ,g) such that $\overline{\gamma_1}=A_1\cdots A_n\cdot(\ell,\mathsf{g})\cdot\overline{\gamma_1}'$. Then for all $i\in\mathbb{N}^*$, we prove by induction on n that $u\cdot\mathrm{next}_i^{\mathrm{np}}(u)=u\cdot\rho\cdot\mathrm{next}_i^{\mathrm{np}}(v)$ where $v=(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),(\ell,\mathsf{g})\cdot\overline{\gamma_1}',A_n\cdots A_1\cdot\overline{\gamma_2},1,2r)$ and $u\cdot\rho$ is a valid run ending in v. The case $n=0$ is trivial as it means that $\rho=\varepsilon$ and $v=u$. If $n>0$, then

$$\begin{aligned}\mathrm{next}_i^{\mathrm{np}}(u)=&(\mathrm{np}^{A_1}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),A_2\cdots A_n\cdot(\ell,\mathsf{g})\cdot\overline{\gamma_1}',\overline{\gamma_2},1,2r)\\ &\cdot(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),A_2\cdots A_n\cdot(\ell,\mathsf{g})\cdot\overline{\gamma_1}',A_1\cdot\overline{\gamma_2},1,2r)\\ &\cdot\mathrm{next}_i^{\mathrm{np}}(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),A_2\cdots A_n\cdot(\ell,\mathsf{g})\cdot\overline{\gamma_1}',A_1\cdot\overline{\gamma_2},1,2r)\end{aligned}$$

which by induction hypothesis can be rewritten as

$$\begin{aligned}\mathrm{next}_i^{\mathrm{np}}(u)=&(\mathrm{np}^{A_1}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),A_2\cdots A_n\cdot(\ell,\mathsf{g})\cdot\overline{\gamma_1}',\overline{\gamma_2},1,2r)\\ &\cdot(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),A_2\cdots A_n\cdot(\ell,\mathsf{g})\cdot\overline{\gamma_1}',A_1\cdot\overline{\gamma_2},1,2r)\\ &\cdot\rho'\cdot\mathrm{next}_i^{\mathrm{np}}(v)\end{aligned}$$

with ρ' ending in v. Let $\rho=(\mathrm{np}^{A_1}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),A_2\cdots A_n\cdot(\ell,\mathsf{g})\cdot\overline{\gamma_1}',\overline{\gamma_2},1,2r)\cdot(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),A_2\cdots A_n\cdot(\ell,\mathsf{g})\cdot\overline{\gamma_1}',A_1\cdot\overline{\gamma_2},1,2r)\cdot\rho'$. Then $u\cdot\rho$ is a valid run of $\mathfrak{G}_{\mathcal{M}}$ because of transitions of type 2 and 3 given in the definition of $\mathfrak{I}_{\mathrm{NP}}$ and it ends in node v. Moreover if $i>1$ then

$$\begin{aligned}\mathrm{next}_i^{\mathrm{np}}(v)=&(\mathrm{np}^{\ell}(s,\mathsf{g},\mathsf{f}_2,\mathsf{pl},r)),\overline{\gamma_1}',A_n\cdots A_1\cdot\overline{\gamma_2},1,2r)\\ &\cdot(\mathrm{np}(s,\mathsf{g},\mathsf{upd}(\mathsf{f}_2,\ell),\mathsf{pl},r),\overline{\gamma_1}',(\ell,\mathsf{f}_2)\cdot A_n\cdots A_1\cdot\overline{\gamma_2},1,2r)\\ &\cdot\mathrm{next}_{i-1}^{\mathrm{np}}(\mathrm{np}(s,\mathsf{g},\mathsf{upd}(\mathsf{f}_2,\ell),\mathsf{pl},r),\overline{\gamma_1}',(\ell,\mathsf{f}_2)\cdot A_n\cdots A_1\cdot\overline{\gamma_2},1,2r)\end{aligned}$$

which again is a valid run of $\mathfrak{G}_{\mathcal{M}}$ because of transitions of type 4 and 5. Finally we can state that if $i>1$ then:

$$\begin{aligned}&\mathrm{next}_i^{\mathrm{np}}(\mathrm{np}(s,\mathsf{f}_1,\mathsf{f}_2,\mathsf{pl},r),A_1\cdots A_n\cdot(\ell,\mathsf{g})\cdot\overline{\gamma_1}',\overline{\gamma_2},1,2r)=\\ &\rho''\cdot\mathrm{next}_{i-1}^{\mathrm{np}}(\mathrm{np}(s,\mathsf{g},\mathsf{upd}(\mathsf{f}_2,\ell),\mathsf{pl},r),\overline{\gamma_1}',(\ell,\mathsf{f}_2)\cdot A_n\cdots A_1\cdot\overline{\gamma_2},1,2r)\end{aligned}$$

[2] Similarly and for simplicity, here we assume that the second stack becomes active.

Conversely, if $i = 1$, then $\text{next}_i^{\text{np}}(v) = (?(s, \ell, \mathsf{g}, \mathsf{f}_2, \mathsf{pl}, r), \overline{\gamma_1}', A_n \cdots A_1 \cdot \overline{\gamma_2}, 1, 2r)$, and $u \cdot \rho \cdot \text{next}_i^{\text{np}}(v)$ is a valid run due to the transition of type 6. Similarly, if $i = 1$ and $u = (\text{np}(s, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r), \overline{\gamma_1}, \overline{\gamma_2}, 1, 2r)$ is a node such that $\text{size}(\overline{\gamma_1}) = 0$, i.e. $\overline{\gamma_1} = A_1 \cdots A_n$, then we have that $\text{next}_i^{\text{np}}(u) = \rho \cdot (?(s, \ell^{init}, \checkmark, \mathsf{f}_2, \mathsf{pl}, r), \varepsilon, A_n \cdots A_1 \cdot \overline{\gamma_2}, 1, 2r)$ which is a valid run of $\mathfrak{G}_{\mathcal{M}}$ due to transitions of type 2 and 3 for ρ and type 7 for the last step.

Now let c, p, p', r defined as stated in the lemma with $p < p' \leq k + 1$, and let $\pi(c, p, r) = ((s, \ell_p, \mathsf{f}_1, \mathsf{f}_2, pl(s), r), \overline{\gamma_1}, \overline{\gamma_2}, 1, 2r)$. By definition, we have that $\mathsf{f}_1 = \mathsf{g}^{>}(c, p)$, $\mathsf{f}_2 = \mathsf{g}^{<}(c, p)$, $\overline{\gamma_1} = \tau_1(c, p)$, and $\overline{\gamma_2} = \tau_2(c, p)$.
Let $u = (\text{np}(s, \mathsf{f}_1, \mathsf{upd}(\mathsf{f}_2, \ell_p), \mathsf{pl}, r), \overline{\gamma_1}, (\ell_p, \mathsf{f}_2) \cdot \overline{\gamma_2}, 1, 2r)$ so that $\text{next}_p^{p'}(\pi(c, p, r)) = u \cdot \text{next}_{p'-p}^{\text{np}}(u)$.

We show that for all $0 \leq i < p' - p$, if we let $u_i = (\text{np}(s, \mathsf{g}^{>}(c, p + i), \mathsf{g}^{<}(c, p + i + 1), \mathsf{pl}, r), \tau_1(c, p + i), (\ell_{p+i}, \mathsf{g}^{<}(c, p + i)) \cdot \tau_2(c, p + i), 1, 2r)$, then we have that $\text{next}_{p'-p-i}^{\text{np}}(u_i)$ ends in node $(?(s, \ell_{p'}, \mathsf{g}^{>}(c, p'), \mathsf{g}^{<}(c, p'), pl(s), r), \tau_1(c, p'), \tau_2(c, p'), 1, 2r)$. Note that the node u defined above is u_0, so proving that this property holds for $i = 0$ proves the Lemma.

Suppose that $i = p' - p - 1$, i.e $p + i$ is the process immediately preceding p'. There are two different cases to study depending on whether $p' = k + 1$ or not.

1) If $p' = k+1$, then $p+i = k$ is the last process of c and therefore $\tau_1(c, p+i) = \gamma_k$, i.e. $\text{size}(\tau_1(c, p + i)) = 0$. Thus $\text{next}_1^{\text{np}}(u_i) = \rho \cdot (?(s, \ell^{init}, \checkmark, \mathsf{g}^{<}(c, k + 1), \mathsf{pl}, r), \varepsilon, \tilde{\gamma}_k \cdot (\ell_k, \mathsf{g}^{<}(c, k)) \cdot \tau_2(c, k), 1, 2r)$ and:

- $\ell^{init} = \ell_{k+1}$ as every new process starts in state ℓ^{init},
- $\mathsf{g}^{>}(c, k + 1) = \checkmark$ since $k + 1$ is a new process so there are no processes above it,
- $\varepsilon = \tau_1(c, k + 1)$ as every new process starts with an empty stack,
- $\tilde{\gamma}_k \cdot (\ell_k, \mathsf{g}^{<}(c, k)) \cdot \tau_2(c, k) = \tau_2(c, k + 1)$.

2) If $p' < k + 1$, then $\tau_1(c, p + i) = \gamma_{p+i} \cdot (\ell_{p'}, \mathsf{g}^{>}(c, p')) \cdot \tau_1(c, p')$. In that case, $\text{next}_1^{\text{np}}(u_i) = \rho \cdot (?(s, \ell_{p'}, \mathsf{g}^{>}(c, p'), \mathsf{g}^{<}(c, p + i + 1), \mathsf{pl}, r), \tau_1(c, p'), \tilde{\gamma}_{p+i} \cdot (\ell_{p+i}, \mathsf{g}^{<}(c, p + i)) \cdot \tau_2(c, p + i), 1, 2r)$, and

- $\mathsf{g}^{<}(c, p + i + 1) = \mathsf{g}^{<}(c, p')$,
- $\tilde{\gamma}_{p+i} \cdot (\ell_{p+i}, \mathsf{g}^{<}(c, p + i)) \cdot \tau_2(c, p + i) = \tau_2(c, p + i + 1) = \tau_2(c, p')$,

which satisfy the conditions stated above.

Now suppose the property holds for some $i > 0$. Necessarily, $\text{size}(\tau_1(c, p + (i - 1))) \geq 1$ since there is at least one process between $p + (i - 1)$ and p', and $\tau_1(c, p + (i - 1)) = \gamma_{p+(i-1)} \cdot (\ell_{p+i}, \mathsf{g}^{>}(c, p + i)) \cdot \tau_1(c, p + i)$. Then

$$\begin{aligned}
&\text{next}_{p'-p-(i-1)}^{\text{np}}(u_{i-1}) = \rho \\
&\cdot \text{next}_{p'-p-i}^{\text{np}}(\text{np}(s, \mathsf{g}^{>}(c, p + i), \mathsf{upd}(\mathsf{g}^{<}(c, p + i), \ell_{p+i}), \mathsf{pl}, r), \tau_1(c, p + (i - 1)), \\
&(\ell_{p+i}, \mathsf{g}^{>}(c, p + i)) \cdot \gamma_{p+(i-1)} \cdot (\ell_{p+(i-1)}, \mathsf{g}^{>}(c, p + (i - 1))) \cdot \tau_2(c, p + (i - 1)), 1, 2r)
\end{aligned}$$

and as

- $\mathsf{upd}(\mathsf{g}^{<}(c, p + i), \ell_{p+i}) = \mathsf{g}^{<}(c, p + i + 1)$, and

- $(\ell_{p+i}, \mathsf{g}^{>}(c, p+i)) \cdot \gamma_{p+(i-1)} \cdot (\ell_{p+(i-1)}, \mathsf{g}^{>}(c, p+(i-1))) \cdot \tau_2(c, p+(i-1)) = (\ell_{p+i}, \mathsf{g}^{>}(c, p+i) \cdot \tau_2(c, p+i)$,

then it can be rewritten as $\mathrm{next}^{\mathrm{np}}_{p'-p-(i-1)}(u_{i-1}) = \rho \cdot \mathrm{next}^{\mathrm{np}}_{p'-p-i}(u_i)$, meaning that the property also holds for $i-1$.

The proof for $\mathrm{next}^{\mathrm{nr}}$ is similar. □

Conversely, we show that a run between a node $u \in S_{\mathrm{sim}}$ and a node in $\mathfrak{I}_?$ is necessarily of the form $\mathrm{next}^p_{p'}(u)$.

Lemma 14. *Let $\hat{\rho} = \hat{\rho}' \cdot u \cdot \hat{\rho}'' \cdot v$ be a finite play of $\mathfrak{G}_{\mathcal{M}}$ such that u is the last state in S_{sim} and $v \in \mathfrak{I}_?$. Then, there exists two distinct p and p' such that $\hat{\rho}'' \cdot v = next^{p'}_p(u)$.*

Proof. Let $u = ((s, \ell, \mathsf{f}_1, \mathsf{f}_2, \mathsf{pl}, r), \overline{\gamma_1}, \overline{\gamma_2}, 1, ph) \in S_{\mathrm{sim}}$ be well-defined, and $p = \mathrm{size}(\overline{\gamma_2}) + 1$. By the transition relation of $\mathfrak{G}_{\mathcal{M}}$, in order to reach $\mathfrak{I}_?$ one must go through either $\mathfrak{I}_{\mathrm{NP}}$ or $\mathfrak{I}_{\mathrm{NR}}$. Since by hypothesis $\hat{\rho}''$ does not contain a node in S_{sim}, all of $\hat{\rho}''$ must occur in either $\mathfrak{I}_{\mathrm{NP}}$ or $\mathfrak{I}_{\mathrm{NR}}$.

Suppose the former is true (the other case will, again, be extremely similar). Necessarily, the first node in $\hat{\rho}''$ is $u' = (\mathrm{np}(s, \mathsf{f}_1, \mathsf{upd}(\mathsf{f}_2, \ell), \mathsf{pl}, r), \overline{\gamma_1}, (\ell, \mathsf{f}_2) \cdot \overline{\gamma_2}, 1, ph)$, as this is the only transition from u that goes in $\mathfrak{I}_{\mathrm{NP}}$ (a transition of type 1 in the description of $\mathfrak{I}_{\mathrm{NP}}$).

Let us show that for all (not strict) suffixes $\hat{\rho}_1$ of $\hat{\rho}''$ such that $\hat{\rho}_1$ starts in $u_1 = (\mathrm{np}(s, \mathsf{f}'_1, \mathsf{f}'_2, \mathsf{pl}, r), \overline{\gamma_1}', \overline{\gamma_2}', 1, ph)$ for some $\mathsf{f}'_1, \mathsf{f}'_2, \overline{\gamma_1}', \overline{\gamma_2}'$, then there exists a $k \geq 1$ such that $\hat{\rho}_1 \cdot v = u_1 \cdot \mathrm{next}^{\mathrm{np}}_k(u_1)$. Suppose that $\hat{\rho}_1 = u_1$, i.e. the next node is v. Then a transition of either type 6 or 7 has been used to go from u_1 to v. However, the kind of transition depends only on $\overline{\gamma_1}'$. If $\overline{\gamma_1}' = (\ell, \mathsf{g}) \cdot \overline{\gamma_1}''$ then only a transition of type 6 can be used, and if $\overline{\gamma_1}' = \varepsilon$, only a transition of type 7 can be used. In both cases, we have that $\hat{\rho}_1 \cdot v = u_1 \cdot \mathrm{next}^{\mathrm{np}}_1(u_1)$.

Now suppose that $\hat{\rho}_1 = u_1 \cdot \hat{\rho}_2$ with $\hat{\rho}_2$ non-empty. Then $\overline{\gamma_1}'$ cannot be empty, otherwise the only transition available would lead to v. Therefore there are two cases to consider:

- If $\overline{\gamma_1}' = A \cdot \overline{\gamma_1}''$, then the only available transition (type 2) leads to $u'_1 = (\mathrm{np}^A(s, \mathsf{f}'_1, \mathsf{f}'_2, \mathsf{pl}, r), \overline{\gamma_1}'', \overline{\gamma_2}', 1, ph)$ from which the only transition (type 3) leads to $u_2 = (\mathrm{np}(s, \mathsf{f}'_1, \mathsf{f}'_2, \mathsf{pl}, r), \overline{\gamma_1}'', A \cdot \overline{\gamma_2}', 1, ph)$, which corresponds to what $\mathrm{next}^{\mathrm{np}}$ would do. Then by induction $\hat{\rho}_2 \cdot v = u'_1 \cdot u_2 \cdot \mathrm{next}^{\mathrm{np}}_k(u_2)$ for some k, so $\hat{\rho}_1 \cdot v = u_1 \cdot u'_1 \cdot u_2 \cdot \mathrm{next}^{\mathrm{np}}_k(u_2) = u_1 \cdot \mathrm{next}^{\mathrm{np}}_k(u_1)$.

- If $\overline{\gamma_1}' = (\ell, \mathsf{g}) \cdot \overline{\gamma_1}''$, then there are two available transitions: type 4 and type 6. However, a transition of type 6 would lead to $\mathfrak{I}_?$ but this cannot happen because we supposed that $\hat{\rho}_2$ is not empty. Therefore a transition of type 4 leads to $u'_1 = (\mathrm{np}^{\ell}(s, \mathsf{g}, \mathsf{f}_2, \mathsf{pl}, r), \overline{\gamma_1}'', \overline{\gamma_2}', 1, ph)$ from which the only possible transition, which is of type 5, leads to $u_2 = (\mathrm{np}(s, \mathsf{g}, \mathsf{upd}(\mathsf{f}'_2, \ell), \mathsf{pl}, r), \overline{\gamma_1}'', (\ell, \mathsf{g}) \cdot \overline{\gamma_2}', 1, ph)$. Then we have $\hat{\rho}_2 \cdot v = u'_1 \cdot u_2 \cdot \mathrm{next}^{\mathrm{np}}_k(u_2)$ for some k, so $\hat{\rho}_1 \cdot v = u_1 \cdot u'_1 \cdot u_2 \cdot \mathrm{next}^{\mathrm{np}}_k(u_2) = u_1 \cdot \mathrm{next}^{\mathrm{np}}_{k+1}(u_1)$.

Therefore this property holds for $\hat{\rho}''$, i.e. $\hat{\rho}'' \cdot v = u' \cdot \mathrm{next}^{\mathrm{np}}_k(u')$ for some $k \geq 1$, i.e. by definition $\hat{\rho}'' \cdot v = \mathrm{next}^{p+k}_p(u)$. □

We can now deduce the two following corollaries.

Corollary 15. *If $\hat{\rho} = \hat{\rho}' \cdot u \cdot \hat{\rho}'' \cdot v$ is a finite run ending in $v \in \mathfrak{I}_?$ such that u is the last node of $\hat{\rho}$ in S_{sim} and $\hat{\pi}(u) = (c, p, r)$, then $\hat{\pi}(v) = (c, p', r')$ for some $p' \neq p$ and $r' = r$ if $p' > p$, $r + 1$ otherwise.*

Proof. Using Lemma 14, $\hat{\rho}'' \cdot v$ is of the form $\text{next}_p^{p'}(u)$ for some $p' \neq p$. Then by Lemma 13 we have that $\hat{\pi}(v) = (c, p', r')$ with $r' = r$ if $p' > p$, $r + 1$ otherwise. □

Corollary 16. *For every reachable node $w \in S_{sim}$ in $\mathfrak{G}_{\mathcal{M}}$, there is a tuple (c, p, r) such that either $\hat{\pi}(w) = (c, p, r)$ or $w = ChangePlayer(u)$ with $\hat{\pi}(u) = (c, p, r)$.*

Proof. By induction on the length of the run $\hat{\rho}$ leading to w: It is easy to see in case where $\hat{\rho} = s_M^{init} \cdot w$. If $\hat{\rho} = \hat{\rho}' \cdot u \cdot \hat{\rho}'' \cdot w$ where u is the last node of $\hat{\rho}$ in S_{sim} before w, then by induction hypothesis $u = \pi(c, p, r)$ for some (c, p, r). Then either:

- $\hat{\rho}'' = \varepsilon$, which means either a Change of Player occurred so the property is satisfied, or a Basic Transition occurred in which case $w = \pi(c', p, r)$ for some c' successor of c, or
- $\hat{\rho}''$ ends in a node $v \in \mathfrak{I}_?$, thus by the previous corollary $\hat{\pi}(v) = (c, p', r')$ with $r' = r$ if $p' > p$, $r+1$ otherwise. Therefore $w = \pi(c', p', r')$ for some c' successor of c.

□

Now we extend the definition of π on plays. We define

$$\pi((s^{init}), 0, 1) = (s_M^{init}, \varepsilon, \varepsilon, 0, 1)$$

and for all pairs of transitions of the form $(s^{init}, a, s) \in T_{\mathsf{glob}}$ and $(\ell^{init}, (a, op, A), \ell) \in T_{\mathsf{loc}}$, we have

$$\pi(((s^{init}), 0, 1) \cdot ((s, (\ell, \gamma_1)), 1, 1)) = (s_M^{init}, \varepsilon, \varepsilon, 0, 1) \cdot ((s, \ell, \checkmark, \checkmark, pl(s), 1), \gamma_1, \varepsilon, 1, 1)$$

Let ρ be a finite play of $\mathcal{G}^B$ ending in $u = (c, p, r)$, with $p \neq 0$, and $u' = (c', p', r')$ a successor of u. Then $\pi(\rho \cdot u')$ is defined as follows:

- If $\mathfrak{F} = \mathsf{Reach}$ and ρ is already winning, then $\pi(\rho \cdot u') = \pi(\rho)$.
- Else, if $\mathfrak{F} = \mathsf{Reach}$ and c' is an accepting configuration, then

$$\pi(\rho \cdot u') = \pi(\rho) \cdot \text{next}_p^{p'}(\pi(u)) \cdot \pi(u') \cdot (\text{win}_0, \tau_1(c', p'), \tau_2(c', p'), 1, 2r')^\omega$$

- Else, if u' has no successor, then

$$\pi(\rho \cdot u') = \pi(\rho) \cdot \text{next}_p^{p'}(\pi(u)) \cdot \pi(u') \cdot \text{Ch}_0 \cdot (\text{win}_1, \tau_1(c', p'), \tau_2(c', p'), 1, 2r')^\omega$$

  where $\text{Ch}_0 = ChangePlayer(\pi(u'))$ if $Player(\pi(u')) = 1$, ε otherwise.

- Otherwise, we simply let

$$\pi(\rho \cdot u') = \pi(\rho) \cdot \text{next}_p^{p'}(\pi(u)) \cdot \pi(u')$$

We extend the mapping to infinite plays in the following way: if ρ is an infinite play, we let $\pi(\rho) = \lim_{\rho' \sqsubseteq \rho} \pi(\rho')$.

Lemma 17. *If ρ is a play of $\mathcal{G}^B$, then $\pi(\rho)$ is a play of $\mathfrak{G}_\mathcal{M}$.*

Proof. We show it by induction on the size of ρ. If $\rho = ((s^{init}), 0, 1)$, then $\pi(\rho) = (s_M^{init}, \varepsilon, \varepsilon, 1, 1)$ which is a play of $\mathfrak{G}_\mathcal{M}$. Assume now that ρ is a finite play ending in $u = (c, p, r)$, and that $\pi(\rho)$ is a play of $\mathfrak{G}_\mathcal{M}$. Let $u' = (c', p', r')$ be such that $\rho \cdot u'$ is a play of $\mathcal{G}^B$.

If c is accepting and $\mathfrak{F} = \mathsf{Reach}$, then ρ is winning and $\pi(\rho \cdot u') = \pi(\rho)$, and by induction hypothesis, $\pi(\rho \cdot u')$ is a play of $\mathfrak{G}_\mathcal{M}$. Otherwise, $\pi(u) \in S^*_{\mathrm{sim}}$, $\pi(\rho \cdot u')$ starts with $\pi(\rho) \cdot \mathrm{next}_p^{p'}(\pi(u)) \cdot \pi(u')$ and $\pi(\rho)$ ends in $\pi(u)$. Let

$$\pi(u) = ((s, \ell_p, \mathsf{g}^>(c,p), \mathsf{g}^<(c,p), pl(s), r), \tau_1(c,p), \tau_2(c,p), 1, 2r)$$

and

$$\pi(u') = ((s', \ell'_{p'}, \mathsf{g}^>(c',p'), \mathsf{g}^<(c',p'), pl(s'), r'), \tau_1(c',p'), \tau_2(c',p'), 1, 2r').$$

- If $p = p'$, by definition, $\mathrm{next}_p^{p'}(\pi(u)) = \varepsilon$. Moreover, since u' is a successor of u in $\mathcal{G}^B$ there is $(s, a, s') \in T_{\mathsf{glob}}$ and $(\ell_p, (a, op, A), \ell'_p) \in T_{\mathsf{loc}}$ for some $a \in \Sigma$, hence $(s, \ell_p, \mathsf{g}^>(c,p), \mathsf{g}^<(c,p), pl(s), r) \xrightarrow{(op,1,A)} (s', \ell'_p, \mathsf{g}^>(c,p), \mathsf{g}^<c,p), pl(s'), r)$ in $\mathfrak{G}_\mathcal{M}$. Observe that in that case $\mathsf{g}^>(c',p') = \mathsf{g}^>(c,p)$ and $\mathsf{g}^<(c',p') = \mathsf{g}^<(c,p)$, hence $\pi(u')$ is indeed a successor of $\pi(u)$ in $\mathfrak{G}_\mathcal{M}$.

- If $p < p'$, then $r = r'$ and $\mathrm{next}_p^{p'}(\pi(u))$ starts with

  $$(\mathrm{np}(s, \mathsf{g}^>(c,p)), \mathsf{upd}(\mathsf{g}^<(c,p), \ell_p), pl(s), r), \tau_1(c,p), (\ell_p, \mathsf{g}^<(c,p)) \cdot \tau_2(c,p), 1, 2r)$$

  which is a successor of $\pi(u)$. By Lemma 13, $\mathrm{next}_p^{p'}(\pi(u))$ is a play of $\mathfrak{G}_\mathcal{M}$ that ends in $(?(s, \ell_{p'}, \mathsf{g}^>(c,p'), \mathsf{g}^<(c,p'), pl(s), r), \tau_1(c,p'), \tau_2(c,p'), 1, 2r)$. Observe that $\mathsf{g}^>(c,p') = \mathsf{g}^>(c',p')$ and $\mathsf{g}^<(c,p') = \mathsf{g}^<(c',p')$, then $\pi(u')$ is indeed a successor of $(?(s, \ell_{p'}, \mathsf{g}^>(c,p'), \mathsf{g}^<(c,p'), pl(s), r), \tau_1(c,p'), \tau_2(c,p'), 1, 2r)$.

- If $p > p'$, then $r' = r + 1$ and $\mathrm{next}_p^{p'}(\pi(u))$ starts with

  $$(\mathrm{nr}(s, \mathsf{upd}(\mathsf{g}^>(c,p), \ell_p), \mathsf{g}^<(c,p), pl(s), r), (\ell_p, \mathsf{g}^>(c,p)) \cdot \tau_1(c,p), \tau_2(c,p), 1, 2r)$$

  which is also a successor of $\pi(u)$. Again by Lemma 13, $\mathrm{next}_p^{p'}(\pi(u))$ is a play of $\mathfrak{G}_\mathcal{M}$ that ends in $(?(s, \ell_{p'}, \mathsf{g}^>(c,p'), \mathsf{g}^<(c,p'), pl(s), r+1), \tau_1(c,p'), \tau_2(c,p'), 1, 2r+2)$. Again, one can check that $\pi(u')$ is a successor of $(?(s, \ell_{p'}, \mathsf{g}^>(c,p'), \mathsf{g}^<(c,p'), pl(s), r+1), \tau_1(c,p'), \tau_2(c,p'), 1, 2r+2)$.

In any case, $\pi(\rho) \cdot \mathrm{next}_p^{p'}(\pi(u)) \cdot \pi(u')$ is a play of $\mathfrak{G}_\mathcal{M}$. Then there are two special cases to consider:

- If now $\mathfrak{F} = \mathsf{Reach}$ and c' is accepting, then by construction, $\mathsf{g}^>(c',p') = \mathsf{g}^<(c',p') = \checkmark$, $s' \in F_{\mathsf{glob}}$, $\ell_{p'} \in F_{\mathsf{loc}}$, hence $(s', \ell'_{p'}, \mathsf{g}^>(c',p'), \mathsf{g}^<(c',p'), pl(s'), r') \in \mathfrak{W}$ and thus $(\mathrm{win}_0, \tau_1(c',p'), \tau_2(c',p'), 1, 2r')$ is a successor of $\pi(u')$ and $\pi(\rho \cdot u')$ is a play of $\mathfrak{G}_\mathcal{M}$.

- If u' has no successor, $\pi(\rho) \cdot \text{next}_p^{p'}(\pi(u)) \cdot \pi(u') \cdot \text{Ch}_0$ is a play that ends in a state v such that $Player(v) = 0$, hence $(\text{win}_1, \tau_1(c', p'), \tau_2(c', p'), 1, 2r')$ is a successor of v and $\pi(\rho \cdot u')$ is a play of $\mathfrak{G}_{\mathcal{M}}$.

So for all finite play ρ in $\mathcal{G}^B$, $\pi(\rho)$ is a play of $\mathfrak{G}_{\mathcal{M}}$. If ρ is an infinite play, then $\pi(\rho)$ is also a play of $\mathfrak{G}_{\mathcal{M}}$, otherwise we can find a finite prefix ρ' of ρ such that $\pi(\rho')$ is not a play of $\mathfrak{G}_{\mathcal{M}}$. □

Conversely, we define

$$\hat{\pi}((s_M^{init}, \varepsilon, \varepsilon, 0, 1)) = ((s^{init}), 0, 1)$$

and, for all plays $\hat{\rho}$ and nodes v in $\mathfrak{G}_{\mathcal{M}}$,

$$\hat{\pi}(\hat{\rho} \cdot v) = \begin{cases} \hat{\pi}(\hat{\rho}) \cdot \hat{\pi}(v) \text{ if } v \text{ is a well-defined node in } S_{\text{sim}} \\ \hat{\pi}(\hat{\rho}) \text{ otherwise.} \end{cases}$$

Lemma 18. *If $\hat{\rho}$ is a play of $\mathfrak{G}_{\mathcal{M}}$ then $\hat{\pi}(\hat{\rho})$ is a play of $\mathcal{G}^B$.*

Proof. Let $\hat{\rho}$ be a finite prefix of a play of $\mathfrak{G}_{\mathcal{M}}$ and assume that $\hat{\pi}(\hat{\rho})$ is a play of $\mathcal{G}^B$. Let v be such that $\hat{\rho} \cdot v$ is a play of $\mathfrak{G}_{\mathcal{M}}$. If $v \notin S_{\text{sim}}$, or if v is not well-defined, then $\hat{\pi}(\hat{\rho} \cdot v) = \hat{\pi}(\hat{\rho})$ and it is then a play of $\mathcal{G}^B$. Otherwise, let $v = ((s, \ell, \mathsf{f}_1, \mathsf{f}_2, pl(s), r), \overline{\gamma_1}, \overline{\gamma_2}, 1, ph)$. Since it is in S_{sim} and well-defined, by definition of the transition relation of $\mathfrak{G}_{\mathcal{M}}$, $\hat{\rho}$ necessarily ends in v', a well-defined node of S_{sim}^* or of $\mathfrak{I}_?$. Let $\hat{\rho} = \hat{\rho}' \cdot v$ and $\hat{\pi}(v) = ((s, (\ell_1, \gamma_1) \dots (\ell_k, \gamma_k)), p, r)$, with $\ell = \ell_p$.

If $v' \in S_{\text{sim}}^*$, then $v' = ((s', \ell', \mathsf{f}_1, \mathsf{f}_2, pl(s'), r), \overline{\gamma_1'}, \overline{\gamma_2}, 1, ph)$, there exists a transition $(s', \ell', \mathsf{f}_1, \mathsf{f}_2, pl(s'), r) \xrightarrow{(op,1,A)} (s, \ell, \mathsf{f}_1, \mathsf{f}_2, pl(s), r)$, and $\hat{\pi}(\hat{\rho})$ ends in $\hat{\pi}(v') = ((s', (\ell_1', \gamma_1'), \dots, (\ell_k, \gamma_k), p, r)$ with $\ell_i' = \ell_i$, $\gamma_i' = \gamma_i$, for all $i \neq p$. By definition, there is a pair of transitions $(s', a, s) \in T_{\mathsf{glob}}$ and $(\ell', (a, op, A), \ell) \in T_{\mathsf{loc}}$ for some $a \in \Sigma$. Moreover, if $op = \text{push}$, $\overline{\gamma_1} = A \cdot \overline{\gamma_1'}$, hence by construction, $\gamma_p = A \cdot \gamma_p'$, and if $op = \text{pop}$, then $\overline{\gamma_1'} = A \cdot \overline{\gamma_1}$, then $\gamma_p' = A \cdot \gamma_p$. Then $\hat{\pi}(v)$ is indeed a successor of $\hat{\pi}(v')$ in $\mathcal{G}^B$. Hence, $\hat{\pi}(\hat{\rho} \cdot v) = \hat{\pi}(\hat{\rho}' \cdot v' \cdot v) = \hat{\pi}(\hat{\rho}') \cdot \hat{\pi}(v') \cdot \hat{\pi}(v)$ is a play of $\mathcal{G}^B$.

In case $v' \in \mathfrak{I}_?$, let $v' = (?(s', \ell', \mathsf{f}_1, \mathsf{f}_2, pl(s'), r), \overline{\gamma_1'}, \overline{\gamma_2}, 1, ph)$ and there exists a pair of transition $(s', a, s) \in T_{\mathsf{glob}}$ and $(\ell', (a, op, A), \ell) \in T_{\mathsf{loc}}$ for some $a \in \Sigma$. In that case, $\hat{\pi}(v') = (s', (\ell_1', \gamma_1'), \dots, (\ell_k', \gamma_k'), p, r)$ with $\ell_i' = \ell_i$, $\gamma_i' = \gamma_i$, for all $i \neq p$. Then again, if $op = \text{push}$, $\overline{\gamma_1'} = A \cdot \overline{\gamma_1}$, then $\overline{\gamma_p'} = A \cdot \overline{\gamma_p}$, and if $op = \text{pop}$, $\overline{\gamma_1} = A \cdot \overline{\gamma_1'}$, then $\overline{\gamma_p} = A \cdot \overline{\gamma_p'}$. Let $\hat{\rho} = \hat{\rho}' \cdot u \cdot \hat{\rho}'' \cdot v'$ with u the last configuration in $\hat{\rho}$ in S_{sim}. By Corollary 15, $\hat{\pi}(u) = ((s', (\ell_1', \gamma_1'), \dots, (\ell_k', \gamma_k')), p', r')$, with either $p' < p$ and $r' = r$ or $p < p'$ and $r = r' + 1$. We then have that $\hat{\pi}(v)$ is a successor in $\mathcal{G}^B$ of $\hat{\pi}(u)$. Hence, $\hat{\pi}(\hat{\rho} \cdot v) = \hat{\pi}(\hat{\rho}') \cdot \hat{\pi}(u) \cdot \hat{\pi}(v)$ is a play of $\mathcal{G}^B$. □

We are now ready to prove Lemma 12.

$\boxed{\Rightarrow}$ Let $f_{\mathcal{M}}$ be a winning strategy of $\mathfrak{G}_{\mathcal{M}}$, which we assume to be memoryless without loss of generality. We build first the function

$$next(v) = \begin{cases} f_{\mathcal{M}}(v) & \text{if } f_{\mathcal{M}}(v) \in S_{\text{sim}} \cup \{\text{win}_0, \text{win}_1\} \\ next(f_{\mathcal{M}}(v)) & \text{if } f_{\mathcal{M}}(v) \in \mathfrak{I} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

that maps any node $v \in V_0$ of Player 0 to the next node in S_{sim} according to $f_{\mathcal{M}}(v)$. Observe that if $f_{\mathcal{M}}(v) \in \mathfrak{I}$, $f_{\mathcal{M}}(v) \in V_0$, by construction of the game. Moreover, by

the structure of $\mathfrak{G}_{\mathcal{M}}$, any $f_{\mathcal{M}}$-run starting from a well-defined node $v \in V_0$ is of the form $v.w.v'$, with $w \in \mathfrak{I}^*$ and $v' \in S_{\text{sim}} \cup \{\text{win}_0, \text{win}_1\}$, then *next* is correctly defined.

Then we define a strategy for System in $\mathcal{G}^B$ as follows. Let $(c, p, r) \in V_0$,

$$f_{\mathcal{P}}(c,p,r) = \begin{cases} \hat{\pi}(\mathit{next}(\pi(c,p,r))) & \text{if } \mathit{next}(\pi(c,p,r)) \in S_{\text{sim}}, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Let ρ be a finite $f_{\mathcal{P}}$-play of $\mathcal{G}^B$. We show by induction on the length of ρ that either:

- $\pi(\rho)$ is an $f_{\mathcal{M}}$-play, or
- $\pi(\rho) = w \cdot v^\omega$ where w is a $f_{\mathcal{M}}$-play and v is in state win_1.

If ρ consists in one node, it is obvious. Let now ρ be an $f_{\mathcal{P}}$-play ending in $u = (c, p, r) \in V_0$, and assume that $\pi(\rho)$ is an $f_{\mathcal{M}}$-play. Let $f_{\mathcal{P}}(u) = u' = (c', p', r')$. If $\pi(\rho)$ is infinite, then $\pi(\rho \cdot u') = \pi(\rho)$ and is immediately an $f_{\mathcal{M}}$-play. Otherwise, $\pi(\rho)$ ends in $\pi(u)$, and $\pi(\rho \cdot u') = \pi(\rho) \cdot \text{next}_p^{p'}(\pi(u)) \cdot \mathit{next}(\pi(u)) \cdot \Gamma$, where Γ is either ε, or of the form $(\text{win}_0)^\omega$, or of the form $\text{Ch}_0 \cdot (\text{win}_1)^\omega$ (by definition of π). By definition of *next*, there exists $w \in \mathfrak{I}^*$ such that $\pi(c,p,r) \cdot w \cdot \pi(c',p',r')$ is an $f_{\mathcal{M}}$-play. By Lemma 14, $w = \text{next}_p^{p'}(\pi(u))$, which means that $\pi(\rho) \cdot \text{next}_p^{p'}(\pi(u)) \cdot \mathit{next}(\pi(u))$ is a $f_{\mathcal{M}}$-play. Hence, if $\Gamma = \varepsilon$ then $\pi(\rho \cdot u')$ is an $f_{\mathcal{M}}$-play. If Γ is of the form $(\text{win}_0)^\omega$, then by definition of π we have that $\mathfrak{F} = \mathsf{Reach}$ and the last configuration is accepting. Therefore, by construction of $\mathfrak{G}_{\mathcal{M}}$, the only transition available leads to the state win_0 which is a sink state, so this is the only possible choice for $f_{\mathcal{M}}$. Finally if Γ is of the form $\text{Ch}_0 \cdot (\text{win}_1)^\omega$ then the second item is verified with $w = \pi(\rho) \cdot \text{next}_p^{p'}(\pi(u)) \cdot \mathit{next}(\pi(u)) \cdot \text{Ch}_0$.

Note that if the second item is verified, i.e. $\pi(\rho) = w \cdot (\text{win}_1, \dots)^\omega$, then w ends in a node $\pi(u) \in S_{\text{sim}}^*$ such that u has no successor. In that case, it can be verified that no possible continuation from $\pi(u)$ can avoid win_1, meaning that $f_{\mathcal{M}}$ can not be winning as the play until that point is a $f_{\mathcal{M}}$-play and as there is no winning continuation. Therefore, for any $f_{\mathcal{P}}$-play ρ of $\mathcal{G}^B$, $\pi(\rho)$ is an $f_{\mathcal{M}}$-play of $\mathfrak{G}_{\mathcal{M}}$.

Now let ρ be a maximal $f_{\mathcal{P}}$-play and assume that it is not winning. If it is finite, it ends in a node u without any successor and $\pi(u) \in S_{\text{sim}}^*$, so $\pi(\rho)$ ends with win_1, hence as explained above $f_{\mathcal{M}}$ is not winning. If it is infinite, then either:

- $\mathfrak{F} = \mathsf{Reach}$, so ρ never visits an accepting configuration, therefore $\pi(\rho)$ never visits a node in $\mathfrak{W}$, which in turn means that win_0 cannot be reached and so $\pi(\rho)$ is not winning,
- $\mathfrak{F} = \mathsf{Büchi}$, so ρ only visits finitely many accepting configurations, so $\pi(\rho)$ also only visits finitely many nodes in $\mathfrak{W}$ so by definition of the ranking function the run is not winning, or
- $\mathfrak{F} = \mathsf{coBüchi}$, so ρ visits infinitely many accepting configurations, so does $\pi(\rho)$ for nodes in $\mathfrak{W}$, and again the run is not winning.

Then ρ is winning, and $f_{\mathcal{P}}$ is a winning strategy for Player 0 in $\mathcal{G}^B$.

$\boxed{\Leftarrow}$ Let $f_{\mathcal{P}}$ be a winning strategy of $\mathcal{G}^B$, we will build a strategy $f_{\mathcal{M}}$ of $\mathfrak{G}_{\mathcal{M}}$. Let $\hat{\rho}$ a play of $\mathfrak{G}_{\mathcal{M}}$ ending in a node of Player 0, and v be the last node in S_{sim} of $\hat{\rho}$. By Lemma 18, $\hat{\pi}(\hat{\rho})$ is a play of $\mathcal{G}^B$ ending in a node $u = \hat{\pi}(v)$.

If $\hat{\pi}(\hat{\rho})$ is also an $f_{\mathcal{P}}$-play, then since $f_{\mathcal{P}}$ is a winning strategy either $\mathfrak{F} = \mathsf{Reach}$ and $\hat{\pi}(\hat{\rho})$ is already a winning play, or u must have at least one successor. In the latter case, we let

$$u' = \begin{cases} f_{\mathcal{P}}(\hat{\pi}(\hat{\rho})) & \text{if } u \in V_0, \\ \text{some successor of } u & \text{if } u \in V_1. \end{cases}$$

By construction $\hat{\pi}(\hat{\rho}) \cdot u'$ is an $f_{\mathcal{P}}$-play and we let $\mathit{next}(\hat{\rho}) = \pi(\hat{\pi}(\hat{\rho}) \cdot u')$. In the former case, let $\mathit{next}(\hat{\rho}) = \pi(\hat{\pi}(\hat{\rho}))$, i.e. $\hat{\rho}$ followed by infinitely many nodes in win_0.

With the previous notations, we define $f_{\mathcal{M}}$ as follows:

$$f_{\mathcal{M}}(\hat{\rho}) = \begin{cases} v' & \text{if } \hat{\pi}(\hat{\rho}) \text{ is an } f_{\mathcal{P}}\text{-play, with } \hat{\rho} \cdot v' \text{ a prefix of } \mathit{next}(\hat{\rho}), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

By induction, if $\hat{\rho}$ is an $f_{\mathcal{M}}$-play, then $\hat{\pi}(\hat{\rho})$ is an $f_{\mathcal{P}}$-play: if $\hat{\pi}(\hat{\rho})$ is an $f_{\mathcal{P}}$-play then either $\hat{\rho} \cdot v'$ is a strict prefix of $\mathit{next}(\hat{\rho})$ in which case $\hat{\pi}(\hat{\rho} \cdot v') = \hat{\pi}(\hat{\rho})$, or $\hat{\pi}(\hat{\rho} \cdot v') = \hat{\pi}(\hat{\rho}) \cdot u'$ which is an $f_{\mathcal{P}}$-play.

Suppose there is an $f_{\mathcal{M}}$-play $\hat{\rho}$ that is maximal (i.e infinite) and not winning. That means either at some point $\hat{\rho}$ reached win_1 which is a sink state, or $\hat{\rho}$ visits S_{sim} infinitely often (as it is not possible for either player to stay in $\mathfrak{I}$ indefinitely).

Since win_1 is only accessible from nodes in V_0, the first case can only happen if there is some prefix $\hat{\rho}'$ of $\hat{\rho}$ such that $f_{\mathcal{M}}(\hat{\rho}')$ leads to win_1, meaning that $\mathit{next}(\hat{\rho}') = \pi(\hat{\pi}(\hat{\rho}') \cdot f_{\mathcal{P}}(\hat{\pi}(\hat{\rho}')))$ leads to win_1. By definition of π, this necessarily means that $f_{\mathcal{P}}(\hat{\pi}(\hat{\rho}')) \notin \mathcal{F}$ and has no successor. As this is a maximal $f_{\mathcal{P}}$-play, necessarily $\hat{\pi}(\hat{\rho}')$ must already be a winning play. In that case, by definition of next, $\mathit{next}(\hat{\rho}') = \pi(\hat{\pi}(\hat{\rho}'))$, which leads to win_0, by definition of π. Hence a contradiction.

In the second case, this means that $\hat{\pi}(\hat{\rho})$ is also infinite and an $f_{\mathcal{P}}$-play so it must be winning. Then either:

- $\mathfrak{F} = \mathsf{Reach}$, so $\hat{\pi}(\hat{\rho})$ visits some node $u = (c, p, r)$ where c is an accepting configuration. By definition of $\hat{\pi}$, we can deduce that $\pi(u) \in \mathfrak{W}$ is visited in $\hat{\rho}$. Then the only possible successor of $\pi(u)$ is a node in win_0, hence contradicting that $\hat{\rho}$ is not winning.
- $\mathfrak{F} = \mathsf{Büchi}$, so $\hat{\pi}(\hat{\rho})$ visits infinitely many accepting configurations, so $\hat{\rho}$ visits infinitely many nodes in $\mathfrak{W}$ and therefore is also winning.
- $\mathfrak{F} = \mathsf{coBüchi}$ and similarly $\hat{\pi}(\hat{\rho})$ visits finitely many accepting configurations, so $\hat{\rho}$ visits finitely many nodes in $\mathfrak{W}$ and therefore is winning.

Hence, $f_{\mathcal{M}}$ is a winning strategy.

This ends the proof of Lemma 12, and therefore, that CONTROL-DPG$_{rb}$ is decidable.

3.3.3 Lower Bound

We now prove that CONTROL-DFG$_{rb}$ is inherently non-elementary. Our lower-bound proof is inspired by [ABKS17], but we reduce from the satisfiability problem for first-order formulas on finite words, which is known to be non-elementary [Sto74].

For simplicity of proof, we use here a slightly different but equivalent syntax for FO formulas than what has been given in Section 2.3.2. Let $\mathcal{V}$ be a countably

infinite set of variables and Σ a finite alphabet. Formulas φ are built by the grammar $\varphi ::= a(x) \mid x < y \mid \neg(x < y) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \mid \forall x.\varphi$ where $x, y \in \mathcal{V}$ and $a \in \Sigma$. Their semantics is defined as in Section 2.3.2. Without loss of generality, we suppose that a formula φ is given in prenex normal form, which means that all quantifiers are put in the front of the formula.

We build a round-bounded DFG that is winning for System if and only if φ is satisfiable. In the first round of the game, Player 0 chooses a word w by creating a different process for each letter of w, each of them holding the corresponding letter in its local state. To prove that w is indeed a model of φ, the following rounds are devoted to the valuation of the variables appearing in φ, $\nu(x) = i$ being represented by memorizing the variable x in the local state of the i^{th} process. If x appears in the scope of a universal quantifier, the choice of the process is made by Player 1, otherwise it is made by Player 0. The last round is used to check the valuation of the variables. To this end, the players will inductively choose a subformula to check, until they reach an atomic proposition: If the subformula is a disjunction $\varphi_1 \vee \varphi_2$, Player 0 chooses either φ_1 or φ_2; if it is a conjunction, Player 1 chooses the next subformula. Finally, to verify whether $a(x)$ is satisfied, we check that there is a process with letter a and variable x in its local state. For $x < y$, we check that the process with x in its local state is eventually followed by a distinct process with y in its local state. This check is done during the same round, which guarantees that the positions corresponding to x and y are in the correct order. The number of states needed and the number of rounds are linearly bounded in the length of the formula. Here is the formalization and proof of this idea.

Let φ be a formula, $\mathrm{Cl}(\varphi)$ the set of subformulas (non-strict) of φ, and $\mathcal{V}_\varphi \subset \mathcal{V}$ the set of variables appearing in φ. Let us first define the DFS:

$$\mathcal{F} = (M_{\mathsf{glob}}, M_{\mathsf{loc}}, F_{\mathsf{glob}}, F_{\mathsf{loc}}, \mathsf{Reach})$$

with $M_{\mathsf{glob}} = (\Sigma', S, T_{\mathsf{glob}}, s^{init})$ and $M_{\mathsf{loc}} = (\Sigma', L, T_{\mathsf{loc}}, \ell^{init})$ defined as follows:

- $S = \{s^{init}, \mathrm{Guess}, \mathrm{Win}\} \cup \{\text{Win-if-}x \mid x \in \mathcal{V}_\varphi\} \cup \{\psi \mid \psi \in \mathrm{Cl}(\varphi)\} \cup \{\boxed{\psi} \mid \psi \in \mathrm{Cl}(\varphi)\}$
  The initial state is s^{init}, and $F_{\mathsf{glob}} = \{\mathrm{Win}\}$.
- $L = \{\ell^{init}, \mathrm{first}\} \cup \left(\Sigma \times 2^{\mathcal{V}_\varphi}\right)$, with initial state ℓ^{init} and $F_{\mathsf{loc}} = L$.
- The transitions are as follows:

#	T_{glob}	T_{loc}
1	$(s, \#, \text{Guess})$	$(\ell^{init}, \#, \text{first})$
2	$(\text{Guess}, a, \text{Guess})$	$(\ell^{init}, a, (a, \emptyset))$
3	$(\text{Guess}, \#, \varphi)$	$(\text{first}, \#, \text{first})$
4	$(\boxed{\psi}, \#, \psi)$	
5	$(?x.\psi, +x, \boxed{\psi})$ for $? \in \{\exists, \forall\}$	$((a, X), +x, (a, X \cup \{x\}))$
6	$(\psi_1 ? \psi_2, \#, \psi_1)$ for $? \in \{\vee, \wedge\}$	
7	$(\psi_1 ? \psi_2, \#, \psi_2)$ for $? \in \{\vee, \wedge\}$	
8	$(a(x), a_x, \text{Win})$	$((a, X), a_x, (a, X))$ if $x \in X$
9	$(\neg a(x), \bar{a_x}, \text{Win})$	$((b, X), \bar{a_x}, (b, X))$ if $x \in X$ and $b \neq a$
10	$(x < y, -x, \text{Win-if-}y)$	$((a, X), -x, (a, \emptyset))$ if $x \in X$
11	$(\neg(x < y), = y, \text{Win-if-}x)$	$((a, X), = y, (a, X))$ if $y \in X$
12	$(\text{Win-if-}x, = x, \text{Win})$	

Then let $B = |\mathcal{V}_\varphi| + 2$, and $\mathcal{G} = (\mathcal{F}, S^{\mathsf{s}}, S^{\mathsf{e}})$ where states of the form $\psi \wedge \psi'$ or $\forall x.\psi$ are in S^{e}, and all others are in S^{s}.

Lemma 19. *There is a winning strategy for System in $\mathcal{G}$ if and only if φ is satisfiable.*

Proof. Given a configuration $c = (s, \text{first}, (a_1, X_1), \dots, (a_n, X_n))$ of size $n+1$, we define the associated (partial) valuation $\nu(c)(x) = i$ if $x \in X_i$, which is well defined as there is no possible way in the game to have a single variable x in X_i and X_j if $i \neq j$. Conversely, given a state s, a word $w = a_1 \dots a_n$ and a valuation ν, the associated configuration is $c(s, w, \nu) = (s, \text{first}, (a, X_1), \dots, (a, X_n))$ where $X_i = \{x \mid \nu(x) = i\}$.

$\boxed{\Rightarrow}$ Let σ be a winning strategy for System. From the initial node (which belongs to System), σ will necessarily first do one transition of type 1, then n transitions of type 2 (for $n \in \mathbb{N}$), then one transition of type 3, reaching a node of the form $v_\varphi = ((\varphi, \text{first}, (a_1, \emptyset), \dots, (a_n, \emptyset)), 1, 2)$. We fix $w_\sigma = a_1 \dots a_n$. Let σ' be a strategy for Environment, and ρ the winning $(\sigma, \sigma')-$play. We show by recursion on the subformula ψ that for all nodes $v_\psi = (c_\psi, 1, r)$ with $c_\psi = (\psi, \text{first}, (a_1, X_1), \dots, (a_n, X_n))$ visited during ρ, we have $w_\sigma, \nu(c_\psi) \models \psi$.

First, note that if ψ is a term (or negated term), then necessarily v_ψ is reached during the last round $r = B = |\mathcal{V}_\varphi| + 2$ as it takes one round to reach v_φ and then $|\mathcal{V}_\varphi|$ rounds to go through every quantifier of φ.

- If $\psi = a(x)$, then as ρ is winning there is a process with local state (a_i, X_i) with $a_i = a$ and $x \in X_i$ in order to perform a transition of type 8 with label a_x, in other words $\nu(c_\psi)(x) = i$, so we have $w_\sigma, \nu(c_\psi) \models \psi$.
- Similarly for $\psi = \neg a(x)$, there is $1 \leq i \leq n$ such that $\nu(c_\psi)(x) = i$ and $a_i \neq a$.
- If $\psi = x < y$, let $v_1 = ((\text{Win-if-}y, \dots, (a_i, \emptyset), \dots), i{+}1, B)$ and $v_2 = ((\text{Win}, \dots), j + 1, B)$ such that ρ ends in $v_\psi v_1 v_2$ (transitions 10 then 12). Then we have $\nu(c_\psi)(x) = i$ and $\nu(c_\psi)(y) = j$. Since v_1 and v_2 are visited in the same round (see note above), then $i \leq j$. And since after v_1 $X_i = \emptyset$, we know that $i \neq j$, thus $i < j$.
- Similarly for $\psi = \neg(x < y)$, we have $\nu(c_\psi)(x) = i$ and $\nu(c_\psi)(y) = j$ with $j \leq i$ but this time no strict inequality.

- If $\psi = \psi_1 \vee \psi_2$, then let $i \in \{1, 2\}$ such that the next node has global state ψ_i (transition 6 or 7), as we know recursively that $w_\sigma, \nu(c_{\psi_i}) \models \psi_i$ and that $\nu(c_{\psi_i}) = \nu(c_\psi)$ (as no local state is changed during the transition), then $w_\sigma, \nu(c_\psi) \models \psi_i$, which in turn means that $w_\sigma, \nu(c_\psi) \models \psi$.

- Similarly if $\psi = \psi_1 \wedge \psi_2$, for every $i \in \{1, 2\}$ representing the choice of Environment we have $w_\sigma, \nu(c_{\psi_i}) \models \psi_i$ and $\nu(c_{\psi_i}) = \nu(c_\psi)$, thus $w_\sigma, \nu(c_\psi) \models \psi$.

- If $\psi = \exists x.\psi'$, let $v_1 = ((\boxed{\psi'}, \dots), i + 1, r)$ with $1 \leq i$ the successor node of v_ψ in ρ, and $v_2 = ((\psi', \dots), 1, r + 1)$ the successor of v_1 (transitions 5 then 4). By recursion we have that $w_\sigma, \nu(c_{\psi'}) \models \psi'$, and $\nu(c_{\psi'}) = \nu(c_\psi) \uplus \{x \to i\}$ by construction. Thus $w_\sigma, \nu(c_\psi) \models \psi$.

- If $\psi = \forall x.\psi'$, for all $1 \leq i \leq n$ we let $v_1^i = ((\boxed{\psi'}, \dots), i + 1, r)$ and $v_2^i = (c^i_{\psi'}, 1, r + 1)$ the i possible successors of v_ψ corresponding to Environment's choice. Similarly, we know that for all i $w_\sigma, \nu(c^i_{\psi'}) \models \psi'$ and $\nu(c^i_{\psi'}) = \nu(c_\psi) \uplus \{x \to i\}$. Thus $w_\sigma, \nu(c_\psi) \models \psi$.

From this we conclude that $w_\sigma, \nu(c_\varphi) \models \varphi$ and as $\nu(c_\varphi)$ is the empty valuation, then w_σ satisfies φ.

$\boxed{\Leftarrow}$ Now suppose that φ is satisfied by $w = a_1 \dots a_n$. We build σ_φ as follows. Let ρ be a run ending in v.

- If $v = ((\text{Guess}, \text{first}, (a_1, \emptyset), \dots, (a_k, \emptyset)), k + 1, 1)$ for $0 \leq k < n$ then $\sigma_\varphi(\rho) = ((\text{Guess}, \text{first}, (a_1, \emptyset) \dots, (a_k, \emptyset), (a_{k+1}, \emptyset)), k + 2, 1)$. If $k = n$, then $\sigma_\varphi(\rho) = ((\varphi, \text{first}, (a_1, \emptyset), \dots, (a_k, \emptyset)), 1, 2)$

- If $v = (c_\psi, 1, r)$ with $c_\psi = (\exists x.\psi', \text{first}, (a_1, X_1), \dots, (a_n, X_n))$, then assuming that $w, \nu(c_\psi) \models \psi$ we pick one $i \leq n$ such that $w, \nu(c_\psi) \uplus \{x \to i\} \models \psi'$ and we define $\sigma_\varphi(\rho) = ((\boxed{\psi'}, (a_1, X'_1), \dots, (a_n, X'_n)), i + 1, r)$ with $X'_i = X_i \uplus \{x\}$ and $X'_j = X_j$ for $j \neq i$.

- If $v = (c_\psi, 1, r)$ with $c_\psi = (\psi_1 \vee \psi_2, \dots)$, then assuming $w, \nu(c_\psi) \models \psi$ we know that there is at least one $i \in \{1, 2\}$ such that $w, \nu(c_\psi) \models \psi_i$. We pick one such i, and define $\sigma_\varphi(\rho) = ((\psi_i, \dots), 1, r)$.

- For all other cases, there is at most one transition available so σ_φ is defined unambiguously.

Let σ' be a strategy for Environment, and $\rho = v_0 \dots v_m$ the resulting $(\sigma_\varphi, \sigma')$-play. We show that ρ is winning.

By definition of σ_φ and since v_0 to v_n are owned by System, we have that $v_{n+1} = (c(\varphi, w, \emptyset), 1, 2)$. Let $k \in \{n + 1, n + 3, \dots, n + 2 \cdot (|\mathcal{V}_\varphi| - 1)\}$. We have the following two properties:

1. If $v_k = (c(\exists x.\psi', w, \nu), 1, r)$ such that $w, \nu \models \exists x.\psi'$, then by construction of σ_φ we have $v_{k+1} = (c(\boxed{\psi'}, w, \nu'), i + 1, r)$ and $v_{k+2} = (c(\psi', w, \nu'), 1, r + 1)$ for some $1 \leq i \leq n$ and $\nu' = \nu \uplus \{x \to i\}$, and furthermore $w, \nu' \models \psi'$.

2. If $v_k = (c(\forall x.\psi', w, \nu), 1, r)$ such that $w, \nu \models \forall x.\psi'$, then for some $1 \leq i \leq n$ (defined by σ') we have $\nu' = \nu \uplus \{x \to i\}$ such that $v_{k+1} = (c(\boxed{\psi'}, w, \nu'), i+1, r)$ and $v_{k+2} = (c(\psi', w, \nu'), 1, r+1)$. By definition since $w, \nu \models \forall x.\psi'$, we deduce that $w, \nu' \models \psi'$.

By those two properties, combined with the fact that $v_{n+1} = (c(\varphi, w, \emptyset), 1, 2)$, we deduce that $v_{n+2\cdot|\mathcal{V}_\varphi|} = (c(\psi', w, \nu), 1, B)$ where ψ' has no quantifiers and $w, \nu \models \psi'$.

Let $k \geq n + 2 \cdot |\mathcal{V}_\varphi|$, again we have two similar-looking properties:

1. If $v_k = (c(\psi_1 \vee \psi_2, w, \nu), 1, B)$ and $w, \nu \models \psi_1 \vee \psi_2$ then by definition of σ_φ, $v_{k+1} = (c(\psi_i, w, \nu), 1, B)$ with $i \in \{1, 2\}$ and $w, \nu \models \psi_i$.

2. If $v_k = (c(\psi_1 \wedge \psi_2, w, \nu), 1, B)$ and $w, \nu \models \psi_1 \wedge \psi_2$ then for some $i \in \{1, 2\}$ defined by σ', $v_{k+1} = (c(\psi_i, w, \nu), 1, B)$. By definition of satisfiability, we also have that $w, \nu \models \psi_i$.

Using those two properties, we deduce that there exists $m' \geq n + 2 \cdot |\mathcal{V}_\varphi|$ such that $v_{m'} = (c(t, w, \nu), 1, B)$ where t is a term or a negated term such that $w, \nu \models t$. Here m' depends not only on φ but also on both strategies σ_φ and σ'. There are 4 possible cases for t:

1. $t = a(x)$: as $w, \nu \models t$ we know that $a_{\nu(x)} = a$, and $v_{m'} = (c(t, w, \nu), 1, B)$ so $v_{m'+1} = (c(\text{Win}, w, \nu), \nu(x) + 1, B)$.

2. $t = \neg a(x)$: similarly, we have $v_{m'+1} = (c(\text{Win}, w, \nu), \nu(x) + 1, B)$.

3. $t = x < y$: we know that $\nu(x) < \nu(y)$ so $v_{m'+1} = (c(\text{Win-if-}y, w, \nu'), \nu(x), B)$ where $\nu' = \nu - \{\{x' \to \nu(x)\} \mid x' \in \mathcal{V}_\varphi\}$. Since $\nu(x) \neq \nu(y)$, $\nu'(y) = \nu(y) > \nu(x)$. So $v_{m'+2} = (c(\text{Win}, w, \nu'), \nu(y), B)$.

4. $t = \neg(x < y)$: in this case $\nu(y) \leq \nu(x)$ and we have $v_{m'+1} = (c(\text{Win-if-}x, w, \nu), \nu(y), B)$ and $v_{m'+2} = (c(\text{Win}, w, \nu), \nu(x), B)$.

Every case ends in an accepting node, therefore ρ is winning. □

3.4 Context-bounded Control

In this section, we show that relaxing the notion of rounds quickly leads to undecidability. It should be noted that our undecidability proof also applies to the notion of context bounds introduced in [ABQ11].

3.4.1 Context-Bounded Runs

We now define *context-bounded runs*. A context is less restrictive than a round. It just requires that every process intervenes once, without fixing a particular order on processes.

Let $\mathcal{P} = ((\Sigma, S, T_{\mathsf{glob}}, s^{init}), (\Sigma, \Gamma, L, T_{\mathsf{loc}}, \ell^{init}), F_{\mathsf{glob}}, F_{\mathsf{loc}}, \mathfrak{F})$ be a DPS. As with round-bounded runs, we define *extended configurations* that contain necessary information to count contexts as follows: an extended configuration is of the form $v = (c, P, p, r)$ where:

- c is a configuration of $\mathcal{P}$ of size, say, k,
- $P \subseteq \{1, \ldots, k\}$ is the set of processes that performed a transition during the current context,
- $p \in P \cup \{0\}$ is the last process that performed an action (or 0 at the beginning), and
- $r \geq 1$ is the current context number.

The initial extended configuration is $v^{init} = (c^{init}, \emptyset, 0, 1)$. A new context is started only when a process that is in $P \setminus p$ performs an action. In other words, if (c, P, p, r) is an extended configuration and c' is an (a, i)-successor of c, then the corresponding (a, i)-successor extended configuration is (c', P', i, r') with:

- $P' = \{i\}$ and $r' = r + 1$ if $i \in P \setminus \{p\}$,
- $P' = P \cup \{i\}$ and $r' = r$ otherwise.

Now let $B \geq 1$ be a *bound*. By $\mathcal{P}^B_{\text{cb}}$, we denote the *context-bounded semantics* which is the transition system where nodes are extended configurations with a context number up to B, and transitions are defined as above. A run ρ of $\mathcal{P}$ is B-context bounded if it is a run of $\mathcal{P}^B_{\text{cb}}$, or in other words, if this run does not use more than B contexts.

Relation to round-bounded runs.

Note that if a run is B-round bounded for some B, then it is trivially B-context bounded too. Conversely for any $n \in \mathbb{N}$, there are 2-context bounded runs that are not n-round bounded: for instance, a run where processes 1 to n do one transition each in the first half, followed in the second half by one transition from processes $n - 1$ down to 1. Such a run is 2-context bounded (one for each half), but it needs at least $n + 1$ rounds to be done.

Context-bounded control.

Let $\mathcal{G}$ be a DPG and $B \geq 1$ be a bound. The context-bounded game denoted by $\mathcal{G}^B_{\text{cb}}$ is the game played on the context-bounded semantics of the DPS, defined in a similar way to round-bounded games. The control problem on these games is also defined similarly:

CONTROL-DPG$_{cb}$	
Input:	A DPG $\mathcal{G}$, a bound $B \in \mathbb{N}_>$ (given in unary)
Question:	Is there a winning strategy for System in $\mathcal{G}^B_{\text{cb}}$?

Context-bounded games for DFG and their respective control problem CONTROL-DFG$_{cb}$ are defined analogously.

3.4.2 Undecidabilty for Context-Bounded Runs

We show that even for DFG, and even with a fixed bound, the control problem is undecidable. This shows that relaxing the round-bounded constraint even a little easily leads to undecidability.

Theorem 20. *CONTROL-DFG$_{cb}$ is undecidable, even if we fix $B = 2$.*

The rest of this section is devoted to the proof of this theorem.

We provide a reduction from the halting problem for 2-counter machines, whose definition we recall in the following.

A *two-counter machine* (2CM) with counters c_1 and c_2 is given by a tuple $M = (Q, T, q_0, q_h)$, where Q is the finite set of states and $T \subseteq Q \times \mathsf{Op} \times Q$ is the transition relation where the set of operations is defined as $\mathsf{Op} = \{\mathsf{c}_i{+}{+}\,,\ \mathsf{c}_i{-}{-}\,,\ \mathsf{c}_i{=}{=}0 \mid i \in \{1, 2\}\}$. As expected, $\mathsf{c}_i{+}{+}$ increments counter c_i, while $\mathsf{c}_i{-}{-}$ decrements it, and $\mathsf{c}_i{=}{=}0$ checks whether its value is 0. Moreover, there are a distinguished initial state $q_0 \in Q$ and a halting state $q_h \in Q$.

The behavior of M is described in terms of a global transition relation over configurations $\gamma = (q, \nu_1, \nu_2) \in Q \times \mathbb{N} \times \mathbb{N}$ where q is the current state and ν_1, ν_2 are the current counter values. Every transition $t \in T$ defines a binary relation $\vdash_t$ on configurations letting $(q, \nu_1, \nu_2) \vdash_t (q', \nu_1', \nu_2')$ if there is $i \in \{1, 2\}$ and $\nu'_{3-i} = \nu_{3-i}$ such that one of the following conditions hold:

- $t = (q, \mathsf{c}_i{+}{+}, q')$ and $\nu_i' = \nu_i + 1$,
- $t = (q, \mathsf{c}_i{-}{-}, q')$ and $\nu_i' = \nu_i - 1$, or
- $t = (q, \mathsf{c}_i{=}{=}0, q')$ and $\nu_i = \nu_i' = 0$.

An (M-)run is a sequence of the form $\gamma_0 \vdash_{t_1} \gamma_1 \vdash_{t_2} \ldots \vdash_{t_n} \gamma_n$ where $\gamma_0 = (q_0, 0, 0)$. The run is successful (or *halting*) if $\gamma_n \in F = \{q_h\} \times \mathbb{N} \times \mathbb{N}$. Now, the 2CM halting problem is to decide whether there is a successful run. It is well known that this problem is undecidable [Min67].

Let $M = (Q, T, q_0, q_h)$ be a 2CM. We define a DFG $\mathcal{G}$ with the following intuition. In the first context, System will simulate a run of the 2CM. The global state of the game will be the state of the 2CM. To encode the values of the counters, there are two local states ℓ_i and $\bar{\ell}_i$ for $i \in \{1, 2\}$, and the value of counter i will be encoded as the number of processes with local state ℓ_i minus the number of processes with local state $\bar{\ell}_i$. To simulate a transition $(q, \mathsf{c}_i{+}{+}, q')$, System will change the global state from q to q' and create a new process with local state ℓ_i. Similarly, for a transition $(q, \mathsf{c}_i{-}{-}, q')$, System will change the global state from q to q' and create a new process with local state $\bar{\ell}_i$. Finally, a transition $(q, \mathsf{c}_i{=}{=}0, q')$ is simulated by changing the global state from q to q' and creating a new process with a dummy local state $\ell_\perp$ that is not counted in the encoding of the values of c_1 and c_2. All local states are accepting and only q_h is an accepting global state, so System wins if she can simulate a run of the 2CM leading to q_h.

However, we must ensure that System does not cheat during the simulation, that is that System does not decrement counter i if its value is 0 or takes a zero-test transition when its value is not 0. To that end, whenever System simulates a decrement or a zero-test transition, we leave the possibility for Environment to

#	T_{glob}	T_{loc}	
1	(q, inc_i, q')	$(\ell^{init}, inc_i, \ell_i)$	for all $(q, \mathsf{c}_i{+}{+}, q') \in T$
2	$(q, dec_i, ?\mathrm{dec}^i_{(q,q')})$	$(\ell^{init}, dec_i, \bar{\ell}_i)$	for all $(q, \mathsf{c}_i{-}{-}, q') \in T$
3	$(?\mathrm{dec}^i_{(q,q')}, nop, q')$	$(\ell^{init}, nop, \ell_\perp)$	
4	$(?\mathrm{dec}^i_{(q,q')}, nop, \mathrm{vdec}^i_1)$		
5	$(q, zero_i, ?\mathrm{zero}^i_{(q,q')})$	$(\ell^{init}, zero_i, \ell_\perp)$	for all $(q, \mathsf{c}_i{=}{=}0, q') \in T$
6	$(?\mathrm{zero}^i_{(q,q')}, nop, q')$		
7	$(?\mathrm{zero}^i_{(q,q')}, nop, \mathrm{vzero}^i_1)$		
8	$(\mathrm{vdec}^i_1, \bar{v}_i, \mathrm{vdec}^i_2)$	$(\bar{\ell}_i, \bar{v}_i, \ell_\perp)$	for all $i \in \{1, 2\}$
9	$(\mathrm{vdec}^i_2, v_i, \mathrm{vdec}^i_1)$	$(\ell_i, v_i, \ell_\perp)$	
10	$(\mathrm{vdec}^i_1, nop, win)$		
11	$(\mathrm{vzero}^i_1, v_i, \mathrm{vzero}^i_2)$		for all $i \in \{1, 2\}$
12	$(\mathrm{vzero}^i_2, \bar{v}_i, \mathrm{vzero}^i_1)$		
13	$(\mathrm{vzero}^i_1, \bar{v}_i, \mathrm{vzero}^i_3)$		
13	$(\mathrm{vzero}^i_3, v_i, \mathrm{vzero}^i_1)$		
14	$(\mathrm{vzero}^i_1, nop, win)$		

Table 3.1: Transitions of $\mathcal{F}$

claim that the transition was incorrectly taken. When that happens, the simulation is stopped and a verification is started. This verification phase uses another context, and the game always ends after this phase (thus 2 contexts are enough for the game). If the transition was a decrement of counter i, then Environment and System will alternately make a transition with a process in state $\bar{\ell}_i$ and ℓ_i respectively, with Environment aiming to prove that there are more $\bar{\ell}_i$ than ℓ_i and System aiming to disprove that. Eventually, the player who cannot make a transition anymore loses the game. Similarly, if the transition was a zero-test, Environment will try to prove that there is an unequal number of ℓ_i and $\bar{\ell}_i$, and System will try to disprove it. Therefore, System's only way to win the game is to correctly simulate an accepting run of the 2CM.

Formally, let us define $\mathcal{F} = ((\Sigma, S, T_{\mathsf{glob}}, s^{init}), (\Sigma, L, T_{\mathsf{loc}}, \ell^{init}), F_{\mathsf{glob}}, F_{\mathsf{loc}}, \mathsf{Reach})$ as follows.

- $S = \quad Q \cup \{win\}$
  $\cup \quad \{?\mathrm{dec}^i_{(q,q')}, ?\mathrm{zero}^i_{(q,q')} \mid i \in \{1, 2\}, q, q' \in Q\}$
  $\cup \quad \{\mathrm{vdec}^i_j \mid i \in \{1, 2\}, j \in \{1, 2\}\}$
  $\cup \quad \{\mathrm{vzero}^i_j \mid i \in \{1, 2\}, j \in \{1, 2, 3\}\}$,
- $s^{init} = q_0$ and $F_{\mathsf{glob}} = \{q_h, win\}$,
- $L = \{\ell^{init}, \ell_\perp\} \cup \{\ell_i, \bar{\ell}_i \mid i \in \{1, 2\}\}$,
- $\ell^{init} = \ell^{init}$, and $F_{\mathsf{loc}} = L$,
- and finally, transitions can be found in Table 3.1.

The DFG $\mathcal{G}$ is defined as $\mathcal{G} = (\mathcal{F}, S^{\mathsf{s}}, S^{\mathsf{e}})$ with Environment states being

$$S^{\mathsf{e}} = \{?\mathrm{dec}^i_{(q,q')}, ?\mathrm{zero}^i_{(q,q')} \mid i \in \{1, 2\}, q, q' \in Q\} \cup \{\mathrm{vdec}^i_1, \mathrm{vzero}^i_1 \mid i \in \{1, 2\}\}$$

and $S^{\mathsf{s}} = S \setminus S^{\mathsf{e}}$. Finally, we take $B = 2$. This ends the definition of $\mathcal{G}^B_{\mathrm{cb}}$. Refer to Figure 3.10 for an illustration.

Lemma 21. *There is an accepting run in M iff System has a winning strategy in $\mathcal{G}^B_{\mathrm{cb}}$.*

Proof. To avoid possible confusions, a configuration of the 2CM M may be referred as an M-configuration and will always be noted γ, whereas a configuration of the game $\mathcal{G}^B_{\mathrm{cb}}$ will be referred to as c. Moreover, runs of M will be denoted by ρ and plays of $\mathcal{G}^B_{\mathrm{cb}}$ by π.

For any $\mathcal{G}^B_{\mathrm{cb}}$-configuration $c = (s, \ell^1, \dots, \ell^p)$ and $i \in \{1, 2\}$, let $n_i(c) = |\{1 \leq j \leq p \mid \ell^j = \ell_i\}| - |\{1 \leq j \leq p \mid \ell^j = \bar{\ell}_i\}|$. Let also $\min_i(c) = \min\{j \mid \ell^j = \ell_i\}$ if it exists.

One can build from any M-run ρ a corresponding $\mathcal{G}^B_{\mathrm{cb}}$-play $\pi(\rho)$ inductively in the following way:

- $\pi(\gamma_0) = ((q_0), \emptyset, 0, 1)$,
- if $\pi(\rho)$ is defined and ends in $(c, \{1, \dots, p\}, p, 1)$ with $c = (q, \ell^1, \dots, \ell^p)$, then $\pi(\rho \vdash_t \gamma) =$

$$\begin{cases} \Big(\pi(\rho) \cdot ((q', \ell^1, \dots, \ell^p, \ell_i), \{1, \dots, p+1\}, p+1, 1) \Big) & \text{if } t = (q, \mathsf{c}_i{+}{+}, q') \\ \left(\begin{array}{ll} \pi(\rho) & \cdot ((?\mathrm{dec}^i_{(q,q')}, \ell^1, \dots, \ell^p, \bar{\ell}_i), \{1, \dots, p+1\}, p+1, 1) \\ & \cdot ((q', \ell^1, \dots, \ell^p, \bar{\ell}_i, \ell_\perp), \{1, \dots, p+2\}, p+2, 1) \end{array} \right) & \text{if } t = (q, \mathsf{c}_i{-}{-}, q') \\ \left(\begin{array}{ll} \pi(\rho) & \cdot ((?\mathrm{zero}^i_{(q,q')}, \ell^1, \dots, \ell^p, \ell_\perp), \{1, \dots, p+1\}, p+1, 1) \\ & \cdot ((q', \ell^1, \dots, \ell^p, \ell_\perp, \ell_\perp), \{1, \dots, p+2\}, p+2, 1) \end{array} \right) & \text{if } t = (q, \mathsf{c}_i{=}{=}0, q'). \end{cases}$$

This construction is such that for any ρ ending in $\gamma = (q, \nu_1, \nu_2)$, we have that $\pi(\rho)$ ends in $(c, P, p, 1)$ with $c = (q, \dots)$ such that $n_1(c) = \nu_1$ and $n_2(c) = \nu_2$. Remark also that $\pi(\rho)$ is winning for System iff q_h is visited in ρ.

We define a strategy σ_{vdec} as follows. If $c = (\mathrm{vdec}^i_2, \ell^1, \dots, \ell^p)$ is a $\mathcal{G}^B_{\mathrm{cb}}$-configuration such that $m = \min_i(c)$ exists (that is, there is at least one process in state ℓ_i), then $\sigma_{\mathrm{vdec}}(c, P, p', 2) = ((\mathrm{vdec}^i_1, \hat{\ell}^1, \dots, \hat{\ell}^p), P \cup \{m\}, m, 2)$ with $\hat{\ell}^m = \ell_\perp$, and $\hat{\ell}^j = \ell^j$ for all $j \neq m$. In all other cases, σ_{vdec} gives an arbitrary successor node. Let $c = (\mathrm{vdec}^i_1, ...)$ such that $n_i(c) \geq 0$, that is there are at least as many processes in local state ℓ_i than in local state $\bar{\ell}_i$. Then it is easy to see that σ_{vdec} is a winning strategy for System from node $(c, \{1, \dots, p\}, p, 1)$, as there will always be at least one process in state ℓ_i for System to make a transition until Environment is forced to go from global state vdec^i_1 to *win* because there are no more processes in state $\bar{\ell}_i$. Conversely, if $n_i(c) < 0$, then System cannot win from $(c, \{1, \dots, p\}, p, 1)$ because Environment can force System to exhaust all processes in state ℓ_i until there are no more left and then be stuck in vdec^i_2.

Similarly, one can build a strategy σ_{vzero} such that for all $c = (\mathrm{vzero}^i_1, \dots)$, System is winning from $(c, \{1, \dots, p\}, p, 1)$ iff $n_i(c) = 0$.

We are now ready to prove Lemma 21.

$\boxed{\Rightarrow}$ Let $\rho = \gamma_0 \vdash_{t_1} \cdots \vdash_{t_k} \gamma_k$ be an accepting M-run. We define a (memoryless) strategy σ for System in $\mathcal{G}^B_{\mathrm{cb}}$ that simulates ρ as follows:

- If $(c, P, p, 1)$ is the last node of $\pi(\gamma_0 \vdash_{t_1} \cdots \vdash_{t_j} \gamma_j)$ for some $j \in \{0, \dots, k-1\}$, then $\sigma(c, P, p, 1)$ is its successor in $\pi(\rho)$.

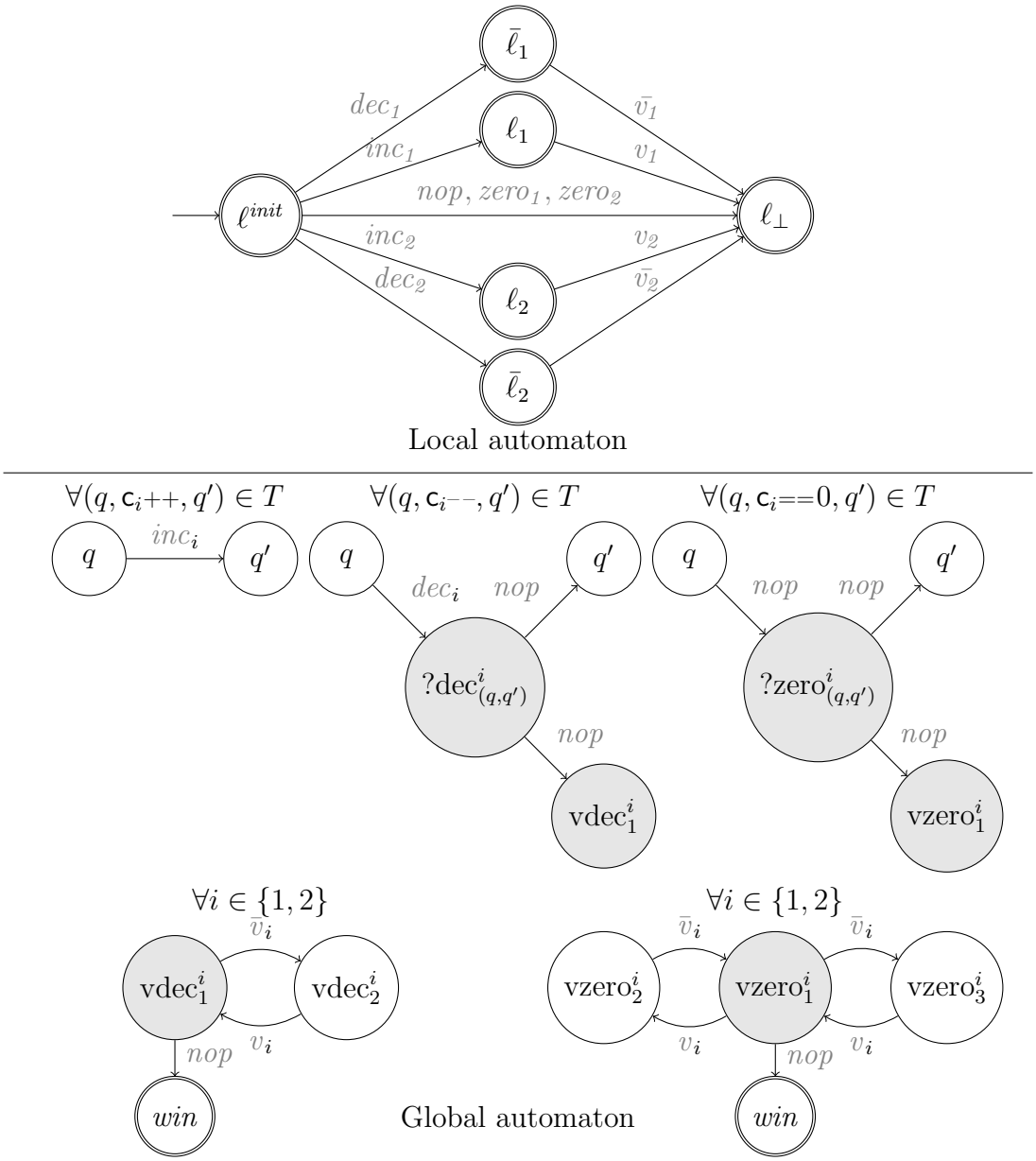

Figure 3.10: Construction of $\mathcal{G}^B_{\mathrm{cb}}$. Global states belonging to Environment are drawn with a light gray background. Not pictured: q_0 is the initial global state and q_h is accepting.

- If $c = (\text{vdec}_2^i, \dots)$ then σ follows σ_{vdec}.
- If $c = (\text{vzero}_2^i, \dots)$ or $c = (\text{vzero}_3^i, \dots)$ then σ follows σ_{vzero}.
- Otherwise σ gives an arbitrary successor.

Suppose that σ is not winning, let σ' be a strategy for Environment such that the maximal (σ, σ')-play π is not winning. There are two cases to study:

If global states vdec_1^i and vzero_1^i are not visited in π for both $i = 1$ and $i = 2$, then necessarily $\pi = \pi(\rho)$. However as ρ is an accepting M-run, then $\pi(\rho)$ is winning.

In the other case, suppose that vdec_1^i is visited in π. Up until visiting vdec_1^i, the play was simulating a prefix of ρ. Then necessarily π is of the form

$$\begin{aligned}\pi \;=\; & \pi(\gamma_0 \vdash_{t_1} \cdots \vdash_{t_j} \gamma_j) \cdot ((?\text{dec}^i_{(q,q')}, \ell^1, \dots, \ell^p, \bar{\ell}_i), \{1, \dots, p+1\}, p+1, 1) \\ & \cdot ((\text{vdec}^i_1, \ell^1, \dots, \ell^p, \bar{\ell}_i, \ell_\perp), \{1, \dots, p+2\}, p+2, 1) \cdot \pi'\end{aligned}$$

with $j < k$, $t_{j+1} = (q, \mathsf{c}_i{--}, q')$, and π' is σ_{vdec}-compatible by definition of σ. Moreover, $\pi(\gamma_0 \vdash_{t_1} \cdots \vdash_{t_j} \gamma_j)$ ends in $(c, \{1, \dots, p\}, p, 1)$ such that with $c = (q, \ell^1, \dots, \ell^p)$ and $\gamma_j = (q, \nu_1, \nu_2)$, we have that $n_i(c) = \nu_i > 0$, otherwise t_{j+1} could not have been taken in ρ. Therefore, $n_i(\text{vdec}^i_1, \ell^1, \dots, \ell^p, \bar{\ell}_i, \ell_\perp) = n_i(c) - 1 \geq 0$. Then because the rest of the play π' follows σ_{vdec}, we showed that π is a winning. The reasoning is similar if vzero_1^i is visited instead of vdec_1^i.

Thus we get a contradiction, so σ is a winning strategy for System.

$\boxed{\Leftarrow}$ Let σ be a winning strategy of System in $\mathcal{G}^B_{\text{cb}}$, and let π be the σ-compatible maximal play when put against the strategy of Environment that never goes to vdec_1^i or vzero_1^i. Let $c_0, c_1, \dots$ be the configurations visited during π. For all j such that c_j is of the form $(q, \dots)$, one can build a valid M-run $\rho(c_0, \dots, c_j)$ that ends in the M-configuration $\gamma = (q, n_1(c_j), n_2(c_j))$ in the following way: first we let $\rho(c_0) = \gamma_0 = (q_0, 0, 0)$, which satisfies the conditions above. Then if $\rho(c_0, \dots, c_j) = \gamma_0 \vdash_{t_1} \cdots \vdash_{t_k} \gamma_k$ has been defined with $c_j = (q, \ell^1, \dots, \ell^p)$ and $\gamma_k = (q, \nu_1, \nu_2)$ which satisfies the conditions, then there are three possible successors to consider:

- If $c_{j+1} = (q', \ell^1, \dots, \ell^p, \ell_i)$ then there exists $t = (q, \mathsf{c}_i{++}, q') \in T$. We then define $\rho(c_0, \dots, c_{j+1}) = \rho(c_0, \dots, c_j) \vdash_t (q', \nu'_1, \nu'_2)$ with $\nu'_i = \nu_i + 1$ and $\nu'_{3-i} = \nu_{3-i}$ which satisfies the conditions as $n_i(c_{j+1}) = n_i(c_j) + 1 = \nu_i + 1 = \nu'_i$ and $n_{3-i}(c_{j+1}) = n_{3-i}(c_j) = \nu_{3-i} = \nu'_{3-i}$.
- If $c_{j+1} = (?\text{dec}^i_{(q,q')}, \ell^1, \dots, \ell^p, \bar{\ell}_i)$ and $c_{j+2} = (q', \ell^1, \dots, \ell^p, \bar{\ell}_i, \ell_\perp)$, then let $t = (q, \mathsf{c}_i{--}, q') \in T$. We define $\rho(c_0, \dots, c_{j+2}) = \rho(c_0, \dots, c_j) \vdash_t (q', \nu'_1, \nu'_2)$ with $\nu'_i = \nu_i - 1$ and $\nu'_{3-i} = \nu_{3-i}$. This is a valid M-run as $\nu_i > 0$, otherwise we would have $n_i(c_{j+1}) = n_i(c_j) - 1 = \nu_i - 1 < 0$ in which case Environment could have chosen to go to $c'_{j+2} = (\text{vdec}^i_1, \ell^1, \dots, \ell^p, \bar{\ell}_i, \ell_\perp)$ which is losing for System because $n_i(c'_{j+2}) = n_i(c_{j+1}) < 0$, contradicting that σ is a winning strategy. Moreover, $\nu'_i = n_i(c_{j+2})$ for $i \in \{1, 2\}$, so this run satisfies the required conditions.
- If $c_{j+1} = (?\text{zero}^i_{(q,q')}, \ell^1, \dots, \ell^p, \ell_\perp)$ and $c_{j+2} = (q', \ell^1, \dots, \ell^p, \ell_\perp, \ell_\perp)$, then let $t = (q, \mathsf{c}_i{==}0, q') \in T$ and $\rho(c_0, \dots, c_{j+2}) = \rho(c_0, \dots, c_j) \vdash_t (q', \nu_1, \nu_2)$. Again this is a valid M-run because $\nu_i = 0$, otherwise Environment would win by going to $c'_{j+2} = (\text{vdec}^i_1, \ell^1, \dots, \ell^p, \ell_\perp, \ell_\perp)$. Since $\nu_i = n_i(c_j) = n_i(c_{j+2})$ for $i \in \{1, 2\}$, the conditions are also satisfied.

Therefore, π is of the form $\pi(\rho)$ for some valid M-run ρ. As π is winning, we deduce that ρ is an accepting run of M. □

Chapter 4

Synthesis for First-order Specifications

In this chapter, we study the synthesis problem, which as explained in the introduction can be seen as a special case of the control problem where the partially defined system given as input is a simple system where all actions are allowed at any time. Therefore, an instance of the problem is simply given by a (finite) alphabet of actions and the specification. Executions here are represented as data words, and for specifications we use first-order logic (abbreviated as FO), whose satisfiability problem was studied in [BDM+11]. The goal is to extend these results to synthesis, at least for fragments of the whole logic.

Moreover, in this chapter we take the parameterized approach to the problem instead of the dynamic one: for each execution, there is a finite set of processes that can participate in the execution, although that set is not known in advance and there is no bound on its size. This ensures a finer control on the processes involved compared to the dynamic case where processes can be added dynamically during an execution. The synthesis problem we define will therefore be parameterized by two constraints:

- the fragment of FO in which the specification is given, and
- the cardinality of the set of processes.

This chapter is organized as follows. In Section 4.1 we formally define executions and the parameterized synthesis problem as well as important notions. Then we study two fragments of FO in the next two sections: Section 4.2 focuses on the two-variable fragment FO^2, and Section 4.3 is about the fragment FO[$\sim$] where only the data equality predicate can be used.

4.1 Preliminaries

4.1.1 Executions and first-order logic

For this section, let us fix an alphabet $A = A_{\mathsf{s}} \uplus A_{\mathsf{e}}$ of *system* and *environment actions*. Let us also fix $\mathbb{P} = \mathbb{P}_{\mathsf{s}} \uplus \mathbb{P}_{\mathsf{e}} \uplus \mathbb{P}_{\mathsf{se}}$, three finite sets of *system*, *environment*, and *mixed* processes respectively. We denote by $\mathbb{T} = \{\mathsf{s}, \mathsf{e}, \mathsf{se}\}$ the set of *process types*.

The idea behind this partition is that both System and Environment do not necessarily have access to all processes, but can only affect *some* of those: System can only affect system and mixed processes, Environment can do the same to environment and mixed processes. This is a finer approach than declaring that all processes are mixed processes, and allows to better model some systems. For instance, with this we can model distributed computing systems where Environment represents user inputs given during the execution of the computing, but where only some machines accept user inputs while others cannot be interacted with directly.

Accordingly, let $\Sigma_{\mathsf{s}} = A_{\mathsf{s}} \times (\mathbb{P}_{\mathsf{s}} \cup \mathbb{P}_{\mathsf{se}})$ be the set of *system events* and $\Sigma_{\mathsf{e}} = A_{\mathsf{e}} \times (\mathbb{P}_{\mathsf{e}} \cup \mathbb{P}_{\mathsf{se}})$ be the set of *environment events*, and let us denote by $\Sigma = \Sigma_{\mathsf{s}} \cup \Sigma_{\mathsf{e}}$ their union. A $\mathbb{P}$-execution is a word $w \in \Sigma^{\infty}$, or in other words, w is a data word whose process identities are contained in $\mathbb{P}$ and whose actions respect process types (system actions on system or mixed processes, and environment actions on environment or mixed processes).

To define specifications over $\mathbb{P}$-executions, we use first-order logic formulas as defined in Section 2.3.2. However, with the logic defined as it is, one can only specify properties over processes that perform at least one action during the execution. Conversely, it is not possible to specify anything about processes that are not part of the execution. So even simple properties such as "All processes must perform at least one action" cannot be expressed. Moreover, there is no way to know what the type of a given process is other than looking at what kind of actions this process performed during the execution. If there is at least one system and one environment action performed by the same process, then it is obviously a mixed process, but what if there is only, say, system actions? Is that process a system process, or a mixed process that simply did not perform any environment action?

With these in mind, we slightly modify the syntax of first order logic as follows:

$$\varphi ::= a(x) \mid \theta(x) \mid x = y \mid \mathrm{succ}(x, y) \mid x < y \mid x \sim y \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi$$

with $x, y \in \mathcal{V}$, $a \in \Sigma$, and $\theta \in \mathbb{T}$. The three new predicates $\theta(x)$ are used to indicate that the process if of type θ. Here are the changes to the semantics of formulas:

- When evaluating a formula φ over a $\mathbb{P}$-execution w, variables can be interpreted either as a position of w or a process in $\mathbb{P}$: a valuation ν is now a partial function from $\mathcal{V}$ to $Pos(w) \cup \mathbb{P}$. Given a valuation ν and a variable x, we denote by $p^{\nu}_{x} \in \mathbb{P}$ the process which is either $\nu(x)$ if $\nu(x) \in \mathbb{P}$, or p if $\nu(x) \in Pos(w)$ with $w[\nu(x)] = (a, p)$.

- Formulas of the form $a(x)$, $\mathrm{succ}(x, y)$, $x < y$, and $x = y$ can only be satisfied if x and y are interpreted as positions of w (since they do not make sense if x and y are interpreted as processes).

- However, formulas of the form $x \sim y$ are satisfied if $p^{\nu}_{x} = p^{\nu}_{y}$.

- Finally, we add the following rule for formulas of the form $\theta(x)$:
  $(w, \nu) \models \theta(x)$ if $p^{\nu}_{x} \in \mathbb{P}_{\theta}$.

Example 18. We can now express properties such as

$$\varphi_1 = \forall x.\, [\mathsf{s}(x) \Rightarrow \exists y.y \sim x \wedge a(y)]$$

which states that every system process of $\mathbb{P}_{\mathsf{s}}$ has performed (at least) an a, or

$$\varphi_2 = \exists x.\, [\mathsf{s}(x) \wedge (\exists y.y \sim x \wedge a(y)) \wedge (\forall y.y \not\sim x \Rightarrow (\exists z.z \sim y \wedge b(z)))]$$

which states that there is a system process with an a and that every other process has a b. Note that φ_1 is satisfied by the $\mathbb{P}$-execution $w = (a,1)(a,2)(a,3)$ if $\mathbb{P}_{\mathsf{s}} = \{1,2,3\}$, but not by the same execution if $\mathbb{P}_{\mathsf{s}} = \{1,2,3,4\}$.

4.1.2 Winning triples, Synthesis, Cutoffs

A $\mathbb{P}$-*strategy* for System, following Definition 7, is a mapping $f : \Sigma^* \to (\Sigma_{\mathsf{s}} \cup \{\varepsilon\})$. The definition of f-compatible and f-fair $\mathbb{P}$-executions follow similarly: a $\mathbb{P}$-execution $w = (a_0, p_0)\ldots$ is f-compatible if:

1. for all $i < |w|$ such that $a_i \in A_{\mathsf{s}}$, we have that $(a_i, p_i) = f((a_0,p_0)\ldots(a_{i-1},p_{i-1}))$.
2. if w is finite, then $f(w) = \varepsilon$.

and w is f-fair if at least one of the following three conditions is satisfied:

1. w is finite,
2. there are finitely many $i \in \mathbb{N}$ such that $f((a_0,p_0)\ldots(a_i,p_i)) \neq \varepsilon$, or
3. there are infinitely many $i \in \mathbb{N}$ such that $a_i \in \Sigma_{\mathsf{s}}$.

Given a first-order formula φ and a $\mathbb{P}$-strategy f, we say that f is $\mathbb{P}$-*winning* for φ is all $\mathbb{P}$-executions that are f-compatible and f-fair satisfy φ.

Winning triples.

The existence of a $\mathbb{P}$-winning strategy for a given formula does not depend on the concrete identities of processes, but only on the cardinality of the sets $\mathbb{P}_{\mathsf{s}}$, $\mathbb{P}_{\mathsf{e}}$, $\mathbb{P}_{\mathsf{se}}$. Indeed, first-order formulas can only tell whether two data values are equal, and have no other way to handle process identities. For instance, in the previous Example 18, one could substitute 1, 2, and 3 by any three different values in execution w and the satisfiability of φ_1 would be unchanged. This motivates the following definition of winning triples for a formula.

Definition 17. Given φ, let $\mathrm{Win}(\varphi)$ be the set of triples $(k_{\mathsf{s}}, k_{\mathsf{e}}, k_{\mathsf{se}}) \in \mathbb{N}^{\mathbb{T}}$ for which there is $\mathbb{P} = \mathbb{P}_{\mathsf{s}} \uplus \mathbb{P}_{\mathsf{e}} \uplus \mathbb{P}_{\mathsf{se}}$ such that $|\mathbb{P}_\theta| = k_\theta$ for all $\theta \in \mathbb{T}$ and there is a $\mathbb{P}$-strategy that is $\mathbb{P}$-winning for φ.

Let $\mathbb{0} = \{0\}$ and $k_{\mathsf{e}}, k_{\mathsf{se}} \in \mathbb{N}$. We focus on the intersection of $\mathrm{Win}(\varphi)$ with the sets:

- $\mathbb{N} \times \mathbb{0} \times \mathbb{0}$ (which corresponds to the usual satisfiability problem),
- $\mathbb{N} \times \{k_{\mathsf{e}}\} \times \{k_{\mathsf{se}}\}$ (there is a constant number of environment and mixed processes),
- $\mathbb{N} \times \mathbb{N} \times \{k_{\mathsf{se}}\}$ (there is a constant number of mixed processes),
- $\mathbb{0} \times \mathbb{0} \times \mathbb{N}$ (each process is controlled by both System and Environment).

Example 19. Let $A_{\mathsf{s}} = \{a\}$, $A_{\mathsf{e}} = \{b\}$, and $\varphi_3 = \forall x.\big(b(x) \Rightarrow \exists y.(x \sim y \wedge x < y \wedge a(y))\big)$. Formula φ_3 essentially states that each b is eventually followed by an a executed by the same process. Observe that if there is at least one environment process, then it is easy to see that there can be no winning strategy for the system, as Environment can perform a b action on an environment process on which System cannot perform any action. Hence necessarily we must have $\mathbb{P}_{\mathsf{e}} = \emptyset$. This condition is actually sufficient, i.e. $\mathrm{Win}(\varphi_3) = \mathbb{N} \times \mathbb{0} \times \mathbb{N}$, and we give two winning strategies to illustrate this.

Fix $\mathbb{P} = \mathbb{P}_{\mathsf{s}} \uplus \mathbb{P}_{\mathsf{se}}$.

- For the first strategy, let $\mathbb{P}_{\mathsf{se}} = \{p_0, \ldots, p_{n-1}\}$ be the set of mixed processes sorted in some arbitrary order. Then the strategy f_1 maps every word $w \in \Sigma^*$ to (a, p_i) where i is the number of occurrences of events in Σ_{s} in w. In simple terms, this strategy plays an a on every mixed process, and repeats over and over again. By the fairness assumption, this guarantees an infinite number of a on every mixed process, and therefore φ_3 is trivially satisfied.

- We can also define a more "economical" strategy. For any $\mathbb{P}$-execution w, let $min_{\mathbb{P}}(w)$ be the process $p \in \mathbb{P}_{\mathsf{se}}$ such that (b, p) is the earliest (w.r.t. to position order) event of w that is not eventually followed by an (a, p), if it exists. We then let

$$f_2(w) = \begin{cases} (a, min_{\mathbb{P}}(w)) & \text{if } min_{\mathbb{P}}(w) \text{ exists,} \\ \varepsilon & \text{otherwise.} \end{cases}$$

  This strategy f_2 is also winning for φ_3, as any "pending" (b, p) is eventually closed by f_2 thanks to the fairness condition.

Example 20. Let $A_{\mathsf{s}} = \{a\}$ and $A_{\mathsf{e}} = \{b\}$. We define the formula $\varphi_4 = (\neg\exists x.b(x)) \Leftrightarrow (\exists x.a(x))$, which states that there is an a action if and only if there is no b. Then unless $\mathbb{P}_{\mathsf{e}} = \mathbb{P}_{\mathsf{se}} = \emptyset$, there is no winning strategy for System, as System can either do nothing in which case Environment also does nothing and the execution will not satisfy φ_4, or System performs an a action on some process at which point Environment replies with a b action on some process, and then System has lost. The formula is however obviously satisfiable (as long as there is at least one process for System to perform an a action on), therefore $\mathrm{Win}(\varphi_4) = \mathbb{N}_{>} \times \mathbb{0} \times \mathbb{0}$.

Synthesis problem.

The parameterized synthesis problem is defined as follows.

Definition 18. For fixed $\mathfrak{F} \in \{\mathrm{FO}, \mathrm{FO}^2\}$, set of relation symbols $R \subseteq \{\sim, \mathrm{succ}, <\}$, and $\mathcal{N}_{\mathsf{s}}, \mathcal{N}_{\mathsf{e}}, \mathcal{N}_{\mathsf{se}} \subseteq \mathbb{N}$, the (parameterized) synthesis problem is given by

$\textsc{Synth}(\mathfrak{F}[R], \mathcal{N}_{\mathsf{s}}, \mathcal{N}_{\mathsf{e}}, \mathcal{N}_{\mathsf{se}})$	
Input:	$\Sigma = \Sigma_{\mathsf{s}} \uplus \Sigma_{\mathsf{e}}$, a formula $\varphi \in \mathfrak{F}[R]$ over Σ
Question:	$\mathrm{Win}(\varphi) \cap (\mathcal{N}_{\mathsf{s}} \times \mathcal{N}_{\mathsf{e}} \times \mathcal{N}_{\mathsf{se}}) \neq \emptyset$?

The *satisfiability problem* for $\mathfrak{F}[R]$ is defined as $\textsc{Synth}(\mathfrak{F}[R], \mathbb{N}, \mathbb{0}, \mathbb{0})$.

Cutoff.

When dealing with parameterized systems in general, searching for a cutoff is a popular method to try reducing the search space. In general, a cutoff is a number such that the behavior of a system is fully known for parameters that are greater than this number. They were introduced in [EN95] to deal with model checking on ring architectures. For our parameterized synthesis problem in particular, a cutoff means that if there are "enough" processes, then either System or Environment always win. See [JB14] for an approach of parameterized synthesis using cutoffs.

Cutoffs are formally defined as follows.

Definition 19. A *cutoff* for a formula φ with respect to $(\mathcal{N}_{\mathsf{s}}, \mathcal{N}_{\mathsf{e}}, \mathcal{N}_{\mathsf{se}})$ is a triple $(k_{\mathsf{s}}, k_{\mathsf{e}}, k_{\mathsf{se}}) \in \mathcal{N}_{\mathsf{s}} \times \mathcal{N}_{\mathsf{e}} \times \mathcal{N}_{\mathsf{se}}$ such that either:

- all $(k'_{\mathsf{s}}, k'_{\mathsf{e}}, k'_{\mathsf{se}}) \in \mathcal{N}_{\mathsf{s}} \times \mathcal{N}_{\mathsf{e}} \times \mathcal{N}_{\mathsf{se}}$ such that $k'_\theta \geq k_\theta$ belong to $\mathrm{Win}(\varphi)$, or
- all $(k'_{\mathsf{s}}, k'_{\mathsf{e}}, k'_{\mathsf{se}}) \in \mathcal{N}_{\mathsf{s}} \times \mathcal{N}_{\mathsf{e}} \times \mathcal{N}_{\mathsf{se}}$ such that $k'_\theta \geq k_\theta$ *do not* belong to $\mathrm{Win}(\varphi)$.

Discovering such a cutoff is an important hint for deciding the synthesis problem. Indeed, if there is a cutoff, then there is a simple algorithm to decide the synthesis problem: first check the (potentially big, but) finite amount of triples below the cutoff for a winning strategy, and if there are none then it is useless to check above the cutoff due to its property.

Example 21. Let $A_{\mathsf{s}} = \{a\}, A_{\mathsf{e}} = \{b\}$, and $\varphi_5 = (\exists x.b(x)) \Leftrightarrow (\exists y, z.y \not\sim z \wedge a(y) \wedge a(z))$, which states that there is (at least) a b action performed by Environment if and only if there are (at least) two a actions by System. Let $(\mathcal{N}_{\mathsf{s}}, \mathcal{N}_{\mathsf{e}}, \mathcal{N}_{\mathsf{se}}) = (\mathbb{0}, \mathbb{0}, \mathbb{N})$. It is easy to see that System wins with 0 mixed processes, loses if there is only 1, and then wins again as long as there are at least 2. Therefore, $(0, 0, 2)$ is a cutoff for φ_5 w.r.t. $(\mathbb{0}, \mathbb{0}, \mathbb{N})$.

Let us now investigate the complexity of the synthesis problem for some instances of the parameters.

4.2 $\mathrm{FO}^2[\sim, \mathrm{succ}, <]$

First, we focus on first-order logic restricted to two variables, with all predicates available. Since the satisfiability problem is decidable for a similar logic [BMS+06], one could hope that the synthesis problem also remains decidable. However, we show that this is not the case when considering only mixed processes.

Theorem 22. $\textsc{Synth}(\mathrm{FO}^2[\sim, \mathrm{succ}, <], \mathbb{0}, \mathbb{0}, \mathbb{N})$ *is undecidable.*

Proof. To prove this, we adapt the proof from [FP18b, FP18a], which show undecidability of finding a winning strategy in Logic of Repeating Values games, to our setting. The proof is a reduction from the halting problem of two-counter machines (2CM), which we already used in Section 3.4.2 for the proof of Theorem 20. Actually, we only consider *deterministic* 2CM: we call a 2CM $M = (Q, T, q_0, q_h)$ *deterministic* if, from a given configuration (q, ν_1, ν_2), at most one transition t is firable. It is known that the halting problem is undecidable even for deterministic 2CM [Min67].

Given a deterministic 2CM M, we write a specification formula φ such that a satisfying execution encodes a correct run of M that ends in a halting configuration.

Environment has to choose the sequence of transitions of M, while System's job is to ensure that Environment does not make an illegal transition. The valuation of counter c_1 at any point in the run is encoded by the number of different processes on which only System has executed actions (and not Environment). Conversely, the valuation of c_2 is the number of different processes on which only Environment has executed actions (and not System). Both players will cooperate to ensure the valuation is correct with respect to the sequence of transitions taken:

- If a transition increments c_1, then first Environment plays on a process on which both System and herself have already executed an action, then System executes an action on a fresh process so that there is one more process unique to System while the value of c_2 stays unchanged.
- If a transition decrements c_1, then Environment executes an action on a process that was unique to System, and System replies on the same process, making one process not unique for System anymore thus decrementing c_1 (while c_2 is unchanged).
- If a transition tests that c_1 is zero, Environment executes an action on a process already shared by both System and Environment, and System either plays on the same process if the transition is legal with this valuation (that is c_1 is actually zero), otherwise she plays on a process that was unique to herself so far (proving that c_1 was not zero) and instantly wins the game.

Transitions involving the other counter are encoded in a similar fashion. The formula ensures that Environment starts the execution, and that System and Environment alternate their actions until a halting configuration is reached. The actions of Environment are the different transitions taken along the simulated run of M. The objective of System is that a halting configuration is reached, while the objective of Environment is that the run ends before reaching such a configuration (because there are no more fresh processes anymore), or that the run continues forever without reaching a halting configuration. One can show that there exists a halting run of M if and only if there is some $k \in \mathbb{N}$ such that for all $\mathbb{P} = \mathbb{P}_{\mathsf{se}}$ of size at least k there exists a winning $\mathbb{P}$-strategy for System for φ.

The alphabet is partitioned in $A_{\mathsf{e}} = \{\mathsf{st}\} \cup T$ and $A_{\mathsf{s}} = \{a\}$. Let us fix x and y the two variables used. The formula will check that at every position of the run, two consecutive transitions played by Environment are compatible with respect to their starting and ending states. Moreover, one has to make sure that the first transition played by Environment can be taken from the initial state. The first two positions will be dummy actions, to ensure that Environment and System share at least one process, while the simulation of the actual 2CM execution will start from the third position.

We use shorthands $\mathrm{First}(x)$, $\mathrm{Second}(x)$, and $\mathrm{Third}(x)$ for the $\mathrm{FO}^2[\mathrm{succ}]$-definable formulas that say that x is the first, second, and third position in the $\mathbb{P}$-execution respectively:

$$\begin{aligned}
\mathrm{First}(x) &\equiv \left(\bigvee_{b \in A} b(x)\right) \wedge \neg \exists y.\mathrm{succ}(y, x) \\
\mathrm{Second}(x) &\equiv \exists y.\, (\mathrm{First}(y) \wedge \mathrm{succ}(y, x)) \\
\mathrm{Third}(x) &\equiv \exists y.\, (\mathrm{Second}(y) \wedge \mathrm{succ}(y, x))
\end{aligned}$$

where $\mathrm{First}(y)$ is the formula $\mathrm{First}(x)$ with x and y swapped, and so on. We also define the following useful formula:

$$\mathrm{Old}_\theta(x) \equiv \exists y.(y < x \wedge y \sim x \wedge \bigvee_{b \in A_\theta} b(x))$$

for $\theta \in \{\mathsf{s}, \mathsf{e}\}$ that says that the value at position x was already seen at an anterior position of player θ. Finally, let

$$\mathrm{Fresh} \equiv \exists x. \forall y. (y \neq x \implies y \not\sim x)$$

be a formula that states that there is at least one fresh process, i.e. a process on which neither player performed an action during the execution.

The specification will force Environment to play first, and then System and Environment to play in turn until eventually a halting configuration is reached. We give first the set of constraints Φ^{e} that Environment must satisfy, then the set of constraints related to System.

The following formulas make up the set Φ^{e}:

- Environment does not play twice in a row:

$$\forall x. \forall y. \left[(\bigvee_{b \in A_{\mathsf{e}}} b(x) \wedge \mathrm{succ}(x, y)) \implies \neg \bigvee_{b \in A_{\mathsf{e}}} b(y) \right]$$

- Environment always executes an action when it is its turn, unless the halting configuration is reached or there are no more fresh processes. We let T_h be the set of transitions in M whose ending state is q_h:

$$\forall x. \left[\left(\bigvee_{b \in A_{\mathsf{e}} \setminus T_h} b(x) \wedge \mathrm{Fresh} \right) \implies \exists y.(x < y \wedge \bigvee_{b \in A_{\mathsf{e}}} b(y)) \right]$$

- Environment starts with an st:

$$\exists x.(\mathrm{First}(x) \wedge \mathsf{st}(x))$$

- There is an st only in the first position:

$$\forall x.(\mathsf{st}(x) \implies \mathrm{First}(x))$$

- The first transition is an initial transition (i.e. starts from an initial state):

$$\forall x.(\mathrm{Third}(x) \Rightarrow \bigvee_{t \text{ is initial}} t(x))$$

- Consecutive transitions are compatible (i.e. the ending state of the n-th one is the starting state of the $n+1$-th):

$$\forall x. \bigwedge_t \left(t(x) \Rightarrow \forall y.(\mathrm{succ}(x, y) \Rightarrow \forall x.(\mathrm{succ}(y, x) \Rightarrow \bigvee_{t' \text{ compatible with } t} t'(x))) \right)$$

- If t increments c_1, Environment plays on a process already shared by System and Environment:

$$\forall x. \bigwedge_{t \text{ increments } c_1} (t(x) \Rightarrow \mathrm{Old}_{\mathsf{e}}(x) \wedge \mathrm{Old}_{\mathsf{s}}(x)))$$

- If t increments c_2, Environment must play on a fresh process:

$$\forall x. \bigwedge_{t \text{ increments } c_2} (t(x) \Rightarrow \neg\mathrm{Old}_{\mathsf{e}}(x) \wedge \neg\mathrm{Old}_{\mathsf{s}}(x))$$

- If t decrements c_1, Environment plays on a process that was unique to System:

$$\forall x. \bigwedge_{t \text{ decrements } c_1} (t(x) \Rightarrow \neg\mathrm{Old}_{\mathsf{e}}(x) \wedge \mathrm{Old}_{\mathsf{s}}(x))$$

- If t decrements c_2, Environment must play on a process that was unique to herself:

$$\forall x. \bigwedge_{t \text{ decrements } c_2} (t(x) \Rightarrow \mathrm{Old}_{\mathsf{e}}(x) \wedge \neg\mathrm{Old}_{\mathsf{s}}(x))$$

- If t checks that c_1 is zero, Environment plays a shared value and System does not reply with a value unique to herself:

$$\forall x. \bigwedge_{t \text{ zero-tests } c_1} (t(x) \Rightarrow \mathrm{Old}_{\mathsf{s}}(x) \wedge \mathrm{Old}_{\mathsf{e}}(x) \wedge \forall y.(\mathrm{succ}(x,y) \Rightarrow \neg\mathrm{Old}_{\mathsf{s}}(y) \vee \mathrm{Old}_{\mathsf{e}}(y)))$$

- If t checks that c_2 is zero, Environment plays a shared value and System does not reply with a value unique to Environment:

$$\forall x. \bigwedge_{t \text{ zero-tests } c_2} (t(x) \Rightarrow \mathrm{Old}_{\mathsf{s}}(x) \wedge \mathrm{Old}_{\mathsf{e}}(x) \wedge \forall y.(\mathrm{succ}(x,y) \Rightarrow \mathrm{Old}_{\mathsf{s}}(y) \vee \neg\mathrm{Old}_{\mathsf{e}}(y)))$$

And now we construct the set Φ^{s} of system constraints:

- System does not play twice in a row:

$$\forall x. \forall y. (a(x) \wedge \mathrm{succ}(x,y)) \implies \neg a(y)$$

- The first move must be played on the same process as Environment:

$$\forall x. [\mathrm{Second}(x) \Rightarrow \exists y. (\mathrm{succ}(y,x) \wedge x \sim y)]$$

- If t increments c_1, System must reply on a fresh process:

$$\forall x. \bigwedge_{t \text{ increments } c_1} (t(x) \Rightarrow \exists y. (\mathrm{succ}(x,y) \wedge \neg\mathrm{Old}_{\mathsf{s}}(y) \wedge \neg\mathrm{Old}_{\mathsf{e}}(y)))$$

- If t increments c_2, System replies on an already shared process:

$$\forall x. \bigwedge_{t \text{ increments } c_2} (t(x) \Rightarrow \exists y. (\mathrm{succ}(x,y) \wedge \neg(x \sim y) \wedge \mathrm{Old}_{\mathsf{s}}(y) \wedge \mathrm{Old}_{\mathsf{e}}(y)))$$

- If t decrements either counter, System replies on the same process:

$$\forall x. \bigwedge_{t \text{ decrements } c_1 \text{ or } c_2} (t(x) \Rightarrow \exists y.(\mathrm{succ}(x,y) \wedge y \sim x))$$

- If t zero-tests either counter, System replies on the same process

$$\forall x. \bigwedge_{t \text{ zero-tests } \mathsf{c}_1 \text{ or } \mathsf{c}_2} (t(x) \Rightarrow \exists y.(\mathrm{succ}(x,y) \wedge (y \sim x)))$$

Put together, this gives the formula

$$\varphi = \Phi^{\mathsf{e}} \implies (\Phi^{\mathsf{s}} \wedge \exists x. \bigvee_{t \in T_h} t(x))$$

Let us fix some finite set $\mathbb{P}$ of mixed processes only. For a given $\mathbb{P}$-execution w, for $\theta \in \{\mathsf{s}, \mathsf{e}\}$, let $\mathbb{P}^\theta(w) = \{p \in \mathbb{P} \mid (b,p) \in w \text{ for some } b \in A_\theta\}$ be the set of processes on which player θ performed an action in w. Then we let $\mathbb{P}^1(w) = \mathbb{P}^{\mathsf{s}}(w) \setminus \mathbb{P}^{\mathsf{e}}(w)$ be the set of processes unique to System, which is used to encode the value of counter 1, and similarly we let $\mathbb{P}^2(w) = \mathbb{P}^{\mathsf{e}}(w) \setminus \mathbb{P}^{\mathsf{s}}(w)$ be the set of processes unique to Environment.

Consider the $\mathbb{P}$-strategy f for System such that $f(\mathsf{st}, p) = (a, p)$ and for each $\mathbb{P}$-execution w ending in some (t, p), $f(w) = (a, p')$ with p' a process such that:

- if t increments c_1, then $p' \in \mathbb{P} \setminus (\mathbb{P}^{\mathsf{s}}(w) \cup \mathbb{P}^{\mathsf{e}}(w))$ is a fresh process,
- if t increments c_2, then $w[1] = (a, p')$,
- if t decrements either counter, then $p' = p$,
- if t zero-test counter $i \in \{1, 2\}$, then $p' \in \mathbb{P}^i(w)$ if $\mathbb{P}^i(w) \neq \emptyset$, otherwise $p' = p$.

In all other cases, or if t is a transition incrementing c_1 but there is no remaining fresh process, then f returns ε. Simply put, f follows the constraints given in Φ^{s} unless Environment tries to cheat and play a zero-test transition when a counter is not zero, in which case System plays in a way to falsify Φ^{e}.

Consider an execution w that is f-compatible and that satisfies Φ^{e}. Let $(\mathsf{st}, t_1, t_2, \dots)$ the sequence of actions performed by Environment. By induction, let us show that for all $n \geq 0$, $\gamma_0 \vdash_{t_1} \cdots \vdash_{t_n} \gamma_n$, where γ_i is the t_i-successor of γ_{i-1}, is a run of M and that with w_n the prefix of w of size $2n+2$, we have that $\mathbb{P}^i(w_n)$ is a set whose size is the valuation of counter i in γ_n. The case of $n = 0$ is easy, as $w_0 = (\mathsf{st}, p)(a, p)$ for some process p and so $\mathbb{P}^1(w_0) = \mathbb{P}^2(w_0) = \emptyset$. Now suppose the affirmation is true for some $n \geq 0$, and let $w_{n+1} = w_n \cdot (t_{n+1}, p)(a, p')$. Since Φ^{e} is satisfied, we know that t_{n+1} is compatible with t_n. Let us check the different cases for t_{n+1}:

- If t_{n+1} increments c_1, then $p \in \mathbb{P}^{\mathsf{s}}(w_n) \cup \mathbb{P}^{\mathsf{e}}(w_n)$ because Φ^{e} is satisfied and $p' \in \mathbb{P} \setminus (\mathbb{P}^{\mathsf{s}}(w_n) \cup \mathbb{P}^{\mathsf{e}}(w_n))$ by compatibility with f. Therefore $\mathbb{P}^1(w_{n+1}) = \mathbb{P}^1(w_n) \uplus \{p'\}$ had its size incremented by one and $\mathbb{P}^2(w_{n+1}) = \mathbb{P}^2(w_n)$ is unchanged. Moreover, as t_{n+1} is compatible with t_n and is an incrementing transition, then it is firable from γ_n, therefore $\gamma_0 \vdash_{t_1} \cdots \vdash_{t_{n+1}} \gamma_{n+1}$ is a run of M.

- If t_{n+1} increments c_2, then $p \in \mathbb{P} \setminus (\mathbb{P}^{\mathsf{s}}(w_n) \cup \mathbb{P}^{\mathsf{e}}(w_n))$ by Φ^{e} and $p' \in \mathbb{P}^{\mathsf{s}}(w_n) \cup \mathbb{P}^{\mathsf{e}}(w_n)$ by f-compatibility, therefore $\mathbb{P}^1(w_{n+1}) = \mathbb{P}^1(w_n)$ and $\mathbb{P}^2(w_{n+1}) = \mathbb{P}^2(w_n) \uplus \{p\}$. Same as above, $\gamma_0 \vdash_{t_1} \cdots \vdash_{t_{n+1}} \gamma_{n+1}$ is a run of M.

- If t_{n+1} decrements c_1, then similarly since Φ^{e} is satisfied we have that $\mathbb{P}^1(w_{n+1}) \uplus \{p\} = \mathbb{P}^1(w_n)$ and since w is f-compatible we have that $p' = p$ and therefore $\mathbb{P}^2(w_{n+1}) = \mathbb{P}^2(w_n)$. Furthermore, since $\mathbb{P}^1(w_n)$ is of size at least one as it contains at least p, then the valuation of c_1 in γ_n must be at least one by induction hypothesis. Thus t_{n+1} is firable from γ_n and $\gamma_0 \vdash_{t_1} \cdots \vdash_{t_{n+1}} \gamma_{n+1}$ is a run of M.

- If t_{n+1} decrements c_2 then we have that $\mathbb{P}^1(w_{n+1}) = \mathbb{P}^1(w_n)$ and $\mathbb{P}^2(w_{n+1}) \uplus \{p'\} = \mathbb{P}^2(w_n)$ and that t_{n+1} is firable from γ_n.

- If t_{n+1} zero-tests c_1, then as Φ^{e} is satisfied we have that $p' \notin \mathbb{P}^1(w_n)$. Because w is also f-compatible, we deduce that $\mathbb{P}^1(w_n) = \emptyset$. Therefore the valuation of c_1 was 0 in γ_n, so t_{n+1} is firable from γ_n. Also for all $i \in \{1, 2\}$, $\mathbb{P}^i(w_{n+1}) = \mathbb{P}^i(w_n)$ are unchanged.

- The same reasoning in the case where t_{n+1} zero-tests c_2 shows that the valuation of c_2 is indeed 0 in γ_n so the properties are satisfied.

Now suppose that there is a halting run of M $\gamma_0 \vdash_{t_1} \cdots \vdash_{t_n} \gamma_n$. Let $\mathbb{P}$ be a finite set of mixed processes of size $> 2n + 2$, f be the $\mathbb{P}$-strategy defined as above, and w be a $\mathbb{P}$-execution that is f-compatible and f-fair. If $w \not\models \Phi^{\mathsf{e}}$ then w is winning, therefore let us suppose that w satisfies Φ^{e}. Let $(\mathsf{st}, t'_1, \ldots, t'_m)$ be the sequence of actions performed by Environment. Suppose there is some i such that $t_i \neq t'_i$. As M is deterministic, this means that no transition outside of t_i can be fired from γ_i. Therefore either t'_i is not compatible with t'_{i-1}, t'_i decrements a counter that is 0, or zero-tests a counter that is not 0. In the first case Φ^{e} is immediately falsified, in the second case there is no unique process for the corresponding counter so Φ^{e} is also falsified whatever the process used by Environment, and in the third case f is defined in such a way that its next move falsifies Φ^{e} if the transition was not firable. Moreover, m cannot be larger than n because Environment cannot continue playing after playing t_n that leads to a halting state without falsifying Φ^{e}, and also m cannot be lower than n because the only way for an execution to stop mid-run and still satisfy Φ^{e} is by lack of fresh processes, but since there are more different processes than the size of the execution this cannot be the case. Finally, f is defined in a way that if Φ^{e} is satisfied, then Φ^{s} is also satisfied as long as there is always at least one fresh process. Therefore, w satisfies φ and so f is winning.

Conversely, suppose that there is some $\mathbb{P}$ such that f is winning. Let w be the $\mathbb{P}$-execution that satisfies Φ^{e} resulting from Environment following its constraints and always choosing a firable transition in M (which it can always do). This execution is necessarily finite, because to be winning there must be a $t \in T_h$ in the execution, and Environment stops performing actions after such a transition. Then with $(\mathsf{st}, t_1, \ldots, t_n)$ being the actions performed by Environment in w, we know that $\gamma_0 \vdash_{t_1} \cdots \vdash_{t_n} \gamma_n$ where γ_i is the t_i-successor of γ_{i-1} is a run of M, and that t_n is a transition leading to the halting state. Thus M is halting. □

4.3 FO[∼]

Since restricting first-order logic to two variables does not lead to decidability, we now investigate fragments of FO where the use of some predicates is restricted. Removing the data equality predicate $\sim$ would mean that data values in an execution cannot be accessed at all, since $\sim$ is the only way to specify anything about processes. Since we are interested in data words and not "ordinary" words, at least $\sim$ needs to be kept available. Therefore, and for the rest of this chapter, we focus on first-order logic with data equality ($\sim$) but no immediate successor (succ) or position order ($<$) predicates.

Without succ and $<$, formulas cannot specify anything about the relative order between positions. For instance, one can have formulas such as

$$\varphi_6 = \forall x.\, [req(x) \implies \exists y.(y \sim x \wedge ack(y))]$$

with $A_{\mathsf{s}} = \{ack\}$ and $A_{\mathsf{e}} = \{req\}$ that indicates that every *req* has a matching *ack* on the same process, but it is not possible to ask that this *ack* comes *after* the *req*. In that example, a possible strategy for System to satisfy the formula would be to play a single *ack* on every process and then do nothing, which works regardless of what Environment does. This is not to say that FO[∼] formulas never require System to actually *react* to what Environment does. For example, with the following formula:

$$\varphi_7 = \forall x.\, \left[(\exists^{=1} y.y \sim x \wedge req(y)) \Leftrightarrow (\exists^{=1} y.y \sim x \wedge ack(y))\right]$$

which states that there is exactly one *req* on a process if and only if there is exactly one *ack*, a winning strategy for System must adapt to the actions of Environment. A winning strategy for this formula would be to wait until Environment does a *req* on a process and then play an *ack* on that process, and if Environment does another *req* on that same process later then System replies with another *ack*.

What those two examples hint at is that FO[∼] formulas can specify for each process and for each action the number of times that the process must perform that action, up to some bound that depends on the formula. In the case of φ_7, this means that every process must either perform 1 *req* and 1 *ack*, or 0, 2 or more *req* and 0, 2 or more *ack*. An FO[∼] formula can also express that there is at least/exactly some number of such processes, again up to some bound. Conversely, it does not seem possible to express something more than just counting since there is no order on the positions.

Let us give a normal form for FO[∼] formulas that formalizes those intuitions.

4.3.1 Normal form

Fix $A = A_{\mathsf{s}} \uplus A_{\mathsf{e}}$ an alphabet of actions. Given a data word $w = (a_1, p_1) \ldots$, a *class* is the subword of w containing all positions with the same data value. First we define a formula that counts the number of occurrences of each letter in a class up to some bound.

More precisely, for a bound $B \in \mathbb{N}$, this number of occurences is described by a mapping ℓ from A to $\{0, \ldots, B\}$. This mapping is referred to as a *local state* (or location). We set $L = \{0, \ldots, B\}^A$ the set of all locations.

We define the formula $\psi_{B,\ell}(y)$ that states that the class containing y has local state ℓ as follows:

$$\psi_{B,\ell}(y) \;=\; \bigwedge_{\substack{a\in A\\ \ell(a)<B}} \exists^{=\ell(a)} z.\big(y \sim z \wedge a(z)\big) \wedge \bigwedge_{\substack{a\in A\\ \ell(a)=B}} \exists^{\geq \ell(a)} z.\big(y \sim z \wedge a(z)\big)$$

Remember that, as defined in Section 2.3.2, $\exists^{\geq k} x.\psi(x)$ (resp. $\exists^{=k} x.\psi(x)$) means that at least (resp. exactly) k distinct positions x satisfy formula ψ. In plain English, formula $\psi_{B,\ell}(y)$ checks for all $a \in A$ that there are exactly or at least $\ell(a)$ distinct positions in the class of y that have action a depending on whether $\ell(a) < B$ or $\ell(a) = B$ respectively. We can then state our normal form for FO[∼] formulas.

Theorem 23. *For every* FO[∼] *formula* φ*, there is a bound* $B \in \mathbb{N}$ *such that* φ *is equivalent to a disjunction of conjunctions of formulas of the form*

$$\exists^{\bowtie m} y.(\theta(y) \wedge \psi_{B,\ell}(y))$$

where $\bowtie \in \{=, \geq\}$*,* $m \in \mathbb{N}$*,* $\theta \in \mathbb{T}$*, and* $\ell \in L$*.*

Before giving the proof, let us first give an example to illustrate this normal form.

Example 22. Recall the previous formula:

$$\varphi_7 = \big[\forall x.(\exists^{=1} y.y \sim x \wedge req(y)) \Leftrightarrow (\exists^{=1} y.y \sim x \wedge ack(y))\big]$$

which states that for every process, there is exactly one *req* iff there is exactly one *ack*. Let $B = 2$, and $L_7 = \{\ell \in L \mid (\ell(req) = 1 \neq \ell(ack)) \vee (\ell(req) \neq 1 = \ell(ack))\}$. Intuitively, L_7 represents the set of local states that every process must avoid to satisfy φ_7. Then an equivalent formula in normal form is the formula:

$$\varphi_7' = \bigwedge_{\theta\in\mathbb{T},\ell\in L_7} \exists^{=0} y.(\theta(y) \wedge \psi_{2,\ell}(y))$$

which states that no class should have a state in L_7.

Let us now prove Theorem 23.

Proof. First let us give an intuition on this proof. We use two known normal-form constructions for general FO logic.

Due to Schwentick and Barthelmann [SB98], any FO[∼] formula is effectively equivalent to a formula of the form

$$\exists x_1 \ldots \exists x_n \forall y.\varphi(x_1, \ldots, x_n, y)$$

where, in $\varphi(x_1, \ldots, x_n, y)$, quantification is always of the form $\exists z.(z \sim y \wedge \ldots)$ or $\forall z.(z \sim y \implies \ldots)$. In other words, all variables quantified in φ must belong to the class of y. Then by guessing the exact relation between the variables $x_1, \ldots, x_n$, one can eliminate these ending up with formulas that only talk about the class of a given event y. Those formulas are then evaluated over multi-sets over the alphabet $\mathbb{T} \cup A$. According to Hanf's theorem [Han65, BK12], they are effectively equivalent

to statements counting elements up to some threshold. This finally leads to the desired normal form.

Let Φ be an FO[$\sim$] formula. Using the Schwentick-Barthelmann normal form [SB98], we know that Φ is equivalent to a formula of the form

$$\Phi_1 = \exists x_1 \dots \exists x_n \forall y. \varphi(x_1, \dots, x_n, y)$$

where, in $\varphi(x_1, \dots, x_n, y)$, quantification is always of the form $\exists z.(z \sim y \wedge \dots)$ or $\forall z.(z \sim y \implies \dots)$. Since φ essentially talks about the class of y, we call it a *class formula* (wrt. y). Let $\mathbb{X} = \{x_1, \dots, x_n\}$. Without loss of generality, we assume that none of the variables in $\mathbb{X} \cup \{y\}$ is quantified in φ.

Class Abstraction. Note that, due to the variables in $\mathbb{X}$, the formula φ may reason about elements that are *outside* the class of y. Our aim is to get rid of these variables so as to end up with formulas that talk about classes only. As the variables in $\mathbb{X}$ are quantified existentially, we can basically guess the relation between them. This is done in terms of a *class abstraction*, which is given by a triple $\mathcal{C} = (P, \approx, \lambda)$ where $P \subseteq 2^{\mathbb{X}}$ is a partition of $\mathbb{X}$ (if $n = 0$, then $P = \emptyset$), $\lambda : \mathbb{X} \to A \uplus \mathbb{T}$ (recall that $\mathbb{T} = \{\mathsf{s}, \mathsf{e}, \mathsf{se}\}$), and $\approx$ is an equivalence relation over $\mathbb{X}$ such that, for all $x_i, x_j \in \mathbb{X}$

- if $x_i \approx x_j$, then $x_i \sim_{\mathcal{C}} x_j$ and $\lambda(x_i) = \lambda(x_j)$, and
- if $x_i \sim_{\mathcal{C}} x_j$ and $x_i \not\approx x_j$, then $\{\lambda(x_i), \lambda(x_j)\} \cap A \neq \emptyset$,

where we write $x_i \sim_{\mathcal{C}} x_j$ if $\{x_i, x_j\} \subseteq X$ for some $X \in P$. Let $\mathfrak{C}_{\mathbb{X}}$ be the set of all class abstractions.

Example 23. Figure 4.1 depicts a class abstraction $\mathcal{C} = (P, \approx, \lambda)$ for $A = \{a, b, c, d\}$. The red areas represent the partition P, the blue ones represent the equivalence classes of $\approx$, which refine P. Moreover, we have $\lambda(x_4) = \lambda(x_7) = \mathsf{s}$, $\lambda(x_3) = \lambda(x_5) = b$, $\lambda(x_9) = c$, etc. The meaning of $\mathcal{C}$ is that x_1, x_4, x_7, x_8 are equivalent wrt. $\sim$, i.e., they belong to the same process. In particular, formulas such as $x_1 \sim x_4$ and $x_3 \sim x_{10}$ are true under this assumption. Moreover, $x_4 \approx x_7$ means that x_4 and x_7 denote *identical* elements. That is, the formula $x_4 = x_7$ would be true, whereas $x_2 = x_3$ does not hold.

Given $\mathcal{C} = (P, \approx, \lambda)$ and $X \subseteq \mathbb{X}$, we can define the formula $\mathsf{sat}(X, \mathcal{C})$, which checks whether the class abstraction $\mathcal{C}$ is consistent with a given execution as far as variables from X are concerned:

$$\mathsf{sat}(X, \mathcal{C}) \quad = \quad \left(\begin{array}{ll} & \bigwedge_{x_i \in X} (\lambda(x_i))(x_i) \\ \wedge & \bigwedge_{(x_i, x_j) \in \approx \cap X^2} x_i = x_j \wedge \bigwedge_{(x_i, x_j) \in X^2 \setminus \approx} x_i \neq x_j \\ \wedge & \bigwedge_{(x_i, x_j) \in \sim_{\mathcal{C}} \cap X^2} x_i \sim x_j \wedge \bigwedge_{(x_i, x_j) \in X^2 \setminus \sim_{\mathcal{C}}} x_i \not\sim x_j \end{array} \right)$$

Moreover, by fixing $\mathcal{C} = (P, \approx, \lambda)$ and $X \in P \cup \{\emptyset\}$, we can transform φ into a class formula (with respect to y)

$$[\![\varphi]\!]_{\mathcal{C}, X}((x_i)_{x_i \in X}, y)$$

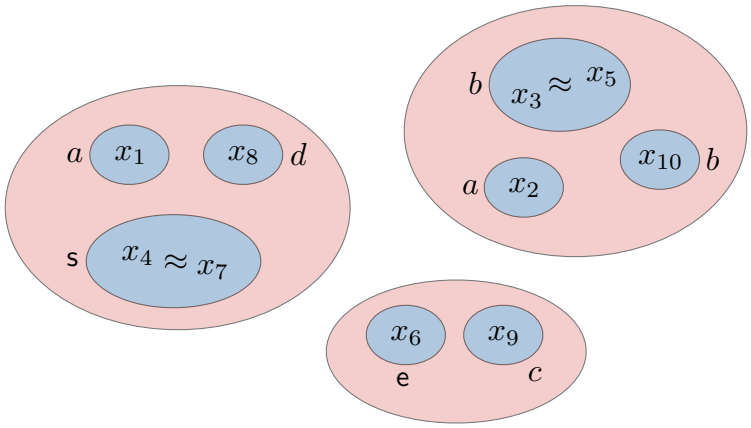

Figure 4.1: A class abstraction $\mathcal{C} = (P, \approx, \lambda)$

without variables from $\mathbb{X} \setminus X$ that "evaluates" φ based on the assumption that $\mathsf{sat}(\mathbb{X}, \mathcal{C}) \wedge (y \sim X)$ holds (in particular, $\mathsf{sat}(\mathbb{X}, \mathcal{C}) \wedge \neg(y \sim \mathbb{X})$ if $X = \emptyset$) where $y \sim X$ is a shorthand for $\bigvee_{x_i \in X} y \sim x_i$. We obtain it from φ inductively as follows (let $\theta \in A \cup \mathbb{T}$):

$$[\![\theta(z)]\!]_{\mathcal{C},X} = \begin{cases} \theta(z) & \text{if } z \notin \mathbb{X} \\ \top & \text{if } z \in \mathbb{X} \text{ and } \lambda(z) = \theta \\ \bot & \text{if } z \in \mathbb{X} \text{ and } \lambda(z) \neq \theta \end{cases}$$

$$[\![z = z']\!]_{\mathcal{C},X} = \begin{cases} z = z' & \text{if } z, z' \notin \mathbb{X} \setminus X \\ \top & \text{if } z, z' \in \mathbb{X} \setminus X \text{ and } z \approx z' \\ \bot & \text{otherwise} \end{cases}$$

$$[\![z \sim z']\!]_{\mathcal{C},X} = \begin{cases} z \sim z' & \text{if } z, z' \notin \mathbb{X} \setminus X \\ \top & \text{if } z, z' \in \mathbb{X} \setminus X \text{ and } z \sim_{\mathcal{C}} z' \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\psi \vee \psi']\!]_{\mathcal{C},X} = [\![\psi]\!]_{\mathcal{C},X} \vee [\![\psi']\!]_{\mathcal{C},X}$$
$$[\![\neg\psi]\!]_{\mathcal{C},X} = \neg[\![\psi]\!]_{\mathcal{C},X}$$
$$[\![\exists z.\psi]\!]_{\mathcal{C},X} = \exists z.[\![\psi]\!]_{\mathcal{C},X}$$

Transformation. Given the above definitions, we can now rephrase Φ_1 as follows. Along with $x_1, \ldots, x_n$, we also guess a class abstraction, which then allows us to reason about each class separately, without looking at the variables outside a class:

$$\Phi_2 = \bigvee_{\mathcal{C}=(P,\approx,\lambda)\in\mathfrak{C}_{\mathbb{X}}} \exists x_1 \ldots \exists x_n. \left(\begin{array}{lll} & \mathsf{sat}(\mathbb{X}, \mathcal{C}) & (\xi_1) \\ \wedge & \bigwedge_{X \in P} \forall y. \Big(y \sim X \implies [\![\varphi]\!]_{\mathcal{C},X}((x_i)_{x_i \in X}, y) \Big) & (\xi_2) \\ \wedge & \forall y. \Big(\neg(y \sim \mathbb{X}) \implies [\![\varphi]\!]_{\mathcal{C},\emptyset}(y) \Big) & (\xi_3) \end{array} \right)$$

In fact, we can push the quantifiers $\exists x_1 \ldots \exists x_n$ further inwards by replacing them

with $\exists(z_X)_{X\in P}$, which chooses one canonical representative z_X per class X:

$$\Phi_3 = \bigvee_{\mathcal{C}=(P,\approx,\lambda)\in\mathfrak{C}_{\mathbb{X}}} \exists(z_X)_{X\in P}.\left(\begin{array}{lll} & \bigwedge_{X\in P} proc(z_X) \wedge \bigwedge_{\substack{X,Y\in P\\ X\neq Y}} z_X \not\sim z_Y & (\xi_4)\\ \wedge & \bigwedge_{X\in P} \varphi_{\mathcal{C},X}(z_X) & (\xi_5)\\ \wedge & \forall z_\emptyset.\Big(\Big(proc(z_\emptyset) \wedge \bigwedge_{X\in P} \neg(z_\emptyset = z_X)\Big) \implies \varphi_{\mathcal{C},\emptyset}(z_\emptyset)\Big) & (\xi_6)\end{array}\right)$$

where $proc(z_X) = \bigvee_{\theta\in\mathbb{T}} \theta(z_X)$ and

$$\varphi_{\mathcal{C},X}(z_X) \;=\; \exists(x_i)_{x_i\in X}.\left(\begin{array}{ll} & z_X \sim X \;\wedge\; \mathsf{sat}(X,\mathcal{C})\\ \wedge & \forall y.\big(y\sim z_X \implies [\![\varphi]\!]_{\mathcal{C},X}((x_i)_{x_i\in X}, y)\big)\end{array}\right)$$

$$\varphi_{\mathcal{C},X}(z_\emptyset) \;=\; \forall y.\big(y\sim z_\emptyset \implies [\![\varphi]\!]_{\mathcal{C},\emptyset}(y)\big)$$

are *class formulas*[1] with respect to z_X.

Lemma 24. *Formulas Φ_2 and Φ_3 are logically equivalent.*

Proof of lemma. Fix some set $\mathbb{P}$ of processes. Suppose $w \models \Phi_2$, say, witnessed by class abstraction $\mathcal{C}$ and valuation $\nu_{\mathbb{X}} = \{x_1 \mapsto e_1, \ldots, x_n \mapsto e_n\}$. That is, $(w, \nu_{\mathbb{X}}) \models \xi_1 \wedge \xi_2 \wedge \xi_3$. Let ν_X denote the restriction of $\nu_{\mathbb{X}}$ to $X \in P$. For $X \in P$, consider the unique process $e_X \in \mathbb{P}$ such that $e_X \sim e_i$ for some $x_i \in X$. Consider the valuation $\nu_{repr} = \{z_X \mapsto e_X \mid X \in P\}$. Let us show $(w, \nu_{repr}) \models \xi_4 \wedge \xi_5 \wedge \xi_6$.

- (ξ_4) Clearly, we have $(w, \nu_{repr}) \models \xi_4$.
- (ξ_5) Let $X \in P$. We have $(w, \nu_X) \models z_X \sim X \wedge \mathsf{sat}(X, \mathcal{C})$. Take any $e \in \mathbb{P} \cup Pos(w)$ such that $e \sim e_X$. By satisfaction of ξ_2, we have $(w, \nu_X \cup \{y \mapsto e\}) \models [\![\varphi]\!]_{\mathcal{C},X}((x_i)_{x_i\in X}, y)$. Therefore, $(w, \nu_{repr}) \models \xi_5$.
- (ξ_6) Let $e_\emptyset \in \mathbb{P}$ such that $e_\emptyset \neq e_X$ for all $X \in P$. Moreover, let $e \in \mathbb{P} \cup Pos(w)$ such that $e \sim e_\emptyset$. Then, $e \not\sim e_X$ for all $X \in P$ so that, by satisfaction of ξ_3, we have $(w, \{y \mapsto e\}) \models [\![\varphi]\!]_{\mathcal{C},\emptyset}(y)$. We obtain $(w, \nu_{repr}) \models \xi_6$.

We conclude that $w \models \Phi_3$.

Conversely, suppose that $w \models \Phi_3$, witnessed by $\mathcal{C}$ and a valuation $\nu_{repr} = \{z_X \mapsto e_X\}_{X\in P}$. That is, $(w, \nu_{repr}) \models \xi_4 \wedge \xi_5 \wedge \xi_6$. For every $X \in P$, as $(w, \{z_X \mapsto e_X\}) \models \varphi_{\mathcal{C},X}(z_X)$ due to ξ_5, there is ν_X such that

$$(w, \{z_X \mapsto e_X\} \cup \nu_X) \models \left(\begin{array}{ll} & z_X \sim X \;\wedge\; \mathsf{sat}(X,\mathcal{C})\\ \wedge & \forall y.\big(y\sim z_X \implies [\![\varphi]\!]_{\mathcal{C},X}((x_i)_{x_i\in X}, y)\big)\end{array}\right)$$

Then let $\nu_{\mathbb{X}} = \{x_1 \mapsto e_1, \ldots, x_n \mapsto e_n\} = \bigcup_{X\in P}(\nu_X)$. Let us show that $(w, \nu_{\mathbb{X}}) \models \xi_1 \wedge \xi_2 \wedge \xi_3$.

- (ξ_1) As $e_i \not\sim e_j$ whenever $e_i \in X$ and $e_i \in Y$ for distinct sets $X, Y \in P$ due to ξ_4, we get $(w, \nu_{\mathbb{X}}) \models \mathsf{sat}(\mathbb{X}, \mathcal{C})$.

[1] in fact, they can be easily rewritten as a class formula

(ξ_2) Take any $X \in P$ and $e \in \mathbb{P} \cup Pos(w)$ such that $e \sim e_i$ for some $i \in \{1, \dots, n\}$ such that $x_i \in X$. As $e \sim e_X$, we obtain $(w, \nu_X \cup \{y \mapsto e\}) \models [\![\varphi]\!]_{\mathcal{C},X}((x_i)_{x_i \in X}, y)$.

(ξ_3) Let $e \in \mathbb{P} \cup Pos(w)$ such that $e \not\sim e_i$ for all $i \in \{1, \dots, n\}$. Let $e_\emptyset \in \mathbb{P}$ such that $e_\emptyset \sim e$. That is, $e_\emptyset \not\sim e_X$ for all $X \in P$. We get $(w, \{z_\emptyset \mapsto e_\emptyset\}) \models \varphi_{\mathcal{C},\emptyset}(z_\emptyset)$ from ξ_6. Since $e \sim e_\emptyset$, we also have that $(w, \{y \mapsto e\}) \models \varphi_{\mathcal{C},\emptyset}(y)$. Therefore, $(w, \nu_{\mathbb{X}}) \models \xi_3$.

We conclude that $w \models \Phi_2$. □

The formulas $\varphi_{\mathcal{C},X}(z_X)$, including the case $X = \emptyset$, are interesting, because they only reason about the class determined by z_X. As, with respect to $\sim$, any two elements from a class are equivalent anyway, we can actually ignore $\sim$. A class can then be seen as a simple multiset, or as a logical structure of degree 0 (there is no binary relation that connects two elements from a class). By Hanf's theorem [Han65, BK12], we can find $B \in \mathbb{N}$ such that every formula $\varphi_{\mathcal{C},X}(z_X)$, including the case $X = \emptyset$, is equivalent to a formula

$$\varphi_{\mathcal{C},X}(z_X) \equiv \bigvee_{(\theta,\ell) \in V_{\mathcal{C},X}} \big(\theta(z_X) \wedge \psi_{B,\ell}(z_X)\big)$$

for some sets $V_{\mathcal{C},X} \subseteq \mathbb{T} \times \{0, \dots, B\}^A$. Note that, for $V_{\mathcal{C},X} = \emptyset$, we get $\bot$. Recall that we had defined:

$$\psi_{B,\ell}(y) = \bigwedge_{\substack{a \in A \\ \ell(a) < B}} \exists^{=\ell(a)} z.\big(y \sim z \wedge a(z)\big) \wedge \bigwedge_{\substack{a \in A \\ \ell(a) = B}} \exists^{\geq \ell(a)} z.\big(y \sim z \wedge a(z)\big)$$

Thus, Φ_3 is equivalent to the following formula (note that the conjunct $\bigwedge_{X \in P} proc(z_X)$ is not needed anymore, as its satisfaction is guaranteed by the second line; other changes with respect to Φ_3 are highlighted in red):

$$\Phi_4 = \bigvee_{\mathcal{C}=(P,\approx,\lambda) \in \mathfrak{C}_{\mathbb{X}}} \exists (z_X)_{X \in P}. \left(\begin{array}{l} \bigwedge_{\substack{X,Y \in P \\ X \neq Y}} z_X \not\sim z_Y \\ \wedge \bigwedge_{X \in P} \bigvee_{(\theta,\ell) \in V_{\mathcal{C},X}} \big(\theta(z_X) \wedge \psi_{B,\ell}(z_X)\big) \\ \wedge \forall z_\emptyset. \Big(\Big(proc(z_\emptyset) \wedge \bigwedge_{X \in P} \neg(z_\emptyset = z_X) \Big) \\ \qquad \Longrightarrow \bigvee_{(\theta,\ell) \in V_{\mathcal{C},\emptyset}} \big(\theta(z_\emptyset) \wedge \psi_{B,\ell}(z_\emptyset)\big) \Big) \end{array} \right)$$

Expanding the expression, we obtain that Φ_4 is equivalent to:

$$\Phi_5 = \bigvee_{\substack{\mathcal{C}=(P,\approx,\lambda)\in\mathfrak{C}_{\mathbb{X}} \\ ((\theta_X,\ell_X))_{X\in P} \\ \in \prod_{X\in P} V_{\mathcal{C},X}}} \exists (z_X)_{X\in P}. \left(\begin{array}{ll} & \bigwedge_{\substack{X,Y\in P \\ X\neq Y}} z_X \not\sim z_Y \\ \wedge & \bigwedge_{X\in P} \big(\theta_X(z_X) \wedge \psi_{B,\ell}(z_X)\big) \\ \wedge & \forall z_\emptyset. \Big(\Big(proc(z_\emptyset) \wedge \bigwedge_{X\in P} \neg(z_\emptyset = z_X) \Big) \\ & \qquad \Longrightarrow \bigvee_{(\theta,\ell)\in V_{\mathcal{C},\emptyset}} \big(\theta(z_\emptyset) \wedge \psi_{B,\ell}(z_\emptyset)\big) \Big) \end{array} \right)$$

Finally, Φ_5 is equivalent to a formula of the desired form:

$$\Phi_6 = \bigvee_{\substack{\mathcal{C}=(P,\approx,\lambda)\in\mathfrak{C}_{\mathbb{X}} \\ \mathbf{v}=((\theta_X,\ell_X))_{X\in P} \\ \in \prod_{X\in P} V_{\mathcal{C},X}}} \left(\begin{array}{ll} & \bigwedge_{(\theta,\ell)\in V_{\mathcal{C},\emptyset}} \exists^{\geq |\mathbf{v}|_{(\theta,\ell)}} y. \big(\theta(y) \wedge \psi_{B,\ell}(y)\big) \\ \wedge & \bigwedge_{(\theta,\ell) \in (\mathbb{T}\times\{0,\dots,B\}^A)\setminus V_{\mathcal{C},\emptyset}} \exists^{=|\mathbf{v}|_{(\theta,\ell)}} y. \big(\theta(y) \wedge \psi_{B,\ell}(y)\big) \end{array} \right)$$

where $|\mathbf{v}|_{(\theta,\ell)}$ is the number of occurrences of (θ,ℓ) in $\mathbf{v} = ((\theta_X,\ell_X))_{X\in P}$, i.e.,

$$|\mathbf{v}|_{(\theta,\ell)} = |\{X \in P \mid (\theta,\ell) = (\theta_X,\ell_X)\}|$$

Then Φ_6 is in normal form, which ends the proof of Theorem 23. □

A direct corollary that can be inferred from this normal form is the decidability of satisfiability for FO[∼], i.e., of the problem Synth(FO[∼], $\mathbb{N}, \mathbb{0}, \mathbb{0}$).

Corollary 25. *The satisfiability problem for* FO[∼] *is decidable. Moreover, if an* FO[∼] *formula has an infinite model, then it also has a finite one.*

Proof. Take a formula in normal form with its associated threshold B. A formula of the form

$$\varphi_{\theta,\ell}^{\bowtie m} = \exists^{\bowtie m} y. \big(\theta(y) \wedge \psi_{B,\ell}(y)\big)$$

is satisfied by the execution

$$\prod_{a\in A} (a,p_1)^{\ell(a)} \dots (a,p_n)^{\ell(a)}$$

where $p_1,\dots,p_n \in \mathbb{P}_\theta$ are pairwise distinct and $n \in \mathbb{N}$ is such that $n \bowtie m$. As long as there are no two inconsistent formulas for the same pair (θ,ℓ) such as $\varphi_{\theta,\ell}^{=k_1} \wedge \varphi_{\theta,\ell}^{=k_2}$ with $k_1 \neq k_2$ or $\varphi_{\theta,\ell}^{=k_1} \wedge \varphi_{\theta,\ell}^{\geq k_2}$ with $k_1 < k_2$, any conjunction of such formulas can also be satisfied by concatenating one satisfying execution for each pair (θ,ℓ), which gives a finite model. Therefore, for a formula φ in normal form, one simply needs to find one such conjunction without inconsistency and pick its model to get a model of φ. If there are none, then φ is unsatisfiable. □

Note that satisfiability for $\mathrm{FO}^2[\sim]$ is already NEXPTIME-hard, which even holds in the presence of unary relations only [Für83, GKV97]. It is NEXPTIME-complete due to the upper bound for $\mathrm{FO}^2[\sim, <]$ [BDM+11]. It is worth mentioning that two-variable logic with one equivalence relation on arbitrary structures also has the finite-model property [KO12].

Given a formula in normal form, let us call *goals* the clauses of the disjunction, that is, a goal is a conjunction of formulas of the from $\exists^{\bowtie m} y.\big(\theta(y) \wedge \psi_{B,\ell}(y)\big)$. The proof just above says that to satisfy a goal, it is enough to play for every ℓ occurring in the goal an execution that puts n distinct processes in local state ℓ, for some $n \in \mathbb{N}$ that satisfies $\bowtie m$ (as long as the constraints on the various ℓ are consistent). This is trivial to do when one player controls every process and has access to all letters, but very difficult when turned to a two-player game. Indeed, System cannot rely on Environment to play the correct amount of its actions on the correct amount of processes and then do nothing after.

We now turn to a game formalism called *parameterized vector games* created in order to better illustrate and reason on the synthesis problem for FO[∼]. This is a turn-based game where the arena is the set of local states, and where a number of *tokens* representing processes are moved from one state to another by the two players. Its acceptance condition reflects the normal form of FO[∼] formulas.

4.3.2 Parameterized vector games

Let us start with a few remarks on FO[∼] formulas to explain the intuitions behind the game formalism that we will present afterwards. Note that, given a formula $\varphi \in \mathrm{FO}[\sim]$ (which we suppose to be in normal form with threshold B), the order of actions in an execution does not matter. Thus, given some $\mathbb{P}$, a reasonable strategy for Environment would be to just "wait and see". More precisely, it does not put Environment into a worse position if, given the current execution $w \in \Sigma^*$, it lets the System execute as many actions as it wants in terms of a word $u \in \Sigma_{\mathsf{s}}^*$. Due to the fairness assumption, System would be able to execute all the letters from u anyway. Environment can even require System to play a word u such that $wu \models \varphi$. If System is not able to produce such a word, Environment can just sit back and do nothing, as System has no way of winning even playing by himself. Conversely, upon wu satisfying φ, Environment has to be able to come up with a word $v \in \Sigma_{\mathsf{e}}^*$ such that $(\mathbb{P}, wuv) \not\models \varphi$. This leads to a turn-based game in which System and Environment play in strictly alternate order and have to provide a satisfying and, respectively, falsifying execution.

In a second step, we can get rid of process identifiers. According to our normal form, all we are interested in is the *number* of processes that agree on their letters counted up to threshold B. That is, a finite execution can be abstracted as a *configuration* $c : L \to \mathbb{N}^{\mathbb{T}}$ where $L = \{0, \dots, B\}^A$ is the set of *local states* or *locations*. For $\ell \in L$ and $c(\ell) = (n_{\mathsf{s}}, n_{\mathsf{e}}, n_{\mathsf{se}})$, n_θ is the number of processes of type θ whose letter count up to threshold B corresponds to ℓ. We can also say that ℓ contains n_θ tokens of type θ. If it is System's turn, it will pick some pairs (ℓ, ℓ') and move some tokens of type $\theta \in \{\mathsf{s}, \mathsf{se}\}$ from ℓ to ℓ', provided $\ell(a) \leq \ell'(a)$ for all $a \in A_{\mathsf{s}}$ and $\ell(a) = \ell'(a)$ for all $a \in A_{\mathsf{e}}$. This actually corresponds to adding more system letters in the corresponding processes. The Environment proceeds analogously, with tokens of type $\theta \in \{\mathsf{e}, \mathsf{se}\}$.

Finally, the formula φ naturally translates to an acceptance condition $\mathcal{F} \subseteq \mathfrak{C}^L$ over configurations, where $\mathfrak{C}$ is the set of *local acceptance conditions*, which are of the form $(\bowtie_{\mathsf{s}} n_{\mathsf{s}}\ ,\ \bowtie_{\mathsf{e}} n_{\mathsf{e}}\ ,\ \bowtie_{\mathsf{se}} n_{\mathsf{se}})$ where $\bowtie_{\mathsf{s}}, \bowtie_{\mathsf{e}}, \bowtie_{\mathsf{se}} \in \{=, \geq\}$ and $n_{\mathsf{s}}, n_{\mathsf{e}}, n_{\mathsf{se}} \in \mathbb{N}$.

We end up with a turn-based game in which, similarly to a VASS game [BJK10, Jan15, AMSS13, RSB05, CS14], System and Environment move tokens along vectors from L. Note that, however, our games have a very particular structure so that undecidability for VASS games does not carry over to our setting. Moreover, existing decidability results do not allow us to infer our cutoff results below.

In the following, we will formalize *parameterized vector games*.

Definition 20. A *parameterized vector game* (or simply *game* when the context is clear) is given by a triple

$$\mathcal{G} = (A, B, \mathcal{F})$$

where $A = A_{\mathsf{s}} \uplus A_{\mathsf{e}}$ is the finite alphabet, $B \in \mathbb{N}$ is a bound, and, letting $L = \{0, \dots, B\}^A$ be the set of *locations*, $\mathcal{F} \subseteq \mathfrak{C}^L$ is a finite set called *acceptance condition*.

Locations. Let ℓ_0 be the location such that $\ell_0(a) = 0$ for all $a \in A$. For $\ell \in L$ and $a \in A$, we define location $\ell + a \in L$ which corresponds to ℓ where the number of a has been incremented by 1 (unless it was already at the bound B) by

$$(\ell + a)(b) = \begin{cases} \ell(b) & \text{if } b \neq a, \\ \min(\ell(a) + 1, B) & \text{otherwise.} \end{cases}$$

This is extended inductively for all words $u \in A^*$ and $a \in A$ by $\ell + \varepsilon = \ell$ and $\ell + ua = (\ell + u) + a$.

We say that ℓ' is a *successor* of ℓ if there some word $w \in A^*$ such that $\ell' = \ell + w$. If $w \in A_{\mathsf{s}}^*$ then we say that ℓ' is a system successor of ℓ, and similarly ℓ' is an environment successor of ℓ if $w \in A_{\mathsf{e}}^*$.

By $\langle w \rangle$, we denote the location $\ell_0 + w$, i.e. the location where the count of each letter is the same as the number of occurrences of that letter in w. Once again, note that the order of the letters of w do not matter, for instance, $\langle aabb \rangle = \langle abab \rangle = \langle baab \rangle$ and so on.

For instance, the set of locations for $A_{\mathsf{s}} = \{a\}$, $A_{\mathsf{e}} = \{b\}$, and $B = 2$ can be represented as in Figure 4.2: the leftmost state is $\ell_0 = \langle \varepsilon \rangle$, and the rightmost state is $\langle aabb \rangle$. Arrows represent the successor relation between locations, in the sense that for instance $\langle ab \rangle = \langle a \rangle + b$. Not represented on the figure are loops for states with at least one letter whose count is equal to the bound B, such as $\langle a^2 \rangle$, which can be reached from itself by adding more a as $\langle a^2 \rangle + a = \langle a^3 \rangle = \langle a^2 \rangle$.

Configurations. As explained above, a *configuration* of $\mathcal{G}$ is a mapping $c : L \to \mathbb{N}^{\mathbb{T}}$ that maps to each location the number of tokens of each type that are in this location. Suppose that, for $\ell \in L$, we have $c(\ell) = (n_{\mathsf{s}}, n_{\mathsf{e}}, n_{\mathsf{se}})$. Then, for all $\theta \in \mathbb{T}$ we let $c(\ell, \theta)$ refer to n_θ. By *Conf*, we denote the set of all configurations.

Transitions. A *system transition* (respectively *environment transition*) is a mapping $\tau : L \times L \to (\mathbb{N} \times \mathbb{0} \times \mathbb{N})$ (respectively $\tau : L \times L \to (\mathbb{0} \times \mathbb{N} \times \mathbb{N})$) such that, for all $(\ell, \ell') \in L \times L$ with $\tau(\ell, \ell') \neq (0, 0, 0)$, ℓ' is a system (respectively environment)

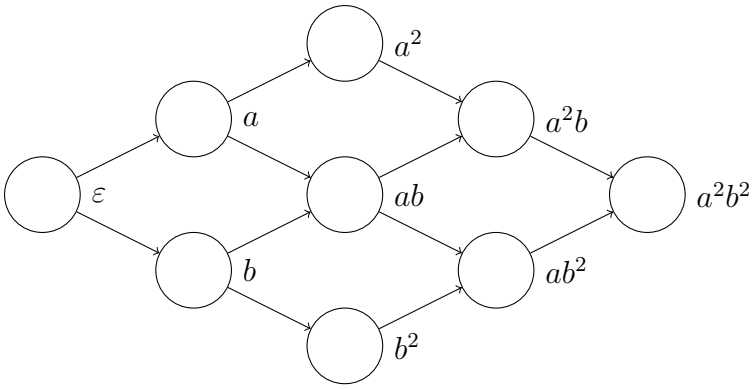

Figure 4.2: Set of locations for $A_{\mathsf{s}} = \{a\}$, $A_{\mathsf{e}} = \{b\}$, and $B = 2$.

successor of ℓ. Intuitively, a transition represents the number of tokens that are transferred from each location to each of their possible successors. Let T_{s} denote the set of system transitions, T_{e} the set of environment transitions, and $T = T_{\mathsf{s}} \cup T_{\mathsf{e}}$ the set of all transitions.

For $\tau \in T$, let the mappings $out_\tau, in_\tau : L \to \mathbb{N}^{\mathbb{T}}$ be defined by

$$out_\tau(\ell) = \sum_{\ell' \in L} \tau(\ell, \ell') \text{ and } in_\tau(\ell) = \sum_{\ell' \in L} \tau(\ell', \ell)$$

where the sum is component-wise. We say that $\tau \in T$ is *applicable* at $c \in Conf$ if, for all $\ell \in L$, we have $out_\tau(\ell) \leq c(\ell)$ (again, the comparison is component-wise). Abusing notation, we let $\tau(c)$ denote the configuration c' defined by $c'(\ell) = c(\ell) - out_\tau(\ell) + in_\tau(\ell)$ for all $\ell \in L$. Moreover, for $\tau(\ell, \ell') = (n_{\mathsf{s}}, n_{\mathsf{e}}, n_{\mathsf{se}})$ and $\theta \in \mathbb{T}$, we let $\tau(\ell, \ell', \theta)$ refer to n_θ.

Plays. Let $c \in Conf$. We write $c \models \mathcal{F}$ if there is $\kappa \in \mathcal{F}$ such that c satisfies all constraints of κ: for all $\ell \in L$ and $\kappa(\ell) = (\bowtie_{\mathsf{s}} n_{\mathsf{s}}\ , \bowtie_{\mathsf{e}} n_{\mathsf{e}}\ , \bowtie_{\mathsf{se}} n_{\mathsf{se}})$, we have $c(\ell, \theta) \bowtie_\theta n_\theta$ for all $\theta \in \mathbb{T}$. A *c-play*, or simply *play*, is a finite sequence

$$\rho = c_0 \tau_1 c_1 \tau_2 c_2 \dots \tau_n c_n$$

alternating between configurations and transitions (with $n \geq 0$) such that $c_0 = c$ and, for all $i \in \{1, \dots, n\}$, $c_i = \tau_i(c_{i-1})$ and

- if i is odd, then $\tau_i \in T_{\mathsf{s}}$ and $c_i \models \mathcal{F}$ (System's move),
- if i is even, then $\tau_i \in T_{\mathsf{e}}$ and $c_i \not\models \mathcal{F}$ (Environment's move).

The set of all c-plays is denoted by $Plays_c$.

Strategies. A *c-strategy* for System is a partial mapping $f : Plays_c \to T_{\mathsf{s}}$ such that $f(c)$ is defined and, for all $\rho = c_0 \tau_1 c_1 \dots \tau_i c_i \in Plays_c$ with $\tau = f(\rho)$ defined, we have that τ is applicable at c_i and $\tau(c_i) \models \mathcal{F}$. Play $\rho = c_0 \tau_1 c_1 \dots \tau_n c_n$ is

- *f-compatible* if, for all odd $i \in \{1, \dots, n\}$, $\tau_i = f(c_0 \tau_1 c_1 \dots \tau_{i-1} c_{i-1})$,

- *f-maximal* if it is not the strict prefix of another f-compatible play,
- *winning* if $c_n \models \mathcal{F}$.

We say that f is *winning* for System (from c) if all f-compatible f-maximal c-plays are winning. Finally, c is *winning* if there is a c-strategy that is winning.

Note that, if the initial configuration c is fixed, then we deal with an acyclic finite reachability game. Indeed, the number of tokens is constant throughout any play, so the total number of possible configurations is finite. And since tokens can only be moved along successor locations, there is a partial order over configurations, and therefore the game is acyclic. An immediate consequence is that, if there is a winning c-strategy, then there is a memoryless one.

For $\mathbf{k} \in \mathbb{N}^{\mathbb{T}}$, let $c_{\mathbf{k}}$ denote the configuration that maps ℓ_0 to $\mathbf{k}$ and all other locations to $(0,0,0)$. We set $\mathrm{Win}(\mathcal{G}) = \{\mathbf{k} \in \mathbb{N}^{\mathbb{T}} \mid c_{\mathbf{k}} \text{ is winning for System}\}$.

Definition 21. For sets $\mathcal{N}_{\mathsf{s}}, \mathcal{N}_{\mathsf{e}}, \mathcal{N}_{\mathsf{se}} \subseteq \mathbb{N}$, the game problem is given as follows:

GAME$(\mathcal{N}_{\mathsf{s}}, \mathcal{N}_{\mathsf{e}}, \mathcal{N}_{\mathsf{se}})$	
Input:	Parameterized vector game $\mathcal{G}$
Question:	$\mathrm{Win}(\mathcal{G}) \cap (\mathcal{N}_{\mathsf{s}} \times \mathcal{N}_{\mathsf{e}} \times \mathcal{N}_{\mathsf{se}}) \neq \emptyset$?

Let us show that parameterized vector games and synthesis for FO[∼] formulas are equivalent in the following sense.

Lemma 26. *For every sentence $\varphi \in$ FO[∼], there is a parameterized vector game $\mathcal{G} = (A, B, \mathcal{F})$ such that Win(φ) = Win($\mathcal{G}$). Conversely, for every parameterized vector game $\mathcal{G} = (A, B, \mathcal{F})$, there is a sentence $\varphi \in$ FO[∼] such that Win($\mathcal{G}$) = Win(φ). Both directions are effective.*

Let us illustrate this equivalence by an example first.

Example 24. Recall from Example 22 the formula

$$\varphi_7 = \forall x.\, \big[(\exists^{=1} y.y \sim x \wedge req(y)) \Leftrightarrow (\exists^{=1} y.y \sim x \wedge ack(y))\big]$$

with $A_{\mathsf{s}} = \{ack\}$, $A_{\mathsf{e}} = \{req\}$, and its associated normal form

$$\varphi_7' = \bigwedge_{\theta \in \mathbb{T}, \ell \in L_7} \exists^{=0} y.(\theta(y) \wedge \psi_{2,\ell}(y))$$

where L_7 is the set of local states of the form $\langle req^i ack^j \rangle \in L$ such that $i = 1 \neq j$ or $i \neq 1 = j$. Recall that $\psi_{2,\ell}(y)$ is an FO[∼] formula checking that y belongs to a process with local state ℓ (with a bound $B = 2$).

To create an equivalent game $\mathcal{G}_7 = (A, 2, \mathcal{F})$, we need to define the acceptance condition $\mathcal{F}$. This acceptance condition simply mirrors the normal form φ_7'. It has a single element $\mathcal{F} = \{\kappa\}$, with κ defined as follows:

$$\kappa(\ell) = \begin{cases} (=0, =0, =0) & \text{if } \ell \in L_7, \\ (\geq 0, \geq 0, \geq 0) & \text{otherwise} \end{cases}$$

It is then easy to check that $\mathrm{Win}(\varphi_7) = \mathrm{Win}(\mathcal{G}_7) = \mathbb{N} \times \mathbb{0} \times \mathbb{N}$.

The rest of this section is dedicated to the proof of Lemma 26. As an intermediate step in the translation of the synthesis problem into games, we first consider a *normalized* version of the former. In a second step, we show equivalence between the normalized synthesis problem and games.

Step 1: Normalized Synthesis Problem for FO[∼]. In the normalized synthesis problem, instead of being fully asynchronous, both players will alternately give a sequence of events instead of a single one. Moreover, after every move from System, the partial word created up to that point should satisfy the formula, whereas after every move from Environment, the word should falsify the formula.

Let us fix, for the rest of the definitions, a sentence $\varphi \in \mathrm{FO}[\sim]$. We call a finite $\mathbb{P}$-execution $w \in \Sigma^*$ *normalized* if it is of the form $w = w_1 \dots w_n$ with $n \geq 1$ such that

- for all odd i such that $1 \leq i \leq n$, $w_i \in \Sigma_{\mathsf{s}}^*$ and $w_1 \dots w_i \models \varphi$,
- for all even i such that $1 \leq i \leq n$, $w_i \in \Sigma_{\mathsf{e}}^*$ and $w_1 \dots w_i \not\models \varphi$.

Note that the decomposition into the w_i, if it exists, is uniquely determined.

A *normalized* $\mathbb{P}$-strategy (for System) is a partial mapping $f : \Sigma^* \to \Sigma_{\mathsf{s}}^*$ such that $f(\varepsilon)$ is defined and, if $f(w)$ is defined, then $w \cdot f(w) \models \varphi$. A normalized $\mathbb{P}$-execution $w = w_1 \dots w_n$ is

- *f-compatible* if, for all odd $1 \leq i \leq n$, we have $w_i = f(w_1 \dots w_{i-1})$,
- *f-maximal* if it is not the strict prefix of an f-compatible normalized $\mathbb{P}$-execution,
- *winning* if $w \models \varphi$.

Finally, a normalized strategy is $\mathbb{P}$-*winning* if all f-compatible f-maximal normalized $\mathbb{P}$-executions are winning.

Similarly to the initial synthesis problem, we define the normalized winning set $\mathrm{Win}_{\mathsf{norm}}(\varphi)$ as the set of triples $(k_{\mathsf{s}}, k_{\mathsf{e}}, k_{\mathsf{se}}) \in \mathbb{N}^{\mathbb{T}}$ for which there is $\mathbb{P} = (\mathbb{P}_{\mathsf{s}}, \mathbb{P}_{\mathsf{e}}, \mathbb{P}_{\mathsf{se}})$ such that

- $|\mathbb{P}_\theta| = k_\theta$ for all $\theta \in \mathbb{T}$, and
- there is a normalized $\mathbb{P}$-strategy that is $\mathbb{P}$-winning.

Now, the original and the normalized synthesis problem are equivalent in the following sense:

Lemma 27. *$Win(\varphi) = Win_{\mathsf{norm}}(\varphi)$.*

Proof. We say that two executions w and w' are *similar*, noted $w \sim w'$, if w' is w with the position of its events rearranged in any combination, i.e. $w \sim w'$ if there exists a letter-preserving bijection from $Pos(w)$ to $Pos(w')$. Note that in FO[∼], there is no way to write constraints on the relative order of positions. In other words, for any $\varphi \in \mathrm{FO}[\sim]$, if $w \models \varphi$ and $w \sim w'$, then $w' \models \varphi$ too. This is the property that we use to prove that the synthesis problem is equivalent to the normalized one.

For the remainder of this proof, let us fix $\mathbb{P} = (\mathbb{P}_{\mathsf{s}}, \mathbb{P}_{\mathsf{e}}, \mathbb{P}_{\mathsf{se}})$ and its corresponding triple $(k_{\mathsf{s}}, k_{\mathsf{e}}, k_{\mathsf{se}})$. $\mathbb{P}$-executions and $\mathbb{P}$-strategies will simply be referred to as executions and strategies respectively.

$\underline{\mathrm{Win}(\varphi) \supseteq \mathrm{Win}_{\mathsf{norm}}(\varphi)}$: Suppose that $(k_{\mathsf{s}}, k_{\mathsf{e}}, k_{\mathsf{se}}) \in \mathrm{Win}_{\mathsf{norm}}(\varphi)$ and let f_N be a normalized winning strategy for System. We want to build f a winning strategy in the Synthesis Problem.

The idea is to simulate f_N by memorizing the word of actions given by f_N and playing it one action at a time. Meanwhile, the actions played by Environment are stored and then processed as if they happened all at once after System finishes playing its word, thus simulating a corresponding normalized run.

We define a function mem such that for all executions $w = \sigma_1\sigma_2\ldots$, $\mathrm{mem}(w) = (w_N, w_\mathsf{s}, w_\mathsf{e})$ where w_N is the corresponding normalized run, w_s is the word that System must play to simulate the choice of f_N, and w_e stores the actions played by Environment in the meantime. It is defined as follows:

- $\mathrm{mem}(\varepsilon) = (\varepsilon, f_N(\varepsilon), \varepsilon)$
- If $\sigma \in \Sigma_\mathsf{s}$ and $\mathrm{mem}(w) = (w_N, w_\mathsf{s}, w_\mathsf{e})$, then

$$\mathrm{mem}(w\cdot\sigma) = \begin{cases} (w_N\cdot\sigma, w'_\mathsf{s}, w_\mathsf{e}) & \text{if } w_\mathsf{s} = \sigma\cdot w'_\mathsf{s}, \\ (w_N\cdot w_\mathsf{e}\cdot\sigma, w'_\mathsf{s}, \varepsilon) & \text{if } w_\mathsf{s} = \varepsilon \text{ and } f_N(w_N\cdot w_\mathsf{e}) = \sigma\cdot w'_\mathsf{s}, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

- If $\sigma \in \Sigma_\mathsf{e}$ and $\mathrm{mem}(w) = (w_N, w_\mathsf{s}, w_\mathsf{e})$, then

$$\mathrm{mem}(w\cdot\sigma) = (w_N, w_\mathsf{s}, w_\mathsf{e}\cdot\sigma)$$

Then we define an auxiliary function f_{aux}:

$$f_{\mathrm{aux}}(w_N, w_\mathsf{s}, w_\mathsf{e}) = \begin{cases} \sigma & \text{if } w_\mathsf{s} = \sigma\cdot w'_\mathsf{s}, \\ \sigma & \text{if } w_\mathsf{s} = \varepsilon \text{ and } f_N(w_N\cdot w_\mathsf{e}) = \sigma\cdot w'_\mathsf{s}, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Finally, we define the strategy f as $f(w) = f_{\mathrm{aux}}(\mathrm{mem}(w))$ when both f_{aux} and mem are defined, otherwise $f(w) = \varepsilon$.

From these definitions, we can immediately state the following properties describing the workings of mem and f:

1. If $w = w'\sigma$ is a f-compatible execution such that $\sigma \in \Sigma_\mathsf{s}$ and $\mathrm{mem}(w') = (w_N, \varepsilon, w_\mathsf{e})$, then $\mathrm{mem}(w) = (w_N w_\mathsf{e}\sigma, \sigma_2\ldots\sigma_n, \varepsilon)$ with $\sigma\sigma_2\ldots\sigma_n = f_N(w_N w_\mathsf{e})$.
2. If $w = w'w_0\sigma_1 w_1\ldots\sigma_m w_m$ is a f-compatible execution such that for all $i \leq m$ $w_i \in \Sigma_\mathsf{e}^*$ and $\sigma_i \in \Sigma_\mathsf{s}$, $\mathrm{mem}(w') = (w_N, \sigma'_1\ldots\sigma'_n, w_\mathsf{e})$, for all $j < m$ $\mathrm{mem}(w'w_0\ldots\sigma_j) \neq (*, \varepsilon, *)$, and $\mathrm{mem}(w) = (*, \varepsilon, *)$, then $n = m$, for all $i \leq n$ $\sigma_i = \sigma'_i$, and $\mathrm{mem}(w) = (w_N\sigma_1\ldots\sigma_n, \varepsilon, w_\mathsf{e}w_0\ldots w_n)$.
3. If $f(w) = \varepsilon$ for some f-compatible execution w then either $w = \varepsilon$ and $f_N(w) = \varepsilon$, or $\mathrm{mem}(w) = (w_N, \varepsilon, w_\mathsf{e})$ with w_N and w_e such that $f_N(w_N\cdot w_\mathsf{e})$ is undefined.

Let w be a finite f-compatible execution such that $\mathrm{mem}(w) = (w_N, \varepsilon, w_\mathsf{e})$. This execution can always be decomposed as $w = w_0\sigma_0 w_1\ldots\sigma_m w_m$ with $w_i \in \Sigma_\mathsf{e}^*$ and $\sigma_i \in \Sigma_\mathsf{s}$ for all $i \leq m$. We show that w can also be written as:

$$w = w_0^1\sigma_1^1\ldots\sigma_{n_1}^1 w_{n_1}^1 \cdot \sigma_0^2 w_0^2\sigma_1^2\ldots\sigma_{n_2}^2 w_{n_2}^2 \cdot \ldots \cdot \sigma_0^k w_0^k\sigma_1^k\ldots\sigma_{n_k}^k w_{n_k}^k$$

where $k, n_1, \ldots, n_k \in \mathbb{N}$ and such that if we define $\mathsf{s}^j = \sigma_0^j\ldots\sigma_{n_j}^j$ and $\mathsf{e}^j = w_0^j\ldots w_{n_j}^j$ for all $j \leq k$ then:

- $w_N = \mathsf{s}^1\mathsf{e}^1\mathsf{s}^2 \dots \mathsf{e}^{k-1}\mathsf{s}^k$,
- $w_\mathsf{e} = \mathsf{e}^k$, and
- for all $j \leq k$, $\mathsf{s}^j = f_N(\mathsf{s}^1\mathsf{e}^1 \dots \mathsf{s}^{j-1}\mathsf{e}^{j-1})$.

We prove this by recursion on the number of prefixes w' of w ending in an action of System such that $\mathrm{mem}(w') = (*, \varepsilon, *)$, that number being k in the decomposition above.

Let $w = w_0\sigma_0 \dots \sigma_m w_m$ a f-compatible execution with $\mathrm{mem}(w) = (w_N, \varepsilon, w_\mathsf{e})$. We note $(w_N^i, w_\mathsf{s}^i, w_\mathsf{e}^i) = \mathrm{mem}(w_0\sigma_0 \dots w_i\sigma_i)$ for all $i \leq m$.

Base case ($k = 1$). Suppose that $w_\mathsf{s}^i \neq \varepsilon$ for all $i < m$ and $w_\mathsf{s}^m = \varepsilon$. As $\mathrm{mem}(\varepsilon) = (\varepsilon, f_N(\varepsilon), \varepsilon)$, if we let $f_N(\varepsilon) = \sigma'_1 \dots \sigma'_n$ then by the second property of mem we have that $n = m$, $\sigma_i = \sigma'_i$ for all $i \leq m$, and that $\mathrm{mem}(w) = (\sigma_1 \dots \sigma_n, \varepsilon, w_0 \dots w_m)$. Then if we let $\mathsf{s}^1 = \sigma_1 \dots \sigma_n$ and $\mathsf{e}^1 = w_0 \dots w_m$, the decomposition holds (with $k = 1$).

Induction step. Suppose $w = w' \cdot \sigma_i w_{i+1} \dots \sigma_m w_m$ where $w' = w_0^1\sigma_1^1 \dots \sigma_{n_k}^k w_{n_k}^k$ with $\mathrm{mem}(w') = (w_N, \varepsilon, w_\mathsf{e})$ satisfying the conditions above. Suppose also that $w_\mathsf{s}^j \neq \varepsilon$ for all $i \leq j < m$ and $w_\mathsf{s}^m = \varepsilon$. By the first property of mem, we know that $\mathrm{mem}(w'\sigma_i) = (w_N \cdot w_\mathsf{e} \cdot \sigma_i, \sigma'_2 \dots \sigma'_n, \varepsilon)$ with $\sigma_i\sigma'_2 \dots \sigma'_n = f_N(w_N \cdot w_\mathsf{e}) = f_N(\mathsf{s}^1\mathsf{e}^1 \dots \mathsf{s}^k \cdot \mathsf{e}^k)$. Then using the second property, we deduce that $n = m - i$, $\sigma'_{j+1} = \sigma_{i+j}$ for all $0 < j \leq n$, and that $\mathrm{mem}((w'\sigma_i) \cdot w_{i+1}\sigma_{i+1} \dots \sigma_m w_m) = (w'_N, \varepsilon, w'_\mathsf{e})$ where $w'_N = w_N \cdot w_\mathsf{e} \cdot \sigma_i\sigma_{i+1} \dots \sigma_m$ and $w'_\mathsf{e} = w_{i+1} \dots w_m$. So we let $\mathsf{s}^{k+1} = \sigma_i \dots \sigma_m$ and $\mathsf{e}^{k+1} = w_{i+1} \dots w_m$, and all conditions of the decomposition have been satisfied.

Thanks to the decomposition we just proved, if w is a finite f-compatible execution such that $\mathrm{mem}(w) = (w_N, \varepsilon, w_\mathsf{e})$ then we can deduce two facts: that $w \sim w_N \cdot w_\mathsf{e}$ and that w_N is a f_N-compatible normalized execution.

Let w be an arbitrary fair f-compatible execution. We distinguish two different cases:

If w is finite and $\mathrm{mem}(w) = (w_N, w_\mathsf{s}, w_\mathsf{e})$, then $f(w) = \varepsilon$ because w is fair, so either $w = \varepsilon = f_N(w)$ in which case $(\mathbb{P}, \varepsilon) \models \varphi$ because f_N is a normalized strategy, or $w_\mathsf{s} = \varepsilon$ and f_N is undefined on $w_N \cdot w_\mathsf{e}$. Moreover, we get that $w \sim w_N \cdot w_\mathsf{e}$ and that w_N is a f_N-compatible normalized execution. Since w_N is a f_N-compatible normalized execution and f_N is undefined on $w_N \cdot w_\mathsf{e}$, then necessarily $w_N \cdot w_\mathsf{e}$ satisfies φ, otherwise $w_N \cdot w_\mathsf{e}$ would be a f_N-compatible maximal normalized execution that is not winning which would contradict that f_N is winning. Therefore, since $w_N \cdot w_\mathsf{e}$ satisfies φ and $w \sim w_N \cdot w_\mathsf{e}$, we have that w satisfies φ.

If w is infinite, let w_i be the prefix of size i of w and $(w_N^i, w_\mathsf{s}^i, w_\mathsf{e}^i) = \mathrm{mem}(w_i)$. We again distinguish two cases. If there are an infinite number of actions from System, then there is an infinite sequence $i_1 < i_2 < \dots$ such that $w_\mathsf{s}^{i_j} = \varepsilon$, which in turn means that there is an increasing sequence of f_N-compatible normalized executions $w_N^{i_1}, w_N^{i_2}, \dots$, so one can find a normalized execution of arbitrary size. This is impossible, as by Theorem 23 there is a bound on the number of letter that can be played on a single process before the satisfiability of φ remains stable, that bound being $B.|A_\theta|$ for processes of type $\theta \in \mathbb{T}$. Since the number of processes is fixed that means there is a bound on the total number of times that an execution can go from satisfying φ to not satisfying it and vice-versa, which in turn limits the size of normalized executions.

Therefore there is a finite number of actions from System, i.e. $w = w' \cdot w_\mathsf{e}^\infty$ where w' is a finite execution ending with an action from System and w_e^∞ is an infinite

execution with only actions from Environment. Let $n = |w'|$. By fairness of w necessarily there is some point $K > n$ such that $f(w_i) = \varepsilon$ for all $i \geq K$. Since there are no actions from System in w_{e}^{∞}, we also know that $w_N^i = w_N^n$ and $w_{\mathsf{s}}^i = w_{\mathsf{s}}^n = \varepsilon$ for all $i > n$, and that w_N^n is a f_N-compatible normalized execution. Thus for all $i \geq K$, $w_i \sim w_N^n \cdot w_{\mathsf{e}}^i$ and $w_N^n \cdot w_{\mathsf{e}}^i$ satisfies φ otherwise f_N would not be winning, therefore w_i satisfies φ for all $i \geq K$. We conclude that w is winning, and therefore that f is a winning strategy in the Synthesis Problem.

$\underline{\mathrm{Win}(\varphi) \subseteq \mathrm{Win}_{\mathsf{norm}}(\varphi)}$: Suppose that $(k_{\mathsf{s}}, k_{\mathsf{e}}, k_{\mathsf{se}}) \in \mathrm{Win}(\varphi)$ and let f be a winning strategy for System. We will define f_N a normalized strategy. Let w be a finite normalized execution; note that w can also be seen as a (regular) execution. Suppose that w is a f-compatible execution that does not satisfy φ. Let $\sigma_1 = f(w)$, $\sigma_2 = f(w\sigma_1)$, $\sigma_3 = f(w\sigma_1\sigma_2)$, and so on. As f is winning, necessarily there exists $i \in \mathbb{N}$ such that $\sigma_1, \ldots, \sigma_i$ are all not ε and such that $w\sigma_1 \ldots \sigma_i$ satisfies φ, otherwise $w\sigma_1\sigma_2 \ldots$ would be an infinite f-compatible fair execution that is not winning. We then take the minimal i satisfying those conditions and we define $f_N(w) = \sigma_1 \ldots \sigma_i$. Remark that in that case, $w \cdot f_N(w)$ is still a f-compatible execution. If $(\mathbb{P}, \varepsilon) \models \varphi$, we let $f_N(\varepsilon) = \varepsilon$, and the remark above still holds. If $w \neq \varepsilon$ either satisfies φ or is not f-compatible, then f_N is undefined.

Let $w = w_{\mathsf{s}}^1 w_{\mathsf{e}}^1 w_{\mathsf{s}}^2 \ldots w_{\theta}^i$ be a f_N-compatible normalized execution with $\theta \in \{\mathsf{s}, \mathsf{e}\}$. Then w is also a f-compatible execution: this is true if $w = \varepsilon$, and if w' is a f-compatible execution then $w' \cdot f_N(w')$ is also one as we remarked earlier, and $w' \cdot f_N(w') \cdot w_{\mathsf{e}}$ as well for any $w_{\mathsf{e}} \in \Sigma_{\mathsf{e}}^+$.

Now suppose that w is also f_N-maximal. If $\theta = \mathsf{s}$ then w is winning. Otherwise, by definition of maximal $f_N(w)$ must be undefined. f_N is undefined when w is not f-compatible or satisfies φ. Since w is f-compatible, it means that w must satisfy φ. Thus all maximal f_N-compatible normalized executions are winning, which means that f_N is a winning strategy. □

Now that we have proven that the synthesis problem is equivalent to the normalized version, let us use the latter to show equivalency with parameterized vector games.

Step 2: Proof of Lemma 26 We split the lemma into two, one for each direction.

Lemma 28. *For every sentence $\varphi \in \mathrm{FO}[\sim]$, there is a parameterized vector game $\mathcal{G}$ such that $\mathit{Win}(\varphi) = \mathit{Win}(\mathcal{G})$.*

Proof. We actually show that parameterized vector games are equivalent to the normalized synthesis problem. Let φ be a sentence in FO[∼]. With the normal form from Theorem 23, we suppose that there is $B \in \mathbb{N}$ and that

$$\varphi = \bigvee_{i=1}^{n} \varphi_i$$

where

$$\varphi_i = \left(\bigwedge_{j=1}^{m_i} \exists^{=k_j^i} y.(\theta_j^i(y) \wedge \psi_{B,\ell_j^i}(y)) \right) \wedge \left(\bigwedge_{j=1}^{\hat{m}_i'} \exists^{\geq \hat{k}_j^i} y.(\hat{\theta}_j^i(y) \wedge \psi_{B,\hat{\ell}_j^i}(y)) \right)$$

with $k^i_j, \hat{k}^i_j \in \mathbb{N}$, $\theta^i_j, \hat{\theta}^i_j \in \mathbb{T}$, $\ell^i_j, \hat{\ell}^i_j \in \{0 \dots B\}^A$ for all $i, j \in \mathbb{N}$. Let $L = \{0 \dots B\}^A$, we can also assume that for all $i \in \mathbb{N}$, any pair $(\theta, \ell) \in \mathbb{T} \times L$ appears at most once in $\cup_j \{(\theta^i_j, \ell^i_j), (\hat{\theta}^i_j, \hat{\ell}^i_j)\}$.

We define the parameterized vector game $\mathcal{G} = (A, B, \mathcal{F})$ where A and B are given by φ, and $\mathcal{F} = \{\kappa_i \mid 1 \leq i \leq n\}$ such that for all $1 \leq i \leq n$ and $\ell \in L$, $\kappa_i(\ell) = (\bowtie^i_{\mathsf{s}} n^i_{\mathsf{s}}, \bowtie^i_{\mathsf{e}} n^i_{\mathsf{e}}, \bowtie^i_{\mathsf{se}} n^i_{\mathsf{se}})$ where

$$\bowtie^i_\theta n^i_\theta = \begin{cases} = k^i_j & \text{if } \exists j.(\theta, \ell) = (\theta^i_j, \ell^i_j), \\ \geq \hat{k}^i_j & \text{if } \exists j.(\theta, \ell) = (\hat{\theta}^i_j, \hat{\ell}^i_j), \\ \geq 0 & \text{otherwise.} \end{cases}$$

$\underline{\mathrm{Win}_{\mathsf{norm}}(\varphi) \subseteq \mathrm{Win}(\mathcal{G})}$: First we show how to obtain a play from a normalized execution. Let w be a normalized $(\mathbb{P}_{\mathsf{s}}, \mathbb{P}_{\mathsf{e}}, \mathbb{P}_{\mathsf{se}})$-execution and $k_\theta = |\mathbb{P}_\theta|$. By abuse of notation, we note $\rho(w)$ the play corresponding to w that we are building. Let $w = w_1 \dots w_\alpha$ with $\alpha \geq 1$ and let $c_0 = c_{(k_{\mathsf{s}}, k_{\mathsf{e}}, k_{\mathsf{se}})}$. For all $p \in \mathbb{P}_{\mathsf{s}} \cup \mathbb{P}_{\mathsf{e}} \cup \mathbb{P}_{\mathsf{se}}$ and $\beta \in \{1, \dots, \alpha\}$, we define ℓ^β_p that track the state of each process p after $w_1 \dots w_\beta$ as:

$$\ell^\beta_p(a) = |\{j \in Pos(w_1 \dots w_\beta) \mid w[j] = (a, p)\}|$$

By convention, we also let $\ell^0_p = \ell_0$ for all p. Then let $\mathbb{P}^\beta_{\theta,\ell}$ be the set of processes of type θ in state ℓ after $w_1 \dots w_\beta$, defined as

$$\mathbb{P}^\beta_{\theta,\ell} = \{p \in \mathbb{P}_\theta \mid \ell = \ell^\beta_p\}$$

Then we define $\rho(w) = c_0 \tau_1 c_1 \dots \tau_\alpha c_\alpha$ where for all $\beta \in \{1, \dots, \alpha\}$ and $(\ell, \ell') \in L^2$, $\tau_\beta(\ell, \ell') = (\tau_{\mathsf{s}}, \tau_{\mathsf{e}}, \tau_{\mathsf{se}})$ with $\tau_\theta = |\mathbb{P}^{\beta-1}_{\theta,\ell} \cap \mathbb{P}^\beta_{\theta,\ell'}|$ and $c_\beta = \tau_\beta(c_{\beta-1})$.

Let $w = w_1 \dots w_\alpha$ be a normalized execution and let $\rho(w) = c_0 \dots c_\alpha$. We prove that for all $\ell \in L$, $\beta \leq \alpha$, and $\theta \in \mathbb{T}$ we have $c_\beta(\ell, \theta) = |\mathbb{P}^\beta_{\theta,\ell}|$. If $\beta = 0$ then $c_\beta(\ell_0, \theta) = k_\theta = |\mathbb{P}^0_{\theta,\ell}|$. If the property is true for $\beta < \alpha$, then

$$\begin{aligned} c_{\beta+1}(\ell) &= c_\beta(\ell) - out_{\tau_{\beta+1}}(\ell) + in_{\tau_{\beta+1}}(\ell) \\ &= c_\beta(\ell) - \sum_{\ell' \in L} \tau_{\beta+1}(\ell, \ell') + \sum_{\ell' \in L} \tau_{\beta+1}(\ell', \ell) \end{aligned}$$

For a given $\theta \in \mathbb{T}$, this simplifies into

$$\begin{aligned} c_{\beta+1}(\ell, \theta) &= |\mathbb{P}^\beta_{\theta,\ell}| - \sum_{\ell' \in L} \left(|\mathbb{P}^\beta_{\theta,\ell} \cap \mathbb{P}^{\beta+1}_{\theta,\ell'}| \right) + \sum_{\ell' \in L} \left(|\mathbb{P}^\beta_{\theta,\ell'} \cap \mathbb{P}^{\beta+1}_{\theta,\ell}| \right) \\ &= |\mathbb{P}^\beta_{\theta,\ell}| - |\{p \in \mathbb{P}_\theta \mid p \in \mathbb{P}^\beta_{\theta,\ell} \wedge p \notin \mathbb{P}^{\beta+1}_{\theta,\ell}\}| + |\{p \in \mathbb{P}_\theta \mid p \notin \mathbb{P}^\beta_{\theta,\ell} \wedge p \in \mathbb{P}^{\beta+1}_{\theta,\ell}\}| \\ &= |\mathbb{P}^{\beta+1}_{\theta,\ell}| \end{aligned}$$

Consequently, we can prove that w is winning iff $\rho(w)$ is winning. Suppose there is $i \leq n$ such that $w_1 \dots w_\alpha \models \varphi_i$. Then by definition of the subformulas $\psi_{B,\ell^i_j,(}(y)$, for all (θ, ℓ) and j such that $(\theta, \ell) = (\theta^i_j, \ell^i_j)$ (respectively $= (\hat{\theta}^i_j, \hat{\ell}^i_j)$), there must be at exactly k^i_j (resp. at least $\hat{k}^i_j$) processes of type θ in state ℓ i.e. $|\mathbb{P}^\alpha_{\theta,\ell}| = k^i_j$ (resp. $\geq \hat{k}^i_j$). Therefore $c_\alpha(\ell, \theta) = k^i_j$ (resp. $\geq \hat{k}^i_j$) for all (θ, ℓ), so c_α satisfies κ_i thus $\rho(w)$ is winning. The other direction is similar.

Now suppose there is a winning normalized $\mathbb{P}$-strategy f. We define a strategy $f_{\mathcal{G}}$ in $\mathcal{G}$ as $f_{\mathcal{G}}(\rho) = \tau_\alpha$ if there is a f-compatible play w such that $\rho = \rho(w)$,

$f(w)$ is defined and $\rho(w \cdot f(w)) = c_0 \dots \tau_\alpha c_\alpha$. Moreover, let $f_\mathcal{G}(\varepsilon) = \tau_1$ where $\rho(f(\varepsilon)) = c_0\tau_1 c_1$. In all other cases, $f_\mathcal{G}$ is undefined.

Finally, we show that $f_\mathcal{G}$ is winning. If $\rho = c_0\tau_1 c_1 \dots c_\alpha$ is a $f_\mathcal{G}$-compatible play, then inductively by definition of $f_\mathcal{G}$ we know that there is $w = w_1 \dots w_\alpha$ such that for all $\beta \leq \alpha$, $c_0\tau_1 c_1 \dots c_\beta = \rho(w_1 \dots w_\beta)$. Furthermore, if ρ is $f_\mathcal{G}$-maximal, then there it is not the prefix of a longer $f_\mathcal{G}$-compatible play. If w was not f-maximal, there would be an execution $w' = ww_{\alpha+1}$ that is f-compatible, but in that case $\rho(w')$ would be a $f_\mathcal{G}$-compatible play which contradicts the maximality of ρ. Therefore w must be f-maximal, and thus winning as f is a winning strategy. Since we proved that w is winning iff $\rho(w)$ is winning, then ρ is a winning play, therefore $f_\mathcal{G}$ is a winning strategy.

$\underline{\mathrm{Win}_{\mathsf{norm}}(\varphi) \supseteq \mathrm{Win}(\mathcal{G})}$: Let $c_0 = c_{(k_\mathsf{s}, k_\mathsf{e}, k_\mathsf{se})}$ for some $k_\mathsf{s}, k_\mathsf{e}, k_\mathsf{se} \in \mathbb{N}$. We define $\mathbb{P}_\theta = \{1, \dots, k_\theta\}$ for all $\theta \in \mathbb{T}$. For all c_0-plays ρ, let us build a normalized $(\mathbb{P}_\mathsf{s}, \mathbb{P}_\mathsf{e}, \mathbb{P}_\mathsf{se})$-execution that we note $w(\rho)$ again by abuse of notation. Since processes in $\mathcal{G}$ do not have identities, we will need to arbitrarily assign one to each of them. To that end, we define a function mem such that for all c_0-plays $\rho \in \mathit{Plays}$, locations $\ell \in L$, and $\theta \in \mathbb{T}$, $\mathrm{mem}(\rho, \ell, \theta) = S$ with $S \subseteq \mathbb{P}_\theta$ storing the identities of all processes in location ℓ at the end of play ρ. First we fix an arbitrary total order $<$ on L^2. Then mem is defined as follows:

$$\mathrm{mem}(c_0, \ell, \theta) = \begin{cases} \mathbb{P}_\theta & \text{if } \ell = \ell_0, \\ \emptyset & \text{otherwise.} \end{cases}$$

and for all $\rho = c_0\tau_1 \dots c_\alpha$ such that $\mathrm{mem}(\rho, \ell, \theta)$ is defined for all $(\ell, \theta) \in L \times \mathbb{T}$, for all τ applicable at c_α and $c = \tau(c_\alpha)$, for all $\ell, \ell' \in L$ such that $\tau(\ell, \ell') = (n_\mathsf{s}, n_\mathsf{e}, n_\mathsf{se})$, for all $\theta \in \mathbb{T}$, we define $S^\theta_{\ell,\ell'}$ as the n_θ lowest (w.r.t. the natural order on $\mathbb{N}$) elements of

$$\mathrm{mem}(\rho, \ell, \theta) \setminus \left(\bigcup_{(\hat{\ell}, \hat{\ell}') < (\ell, \ell')} S^\theta_{\hat{\ell}, \hat{\ell}'} \right)$$

which is always well-defined if τ is applicable as we supposed. Then we let

$$\mathrm{mem}(\rho\tau c, \ell, \theta) = \mathrm{mem}(\rho, \ell, \theta) \cup \left(\bigcup_{\ell' \neq \ell} S^\theta_{\ell', \ell} \right) \setminus \left(\bigcup_{\ell' \neq \ell} S^\theta_{\ell, \ell'} \right)$$

With that being done, we define $w(\rho)$ recursively. Let $w(c_0) = \varepsilon$. For all $\rho = c_0\tau_1 c_1 \dots c_\alpha$ such that $w(\rho)$ is defined, for all τ applicable at c_α and $c = \tau(c_\alpha)$, we let

$$w(\rho\tau c) = w(\rho) \cdot \prod_{\substack{\ell' = \ell + a_1 \dots a_j, \\ \theta \in \mathbb{T} \\ p \in \mathrm{mem}(\rho, \ell, \theta) \cap \\ \mathrm{mem}(\rho\tau c, \ell', \theta)}} (a_1, p) \cdot \dots \cdot (a_j, p)$$

We prove that for all c_0-plays $\rho = c_0\tau_1 \dots c_\alpha$ and $w(\rho) = w_1 \dots w_\alpha$, for all $\ell \in L$, $\beta \leq \alpha$, and $\theta \in \mathbb{T}$ we have $\mathrm{mem}(c_0\tau_1 \dots c_\beta, \ell, \theta) = \mathbb{P}^\beta_{\theta,\ell}$ with $\mathbb{P}^\beta_{\theta,\ell}$ defined as before. If $\beta = 0$, for all processes p we have that $\ell^0_p = \ell_0$, so $\mathbb{P}^0_{\theta,\ell} = \mathbb{P}_\theta$ if $\ell = \ell_0$ and $\emptyset$ otherwise, therefore $\mathbb{P}^0_{\theta,\ell} = \mathrm{mem}(c_0, \ell, \theta)$. If the property holds for some $\beta < \alpha$, then

$$\mathrm{mem}(c_0\tau_1 \dots \tau_{\beta+1} c_{\beta+1}, \ell, \theta) = \mathbb{P}^\beta_{\theta,\ell} \cup \left(\bigcup_{\ell' \neq \ell} S^\theta_{\ell', \ell} \right) \setminus \left(\bigcup_{\ell' \neq \ell} S^\theta_{\ell, \ell'} \right)$$

Moreover,

$$\begin{aligned}\mathbb{P}^{\beta+1}_{\theta,\ell} &= \mathbb{P}^{\beta}_{\theta,\ell} \cup \{p \in \mathbb{P}_\theta \mid p \notin \mathbb{P}^{\beta}_{\theta,\ell} \wedge p \in \mathbb{P}^{\beta+1}_{\theta,\ell}\} \setminus \{p \in \mathbb{P}_\theta \mid p \in \mathbb{P}^{\beta}_{\theta,\ell} \wedge p \notin \mathbb{P}^{\beta+1}_{\theta,\ell}\} \\ &= \mathbb{P}^{\beta}_{\theta,\ell} \cup \left(\bigcup_{\ell' \neq \ell} \{p \in \mathbb{P}_\theta \mid p \in \mathbb{P}^{\beta}_{\theta,\ell'} \wedge p \in \mathbb{P}^{\beta+1}_{\theta,\ell}\}\right) \setminus \left(\bigcup_{\ell' \neq \ell} \{p \in \mathbb{P}_\theta \mid p \in \mathbb{P}^{\beta}_{\theta,\ell} \wedge p \in \mathbb{P}^{\beta+1}_{\theta,\ell'}\}\right)\end{aligned}$$

If $p \in \mathbb{P}_\theta$ is such that $p \in \mathbb{P}^{\beta}_{\theta,\ell'}$ and $p \in \mathbb{P}^{\beta+1}_{\theta,\ell}$ for some $\ell' \neq \ell$, then by definition of $w(\rho)$ necessarily $\ell = \ell' + a_1 \dots a_j$ and $p \in \mathrm{mem}(c_0 \dots c_\beta, \ell', \theta) \cap \mathrm{mem}(c_0 \dots c_{\beta+1}, \ell, \theta)$, and therefore $p \in S^{\theta}_{\ell',\ell}$. The reverse is also true. Therefore,

$$\begin{aligned}\mathbb{P}^{\beta+1}_{\theta,\ell} &= \mathrm{mem}(c_0 \dots c_\beta, \ell, \theta) \cup \left(\bigcup_{\ell' \neq \ell} S^{\theta}_{\ell',\ell}\right) \setminus \left(\bigcup_{\ell' \neq \ell} S^{\theta}_{\ell,\ell'}\right) \\ &= \mathrm{mem}(c_0 \dots c_\beta \tau_{\beta+1} c_{\beta+1}, \ell, \theta)\end{aligned}$$

Furthermore, it is easy to see that $c_\beta(\ell, \theta) = |\mathrm{mem}(c_0 \dots c_\beta, \ell, \theta)|$ for all β, ℓ, θ. Therefore, as in the other direction, we have that $c_\beta(\ell, \theta) = |\mathbb{P}^{\beta}_{\theta,\ell}|$, which in turn gives us that ρ is winning iff $w(\rho)$ is winning.

Now suppose there is a winning strategy $f_{\mathcal{G}}$ in $\mathcal{G}$. We define a normalized strategy f as $f(\varepsilon) = w(c_0 \tau c_1)$ with $\tau = f_{\mathcal{G}}(c_0)$ and $c_1 = \tau(c_0)$, and for all w we define $f(w) = w'$ if there is a play ρ ending in c such that $w = w(\rho)$, $f_{\mathcal{G}}(\rho) = \tau$ is defined and $w(\rho \tau c') = ww'$ where $c' = \tau(c)$. In all other cases, $f(w)$ is undefined. The proof that f is a winning strategy is the same as the other direction, with the roles of f and $f_{\mathcal{G}}$ as well as w and ρ swapped, since the definitions of compatibility and maximality are the same for the normalized synthesis and the parameterized vector games. □

Lemma 29. *For every parameterized vector game $\mathcal{G}$, there is a sentence $\varphi \in \mathrm{FO}[\sim]$ such that $Win(\mathcal{G}) = Win(\varphi)$.*

Proof. Let $\mathcal{G} = (A, B, \mathcal{F})$ be a parameterized vector game, and let $\mathcal{F} = \{\kappa_i \mid 1 \leq i \leq n\}$. As usual, let $L = \{0, \dots, B\}^A$. For all $\ell \in L$ and $1 \leq i \leq n$, if $\kappa_i(\ell) = (\bowtie^i_{\mathsf{s}} n^i_{\mathsf{s}}, \bowtie^i_{\mathsf{e}} n^i_{\mathsf{e}}, \bowtie^i_{\mathsf{se}} n^i_{\mathsf{se}})$, then for all $\theta \in \mathbb{T}$ we let:

$$\varphi_{i,\ell,\theta} = \exists^{\bowtie^i_\theta n^i_\theta} y.(\theta(y) \wedge \psi_{B,\ell}(y))$$

which is a FO[∼] formula and then we define:

$$\varphi = \bigvee_{i=1}^{n} \bigwedge_{\substack{\ell \in L \\ \theta \in \mathbb{T}}} \varphi_{i,\ell,\theta}$$

The proof that $\mathrm{Win}(\mathcal{G}) = \mathrm{Win}(\varphi)$ is similar to the one from Lemma 28. □

This ends the proof of Lemma 26, in other words, parameterized vector games are indeed equivalent to FO[∼] formula with respect to the synthesis problem. We use these games to study the synthesis problem when only mixed processes are involved $(\mathbb{0}, \mathbb{0}, \mathbb{N})$, and when all processes are either system or environment processes only $(\mathbb{N}, \mathbb{N}, \mathbb{0})$.

4.3.3 Cases of $(\mathbb{0},\mathbb{0},\mathbb{N})$ and $(\mathbb{N},\mathbb{N},\mathbb{0})$

The notion of cutoff from the synthesis problem can also be transposed to the parameterized vector games formalism: a triple $\mathbf{k} \in \mathbb{N}^{\mathbb{T}}$ is said to be a cutoff with respect to $(\mathcal{N}_{\mathsf{s}}, \mathcal{N}_{\mathsf{e}}, \mathcal{N}_{\mathsf{se}})$ if either $\mathbf{k}'$ is winning for all $\mathbf{k}' \geq \mathbf{k} \in \mathcal{N}_{\mathsf{s}} \times \mathcal{N}_{\mathsf{e}} \times \mathcal{N}_{\mathsf{se}}$, or $\mathbf{k}'$ is not winning for all $\mathbf{k}' \geq \mathbf{k} \in \mathcal{N}_{\mathsf{s}} \times \mathcal{N}_{\mathsf{e}} \times \mathcal{N}_{\mathsf{se}}$.

Remember that the notion of cutoff is helpful to know whether the existence of a winning strategy with respect to some parameters $(\mathcal{N}_{\mathsf{s}}, \mathcal{N}_{\mathsf{e}}, \mathcal{N}_{\mathsf{se}})$ is decidable. Indeed, if there is a cutoff, then it is enough to check for all triples below the cutoff (of which there are a finite number) the existence of a strategy for the game with this initial configuration (which is a finite acyclic game) which can be done in a finite amount of time. Then there is no need to check for triples above the cutoff due to its property. All in all, this means that proving that there is a cutoff implies that the game problem for $(\mathcal{N}_{\mathsf{s}}, \mathcal{N}_{\mathsf{e}}, \mathcal{N}_{\mathsf{se}})$ is decidable.

We show that there is no cutoff with respect to $(\mathbb{0},\mathbb{0},\mathbb{N})$ and $(\mathbb{N},\mathbb{N},\mathbb{0})$ by giving two games, one for each case, without cutoffs.

Lemma 30. *There is a game $\mathcal{G} = (A, B, \mathcal{F})$ such that Win($\mathcal{G}$) does not have a cutoff with respect to $(\mathbb{0},\mathbb{0},\mathbb{N})$.*

Proof. The idea is that System wins the game if there is an even number of tokens in the initial configuration. The acceptance conditions constrain System and Environment so that they have to move 2 tokens at the same time along a path from the initial location to the final location, and when the tokens reach the end of the path then the players have to do the same thing with 2 new tokens. This keeps going until either all tokens have been moved from the initial location to the final location, in which case System wins, or there is a single token remaining, in which case System has no more winning move.

We let $A_{\mathsf{s}} = \{a\}$ and $A_{\mathsf{e}} = \{b\}$, as well as $B = 2$. For $k \in \{0,1,2\}$, define the local acceptance conditions ${}^{=}k = (=0\,, =0\,, =k)$ and ${}^{\geq}k = (=0\,, =0\,, \geq k)$. Set $\ell_1 = \langle a \rangle, \ell_2 = \langle ab \rangle, \ell_3 = \langle a^2 b \rangle$, and $\ell_4 = \langle a^2 b^2 \rangle$. For $k_0, \ldots, k_4 \in \{0,1,2\}$ and $\bowtie_0, \ldots, \bowtie_4 \in \{=, \geq\}$, let $[{}^{\bowtie_0}k_0\,, {}^{\bowtie_1}k_1\,, {}^{\bowtie_2}k_2\,, {}^{\bowtie_3}k_3\,, {}^{\bowtie_4}k_4]$ denote $\kappa \in \mathfrak{C}^L$ where $\kappa(\ell_i) = ({}^{\bowtie_i}k_i)$ for all $i \in \{0,\ldots,4\}$ and $\kappa(\ell') = ({}^{=}0)$ for $\ell' \notin \{\ell_0, \ldots, \ell_4\}$. Finally,

$$\mathcal{F} = \left\{ \begin{array}{lll} [{}^{\geq}0\,, {}^{=}2\,, {}^{=}0\,, {}^{=}0\,, {}^{\geq}0], & [{}^{\geq}0\,, {}^{=}0\,, {}^{=}0\,, {}^{=}2\,, {}^{\geq}0], & [{}^{=}0\,, {}^{=}0\,, {}^{=}0\,, {}^{=}0\,, {}^{\geq}2], \\ [{}^{\geq}0\,, {}^{=}1\,, {}^{=}1\,, {}^{=}0\,, {}^{\geq}0], & [{}^{\geq}0\,, {}^{=}0\,, {}^{=}0\,, {}^{=}1\,, {}^{\geq}1] & \end{array} \right\} \cup K_{\mathsf{e}}$$

where $K_{\mathsf{e}} = \{\kappa_\ell \mid \ell \in L \text{ such that } \ell(b) > \ell(a)\}$ with $\kappa_\ell(\ell') = ({}^{\geq}1)$ if $\ell' = \ell$, and $\kappa_\ell(\ell') = ({}^{\geq}0)$ otherwise. This is illustrated in Figure 4.3.

There is a winning strategy for System from any initial configuration of size $2n$: Move two tokens from ℓ_0 to ℓ_1, wait until Environment sends them both to ℓ_2, then move them to ℓ_3, wait until they are moved to ℓ_4, then repeat with two new tokens from ℓ_0 until all the tokens are removed from ℓ_0, and Environment cannot escape $\mathcal{F}$ anymore.

However, there is no winning strategy for initial configurations of size $2n+1$ by induction on n. Indeed, it is easy to check that there is no possible move for System when $n = 0$, and when $n > 0$ then the only possible sequence of moves for System and Environment necessarily move 2 tokens from the initial state to the rightmost one, so we are left with $2n-1$ tokens in the initial locations which is losing for System by induction hypothesis. □

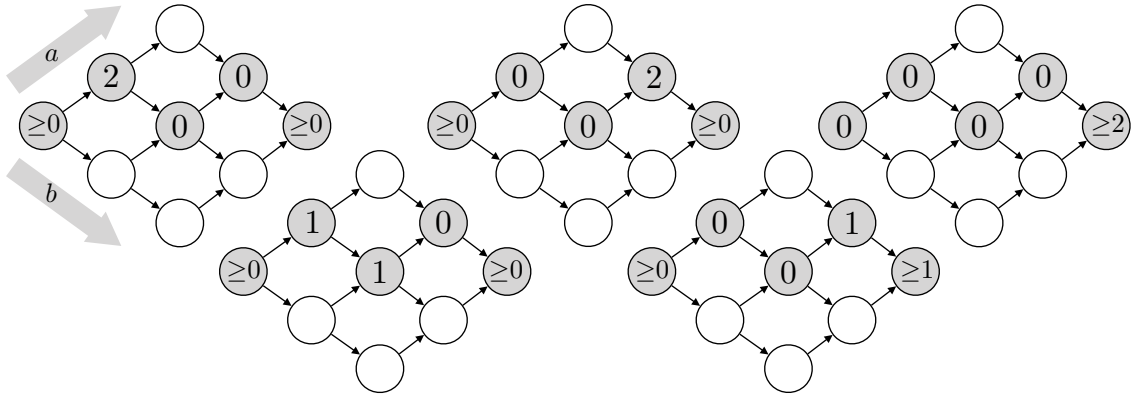

Figure 4.3: Acceptance conditions for a game with no cutoff with respect to $(\mathbb{0}, \mathbb{0}, \mathbb{N})$

Lemma 31. *There is a game $\mathcal{G} = (A, B, \mathcal{F})$ such that Win($\mathcal{G}$) does not have a cutoff with respect to $(\mathbb{N}, \mathbb{N}, \mathbb{0})$.*

Proof. We define $\mathcal{G}$ such that System wins only if she has at least as many processes as Environment. The idea is that System must play an a on a fresh process, then Environment does a b on another fresh process, System does another a on the earlier process, Environment does another b on its process, and then both players start again with two new processes. Any player that does not follow this loses, and System also wins if there are no tokens remaining in the initial location.

Formally, let $A_{\mathsf{s}} = \{a\}$, $A_{\mathsf{e}} = \{b\}$, and $B = 2$. As there are no shared processes, we can safely ignore locations with a letter from both System and Environment. We set $\mathcal{F} = \{\kappa_1, \kappa_2, \kappa_3, \kappa_4\}$ where

$$\begin{array}{lll} \kappa_1(\langle a \rangle) = (=1\,, =0\,, =0) & \kappa_2(\langle a \rangle) = (=1\,, =0\,, =0) & \kappa_3(\langle a \rangle) = (=0\,, =0\,, =0) \\ \kappa_1(\langle b \rangle) = (=0\,, =0\,, =0) & \kappa_2(\langle b \rangle) = (=0\,, \geq 2\,, =0) & \kappa_3(\langle b \rangle) = (=0\,, \geq 1\,, =0)\,, \end{array}$$

$\kappa_4(\ell_0) = (=0\,, =0\,, =0)$, and $\kappa_i(\ell') = (\geq 0\,, \geq 0\,, =0)$ for all other $\ell' \in L$ and $i \in \{1, 2, 3, 4\}$. Conditions κ_1 to κ_3 force players to follow the protocol explained earlier, and κ_4 ensures that System wins if all tokens are succesfully moved from the initial location. The strategy for System that follows this protocol wins as long as there are at least as many system tokens as there are environment tokens in the initial configuration, and conversely it is easy to check that no strategy wins when this is not the case. □

Note that the absence of a cutoff does not imply undecidability of the game problem. However, we prove that in the case of $(\mathbb{0}, \mathbb{0}, \mathbb{N})$, the game problem is actually undecidable.

Theorem 32. Game$(\mathbb{0}, \mathbb{0}, \mathbb{N})$ *and* Synth$(\text{FO}[\sim], \mathbb{0}, \mathbb{0}, \mathbb{N})$ *are undecidable.*

Proof. We once again provide a reduction from the halting problem for 2-counter machines (2CM) to Game$(\mathbb{0}, \mathbb{0}, \mathbb{N})$. Refer to the proof of Theorem 20 for a definition of 2-counter machines.

We fix a 2CM $M = (Q, T, q_0, q_h)$. Without loss of generality, let us assume that $q_0 \neq q_h$. Let $A_{\mathsf{s}} = Q \cup T \cup \{a_1, a_2\}$ and $A_{\mathsf{e}} = \{b\}$ with a_1, a_2, and b three fresh symbols. We consider the game $\mathcal{G} = (A, B, \mathcal{F})$ with $A = A_{\mathsf{s}} \uplus A_{\mathsf{e}}$, $B = 4$, and $\mathcal{F}$

defined below. Let $L = \{0, \ldots, B\}^A$ be the set of locations. Since there are only processes shared by System and Environment, we alleviate notation and consider that a configuration is simply a mapping $c : L \to \mathbb{N}$, instead of mapping each location to a triple $(k_{\mathsf{s}}, k_{\mathsf{e}}, k_{\mathsf{se}})$ whose first two components are always necessarily 0. From now on, to avoid confusion, we refer to configurations of the 2CM M as M-configurations, and to configurations of $\mathcal{G}$ as $\mathcal{G}$-configurations.

Intuitively, every valid run of M will be encoded as a play in $\mathcal{G}$, and the acceptance condition will enforce that, if a player in $\mathcal{G}$ deviates from a valid play, then she will lose immediately. At any point in the play, there will be at most one process with only a letter from Q played, which will represent the current state in the simulated 2CM run. Similarly, there will be at most one process with only a letter from T to represent what transition will be taken next. Finally, the value of counter c_i will be encoded by the number of processes with exactly two occurrences of a_i and two occurrences of b (i.e., $c(\langle a_i^2 b^2 \rangle)$).

To increase counter c_i, the players will move a new token to $\langle a_i^2 b^2 \rangle$, and to decrease it, they will move, together, a token from $\langle a_i^2 b^2 \rangle$ to $\langle a_i^4 b^4 \rangle$. Observe that, if c_i has value 0, then $c(\langle a_i^2 b^2 \rangle) = 0$ in the corresponding configuration of the game. As expected, it is then impossible to simulate the decrement of c_i. Environment's only role is to acknowledge System's actions by playing its (only) letter when System simulates a valid run. If System tries to cheat, she loses immediately.

Encoding an M-configuration. Let us be more formal. Suppose $\gamma = (q, \nu_1, \nu_2)$ is an M-configuration and c a $\mathcal{G}$-configuration. We say that c *encodes* γ if

- $c(\langle q \rangle) = 1$, $c(\langle a_1^2 b^2 \rangle) = \nu_1$, $c(\langle a_2^2 b^2 \rangle) = \nu_2$,
- $c(\ell) \geq 0$ for all $\ell \in \{\ell_0\} \cup \{\langle \hat{q}^2 b^2 \rangle, \langle t^2 b^2 \rangle, \langle a_i^4 b^4 \rangle \mid \hat{q} \in Q, t \in T, i \in \{1, 2\}\}$,
- $c(\ell) = 0$ for all other $\ell \in L$.

We then write $\gamma = \mathsf{m}(c)$. In simple terms, there must be one process in the location $\langle q \rangle$ corresponding to the state q of the 2CM, as many processes in $\langle a_i^2 b^2 \rangle$ as the valuation ν_i of counter c_i, and then the rest of processes are either in the initial location ℓ_0 or in states of the form $\langle \hat{q}^2 b^2 \rangle, \langle t^2 b^2 \rangle$, or $\langle a_i^4 b^4 \rangle$ which are "sink states" for processes previously used for the simulation but that are not needed anymore.

Let $\mathbb{C}(\gamma)$ be the set of $\mathcal{G}$-configurations c that encode γ. We say that a $\mathcal{G}$-configuration c is *valid* if $c \in \mathbb{C}(\gamma)$ for some γ.

Simulating a transition of M. Let us explain how we go from a $\mathcal{G}$-configuration encoding γ to a $\mathcal{G}$-configuration encoding a successor M-configuration γ'. Observe that System cannot change by herself the M-configuration encoded. If, for instance, she tries to change the current state q, she might move one process from ℓ_0 to $\langle q' \rangle$, but then the $\mathcal{G}$-configuration is not valid anymore. We need to move the process in $\langle q \rangle$ into $\langle q^2 b^2 \rangle$ and this requires the cooperation of Environment.

Assume that the game is in configuration C encoding $\gamma = (q, \nu_1, \nu_2)$. System will pick a transition t starting in state q, say, $t = (q, \mathsf{c}_1{+}{+}, q')$. From configuration c, System will go to the configuration c_1 defined by $c_1(\langle t \rangle) = 1$, $c_1(\langle a_1 \rangle) = 1$, and $c_1(\ell) = c(\ell)$ for all other $\ell \in L$.

If the transition t is correctly chosen, Environment will go to a configuration c_2 defined by $c_2(\langle q\rangle) = 0$, $c_2(\langle qb\rangle) = 1$, $c_2(\langle t\rangle) = 0$, $c_2(\langle tb\rangle) = 1$, $c_2(\langle a_1\rangle) = 0$, $c_2(\langle a_1 b\rangle) = 1$ and, for all other $\ell \in L$, $c_2(\ell) = c_1(\ell)$. This means that Environment moves processes in locations $\langle t\rangle$, $\langle q\rangle$, $\langle a_1\rangle$ to locations $\langle tb\rangle$, $\langle qb\rangle$, $\langle a_1 b\rangle$, respectively.

To finish the transition, System will now move a process to the destination state q' of t, and go to configuration c_3 defined by $c_3(\langle q'\rangle) = 1$, $c_3(\langle tb\rangle) = 0$, $c_3(\langle t^2 b\rangle) = 1$, $c_3(\langle qb\rangle) = 0$, $c_3(\langle q^2 b\rangle) = 1$, $c_3(\langle a_1 b\rangle) = 0$, $c_3(\langle a_1^2 b\rangle) = 1$, and $c_3(\ell) = c_2(\ell)$ for all other $\ell \in L$. In other words, System moves one process from ℓ_0 to the location $\langle q'\rangle$ corresponding to the destination state of t, and then move the only process in $\langle tb\rangle$, $\langle qb\rangle$, $\langle a_1 b\rangle$ to $\langle t^2 b\rangle$, $\langle q^2 b\rangle$, and $\langle a_i^2 b\rangle$ respectively.

Finally, Environment simply adds a b to the last three mentionned processes and moves to configuration c_4 given by $c_4(\langle t^2 b\rangle) = 0$, $c_4(\langle t^2 b^2\rangle) = c_3(\langle t^2 b^2\rangle) + 1$, $c_4(\langle q^2 b\rangle) = 0$, $c_4(\langle q^2 b^2\rangle) = c_3(\langle q^2 b^2\rangle) + 1$, $c_4(\langle a_1^2 b\rangle) = 0$, $c_4(\langle a_1^2 b^2\rangle) = c_3(\langle a_1^2 b^2\rangle) + 1$, and $c_4(\ell) = c_3(\ell)$ for all other $\ell \in L$. Observe that $c_4 \in \mathbb{C}((q', \nu_1 + 1, \nu_2))$.

Other types of transitions will be simulated similarly. To force System to start the simulation in γ_0, and not in any M-configuration, the configurations c such that $c(\langle q_0^2 b^2\rangle) = 0$ and $c(\langle q\rangle) = 1$ for $q \neq q_0$ are not valid, and will be losing for System.

Acceptance condition. It remains to define $\mathcal{F}$ in a way that enforces the above sequence of $\mathcal{G}$-configurations. Let

$$L_\checkmark = \{\ell_0\} \cup \{\langle a_i^2 b^2\rangle, \langle a_i^4 b^4\rangle \mid i \in \{1,2\}\} \cup \{\langle q^2 b^2\rangle \mid q \in Q\} \cup \{\langle t^2 b^2\rangle \mid t \in T\}$$

be the set of elements in L whose values do not affect the acceptance of the configuration. By $[\ell_1 \bowtie_1 n_1, \ldots, \ell_k \bowtie_k n_k]$, we denote $\kappa \in \mathfrak{C}^L$ such that $\kappa(\ell_i) = (\bowtie_i n_i)$ for $i \in \{1, \ldots, k\}$ and $\kappa(\ell) = (=0)$ for all $\ell \in L \setminus \{\ell_1, \ldots, \ell_k\}$. Moreover, for a set of locations $\hat{L} \subseteq L$, we let $\hat{L} \geq 0$ stand for "$(\ell \geq 0)$ for all $\ell \in \hat{L}$".

First, we force Environment to play only in response to System by making System win as soon as there is a process where Environment has played more letters than System. Let $L_{\mathsf{s}<\mathsf{e}} = \left\{\ell \in L \mid \left(\sum_{\alpha \in A_{\mathsf{s}}} \ell(\alpha)\right) < \ell(b)\right\}$. For all $\ell \in L_{\mathsf{s}<\mathsf{e}}$, we let

$$\kappa_\ell = [\ell \geq 1, (L \setminus \{\ell\}) \geq 0]$$

which is satisfied as long as at least one process is in location ℓ, and we let $\mathcal{F}_{\mathsf{s}<\mathsf{e}} = \bigcup_{\ell \in L_{\mathsf{s}<\mathsf{e}}} \kappa_\ell$.

If γ is not halting, the configurations in $\mathbb{C}(\gamma)$ will not be winning for System. Hence, System will have to move to win. We distinguish two cases: one for the very first transition taken, and another for all other transitions.

For all transitions $t = (q_0, op, q') \in T$ such that $op \in \{\mathsf{c}_i{+}{+}, \mathsf{c}_i{=}{=}0\}$, we let

$$\kappa_t = \begin{cases} [\langle q_0\rangle = 1, \langle t\rangle = 1, \langle a_i\rangle = 1, \ell_0 \geq 0] & \text{if } op = \mathsf{c}_i{+}{+} \\ [\langle q_0\rangle = 1, \langle t\rangle = 1, \ell_0 \geq 0] & \text{if } op = \mathsf{c}_i{=}{=}0 \end{cases}$$

and $\mathcal{F}_0 = \bigcup_{t=(q_0,op,q') \in T} \kappa_t$. Note that to satisfy this condition, there can be no process in any state of $L_\checkmark$, ensuring that this can only be satisfied at the very beginning of the simulation.

For all $t = (q, op, q') \in T$, we let

$$\kappa^{\hat{q}}_{(q,t)} = \begin{cases} [\langle q\rangle = 1, \langle t\rangle = 1, \langle a_i\rangle = 1, \langle \hat{q}^2 b^2\rangle \geq 1, (L_\checkmark \setminus \{\langle \hat{q}^2 b^2\rangle\}) \geq 0] & \text{if } op = \mathsf{c}_i{+}{+} \\ [\langle q\rangle = 1, \langle t\rangle = 1, \langle a_i^3 b^2\rangle = 1, \langle \hat{q}^2 b^2\rangle \geq 1, (L_\checkmark \setminus \{\langle \hat{q}^2 b^2\rangle\}) \geq 0] & \text{if } op = \mathsf{c}_i{-}{-} \\ [\langle q\rangle = 1, \langle t\rangle = 1, \langle a_i^2 b^2\rangle = 0, \langle \hat{q}^2 b^2\rangle \geq 1, (L_\checkmark \setminus \{\langle \hat{q}^2 b^2\rangle, \langle a_i^2 b^2\rangle\}) \geq 0] & \text{if } op = \mathsf{c}_i{=}{=}0 \end{cases}$$

and $\mathcal{F}_{(q,t)} = \bigcup_{\hat{q}\in Q} \kappa^{\hat{q}}_{(q,t)}$.

Whether the transition was the first transition of the run or not, Environment then needs to reply by adding a b on all three (respectively two if the transitions was a zero-test) processes on which System performed an action. Let $t = (q, op, q') \in T$.

- If $op = \mathsf{c}_i{+}{+}$, let $L_1 = \{\langle q\rangle, \langle qb\rangle\}$, $L_2 = \{\langle t\rangle, \langle tb\rangle\}$, $L_3 = \{\langle a_i\rangle, \langle a_i b\rangle\}$, and $L^3_X = \{(\langle q\rangle, \langle t\rangle, \langle a_i\rangle), (\langle qb\rangle, \langle tb\rangle, \langle a_i b\rangle)\}$. We let

$$\mathcal{F}^{\mathsf{e}}_{(q,t)} = \left\{ [\ell_1 = 1, \ell_2 = 1, \ell_3 = 1, L_\checkmark \geq 0] \mid \begin{array}{c} \ell_1 \in L_1, \ell_2 \in L_2, \ell_3 \in L_3 \\ \text{and } (\ell_1, \ell_2, \ell_3) \notin L^3_X \end{array} \right\}$$

  This means that $\mathcal{F}^{\mathsf{e}}_{(q,t)}$ is *not* satisfied if either Environment does nothing on all three processes mentionned earlier, or if Environment performs a b on all three. This will ensure that Environment has no choice but to play a b on all three processes.

- Similarly if $op = \mathsf{c}_i{-}{-}$, let $L_1 = \{\langle q\rangle, \langle qb\rangle\}$, $L_2 = \{\langle t\rangle, \langle tb\rangle\}$, $L_3 = \{\langle a_i^2 b^2\rangle, \langle a_i^3 b^2\rangle\}$, and $L^3_X = \{(\langle q\rangle, \langle t\rangle, \langle a_i^2 b^2\rangle), (\langle qb\rangle, \langle tb\rangle, \langle a_i^3 b^2\rangle)\}$. Then

$$\mathcal{F}^{\mathsf{e}}_{(q,t)} = \left\{ [\ell_1 = 1, \ell_2 = 1, \ell_3 = 1, L_\checkmark \geq 0] \mid \begin{array}{c} \ell_1 \in L_1, \ell_2 \in L_2, \ell_3 \in L_3 \\ \text{and } (\ell_1, \ell_2, \ell_3) \notin L^3_X \end{array} \right\}$$

- Finally if $op = \mathsf{c}_i{=}{=}0$, let $L_1 = \{\langle q\rangle, \langle qb\rangle\}$, $L_2 = \{\langle t\rangle, \langle tb\rangle\}$, $L^2_X = \{(\langle q\rangle, \langle t\rangle), (\langle qb\rangle, \langle tb\rangle)\}$, and then

$$\mathcal{F}^{\mathsf{e}}_{(q,t)} = \left\{ [\ell_1 = 1, \ell_2 = 1, L_\checkmark \geq 0] \mid \ell_1 \in L_1, \ell_2 \in L_2 \text{ and } (\ell_1, \ell_2) \notin L^2_X \right\}$$

For the next step, System must perform the same action on the three (resp. two) processes, as well as move a process from ℓ_0 to $\langle q'\rangle$ corresponding to the ending state of transition t. For all $t = (q, op, q') \in T$, let

$$\mathcal{F}_{(q,t,q')} = \begin{cases} \{[\langle q^2 b\rangle = 1, \langle t^2 b\rangle = 1, \langle a_i^2 b\rangle = 1, \langle q'\rangle = 1, L_\checkmark \geq 0]\} & \text{if } op = \mathsf{c}_i{+}{+} \\ \{[\langle q^2 b\rangle = 1, \langle t^2 b\rangle = 1, \langle a_i^4 b^3\rangle = 1, \langle q'\rangle = 1, L_\checkmark \geq 0]\} & \text{if } op = \mathsf{c}_i{-}{-} \\ \{[\langle q^2 b\rangle = 1, \langle t^2 b\rangle = 1, \langle q'\rangle = 1, L_\checkmark \geq 0]\} & \text{if } op = \mathsf{c}_i{=}{=}0 \end{cases}$$

For the final step, Environment must again perform a b on the three (resp. two) previous processes, and must *not* do the same on the process in location $\langle q'\rangle$. Given a transition $t = (q, op, q')$, we let

$$\kappa^t_{\neg\langle q'b\rangle} = \begin{cases} [\langle q'b\rangle = 1, L_\checkmark \cup \{\langle q^2 b\rangle, \langle t^2 b\rangle, \langle a_i^2 b\rangle\} \geq 0] & \text{if } op = \mathsf{c}_i{+}{+} \\ [\langle q'b\rangle = 1, L_\checkmark \cup \{\langle q^2 b\rangle, \langle t^2 b\rangle, \langle a_i^4 b^3\rangle\} \geq 0] & \text{if } op = \mathsf{c}_i{-}{-} \\ [\langle q'b\rangle = 1, L_\checkmark \cup \{\langle q^2 b\rangle, \langle t^2 b\rangle\} \geq 0] & \text{if } op = \mathsf{c}_i{=}{=}0 \end{cases}$$

which forces Environment not to play a b on the process in q'. Then

- if $op = \mathsf{c}_i{+}{+}$, we let

$$\mathcal{F}^{\mathsf{e}}_{(q,t,q')} = \kappa^t_{\neg\langle q'b\rangle} \cup \left\{ \begin{array}{l} [\langle q'\rangle = 1, \langle q^2 b\rangle = 1, \langle t^2 b\rangle \geq 0, \langle a_i^2 b\rangle \geq 0, L_\checkmark \geq 0], \\ [\langle q'\rangle = 1, \langle q^2 b\rangle \geq 0, \langle t^2 b\rangle = 1, \langle a_i^2 b\rangle \geq 0, L_\checkmark \geq 0], \\ [\langle q'\rangle = 1, \langle q^2 b\rangle \geq 0, \langle t^2 b\rangle \geq 0, \langle a_i^2 b\rangle = 1, L_\checkmark \geq 0] \end{array} \right\}$$

- if $op = \mathsf{c}_i{--}$, we let

$$\mathcal{F}^{\mathsf{e}}_{(q,t,q')} = \kappa^t_{\neg\langle q'b\rangle} \cup \left\{ \begin{array}{l} [\langle q'\rangle = 1, \langle q^2b\rangle = 1, \langle t^2b\rangle \geq 0, \langle a_i^4b^3\rangle \geq 0, L_\checkmark \geq 0], \\ [\langle q'\rangle = 1, \langle q^2b\rangle \geq 0, \langle t^2b\rangle = 1, \langle a_i^4b^3\rangle \geq 0, L_\checkmark \geq 0], \\ [\langle q'\rangle = 1, \langle q^2b\rangle \geq 0, \langle t^2b\rangle \geq 0, \langle a_i^4b^3\rangle = 1, L_\checkmark \geq 0] \end{array} \right\}$$

- if $op = \mathsf{c}_i{==}0$, we let

$$\mathcal{F}^{\mathsf{e}}_{(q,t,q')} = \kappa^t_{\neg\langle q'b\rangle} \cup \left\{ \begin{array}{l} [\langle q'\rangle = 1, \langle q^2b\rangle = 1, \langle t^2b\rangle \geq 0, L_\checkmark \geq 0], \\ [\langle q'\rangle = 1, \langle q^2b\rangle \geq 0, \langle t^2b\rangle = 1, L_\checkmark \geq 0] \end{array} \right\}$$

Then after those 4 steps the new $\mathcal{G}$-configuration is valid and encodes the t-successor of the previous M-configuration.

The last remaining part states that if the winning state q_h is reached, then System can win by playing another q_h on the process in location $\langle q_h\rangle$. From there, all $\mathcal{G}$-configurations reachable by Environment will still be winning for System. For all $t \in T \setminus \{q_h\}$, $\ell \in \{\langle q_h^2\rangle, \langle q_h^2b\rangle, \langle q_h^2b^2\rangle\}$, let

$$\kappa_{(t,\ell)} = [\ell = 1, \langle t^2b^2\rangle \geq 1, L_\checkmark \setminus \{\langle t^2b^2\rangle\} \geq 0]$$

which states that there is one process with two q_h and some number of b, and at least another process in location $\langle t^2b^2\rangle$ for some transition t (which ensures that at least one transition has been simulated). Then we let $\mathcal{F}_{q_h} = \bigcup_{t,\ell} \kappa_{(t,\ell)}$.

Putting all those parts together, the acceptance condition of $\mathcal{G}$ is

$$\mathcal{F} = \mathcal{F}_{\mathsf{s}<\mathsf{e}} \cup \mathcal{F}_0 \cup \bigcup_{t=(q,op,q')\in T} \left(\mathcal{F}_{(q,t)} \cup \mathcal{F}^{\mathsf{e}}_{(q,t)} \cup \mathcal{F}_{(q,t,q')} \cup \mathcal{F}^{\mathsf{e}}_{(q,t,q')}\right) \cup \mathcal{F}_{q_h}$$

Note that a correct play can end in three different ways: either there is a process in $\langle q_h\rangle$ and System moves it to $\langle q_h^2\rangle$, or System has no transition to pick, or there are not enough processes in ℓ_0 for System to simulate a new transition. Only the first kind is winning for System.

We now show that there is an accepting run in M if and only if there is some $k \in \mathbb{N}$ such that $c_{(0,0,k)}$ is winning for System.

$\boxed{\Rightarrow}$ Suppose there is an accepting run $\rho : (q_0, 0, 0) \vdash_{t_1} (q_1, \nu_1^1, \nu_2^1) \vdash_{t_2} \cdots \vdash_{t_n} (q_n, \nu_1^n, \nu_2^n)$ with $q_n = q_h$ for M, and fix some $k \geq 3n+1$. Without loss of generality, we can assume that the configurations $\gamma_0, \ldots, \gamma_n$ visited in ρ are pairwise different. The memoryless strategy f for System that faithfully simulates ρ is formally defined as follows. In the following, a transition $(q, op, q') \in T$ is written $q \xrightarrow{op} q'$.

Initialization. Let c_0 be the initial $\mathcal{G}$-configuration.

- If $t_1 = q_0 \xrightarrow{\mathsf{c}_i{+}{+}} q_1$, we let τ_1 be defined by $\tau_1(\ell_0, \langle q_0\rangle) = 1$, $\tau_1(\ell_0, \langle t_1\rangle) = 1$, $\tau_1(\ell_0, \langle a_i\rangle) = 1$, and $\tau_1(\ell, \ell') = 0$ for all other $\ell, \ell' \in L$.

- If $t_1 = q_0 \xrightarrow{\mathsf{c}_i{=}{=}0} q_1$, we let τ_1 be defined by $\tau_1(\ell_0, \langle q_0\rangle) = 1$, $\tau_1(\ell_0, \langle t_1\rangle) = 1$, and $\tau_1(\ell, \ell') = 0$ for all other $\ell, \ell' \in L$.

We then let $f(c_0) = \tau_1$.

Simulation of a new transition. For $0 < j < n$, for any k-configuration $c \in \mathbb{C}((q_j, \nu_1^j, \nu_2^j))$, we let $f(c) = \tau_{j+1}$, with τ_{j+1} defined as follows.

- If $t_{j+1} = q_j \xrightarrow{\mathsf{c}_i++} q_{j+1}$, then $\tau_{j+1}(\ell_0, \langle t_{j+1}\rangle) = 1$, $\tau_{j+1}(\ell_0, \langle a_i\rangle) = 1$, and $\tau_{j+1}(\ell, \ell') = 0$ for all other $\ell, \ell' \in L$.
- If $t_{j+1} = q_j \xrightarrow{\mathsf{c}_i--} q_{j+1}$, then $\tau_{j+1}(\ell_0, \langle t_{j+1}\rangle) = 1$, $\tau_{j+1}(\langle a_i^2 b^2\rangle, \langle a_i^3 b^2\rangle) = 1$ and $\tau_{j+1}(\ell, \ell') = 0$ for all other $\ell, \ell' \in L$.
- If $t_{j+1} = q_j \xrightarrow{\mathsf{c}_i==0} q_{j+1}$, then $\tau_{j+1}(\ell_0, \langle t_{j+1}\rangle) = 1$, and $\tau_{j+1}(\ell, \ell') = 0$ for all other $\ell, \ell' \in L$.

Note that if $c \in \mathbb{C}((q_j, \nu_1^j, \nu_2^j))$ then $c \notin \mathbb{C}((q_{j'}, \nu_1^{j'}, \nu_2^{j'}))$ for $j' \neq j$ as we assumed that $\gamma_j \neq \gamma_{j'}$, therefore $f(c)$ is well-defined.

Second step of the simulation of a transition. Let c be a k-configuration such that $c(\langle qb\rangle) = 1$ for some $q \in Q$, $c(\langle tb\rangle) = 1$ for some $t \in T$, $c(\langle a_i^2 b^2\rangle) \geq 0$, $c(\langle a_i^4 b^4\rangle) \geq 0$ for $i = 1, 2$, $c(\langle t^2 b^2\rangle) \geq 0$ for all $t \in T$, $c(\langle q^2 b^2\rangle) \geq 0$ for all $q \in Q$, $c(\ell_0) > 0$ and $c(\ell) = 0$ for all other $\ell \in L$. We define $f(c) = \tau$ with τ defined as follows.

- If $t = q \xrightarrow{c_i++} q'$ and c is such that $c(\langle a_i b\rangle) = 1$. Then $\tau(\langle qb\rangle, \langle q^2 b\rangle) = 1$, $\tau(\langle tb\rangle, \langle t^2 b\rangle) = 1$, $\tau(\langle a_i b\rangle, \langle a_i^2 b\rangle) = 1$, $\tau(\ell_0, \langle q'\rangle) = 1$ and $\tau(\ell, \ell') = 0$ for all other $\ell, \ell' \in L$.
- If $t = q \xrightarrow{c_i--} q'$ c is such that $c(\langle a_i^3 b^3\rangle) = 1$, then τ is defined by $\tau(\langle qb\rangle, \langle q^2 b\rangle) = 1$, $\tau(\langle tb\rangle, \langle t^2 b\rangle) = 1$, $\tau(\langle a_i^3 b^3\rangle, \langle a_i^4 b^3\rangle) = 1$, $\tau(\ell_0, \langle q'\rangle) = 1$ and $\tau(\ell, \ell') = 0$ for all other $\ell, \ell' \in L$.
- If $t = q \xrightarrow{\mathsf{c}_i==0} q'$, $\tau(\langle qb\rangle, \langle q^2 b\rangle) = 1$, $\tau(\langle tb\rangle, \langle t^2 b\rangle) = 1$, $\tau(\ell_0, \langle q'\rangle) = 1$ and $\tau(\ell, \ell') = 0$ for all other $\ell, \ell' \in L$.

Other cases. If ρ is a partial play ending in $c \in \mathbb{C}(\gamma_n)$, then $f(c)$ is the update function τ such that $\tau(\langle q_n\rangle, \langle q_n^2\rangle) = 1$ and $\tau(\ell, \ell') = 0$ for all other $\ell, \ell' \in L$.

And for any other configuration, f is undefined.

We show that f is winning by contradiction: suppose there is a winning strategy f_{e} for Environment. Let $\pi = c_0 \tau_0' c_0' \tau_1 c_1 \dots$ be the maximal play compatible with f and f_{e}. We show the following by recursion:

Lemma 33. *For all $0 < j \leq n$, $c_{2j} \in \mathbb{C}(\gamma_j)$ and $c_{2j}(\ell_0) \geq k - (3j + 1)$*

Intuitively, this lemma states that ρ correctly simulates ρ and that there are always enough process in ℓ_0 for System to do his transitions.

Proof. We prove it by induction on j.

Base step ($c_2 \in \mathbb{C}(\gamma_1)$): $c_0 = c_{(0,0,k)}$ is the initial configuration. Suppose that $t_1 = q_0 \xrightarrow{\mathsf{c}_1++} q_1$, then by definition of f, $c_0'(\langle q_0\rangle) = 1$, $c_0'(\langle t_1\rangle) = 1$, $c_0'(\langle a_1\rangle) = 1$, $c'(\ell_0) = k - 3 > 1$, and $c_0'(\ell) = 0$ for all other $\ell \in L$. Since $c_0' \models \kappa_{t_1}$ and thus $c_0' \models \mathcal{F}_0$, $f_{\mathsf{e}}(c_0') = \tau_1$ is defined, otherwise f_{e} is not winning. Then,

- If $\tau_1(\ell_0, \langle b^m\rangle) \geq 1$ for $m \geq 1$, then $c_1 \models \mathcal{F}_{\mathsf{s}<\mathsf{e}}$.

- If $\tau_1(\langle q_0\rangle, \langle q_0 b^m\rangle) = 1$ or $\tau_1(\langle t_1\rangle, \langle t_1 b^m\rangle) = 1$ or $\tau_1(\langle a_1\rangle, \langle a_1 b^m\rangle) = 1$ for some $m > 1$, then $c_1 \models \mathcal{F}_{\mathsf{s}<\mathsf{e}}$ too.
- Else if $\tau_1(\langle q_0\rangle, \langle q_0 b\rangle) = 0$, or if $\tau_1(\langle t_1\rangle, \langle t_1 b\rangle) = 0$ or if $\tau_1(\langle a_1\rangle, \langle a_1 b\rangle) = 0$, then $c_1 \models \mathcal{F}^{\mathsf{e}}_{(q_0,t_1)}$.

Hence $\tau_1(\langle q_0\rangle, \langle q_0 b\rangle) = \tau_1(\langle t_1\rangle, \langle t_1 b\rangle) = \tau_1(\langle a_1\rangle, \langle a_1 b\rangle) = 1$, and for all other $\ell \in L$, $\tau_1(\ell_0, \ell) = 0$ and $c_1(\langle q_0 b\rangle) = c_1(\langle t_1 b\rangle) = c_1(\langle a_1 b\rangle) = 1$, $c_1(\ell_0) = k - 3$ and $c_1(\ell) = 0$ for all other $\ell \in L$.

Following the definition of f, $c'_1(\langle q_0^2 b\rangle) = c'_1(\langle t_1^2 b\rangle) = c'_1(\langle a_1^2 b\rangle) = c'_1(\langle q_1\rangle) = 1$, $c'_1(\ell_0) = k - 4 > 0$, and $c'_1(\ell) = 0$ for all other $\ell \in L$. Since, $c'_1 \models \mathcal{F}_{(q_0,t_1,q_1)}$, $f_{\mathsf{e}} = \tau_2$ is defined.

Again we look at all possible transitions for Environment:

- As before, if $\tau_2(\ell_0, \langle b^m\rangle) \geq 1$ for $m \geq 1$, then $c_2 \models \mathcal{F}_{\mathsf{s}<\mathsf{e}}$.
- If $\tau_2(\langle q_0^2 b\rangle, \langle q_0^2 b^m\rangle) = 1$ or $\tau_2(\langle t_1^2 b\rangle, \langle t_1^2 b^m\rangle) = 1$ or $\tau_2(\langle a_1^2 b\rangle, \langle a_1^2 b^m\rangle) = 1$, for $m > 2$, then $c_2 \models \mathcal{F}_{\mathsf{s}<\mathsf{e}}$.
- If $\tau_2(\langle q_0^2 b\rangle, \langle q_0^2 b^2\rangle) = 0$, or if $\tau_2(\langle t_1^2 b\rangle, \langle t_1^2 b^2\rangle) = 0$ or if $\tau_2(\langle a_1^2 b\rangle, \langle a_1^2 b^2\rangle) = 0$, then $c_2 \models \mathcal{F}^{\mathsf{e}}_{(q_0,t_1,q_1)}$.
- Finally, if $\tau_2(\langle q_1\rangle, \langle q_1 b\rangle) = 1$ then $c_2 \models \mathcal{F}^{\mathsf{e}}_{(q_0,t_1,q_1)}$ and if $\tau_2(\langle q_1\rangle, \langle q_1 b^m\rangle) = 1$, for $m > 2$, then $c_2 \models \mathcal{F}_{\mathsf{s}<\mathsf{e}}$.

Thus, necessarily, $c_2(\langle q_0^2 b^2\rangle) = c_2(\langle t_1^2 b^2\rangle) = c_2(\langle a_1^2 b^2\rangle) = c_2(\langle q_1\rangle) = 1$, $c_2(\ell_0) = k - 4$, and $c_2(\ell') = 0$ for all other $\ell' \in L$. Hence, $c_2 \in \mathbb{C}(q_1, 1, 0)$, and $c_2(\ell_0) \geq k-(3*1+1) = k - 4$.

If $t_1 = q_0 \xrightarrow{c_2++} q_1$, the proof is identical, but with a_2 replacing a_1. If now $t_1 = q_0 \xrightarrow{c_i==0} q_1$, the proof goes along the same lines, without difficulty.

Induction step: Let $0 < j < n$ and $\gamma_j = (q_j, \nu_1^j, \nu_2^j)$, and suppose that $c_{2j} \in \mathbb{C}(\gamma_j)$ and $c_{2j}(\ell_0) \geq k - (3j + 1) \geq 3$. There are six cases depending on the type of t_{j+1}. Without loss of generality, we consider here only the three cases involving c_1.

▶ If $t_{j+1} = q_j \xrightarrow{\mathsf{c}_1++} q_{j+1}$ then $\gamma_{j+1} = (q_{j+1}, \nu_1^j + 1, \nu_2^j)$. Following f, we obtain that $c'_{2j}(\langle t_{j+1}\rangle) = 1$, $c'_{2j}(\langle a_1\rangle) = 1$, $c'_{2j}(\ell_0) = c_{2j}(\ell_0) - 2$ and $c'_{2j}(\ell) = (c_{2j})(\ell)$ for all other $\ell \in L$.

With the same arguments as in the base case, the only possibility is that $f_{\mathsf{e}}(c'_{2j}) = \tau_{2j+1}$ with $\tau_{2j+1}(\langle q_j\rangle, \langle q_j b\rangle) = 1$, $\tau_{2j+1}(\langle t_{j+1}\rangle, \langle t_{j+1} b\rangle) = 1$, $\tau_{2j+1}(\langle a_1\rangle, \langle a_1 b\rangle) = 1$ and $\tau_{2j+1}(\ell, \ell') = 0$ for all other $\ell, \ell' \in L$. This yields the configuration $c_{2j+1}(\langle q_j b\rangle) = 1$, $c_{2j+1}(\langle a_1 b\rangle) = 1$, $c_{2j+1}(\langle t_{j+1} b\rangle) = 1$, $c_{2j+1}(\langle q_j\rangle) = c_{2j+1}(\langle a_1\rangle) = c_{2j+1}(\langle t_{j+1}\rangle) = 0$ and $c_{2j+1}(\ell) = c'_{2j}(\ell)$ for all other $\ell \in L$.

By definition of f, the action of System leads to c'_{2j+1} defined by $c'_{2j+1}(\langle q_j^2 b\rangle) = c'_{2j+1}(\langle t_{j+1}^2 b\rangle) = c'_{2j+1}(\langle a_1^2 b\rangle = c'_{2j+1}(\langle q_{j+1}\rangle) = 1$, $c'_{2j+1}(\langle q_j b\rangle) = c'_{2j+1}(\langle t_{j+1} b\rangle) = c'_{2j+1}(\langle a_1 b\rangle = 0$, $c'_{2j+1}(\ell_0) = c_{2j+1}(\ell_0) - 1 = c_{2j}(\ell_0) - 3$, and $c'_{2j+1}(\ell) = 0$ for all other $\ell \in L$.

Finally, again as in the base case, we necessarily have that $f_{\mathsf{e}} = \tau_{2j+2}$ such that $\tau_{2j+2}(\langle q_j^2 b\rangle, \langle q_j^2 b^2\rangle) = 1$, $\tau_{2j+2}(\langle t_{j+1}^2 b\rangle, \langle t_{j+1}^2 b^2\rangle) = 1$, $\tau_{2j+2}(\langle a_1^2 b\rangle, \langle a_1^2 b^2\rangle) = 1$. Hence, $c_{2j+2}(\langle q_j^2 b^2\rangle) = c_{2j}(\langle q_j^2 b^2\rangle) + 1$, $c_{2j+2}(\langle t_j^2 b^2\rangle) = c_{2j}(\langle t_j^2 b^2\rangle) + 1$, $c_{2j+2}(\langle a_1^2 b^2\rangle) = c_{2j}(\langle a_1^2 b^2\rangle) + 1$, $c_{2j+2}(\langle q_{j+1}\rangle) = 1$, $c_{2j+2}(\ell_0) = c_{2j}(\ell_0) - 3$ and $c_{2j+1}(\ell) = c_{2j}(\ell)$ for all other $\ell \in L$.

Since $c_{2j} \in \mathbb{C}(\gamma_j)$ and $\gamma_{j+1} = (q_{j+1}, \nu_1^j + 1, \nu_2^j)$ it is easy to verify that c_{2j+2} is indeed in $\mathbb{C}(\gamma_{j+1})$. Moreover, $c_{2j+2}(\ell_0) = c_{2j}(\ell_0) - 3 \geq k - (3(j+1)+1)$.

▶ If $t_{j+1} = q_j \xrightarrow{\mathsf{c}_1--} q_{j+1}$, then we know that $\nu_1^j \geq 1$. Since $c_{2j} \in \mathbb{C}(\gamma_j)$, we deduce that $c_{2j}(\langle a_1^2 b^2 \rangle) \geq 1$. Following f, $c'_{2j}(\langle t_{j+1} \rangle) = c'_{2j}(\langle a_1^3 b^2 \rangle) = 1$, $c'_{2j}(\ell_0) = c_{2j}(\ell_0) - 1$, $c'_{2j}(\langle a_1^2 b^2 \rangle) = c_{2j}(\langle a_1^2 b^2 \rangle) - 1$ and $c'_{2j}(\ell) = c_{2j}(\ell)$ for all other $\ell \in L$.

To avoid reaching configurations in $\mathcal{F}_{\mathsf{s<e}}$ or in $\mathcal{F}^{\mathsf{e}}_{(q_j, t_{j+1})}$, Environment necessarily updates the configuration to $c_{2j+1}(\langle t_{j+1} b \rangle) = c_{2j+1}(\langle a_1^3 b^3 \rangle) = c_{2j+1}(\langle q_j b \rangle) = 1$, and $c_{2j+1}(\ell) = c'_{2j}(\ell)$ for all other $\ell \in L$.

Again, the strategy defined for System leads to the configuration $c'_{2j+1}(\langle q_j^2 b \rangle) = c'_{2j+1}(\langle t_{j+1}^2 b \rangle) = c'_{2j+1}(\langle a_1^4 b^3 \rangle) = c'_{2j+1}(\langle q_{j+1} \rangle) = 1$, $c'_{2j+1}(\langle t_{j+1} b \rangle) = c'_{2j+1}(\langle a_1^3 b^3 \rangle) = c'_{2j+1}(\langle q_j b \rangle) = 0$, $c'_{2j+1}(\ell_0) = c_{2j+1}(\ell_0) - 1$, and $c'_{2j+1}(\ell) = c_{2j+1}(\ell)$ for all other $\ell \in L$.

Finally, the only possible move for Environment is $c_{2j+2}(\langle q_j^2 b^2 \rangle) = c'_{2j+1}(\langle q_j^2 b^2 \rangle) + 1$, $c_{2j+2}(\langle t_{j+1}^2 b^2 \rangle) = c'_{2j+1}(\langle t_{j+1}^2 b^2 \rangle) + 1$, $c_{2j+2}(\langle a_1^4 b^4 \rangle) = c'_{2j+1}(\langle a_1^4 b^4 \rangle) + 1$, $c_{2j+2}(\langle q_j^2 b \rangle) = c_{2j+2}(\langle t_{j+1}^2 b \rangle) = c_{2j+2}(\langle a_1^4 b^3 \rangle) = 0$ and $c_{2j+2}(\ell) = c'_{2j+1}(\ell)$ for all other $\ell \in L$.

From this, we deduce that $c_{2j+2}(\langle q_{j+1} \rangle) = c'_{2j}(\langle q_{j+1} \rangle) = 1$, $c_{2j+2}(\langle a_2^2 b^2 \rangle) = c_{2j}(\langle a_2^2 b^2 \rangle)$, and $c_{2j+2}(\langle a_1^2 b^1 \rangle) = c_{2j}(\langle a_1^2 b^2 \rangle) - 1$. Moreover, we have that $c_{2j+2}(\langle q_j^2 b^2 \rangle) \geq 0$, $c_{2j+2}(\langle t_{j+1}^2 b^2 \rangle) \geq 0$, $c_{2j+2}(\langle a^4 b^4 \rangle) \geq 0$, $c_{2j+2}(\ell_0) = c_{2j}(\ell_0) - 2 \geq 0$ by induction hypothesis, and for all other $\ell \in L$, $c_{2j+2}(\ell) = c_{2j}(\ell)$. Since $c_{2j} \in \mathbb{C}(\gamma_j)$, this implies that $c_{2j+2} \in \mathbb{C}(\gamma_{j+1})$, as expected. Also, $c_{2j+2}(\ell_0) \geq k - (3j+1) - 2 \geq k - (3(j+1)+1)$.

▶ If $t_{j+1} = q_j \xrightarrow{\mathsf{c}_1==0} q_{j+1}$, then $\nu_1^j = 0 = c_{2j}(a_1^2 b^2)$. The proof goes along the same lines as before. Now the sequence of configurations is necessarily: $c'_{2j}(\langle t_{j+1} \rangle) = 1$, $c'_{2j}(\ell_0) = c_{2j}(\ell_0) - 1$, and $c'_{2j}(\ell) = c_{2j}(\ell)$ for all other $\ell \in L$.

Then $c_{2j+1}(\langle t_{j+1} b \rangle) = c_{2j+1}(\langle q_j b \rangle) = 1$, $c_{2j+1}(\langle t_{j+1} \rangle) = c_{2j+1}(\langle q_j \rangle) = 0$, and $c_{2j+1}(\ell) = c'_{2j}(\ell)$ for all other $\ell \in L$.

Then we have $c'_{2j+1}(\langle q_{j+1} \rangle) = c'_{2j+1}(\langle t_{j+1}^2 b \rangle) = c'_{2j+1}(\langle q_j^2 b \rangle) = 1$, $c'_{2j+1}(\langle t_{j+1} b \rangle) = c'_{2j+1}(\langle q_j b \rangle) = 0$, $c'_{2j+1}(\ell_0) = c_{2j+1}(\ell_0) - 1$, and $c'_{2j+1}(\ell) = c_{2j+1}(\ell)$ for all other $\ell \in L$.

Finally, $c_{2j+2}(\langle t_{j+1}^2 b^2 \rangle) = c'_{2j+1}(\langle t_{j+1}^2 b^2 \rangle) + 1$, $c_{2j+2}(\langle q_j^2 b^2 \rangle) = c'_{2j+1}(\langle q_j^2 b^2 \rangle) + 1$, $c_{2j+2}(\langle t_{j+1}^2 b \rangle) = c_{2j+2}(\langle q_j^2 b \rangle) = 0$, and $c_{2j+2}(\ell) = c'_{2j+1}(\ell)$ for all other $\ell \in L$.

One can check that $c_{2j+1}(\ell_0) = c_{2j}(\ell_0) - 2 \geq k - (3j+1) - 2 \geq k - (3(j+1)+1) \geq 0$ and that $c_{2j+2} \in \mathbb{C}(\gamma_{j+1})$. □

With this lemma, we know that c_{2n} exists and $c_{2n} \in \mathbb{C}(\gamma_n)$. By definition of f, $c'_{2n}(\langle q_n^2 \rangle) = 1$, $c'_{2n}(\langle q_n \rangle) = 0$ and, for all other $\ell \in L$, $c'_{2n}(\ell) = c_{2n}(\ell)$. But this time, there are no more possible transition for Environment:

- Moving the process in $\langle q_n^2 \rangle$ to $\langle q_n^2 b^m \rangle$ for some $m \geq 1$ leads to a configuration either in $\mathcal{F}_{q_h}$ or in $\mathcal{F}_{\mathsf{s<e}}$ if $m \geq 3$.
- Moving any other process leads to a configuration in $\mathcal{F}_{\mathsf{s<e}}$ too.

Therefore the play is winning for System and we get a contradiction, there is no winning strategy for Environment. Thus f is a winning k-strategy for System. The same strategy f also work for any $k' > k$, which completes the first direction of the proof.

$\boxed{\Leftarrow}$ Suppose that there is a constant $k \in \mathbb{N}$ and f a winning k-strategy for System.

Lemma 34. *For any f-compatible play $\rho = c_0 \tau_0 c'_0 \tau'_0 c_1 \tau_1 c'_1 \tau'_1 c_2 \dots \tau_{2n} c_{2n}$, there exists a run $\gamma_0 \vdash_{t_1} \gamma_1 \vdash_{t_2} \dots \gamma_n$ of M such that $c_{2i} \in \mathbb{C}(\gamma_i)$, for all $1 \leq i \leq n$.*

Proof. Let $\rho = c_0\tau_0c'_0\tau'_0c_1\tau_1c'_1\tau'_1c_2\dots\tau_{2n}c_{2n}$ be a f-compatible play, not necessarily maximal. From c_0, the only winning configurations reachable for System, without any past action of Environment are the ones in $\kappa_t \in \mathcal{F}_0$ for some transition $t \in T$ of the form $t : q_0 \xrightarrow{c_i++} q_1$ or $t : q_0 \xrightarrow{c_i==0} q_1$. Let t_1 be the transition such that $c'_0 \in \kappa t_1$ (since they are mutually exclusive, t_1 is well-defined). For simplicity, assume that $t_1 : q_0 \xrightarrow{c_1++} q_1$, but the other cases are similar. From c'_0, there is only one configuration reachable by Environment which is not winning for System: the one where there is exactly one process in locations $\langle t_1 b\rangle$, $\langle q_0 b\rangle$, $\langle a_1 b\rangle$. Now that the transition t_1 has been selected, the only winning configuration reachable by System is c'_1 such that $c'_1(\langle t_1^2 b\rangle) = c'_1(\langle q_0^2 b\rangle) = c'_1(\langle a_1^2 b\rangle) = c'_1(\langle q_1\rangle) = 1$, $c'_1(\ell_0) \geq 0$, and $c'_1(\ell) = 0$ for all other $\ell \in L$. Indeed, all other winning configurations require moves of Environment to be reached, or require that Environment has never played. Now, the first accepting condition prevents Environment to play b on any new process, or to play several b on processes that have already played. Moreover, if she plays b on the process already in the location $\langle q_1\rangle$, the configuration reached is in $\mathcal{F}^{\mathsf{e}}_{(q_0,t_1,q_1)}$. The only possibility to leave the set of winning configurations is then to reach c_2 defined by $c_2(\langle q_1\rangle) = c_2(\langle a_1^2 b^2\rangle) = c_2(\langle t_1^2 b^2\rangle) = c_2(\langle q_0^2 b^2\rangle) = 1$, $c_2(\ell_0) = k-3$ and $c_2(\ell) = 0$ for all other $\ell \in L$. Hence, c_2 is valid and $\mathsf{m}(c_2) = (q_1, 1, 0) = \gamma_1$. Moreover, $\gamma_0 \vdash_{t_1} \gamma_1$.

Let now $j < n$ and suppose that we have built $\gamma_0 \vdash_{t_1} \gamma_1 \cdots \vdash_{t_j} \gamma_j$ with $\gamma_i = (q_i, \nu_1^i, \nu_2^i) = \mathsf{m}(c_{2i})$ for all $1 \leq i \leq j$. From the valid configuration c_{2i} such that $c_{2i}(\langle q_0^2 b^2\rangle) > 0$, the only winning configurations reachable by System are the ones in $\mathcal{F}_{(q_j,t)}$ for some t starting in q_j. Let t_{j+1} be the transition such that $c'_{2i} \in \mathcal{F}_{(q_j,t_{j+1})}$. Assume for example that $t_{j+1} : q_j \xrightarrow{c_1--} q_{j+1}$.

Then $c'_{2j}(\langle t_{j+1}\rangle) = c'_{2j}(\langle a_1^3 b^2\rangle) = 1$, $c'_{2j}(\ell_0) = c_{2j}(\ell_0) - 1$, $c'_{2j}(\langle a_1^2 b^2\rangle) = c_{2j}(\langle a_1^2 b^2\rangle) - 1$, and $c'_{2j}(\ell) = c_{2j}(\ell)$ for all other $\ell \in L$, with $c_{2j}(\ell) \neq 0$ implies that $\ell \in L_{\checkmark}$ since c_{2j} is valid.

As before, in order to reach a non winning configuration, the only possibility for Environment is to go to c_{2j+1} such that $c_{2j+1}(\langle t_{j+1} b\rangle) = c_{2j+1}(\langle a_1^3 b^3\rangle) = c_{2j+1}(\langle q_j b\rangle) = 1$, $c_{2j+1}(\langle t_{j+1}\rangle) = c_{2j+1}(\langle q_j\rangle) = c_{2j+1}(\langle a_1^3 b^2\rangle) = 0$, and $c_{2j+1}(\ell) = c'_{2j}(\ell)$ for all other $\ell \in L$.

Again, the only winning configuration System can reach without the help of Environment is c'_{2j+1} such that $c'_{2j+1}(\langle t_{j+1}^2 b\rangle) = c_{2j+1}(\langle a_1^4 b^3\rangle) = c_{2j+1}(\langle q_j^2 b\rangle) = 1$, $c_{2j+1}(\langle q_{j+1}\rangle) = 1$, $c_{2j+1}(\langle q_{j+1} b\rangle) = c_{2j+1}(\langle a_1^3 b^3\rangle) = 0$, and $c_{2j+1}(\ell) = c'_{2j}(\ell)$ for all other $\ell \in L$.

Finally, the analysis of all the winning conditions shows that Environment cannot play anything else that $c_{2j+2}(\langle t_{j+1}^2 b^2\rangle) = c_{2j}(\langle t_{j+1}^2 b^2\rangle) + 1$, $c_{2j+2}(\langle a_1^4 b^4\rangle) = c_{2j}(\langle a_1^4 b^4\rangle) + 1$, $c_{2j+2}(\langle q_j^2 b^2\rangle) = c_{2j}(\langle q_j^2 b^2\rangle) + 1$, and $c_{2j+2}(\ell) = c'_{2j+1}(\ell)$ for all other $\ell \in L$.

In particular, $c_{2j+2}(\langle q_{j+1}\rangle) = 1$, $c_{2j+2}(\langle a_1^2 b^2\rangle) = c_{2j}(\langle a_1^2 b^2\rangle) - 1$, $c_{2j+2}(\langle a_2^2 b^2\rangle) = c_{2j}(\langle a_2^2 b^2\rangle)$. Hence, c_{2j+2} is valid, and $\mathsf{m}(c_{2j+2}) = (q_{j+1}, \nu_1^j - 1, \nu_2^j) = \gamma_{j+1}$, with $\gamma_j \vdash_{t_{j+1}} \gamma_{j+1}$. □

Assume now that there is no accepting run of M and consider a maximal f-compatible play π. Since π is winning, it ends in a configuration reached by System, so it is of the form $\pi = c_0\tau_0\dots c_{2n}\dots c'_m$ with $m \in \{2n, 2n+1\}$, for some $n \in \mathbb{N}$, and $c'_m \models \mathcal{F}$. By Lemma 34, we have the corresponding run $\gamma_0 \vdash_{t_1} \gamma_1 \vdash_{t_2} \cdots \vdash_{t_n} \gamma_n$, with $c_{2n} \in \mathbb{C}(\gamma_n)$ and γ_n not a halting configuration. An analysis of the

possible moves of System in that case shows that $c'_{2n}(\langle t \rangle) = 1$ for some transition $t \in T$. But from such a configuration, Environment can easily reach a non winning configuration, by playing b on every location where the number of b is strictly smaller than the number of letters of System. Again, System moves to configuration c'_{2n+1}, which is winning. According to the precise definition of c'_{2n+1}, there is only one possibility for System: $c'_{2n+1}(\langle t^2 b \rangle) = 1$, $c'_{2n+1}(\langle q' \rangle) = 1$ for some $q' \in Q$, and possibly $c'_{2n+1}(\langle a_i^2 b \rangle) = 1$ or $c'_{2n+1}(\langle a_i^4 b^3 \rangle) = 1$. In any case, Environment can still reach a non winning configuration by playing b on all these locations. Then, either the play is not maximal, or it is not winning, both of which contradict our hypotheses. Hence, there is an accepting run of M.

This ends the proof of undecidability of $\textsc{Game}(\mathbb{0}, \mathbb{0}, \mathbb{N})$, and therefore also proves the undecidability of $\textsc{Synth}(\text{FO}[\sim], \mathbb{0}, \mathbb{0}, \mathbb{N})$ since they are equivalent by Lemma 26. □

Whether $\textsc{Game}(\mathbb{N}, \mathbb{N}, \mathbb{0})$ is decidable or not is left open. Now obviously the last proof heavily uses the fact that there is an unbounded number of processes Environment can affect. Therefore a natural question is to ask whether the game problem becomes decidable when there is only a fixed number of Environment and mixed processes. This case, which is the last we study, is the focus of the next section.

4.3.4 Case of $(\mathbb{N}, \{k_{\mathsf{e}}\}, \{k_{\mathsf{se}}\})$

We show that when Environment can affect only a fixed number of processes, then any game has a cutoff, and therefore the game problem is decidable. Let us fix $k_{\mathsf{e}}, k_{\mathsf{se}} \in \mathbb{N}$ the number of Environment and mixed processes respectively. The number of System processes is, of course, still unbounded.

Theorem 35. *Given $k_{\mathsf{e}}, k_{\mathsf{se}} \in \mathbb{N}$, every game $\mathcal{G} = (A, B, \mathcal{F})$ has a cutoff with respect to $(\mathbb{N}, \{k_{\mathsf{e}}\}, \{k_{\mathsf{se}}\})$. More precisely: Let K be the largest constant that occurs in $\mathcal{F}$. Moreover, let $Max = (k_{\mathsf{e}} + k_{\mathsf{se}}) \cdot |A_{\mathsf{e}}| \cdot B$ and $N_{cut} = |L|^{Max+1} \cdot K$. Then, $(N_{cut}, k_{\mathsf{e}}, k_{\mathsf{se}})$ is a cutoff of $Win(\mathcal{G})$ with respect to $(\mathbb{N}, \{k_{\mathsf{e}}\}, \{k_{\mathsf{se}}\})$.*

The intuition is that when there is a "large enough" number of System processes, then if System can win then she can also win with one more System process by somehow emulating the previous winning strategy. Since this additional process cannot be affected by Environment, then this new strategy must also be winning.

Proof. We will show that, for all $N \geq N_{\text{cut}}$,

$$(N, k_{\mathsf{e}}, k_{\mathsf{se}}) \in \text{Win}(\mathcal{G}) \Leftrightarrow (N+1, k_{\mathsf{e}}, k_{\mathsf{se}}) \in \text{Win}(\mathcal{G})$$

The main observation is that, when c contains more than K tokens in a given $\ell \in L$, adding more tokens in ℓ will not change whether $c \models \mathcal{F}$. Given $c, c' \in Conf$, we write $c <_{\mathsf{e}} c$ if $c \neq c'$ and there is $\tau \in T_{\mathsf{e}}$ such that $\tau(c) = c'$. Note that the length d of a chain $c_0 <_{\mathsf{e}} c_1 <_{\mathsf{e}} \ldots <_{\mathsf{e}} c_d$ is bounded by Max. In other words, Max is the maximal number of transitions that Environment can do in a play. For all $d \in \{0, \ldots, Max\}$, let $Conf_d$ be the set of configurations $c \in Conf$ such that the longest chain in $(Conf, <_{\mathsf{e}})$ starting from c has length d. For instance, the initial configuration $c_{(N,k_{\mathsf{e}},k_{\mathsf{se}})}$ is in $Conf_{Max}$, and $Conf_0$ is the set of configurations where

all Environment and mixed tokens are in locations with the maximum number of Environment actions.

We claim that the following proposition holds: Suppose that $c \in \mathit{Conf}_d$ and $\ell \in L$ such that $c(\ell) = (N, n_{\mathsf{e}}, n_{\mathsf{se}})$ with $N \geq |L|^{d+1} \cdot K$ and $n_{\mathsf{e}}, n_{\mathsf{se}} \in \mathbb{N}$. Set $\hat{c} \in \mathit{Conf}_d$ such that

$$\hat{c}(\ell') = \begin{cases} (N+1, n_{\mathsf{e}}, n_{\mathsf{se}}) & \text{if } \ell' = \ell, \\ c(\ell') & \text{otherwise.} \end{cases}$$

Then,

$$c \text{ is winning for System} \iff \hat{c} \text{ is winning for System.}$$

To show the claim, we proceed by induction on $d \in \mathbb{N}$. In each implication, we distinguish the cases $d = 0$ and $d \geq 1$. For the latter, we assume that equivalence holds for all values strictly smaller than d.

For $\tau \in T_{\mathsf{s}}$ and $\ell, \ell' \in L$, we let $\tau[(\ell, \ell', \mathsf{s}){+}{+}]$ denote the transition $\hat{\tau} \in T_{\mathsf{s}}$ given by $\hat{\tau}(\ell_1, \ell_2, \mathsf{e}) = \tau(\ell_1, \ell_2, \mathsf{e}) = 0$, $\hat{\tau}(\ell_1, \ell_2, \mathsf{se}) = \tau(\ell_1, \ell_2, \mathsf{se})$, $\hat{\tau}(\ell_1, \ell_2, \mathsf{s}) = \tau(\ell_1, \ell_2, \mathsf{s}) + 1$ if $(\ell_1, \ell_2) = (\ell, \ell')$, and $\hat{\tau}(\ell_1, \ell_2, \mathsf{s}) = \tau(\ell_1, \ell_2, \mathsf{s})$ if $(\ell_1, \ell_2) \neq (\ell, \ell')$. We define $\tau[(\ell, \ell', \mathsf{s}){-}{-}]$ similarly (provided $\tau(\ell, \ell', \mathsf{s}) \geq 1$).

$\boxed{\Rightarrow}$ Let f be a winning strategy for System from $c \in \mathit{Conf}_d$. Let $\tau' = f(c)$ and $c' = \tau'(c)$. Note that $c' \models \mathcal{F}$ by definition of a strategy.

Now, since there is a "large" amount of tokens in state ℓ in configuration c, then whatever System chose for the next transition, there is a state ℓ' which is a successor of ℓ (possibly ℓ itself) and which ends up with a "large" amount of tokens in c'. Formally, since $c(\ell, \mathsf{s}) = N \geq |L|^{d+1} \cdot K$, there is $\ell' \in L$ such that $\ell + w = \ell'$ for some $w \in A_{\mathsf{s}}^*$ and $c'(\ell', \mathsf{s}) = N' \geq |L|^d \cdot K$. Indeed, as there are $|L|$ local states in total, at least one of them must have at least $N/|L|$ tokens after the transition is applied.

We show that $\hat{c}$ is winning for System by exhibiting a corresponding winning strategy $\hat{f}$ from $\hat{c}$ that will carefully control the position of the additional token. First, set $\hat{f}(\hat{c}) = \hat{\tau}'$ where $\hat{\tau}' = \tau'[(\ell, \ell', \mathsf{s}){+}{+}]$. Let $\hat{c}' = \hat{\tau}'(\hat{c})$. We obtain $\hat{c}'(\ell', \mathsf{s}) = N' + 1$. Note that, since $N' \geq K$, the acceptance condition $\mathcal{F}$ cannot distinguish between c' and $\hat{c}'$. Thus, we have $\hat{c}' \models \mathcal{F}$.

Case $d = 0$: As, for all transitions $\hat{\tau}'' \in T_{\mathsf{e}}$, we have $\hat{\tau}''(\hat{c}') = \hat{c}' \models \mathcal{F}$, we reached a maximal play that is winning for System. We deduce that $\hat{c}$ is winning for System.

Case $d \geq 1$: Take any $\hat{\tau}'' \in T_{\mathsf{e}}$ and $\hat{c}''$ such that $\hat{c}'' = \hat{\tau}''(\hat{c}') \not\models \mathcal{F}$. Let $\tau'' = \hat{\tau}''$ and $c'' = \tau''(c')$. Note that $\hat{c}''(\ell', \mathsf{s}) = \hat{c}'(\ell', \mathsf{s}) = N' + 1$ and $c''(\ell', \mathsf{s}) = c'(\ell', \mathsf{s}) = N'$ because a transition from Environment cannot move System tokens, and $c'', \hat{c}'' \in \mathit{Conf}_{d^-}$ for some $d^- < d$. As f is a winning strategy for System from c, we have that c'' is winning for System.

By induction hypothesis, $\hat{c}''$ is winning for System, say by winning strategy $\hat{f}'$. We let

$$\hat{f}(\hat{c}\,\hat{\tau}'\,\hat{c}'\,\hat{\tau}''\,\pi) = \hat{f}'(\rho)$$

for all $\hat{c}''$-plays ρ. For all unspecified plays, let $\hat{f}$ be undefined. Altogether, for any choice of $\hat{\tau}''$, we have that $\hat{f}'$ is winning from $\hat{c}''$. Thus, $\hat{f}$ is a winning strategy from $\hat{c}$.

$\boxed{\Leftarrow}$ Suppose $\hat{f}$ is a winning strategy for System from $\hat{c}$. Thus, for $\hat{\tau}' = \hat{f}(\hat{c})$ and $\hat{c}' = \hat{\tau}'(\hat{c})$, we have $\hat{c}' \models \mathcal{F}$. Recall that $\hat{c}(\ell, \mathsf{s}) \geq (|L|^{d+1} \cdot K) + 1$. We distinguish two cases:

1. Suppose there is $\ell' \in L$ such that $\ell \neq \ell'$, $\hat{c}'(\ell', \mathsf{s}) = N'+1$ for some $N' \geq |L|^d \cdot K$, and $\hat{\tau}'(\ell, \ell', \mathsf{s}) \geq 1$. Then, we set $\tau' = \hat{\tau}'[(\ell, \ell', \mathsf{s})\text{--}]$.
2. Otherwise, we have $\hat{c}'(\ell, \mathsf{s}) \geq (|L|^d \cdot K)+1$, and we set $\tau' = \hat{\tau}'$ (as well as $\ell' = \ell$ and $N' = N$).

Let $c' = \tau'(c)$. Since $\hat{c}' \models \mathcal{F}$, one obtains $c' \models \mathcal{F}$.

Case $d = 0$: For all transitions $\tau'' \in T_{\mathsf{e}}$, we have $\tau''(c') = c' \models \mathcal{F}$. Thus, we reached a maximal play that is winning for System. We deduce that c is winning for System.

Case $d \geq 1$: Take any $\tau'' \in T_{\mathsf{e}}$ such that $c'' = \tau''(c') \not\models \mathcal{F}$. Let $\hat{\tau}'' = \tau''$ and $\hat{c}'' = \hat{\tau}''(\hat{c}')$. We have $c'' = \hat{c}''[(\ell', \mathsf{s}) \mapsto N']$, $\hat{c}'' = c''[(\ell', \mathsf{s}) \mapsto N' + 1]$, and $c'', \hat{c}'' \in \mathit{Conf}_{d^-}$ for some $d^- < d$.

As $\hat{c}''$ is winning for System, by induction hypothesis, c'' is winning for System, say by winning strategy f'. We let

$$f(c\,\tau'\,c'\,\tau''\,\pi) = f'(\pi)$$

for all c''-plays π. For all unspecified plays, let f be undefined. Again, for any choice of τ'', f' is winning from c''. Thus, f is a winning strategy from c.

This concludes the proof of the claim and, therefore, of Theorem 35. □

Corollary 36. *Let $k_{\mathsf{e}}, k_{\mathsf{se}} \in \mathbb{N}$ be the number of environment and the number of mixed processes, respectively. The problems* Game$(\mathbb{N}, \{k_{\mathsf{e}}\}, \{k_{\mathsf{se}}\})$ *and* Synth$(\mathrm{FO}[\sim], \mathbb{N}, \{k_{\mathsf{e}}\}, \{k_{\mathsf{se}}\})$ *are decidable.*

In particular, by Theorem 35, the game problem can be reduced to an exponential number of acyclic finite-state games whose size (and hence the time complexity for determining the winner) is exponential in the cutoff and, therefore, doubly exponential in the size of the alphabet, the bound B, and the fixed number of processes that are controllable by the environment.

Chapter 5

Conclusion

The scope of this work was to study the control and synthesis problems for open parameterized systems.

5.1 Summary of our contributions

In the first half of this thesis, we studied the control problem for dynamic pushdown systems. As in the general case even emptiness is undecidable for such systems, we restricted ourselves to round-bounded executions, which is a natural restriction that can be found in many real-life scenarios. We showed that the control problem is decidable under this restriction by a reduction to phase-bounded multi-pushdown games, and then we showed that the problem is inherently non-elementary. Finally we showed that even slightly relaxing this round restriction to another restriction that we call context-bounded executions immediately leads to undecidability.

In the second half, we focused on the synthesis problem for specifications given in some fragments of first-order logic. This synthesis problem is parameterized by the FO fragment used and by the cardinality of the sets of processes involved in executions. First we proved that the synthesis problem is undecidable for FO with two variables when only mixed processes are involved. Then, we studied FO with only the data equality predicate by giving a normal form for those formulas and translating them into a game formalism more suited for reasoning. Thanks to these results, we showed that synthesis for this fragment is undecidable again when only mixed processes are involved, but that when the environment can only affect a finite, bounded number of processes then the problem becomes decidable.

5.2 Perspectives

In both cases, our results lead to various possible future works. In the case of the control problem, it would be interesting to see if our results may be used for model checking branching-time properties. Indeed, such games are heavily used in that context (see for instance [LS02]), so that we could use some variant of branching-time logic such as in [Kar16] and then reduce the model-checking problem to a dynamic pushdown game. It would also be interesting to study the relation between our control problem and the parameterized population control problem as defined

in [BDG+18] where one player chooses an action to perform in a non-deterministic automaton and the other player resolves the non-determinism for a number of agents.

In the case of synthesis, it would be nice to have tight bounds in the decidable cases for the complexity of the decision algorithm. Furthermore, several open cases are left which are interesting in their own. For instance, we did not explore whether the synthesis problem for FO^2 becomes decidable again when we restrict the number of processes controlled by Environment, as in the case with FO[$\sim$]. Since the satisfiability problem (i.e. with no Environment) is decidable [BDM+11], there is a gap with our undecidability result, so finding the exact boundary between the two (or at least reducing that gap) would be interesting. Also, one possible lead is to try to extend these results to the dynamic case, as our results crucially rely on the fact that there is a finite number of processes in each execution.

More generally, our results here build strategies for the system as a whole, that is, our strategies have a global view of the whole system and then tell each process what actions to perform. It would be interesting to synthesize strategies for individual processes, which may only have some partial information on the state of the system. For instance, maybe a process can only observe its own past actions and those of other processes it has synchronized with, or only the actions of its neighbors for some kind of fixed architecture. See, e.g., the works in [FO17, BFHH19] in the case of a fixed number of processes, and [JB14] for parameterized systems over ring architectures. This problem could also be related to parameterized games where the number of adversaries is not fixed such as in [BBM19]. A possible goal for the future is to work toward a more general case with dynamic creation of processes and with relaxed architecture constraints.

Bibliography

[ABd03] P. A. Abdulla, A. Bouajjani, and J. d'Orso. Deciding monotonic games. In *CSL'03*, volume 2803 of *LNCS*, pages 1–14. Springer, 2003.

[ABKS17] M. F. Atig, A. Bouajjani, K. Narayan Kumar, and P. Saivasan. Parity games on bounded phase multi-pushdown systems. In *NETYS'17*, volume 10299 of *LNCS*, pages 272–287, 2017.

[ABQ11] M. F. Atig, A. Bouajjani, and S. Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *Log. Methods Comput. Sci.*, 7(4), 2011.

[AMSS13] P. A. Abdulla, R. Mayr, A. Sangnier, and J. Sproston. Solving parity games on integer vectors. In *CONCUR'13*, volume 8052, pages 106–120. Springer, 2013.

[BBLS20] Béatrice Bérard, Benedikt Bollig, Mathieu Lehaut, and Nathalie Sznajder. Parameterized synthesis for fragments of first-order logic over data words. In *FoSSaCS*, pages 97–118, 2020.

[BBM19] Nathalie Bertrand, Patricia Bouyer, and Anirban Majumdar. Concurrent parameterized games. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[BDG+18] Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, Hugo Gimbert, and Adwait Amit Godbole. Controlling a population. *arXiv preprint arXiv:1807.00893*, 2018.

[BDM+11] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27, 2011.

[BESS05] Ahmed Bouajjani, Javier Esparza, Stefan Schwoon, and Jan Strejček. Reachability analysis of multithreaded software with asynchronous communication. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 348–359. Springer, 2005.

[BFHH19] Raven Beutner, Bernd Finkbeiner, and Jesko Hecking-Harbusch. Translating Asynchronous Games for Distributed Synthesis. In Wan Fokkink

and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory (CONCUR 2019)*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[BJK10] T. Brázdil, P. Jancar, and A. Kucera. Reachability games on extended vector addition systems with states. In *ICALP'10, Part II*, volume 6199 of *LNCS*, pages 478–489. Springer, 2010.

[BJK+15] Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. Decidability of parameterized verification. *Synthesis Lectures on Distributed Computing Theory*, 6(1):1–170, 2015.

[BJNT00] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *International Conference on Computer Aided Verification*, pages 403–418. Springer, 2000.

[BK12] Benedikt Bollig and Dietrich Kuske. An optimal construction of Hanf sentences. *J. Applied Logic*, 10(2):179–186, 2012.

[BL69] Julius R. Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, April 1969.

[BLS18] Benedikt Bollig, Mathieu Lehaut, and Nathalie Sznajder. Round-bounded control of parameterized systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 370–386. Springer, 2018.

[BMOT05] Ahmed Bouajjani, Markus Müller-Olm, and Tayssir Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *International Conference on Concurrency Theory*, pages 473–487. Springer, 2005.

[BMS+06] Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 7–16. IEEE, 2006.

[BS10] Henrik Björklund and Thomas Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411(4-5):702–715, 2010.

[BT16] B. Brütsch and W. Thomas. Playing games in the Baire space. In *Proc. Cassting Workshop on Games for the Synthesis of Complex Systems and 3rd Int. Workshop on Synthesis of Complex Parameters*, volume 220 of *EPTCS*, pages 13–25, 2016.

[Chu57] Alonzo Church. Applications of recursive arithmetic to the problem of circuit synthesis. In *Summaries of the Summer Institute of Symbolic Logic – Volume 1*, pages 3–50. Institute for Defense Analyses, 1957.

[CHVB18] Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of model checking*, volume 10. Springer, 2018.

[CJGK+18] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking*. MIT press, 2018.

[CS14] J.-B. Courtois and S. Schmitz. Alternating vector addition systems with states. In *MFCS'14*, volume 8634 of *LNCS*, pages 220–231. Springer, 2014.

[DDG12] Stéphane Demri, Deepak D'Souza, and Régis Gascon. Temporal logics of repeating values. *J. Log. Comput.*, 22(5):1059–1096, 2012.

[DGK08] Volker Diekert, Paul Gastin, and Manfred Kufleitner. A survey on small fragments of first-order logic over finite words. *International Journal of Foundations of Computer Science*, 19(03):513–548, 2008.

[DL09] S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), 2009.

[EFR19] Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. Synthesis of Data Word Transducers. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory (CONCUR 2019)*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[EJ91] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of FOCS'91*, pages 368–377. IEEE Computer Society, 1991.

[EN95] E Allen Emerson and Kedar S Namjoshi. Reasoning about rings. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 85–94, 1995.

[Esp14] J. Esparza. Keeping a crowd safe: On the complexity of parameterized verification. In *STACS'14*, volume 25 of *Leibniz International Proceedings in Informatics*, pages 1–10. Leibniz-Zentrum für Informatik, 2014.

[FGS19] Hadar Frenkel, Orna Grumberg, and Sarai Sheinvald. An automata-theoretic approach to model-checking systems and specifications over infinite data domains. *J. Autom. Reasoning*, 63(4):1077–1101, 2019.

[FO17] Bernd Finkbeiner and Ernst-Rüdiger Olderog. Petri games: Synthesis of distributed systems with causal memory. *Inf. Comput.*, 253:181–203, 2017.

[FP18a] D. Figueira and M. Praveen. Playing with repetitions in data words using energy games. In *Proceedings of LICS'18*, pages 404–413. ACM, 2018.

[FP18b] Diego Figueira and M Praveen. Playing with repetitions in data words using energy games. *arXiv preprint arXiv:1802.07435*, 2018.

[FS05] Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS'05)*, pages 321–330. IEEE, 2005.

[Für83] Martin Fürer. The computational complexity of the unconstrained limited domino problem (with implications for logical decision problems). In Egon Börger, Gisbert Hasenjaeger, and Dieter Rödding, editors, *Logic and Machines: Decision Problems and Complexity, Proceedings of the Symposium "Rekursive Kombinatorik" held from May 23-28, 1983 at the Institut für Mathematische Logik und Grundlagenforschung der Universität Münster/Westfalen*, volume 171 of *Lecture Notes in Computer Science*, pages 312–319. Springer, 1983.

[GKV97] Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

[Han65] W. Hanf. Model-theoretic methods in the study of elementary logic. In J. W. Addison, L. Henkin, and A. Tarski, editors, *The Theory of Models*. North-Holland, Amsterdam, 1965.

[HMRU00] J. E. Hopcroft, R. Motwani, Rotwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computability.* Addison-Wesley Longman Publishing Co., Inc., 2nd edition, 2000.

[HTWZ15] Florian Horn, Wolfgang Thomas, Nico Wallmeier, and Martin Zimmermann. Optimal strategy synthesis for request-response games. *RAIRO - Theor. Inf. and Applic.*, 49(3):179–203, 2015.

[Jan15] P. Jancar. On reachability-related games on vector addition systems with states. In *RP'15*, volume 9328 of *LNCS*, pages 50–62. Springer, 2015.

[JB14] S. Jacobs and R. Bloem. Parameterized synthesis. *Log. Methods Comput. Sci.*, 10(1), 2014.

[JORW11] Mark Jenkins, Joël Ouaknine, Alexander Rabinovich, and James Worrell. The church synthesis problem with metric. In Marc Bezem, editor, *Computer Science Logic, 25th International Workshop / 20th Annual Conference of the EACSL, CSL 2011, September 12-15, 2011, Bergen, Norway, Proceedings*, volume 12 of *LIPIcs*, pages 307–321. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[JTZ18] Swen Jacobs, Leander Tentrup, and Martin Zimmermann. Distributed synthesis for parameterized temporal logics. *Inf. Comput.*, 262(Part):311–328, 2018.

[Kar16] A. Kara. *Logics on data words: Expressivity, satisfiability, model checking.* PhD thesis, Technical University of Dortmund, 2016.

[KF94] M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

[KK19] Ayrat Khalimov and Orna Kupferman. Register-Bounded Synthesis. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory (CONCUR 2019)*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[KMB18] Ayrat Khalimov, Benedikt Maderbacher, and Roderick Bloem. Bounded synthesis of register transducers. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2018.

[KO12] E. Kieronski and M. Otto. Small substructures and decidability issues for first-order logic with two variables. *J. Symb. Log.*, 77(3):729–765, 2012.

[Koz77] D. Kozen. Lower bounds for natural proof systems. In *Proceedings of SFCS'77*, pages 254–266. IEEE Computer Society, 1977.

[KV01] Orna Kupferman and MY Vardi. Synthesizing distributed systems. In *Proceedings 16th Annual IEEE Symposium on Logic in Computer Science*, pages 389–398. IEEE, 2001.

[LMP07] S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS'07*, pages 161–170. IEEE Computer Society Press, 2007.

[LMP10a] S. La Torre, P. Madhusudan, and G. Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *CAV'10*, volume 6174 of *LNCS*, pages 629–644. Springer, 2010.

[LMP10b] S. La Torre, P. Madhusudan, and G. Parlato. Model-checking parameterized concurrent programs using linear interfaces. Technical Report 2142/15410, University of Illinois, 2010. Available at http://hdl.handle.net/2142/15410.

[LS02] M. Lange and C. Stirling. Model checking games for branching time logics. *J. Log. Comput.*, 12(4):623–639, 2002.

[LS15] Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 56–67. IEEE, 2015.

[LTKR08] Akash Lal, Tayssir Touili, Nicholas Kidd, and Thomas Reps. Interprocedural analysis of concurrent programs under a context bound. In *International conference on Tools and algorithms for the construction and analysis of systems*, pages 282–298. Springer, 2008.

[LTV15] Leonid Libkin, Tony Tan, and Domagoj Vrgoc. Regular expressions for data words. *J. Comput. Syst. Sci.*, 81(7):1278–1297, 2015.

[May84] Ernst W Mayr. An algorithm for the general petri net reachability problem. *SIAM Journal on computing*, 13(3):441–460, 1984.

[Min67] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, Upper Saddle River, NJ, USA, 1967.

[MP71] Robert McNaughton and Seymour A Papert. *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press, 1971.

[NSV04] Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic (TOCL)*, 5(3):403–435, 2004.

[PP86] Dominique Perrin and Jean-Eric Pin. First-order logic and star-free sets. *Journal of Computer and System Sciences*, 32(3):393–406, 1986.

[PP04] Dominique Perrin and Jean-Éric Pin. *Infinite words: automata, semigroups, logic and games*. Academic Press, 2004.

[PR90] Amir Pneuli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 746–757. IEEE, 1990.

[QR05] Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *International conference on tools and algorithms for the construction and analysis of systems*, pages 93–107. Springer, 2005.

[Rab72] Michael O. Rabin. *Automata on infinite objects and Church's problem*. Number 13 in Regional Conference Series in Mathematics. American Mathematical Soc., 1972.

[Ram00] G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, 2000.

[RSB05] Jean-François Raskin, Mathias Samuelides, and Laurent Van Begin. Games for counting abstractions. *Electr. Notes Theor. Comput. Sci.*, 128(6):69–85, 2005.

[Sav70] Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

[SB98] Thomas Schwentick and Klaus Barthelmann. Local normal forms for first-order logic with applications to games and automata. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 444–454. Springer, 1998.

[Seg06] Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *International Workshop on Computer Science Logic*, pages 41–57. Springer, 2006.

[Set09] A. Seth. Games on multi-stack pushdown systems. In *LFCS'09*, volume 5407 of *LNCS*, pages 395–408. Springer, 2009.

[SKMW17] Lutz Schröder, Dexter Kozen, Stefan Milius, and Thorsten Wißmann. Nominal automata with name binding. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 124–142, 2017.

[Sto74] L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, MIT, 1974.

[Wal01] I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.

[Zie98] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *TCS*, 200(1-2):135–183, 1998.