Games on Asynchronous Transition Systems

Submitted in partial fulfillment of the requirements
of the degree of

Doctor of Philosophy

by

Nehul Jain
(Roll No. 134050006)

Supervisor:
Prof. Bharat Adsul

Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
2025

Dedicated to my beloved parents.

Thesis Approval

This thesis entitled **Games on Asynchronous Transition Systems** by **Nehul Jain** is approved for the degree of **Doctor of Philosophy**.

Examiners:

.....

.....

.....

.....

Supervisor:

.....

Prof. Bharat Adsul

(Research Supervisor)

Chairperson:

.....

(Chairperson)

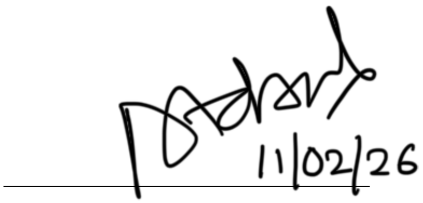
Date: 11.2.2026

Place: IITB, Mumbai

Certificate of Authorization

This is to certify that the thesis entitled *"Games on Asynchronous Transition Systems"* submitted by **Nehul Jain** (Roll No. **134050006**) to the **Indian Institute of Technology Bombay**, in partial fulfillment of the requirements for the award of the degree of **Doctor of Philosophy** in the Department of **Computer Science and Engineering**, has been prepared under my/our supervision.

The thesis is hereby authorized for submission.



Prof. Bharat Adsul

(Research Supervisor)

Declaration

I declare that this written submission represents my ideas in my own words and, where others' ideas or words have been included, I have adequately cited and referenced the original sources.

I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented, fabricated, or falsified any idea, data, fact, or source in my submission.

I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have not been properly cited, or from whom proper permission has not been taken when needed.

I further declare that I have not used any modern AI tools or any other similar tools for writing the thesis.

Nehul Jain 11·2·2026

(Signature of the Candidate)

Nehul Jain

134050006

Date: 11·2·2026

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY, INDIA

CERTIFICATE OF COURSE WORK

This is to certify that **Nehul Jain** (Roll No. 134050006) was admitted to the candidacy of Ph.D. degree on 16 July 2013, after successfully completing all the courses required for the Ph.D. programme. The details of the course work done are given below.

S.No	Course Code	Course Name	Credits
1	CS 713	Special Topics in Automata and Logics	6
2	CS 735	Formal Models for Concurrent and Asynchronous Systems	6
3	CSS 801	Seminar	4
4	GC 101	Gender in the Workplace	PP
		Total Credits	**16**

IIT Bombay

Date: Dy. Registrar (Academic)

Abstract

This thesis introduces and develops a novel framework of *asynchronous transition system (ATS) games* played on non-deterministic asynchronous automata involving multiple processes. This game is played between an environment and the distributed team of processes where each process has only partial information of the ongoing play – namely its causal past. The key algorithmic question is to decide the existence of a distributed co-operative winning strategy for the team of processes. In order to represent finite-state distributed strategies, we formulate distributed *memory automata*, which deterministically specify responses to environment moves and update local memory states accordingly.

We carry out a thorough analysis of these games in the special case of two processes. For safety and local reachability objectives, finite memory suffices, with processes remembering only the last global synchronization and are shown to be *NP-complete*. For global reachability, additional memory of all local states since the last synchronization is required. For global reachability, the complexity increases sharply: the problem is *PSPACE-hard* with an *NEXPTIME* upper bound.

We also study *CDM games* which feature a *central decision maker* process that participates in all key decision making tasks. We show that the problem of deciding the existence of a distributed winning strategy is tractable for global safety and local parity objectives and are *EXPTIME-complete*. A central insight is a transformation from sequential strategies in associated graph games into distributed strategies for CDM games, achieved through a careful linearization of causal pasts. Our findings offer new insights into the memory requirements and the strategic complexities of these distributed games. We provide novel constructions of *finite-state* distributed winning strategies for these objectives by employing *gossip automaton* to update their best knowledge of the global state during synchronization. We also establish optimality by showing instances where that decision maker process do require local memory exponential in the number of processes.

The picture changes fundamentally when multiple decision makers are introduced: ATS games with two independent decision makers are shown to be *undecidable*, even for simple safety objectives. This result, proved via an adaptation of asynchronous control game techniques, underscores the intrinsic hardness of concurrency in decision making.

It turns out ATS games are equivalent to recently studied *asynchronous control games*, where actions are partitioned into controllable and uncontrollable sets and processes are allowed to block controllable actions. We present translations in both directions, indicating that undecidability and complexity results transfer between the two models.

Acknowledgments

I wish to record my deep sense of gratitude to my supervisor, Prof. Bharat Adsul, for his valuable guidance and constant support at all stages of my Ph.D. study and related research. I am also thankful to the members of my Research Progress Committee—Prof. Milind A. Sohoni, Prof. Krishna Shankar Narayanan, and Prof. Akshay S.—for their constructive feedback and encouragement throughout this journey.

I am grateful to my seniors, friends, and colleagues in the lab for their companionship, discussions, and support that made this experience enriching, in particular Abhisekh Sankaran and Shantanu Kulkarni.

Finally, I would like to express my sincere gratitude to my family for their constant support, and especially to my daughter, whose love and cheerfulness have been a continual source of strength and inspiration.

Contents

List of Abbreviations

ATS	Asynchronous Transition System
CDM	Central Decision Maker
2DM	Two Decision Makers
MSO	Monadic Second-Order Logic
NP	Nondeterministic Polynomial Time
PSPACE	Polynomial Space
EXPTIME	Exponential Time
NEXPTIME	Nondeterministic Exponential Time

List of Figures

Chapter 1

Introduction

Full information two player graph games [Grädel et al., 2002] are played by two competing players. The study of these games has been central to reactive synthesis. The two players, we call system and environment, alternate to make their moves. This generates a sequence of interactions. It is expected that this sequence of interactions satisfies certain properties for the system to win. These desirable properties are expressed as a specification using some formalism.

Two player full information games are well studied and provide the foundation for *reactive synthesis*, where the system is required to respond correctly to all possible behaviors of the environment. A landmark formulation of this problem is Church's synthesis question (see [Thomas, 2008]), posed in 1957, which asks whether it is possible to algorithmically construct a *finite-state procedure* that transforms an infinite input sequence α into an output sequence β so that the pair (α, β) satisfies a given logical specification. These specifications are usually expressed in *monadic second-order logic (MSO)* over the structure $(\mathbb{N}, +1)$, also known as *S1S*. Unlike standard program design, where inputs and outputs are related only at the beginning and end of a computation, Church's formulation requires the transformation to be carried out *bit by bit*: at every time step t, the procedure must produce the output $\beta(t)$ immediately after

receiving the input $\alpha(t)$. Moreover, the solution must be realized by a finite-state device such as a Mealy automaton, which operates with bounded memory and processes inputs and outputs in real time.

The specifications express logical relations between the two streams of bits. Typical examples include safety requirements such as "every time an input is 1, the output must also be 1" ($\forall t(\alpha(t) = 1 \rightarrow \beta(t) = 1)$), liveness conditions such as "the output never produces two consecutive zeros" ($\neg\exists t(\beta(t) = \beta(t+1) = 0)$), and fairness constraints requiring that certain events recur infinitely often. Importantly, the problem concerns *non-terminating computations*: infinity arises not from unbounded data, but from the passage of time, since the input-output process continues without end. The central task is therefore to construct a finite-state machine that satisfies the specification, or to prove that no such machine exists.

The problem was elegantly solved by Büchi and Landweber (see [Thomas, 2008]) in 1969, who showed that for every S1S specification one can effectively decide whether a solution exists, and if so, construct a finite-state procedure that realizes it.

A modern view of their solution [Thomas, 2008] uses the connections between logic, automata, and *infinite two-player games*. First, the logical specification $\varphi(X,Y)$ is translated into a deterministic parity automaton over infinite words, which accepts exactly those input-output pairs that satisfy the requirement. This automaton is then interpreted as a game in which two players move alternately: Player A (the environment) supplies the input bits, while Player B (the system) produces the outputs. Player B's goal is to satisfy a given *state-based winning condition* while producing an output stream in response to the input stream (given by the environment). In this game-theoretic setting, the synthesis problem reduces to finding a winning strategy for the system (player B).

This perspective connects naturally to well-studied classes of infinite games on graphs—such as safety, reachability, and parity games. Parity games generalize both safety and reachability by allowing the winning condition to be defined through an assignment of priorities (non-negative integers) to game positions. The system wins if the maximum priority that occurs infinitely often along the play is even. Parity conditions are particularly important because they can express a wide range of liveness and fairness properties. While there are polynomial-time algorithms for solving safety and reachability games, no polynomial-time algorithm is known for solving parity games. Notably, for all three objectives—safety, reachability, and parity—there always

exist *zero-memory* (memoryless) *uniform*(initial-state independent) strategies: the system can choose its next move based only on the current state, without needing to remember the history of the play.

This automaton-to-game view of synthesis has proved remarkably robust in the sequential setting. However, when concurrency and distribution are introduced, the sequential model no longer suffices: processes have only partial views of their past, and actions may occur asynchronously. Our model, asynchronous transition system games (ATS games), can be seen as an extension to these sequential games. Instead of playing on a sequential automaton, the game is played on a non-deterministic asynchronous transition system that models distributed processes and concurrency explicitly. This generalization allows us to capture richer distributed synthesis problems beyond the purely sequential setting addressed by Büchi and Landweber.

The design and analysis of concurrent programs and distributed protocols present significant challenges, which have motivated extensive research on extending synthesis techniques to distributed settings. In this context, synthesis refers to the automatic construction of correct-by-design implementations that interact with their environment while satisfying specified behavioral requirements. To address these challenges, researchers have explored a variety of logical formalisms, game models, and algorithmic approaches, aiming to make distributed synthesis both expressive and tractable.

Related Work

In the sequential setting, a rich body of work has developed logics and algorithms that make the synthesis of reactive systems possible. However, once concurrency is introduced, the problem becomes substantially more intricate: processes operate with only partial views of their past and communication can be restricted. These factors have motivated extensive research into models and frameworks that can capture distributed behaviors while still admitting algorithmic analysis.

In the foundational work on distributed synthesis [Pnueli and Rosner, 1990] the authors consider finite state distributed reactive system. A set of processes communicate synchronously using a set of single reader/writer variables. Propositional Temporal specification (PTL/LTL)

that is to be realized is given over the set of input, output variables. In this setting when processes have access to only purely local information and limited communication capabilities the problem of distributed synthesis is proved to be undecidable in general, and non elementarily decidable for a very restricted class of pipeline architecture.

There have been various attempts to incorporate features such as independence between distributed components, concurrency, and communication into a formal computational model. Significant work has focused on models where components share a *causal past*—that is, the history of events and decisions that causally precede and influence a given point in the system's execution. By explicitly tracking this causal history, these models can more accurately capture the partial ordering of events that arises in concurrent and distributed systems.

Works such as [Gastin et al., 2004; Madhusudan et al., 2005; Genest et al., 2013; Muscholl and Walukiewicz, 2014; Finkbeiner and Olderog, 2017; Gimbert, 2017; Adsul and Jain, 2025] have introduced richer models in which processes have access to their entire *causal past*, thereby broadening the range of problems that can be addressed. These models typically rely on asynchronous automata introduced by Zielonka [Zielonka, 1987], which operate over Mazurkiewicz traces: computation structures that capture concurrency and causal dependence using partial order of events [Diekert and Rozenberg, 1995; Mukund, 2012]. In this setting, each process evolves independently, and synchronizations allow the processes to share their causal past, enabling coordinated decision-making. In particular, [Gastin et al., 2004] introduced the notion of causal past and addressed the distributed controller synthesis problem in series-parallel systems and established that controlled reachability is decidable. The work [Madhusudan et al., 2005] introduced systems with connectedly communicating processes and proved that the MSO theory in this setting is decidable which implies that the associated distributed synthesis problems are decidable.

Two closely related contemporary lines of work are *control games* [Genest et al., 2013] and *Petri games* [Finkbeiner and Olderog, 2017]. Both frameworks aim to capture the challenges of distributed decision making under partial information, where individual processes have access only to their *causal past* and can coordinate solely through synchronization. While control games are formulated on asynchronous automata and emphasize controller synthesis in distributed architectures, Petri games extend Petri nets with a game-theoretic interpretation in which tokens act as players carrying information. These frameworks capture partial-information

strategies and yield decidability in certain constrained cases, but also establish strong undecidability results even for simple objectives with constant number of processes. We briefly outline these two approaches below.

Control games [Genest et al., 2013] are played on deterministic asynchronous automata over a distributed alphabet, where processes make decisions based on their causal past. In such games, processes move independently on disjoint actions and synchronize on overlapping actions, at which point they share the information they might have stored in their local states, effectively gaining access to their entire causal past. The controller synthesis problem is formulated, where the specification is given over a deterministic asynchronous automaton on traces. The set of actions is partitioned into controllable and uncontrollable ones. At any state, a process may block some of the controllable actions, based on its causal past, while uncontrollable actions can never be blocked. An action can be executed only if it is allowed by all participating processes.

These games are shown to be decidable in the interesting setting of acyclic architectures [Genest et al., 2013; Muscholl and Walukiewicz, 2014] whose underlying process-communication graph is acyclic. More general formulations like decomposable games [Gimbert, 2017] have also been identified and proven to be decidable. However, the remarkable undecidability results for asynchronous games with simple objectives such as local reachability or termination or deadlock-freeness, even in systems with six processes, from [Gimbert, 2022] highlights the complexity of distributed synthesis problems.

Petri games [Finkbeiner and Olderog, 2017] are distributed games defined on Petri nets, where places are divided into *system* and *environment* places, and the players are represented by tokens. In Petri nets, causality is modeled by the flow of tokens; hence tokens naturally serve as carriers of information. While players in concurrent places have no knowledge of each other, synchronization at a joint transition allows them to exchange information: each participant learns the causal history of the others, i.e., all places and transitions on which the joint transition depends.

Strategies in Petri games are defined on the *unfolding* of the net. Unfoldings separate places reached through different causal histories into distinct copies, so that whenever a place is reachable by two distinct paths, it is split. A strategy then prescribes which transitions system places may refuse, based on the causal histories revealed at synchronizations. In this way,

strategies describe consistent fragments of possible behaviors of the Petri net.

Several classes of Petri games have been shown decidable: in particular, games with a bounded number of system players against one environment player [Finkbeiner and Olderog, 2017], and games with one system player against a bounded number of environment players [Finkbeiner and Gölz, 2018]. However, later work [Finkbeiner et al., 2021] proved undecidability for Petri games with global winning conditions, even with only two system players and one environment player. The undecidability arises because global conditions can enforce specific linearizations of parallel runs, which suffices to encode the Post Correspondence Problem. Further work [Beutner et al., 2019] also establishes formal connections with control games.

Contributions

In this thesis, we introduce the framework of *asynchronous transition system (ATS) games* as a formal model for studying distributed synthesis in concurrent systems. An ATS game, denoted $G = (A, s_0, Win)$, consists of an *asynchronous transition system* A, which specifies local states and transitions for distributed processes over a distributed alphabet Σ; an *initial global state* s_0; and a *winning condition* Win, describing the objective of the system (for example, safety or reachability). A *play* is an interleaved sequence of environment and system moves. The environment acts as a scheduler of actions without revealing any information about previous scheduling events. The processes comprising the system, on the other hand, have only partial information and must respond by choosing an enabled transition based solely on their collective causal past. For instance, under a *global safety objective*, the system wins if no unsafe global state is ever reached during a play.

A *distributed strategy* is formalized as a partial function mapping each partial play to a global state, thereby advising processes on how to respond to the environment's actions based on their *causal* history. A strategy is winning if all maximal plays conforming to it satisfy the winning condition. The work introduces *memory automata* to represent *finite-state distributed strategies*. This provides a concrete mechanism for encoding causal dependencies. These automata deterministically specify responses to environment moves and update local memory states accordingly. The formal definition of ATS games and the basic framework, including the

concepts of plays and strategies, are developed in Chapter 2.

Mutual exclusion protocols are interesting even in the two-process setting, where they admit a non-trivial solution. Motivated by this, we explore various winning conditions in the two process setting. We also study *CDM games*, where a single process participates in and thereby controls all non-deterministic choices. This model has practical relevance for distributed systems, with version control systems (e.g., SVN) serving as a notable example. Finite-state distributed strategies are particularly significant in both two-process games and *Central Decision Maker (CDM) games*, as we provide their algorithmic construction for the winning conditions we address.

The study establishes several results for two-process ATS games, where concurrency makes the problem more complex than in single-process games (which align with classical graph games). For safety, local reachability, and global reachability objectives, memoryless strategies are insufficient. Nevertheless, finite memory suffices: for safety and local reachability, processes need only store the last global state from synchronization, while for global reachability, processes must additionally remember all local states visited since the last synchronization. Deciding the existence of winning strategies is *NP-complete* for safety and local reachability, and *NEXPTIME-solvable* (and *PSPACE-hard*) for global reachability. These results are developed in Chapter 3.

Our study also analyzes *CDM games*, where a single process controls all non-deterministic choices, while the other processes propagate causal information. Despite centralization, all processes remain active participants by synchronizing and exchanging information. It is shown that winning strategies for safety and parity objectives are *EXPTIME-complete*. All processes track their best knowledge of the global state and update it during synchronizations via a *gossip automaton*. We also show that optimal memory automata for the decision maker do require exponential local memory for at least the CDM process. Our work demonstrates how sequential strategies from associated sequential games can be systematically transformed into distributed strategies for CDM games, with a focus on the existence of winning strategies under global safety and local parity conditions. This transformation critically depends on selecting a suitable linearization of finite traces that capture the causal past. These results are presented in Chapter 4.

The introduction of *multiple decision makers* fundamentally complicates ATS games: this

thesis proves that ATS games with two decision makers are *undecidable*, even for basic safety objectives. The proof adapts techniques from asynchronous control games, showing that concurrency in decision-making inherently leads to undecidability. These results are presented in the later sections of Chapter 4.

A further contribution is the demonstration that ATS games are equivalent in expressive power to *control games*, where actions are partitioned into controllable and uncontrollable sets. The equivalence is established by mutual transformations. In fact, for any state-based winning condition, ATS games and control games are equivalent in power. More precisely, starting from an ATS game, we can build a control game with the same number of processes and a comparable number of actions, and vice versa. Given a control game to construct an equivalent ATS game, we introduce nondeterministic local choice actions that let each process select a subset of controllable actions as there next state. Given an ATS game an equivalent control game is constructed by modeling environment moves as uncontrollable actions. To establish the equivalence we prove that, a winning strategy exists in the control game if and only if a winning strategy exists in the corresponding ATS game and vice versa. This equivalence implies that undecidability results for control games extend to ATS games, and also that techniques and complexity results can transfer between the two frameworks, unifying distributed synthesis approaches. This equivalence is established in Chapter 5.

Chapter 2

The ATS game model

In this chapter, we propose our model called *asynchronous transition system games (ATS games)*. These games are played on a non-deterministic asynchronous transition system involving two entities: a distributed system composed of cooperating processes and an environment. A play is an interleaved sequence of *environment* and *system* moves.

The environment is a singular entity and manifests itself in the form of a *scheduler* of actions. It is very important to keep in mind that the environment is simply a scheduler and *does not reveal* any information about prior scheduling events. The system responds to this scheduling of actions by selecting enabled transitions based on the observed history.

To formally capture concurrent behavior, we rely on Mazurkiewicz traces [Diekert and Rozenberg, 1995; Mukund, 2012], which model causality and concurrency among events distributed across multiple processes. These traces, together with the notion of asynchronous transition systems that we describe in the coming section, form the basis of our model.

In the section that follows, we define ATS games and describe the interleaved semantics of a play. We illustrate the model with an example, and introduce the notion of a system strategy as a function from past history to future moves. We also formalize finite-state distributed strategies,

represented by memory automata. Finally, we state the objective of the game: deciding whether there exists a winning strategy for the system, and describe one when it exists—topics that are explored in detail for specific instances in the subsequent chapters.

2.1 Preliminaries and conceptual framework

In this section, we set up the basic notations about (Mazurkiewicz) traces (see [Diekert and Rozenberg, 1995]). A trace is a well-established model of a concurrent behaviour in which the causality and concurrency information between events (occurrences of actions, distributed across a set of processes) is captured as a labelled partial order. *Asynchronous transition systems ATS*, proposed by Zielonka, are finite-state distributed devices that operate on these traces.

2.1.1 Introduction to traces

Let $\mathscr{P}$ be a finite non-empty set of processes. We let i range over $\mathscr{P}$ and use $\{X_i\}$ to denote a $\mathscr{P}$-indexed family $\{X_i\}_{i\in\mathscr{P}}$. A *distributed alphabet* over $\mathscr{P}$ is a family $\widetilde{\Sigma} = \{\Sigma_i\}$ of finite sets. Let $\Sigma = \bigcup_{i\in\mathscr{P}} \Sigma_i$ be the total alphabet, that is, the set of all actions. For $a \in \Sigma$, we set $\mathrm{loc}(a) = \{i \in \mathscr{P} \mid a \in \Sigma_i\}$. Thus $\mathrm{loc}(a)$ is the set of processes which participate in the action a. We let $I = \{(a,b) \in \Sigma\times\Sigma \mid \mathrm{loc}(a)\cap\mathrm{loc}(b) = \emptyset\}$ and $D = (\Sigma\times\Sigma)\setminus I$ be the induced *independence* and *dependence* binary relations on Σ respectively. Let $(X,\leq)$ be a poset. Then, for a subset $Y \subseteq X$, the *down-set* of Y is $\downarrow Y = \{x \in X \mid \exists y \in Y : x \leq y\}$. For a single element $x \in X$, we set $\downarrow x = \downarrow\{x\}$ and $\Downarrow x = \downarrow x \setminus \{x\}$.

Definition 2.1. *A trace over $\widetilde{\Sigma}$ is a Σ-labelled poset $t = (E,\leq,\lambda)$ where,*

- *E is a (possibly infinite) set of* events *and $\lambda : E \to \Sigma$ is a labelling function.*
- *$\leq$ is a partial order on E such that*
  - *for each $e,e' \in E$, $(\lambda(e),\lambda(e')) \in D$ implies $e \leq e'$ or $e' \leq e$.*
  - *for each $e,e' \in E$, $e \lessdot e'$ implies $(\lambda(e),\lambda(e')) \in D$, where $e \lessdot e'$ if $e < e'$ and for each e'' with $e \leq e'' \leq e'$, either $e'' = e$ or $e'' = e'$.*

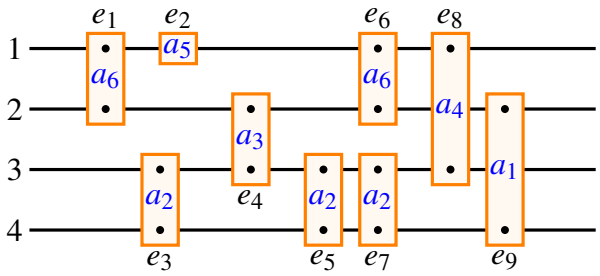


Figure 2.2: A trace with labelled events: There are 4 processes. A process line represents the sequence of actions in which the corresponding process participates. Each event is represented by a rectangle with its tied action inside.

– *for each* $e \in E, \downarrow e$ *is finite where* $\downarrow e = \{f \in E | f \leq e\}$.

Let $t = (E, \leq, \lambda)$ be a trace over $\widetilde{\Sigma}$. Recall that the elements of E are referred to as *events* in t and for an event e in t, $\mathrm{loc}(e)$ abbreviates $\mathrm{loc}(\lambda(e))$. Let $i \in \mathscr{P}$. The set of *i-events* in t is $E_i = \{e \in E \mid i \in \mathrm{loc}(e)\}$. This is the set of events in which process i participates. It is clear that E_i is totally ordered by $\leq$. We write $\mathrm{loc}(t) = \{i \in \mathscr{P} \mid E_i \neq \emptyset\}$ to denote the set of processes which participate in *some* event in t. It is easy to see that $\mathrm{loc}(t) = \cup_{e \in E} \mathrm{loc}(e)$. Further, $\omega\mathrm{loc}(t) = \{i \in \mathscr{P} \mid E_i \text{ is infinite}\}$ denotes the set of of processes which participate in infinitely many events in t.

Example 2.3. *A trace with 9 events* $\{e_1, e_2, \ldots\}$ *over 4 processes with* $\mathscr{P} = \{1,2,3,4\}$ *is shown in Fig. 2.2 over a distributed alphabet* $\tilde{\Sigma} = \{\Sigma_i\}_{i \in \mathscr{P}}$. *Here* $\Sigma_1 = \{a_4, a_5, a_6\}$ *and* $\Sigma_2 = \{a_1, a_3, a_6\}$ *and so on. Each process is indicated by a horizontal line and time flows rightward. Each event is represented by a vertical box and is labelled by a letter in* $\Sigma = \{a_1, a_2, a_3, a_4, a_5, a_6\}$. *Some events such as* e_2 *are purely local to a single process. Dots in synchronizing events indicate the participating processes. It is easy to infer causality relation* ($\leq$) *looking at the diagram, for example,* $e_1 < e_4 < e_5 < e_7 < e_8$. *We can also infer which events are concurrent (not related by* $\leq$*): for example, event* e_6 *is concurrent to events* e_5, e_7, *while the pairs of events* e_4, e_5 *and* e_1, e_4 *are causally ordered by process* 3 *and process* 2, *respectively.*

Let $\mathrm{TR}(\widetilde{\Sigma})$ denote the set of all traces over $\widetilde{\Sigma}$. As the distribution of Σ across processes will be clear from the context, we use the shorthand Σ instead of $\widetilde{\Sigma}$. So we use $\mathrm{TR}(\Sigma)$ instead of $\mathrm{TR}(\widetilde{\Sigma})$. Henceforth a trace means a trace over $\widetilde{\Sigma}$ unless mentioned otherwise and we further abbreviate $\mathrm{TR}(\Sigma)$ by TR. A trace is finite if the underlying set of events is finite. We use $\mathrm{TR}^*(\Sigma)$ and $\mathrm{TR}^\omega(\Sigma)$ (or simply TR^* and TR^ω) to denote the set of all finite and infinite traces.

Now we describe the important operation of concatenation of traces. Let $t = (E, \leq, \lambda)$ and $t' = (E', \leq', \lambda')$ be finite traces, that is, elements of TR^*. We define $tt' \in \mathrm{TR}^*$ to be the trace $(E'', \leq'', \lambda'')$ where

- $E'' = E \cup E'$,
- $\leq''$ is the transitive closure of $\leq \cup \leq' \cup \{(e, e') \in E \times E' \mid (\lambda(e), \lambda'(e')) \in D\}$,
- $\lambda'' \colon E'' \to \Sigma$ where $\lambda''(e) = \lambda(e)$ if $e \in E$; otherwise, $\lambda''(e) = \lambda'(e)$.

This operation of *trace concatenation* gives TR^* a monoid structure. The empty trace ε denotes the identity of the monoid TR^*. Observe that, with a (resp. b) denoting the singleton trace whose only event is labelled a (resp. b), if $(a,b) \in I$ then $ab = ba$ in TR^*.

The trace concatenation operation on finite traces can also be defined for infinite traces under certain conditions. We explain this now. The above mentioned definition of tt' results in a valid trace if t is finite. That is, if $t \in \mathrm{TR}^*$ and $t' \in \mathrm{TR}$ then $tt' \in \mathrm{TR}$. Moreover, if $t, t' \in \mathrm{TR}$ are such that $\omega\mathrm{loc}(t) \cap \mathrm{loc}(t') = \emptyset$, then $tt' \in \mathrm{TR}$. It is easy to see that, in this case, $\omega\mathrm{loc}(tt') = \omega\mathrm{loc}(t) \cup \omega\mathrm{loc}(t')$. We say that a trace t' is a *prefix* of a trace t if $t = t't''$ for some trace t''. It is important to observe that this definition permits a prefix to be infinite.

Configuration We now come to the very important notion of a configuration of a trace $t = (E, \leq, \lambda)$. Recall that for $X \subseteq E$, we let $\downarrow X = \{y \in E \mid y \leq x \text{ for some } x \in X\}$. For $e \in E$, $\downarrow e = \downarrow\{e\}$ is the causal past of e and $\Downarrow e = \downarrow e \setminus \{e\}$ is the strict causal past of e. A subset $c \subseteq E$ is a *configuration* of t if c is *finite* and $\downarrow c = c$. Note that, if we restrict the trace t to a configuration c, we get another *finite* trace $t_c = (c, \leq, \lambda)$ which turns out to be a *prefix* of t. Conversely, given a finite prefix t' of t, we can identify a configuration c of t such that $t' = t_c$. In this sense, configurations of t are essentially finite prefixes of t. Examples of configurations (or, equivalently, finite prefixes) are the empty set $\emptyset$ (corresponding to trace ε), $\downarrow e$ or $\Downarrow e$, for every event $e \in E$. Note that E itself is a configuration of t iff t is finite.

We let C_t denote the set of all configurations of t. The *action based successor relation* $\to_t \subseteq C_t \times \Sigma \times C_t$ is defined by $c \xrightarrow{a}_t c'$ if and only if there exists $e \in E$ such that $e \notin c$, $c \cup \{e\} = c'$ and $\lambda(e) = a$.

Let $c \in C_t$ and $P \subseteq \mathscr{P}$. The P-view of c is the set of all events that processes in P have seen in their past in c. Formally, the i-view of c is defined as $\partial_i(c) = \downarrow(E_i \cap c)$, where recall that $E_i = \{e \in E \mid i \in \mathrm{loc}(e)\}$, and the P-view is $\partial_P(c) = \bigcup_{i \in P} \partial_i(c)$. Then, $\partial_P(c)$ is itself a configuration, representing the collective knowledge of the processes in P about c. Moreover, for $i \in P$, if $\partial_i(c) \neq \emptyset$ then there exists $e \in E_i$ such that $\partial_i(c) = \downarrow e$. We use $\max_i(c)$ to denote this event. Finally, a finite trace $t = (E, \leq, \lambda)$ is said to be *prime* if it has a unique maximum event, that is, there exists $e \in E$ such that $E = \downarrow e$. We let $\mathrm{last}(t)$ denote this maximum event of t.

Example 2.4. *For the trace t in Fig. 2.2, $c = \{e_1, e_3, e_4\}$ is a configuration and so is $c' = \{e_1, e_2, e_3, e_4\}$. Further, the subsets $c_1 = \downarrow e_5 = \{e_1, e_3, e_4, e_5\}$ and $c_2 = \downarrow e_6 = \{e_1, e_2, e_3, e_4, e_6\}$ are also configurations of t. Observe that $c \xrightarrow{a_5}_t c'$ and $c \xrightarrow{a_2}_t c_1$. Note that the set $\{e_1, e_2, e_4\}$ is not a configuration as $\downarrow\{e_1, e_2, e_4\} = \{e_1, e_2, e_3, e_4\}$.*

2.1.2 Asynchronous transition systems

Asynchronous automata and the related transition systems are fundamental finite-state distributed devices due to [Zielonka, 1987] which operate/run on traces.

We now recall the notion of an asynchronous transition system. We equip each process $i \in \mathscr{P}$ with a finite non-empty set S_i of local i-states. Further, we set $S_{\mathscr{P}} = \prod_{i \in \mathscr{P}} S_i$ and call it the set of global states. For a non-empty set $P \subseteq \mathscr{P}$ of processes, $S_P = \prod_{i \in P} S_i$ is the set of (joint) P-states. For a P-state $s \in S_P$ and $i \in P$, $s(i) \in S_i$ denotes the local i-state in the P-tuple s. More generally, for $Q \subseteq P$ and $s \in S_P$, we denote by s_Q the projection of s on Q – that is, s_Q is the unique Q-state such that $s_Q(i) = s(i)$ for all $i \in Q$. For $a \in \Sigma$, we use a to abbreviate $\mathrm{loc}(a)$ when talking about states. Thus $S_a = S_{\mathrm{loc}(a)}$ denotes the set of all a-states and if $\mathrm{loc}(a) \subseteq P$ and s is a P-state, we write s_a for $s_{\mathrm{loc}(a)}$.

Similar to the convention of writing a $\mathscr{P}$-indexed family as $\{X_i\}$, we will follow the convention of writing $\{Y_a\}$ to denote the Σ-indexed family $\{Y_a\}_{a \in \Sigma}$.

Definition 2.5. *An asynchronous transition system (ATS) over $\widetilde{\Sigma}$ is a tuple $A = (\{S_i\}, \{\xrightarrow{a}\})$ where,*

- *For each process $i \in \mathscr{P}$, S_i is a finite non-empty set of local i-states.*
- *For each action $a \in \Sigma$, $\xrightarrow{a} \subseteq S_a \times S_a$ is a* non-deterministic *transition relation on a-states.*

Let $A = (\{S_i\}, \{\xrightarrow{a}\})$ be an ATS. Note that a transition in A on an action a is *local* in the sense that it involves only processes in $\mathrm{loc}(a)$ and a-states. An action a is *enabled* in an a-state s_a if there exists a state s'_a such that $(s_a, s'_a) \in \xrightarrow{a}$. We extend these local transition relations naturally to global states. More precisely, for $a \in \Sigma$, we define $\overset{a}{\Rightarrow} \subseteq S_{\mathscr{P}} \times S_{\mathscr{P}}$ as follows: for $s, s' \in S_{\mathscr{P}}$, $(s, s') \in \overset{a}{\Rightarrow}$ if $(s_a, s'_a) \in \xrightarrow{a}$ and $s_{\mathscr{P}\setminus \mathrm{loc}(a)} = s'_{\mathscr{P}\setminus \mathrm{loc}(a)}$. We use notation $s_a \xrightarrow{a} s'_a$ for $(s_a, s'_a) \in \xrightarrow{a}$ and $s \overset{a}{\Rightarrow} s'$ for $(s, s') \in \overset{a}{\Rightarrow}$. We say that action a is enabled in global state s if there exists a state s' such that $s \overset{a}{\Rightarrow} s'$. Notice that the locality of actions is conserved. When an action a occurs, only participating processes can change their states. We call this locality of action. We emphasize this for future use.

Lemma 2.6 (locality of action)**.** *Given an ATS $A = (\{S_i\}, \{\xrightarrow{a}\})$ let action a be enabled at s, b be an action such that aIb and $s \overset{a}{\Rightarrow} r$. Let $s' \in S_{\mathscr{P}}$ be such that $s'_a = s_a$. Then $\exists r'$ such that $s' \overset{a}{\Rightarrow} r'$ and $r'_a = r_a$ and $s'_{\mathscr{P}\setminus \mathrm{loc}(a)} = r'_{\mathscr{P}\setminus \mathrm{loc}(a)}$. Moreover, if b is enabled at s, then it is also enabled at r.*

Now we define the important notion of a *run of an ATS A* on a trace t over the alphabet $\widetilde{\Sigma}$.

Definition 2.7. *Let us fix an initial global state $s_0 \in S_{\mathscr{P}}$. A run of A on $t \in \mathrm{TR}$, starting from s_0, is a map $\rho : C_t \to S_{\mathscr{P}}$ such that $\rho(\emptyset) = s_0$ and for every $c, c' \in C_t$, $c \xrightarrow{a}_t c'$ implies that $\rho(c) \overset{a}{\Rightarrow} \rho(c')$.*

Note that, as A is non-deterministic, there can be multiple runs of A on the same trace. The next lemma states that any run is determined by its effect on the prime traces in its domain. It is an easy consequence of the locality of transitions of A.

Lemma 2.8. *Let $t = (E, \leq, \lambda)$ be a trace and $\rho : C_t \to S_{\mathscr{P}}$ be a run of A on t starting at s_0. Then ρ is determined by $\rho(\emptyset)$ and $\rho(\downarrow e)$ for each $e \in E$. Moreover, for any trace t, $\forall i \in \mathscr{P}$: $\rho(t)_i = \rho(\partial_i(t))_i$.*

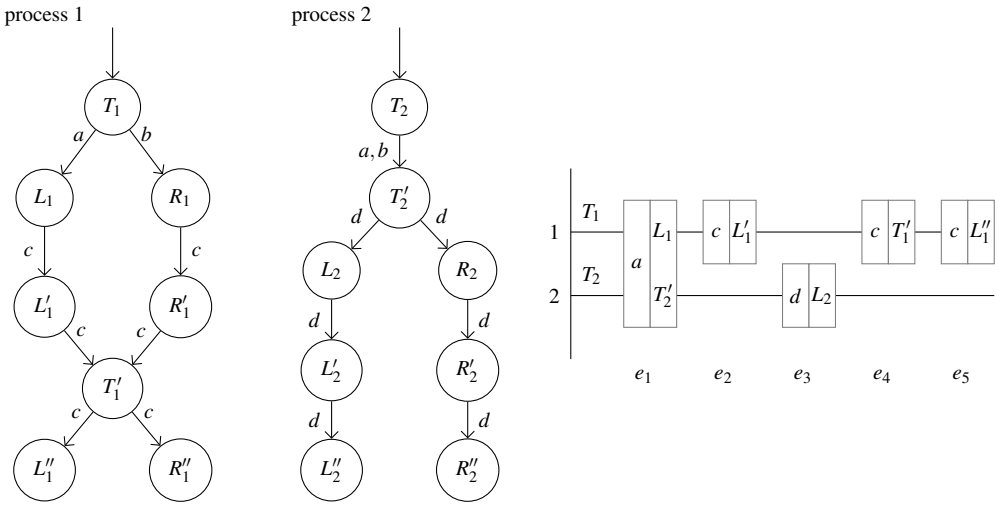
Proof. We use induction to prove that given a trace t, $\forall c \in C_t \forall c_i = \partial_i(c) : \rho(c)_i = \rho(c_i)_i$. We induct over the size of c', where $c = c_i.c'$. For the base case when $c' = \emptyset$ then $c = c_i$. Therefore $\rho(c)_i = \rho(c_i)_i$.

Let $c' = a.c''$. Then, $c_i \xrightarrow{a}_t c_i.a$. Let $\rho(c) = s, \rho(c_i) = s'$ and $\rho(c_i.a) = s''$. Due to locality of action Lemma 2.6 as $s' \stackrel{a}{\Rightarrow} s''$ and $i \notin \mathrm{loc}(a)$ we have $s'_i = s''_i$. By inductive hypothesis we know that as $|c''| < |c'|$ and therefore, $\rho(c)_i = \rho(c_i a)_i$ i.e. $s_i = s''_i$. Thus, $s_i = s''_i = s'_i$. That is, $\rho(c)_i = \rho(c_i)_i$

This completes the proof of the lemma. It is easy to see that,

$$s(i) = s_0(i) \text{ if } c \cap E_i = \emptyset$$
$$= s'(i) \text{ if } s' = \rho(\downarrow e_i) \text{ where } e_i = \max_i(c) \text{ i.e. } e_i \text{ is the maximum event in } \partial_i(c)$$

□



(a) (b)

Figure 2.10: *Example of ATS and runs*

Example 2.9. *We consider an example illustrated in Fig. 2.10(a). The set of processes* $\mathscr{P} = \{1,2\}$ *and set of actions* $\widetilde{\Sigma}$ *is given by* $\Sigma_1 = \{a,b,c\}$, $\Sigma_2 = \{a,b,d\}$. *Thus c and d are purely local actions. The local state sets are* $S_1 = \{T_1, L_1, \ldots\}$ *and* $S_2 = \{T_2, T'_2, \ldots\}$. *The only transitions on joint actions* a,b *are* $(T_1,T_2) \xrightarrow{a} (L_1,T'_2)$ *and* $(T_1,T_2) \xrightarrow{b} (R_1,T'_2)$. *The purely local transitions are clear from the figure. For instance, we have* $T'_1 \xrightarrow{c} L''_1$, $T'_1 \xrightarrow{c} R''_1$ *and* $T'_2 \xrightarrow{d} L_2$.

We look at a run beginning at initial state (T_1,T_2). *This run is illustrated in Fig. 2.10(b). The first action to be in the run is the synchronous action a causing the processes* 1 *and* 2 *to transition deterministically to* (L_1,T'_2). *Scheduling of an action and change of state in partic-*

ipating processes characterizes each event. An event is labeled by this pair of information in the figure. Change of state of participating processes gives the information of which processes are participating. Due to the above Lemma 2.8 an event is mapped to the new states of only the participating process in figures of runs.

Next, on action c, process 1 *deterministically moves to* L_1' *after event* e_2*. Either action c or d can be occur in state* (L_1', T_2')*. Suppose the next action is d. Process* 2 *state can change to* L_2 *or* R_2*. Say it changes to* L_2*. Then on c, process* 1 *goes to* T_1'*. Next in this run another c occurs and* 1*'s next state can be* L_1'' *or* R_1''*. The subsequent sequence generated from this run is-* $(T_1, T_2)a(L_1, T_2')c(L_1', T_2')d(L_1', L_2)c(T_1', L_2)c(L_1'', L_2)$.

Note that the actions c and d are independent and hence event e_3 *is concurrent with events* e_2, e_4, e_5*. Another sequence of interleaved actions and states that represents the same run is-* $(T_1, T_2)a(L_1, T_2')c(L_1', T_2')c(T_1', T_2')d(T_1', L_2)c(L_1'', L_2)$.

As e_3 *is concurrent to* e_4 *and* e_5 *this event occurring later in the sequence results in the same run. Note that the sequential semantics in both cases is different. As the events in these runs are only partially ordered we use traces to represent such an run. .*

2.2 ATS game

Building on the notions of Mazurkiewicz traces and asynchronous transition systems discussed above, we now introduce our model, called **a**synchronous **t**ransition **s**ystem games (ATS games), which are played between a system and an environment.

We continue with the notation from the previous section. In particular, we fix a distributed alphabet $\widetilde{\Sigma}$ over a fixed team $\mathscr{P}$ of processes.

Definition 2.11. *An ATS game is of the form* $\mathscr{G} = (A, s_0, \mathrm{Win})$ *where*

- $A = (\{S_i\}, \{\xrightarrow{a}\})$ *is an asynchronous transition system over* $\widetilde{\Sigma}$*.*
- $s_0 \in S_{\mathscr{P}}$ *is an initial global state of A.*
- Win *is a specification of the winning condition.*

2.2.1 Play of a game: an interleaved semantics

A play of an ATS game corresponds to an ongoing and possibly nonterminating interaction between the distributed system and the environment. Now we describe the key notion of a *distributed play*. Towards this, let us fix an ATS game $\mathscr{G} = (A, s_0, \mathrm{Win})$ with $A = (\{S_i\}, \{\stackrel{a}{\rightarrow}\})$.

We begin with a description of the interleaved semantics of a distributed play. Such a play begins in the initial global state s_0. Environment makes the first move by *scheduling* an action (say a) which is enabled at s_0. The processes participating *in* a respond by selecting an available a-transition and thus advancing the 'current' global state (say, to s). At this point, it is environment's turn to schedule another action (say b) which is enabled at s. The processes participating *in* b respond by choosing a suitable b-transition at s. Let us assume that a and b are independent actions. Then the processes participating in b are oblivious to the 'concurrent' scheduling of 'prior' a-action. It is important to observe that, in this situation, thanks to the locality of transitions in A, b is also enabled at s_0 and environment also had the option of scheduling b first and then a later. On the other hand, if a and b are dependent, then there is a process which participates in both a and b and every process participating in b *comes to know* about the prior a-action which is "causally" before the current b-action.

So, in this interleaved semantics of a distributed play, a play consists of an alternate sequence of moves of the environment (which schedules actions) and the distributed system (where the participating processes respond by matching transitions).

Observe that each process in the distributed system has only partial information; among all the actions which are already scheduled, it only *knows* those which are in its *causal past*. The causal past of a process contains the information that a process comes to know directly or indirectly through its interactions with the other processes. Further when two or more processes interact (by participating in a shared/joint action), they exchange their complete causal pasts.

A (partial) distributed play of $\mathscr{G}$ is accurately modelled as a tuple (t, ρ) where $t = (E, \leq, \lambda) \in \mathrm{TR}(\Sigma)$ is a trace over $\widetilde{\Sigma}$ and $\rho : C_t \rightarrow S_{\mathscr{P}}$ is a run of A on t starting at s_0. The idea is that the (labelled) events in t represent the scheduler/environment choices and the partial-order $\leq$ captures the causality between these events arising out of the *distribution* of these events across processes. The fact that an event e is labelled a (that is, $\lambda(e) = a$) means that the environment

scheduled the shared action a which lead the processes participating in a to synchronize. Their collective *causal* past-information at e corresponds to events in $\Downarrow e$. Based on this common information, these processes respond to e by choosing a local transition on a which advances their prior joint a-state s_a to s'_a where $s = \rho(\Downarrow e)$ and $s' = \rho(\downarrow e)$. Recall that $\Downarrow e = (\downarrow e) \setminus \{e\}$. Thus the run ρ correctly captures the decisions of the distributed team of processes during the play. In summary, the event e models environment's scheduling of action $\lambda(e)$ at the global state $\rho(\Downarrow e)$; and the local $\mathrm{loc}(e)$-transition from $\rho(\Downarrow e)$ to $\rho(\downarrow e)$ models the collective response of the processes participating in e. Thanks to locality of transitions of A, the response of the distributed team to a purely local action *does not* change the local state of the other process, and is viewed as the response of the action owner process.

Definition 2.12. *A (partial and distributed) play of the game $\mathscr{G}$ is a tuple (t, ρ) where*

- *$t = (E, \leq, \lambda)$ is a trace over $\widetilde{\Sigma}$.*
- *$\rho : C_t \to S_{\mathscr{P}}$ is a run of A on t starting at s_0*

A play (t', ρ') *extends* a (possibly infinite) play (t, ρ) if t is a proper prefix of t' and ρ is the restriction of ρ' to t. A (distributed) play (t, ρ) is said to be maximal if it cannot be extended further by any play. Note that if (t, ρ) is maximal then the processes P which have moved only finitely often in t can not be further scheduled by the environment as no action involving *only them* is enabled in $\rho(\partial_P(t))_P$. It is important to note that a maximal play could be either finite or infinite. If a maximal play $(t = (E, \leq, \lambda), \rho)$ is finite, then no action is enabled at the final global state $\rho(E)$.

2.2.2 Winning condition: An example

We next turn our attention to the winning condition Win which specifies the winner of a maximal play. At an abstract level, Win is simply the collection of all maximal plays in which the distributed system wins. As an example, consider a *global safety* winning condition described using a set $F \subset S_{\mathscr{P}}$ of *global safe states*. A maximal play (t, ρ) is won by the distributed team (or satisfies the global safety condition) if for all $c \in C_t$, $\rho(c) \in F$. In other words, to win a maximal play the distributed team needs to ensure that a global unsafe state *never* occurs.

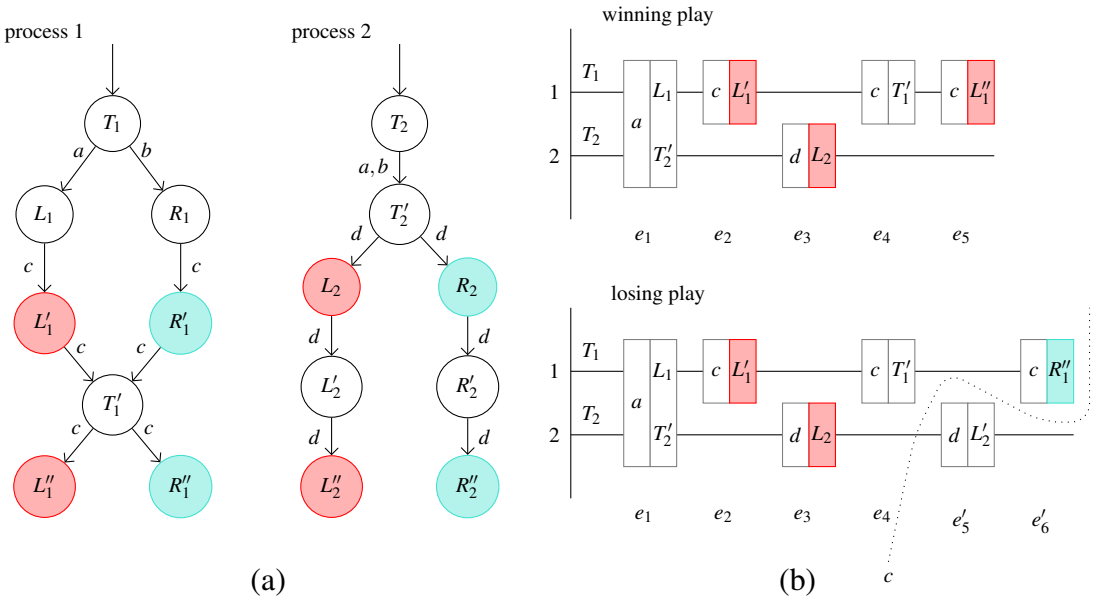


Figure 2.14: An ATS game: unsafe state $= \{L_1', L_1''\} \times \{R_2, R_2''\} \cup \{R_1', R_1''\} \times \{L_2, L_2''\}$

Example 2.13. *We extend the example Example 2.9. In this game the initial state is* (T_1, T_2). *The only states not in* F *are* $\{L_1', L_1''\} \times \{R_2, R_2''\} \cup \{R_1', R_1''\} \times \{L_2, L_2''\}$. *This is illustrated in Fig. 2.14(a). In other words, the set* F *of global safe states is the complement of the unsafe set.*

Fig. 2.14(b) shows some partial plays of the game. In the first play the environment begins by playing a synchronous action a *on* (T_1, T_2), *causing the processes* 1 *and* 2 *to move deterministically to* (L_1, T_2'). *Next, the environment plays* c, *and process* 1 *deterministically moves to* L_1' *after event* e_2. *The environment can then play* c *or* d *in* (L_1', T_2'). *Suppose it plays* d; *process* 2 *must then choose between* L_2 *and* R_2, *relying only on its causal past* $\{e_1\}$.

Given this past, process 2 *can predicts or strategizes that process* 1 *would move from* L_1 *to* L_1', *so it selects* L_2 *to avoid* (L_1', R_2). *The environment then plays* c, *and the system transitions to* T_1'. *When the environment plays* c *again, process* 1 *must choose between* L_1'' *and* R_1'' *without knowing if process* 2 *went to* L_2 *(leading to* L_2''*) or* R_2 *(leading to* R_2''*). At this stage, process* 1 *can recall from its causal past* $\{e_1, e_2, e_4\}$ *that process* 2 *would go to* L_2 *and then to* L_2'' *to correctly choose* L_1''.

In the second play, no global state encountered in the sequence of events $e_1e_2e_3e_4e_5'e_6'$ *is in the unsafe set. But the configuration* $c_3 = \{e_1, e_2, e_3, e_4, e_6'\}$ *in this play is in state* (R_1'', L_2) *and therefore the system loses the play. As* e_5' *and* e_6' *are independent, the sequence of events* $e_5'e_6'$ *is equivalent to* $e_6'e_5'$.

2.2.3 A distributed strategy

Now we are ready to define the important and crucial notion of a distributed strategy. Intuitively speaking, a distributed strategy is an advice function that the team of processes in the distributed system uses to respond to the environment's actions. More importantly, this response can only depend on the collective causal history of the processes participating in this action. Further, the advice function does not restrict the choices available to the environment in all situations.

Definition 2.15. *A distributed strategy in $\mathscr{G}$ is a partial function $\sigma : \mathrm{TR}^* \to S_{\mathscr{P}}$ with the smallest domain such that*

- *the domain of σ is prefix closed, that is, if $t \in \mathrm{TR}^*$ is such that $\sigma(t)$ is defined and $t' \in \mathrm{TR}^*$ is a prefix of t then $\sigma(t')$ is also defined.*
- *$\sigma(\varepsilon) = s_0$ where ε denotes the empty trace.*
- *for every $t \in \mathrm{TR}^*$ if $\sigma(t)$ is defined and a is enabled at $\sigma(t)$, then $\sigma(ta)$ is also defined, and $\sigma(t) \stackrel{a}{\Rightarrow} \sigma(ta)$ (recall that ta denotes the trace concatenation of t with the singleton trace a).*

By requiring the domain to be the smallest one satisfying these conditions, we ensure that σ is defined exactly on those traces that can be generated by iteratively extending the empty trace along enabled actions. In particular, this implies that the domain of σ is prefix-closed: if $\sigma(t)$ is defined, then it must also be defined for all prefixes of t. In view of this, the first condition in the above definition is redundant.

The next lemma states that a distributed strategy is determined by its effect on the prime traces in its domain. It is a direct consequence of Lemma 2.8.

Lemma 2.16. *Let $\sigma : \mathrm{TR}^* \to S_{\mathscr{P}}$ be a distributed strategy in $\mathscr{G}$. Then σ is completely determined by $\sigma(\varepsilon)$ and $\sigma(t)$ for every prime trace t in the domain of σ. Moreover, for any trace t, $\forall i \in \mathscr{P} : \sigma(t)_i = \sigma(\partial_i(t))_i$.*

Definition 2.17. *Let $\sigma : \mathrm{TR}^* \to S_{\mathscr{P}}$ be a distributed strategy in $\mathscr{G}$. A distributed play $(t, \rho : C_t \to S_{\mathscr{P}})$ of $\mathscr{G}$ is said to* conform *σ if, for all $c \in C_t$, $\rho(c) = \sigma(c)$. Recall that, for a configuration c of t, by a slight abuse of notation, c also denotes the finite trace $t_c = (c, \leq, \lambda)$.*

A distributed strategy $\sigma : \mathrm{TR}^* \to S_{\mathscr{P}}$ *is winning in* $\mathscr{G}$ *if all maximal plays conforming it belong to* Win. *In other words, all maximal plays where the distributed team employs* σ *are won by the distributed system.*

Example 2.18. *We revisit the global safety game from Fig. 2.14 and explained in Example 2.13. We intuitively explain a protocol illustrated in Fig. 2.19(a) for the system to win this game. Each process stores in its memory the state of process* 1 *after the* last synchronization in the past. *If the environment chooses a (resp. b) then it is* l_1 *(resp.* r_1*). With this bit of information, they can ensure the safety objective as follows. If the memory state is* l_1*, then both processes move* left *at their respective decision points. That is, process* 1 *at* T_1' *moves to* L_1'' *on c, and process* 2 *at* T_2' *moves to* L_2 *on d. In case the memory state is* r_1*, then both processes move* right. *It is easy to see that this is indeed a winning distributed strategy that can be realized by a memory automaton. Note that the choices at* T_1' *and* T_2' *are really made after the first synchronization at joint action a or b, after which the processes follow their own local plans.*

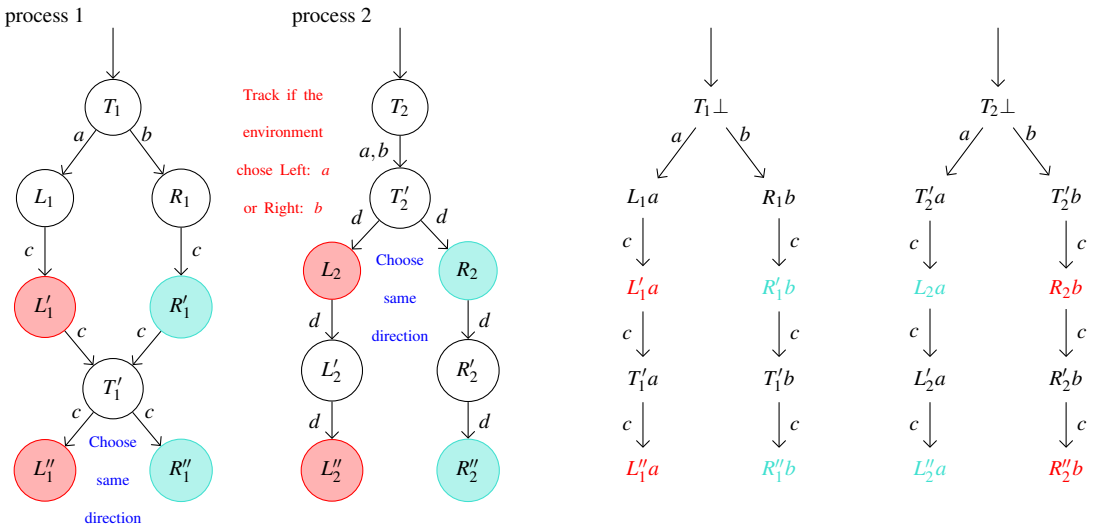


Figure 2.19: Strategy and corresponding memory automaton

Let us now introduce the notion of a finite-state distributed strategy. Let $\mathscr{G} = (A, s_0, \mathrm{Win})$ be an ATS game with $A = (\{S_i\}, \{\xrightarrow{a}\})$.

Definition 2.20. *A* memory automaton *for* $\mathscr{G}$ *is a* deterministic *asynchronous automaton* $\mathscr{A} = (\{S_i \times M_i\}, \{\delta_a : S_a \times M_a \to S_a \times M_a\}, (s_0, m_0))$ *over* $\widetilde{\Sigma}$ *such that*

- *For $a \in \Sigma$, $(s_a, m_a) \in S_a \times M_a = \Pi_{i \in \mathrm{loc}(a)}(S_i \times M_i)$, if a is enabled at s_a in A then a is also enabled at (s_a, m_a) in $\mathscr{A}$. Further, with $\delta_a((s_a, m_a)) = (s'_a, m'_a)$, $(s_a, s'_a) \in \xrightarrow{a}$; that is, there is an a-transition from s_a to s'_a in A.*

- *$(s_0, m_0) \in S_{\mathscr{P}} \times M_{\mathscr{P}}$ is a global state of $\mathscr{A}$ where s_0 is the initial global state of A.*

In the memory automaton $\mathscr{A}$, process i has access to local memory states M_i. The memory automaton is a distributed automaton which starts in the initial global state (s_0, m_0) and provides a deterministic response to *every* enabled scheduler action using the current joint memory-state. Further, it also updates the joint memory-state deterministically.

A memory automation is *zero-memory* if for each i, M_i is a singleton set. A zero-memory automaton may be viewed as a deterministic *asynchronous sub-automaton* of A.

We will refer to a finite-memory strategy automaton as simply a memory automaton. Intuitively, in the above definition of a memory automaton $\mathscr{A}$, the first condition says that the environment is not restricted, and the second condition says that the response (as well as the memory-update) of the memory automaton is deterministic.

Note that, as $\mathscr{A}$ is deterministic, if a trace t admits a run of $\mathscr{A}$ then the run is unique, say ρ. Specifically the unique run ρ of $\mathscr{A}$ on a trace $t \in \mathrm{TR}$ starting from initial state s_0 is a map $\rho : C_t \to S_{\mathscr{P}}$ given by $\rho(\emptyset) = s_0$, and for every $c, c' \in C_t$, $c \xrightarrow{a}_t c'$ implies that $\rho(c) \overset{a}{\Rightarrow} \rho(c')$ is the unique a transition in the a- state $\rho(c)_a$. By an abuse of notation, we use $\mathscr{A}(t)$ to denote $\rho(t)$ when the initial state is clear from context and $\mathscr{A}$ is deterministic. Thus, the memory automaton $\mathscr{A}$ gives rise to a natural distributed strategy as follows.

Definition 2.21. *Let $\mathscr{G}$ be a game and $\mathscr{A}$ be a memory automaton for $\mathscr{G}$. The memory automaton $\mathscr{A}$ naturally induces a function $\sigma_{\mathscr{A}} : \mathrm{TR}^* \to S_{\mathscr{P}}$ as follows: for trace t with a partial run in $\mathscr{A}$, $\sigma_{\mathscr{A}}(t) = s$ if $\mathscr{A}(t) = (s, m)$. Recall that $\mathscr{A}(t)$ denotes the final global state attained on the unique run of $\mathscr{A}$ on the finite trace t.*

We now show that the function $\sigma_{\mathscr{A}}$ is a distributed strategy.

Proposition 2.22. *Let $\mathscr{A} = (\{S_i \times M_i\}, \{\delta_a\}, (s_0, m_0))$ be a memory strategy in $\mathscr{G}$. Then $\sigma_{\mathscr{A}}$ is a distributed strategy in $\mathscr{G}$.*

Proof. Naturaly, the domain of $\mathscr{A}$ is prefix closed. Also $\sigma_{\mathscr{A}}(\varepsilon) = s_0$. Let a trace t be in domain of $\sigma_{\mathscr{A}}(ta)$. Say $\mathscr{A}(t) = (s, m)$. Then, by Definition 2.20 if a is enabled at (s_a, m_a), then there

is a unique $(s'_a, m'_a) \in S_a \times M_a$ such that $s_a \xrightarrow{a} s'_a$ is a transition of A and $\delta_a(s_a, m_a) = (s'_a, m'_a)$ is a transition of $\mathscr{A}$. Hence $\sigma_{\mathscr{A}}(ta)$ is also defined and $\sigma_{\mathscr{A}}(t) \stackrel{a}{\Rightarrow} \sigma_{\mathscr{A}}(ta)$. Moreover, as the a transition is uniquely defined in $\mathscr{A}$, the domain of $\mathscr{A}$ is the smallest that allows the above properties.

Therefore, by Definition 2.15 $\sigma_{\mathscr{A}}$ is a distributed strategy in $\mathscr{G}$. □

Definition 2.23. *Let $\mathscr{A} = (\{S_i \times M_i\}, \{\delta_a\}, (s_0, m_0))$ be a memory automaton. A play $(t, \rho : C_t \to S_{\mathscr{P}})$ of $\mathscr{G}$ is said to* conform *$\mathscr{A}$ if it conforms to $\sigma_{\mathscr{A}}$.*

Definition 2.24. *Let $\mathscr{A}$ be a memory automaton for $\mathscr{G}$ and $\sigma : \mathrm{TR}^* \to S_{\mathscr{P}}$ be a distributed strategy in $\mathscr{G}$. We say that σ is* realized by *$\mathscr{A}$ if, $\sigma = \sigma_{\mathscr{A}}$. A memory automaton is said to be winning if it realizes a winning distributed strategy.*

A distributed strategy σ in $\mathscr{G}$ is said to be finite-state *if there exists $\mathscr{A}$ which realizes it. Thus the responses of σ are determined by the finite-state device $\mathscr{A}$. A distributed strategy σ realized by a zero-memory automaton is called a* zero-memory distributed strategy.

Example 2.25. *We now formally present the winning strategy in Example 2.18 in the form of a memory automaton $\mathscr{A} = (\{S_i \times M_i\}, \{\delta_a\}, (s_0, m_0))$ illustrated in Fig. 2.19(b) as $M_1 = M_2 = \{\bot, a, b\}$ $m_0 = (\bot, \bot)$. $\delta_a(T_1, \bot, T_2, \bot) = (L_1, a, T'_2, a)$, $\delta_b(T_1, \bot, T_2, \bot) = (R_1, b, T'_2, b)$, $\delta_c(T'_1, a) = (L''_1, a)$, $\delta_c(T'_1, b) = (R''_1, b)$, $\delta_d(T'_2, a) = (L_2, a)$, $\delta_d(T'_2, b) = (R_2, b)$. All other transitions are deterministic in S_i and do not change the memory state and are thus well defined.*

2.2.4 The problem statement

In this section, we state the problem we aim to address in this work.

Given an ATS game $\mathscr{G} = (\mathscr{A}, s_0, \mathrm{Win})$, the central decidability question is to determine whether there exists a winning strategy for the system.

Beyond deciding the mere existence of such a strategy, we are also interested in the algorithmic problem of synthesis: constructing a finite-state winning strategy. In particular, can we synthesize a distributed memory automaton that realizes a winning strategy? This is motivated by practical considerations, since an explicit representation enables implementation in real distributed systems.

This page was intentionally left blank.

Chapter 3

Two processes ATS games and their analysis

In this chapter, we study ATS games with two processes, starting with the simpler case of a single process. Here, there is no concurrency, and the game is purely sequential—matching the setting of standard graph games. As a result, memoryless (zero-memory) fixpoint solutions are sufficient for simple winning conditions such as safety, reachability and parity. In the first section, we show that these games correspond directly to the classical two-player graph games of [Grädel et al., 2002; McNaughton, 1993], which are central to reactive synthesis.

This makes the case of two-process games particularly interesting, as the presence of concurrency fundamentally changes the landscape. Even with just two processes, non-trivial challenges arise due to the interplay between independent actions and synchronizations. A classical example is Peterson's mutual exclusion algorithm, which precisely illustrates this complexity. In this setting, mutual exclusion can be formulated as a global safety objective that requires the system to never reach a state where both processes are in the critical section simultaneously. At the same time, guaranteeing progress or fairness for each process—i.e., that each process will eventually enter its critical section—corresponds to repeatable local reachability

objectives. Thus, Peterson's algorithm exemplifies how even simple concurrent systems require reasoning about both global safety and local liveness.

We discuss two process games in the proceeding sections. We discuss strategies and algorithms for solving ATS games with two processes under various state-based winning conditions, highlighting both the decidability of the existence of winning strategies and the memory structures required to implement them.

3.1 ATS games with one process and full-information graph games

We present a general construction that translates one-process ATS games into standard full-information graph games, thereby enabling the direct application of classical solution techniques.

Formally, let $\mathscr{G} = (A, s_0, \mathrm{Win})$ be an ATS game with $A = (\{S_1\}, \{\xrightarrow{a}\})$. We construct an equivalent two-player full-information game $G_{\mathrm{graph}} = (A_{\mathrm{graph}}, s_0, \mathrm{Win}_{\mathrm{graph}})$, over a finite bipartite graph A_{graph}, with players Sys and Env.

Definition 3.1. *The game arena A_{graph} is a bipartite graph $A_{\mathrm{graph}} = (V_{\mathrm{env}}, V_{\mathrm{sys}}, \rightsquigarrow \subseteq V_{\mathrm{env}} \times V_{\mathrm{sys}} \cup V_{\mathrm{sys}} \times V_{\mathrm{env}})$, where:*

- *$V_{\mathrm{env}} = S_1$, and $V_{\mathrm{sys}} = \{(s,a) \in S_1 \times \Sigma \mid a$ is enabled at $s\}$.*
- *For $s \in V_{\mathrm{env}}$, $(s',a) \in V_{\mathrm{sys}}$, we have $s \rightsquigarrow (s',a)$ iff $s = s'$.*
- *For $(s',a) \in V_{\mathrm{sys}}$, $s \in V_{\mathrm{env}}$, we have $(s',a) \rightsquigarrow s$ iff $s' \xrightarrow{a} s$ in A.*

In $G_{\mathrm{graph}} = (A_{\mathrm{graph}}, s_0, \mathrm{Win}_{\mathrm{graph}})$, players take turns moving a token along edges of A_{graph}, starting from s_0. Player Env chooses an action a enabled at s, and Sys selects a corresponding transition target.

Each play in G_{graph} corresponds precisely to a play in $\mathscr{G}$, and strategies in one game translate naturally to the other. A strategy for Sys maps histories of Env's actions to valid transitions, aligning with ATS game strategies.

By choosing $\text{Win}_{\text{graph}}$ to reflect safety, reachability, or parity conditions of Win, this reduction allows us to leverage known results—such as the existence of *memoryless* (zero-memory, i.e., depending only on the current position without accessing history) *uniform* (independent of initial state) winning strategies for these objectives in graph games. Without going into technical details, we summarize the consequences of this translation with the following proposition.

Proposition 3.2. *Given a single-process ATS* $A = (\{S_1\}, \{\xrightarrow{a}\})$ *and a safety, reachability, or parity winning condition* Win*, the winning region*

$$W = \{\, s \in S_1 \mid \mathscr{G}_s = (A, s, \text{Win}) \textit{ admits a winning strategy} \,\}$$

can be effectively computed. Moreover, one can compute a uniform memory automaton with zero memory.

3.2 Two process games

We now undertake the study of two process ATS games. We discuss strategies and algorithms for solving ATS games with two processes under various state-based winning conditions, highlighting both the decidability of winning strategies and the memory structures required to implement them.

We first consider the global safety objective, where the system wins if no "bad" global state is ever reached during the play. Next, we study local and global reachability objectives. In local reachability, each individual process must eventually visit its own designated goal set at least once during the play. In contrast, under global reachability, the system wins if there exists a configuration of the play in which a global goal state is reached simultaneously.

Surprisingly, memoryless strategies are insufficient for all these objectives; winning strategies necessarily require memory. Nevertheless, whenever a winning strategy exists, there also exists a finite memory strategy.We provide tight bounds on the computational complexity of the problem of deciding the existence of a distributed winning strategy for these objectives. Another important contribution of this work is the identification of *distributed memory* requirements to meet these objectives. Towards this we fix some notations that are particularly useful for two

process case.

For ease of presentation, in this chapter, we make the *simplifying assumption* that A is complete, that is, for every $a \in \Sigma$ and $s_a \in S_a$, there exists an $s'_a \in S_a$ such that $s_a \xrightarrow{a} s'_a$. Under this assumption, for every trace t, A has at least one run on t.

Given a safety or reachability (local or global) game $\mathscr{G}$, we can construct a modified game $\mathscr{G}'$ on a complete automaton. For each process, we introduce a dummy local action together with a dummy local state. From every local state, there is a transition to its corresponding dummy state on the dummy action and on any local actions that are not enabled. Moreover, for every global state, if a joint action is not enabled, we add a transition to the pair of dummy states.

The winning conditions are updated as follows. For safety, the set of unsafe states remains unchanged. For local reachability, in addition to the existing goal states the system wins as soon as the dummy state is reached, which may be viewed as a local goal state. For global reachability, the system wins whenever a global state containing at least one dummy component is reached. This is also in addition to the existing goal states. With this the system has a winning strategy in $\mathscr{G}$ iff it has a winning strategy in $\mathscr{G}'$. Thus, the simplifying assumption is made purely for the sake of presentation and is without loss of generality.

We continue using the notations from the previous chapter. In particular, as before, $\widetilde{\Sigma} = (\Sigma_1, \Sigma_2)$ is a fixed distributed alphabet over two processes. Let $\Sigma_{12} = \Sigma_1 \cap \Sigma_2$ denote the set of all *shared/joint/synchronizing* actions. Thus, both processes participate/synchronize on actions in Σ_{12}. We write $\Sigma_i^\ell = \Sigma_i \setminus \Sigma_{12}$ for the set of *purely local* actions of process i. Clearly, *only* process i participates in actions from Σ_i^ℓ.

Fix a distributed strategy σ in $\mathscr{G}$. Let t be a finite trace with a unique maximal event whose label is in Σ_{12}. So, the *last* event in t is a synchronizing event in which both processes participate. Note that at t both processes have identical causal past. The strategy σ induces two natural "local" strategies "rooted" at t. Intuitively, these local strategies capture the σ-response of individual processes to sequences of purely local actions which extend the trace t.

Definition 3.3. *Continuing above notation, we define $\sigma_i[t] : (\Sigma_i^\ell)^* \to S_i$ as follows: for $w \in (\Sigma_i^\ell)^*$, $\sigma_i[t](w)$ is the i-local component of the global state $\sigma(t.w)$.*

Remark 3.4. *Let $\sigma(t) = (s_1, s_2)$, $w_1 \in (\Sigma_1^\ell)^*$ and $w_2 \in (\Sigma_2^\ell)^*$. Thanks to the locality of A, we*

have $\sigma(t.w_1) = (\sigma_1[t](w_1), s_2)$ *and* $\sigma(t.w_2) = (s_1, \sigma_2[t](w_2))$. *Furthermore,* $\sigma_1[t](w_1) = s_1'$ *and* $\sigma_2[t](w_2) = s_2'$ *iff* $\sigma(t.(w_1||w_2)) = (s_1', s_2')$ *where* $t.(w_1||w_2) = t.w_1.w_2 = t.w_2.w_1$ *is the extension of* t *by the* parallel *composition* $w_1||w_2$. *More precisely,* $w_1||w_2$ *is the trace* $w_1.w_2 = w_2.w_1$ *and this trace equality is a consequence of the fact that only process* i *participates in* w_i.

It will be useful to think of a distributed strategy σ as made up of a family $\{(\sigma_1[t], \sigma_2[t])\}_{t \in \mathrm{TR}^*(\Sigma).\Sigma_{12}}$ of pairs of local strategies parameterized by finite traces with a unique maximum synchronizing event. A play conforming σ is initially played according to the pair $(\sigma_1[\varepsilon], \sigma_2[\varepsilon])$ of *local* strategies until the *first* synchronizing event e_1 occurs which is responded by the matching transition $(\sigma(\Downarrow e_1), \sigma(\downarrow e_1))$. After this point, the new pair $(\sigma_1[\downarrow e_1], \sigma_2[\downarrow e_1])$ of local strategies is employed until the second synchronizing event e_2 occurs and so on.

3.3 Global safety objective

In this section, we present a fixpoint based algorithm for solving two process global safety ATS games. Fix such a game $\mathscr{G}$ with ATS $A = (\{S_i\}, \{\xrightarrow{a}\})$ over $\widetilde{\Sigma}$, the initial state $s_0 \in S_1 \times S_2$ and the safe set $F \subseteq S_1 \times S_2$. A maximal play (t, ρ) is won by the distributed team/system if for all $c \in C_t$, $\rho(c) \in F$.

A key element in our solution is the notion of a trap. The construction of traps (see [Grädel et al., 2002]) is central to the solution of standard two-player safety games. We define a subset $X_i \subseteq S_i$ to be an *i-trap* if, for every $s \in X_i$ and for each $a \in \Sigma_i^\ell$, there exists $s' \in X$ such that $s \xrightarrow{a} s'$. The crucial property of an i-trap is, within X_i, that process i has a "uniform local strategy" to ensure that plays on purely local actions stay within X_i. To do so, for an i-state $s \in X_i$ and for each $a \in \Sigma_i^\ell$, process i simply selects an i-state s' such that $s \xrightarrow{a} s'$. Below we assume a fixed choice of such a uniform local strategy f_{X_i} for an i-trap X_i. A *rectangle* R is a subset of global states of the form $X_1 \times X_2 \subseteq S_1 \times S_2$ where X_i is an i-trap.

We now describe a fixpoint argument to construct the *winning region* for our *uninitialized* global safety game $\mathscr{G}$. The key idea is to consider a restricted environment which is allowed to play at most j *synchronizing* actions and build a winning region for this restriction by induction on j.

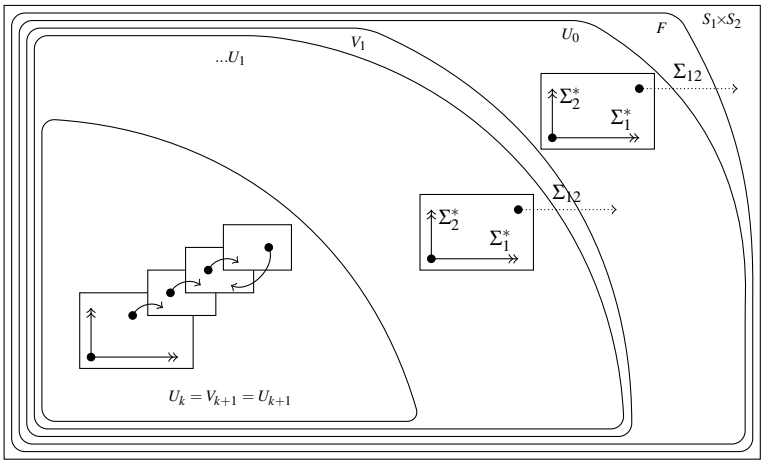


Figure 3.5: *Global safety fixpoint: Rectangle illustrates a trap pair: under local actions, the system remains inside it. Dotted edges indicate an action whose transitions all leave the set U_j, hence it is excluded from V_{j+1}. From V_j, the environment cannot enforce a synchronization outside U_{j-1}. The fixpoint construction proceeds as $F = V_0 \supseteq U_0 \supseteq V_1 \supseteq U_1 \supseteq \cdots$, with $U_k = V_{k+1} = U_{k+1}$ at convergence.*

We inductively define, as illustrated in Fig. 3.5, for $j \geq 0$, two interleaved decreasing families of sets of global states V_j and U_j as follows: $V_0 = F$,

$$U_j = \{s \in V_j | \exists \text{ a rectangle } R = X_1 \times X_2 \text{ such that } s \in R \text{ and } R \subseteq V_j\}$$

$$V_{j+1} = \{s \in U_j \mid \forall a \in \Sigma_{12}\ \exists s' \in U_j \text{ such that } s \xrightarrow{a} s'\}$$

Observe that $F = V_0 \supseteq U_0 \supseteq V_1 \supseteq U_1 \supseteq V_2 \supseteq U_2 \ldots$. It is important to note that, if $s \in U_j$ with a "witness" rectangle R as in the above definition, then $R \subseteq U_j$. So, if $s \in V_j \setminus U_j$ then there is no rectangle containing s which is completely inside V_j. Also note that, if $s \in U_j \setminus V_{j+1}$ then there is an action $a \in \Sigma_{12}$, such that for all s' with $s \xrightarrow{a} s'$, $s' \notin U_j$.

Lemma 3.6. *Let $j \geq 0$. If $s \notin U_j$, then for* every *distributed strategy from s there exists a losing (partial) play conforming it in which the environment chooses at most j synchronizing actions.*

Proof. We prove the statement by induction on j. Let $s \notin U_j$ and let σ^s be a distributed strategy from s. Let $(f_1 = \sigma_1^s[\varepsilon], f_2 = \sigma_2^s[\varepsilon])$ be the initial pair of local strategies (see Definition 3.3) of σ^s. Then, $X_i = f_1(\Sigma_i^{\ell *})$ is an i-trap and s belongs to the rectangle $R = X_1 \times X_2$. The fact that

$s \notin U_j$ implies that $R \not\subseteq V_j$. Therefore there exist words $w_i \in (\Sigma_i)^*$ and $s'_i = f_i(w_i)$ such that $(s'_1, s'_2) \notin V_j$. So, with $t = w_1||w_2$, $\sigma(t) = s' = (s'_1, s'_2) \notin V_j$. For the base case when $V_0 = F$, this t gives a losing play, where the environment does not choose a synchronization action at all. For $j > 0$, we consider the following two exhaustive cases.

Case [$s' \in U_{j-1} \setminus V_j$]: In this case, there is an $a \in \Sigma_{12}$ such that for all s'' with $s' \xrightarrow{a} s''$, $s'' \notin U_{j-1}$. The environment can extend the partial play t by such an a. So, we have $s'' = \sigma^s(t.a) \notin U_{j-1}$. We now view σ^s rooted at $t.a$ as a strategy from s'' and use induction to find a losing play t' of it with at most $j-1$ synchronizing actions. This implies that the play $t.a.t'$ is a losing play of σ^s with at most j synchronizations, and we are done.

Case [$s' \notin U_{j-1}$]: Note that σ^s rooted at t may also be viewed as strategy from s' and we can use induction to finish the proof in this case. □

Let $k \geq 0$ be the least index so that $U_k = U_{k+1}$. Clearly, $U_k = V_{k+1} = U_{k+1}$. For every $s \in U_k$, we fix a rectangle $R_s = X_1^s \times X_2^s \subseteq U_k$. Let $f_i^s : X_i^s \times \Sigma_i^\ell \to X_i^s$ be a fixed choice of uniform local strategy for the i-trap X_i^s. Further, for every $s \in U_k$ and $a \in \Sigma_{12}$, we fix $s' \in U_k$ such that $s \xrightarrow{a} s'$ and denote this a-successor of s in U_k by $\mathrm{succ}(s,a)$.

We now construct a uniform winning memory automaton from U_k. Towards this, we define the deterministic ATS $\mathscr{A} = (\{S_i \times M_i\}, \{\delta_a\})$ where $M_i = S_1 \times S_2$. The transition function δ_a of $\mathscr{A}$ is defined as follows: if $a \in \Sigma_i^\ell$, then $\delta_a((m_i, s_i)) = (m_i, f_i^{m_i}(s_i, a))$. If $a \in \Sigma_{12}$, $\delta_a(((m_1,s_1),(m_2,s_2))) = ((s',s'_1),(s',s'_2))$ where $s' = (s'_1, s'_2) = \mathrm{succ}((s_1,s_2),a)$. An important observation is that the local memory states in $\mathscr{A}$ simply store the global state (m) of $\mathscr{G}$ at the *last synchronization*. Both processes agree on a rectangle at this synchronization and continue using the local strategies (f_i^m) until the next synchronization.

Theorem 3.7. *A state $s \in U_k$ iff there exists a distributed winning strategy from s. Moreover, if $s \in U_k$, then $\mathscr{A}$ with initial memory state $(s,s) \in M_1 \times M_2$ is a uniform winning memory automaton in $\mathscr{G}$ from the initial state s.*

Proof. The Lemma 3.6 shows that if $s \notin U_k$ then there does not exist a distributed winning strategy from s. If $s = (s_1, s_2) \in U_k$, then we show that all reachable global states of $\mathscr{A}$ from the initial state $((s,s_1),(s,s_2))$ are of the form $((m,s'_1),(m,s'_2))$ where $m \in S_1 \times S_2$ and, $(s'_1, s'_2) \in U_k$.

It follows from the definition of $\mathscr{A}$. Let $((m_1,s_1),(m_2,s_2))$ be a state reachable from

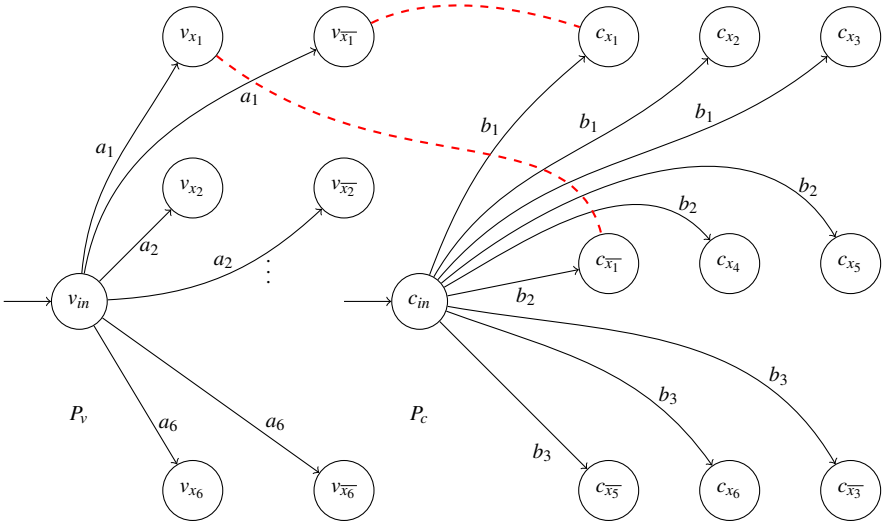


Figure 3.9: NP-hardness reduction: Two of the unsafe states are connected by red dashed lines. For $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_4 \vee x_5) \wedge (\overline{x_5} \vee x_6 \vee \overline{x_3})$ $\mathscr{G}_\varphi$ is given in this figure.

the initial state on t. Then, if $a \in \Sigma_{12}$, $\delta_a(((m_1,s_1),(m_2,s_2))) = ((s',s'_1),(s',s'_2))$ where $s' = (s'_1,s'_2) = \mathrm{succ}((s_1,s_2),a))$. And we have fixed succ such that $s' \in U_k$. Further, f^s_1, f^s_2 are uniform local strategy in rectangle $R_s \subseteq U_k$ and therefore local actions all states reachable are in U_k.

As $U_k \subseteq F$, global states of A encountered during plays conforming $\mathscr{A}$ are safe. So $\mathscr{A}$ is winning. □

Theorem 3.8. *Two process global safety ATS games are NP-complete.*

Proof. By Theorem 3.7, if there is a distributed winning strategy, there is a uniform winning memory automaton whose local memory state stores the global state at the *last* synchronization. This serves as a polynomial size certificate for yes instances of the decision problem of the existence of a distributed winning strategy. It is easy to see that the resulting verification task can be achieved by a poly-time algorithm. This shows membership in NP.

To show NP-hardness, we provide a reduction from 3-SAT. Consider a 3-SAT instance φ with n variables and m clauses. We construct a two process safety ATS game $\mathscr{G}_\varphi$ with $|S_1| = 2n+1$, $|S_2| = 3m+1$ over the alphabet $|\Sigma_1| = n$, $|\Sigma_2| = m$ and $\Sigma_{12} = \emptyset$. Fig. 3.9 illustrates the reduction for the formula

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_4 \vee x_5) \wedge (\overline{x_5} \vee x_6 \vee \overline{x_3})$$

As seen from the figure, the initial state of $\mathcal{G}_\varphi$ is (v_{in}, c_{in}). At this initial state, the environment can play a purely local action $a_i \in \Sigma_1$ of process 1 "asking" it to decide the truth value of variable x_i. Process 1 can respond by moving to either v_{x_i} or $v_{\bar{x}_i}$. This response naturally corresponds to setting the truth value of x_i. Concurrently, environment can play a purely local action $b_j \in \Sigma_2$ of process 2 "asking" it to pick the term in the clause C_j which makes the clause true. As shown in the Fig. 3.9, process 2 can respond by choosing one of the three terms from clause C_j. The unsafe global states correspond to "conflicting" choices i.e.$(v_{x_i}, c_{\overline{x_i}})$ by the two processes (see Fig. 3.9).

We now claim that the resulting game $\mathcal{G}_\varphi$ has a distributed winning strategy iff φ is satisfiable. Assume that φ is satisfiable. A satisfying assignment of φ naturally leads to a distributed winning strategy as follows. The initial local strategy of process 1 selects, for each variable, the literal according to the satisfying assignment. The initial local strategy of process 2 selects, for each clause, the literal in the clause which makes it true under the satisfying assignment. This strategy is a winning strategy due to absence of conflicting choices.

Next, any assignment induced by a winning strategy must satisfy φ. Indeed, if the assignment does not satisfy φ, then there exists a clause that is not true. In such a case, any choice of a term by process 2 on this clause conflicts with the choice of value by process 1 for the variable corresponding to the term process 2 chose, thus reaching an unsafe state.

□

3.4 Local and global reachability objectives

Let $\mathcal{G}$ be an uninitialized game with ATS $A = (\{S_i\}, \{\xrightarrow{a}\})$. A local reachability objective is given by two subsets $F_i \subseteq S_i$. A maximal play (t, ρ) is won by the distributed team if there exists configurations c and c' such that, with $\rho(c) = s = (s_1, s_2)$, $\rho(c') = s' = (s'_1, s'_2)$, we have $s_1 \in F_1$ and $s'_2 \in F_2$. In short, to win a maximal play, process i must ensure a visit to some local state in F_i.

In contrast, a global/simultaneous reachability objective is given by a single subset $F \subseteq S_1 \times S_2$ and maximal play (t, ρ) is won by the team if there exists a configuration c such that

$\rho(c) \in F$. So, to meet the global reachability objective, the team needs to ensure a visit to a global state in F.

3.4.1 Local reachability

We continue with the above notation where the local reachability objective is specified by a pair of subsets $F_i \subseteq S_i$. We first perform an *asynchronous game simulation* wherein each process simply maintains additional information in its local state to record if it has *already* visited a state in F_i. So, we set $Q_i = S_i \times \{0,1\}$ and *extend* $\xrightarrow{a}$ naturally to $\overset{a}{\rightarrowtail}$ over Q_a as follows: for $a \in \Sigma_i^\ell$, for $s \xrightarrow{a} s'$, we add a-transitions $((s,1),(s',1))$, $((s,0),(s',1))$ i.e. $(s,1) \overset{a}{\rightarrowtail} (s',1)$, $(s,0) \overset{a}{\rightarrowtail} (s',1)$ if $s' \in F_i$ and $((s,0),(s',0)),((s,1),(s',1))$ i.e. $(s,0) \overset{a}{\rightarrowtail} (s',0)$, $(s,1) \overset{a}{\rightarrowtail} (s',1)$ if $s' \notin F_i$. For $a \in \Sigma_{12}$, and $s \xrightarrow{a} s'$ with $s = (s_1,s_2)$ and $s' = (s_1',s_2')$, we add a-transitions $(((s_1,b_1),(s_2,b_2)),((s_1',b_1'),(s_2',b_2')))$ where $b_i' = b_i \vee (s_i' \overset{?}{\in} F_i)$. Let $A' = (\{Q_i\},\{\overset{a}{\rightarrowtail}\})$ be the resulting ATS. Observe that, we have used $\overset{a}{\rightarrowtail}$ to denote the extension of $\xrightarrow{a}$ from S_a to Q_a.

We also define a map $\theta : S_1 \times S_2 \to Q_1 \times Q_2$ as follows: for $s = (s_1,s_2)$, we set $\theta(s) = ((s_1,b_1),(s_2,b_2))$ where $b_i = 1$ iff $s_i \in F_i$. Further, we set new local target set to be $F_i' = S_i \times \{1\} \subseteq Q_i$. Let $\mathscr{G}'$ be a new uninitialized game with underlying ATS A' and local reachability condition $\{F_i'\}$. It is easy to see that $\mathscr{G}$ has a distributed winning strategy from s iff $\mathscr{G}'$ has a distributed winning strategy from $\theta(s)$. It is important to note that, in order to win $\mathscr{G}'$, process i needs to ensure a visit to a local state with bit-component 1, that is, a state of the form $(s,1)$ where $s \in S_i$.

Our solution relies on the notion of an i-attractor. See [Grädel et al., 2002] for this notion which plays a key role in the solution of standard two-player reachability games. A subset $X_i \subseteq Q_i$ is an i-attractor (wrt F_i') if it is an i-trap and from *every* state in X_i, process i has a (sequential/local) strategy to *force* a visit to $F_i' = S_i \times \{1\}$. It follows from standard results about reachability games [Grädel et al., 2002] discussed in Proposition 3.2 that process i, in fact, has a zero-memory strategy to force a visit to F_i'.

We now provide a fixpoint construction to compute the winning region in $\mathscr{G}'$. This construction is similar in spirit and *dual* to that of global safety games. The key idea is to consider a *stronger* winning requirement to meet the local reachability objective of $\mathscr{G}'$ *within at most* j

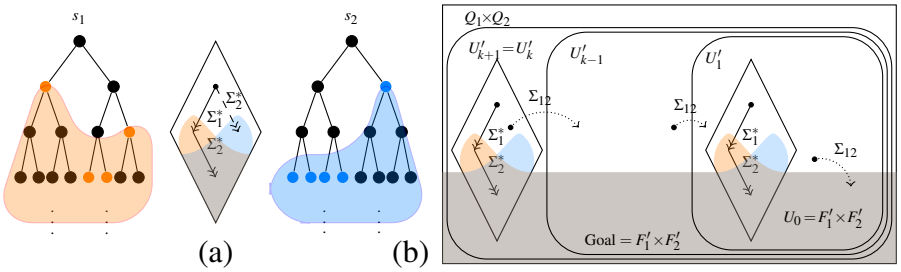


Figure 3.10: Local Reachability fixpoint: (a) A pair of i-attractors. (b) Fixpoint sets $U_j = U'_j \cup U_0$.

synchronizations. Starting with $j = 0$, we progressively increase j to relax the winning requirement. Finally, we reach a fixpoint where the stronger winning requirement matches the local reachability objective.

In case $\Sigma_{12} = \emptyset$ the winning region U_ℓ is given by-

$$U_\ell = \left\{ q \in Q_1 \times Q_2 \,\middle|\, \exists\ i\text{-attractor pair } (X_i) \text{ such that } q \in X_1 \times X_2 \right\}$$

In case $\Sigma_{12} \neq \emptyset$, we also assume that Σ_i^ℓ are non empty, and inductively define, as illustrated in Fig. 3.10, for $j \geq 0$, an increasing chain of sets $U_j \subseteq Q_1 \times Q_2$ as follows: $U_0 = F'_1 \times F'_2 = (S_1 \times \{1\}) \times (S_2 \times \{1\})$.

$$U_j = \left\{ q \in Q_1 \times Q_2 \,\middle|\, \begin{array}{c} \exists\ i\text{-attractor pair } (X_i) \text{ such that } q \in X_1 \times X_2 \text{ and} \\ \forall q' \in (X_1 \times X_2)\ \forall a \in \Sigma_{12}\ \exists q'' : (q' \overset{a}{\rightarrowtail} q'' \ \&\ q'' \in U_{j-1}) \end{array} \right\}$$

It is easy to see that $U_0 \subseteq U_1 \subseteq U_2 \subseteq U_3 \ldots$. Let U_k be the fixpoint of this increasing chain. For every $q \in U_k$, we fix an i-attractor pair (X_i^q) such that $q \in X_1^q \times X_2^q$ and $\forall q' \in (X_1^q \times X_2^q)\ \forall a \in \Sigma_{12}\ \exists q'' : (q' \overset{a}{\rightarrowtail} q'' \ \&\ q'' \in U_{j-1})$ Let $f_i^q : X_i^q \times \Sigma_i^\ell \to X_i^q$ be a fixed choice of uniform local strategy for the i-attractor X_i^q. Further, for every $q \in U_j$ and $a \in \Sigma_{12}$, we fix q' such that $q \xrightarrow{a} q'$ such that, $q \xrightarrow{a} q' = \arg\min_{q \xrightarrow{a} q'} \{j' \mid q' \in U'_j\}$ and denote this a-successor of q in U'_j by $\mathrm{succ}(q,a)$ for local reachability objective. Clearly $j' < j$. We then construct a uniform winning memory automaton from U_k similar to safety games.

Let $((m_1,q_1),(m_2,q_2))$ with $q=(q_1,q_2)\in U_j$ be a state reachable from the initial state on some trace with some memory state m. Then, if $a\in\Sigma_{12}$, $\delta_a(((m_1,q_1),(m_2,q_2)))=((q',q'_1),(q',q'_2))$ where $q'=(q'_1,q'_2)=\text{succ}((q_1,q_2),a))$. Let us assume j,j',j'' be the minimal index such that $q\in U_j, q'\in U'_j$ and $m_1=m_2\in U''_j$. Then, note that we have fixed succ such that $j'<j$. Also, on local strategies $f_i^{m_i}$ the state q reached is definitely in U''_j. Then $j\leq j''$ and therefore $j'<j''$ as well.

Therefore, from any state $q\in U_j$ the system can force the play inside U_{j-1} on the next synchronization. This makes it winning. On the other hand, from a state $q\notin U_j$ for any local strategies f_i their exists words $w_i\in(\Sigma_i)^*$ and $q'_i=f_i(w_i)$ such that $\forall a\in\Sigma_{12}\ \ \nexists q'':(q'\overset{a}{\rightarrowtail} q''\ \&\ q''\in U_{j-1})$. Then, on an a the next state $q''\notin U_{j-1}$. Therefore, as $U_k=U_{k+1}$ from a state $s\notin U_k$ for any system strategy there is a play that remains outside U_k. This is stated in the next lemma.

Lemma 3.11. *Let $j\geq 0$. If $s\notin U_j$, then for* every *distributed strategy from s there exists a losing play conforming it in which the local reachability objective is not reached within j synchronizations.*

Our main results about local reachability games are captured in the next two theorems. With the minor additions mentioned above the proofs of the corresponding results about global safety games there can be adapted here.

Theorem 3.12. *There exists a distributed winning strategy in $\mathcal{G}$ from s iff the state $q=\theta(s)\in U_k$. Moreover, if $\theta(s)\in U_k$, there is a winning memory automaton in $\mathcal{G}$ from s, whose local memory state of process i stores the global state of A at the* last *synchronization and a bit-record of whether or not process i has* already *visited a state in F_i.*

Theorem 3.13. *Two process local reachability ATS games are NP-complete.*

The NP-hardness construction for global safety can be adapted to local reachability by adding a local goal state for each process and then adding joint transitions from terminal states that are safe in the construction to the local goal state pair.

3.4.2 Global reachability

We now focus on global reachability objective. As usual, we fix an uninitialized game $\mathscr{G}$ with ATS $A = (\{S_i\}, \{\xrightarrow{a}\})$ and consider the global reachability objective specified by a set $F \subseteq S_1 \times S_2$. Recall that the distributed team meets the global reachability objective along a maximal play (t, ρ) if there exists a configuration c such that $\rho(c) \in F$.

We now relate the global reachability game $\mathscr{G}$ to a local reachability game $\mathscr{G}'$ on ATS A'. In A', we set local state set to be $Q_i = S_i \times 2^{S_i} \times \{0,1\}$ and *extend* a-transitions $\xrightarrow{a}$ on S_a to a-transitions $\overset{a}{\rightarrowtail}$ on Q_a.

Let $a \in \Sigma_i^\ell$ and $s \xrightarrow{a} s'$. Further, let $Y \subseteq S_i$ and $b \in \{0,1\}$. With $q = (s, Y, b)$, we set $q \overset{a}{\rightarrowtail} q'$ where $q' = (s', Y \cup \{s'\}, b)$. On local action a, process i simply updates the set Y of local states that have been visited *since the last synchronization*. Now let $a \in \Sigma_{12}$ and $s \xrightarrow{a} s'$ where $s = (s_1, s_2)$ and $s' = (s_1', s_2')$. For $Y_i \subseteq S_i$ and $b_i \in \{0,1\}$, we set

$$((s_1, Y_1, b_1), (s_2, Y_2, b_2)) \overset{a}{\rightarrowtail} \big((s_1', Y_1', b_1'), (s_2', Y_2', b_2')\big)$$

where $Y_1' = \{s_1'\}, Y_2' = \{s_2'\}$ and the bits b_1' and b_2' are set according to the following rule. If $((Y_1 \times Y_2) \cup \{(s_1', s_2')\}) \cap F \neq \emptyset$, then $b_1' = 1, b_2' = 1$. Otherwise $b_1' = b_1$ and $b_2' = b_2$. It is important to note that the condition $((Y_1 \times Y_2) \cup \{(s_1', s_2')\} \cap F \neq \emptyset$ asserts that *since the previous synchronization*, a target global state in F has been visited. We include the current global state (s_1', s_2') as part of this check. As Y_i represents the set of local i-states since the last synchronization, $Y_1 \times Y_2$ represents the set of global states that have occurred since then, and it may not include the global state (s_1', s_2') at the current synchronization. Further, note that, we *reset* $Y_i' = \{s_i'\}$ in order to correctly keep track of only local states of process i since the *last* synchronization.

As expected, we define A' to be the ATS $(\{Q_i\}, \{\overset{a}{\rightarrowtail}\})$. We also define (an initial state simulation) map $\theta : S_1 \times S_2 \to Q_1 \times Q_2$ as: for $s = (s_1, s_2)$, if $(s_1, s_2) \notin F$, $\theta(s) = ((s_1, \{s_1\}, 0), (s_2, \{s_2\}, 0))$; otherwise $\theta(s) = ((s_1, \{s_1\}, 1), (s_2, \{s_2\}, 1))$. The local reachability objective is $F_i' = S_i \times 2^{S_i} \times \{1\}$.

The above setup captures plays of $\mathscr{G}$ where a global state from the target set F is *visited*

before a synchronization where both processes flag it by setting their control bits to 1 in $\mathscr{G}'$.

We also need to worry about games where $\Sigma_{12} = \emptyset$. It is indeed possible to handle these "decoupled" purely local games. We fix an uninitialized decoupled game $\mathscr{G}_\ell$ with the global reachability objective specified by the set $F \subseteq S_1 \times S_2$.

To solve this game, we consider a number of local reachability objectives. We consider the set $\mathscr{F} = \{(\hat{F}_1, \hat{F}_2) \subseteq Q_1 \times Q_2 \mid \forall((s_1,Y_1,b_1),(s_2,Y_2,b_2)) \in \hat{F}_1 \times \hat{F}_2 : Y_1 \times Y_2 \cap F \neq \emptyset\}$. For any element $\hat{F}_i$ of this set the winning region of local reachability game $\hat{G}_{\hat{F}_i}$ is given by

$$\hat{U}_{\hat{F}_i} = \left\{ q \in Q_1 \times Q_2 \,\middle|\, \exists\ i\text{-attractor pair } (X_i) \text{ wrt } \hat{F}_i \text{ such that } q \in X_1 \times X_2 \right\}$$

Clearly, from a state, if there is a winning strategy (consisting of a pair of local strategies) for this game, then the same strategy is also winning from the same state as the initial state for the decoupled game with global reachability objective F. One must consider all such local reachability games, and for a state with a winning strategy in any of these games, the two processes can choose local strategies that are winning even in the decoupled game for that state.

Also, from an initial state, let there be a winning strategy in the decoupled game. Then, each play consistent with it is winning and thus a state $((s_1,Y_1,b_1),(s_2,Y_2,b_2))$ is reached such that $Y_1 \times Y_2 \cap F$. Since each Y_i records the set of local states visited since the last synchronization and grows monotonically until reset, every consistent play *converges* to some sets Y_F, Y_F. Given a maximal play $(w_1 w_2, \sigma)$ consistent with strategy σ we say that the Y_i component converges to Y_F if there exist infinitely many prefixes w_i' of w_i such that $\sigma(w_i')_i = (s_i, Y_F, b_i)$ for some s_i, b_i. This strategy is then also winning from the initial state with the local winning condition given by the sets-

$$\hat{F}_i \;=\; \left\{ ((s_1,Y_F,b_1),(s_2,Y_F,b_2)) \in Q_1 \times Q_2 \;\middle|\; \begin{array}{l} \text{there is a maximal play in the strategy} \\ \text{such that the } Y_i \text{ component converges to } Y_F \end{array} \right\}.$$

Moreover, $\hat{F}_i$ is such that $\forall((s_1,Y_1,b_1),(s_2,Y_2,b_2)) \in \hat{F}_1 \times \hat{F}_2 : Y_1 \times Y_2 \cap F \neq \emptyset$.

The winning region U_ℓ of the decoupled game is thus the union $\bigcup_{\hat{F}_i \in \mathscr{F}} \hat{U}_{\hat{F}_i}$ which can also be stated as-

$$U_\ell = \left\{ q \in Q_1 \times Q_2 \,\middle|\, \exists \hat{F}_i \in \mathscr{F} \text{ s.t.} \exists\ i\text{-attractor pair } (X_i) \text{ wrt } \hat{F}_i \text{ such that } q \in X_1 \times X_2 \right\}$$

For every $q \in U_\ell$, we fix an $\hat{F}_i \in \mathscr{F}$ and i-attractor pair (X_i^q) wrt $\hat{F}_i'$ such that $q \in X_1^q \times X_2^q$. Let $f_i^q : X_i^q \times \Sigma_i^\ell \to X_i^q$ be a fixed choice of uniform local strategy for the i-attractor X_i^q. For every state q in the winning region, it is associated with these local strategies f_i^q for the two processes. This constitutes winning memory automaton from U_ℓ. For each state s in game $\mathscr{G}$ we associate with it the strategy associated with $\theta(s)$ in this decoupled game. If there is a distributed winning strategy in this decoupled game from s then there is also a winning memory automaton whose local memory state of process i stores the the *set of local states* of A visited from the initial state. Thus, a winning memory automata of exponential size establishes the membership of deciding the existence of a winning strategy in NEXPTIME.

Theorem 3.14. *If there is a distributed winning strategy in a decoupled game $\mathscr{G}_\ell$ where $\Sigma_{12} = \emptyset$ from s then there is also winning memory automaton whose local memory state of process i stores the the* set of local states *of A visited so far.*

We now present a fixpoint construction over the global states of $\mathscr{G}'$ to compute the winning region of $\mathscr{G}$ when $\Sigma_{12} \neq \emptyset$. We also assume that Σ_i^ℓ are non empty. For $j \geq 0$, we give an increasing chain of sets $U_j \subseteq Q_1 \times Q_2$ comprising of states from where the system can satisfy the objective before the environment can schedule the jth synchronization is given by:

$$U_0 = F_1' \times F_2' \cup \{((s_1, Y_1, b_1), (s_2, Y_2, b_2)) \mid Y_1 \times Y_2 \cap F \neq \emptyset\}$$

Essentially from states in U_0 the system has already seen the goal state or is starting in a goal state.

$$U_j = \left\{ q \in Q_1 \times Q_2 \;\middle|\; \begin{array}{r} \exists \hat{F}_i \in \mathscr{F} \text{ s.t.} \exists \text{ } i\text{-attractor pair } (X_i) \text{ wrt } \hat{F}_i' \text{ such that } q \in X_1 \times X_2 \text{ and} \\ \forall q' = ((s_1', Y_1', b_1'), (s_2', Y_2', b_2')) \in (X_1 \times X_2) \text{ if } Y_1' \times Y_2' \cap F = \emptyset \\ \text{then } \forall a \in \Sigma_{12} \; \exists q'' : (q' \overset{a}{\rightarrowtail} q'' \;\&\; q'' \in U_{j-1}) \end{array} \right\}$$

It is easy to see that $U_0 \subseteq U_1 \subseteq U_2 \subseteq U_3 \ldots$. Let U_k be the fixpoint of this increasing chain.

For every $q \in U_k$, we fix an $\hat{F}_i \in \mathscr{F}$ and i-attractor pair (X_i^q) wrt $\hat{F}_i'$ such that $q \in X_1^q \times X_2^q$ and $\forall q' = ((s_1', Y_1', b_1'), (s_2', Y_2', b_2')) \in (X_1^q \times X_2^q)$ if $Y_1' \times Y_2' \cap F = \emptyset$ then $\forall a \in \Sigma_{12} \; \exists q'' : (q' \overset{a}{\rightarrowtail} q'' \;\&\; q'' \in U_{j-1})$. Let $f_i^q : X_i^q \times \Sigma_i^\ell \to X_i^q$ be a fixed choice of uniform local strategy for the i-attractor X_i^s. Further, For every $q = ((s_1, Y_1, b_1), (s_2, Y_2, b_2)) \in U_j$ and $a \in \Sigma_{12}$, if $b_1 = b_2 = 0$ and $Y_1 \times Y_2 \cap F = \emptyset$ then, we fix q' as in local reachability games. That is $q \overset{a}{\to} q'$ such that, $q \overset{a}{\to} q' = \arg\min_{q \overset{a}{\to} q'} \{j' \mid q' \in U_j'\}$ and denote this a-successor of q in U_j' by $\mathrm{succ}(q,a)$ for global reachability objective. We then construct a winning memory automaton from U_k similar to local reachability games. The fixpoint U_k of this increasing chain is the winning region of $\mathscr{G}'$.

A key consequence of this analysis is the next theorem. With the minor additions mentioned above the proofs of the corresponding results about local reachability games can be adapted here.

Theorem 3.15. *If there is a distributed winning strategy in $\mathscr{G}$ from s then there is also winning memory automaton whose local memory state of process i stores the global state of A at the* last *synchronization, the* set of local states *of A visited since the last synchronization and a bit-record indicating whether or not a visit to a state in F has occurred in its causal past.*

The existence of such a memory strategy implies that global reachability ATS games are in NEXPTIME. There is a somewhat surprising connection between two process ATS games and *generalized* two-player reachability games studied in [Fijalkow and Horn, 2012a]. This allows us to provide lower bounds on computational and memory complexity of global reachability games.

Theorem 3.16. *We have the following lower bounds for global reachability games.*

1. *Two process global reachability ATS games are PSPACE-hard.*

2. *One can construct two process global reachability ATS games with each process having $O(n)$ local states in which a distributed winning strategy exists. However, any winning memory automaton needs at least $2^n - 1$ local memory states.*

Proof of 1. We do a reduction from QBF to two process global reachability. i.e. Given a QBF instance we generate a reachability instance such that there is a winning strategy iff QBF formula is true. Note that this game does not require any joint actions. This proof is an adaptation from [Fijalkow and Horn, 2012a].

Consider the QBF with n variables and m clauses

$$Q_1 v_1 Q_2 v_2 Q_3 v_3 ... Q_n v_n \bigwedge_{i \in \{1,....m\}} C_i$$

where $C_i = t_{i,1} \vee t_{i,2} \vee ... \vee t_{i,j_i}$ and $t_{i,j} = v_k$ or $\overline{v_k}$ for some $k \in \{1,...,n\}$.

We reduce this problem to a two process reachability game as described in Fig. 3.17. The first process facilitates that the environment chooses the value of universally quantified variables and system chooses the value of existentially quantified variables. Process two facilitates the environment to choose a clause and ask "which is the term that makes it true?" So the system should have chosen at least one term true to satisfy this clause. Formally,

Process 1: For each variable v_k, there are three states in process 1, namely $k, v_k, \overline{v_k}$. If v_k is existential, the transitions at k are non-deterministic on action a. If v_k is universal, the transitions at k are deterministic on actions $\top, \bot$. There is a sink state from where there is an infinite local play. The initial state is 1.

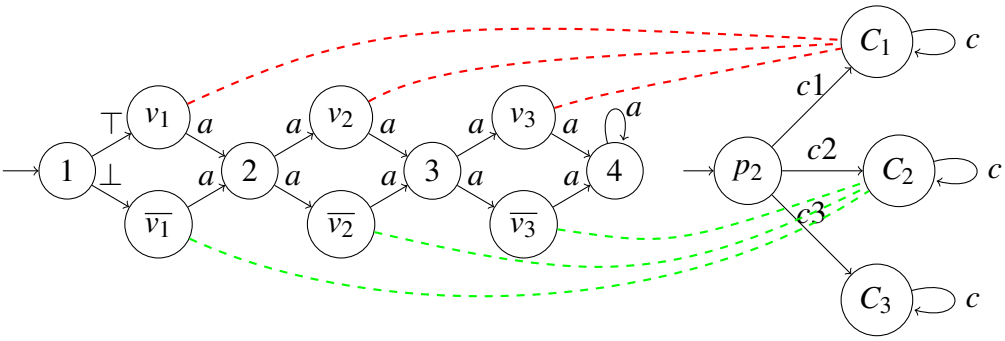
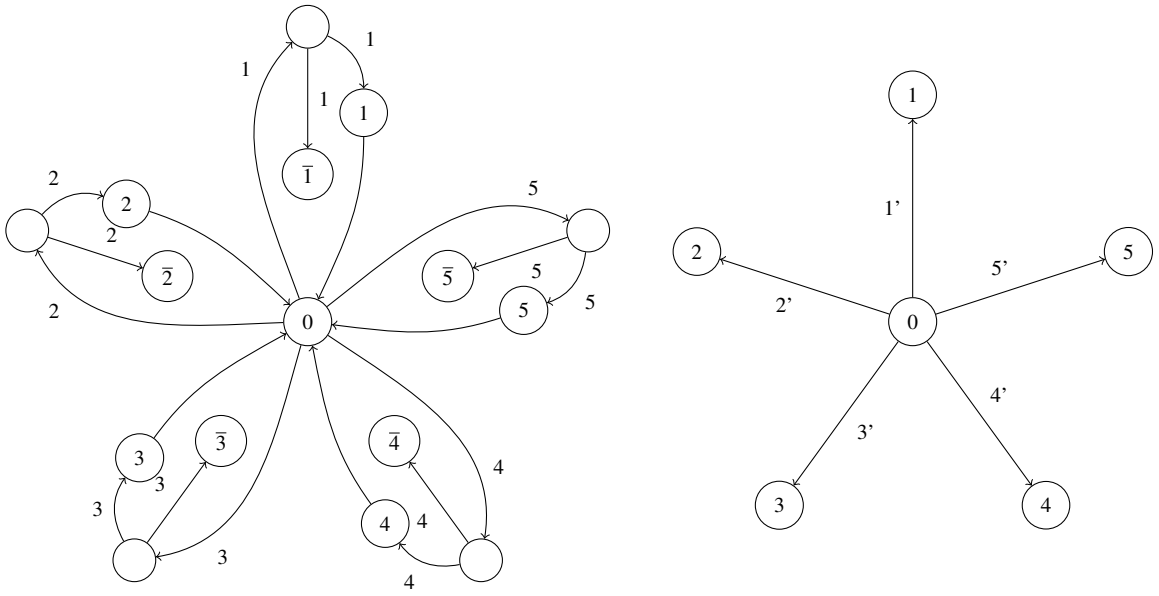


Figure 3.17: PSPACE-hard: $\forall v_1 \exists v_2 \exists v_3 ... ((v_1 \vee v_2 \vee v_3) \bigwedge (\overline{v_1} \vee \overline{v_2} \vee \overline{v_3}) \bigwedge ...)$



(a) Process 1 (b) Process 2

Figure 3.18: Memory Lower bound: process 1 and 2

Process 2: It has an initial state p_2. For each clause C_i it has a state C_i. Transitions from p_2 to each clause state is deterministic.

Reachability objective is given by $\bigcup_{i\in\{1,\ldots m\}} \{t_{i,1}, t_{i,2}, \ldots, t_{i,j_i}\} \times \{C_i\}$

There is a natural bijection between assignments of the variables and a local play of process 1. An assignment satisfies the formula $\bigwedge_{i\in\{1,\ldots m\}} C_i$ iff for all possible local plays of process 2 the concurrent play thus generated satisfies the reachability objective. Therefore, the system has a winning strategy iff the QBF formula is true.

□

Proof of 2. There exists a game $\mathscr{G} = (A, F)$ with $|F| = k^2$, where system needs $2^k - 1$ memory states to win. This example is an adaptation from [Fijalkow and Horn, 2012a].

The initial state is $(0,0)$. States and transitions are according to Fig. 3.18. In process 1, from the initial state, first the environment chooses to play an action i, then process 1 chooses either to reach the state i and return to the initial state 0 or to reach the state $\bar{i}$ and stay in it thereafter.

Goal set is given by

$$\begin{aligned} F = \{(1,1),(2,2),(3,3),(4,4),(5,5)\} \\ \cup \bar{1} \times \{2,3,4,5\} \\ \cup \bar{2} \times \{1,3,4,5\} \\ \cup \bar{3} \times \{1,2,4,5\} \\ \cup \bar{4} \times \{1,2,3,5\} \\ \cup \bar{5} \times \{1,2,3,4\}\} \end{aligned}$$

System wins with the following strategy- The first time environment plays action i, process 1 goes to state i and returns to initial state, the second time process 1 goes to state $\bar{i}$. This way, for any local play of process 2, the system would have visited at least one goal state. Process 1 has memory automaton $M_1 = 2^{1,\ldots k}$ and remembers the petals visited previously. Process 1 need not remember the set $\{1,\ldots k\}$ as it in this case the system will definitely win for any choice of action in process 2. Therefore, the size of memory for process 1 is $|M_1| = 2^{1,\ldots k} \backslash \{1,\ldots k\}| = 2^k - 1$. Process 2 has only deterministic actions so never uses any memory $M_2 = \phi$.

Lemma 3.19. *There is no winning strategy for this game with less than* $2^k - 1$ *memory states.*

Proof. Let there be a memory automaton $\mathscr{A} = (\{S_j \times M_j\}_{j\in\{1,2\}}, \{\delta_a\})$ with $|M_1| < 2^k - 1$. We show that it cannot be winning. We define stopping function as $\text{Stop} : M_1 \to 2^{\{1,\ldots k\}} \backslash \{1,\ldots k\}$ where $m \mapsto S_m$ if Process 1 in local state 0 and memory state m chooses $\bar{i}$ when environment plays(twice) action $i \in S_m$. i.e. $S_m = \{i \mid \delta_{ii}(m,0) = (m', \bar{i})$ for some memory state $m'\}$. Therefore in the memory state m on action $i \notin S_m$ process 1 goes to state i.

As $|M_1| < 2^k - 1$, there is a set which has no preimage in the function Stop. We call it X. The only state where the environment has more than one action enabled in process 1 is the initial state 0. Environment wins against this strategy in the play described as follows- Each time process 1 is in state 0, if process 1 is in memory state m environment plays an action from $(X - \text{Stop}(m)) \cup (\text{Stop}(m) - X)$. This set is never empty because X is not an image for any m.

If for process 1 environment plays forever in X, then only states in X are reached for process 1 and environment wins in the play where it plays an action $i' \in \{1,\ldots k\} - X$ in process 2.

If the environment does play an action i from $\text{Stop}(m) - X$ when process 1 is in memory state m, then we can show that action i has not been played earlier. Say in memory state m' action i was played earlier. It could not have been in $X - \text{Stop}(m')$ as $i \notin X$. It could not have been in $\text{Stop}(m') - X$ as then then process 1 would have reached state $\bar{i}$, stayed in that state and never return to state 0. Here, as $i \in \text{Stop}(m) - X$ process 1 reaches $\bar{i}$ and stays in that state. Therefore, state i is never visited by process 1. In this case, the environment wins in the play where it plays action 1 in process 2.

□

Chapter 4

CDM Model

In this chapter, we introduce *Central Decision Maker (CDM) games* played within the framework of *asynchronous transition system (ATS)* games. In these games, a designated central process participates in all actions that are non-deterministic, making critical decisions that influence the behavior of the distributed system.

Throughout the chapter, we mainly address the CDM setting. Our final result concerns an extension of the CDM model which permits two decision making processes. In this natural extension, in every action which is not deterministic, at least one of the two decision makers participates. We build on the technical machinery of [Gimbert, 2022] to show the surprising result that safety games with two decision makers are undecidable.

Note that ATS games with one process ($|\mathscr{P}| = 1$) (Section 3.1) are an instance of CDM games and can be analyzed using the theory of two-player full-information games [Thomas, 2008]. CDM games extend this to partial-information settings where processes have different knowledge and are unaware of concurrent scheduling decisions. This setting has practical applications. We also examine memory requirements for distributed strategies.

We show that the key problem of solving a CDM game, that is, deciding the existence of

a distributed winning strategy, is efficiently solvable for global safety and local parity winning objectives. We also provide hardness results for these problems which demonstrate that our algorithmic solutions are optimal. More precisely, we show that the problems of solving CDM games for the above objectives are EXPTIME-complete. Our EXPTIME-completeness result for global safety CDM games has some resemblance with a similar result for the Petri game model with one system token studied in [Finkbeiner and Gölz, 2018]. It is important to note that [Finkbeiner and Gölz, 2018] is only concerned with safety objectives. In contrast, we provide a uniform method to address both safety as well as parity objectives. Towards this, we build upon the theory of Mazurkiewicz traces and the theory of two-player games. More precisely, given a CDM game, we associate with it a natural two-player full-information game. Our key result allows to "extract" from a winning strategy in this two-player game, a distributed winning strategy in the CDM game. This is achieved by crucially relying on novel *special* linearizations of traces that we develop in this work. In the special linearization, all events of the CDM process appear as early as possible. As a result, the response of a sequential strategy along this linearization is based on the least amount of information available to the central decision maker. We then lift these sequential responses to extract a distributed strategy. Crucially, the linearization of the causal past of each CDM event extends that of the preceding CDM events. This property ensures that the distributed strategy can faithfully follow the sequential one on CDM events. In fact, on any trace, the distributed strategy maps the trace to the same global state that the sequential strategy would reach when run on the special linearization of that trace.

We believe that these special linearizations have much wider applicability. Our analysis provides richer insights into the non-trivial distributed memory requirements and the structure of the winning strategies thus formed.

Let us mention that our analysis applies to the more general setting of distributed games in which the 'decision' events are causally ordered. In this general setting, any pair of actions, which are *not deterministic*, must have *some* process participating in both of them. We thus rule out concurrent/independent actions which are not deterministic. Clearly, this setting includes CDM games as the central decision maker participates in *all* actions which are not deterministic.

Here are some prototypical examples of CDM systems. In many server-client architectures the server process is primarily responsible for maintaining overall integrity and ensuring that the demands of the client processes are met. These systems serve as prototypical examples

of CDM systems. A distributed version control system such as SVN may also be viewed as a CDM system in which a single authoritative main copy is maintained at the central repository. Users make concurrent changes to local copies; in order to effect these changes to the main copy, they acquire an exclusive access to the central repository using a "lock". One can model this locking mechanism by a central decision maker process which synchronizes with different user processes.

Another illustrative example of a CDM system is the working of an institution/organization consisting of multiple agents with a designated *head*. Various committees are formed to address different aspects of the organization, expedite work and increase overall efficiency. An agent is typically a member of several such committees. These committees conduct procedural meetings from time to time to discuss, gather relevant information and propose solutions. However, all the key decisions are taken in only those committee meetings in which the head participates. The head's presence in all decision making activities ensures consistency and alignment with the organizational goals. Note that concurrent meetings of different committees are allowed.

With this we now formalize the notion of *decision making processes*.

Definition 4.1. *Let $\mathscr{G} = (A, s_0, \mathrm{Win})$ be an ATS game with $A = (\{S_i\}, \{\xrightarrow{a}\})$. Recall that an action a of A is* not *deterministic if there exists $s_a, s'_a, s''_a \in S_a$ such that both (s_a, s'_a) and (s_a, s''_a) are a-transitions (that is, belong to $\xrightarrow{a}$) and $s'_a \neq s''_a$.*

A subset $Q \subset \mathscr{P}$ of processes is said to be decision makers *in $\mathscr{G}$ if for every a which is* not *deterministic, $Q \cap \mathrm{loc}(a) \neq \emptyset$. In other words, if no process in Q participates in an action b, then the transition relation $\xrightarrow{b}$ of A is a (partial) deterministic function from S_b to S_b.*

A **c***entral* **d***ecision* **m***aker (CDM) game is an ATS game with decision makers $\{\ell\}$ for some process $\ell \in \mathscr{P}$. This central decision maker process ℓ participates in* every *action of A which is* not *deterministic.*

Example 4.3. *Consider a CDM game illustrated in Fig. 4.2 with $\mathscr{P} = \{1,2,3\}$ and process 1 as the central decision maker. Note that only action c is* not *deterministic and is shared by 1 and 2. Process i has purely local action d_i and actions a and b are shared by 2 and 3.*

All 3 processes start in top local states and want to avoid states where colors mismatch. This can be captured as a global safety winning condition where color-mismatch signifies occurrence of an unsafe global state. Observe that, initially, the scheduler can play exactly one

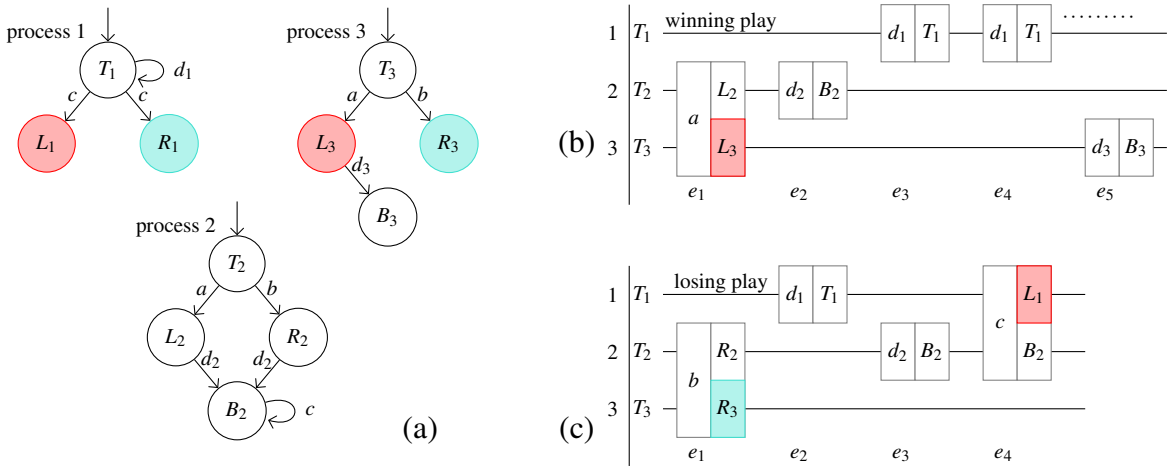


Figure 4.2: A safety CDM game with unsafe set $\{(L_1, B_2, R_3), (R_1, B_2, L_3)\}$ and its plays

of the deterministic actions a or b forcing *the next joint state of process* 2 *and process* 3 *to be either* (L_2, L_3) *or* (R_2, R_3). *It can concurrently schedule any number of* d_1 *actions. Afterwards, the scheduler can* force *process* 2 *to the bottom local state* B_2 *by playing local action* d_2. *At this point, actions c and* d_1 *are* enabled *and the scheduler is allowed to play any of these actions. Note that, if process* 3 *is in local state* L_3 *then the scheduler must also schedule* d_3 *in a maximal play. Some plays of this CDM game are depicted in Fig. 4.2.*

If action c is never scheduled, the distributed system wins along such maximal plays. However, if action c is ever scheduled then the resulting response determines the winner. In order to win in such situations, the central decision maker process 1 *needs to inspect its* causal past *to find out which of the actions a or b was played in the past. With this information, process* 1 *makes the correct decision and ensures a win for the team.*

The description outlined towards the end of Example 4.3, for the CDM game from Fig. 4.2, can be easily translated into a formal winning distributed strategy. Recall that given an ATS game $\mathscr{G} = (A, s_0, \mathrm{Win})$, the key algorithmic question is to decide if there exists a distributed winning strategy in $\mathscr{G}$. We answer this question for CDM games with global safety and local parity winning conditions. Note that a CDM game is a partial-information game and different processes have mutually incomparable partial informations about scheduler's actions; a process is oblivious to concurrent scheduling decisions on other processes and there is no bound on the number of such concurrent scheduling decisions. As the decisions taken by the central decision maker (based solely on its causal past) influence the future behaviour of other processes, these

must be taken somehow to achieve the overall goal.

4.1 Extraction of distributed strategies via special linearizations

We now develop a general construction which aids in the solution of CDM games. Let us fix a CDM game $\mathscr{G} = (A, s_0, \mathrm{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$ and process $1 \in \mathscr{P}$ as the central decision maker. Towards solving $\mathscr{G}$, we introduce a standard *full-information two-player* game G_{seq} played on a finite graph A_{seq} between two players called Sys and Env. See [Grädel et al., 2002] for a study of these games and its relevance to reactive synthesis in the sequential setting.

Definition 4.4. *The game arena A_{seq} is a bipartite graph $A_{\mathrm{seq}} = (V_{\mathrm{env}}, V_{\mathrm{sys}}, \rightsquigarrow \subseteq V_{\mathrm{env}} \times V_{\mathrm{sys}} \cup V_{\mathrm{sys}} \times V_{\mathrm{env}})$ whose vertices and directed edges (that is, elements of $\rightsquigarrow$) are as follows:*

- *$V_{\mathrm{env}} = S_{\mathscr{P}}$ and $V_{\mathrm{sys}} = \{(s,a) \in S_{\mathscr{P}} \times \Sigma \mid a$ is enabled at $s\}$*
- *For $s \in V_{\mathrm{env}}, (s',a) \in V_{\mathrm{sys}}$, we have an edge from s to (s',a) iff $s = s'$*
- *For $(s',a) \in V_{\mathrm{sys}}$ and $s \in V_{\mathrm{env}}$, we have an edge from (s',a) to s iff $s' \overset{a}{\Rightarrow} s$ in the ATS A. Recall that $\overset{a}{\Rightarrow}$ naturally extends the local transition relation $\xrightarrow{a}$ of A to global-states.*

We define $G_{\mathrm{seq}} = (A_{\mathrm{seq}}, s_0, \mathrm{Win}_{\mathrm{seq}})$ to be a standard two-player graph game of complete information whose winning condition $\mathrm{Win}_{\mathrm{seq}}$ is currently *unspecified*. Note that $s_0 \in V_{\mathrm{env}}$. The game G_{seq} is played by players Sys and Env by alternately moving a 'token' along the edges of the bipartite graph A_{seq}. Player Env makes moves from V_{env} and player Sys makes moves from V_{sys}. Initially the token is at vertex $s_0 \in V_{\mathrm{env}}$. Thus a play of G_{seq} is a sequence (possibly infinite) of vertices visited by the token starting with vertex s_0 and it is of the form: $s_0, (s_0,a_0), s_1, (s_1,a_1), s_2, \ldots$. A maximal play is either infinite, or finite in which case it ends in $s \in V_{\mathrm{env}}$ such that no action is enabled at s and as a result, there is no outgoing-edge in A_{seq} from the vertex s. As mentioned above, we do not specify the winning condition $\mathrm{Win}_{\mathrm{seq}}$ in this section and it will be suitably instantiated in later sections.

It will be useful to view a play of G_{seq} as simply a sequence $s_0, a_0, s_1, a_1, s_2, \ldots$ starting at s_0 such that, for each i, a_i is enabled at s_i and $(s_i, s_{i+1}) \in \overset{a_i}{\Rightarrow}$. In this viewpoint, a_i corresponds

to player Env moving the token from s_i to (s_i, a_i) and, s_{i+1} corresponds to player Sys moving the token subsequently from (s_i, a_i) to s_{i+1}.

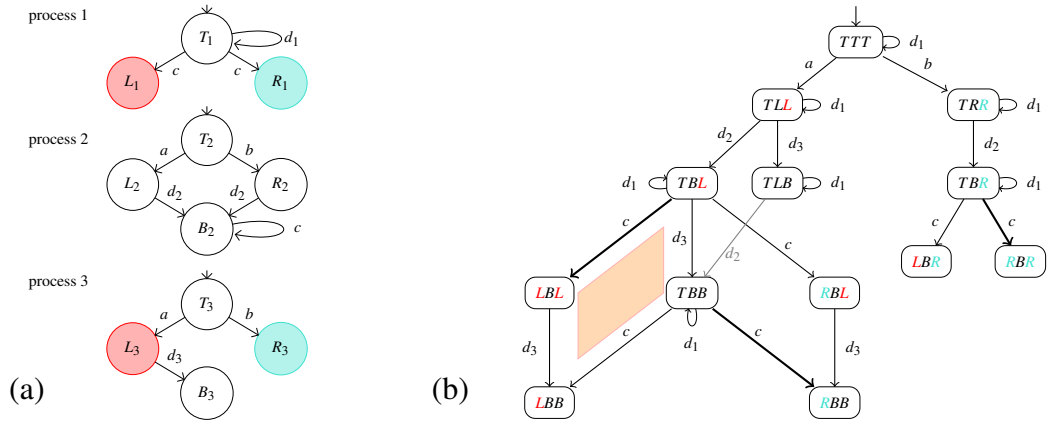


Figure 4.5: An ATS A and the derived sequential game arena A_{seq}

Example 4.6. *Fig. 4.5 shows an ATS A and a simplified view of the derived game arena A_{seq}. The simplified view represents the* sequential automaton *underlying A_{seq}. For instance, in the graph A_{seq} we have two edges from TBL (short for $T_1B_2L_3$) to (TBL, c) and (TBL, d_3) corresponding to two enabled actions, namely c and d_3, at TBL. We also have two edges in A_{seq} from (TBL, c) to LBL and RBL corresponding to two global c-transitions $(TBL, LBL), (TBL, RBL)$ of A. A play $s_0, a_0, s_1, a_1, s_2, \ldots$ starting at s_0 in G_{seq} described above, can be simply viewed as a* run *of this sequential automaton on $a_0a_1a_2\ldots$ from s_0.*

A (sequential full-information) strategy for player Sys in G_{seq} maps its history leading to current vertex to the next valid vertex. We formalize it as follows.

Definition 4.7. *A strategy for player* Sys *in G_{seq} is a partial function $\tau : \Sigma^* \to S_{\mathscr{P}}$ with the smallest domain such that*

- *the domain of τ is a prefix closed subset of Σ^*.*
- *$\tau(\varepsilon) = s_0$ where ε denotes the empty word.*
- *for every $w \in \Sigma^*$ if $\tau(w)$ is defined and a is enabled at $\tau(w)$, then $\tau(wa)$ is also defined and furthermore $(\tau(w), \tau(wa)) \in \overset{a}{\Rightarrow}$. In other words, we have a directed edge in A_{seq} from $(\tau(w), a) \in V_{\text{Sys}}$ to $\tau(wa) \in V_{\text{Env}}$.*

Henceforth, by a strategy in G_{seq} we always mean a strategy *for the player* Sys in G_{seq}. It turns out that every distributed strategy in $\mathscr{G}$ naturally gives rise to a sequential strategy in G_{seq}. We develop some more notation to explain this. Each finite word $w \in \Sigma^*$ naturally induces a finite trace $t \in \text{TR}^*$. More precisely, if $w = a_1 a_2 \cdots a_n$, we associate with w the finite trace $t = (E, \leq, \lambda)$ which is defined as follows: $E = \{e_1, e_2, \ldots, e_n\}$ has n events corresponding to n positions in the word w; we set $\lambda(e_i) = a_i$ and the partial order $\leq$ on E is the transitive-closure of the relation $\{(e_i, e_j) | i \leq j \text{ and } (a_i, a_j) \in D\}$. It is easy to check that t is indeed a trace over $\widetilde{\Sigma}$. Let us denote this association by the map $\eta : \Sigma^* \to \text{TR}^*$. It turns out [Diekert and Rozenberg, 1995; Diekert and Métivier, 1997; Mukund, 2012] that for a trace $t \in \text{TR}^*$, the set $\eta^{-1}(t) \subseteq \Sigma^*$ is precisely all those words which correspond to different linearizations of t. In fact, η is a surjective monoid morphism and $\eta(w) = \eta(w')$ iff w can be obtained from w' by a sequence of operations where each operation is an *exchange* of two *adjacent independent* letters.

The next proposition states that, a distributed strategy σ in $\mathscr{G}$ naturally induces a (sequential) strategy σ_{seq} in G_{seq}.

Proposition 4.8. *Let $\sigma : \text{TR}^* \to S_{\mathscr{P}}$ be a distributed strategy in $\mathscr{G}$. Consider the induced partial* function $\sigma_{\text{seq}} : \Sigma^* \to S_{\mathscr{P}}$ *defined as follows: for $w \in \Sigma^*$, $\sigma_{\text{seq}}(w) = \sigma(\eta(w))$. So w belongs to the domain of σ_{seq} iff $\eta(w)$ belongs to the domain of σ. Then $\sigma_{\text{seq}} : \Sigma^* \to S_{\mathscr{P}}$ is in fact a strategy for player* Sys *in G_{seq}. Furthermore, σ_{seq} is* diamond-closed*; that is, for $w, w' \in \Sigma^*$ and $(a,b) \in I$, $\sigma_{\text{seq}}(wabw') = \sigma_{\text{seq}}(wbaw')$.*

Proof. We show that $\sigma_{\text{seq}} : \Sigma^* \to S_{\mathscr{P}}$ is in fact a strategy for player Sys in G_{seq}.

We see that w belongs to the domain of σ_{seq} iff $\eta(w)$ belongs to the domain of σ. As σ is a distributed strategy the domain of σ is a prefix closed subset of TR^*, it follows that the domain of σ_{seq} is also a prefix closed subset of Σ^*. Also, $\sigma(\varepsilon) = \sigma_{\text{seq}}(\varepsilon) = s_0$.

As σ is a distributed strategy for every $t \in \text{TR}^*$ if $\sigma(t)$ is defined and a is enabled at $\sigma(t)$, then $\sigma(ta)$ is also defined, and $(\sigma(t), \sigma(ta)) \in \overset{a}{\Rightarrow}$. Again w belongs to the domain of σ_{seq} iff $\eta(w)$ belongs to the domain of σ. Therefore, for every $w \in \Sigma^*$ if $\tau(w)$ is defined and a is enabled at $\tau(w)$, then $\tau(wa)$ is also defined and furthermore $(\tau(w), \tau(wa)) \in \overset{a}{\Rightarrow}$. In other words, we have a directed edge in A_{seq} from $(\tau(w), a) \in V_{\text{Sys}}$ to $\tau(wa) \in V_{\text{Env}}$. Thus, σ_{seq} is a strategy in G_{seq}.

Next, we show that σ_{seq} is diamond-closed. As $wabw'$ can be obtained from $wbaw'$ by an

exchange of the two *adjacent independent* letters a and b we see that $\eta(wabw') = \eta(wbaw')$. Therefore, $\sigma_{\text{seq}}(wabw') = \sigma(\eta(wabw')) = \sigma(\eta(wbaw')) = \sigma_{\text{seq}}(wbaw')$. □

So, the strategies in G_{seq} induced by distributed strategies in $\mathscr{G}$ are diamond-closed. However, an arbitrary strategy in G_{seq} is not necessarily diamond-closed.

Example 4.9. *In Fig. 4.5(b), we have used* **bold** *edges to depict the response of a* positional *strategy in G_{seq} on the action c which is not deterministic. This strategy responds to c at TBL by using the bold edge to LBL, at TBB by using the bold edge to LBB and at TBR by using the bold edge to RBR. As a result, at TBL the response to cd_3 ends at LBB while that for d_3c ends at RBB. As c and d_3 are independent, this strategy is not diamond-closed.*

Our main goal in this section is to extract a distributed strategy in $\mathscr{G}$ from an arbitrary strategy crucially exploiting the presence of a central decision maker. Towards this, we now develop special linearizations of finite traces.

We first fix a total order $\lhd_\Sigma$ on Σ such that $\Sigma \setminus \Sigma_1 \lhd_\Sigma \Sigma_1$. Note that, in the total order $\lhd_\Sigma$, if $a, b \in \Sigma$ are such that $1 \notin \text{loc}(a)$ and $1 \in \text{loc}(b)$ then $a \lhd_\Sigma b$. With our implicit assumption that process 1 is the central decision maker, every action in which it does not participate comes before every action in which it participates in the order $\lhd_\Sigma$. We now outline an inductive procedure to compute a special linearization map $\text{Lin} : \text{TR}^* \to \Sigma^*$.

Definition 4.10. *Let $t = (E, \leq, \lambda) \in \text{TR}^*$. If $t = \varepsilon$, then $\text{Lin}(t) = \varepsilon$. Otherwise, let M be the set of all maximal (wrt $\leq$) events of the trace t and e be the unique event in M whose label is $\lhd_\Sigma$-least among the labels in $\{\lambda(f) \mid f \in M\}$. Note that no two events in M can have the same label as events with the same label are ordered/comparable wrt $\leq$.*

It is easily verified that with t' as the trace corresponding to the configuration $E \setminus \{e\}$, we have $t = t'a$ where $a = \lambda(e)$. We now define $\text{Lin}(t) = \text{Lin}(t')a$.

In short, we compute $\text{Lin}(t)$ starting from its last action/letter. Towards this, we peel off that *maximal* event of t which is $\lhd_\Sigma$-least labelled and put its label as the *last* letter in $\text{Lin}(t)$ and continue this process until all events are covered.

Now we exploit the key consequences of the $\lhd_\Sigma$ requirement that $\Sigma \setminus \Sigma_1 \lhd_\Sigma \Sigma_1$. It is very important to observe that, in general, t' is a trace-prefix of t *does not* imply that $\text{Lin}(t')$ is a word-prefix of $\text{Lin}(t)$. Throughout, we use the term prefix for both trace and word prefixes; when

ambiguity may arise, we explicitly write $\sqsubseteq$ to denote the word-prefix relation. The following lemma plays a crucial role later.

Lemma 4.11. *Let $t = (E, \leq, \lambda) \in \mathrm{TR}^*$ and $e \in E$ be such that process 1 participates in e, that is $\lambda(e) \in \Sigma_1$. Then $\mathrm{Lin}(\downarrow e)$ is prefix of $\mathrm{Lin}(t)$. Here we identify the configuration $\downarrow e$ with the corresponding trace-prefix $(\downarrow e, \leq, \lambda)$ of t.*

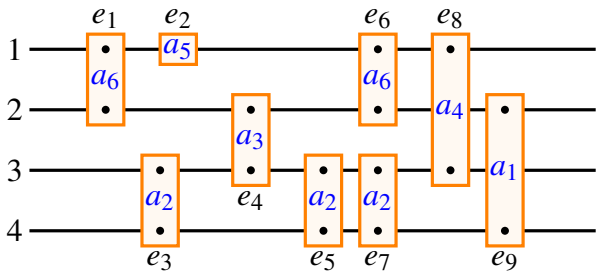
Proof. The proof is by induction on the size of $E \setminus \downarrow e$. For the base case when this is zero, $\mathrm{Lin}(t) = \mathrm{Lin}(\downarrow e)$ and we are done.

We now assume $E \setminus \downarrow e \neq \emptyset$. Let M be the set of all maximal events of t and let f be the element of M whose label is $\lhd_\Sigma$-least.

We claim that $f \neq e$. For otherwise, $e = f$ is a maximal event of t. Further as $\lambda(e) \in \Sigma_1$, every other event in M has a label which is independent of $\lambda(e)$ and larger than $\lambda(e)$ in the order $\lhd_\Sigma$. Due to the very definition of $\lhd_\Sigma$, such labels do not exist. This forces $M = \{e\}$ and contradicts the assumption that $E \setminus \downarrow e = \emptyset$.

By definition of $\mathrm{Lin}(t)$, with t' as the trace induced by the configuration $E \setminus \{f\}$ of t, we have $\mathrm{Lin}(t) = \mathrm{Lin}(t')\lambda(f)$. Thus $\mathrm{Lin}(t') \sqsubseteq \mathrm{Lin}(t)$. As e is also an event of t', by induction (applied to t'), $\mathrm{Lin}(\downarrow e) \sqsubseteq \mathrm{Lin}(t')$. Thus, we obtain that $\mathrm{Lin}(\downarrow e) \sqsubseteq \mathrm{Lin}(t)$ and this completes the inductive step. □

Corollary 4.12. *Given $t = (E, \leq, \lambda)$ and $e, f \in E_1$ such that $e < f$, $\mathrm{Lin}(\downarrow e) \sqsubseteq \mathrm{Lin}(\downarrow f)$.*

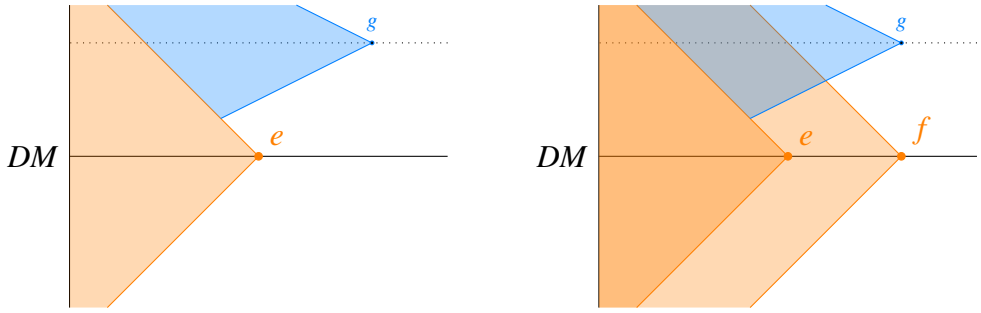


Example 4.13. *We revisit the trace t from Fig. 2.2 with process 1 as the central decision maker and event set $E = \{e_1, e_2, \ldots, e_9\}$. The figure is recreated here for convenience. Let the ordering $\lhd_\Sigma$ be given by $a_i \lhd_\Sigma a_j$ iff $i \leq j$. Note that $\Sigma \setminus \Sigma_1 \lhd_\Sigma \Sigma_1$. We now compute $\mathrm{Lin}(t)$ for the trace t. Maximal events of t are e_8 and e_9 and as $\lambda(e_9) = a_1 \lhd_\Sigma a_4 = \lambda(e_8)$, $\mathrm{Lin}(t) = \mathrm{Lin}(t').a_1$ where t' is given by $E' = E \setminus \{e_9\}$. Successively the maximal event sets are*

$\{e_8\},\{e_6,e_7\},\{e_6,e_5\},\{e_6\},\{e_2,e_4\},\{e_2,e_3\},\{e_2\},\{e_1\}$. *This results in* $\mathrm{Lin}(t) = a_6a_5a_2a_3a_6a_2a_2a_4a_1$ *with event-linearization* $e_1e_2e_3e_4e_6e_5e_7e_8e_9$.

As an illustration of Lemma 4.11, consider event $e_6 \in E_1$. *We have* $\downarrow e_6 = \{e_1,e_2,e_3,e_4,e_6\}$ *and* $\mathrm{Lin}(\downarrow e_6) = a_6a_5a_2a_3a_6$ *which is a prefix of* $\mathrm{Lin}(t)$. *However,* $e_7 \notin E_1$ *and it is easy to see that* $\mathrm{Lin}(\downarrow e_7) = a_6a_5a_2a_3a_2a_2$ *which is* not *a prefix of* $\mathrm{Lin}(t)$.

The Corollary 4.12 summarizes the desirable property of the special linearization is illustrated intuitively in Fig. 4.14. That is special linearization of a later CDM event *extends* that of an earlier CDM event. This is illustrated intuitively in Fig. 4.14.



(a) (b)

Figure 4.14: *Linearization: special linearization of a later CDM event* extends *that of an earlier CDM event.* e,f *are CDM events while* g *is not. (a)* $t = \downarrow\{e,g\}$, $\mathrm{Lin}(\downarrow e) \sqsubseteq \mathrm{Lin}(t)$. *In Linearization DM actions occur before every other independent actions. (b)* $\mathrm{Lin}(\downarrow e) \sqsubseteq \mathrm{Lin}(\downarrow f)$ *Linearization of every DM prime trace extends the Linearization of previous DM prime trace* $t' = \downarrow\{f,g\}$. $\mathrm{Lin}(\downarrow e) \sqsubseteq \mathrm{Lin}(\downarrow f) \sqsubseteq \mathrm{Lin}(t')$. *Note that* in $\mathrm{Lin}(t')$, event f precedes g.

Now we use special linearizations to lift a strategy in G_{seq} to a distributed strategy in $\mathscr{G}$. Intuitively, in the special linearization of a trace, all the events of process 1 appear the *earliest*. Thus the response of a sequential strategy along the special linearization is based on the least amount of information to the central decision maker. We lift these responses to construct a distributed strategy.

Definition 4.15. *Let* $\tau : \Sigma^* \to S_{\mathscr{P}}$ *be a strategy in* G_{seq}. *We define a partial function* $\tau_{\mathrm{dstr}} : \mathrm{TR}^* \to S_{\mathscr{P}}$ *as follows: for* $t \in \mathrm{TR}^*$, $\tau_{\mathrm{dstr}}(t) = \tau(\mathrm{Lin}(t))$. *Note that,* t *belongs to the domain of* τ_{dstr} *iff* $\mathrm{Lin}(t)$ *belongs to the domain of* τ.

Let us illustrate the above definition for the non-diamond-closed strategy τ from Ex-

ample 4.9 with t being the trace induced by the word ad_2d_3c. As $\mathrm{Lin}(t) = ad_2cd_3$, we set $\tau_{\mathrm{dstr}}(t) = \tau(ad_2cd_3) = LBB$. Note that $\tau(ad_2d_3c) = RBB$.

We now state the following main proposition which shows that τ_{dstr} is in fact a distributed strategy in $\mathscr{G}$. The proof crucially uses Lemma 4.11 which implies that, restricted to central decision maker events, special linearization of a later event *extends* that of an earlier one.

Proposition 4.16. *Let $\tau : \Sigma^* \to S_{\mathscr{P}}$ be a strategy for player* Sys *in G_{seq}. Then $\tau_{\mathrm{dstr}} : \mathrm{TR}^* \to S_{\mathscr{P}}$ is a distributed strategy in $\mathscr{G}$.*

To prove this proposition we recall locality of action in the automata A and emphasize its implications in A_{seq}. We also formalize the simple notion that a deterministic action leads to the same change of state regardless of when it is scheduled in the next claim. By definition of η and Definition 4.4 of A_{seq} and Lemma 2.6 about locality of action we have the following lemma.

Lemma 4.17 (locality of action in A_{seq})**.** *Given A_{seq} derived from A for any strategy τ in $\mathscr{G}_{\mathrm{seq}}$ and words $w \in \Sigma^*$, $w_1, w_2 \in (\Sigma \backslash \Sigma_1)^*$ such that $\eta(w_1) = \eta(w_2)$ if ww_1 is in the domain of τ then so is ww_2 and*

$$\tau(ww_1) = \tau(ww_2)$$

Claim 4.18 (determinism of non CDM actions)**.** *Let a, b be two independent actions aIb and b be deterministic. From state s let $s \overset{a}{\Rightarrow} r$ be a transition. Let s' and r' be the* unique *b successors of s and r. (i.e. $s \overset{b}{\Rightarrow} s'$ and $r \overset{b}{\Rightarrow} r'$). Then, $s' \overset{a}{\Rightarrow} r'$ is a valid transition.*

Proof. As seen in Fig. 4.19 the unique b transition enabled at s is also enabled at r and the a transition enabled at s is also enabled at s'. And the resulting state is r'. All of this follows from Lemma 2.6.

$$\begin{array}{ccc} s = (s_a, s_b, s_P) & \xrightarrow{\quad a \quad} & r = (r_a, s_b, s_P) \\ \Big\downarrow b & & \Big\downarrow b \\ s' = (s_a, s'_b, s_P) & \xrightarrow{\quad a \quad} & r' = (r_a, s'_b, s_P) \end{array}$$

Figure 4.19: *We represent s as tuple (s_a, s_b, s_P). $aIb, P = \mathscr{P} \backslash (\mathrm{loc}(a) \cup \mathrm{loc}(b))$*

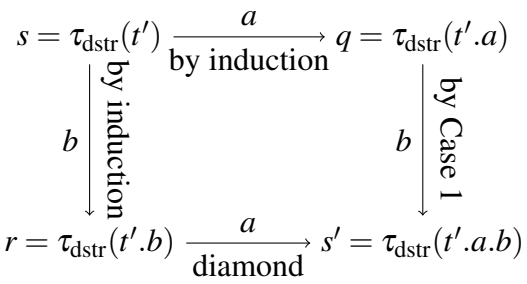


Figure 4.20: aIb and $DM \notin \text{loc}(b)$

□

We now prove the main proposition of this section.

Proof. Note that, t belongs to the domain of τ_{dstr} iff $\text{Lin}(t)$ belongs to the domain of τ.

Clearly $\tau_{\text{dstr}}(\varepsilon) = \tau(\varepsilon) = s_0$. We claim that for any trace $t = (E, \leq, \lambda)$ and an action a, $\tau_{\text{dstr}}(t) \stackrel{a}{\Rightarrow} \tau_{\text{dstr}}(ta)$ is a valid transition in A. The proof of this claim is by induction on the size of t, that is, $|E|$. Towards the proof of the claim, we consider two cases:

The first is when $\text{Lin}(ta)$ ends in the action a. It follows, from Definition 4.10 of $\text{Lin}(.)$, that $\text{Lin}(ta) = \text{Lin}(t)a$. Recall that as τ is a strategy and therefore $\tau(\text{Lin}(t)) \stackrel{a}{\Rightarrow} \tau(\text{Lin}(t)a)$ is a valid transition in A_{seq} and hence in A. By definition of τ_{dstr}, $\tau_{\text{dstr}}(ta) = \tau(\text{Lin}(ta))$ and $\tau_{\text{dstr}}(t) = \tau(\text{Lin}(t))$ therefore

$$\tau_{\text{dstr}}(t) \stackrel{a}{\Rightarrow} \tau_{\text{dstr}}(ta)$$

Alternatively, we can skip using the definition of Lin and use just Lemma 4.11 about Lin to show the same as follows. Let t_1 be the largest prime prefix of t such that $\text{last}(t_1) \in \Sigma_1$ and if $\text{Lin}(t) = \text{Lin}(t_1)w_1$. We use Lemma 4.11 and Corollary 4.12 respectively to say for $a \notin \Sigma_1$ and $a \in \Sigma_1$ that- $\text{Lin}(ta) = \text{Lin}(t_1)w_2a$ such that $\eta(w_1) = \eta(w_2)$. We next use Lemma 4.17 to infer $\tau(\text{Lin}(t_1)w_1) = \tau(\text{Lin}(t_1)w_2)$. Thus, we come to the same reasoning as follows- as τ is a strategy and therefore $\tau(\text{Lin}(t_1)w_2) \stackrel{a}{\Rightarrow} \tau(\text{Lin}(t_1)w_2a)$ is a valid transition in A_{seq} and hence in A. Then, by definition of τ_{dstr}, $\tau_{\text{dstr}}(ta) = \tau(\text{Lin}(ta))$ and $\tau_{\text{dstr}}(t) = \tau(\text{Lin}(t))$. Therefore,

$$\tau_{\text{dstr}}(t) \stackrel{a}{\Rightarrow} \tau_{\text{dstr}}(ta).$$

Now, we consider the case when $\mathrm{Lin}(ta)$ ends in an action b where $b \neq a$. It follows that t has a maximal event (say f) labelled b and $(a,b) \in I$. Then, as $\downarrow f$ is not a prefix of ta, it follows from Lemma 4.11 that process 1 does not participate in b, Thus, b is *deterministic*.

Let $t = t'b$. Then, $ta = t'ba = t'ab$. Let $s = \tau_{\mathrm{dstr}}(t')$ and $r = \tau_{\mathrm{dstr}}(t'b) = \tau(\mathrm{Lin}(t'b))$ and $q = \tau_{\mathrm{dstr}}(t'a) = \tau(\mathrm{Lin}(t'a))$. By applying induction to t' (whose event set has size $|E|-1$), we get that $s \stackrel{b}{\Rightarrow} r$ and $s \stackrel{a}{\Rightarrow} q$.

By the case assumption that $\mathrm{Lin}(ta)$ ends in b, we have by Definition 4.10 $\mathrm{Lin}(ta) = \mathrm{Lin}(t'a)b$. Let, $s' = \tau_{\mathrm{dstr}}(ta) = \tau(\mathrm{Lin}(t'ab))$. Then, as τ is a strategy, we have $q \stackrel{b}{\Rightarrow} s'$ is valid transition in A_{seq} and hence in A.

Now, $s \stackrel{a}{\Rightarrow} q$ and $q \stackrel{b}{\Rightarrow} s'$ are valid transitions and b is deterministic Fig. 4.20. Then due to determinism on non-CDM actions Claim 4.18 allows us to infer that $r \stackrel{a}{\Rightarrow} s'$ is a valid transition. This completes the proof.

□

4.2 Global safety and local parity CDM games

This section is concerned with CDM games with global safety and local parity objectives.

For an ATS $A = (\{S_i\}, \{\stackrel{a}{\rightarrow}\})$, $\max_i |S_i|$ denotes the maximum number of local states per process. For complexity analysis, we make the realistic assumption that every action involves at most a *fixed constant* number of participating processes. Thanks to this assumption and the local nature of transitions of A, it is easy to check that the size of A, denoted by $||A||$, is *polynomial* in $\max_i |S_i|$, $|\Sigma|$ and $|\mathscr{P}|$. Besides $||A||$, the other component which contributes to the size of a CDM game $\mathscr{G}$, denoted $||\mathscr{G}||$, is the specification of the winning condition. Observe that the number of vertices in the game arena A_{seq} (see Definition 4.4) is atmost $2 * |\Sigma| * (\max_i |S_i|)^{|\mathscr{P}|}$. Note that $(\max_i |S_i|)^{|\mathscr{P}|}$ is the maximum number of global-states of A. As a result, the size of A_{seq}, denoted by $||A_{\mathrm{seq}}||$ is polynomial in $(\max_i |S_i|)^{|\mathscr{P}|}$ and $|\Sigma|$.

4.2.1 Global safety objective

Let us fix a CDM game $\mathscr{G} = (A, s_0, \text{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$ and a global-safety winning condition. A global-safety winning condition is given by a subset $F \subseteq S_{\mathscr{P}}$ of *safe* global-states. A maximal play (t, ρ) is won by the distributed system if, for *all* $c \in C_t$, $\rho(c) \in F$. By abuse of notation, we also write $\mathscr{G} = (A, s_0, F)$ and refer to it as a safety CDM game.

Our solution for the safety CDM game uses the full information two-player token game $G_{\text{seq}} = (A_{\text{seq}}, s_0, \text{Win}_{\text{seq}})$ from the previous section. We instantiate the winning condition Win_{seq} and the resulting G_{seq} as follows:

Definition 4.21. *Let $G_{\text{seq}} = (A_{\text{seq}}, s_0, F_{\text{seq}})$ be the full-information two-player safety graph game where the safety objective for player* Sys *is given by*

$$F_{\text{seq}} = \{s \in V_{\text{sys}} \mid s \in F\} \cup \{(s,a) \in V_{\text{env}} \mid s \in F\}$$

In order to win G_{seq}, player Sys *must have a strategy to ensure that the token never leaves the safe-set F_{seq} of vertices of A_{seq}.*

Recall that a strategy in G_{seq} (winning or not) always means a strategy *for player* Sys.

Definition 4.22. *Let $\tau : \Sigma^* \to S_{\mathscr{P}}$ be a strategy in G_{seq}. A play $\alpha = s_0, a_0, s_1, a_1, s_2, \ldots$ of G_{seq}* conforms *τ if, for all i, $\tau(a_0 a_1 \cdots a_i) = s_{i+1}$. The fact that τ is a strategy ensures that we have a directed edge in A_{seq} from (s_i, a_i) to s_{i+1}. The strategy τ is* winning *if all* maximal *plays $\alpha = s_0, a_0, s_1, a_1, s_2, \ldots$ conforming it have the property that, for all i, $s_i \in F_{\text{seq}}$.*

Theorem 4.23. *There is a distributed winning strategy in the safety CDM game $\mathscr{G}$ iff there is a winning strategy in G_{seq}. Moreover, it can be decided in time polynomial (more accurately, quadratic) in the size of A_{seq} whether player* Sys *has a winning strategy in G_{seq}. The running time complexity of the resulting decision procedure for $\mathscr{G}$ is polynomial in $(\max_i |S_i|)^{|\mathscr{P}|}, |\Sigma|, |F|$.*

Proof. Let $\sigma : \text{TR}^* \to S_{\mathscr{P}}$ be a distributed winning strategy. We know from Proposition 4.8 that $\sigma_{\text{seq}} : \Sigma^* \to S_{\mathscr{P}}$ is a strategy in G_{seq}. It is also winning because every state $\sigma_{\text{seq}}(w)$ reached in σ_{seq} is seen in σ as $\sigma(\eta(w))$.

Let $\tau : \Sigma^* \to S_{\mathscr{P}}$ be a winning strategy in G_{seq}. By Proposition 4.16, $\tau_{\text{dstr}} : \text{TR}^* \to S_{\mathscr{P}}$ is a distributed strategy. It is winning because by Definition 4.15 for every trace t, $\tau_{\text{dstr}}(t) = \tau(\text{Lin}(t)) \in F$

Moreover, it can be decided in time polynomial (more accurately, quadratic) in the size of A_{seq} specifically $(\max_i |S_i|)^{|\mathscr{P}|}, |\Sigma|, |F|$ whether player Sys has a winning strategy in G_{seq}. This concludes that the running time complexity of the resulting decision procedure for $\mathscr{G}$ is polynomial in $(\max_i |S_i|)^{|\mathscr{P}|}, |\Sigma|, |F|$. □

The decision procedure for safety CDM games in the above theorem is EXPTIME due to the exponential dependence on $|\mathscr{P}|$. The next proposition states that these games are also EXPTIME-hard. Together these results imply that safety CDM games are EXPTIME-complete. Our proof of the next proposition is similar to the corresponding result for the Petri game model with one system token studied in [Finkbeiner and Gölz, 2018]. The details are given the subsection coming after the next one.

Proposition 4.24. *Global safety CDM games are EXPTIME-hard.*

Proof. The Section 4.2.3 is dedicated to the proof of this proposition. It is a direct consequence of Lemma 4.30, which reduces the $\overline{G_5}$ games to safety CDM games; since $\overline{G_5}$ games are known to be EXPTIME-complete, the result follows. □

4.2.2 Local parity objective

Now we analyze CDM games with local parity objectives. Parity winning conditions are central to the theory of two-player graph games and are studied extensively in literature [Grädel et al., 2002]. A standard parity game on a graph of size n and m colors can be solved in time $n^{m+O(1)}$ [McNaughton, 1993]. A recent advance from [Calude et al., 2022] brings it down to $n^{\log(m)+6}$ - a quasi-polynomial bound.

A local-parity winning condition is given by a color function $\chi : S_1 \to \{0,1,\ldots,k\}$ that assigns each CDM local state a color from the finite color set $C = \{0,1,\ldots,k\}$. Given a maximal play (t,ρ) with $t = (E,\leq,\lambda)$, we define $\inf(t,\rho) = \{s \in S_1 \mid \exists^{\infty} e \in E_1, \rho(\downarrow e)(1) = s\}$. Note that if E_1 - the set of events in which process 1 participates, is finite then $\inf(t,\rho) = \emptyset$. The

system wins if $\inf(t,\rho)$ is empty or $\max\{\chi(s)_{s\in\inf(\rho)}\}$ is even. We also denote such a game by $\mathscr{G} = (A, s_0, \chi)$ and refer to it as a (local) parity CDM game. Towards solving $\mathscr{G}$, we instantiate G_{seq} by an appropriate *sequential* parity condition.

Definition 4.25. *Let $G_{\text{seq}} = (A_{\text{seq}}, s_0, \chi_{\text{seq}})$ be the standard two-player parity game where $\chi_{\text{seq}} : V_{\text{sys}} \cup V_{\text{env}} \to \{0,1,\ldots,k\}$ is defined as: $\chi_{\text{seq}}(s) = 0$ and*

$$\chi_{\text{seq}}((s,a)) = \begin{cases} \chi(s(1)) & \text{if } a \in \Sigma_1, \\ 0 & \text{if } a \notin \Sigma_1. \end{cases}$$

We now describe the winning condition Win_{seq} of G_{seq} using χ_{seq}. As discussed in Section 4.1, a maximal play of G_{seq} may be viewed as a sequence $\alpha = s_0, a_0, s_1, a_1, s_2, \ldots$ and it corresponds to the maximal movement α' of the token along the path $s_0, (s_0,a_0), s_1, (s_1,a_1), s_2, \ldots$ in A_{seq}. On this maximal sequence α, player Sys wins if it is finite or the highest color occurring infinitely often in $\chi^{\alpha'}_{\text{seq}} = \chi_{\text{seq}}(s_0), \chi_{\text{seq}}((s_0,a_0)), \chi_{\text{seq}}(s_1), \chi_{\text{seq}}((s_1,a_1)), \chi_{\text{seq}}(s_2), \ldots$ is even. The following lemma brings out the connection between χ_{seq} and χ.

Lemma 4.26. *If α is finite then player* Sys *wins in α. We now assume that α is infinite. If actions from Σ_1 occur finitely often in α then only color 0 occurs infinitely often in $\chi^{\alpha'}_{\text{seq}}$ and player* Sys *wins in α. If actions from Σ_1 occur infinitely often in α, consider the* infinite *subsequence β' obtained from α' by restricting it to vertices in $V_{\text{sys}} \cap (S_{\mathscr{P}} \times \Sigma_1)$. Suppose $\beta' = (s_{i_1}, a_{i_1}), (s_{i_2}, a_{i_2}), \ldots$. Then the highest color occurring infinitely often in $\chi^{\beta'}_{\text{seq}} = \chi_{\text{seq}}((s_{i_1}, a_{i_1})), \chi_{\text{seq}}((s_{i_2}, a_{i_2})), \ldots$ is same as that of $\chi^{\alpha'}_{\text{seq}}$. Furthermore it is equal to the highest color occurring infinitely often in the sequence $\chi(s_{i_1}(1)), \chi(s_{i_2}(1)), \ldots$.*

Proof. If actions from Σ_1 occur only finitely often in α, then, along α', after some point, *only* vertices of color 0 occur. This is a consequence of the definition of χ_{seq} which colors all vertices in V_{env} as well as all vertices in V_{sys} of the form (s,a) with $a \notin \Sigma_1$ by color 0. As a result, the highest color occuring infinitely often in $\chi^{\alpha'}_{\text{seq}}$ is 0 and player Sys wins in α.

Now assume that actions from Σ_1 occur infinitely often. Observe that, by definition of β', all the vertices omitted from α' to obtain β' are colored 0 by χ_{seq}. Now the claim in the statement of the lemma about $\chi^{\alpha'}_{\text{seq}}$ and $\chi^{\beta'}_{\text{seq}}$ follows immediately as 0 is the smallest color used

by χ_{seq}. The remaining part of the lemma is a direct consequence of the definition of χ_{seq} in terms of the (local) coloring function χ on S_1.

□

Now we state our main result for local parity CDM games.

Theorem 4.27. *There is a distributed winning strategy in the local parity CDM game $\mathscr{G}$ iff the player* Sys *has a winning strategy in G_{seq}.*

Proof. Let $\tau : \Sigma^* \to S_{\mathscr{P}}$ be a winning strategy in G_{seq}. Then, by Proposition 4.16, $\tau_{\text{dstr}} : \text{TR}^* \to S_{\mathscr{P}}$ is a distributed strategy. We next prove that it is winning.

For any given play (t, τ_{dstr}) confirming τ_{dstr} where $t = (E, \leq, \lambda)$ and for each $e_i \in E_1$, it is established that $\tau_{\text{dstr}}(\downarrow e_i) = \tau(\text{Lin}(\downarrow e_i))$. Additionally, by virtue of Corollary 4.12, the ordering $\text{Lin}(\downarrow e_1) \sqsubseteq \text{Lin}(\downarrow e_2) \sqsubseteq \text{Lin}(\downarrow e_3) \sqsubseteq \ldots$ is confirmed. Consequently, the colors observed on t follow the sequence $\tau_{\text{dstr}}(\downarrow e_1), \tau_{\text{dstr}}(\downarrow e_2), \ldots, \tau_{\text{dstr}}(\downarrow e_i), \ldots$. This sequence mirrors the colors witnessed in the singular sequential play governed by $\text{Lin}(\downarrow E_1)$ within strategy τ at partial plays $\text{Lin}(\downarrow e_i)$. To be exact, lets look at the sequential play α on $\text{Lin}(\downarrow E_1)$ as, $s_0, a_0, s_1, a_1, s_2, \ldots$. Let $i_1, i_2 \ldots$ be the indices at which this play sees events in $E_1 = \{e_1, e_2, \ldots e_j, \ldots\}$ i.e. e_j event is seen in index i_j as, $(s_{i_j}, a_{i_j})(s_{i_j+1})$ where $\lambda(e_j) = a_{i_j}$. Note that the local 1-state at s_{i_j} is the same as the local 1-state that was seen in $s_{i_{j-1}+1}$. This is so because if the previous action a_{i_j-1} is deterministic then actions in $\Sigma \backslash \Sigma_1$ do not change the local 1-state even in sequential plays of G_{seq}, else if it is nondeterministic then $i_j = i_{j-1} + 1$. Therefore, in either case $s_{i_j}(1) = s_{i_{j-1}+1}(1)$. Therefore, by Lemma 4.26 highest color occuring infinitely often in $\chi^{\alpha}_{\text{seq}}$ is equal to the highest color occuring infinitely often in the sequence $\chi(s_{i_1}(1)), \chi(s_{i_2}(1)), \ldots$. Therefore, the color seen at $\downarrow e_{j-1}$ in τ_{dstr}, at state $\tau_{\text{dstr}}(\downarrow e_{j-1}) = \tau(\text{Lin}(\downarrow e_{j-1})) = s_{i_{j-1}+1}$ is seen at $\text{Lin}(\downarrow e_j)$ in τ in the state (s_{i_j}, a_{i_j}).

In the scenario where set E_1 is infinite, the play adheres to the parity condition χ if and only if the corresponding play $(\text{Lin}(\downarrow E), \tau)$ satisfies the sequential parity condition χ_{seq}. Importantly, the deterministic part $t \backslash \downarrow E_1$ in the play, as determined by t, holds no relevance to the play's winning outcome. If τ is winning then so is τ_{dstr}.

Let $\sigma : \text{TR}^* \to S_{\mathscr{P}}$ be a distributed winning strategy. We know from Proposition 4.8 that $\sigma_{\text{seq}} : \Sigma^* \to S_{\mathscr{P}}$ is a strategy in G_{seq}. We now show that it is winning.

Consider a maximal play α on word w conforming σ_{seq}- $s_0, a_0, s_1, a_1, s_2, \ldots$ By Lemma 4.26 highest color occuring infinitely often in $\chi^{\alpha}_{\text{seq}}$ is equal to the highest color occurring infinitely often in the sequence $\chi(s_{i_1}(1)), \chi(s_{i_2}(1)), \ldots$ where $a_{i_1}, a_{i_2} \ldots$ are actions in Σ_1. By Proposition 4.8 we know that $\forall w : \sigma_{\text{seq}}(w) = \sigma(\eta(w))$. Therefore $\sigma_{\text{seq}}(a_0 \ldots a_{i_j}) = \sigma(\eta(a_0 \ldots a_{i_j}))$. Also $\sigma(\eta(a_0 \ldots a_{i_j}))(1) = \sigma(\downarrow e_{i_j})(1)$. Now, (t, σ) is winning if the max color seen infinitely often in $\chi(\sigma(\downarrow e_{i_1})(1)), \chi(\sigma(\downarrow e_{i_2})(1)) \ldots$ is even where $E_1 = \{i_1, i_2 ..\}$. Therefore, if σ is winning then so is σ_{seq}.

□

It turns out that we can design an EXPTIME algorithm for solving local parity CDM games. Thanks to our assumption that each action involves at most a fixed constant number of processes, the size $||\mathscr{G}||$ for a local parity CDM game is polynomial in $\max_i |S_i|$, $|\Sigma|$, $|\mathscr{P}|$ and $|C|$ where C is the color set for the parity condition. In contrast, the derived standard parity game G_{seq} has an arena of size polynomial in $(\max_i |S_i|)^{|\mathscr{P}|}$ and $|\Sigma|$, while using only $|C|$ colors. As mentioned before, there are algorithms for solving parity games on arena of size n and m colors in time $n^{m+O(1)}$ [McNaughton, 1993] and $n^{\log(m)+6}$ [Calude et al., 2022]. Using such an algorithm for solving G_{seq} results into an algorithm for solving a local parity CDM game $\mathscr{G}$ whose running time is exponential in $||\mathscr{G}||$. This discussion shows that we have EXPTIME decision procedures for solving local parity CDM games. We can also show that local parity CDM games, even with two colors, are EXPTIME-hard. Thus we obtain the following theorem.

Theorem 4.28. *Local parity CDM games are EXPTIME-complete.*

Proof. We have presented an EXPTIME algorithm for solving local parity CDM games, thereby establishing that this problem belongs to EXPTIME.

EXPTIME-hardness of parity CDM games follows directly from Lemma 4.31, which reduces the G_5 games to parity CDM games. Since G_5 games are known to be EXPTIME-complete, the result holds. □

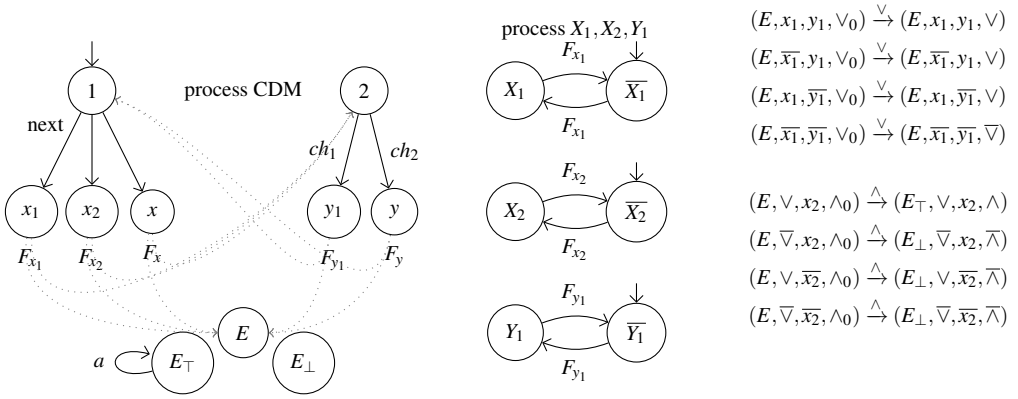


Figure 4.29: $\overline{G_5}$ *reduces to global safety game with local unsafe state* $E_\top$ *and* G_5 *reduces to parity game with coloring* $\chi(E_\top) = 2$ *and if* $s_1 \neq E_\top$ *then* $\chi(s_1) = 1$. *Formula is* $(x_1 \vee y_1) \wedge x_2$.

4.2.3 EXP lower bound for CDM global safety and local parity

Our proof of EXP lower bound for CDM games is similar to the corresponding result for the Petri game model with one system token studied in [Finkbeiner and Gölz, 2018].

The results in this section, although found independently, are similar to [Finkbeiner and Gölz, 2018]. For the lower bound we simulate a combinatorial game known to be EXP-complete given by [Stockmeyer and Chandra, 1979].

The game $G_5 = (P_I, P_{II}, R)$ can be described as follows. The move relation R is described by pairs of positions. A position is a triple $(\tau, F(X,Y), \alpha)$. Here, $\tau \in \{1,2\}$ differentiates positions P_I of Player I from positions P_{II} of Player II. The formula F is in CNF whose variables have been partitioned into disjoint sets X, Y and a is an assignment for the set of variables $V(F)$ in F. Player $I(II)$ moves by changing at most one variable in $X(Y)$: passing is allowed. Player I wins if the formula F is ever true. We define game $\overline{G_5}$ when Player I wins if the formula F is never true.

We consider Fig. 4.29 for the example instance of G_5 and $\overline{G_5}$ is given by $\tau = 1$, formula $F = (x_1 \vee y_1) \wedge x_2$ with initial evaluation α given by $x_1 = x_2 = y_1 = 0$. The set of processes $\mathscr{P} = \{CDM, X_1, X_2, Y_1, \vee, \wedge\}$ and set of actions Σ given by $\Sigma_{CDM} = \{next, ch_1, ch_2, F_{x_1}, F_{x_2}, F_x, F_{y_1}, F_y, \wedge, \vee, a\}$, $\Sigma_{X_1} = \{F_{x_1}\}$, $\Sigma_{X_2} = \{F_{x_2}\}$, $\Sigma_{Y_1} = \{F_{y_1}\}$, $\Sigma_\vee = \{\vee\}$, $\Sigma_\wedge = \{\wedge\}$.

Variable assignment by both players: For every variable in $X \cup Y$ there is a process whose current state reflects its current value. This value is chosen by the system/player I using non-deterministic actions if the variable is in set X or environment/player II using deterministic actions if the variable is in set Y.

Transitions: Player I moves by choosing the next CDM state on a non-determinisitc action thereby stating the variable whose value is to be changed or asking for an evaluation or passing. $(1) \xrightarrow{1} (1,x)$ for every $x \in X$. $((1,x),0_x) \xrightarrow{x} ((2),1_x)$ The system can request an evaluation after any move using $((1,x),0_x) \rightarrow (eval,1_x)$ and $(1) \xrightarrow{1} (eval)$. Deterministic actions are used to evaluate α using $|F|$ processes and binary transitions. States of a process in this group are $\{in_F,0_F,1_F\}$. A formula F contains variables and connectives. Player II moves by choosing a deterministic action to change the value of one variable y in Y. This action is a deterministic action on process y and CDM when CDM is in state (2). $(2,0_y) \xrightarrow{y} (1,1_y)$

Evaluation gadget: After any assignment, the system can call for an evaluation for reduction to G_5. For $\overline{G_5}$ the environemnt can call for an evaluation. Then, after each evaluation the choice of going to the next player or the E state is made by the environment using deterministic actions instead of non-deterministic ones i.e. $F_{x_1},F_{y_1}....$

There are another $|F|$ processes to evaluate F at the current assignment. A process for each binary connector synchronizes with CDM in state E and the 2 processes responsible for assessing its sub-formulas to evaluate this connector using deterministic actions. This requires 4 transitions to evaluate each connector. The final value determines the winner of the game.

This evaluation can be true resulting CDM to be in state $E_\top$ or it can be false forcing the CDM to state $E_\bot$. The only transition from true state $E_\top$ is a deterministic self loop.

Objective: $\overline{G_5}$ reduces to global safety game with $|X \cup Y|$ processes. The local unsafe state of the global safety game is $E_\top$.

G_5 reduces to parity game with with $|X \cup Y|$ processes. The coloring function is given by $\chi(s_1) = 2$ if $s_1 = (E_\top)$ else $\chi(s_1) = 1$.

Lemma 4.30. *The game $\overline{G_5}$ has a winning strategy iff the corresponding global safety CDM game has a winning strategy.*

Proof. There is a one-to-one correspondence between strategies in ATS game and the propositional games. For $\overline{G_5}$ we get a local safety winning condition and as soon as the formula is true it is the Environment's responsibility to call for an evaluation and reach an unsafe state. □

Lemma 4.31. *The game G_5 has a winning strategy iff the corresponding parity CDM game has one.*

Proof. There is a one-to-one correspondence between strategies in ATS game and the propositional games. For G_5 we get a parity winning condition and as soon as the formula is true in the ATS game it is the system's responsibility to call for an evaluation and reach an even state.

□

4.3 Finite-state distributed strategies

In this section we investigate distributed strategies in CDM games which admit a finite-state implementation.

It is well-known (see [Grädel et al., 2002]) that for standard two-player full-information sequential games with safety and parity objectives, existence of a winning strategy implies existence of a positional/"zero-memory" winning strategy. Unfortunately, this is not so for CDM games. It is easy to verify that the safety CDM game from Fig. 4.2 does *not* admit a *zero-memory distributed* winning strategy. However, we have already exhibited a valid distributed winning strategy for this game in Example 4.3.

Let us fix a safety/parity CDM game $\mathscr{G} = (A, s_0, \mathrm{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$. Recall that in Sections 4.2.1 and 4.2.2, we have shown that we can extract a distributed winning strategy τ_{dstr} in $\mathscr{G}$ from a sequential winning strategy τ in G_{seq}. The key property shared by G_{seq} in both instantiations is that: if player Sys has a winning strategy in G_{seq} then it also has a positional winning strategy. Our main goal now is to obtain a memory automaton realizing the distributed strategy τ_{dstr} in $\mathscr{G}$ which is extracted from a positional strategy τ in G_{seq}.

Definition 4.32. *Let $\tau : \Sigma^* \to S_{\mathscr{P}}$ be a strategy in G_{seq}. We call τ positional if there exists a function $f_\tau : V_{\mathrm{sys}} \to V_{\mathrm{env}}$ such that, for every w in the domain of τ and for every a which is enabled at $\tau(w)$, $\tau(wa) = f_\tau((\tau(w), a))$. We refer to f_τ as the witness function of τ.*

Proposition 4.33. *Let $\tau : \Sigma^* \to S_{\mathscr{P}}$ be a positional strategy in G_{seq} with $f_\tau : V_{\text{sys}} \to V_{\text{env}}$ as the witness function of τ. Further, let $\tau_{\text{dstr}} : \text{TR}^* \to S_{\mathscr{P}}$ be the "extracted" distributed strategy in $\mathscr{G}$ defined as: for $t \in \text{TR}^*$, $\tau_{\text{dstr}}(t) = \tau(\text{Lin}(t))$. Then, for a prime trace t of the form $t = t'a$, $\tau_{\text{dstr}}(t) = f_\tau((\tau_{\text{dstr}}(t'), a))$.*

Proof. We have $\tau_{\text{dstr}}(t) = \tau(\text{Lin}(t))$ and $\tau_{\text{dstr}}(t') = \tau(\text{Lin}(t'))$. By Definition 4.10, we have $\text{Lin}(t) = \text{Lin}(t').a$. Further, by Definition 4.32, it follows that $\tau(\text{Lin}(t)) = f_\tau((\tau(\text{Lin}(t')), a))$.

Therefore, $\tau_{\text{dstr}}(t) = \tau(\text{Lin}(t)) = f_\tau((\tau_{\text{dstr}}(t'), a))$. □

The above proposition shows that the response of τ_{dstr} at the last a-event e of a prime trace $t = t'a$ can be determined (using f_τ) by the processes participating in e provided they can compute $\tau_{\text{dstr}}(t')$. Note that $t' = \Downarrow e$ is their collective causal past at e. Hence $\tau_{\text{dstr}}(\Downarrow e)$ represents the *last/latest global-state* about the entire system that they are aware of at e. This suggests that, in order to realize a finite-state implementation of τ_{dstr}, each process should keep track of the latest global-state that it is aware of, in its local memory. When a subset of processes synchronize on a shared action, they need a *finite-state* mechanism to compute the best global-state that they are *collectively* aware of. It turns out that this can be achieved with the help of the *gossip automaton* from [Mukund and Sohoni, 1997].

We first develop some more notation. Let $t = (E, \leq, \lambda)$ be a finite trace, $i \in \mathscr{P}$ and $P \subseteq \mathscr{P}$. Recall that E_i is the set of events in which process i participates. We earlier defined $\partial_i(t)$ to be the trace induced by $\downarrow E_i$ – the events in the causal past of process i. Similarly, $\partial_P(t)$ is the trace induced by $\cup_{j \in P} \downarrow E_j$ and it represents the collective causal past of processes in P. We now define function $\mathsf{latest}_Q : \text{TR}^*(\Sigma) \times \mathscr{P} \to 2^Q$ that gives which processes in the set Q have the latest information about a given process k in a given trace t. For $Q = \{i_1, i_2, \ldots, i_l\}$, $j \in \mathsf{latest}_Q(t,k)$ iff $\partial_k(\partial_Q(t)) = \partial_k(\partial_j(t))$. This is illustrated in Fig. 4.34. We now state the key result from [Mukund and Sohoni, 1997].

Theorem 4.35. *There exists an effectively constructible deterministic complete asynchronous automaton $A_{\text{gossip}} = (\{V_i\}, \{\nabla_a\}, v_0)$ over $\widetilde{\Sigma}$, called the gossip automaton, and, for each $Q = \{i_1, i_2, \ldots, i_l\} \subseteq \mathscr{P}$ there exists an effectively computable function $gossip_Q : V_{i_1} \times \ldots \times V_{i_l} \times \mathscr{P} \to 2^Q$ such that, for every trace t and every process k, $\mathsf{latest}_Q(t,k) = gossip_Q(v_Q, k)$, where v is the global state of A_{gossip} reached on the unique run of A_{gossip} on t.*

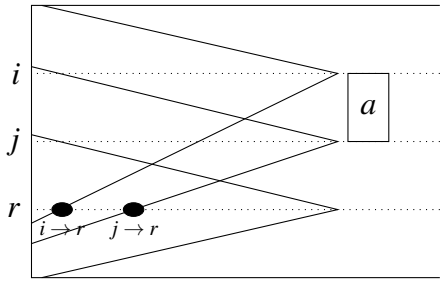


Figure 4.34: *Gossip automaton:* $\mathsf{latest}_Q(t,k) = gossip_Q(v_Q,k)$*: At trace* ta $\tau_{\text{dstr}}(ta)_a = f_\tau(\tau_{\text{dstr}}(\partial_a(t)),a)_a$*. How do participating processes know the global state at their combined view? Each process keeps track of its best knowledge of the global state. When a set of processes meet they would like to update this information. Gossip automaton allows just that. When two processes* i,j *meet it can answer this question - who has a better information about a third process* r*? In this trace* ta*,* $j \in \mathsf{latest}_a(t,r)$*.*

We now describe the memory automaton $\mathscr{A}$ which realizes τ_{dstr} from Proposition 4.33. We set $\mathscr{A} = (\{S_i \times M_i\}, \{\delta_a\}, (s_0, m_0))$ where for each i, $M_i = S_{\mathscr{P}} \times V_i$. So, a local memory i-state $m_i = (q_i, v_i) \in M_i$ of process i contains a local gossip i-state v_i and a global state $q_i \in S_{\mathscr{P}}$ in A. Intuitively, q_i is the best global state that process i is aware of. The initial global memory state $m_0 \in M_{\mathscr{P}}$ is defined as: for each $i \in \mathscr{P}$, $m_0(i) = (s_0, v_0(i))$. Note that s_0 (resp. v_0) is the initial global state of A (resp. A_{gossip}).

We now turn our attention to the definition of the transition function δ_a of $\mathscr{A}$. Let us suppose that $\text{loc}(a) = Q = \{i_1, i_2, \ldots, i_l\}$. Fix $(s_a, m_a) \in S_a \times M_a$ such that a is enabled at s_a in A. For each $i \in \text{loc}(a)$, $m_a(i) = (q_i, v_i) \in S_{\mathscr{P}} \times V_i$. Recall that ∇_a is the a-transition function of A_{gossip} and let $v_a \in V_a$ be such that, for each $i \in \text{loc}(a)$, $v_a(i) = v_i$ and $v'_a = \nabla_a(v_a)$. We now define $\delta_a((s_a, m_a))$ as follows. We first compute the best global state $\hat{s} \in S_{\mathscr{P}}$ that processes in $Q = \text{loc}(a)$ are aware of. For each $k \in \mathscr{P}$, we set

$$\hat{s}(k) = q_j(k) \text{ where } j \in Q \text{ is such that } j \in \text{gossip}_Q((v_{i_1}, v_{i_2}, \ldots, v_{i_l}), k)$$

We next compute $\hat{s}' = f_\tau((\hat{s}, a))$ and define $\delta_a((s_a, m_a))$ to be $(\hat{s}'_a, m'_a)$ where, for each $i \in \text{loc}(a)$, $m'_a(i) = (\hat{s}', v'_a(i))$.

The key property of this construction can be stated as follows-

Lemma 4.36. *For any trace* t*, if* $\mathscr{A}(t) = (s,m)$*, with* $m = (q,v)$ *then* $q_i = \tau_{\text{dstr}}(\partial_i(t))$ *and* $s =$

$\tau_{\text{dstr}}(t)$.

Proof. We prove the claim by *induction* on the size $|E|$ of the trace $t = (E, \leq, \lambda)$. Let t be a trace such that $\mathscr{A}(t) = (s, m)$ with $m = (q, v)$. We show that for every process i, it holds that $q_i = \tau_{\text{dstr}}(t_i)$, where $t_i = \partial_i(t)$, and moreover $s = \tau_{\text{dstr}}(t)$. Proposition 4.16 states that τ_{dstr} is a valid distributed strategy. As t_i is a prime trace Lemma 2.16 implies that $q_i(r) = \tau_{\text{dstr}}(\partial_r(t_i))$. For the *base case*, $\mathscr{A}(\varepsilon) = (s^0, m^0)$ and the claim follows. Say the statement is true for t we prove it for ta.

Note that as τ_{dstr} is a distributed strategy, for any trace t, $\tau_{\text{dstr}}(ta)_a = \tau_{\text{dstr}}(\partial_a(t)a)_a$ and $\partial_a(t)a$ is always a prime trace. Therefore when τ is *memoryless* with witness function f_τ, Proposition 4.33 says that $\tau_{\text{dstr}}(ta)_a = f_\tau(\tau_{\text{dstr}}(\partial_a(t)), a)_a$.

Let $t_r = \partial_r(\partial_a(t))$ Now, let $\mathscr{A}(t_r) = (m^{(r)}, s^{(r)})$. Then, because of the induction step $q_r^{(r)} = s^{(r)}$ and $s^{(r)} = \tau_{\text{dstr}}(t_r)$. Next, using the *gossip* automata state let $j = \text{gossip}_a(v_a, r)$. Then, by Theorem 4.35 $j = \text{latest}_a(t, r)$ and $\partial_r(\partial_a(t)) = \partial_r(\partial_j(t))$. Therefore, $t_r = \partial_r(\partial_j(t))$. By induction, $q_j = \tau_{\text{dstr}}(\partial_j(t))$. Therefore, $q_j(r) = \tau_{\text{dstr}}(t_r)_r = \tau_{\text{dstr}}(t_a)_r$.

In the *construction* of $\mathscr{A}$, the intermediate state $\hat{s}$ is given by- $\hat{s}(r) = q_j(r)$ where $j \in \text{loc}(a)$ is such that $j \in \text{gossip}_a((v_a), r)$ therefore we have $\hat{s}(r) = \tau_{\text{dstr}}(\partial_r(\partial_a(t)))_r = \tau_{\text{dstr}}(\partial_a(t))_r$ and $\hat{s} = \tau_{\text{dstr}}(\partial_a(t))$. Let, $\mathscr{A}(ta) = (s', m')$ with $m' = (q', v')$. Then, for each $i \in \text{loc}(a)$, $q'_a(i)$ given by $\hat{s}'$ satisfies $q'_a(i) = \tau_{\text{dstr}}(\partial_i(ta))$. For $i \notin \text{loc}(a)$, $q'(i) = q(i) = \tau_{\text{dstr}}(\partial_i(t))$. As $i \notin \text{loc}(a)$, $\partial_i(ta) = \partial_i(t)$. Therefore, $q'(i) = \tau_{\text{dstr}}(\partial_i(ta))$. Next, s' given by $s_{\overline{a}}$ and $\hat{s}'_a$ satisfies $s' = \tau_{\text{dstr}}(ta)$.

Thus, the induction step is complete.

□

We prove in Lemma 4.36 that at trace t when action a is played this distributed memory automaton successfully updates the a state to $\tau_{\text{dstr}}(ta)_a$ from $\tau_{\text{dstr}}(t)_a$. As a consequence this finite construction allows the distributed system to lift off the distributed strategy from the sequential strategy and follow it. We thus have-

Theorem 4.37. *The distributed strategy τ_{dstr} extracted from the positional sequential strategy τ is finite-state. In fact, it is realized by the memory automaton $\mathscr{A}$ defined above.*

When applied to safety/parity CDM games, the above theorem provides a memory au-

tomaton realizing a distributed winning strategy. Note that A_{gossip} does not depend on the specification of the game $\mathscr{G}$ at all. *Distributed memory complexity* $\max_i |M_i|$ of this memory automaton is bounded by $(\max_i |S_i|)^{|\mathscr{P}|}.(\max_i |V_i|)$. This exponential dependence on $\mathscr{P}$ is due to the fact that each process stores in its local memory state a *global-state* of A. Our next result shows that this exponential dependence on $\mathscr{P}$ is necessary. Its proof is based on an adaptation of the memory lower bound argument for *generalized reachability* in sequential games from [Fijalkow and Horn, 2012b].

Theorem 4.38. *For every k, there exists a global safety CDM game $\mathscr{G} = (A, s_0, \text{Win})$ with $A = (\{S_i\}, \{\xrightarrow{a}\})$ with $|\mathscr{P}| = k+1$ such that the central decision maker process has $O(k)$ local states and every other process has $O(1)$ local states. Further, there exists a distributed winning strategy in $\mathscr{G}$. However, any memory automaton realizing a winning distributed strategy in $\mathscr{G}$ must have at least 2^k local memory states for the central decision maker.*

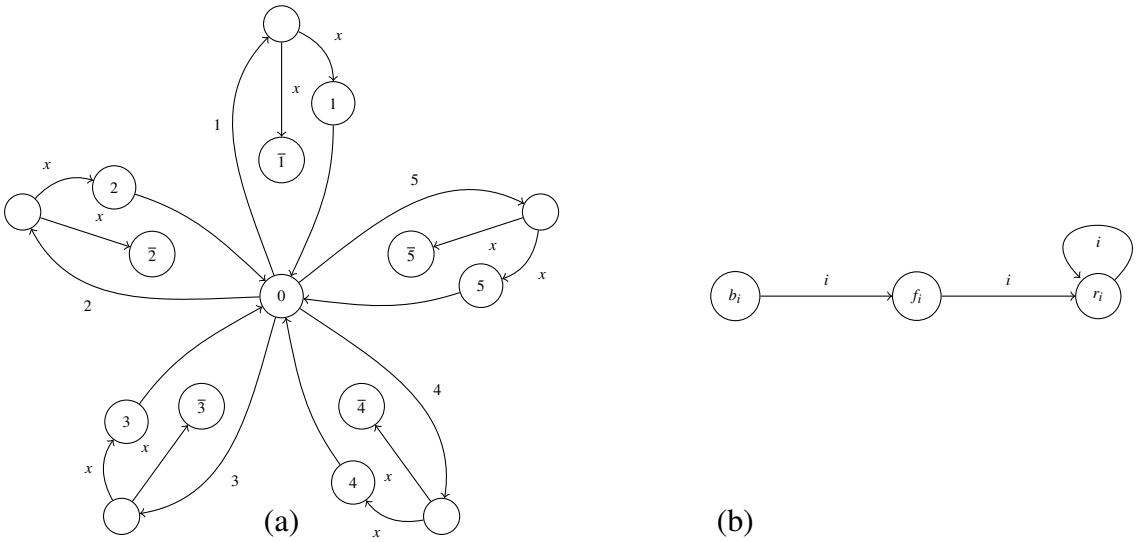


Figure 4.39: Memory lower bound: The pairing (i, r_i) and $(\bar{i}, f_i)$ in any global state makes it unsafe

Proof. We describe the safety game where system needs 2^k memory states to win. The decision maker process is shown in Fig. 4.39(a) for $k = 5$. For the k deterministic processes, the transitions for process i are shown in Fig. 4.39(b). The pairing (i, r_i) and $(\bar{i}, f_i)$ in any global state makes it unsafe.

Action i is a shared action between the CDM and process i. Local actions of the CDM are not named. A play starts from the initial state $(0, (b_i)_{i \in [1..k]})$. First environment chooses a petal

i, and system chooses either for the CDM processes to reach state i before going back to the 0 state (the play goes on), or to reach $\bar{i}$ and to stop the play.

CDM wins with the following strategy- The first time environment plays action i, process 1 goes to state i and returns to initial state, the second time process 1 goes to state $\bar{i}$. This way the system would remain safe. Process 1 has memory automaton $M_1 = 2^{\{1,\ldots k\}}$ and remembers the petals visited previously.Therefore, the size of memory for process 1 is $|M_1| = 2^{\{1,\ldots k\}} = 2^k$. All other processes have only deterministic actions and never use any memory.

Next we prove that there is no winning strategy for this game with less than 2^k memory states in M_1.

Let there be a strategy $\mathscr{A} = (\{S_i \times M_i\}, \{\delta_a\}, (s_0, m_0))$ with $|M_1| < 2^k$. We show that it cannot be winning. We define stopping function as Stop : $M_1 \to 2^{\{1,\ldots k\}}$ where $m \mapsto S_m$ if Process CDM in memory state m chooses $\bar{i}$ when environment plays action $i \in S_m$. Formally,

$$S_m = \{\, i \mid \delta_{ix}(m,0) = (m', \bar{i}) \text{ for some memory state } m' \,\}.$$

Therefore in the memory state m on action $i \notin S_m$ process CDM goes to state i.

As $|M_1| < 2^k$ there is a set which has no preimage in function Stop. We call it X. The only state where environment has more than one action enabled in process CDM is the initial state 0. Environment wins against this strategy in the play described as follows- Each time process CDM is in state 0, and memory state m environment plays an action from $(X - \text{Stop}(m)) \cup (\text{Stop}(m) - X)$. This set is never empty because X is not an image for any memory state.

If environment plays in X for more than $X+1$, then state combination (i, r_i) is seen and therefore system looses. If environment does play an action i from $\text{Stop}(m) - X$ when process CDM is in memory state m then, we can show that action i has not been played earlier. Say in memory state m' action i was played earlier.It could not have been in $X - \text{Stop}(m')$ as $i \notin X$. It could not have been in $\text{Stop}(m') - X$ as then then process 1 would have reached $\bar{i}$, stayed in that state and never return to state 0. As $i \in \text{Stop}(m) - X$ process 1 reaches $\bar{i}$ and stays in that state. Therefore, state i is never visited by process 1. In this case environment wins in the play because then state $(\bar{i}, f_i)$ is seen.

□

4.4 ATS games with two decision makers

It is natural to ask about the decidability status of ATS games in the presence of multiple decision making processes. We have established that when there is a single decision maker, all actions scheduled concurrently with the decision maker are deterministic, and determining whether there exists a winning strategy is an EXPTIME-complete problem. However, as soon as the game involves two decision makers who can make choices concurrently, the problem of determining whether there exists a winning strategy becomes undecidable, even for safety winning conditions. Towards this, we establish the following theorem.

Theorem 4.40. *ATS games with* two *decision makers are undecidable.*

Our proof of the above theorem uses the key ideas from [Gimbert, 2022] which showed that six-process asynchronous control games are undecidable. More precisely, we use the intermediate problem *infinite bipartite coloring* introduced in [Gimbert, 2022] for our reduction. We then revisit the ideas in the proof of the undecidability of six-process asynchronous control games and adapt them to the setting of fourteen-process safety ATS games with only two decision making processes. The undecidability proof relies crucially on the ability of the two decision makers to take truly *concurrent* decisions. If the decision events are causally ordered despite involving disjoint sets of processes, then we can in fact adapt the arguments of the current work to get back the decidability. We have presented these arguments for the CDM setting for convenience.

Proof idea. For a finite set of colors C, an infinite bipartite coloring is a function $f : \mathbb{N} \times \mathbb{N} \to C$ mapping each pair of numbers to a unique color. We use A and B to denote the two infinite numbered sets of nodes in this graph. The coloring constraints allow or disallow colorings adjacent in either one or both partitions as follows. Given $c = f(x,y)$ for $(x,y) \in A \times B$ a constraint specifies if $c' = f(x+1,y)$ or $f(x,y+1)$ or $f(x+1,y+1)$ is allowed or not. cc' is in set upper triangle(UT), lower triangle(LT) or square(S) respectively if $(f(x,y), f(x+1,y))$ or $(f(x,y), f(x,y+1))$ or $(f(x,y), f(x+1,y+1))$ is not allowed. Given a set of constraints, the problem is to decide if there exists a coloring that satisfies the constraints.

[Gimbert, 2022] shows that this infinite graph coloring problem is undecidable and reduces

6 processes asynchronous games to BCP. The ATS games reduced from BCP in a similar fashion satisfy the following interesting properties.

In these games, there are two pools of processes one pool for each partition A and B of the graph. Each pool consists of three processes. These processes participate in deterministic actions among themselves called increments. Each process in a pool participates in two out of three increments. Every three successive increments in a pool constitute a round.

The increments are designed such that the three processes are either all in the same round or in adjacent rounds. The information of which processes is ahead or if both processes are in the same round can be inferred from the state space of the two processes.

Both pools participate in a number of rounds. Thus, each process can keep track of rounds in its causal past. The two pools advance in their rounds concurrently. For any tuple $(x,y) \in A \times B$ a process in pool A, along with a corresponding process in pool B can be queried the color $f(x,y)$ when pools A and B are in rounds x,y. Thus, a process in round x in Pool A is held answerable for node x in Partition A. Concurrently, another process in pool A, along with a corresponding process in pool B can be queried the color of round (x,y) or an adjacent round $(x+1,y)$ or $(x,y+1)$ or $(x+1,y+1)$.Any one of these queries can be made in different plays where the first query for $f(x,y)$ has the same causal past. As the two queries are concurrent in the corresponding play, it is ensured that the processes give the same answer for the same query or it allows a suitable constraint to be checked.

The query for $f(x,y)$ can also be made concurrently to queries for $f(x-1,y), f(x,y-1)$ or $f(x-1,y-1)$ in different plays. This is the reason why the system is forced to give a valid coloring satisfying all constraints.

This suggests that 6DM is undecidable. Further, the processes responsible for making the non-deterministic choices of a color can be w.l.o.g. to be three processes from a single pool. We could then infer that 3DM is undecidable. We extend this proof by adding more processes to the game to serve our purpose of proving that 2DM to be undecidable.

We add two oracle processes external to the pools. We replace each process in a pool by a pair of processes each corresponding to an oracle. Now, the two oracles always participate in the two queries that are done concurrently. The query responses by each process in a pair are always the same because they can be asked the same query concurrently as well. Thus, all

non-deterministic choices involve the oracles making 2DM undecidable. □

4.4.1 Infinite bipartite coloring problem and its undecidability

Definition 4.41 (infinite bipartite C-coloring)**.** . *For a finite set of colors C, an infinite bipartite coloring is a function* $f : \mathbb{N} \times \mathbb{N} \to C$ *mapping each pair of numbers to a unique color.* $f(0,0)$ *is called the initial color.*

A coloring $f : \mathbb{N} \times \mathbb{N} \to C$ induces the following patterns.

- the squares of f are all pairs $\{(f(x,y), f(x+1,y+1) \mid (x,y) \in \mathbb{N} \times \mathbb{N}\}$.
- the upper-triangles of f are all pairs $\{(f(x,y), f(x+1,y) \mid (x,y) \in \mathbb{N} \times \mathbb{N}\}$.
- the lower-triangles of f are all pairs $\{(f(x,y), f(x,y+1) \mid (x,y) \in \mathbb{N} \times \mathbb{N}\}$.

A coloring constraint is given by three subsets S, UT, LT of C^2 called the forbidden patterns, and a subset C_i of C, the allowed initial colors. A coloring f satisfies the constraint if its initial color is in C_i, and none of the patterns induced by f are forbidden. The infinite bipartite coloring *problem* asks whether there exists an infinite coloring satisfying given constraints on the induced patterns. While the paper [Gimbert, 2022] formally presents a reduction from PCP to BCP, it explicitly states(see conclusion of [Gimbert, 2022]) that the construction can be adapted to encode infinite PCP instances in order to handle deadlock-freeness objectives. Consequently, it can also be adapted to reduce ω-PCP to ω-BCP.

An instance of ω-PCP is a finite collection of tiles $(u_i, v_i)_{i \in I}$ where, each tile (u_i, v_i) consists of two non-empty strings in Σ^*, u_i the top word and v_i the bottom word. The problem is to determine whether there exists an infinite sequence of indices in I such that, the concatenation of tiles in this sequence produces the same top word and bottom word. In case such a sequence exists, the ω-PCP instance is said to have a solution. Checking whether an instance has such a solution is known to be undecidable.

4.4.2 Undecidability of 2 decision maker case with 12 more deterministic processes

The main result of this section is:

Theorem 4.42. *There is an effective reduction from the infinite bipartite coloring problem to 2 decision maker problem.*

An ATS game $\mathscr{G} = (A, s_0, \text{Win})$ is said be a 2DM game if there exists two processes $1, 2 \in \mathscr{P}$ such that, at least one of them participates in *every* non-deterministic action of the ATS $A = (\{S_i\}, \{\xrightarrow{a}\})$. Recall that an action a of A is non-deterministic if there exists $s_a, s'_a, s''_a \in S_a$ such that both (s_a, s'_a) and (s_a, s''_a) are a-transitions (that is, belong to $\xrightarrow{a}$) and $s'_a \neq s''_a$. Stated differently, if neither 1 nor 2 participate in an action b, then the local transition function $\xrightarrow{b}$ of A is deterministic.

We show how to effectively transform an instance (C_i, S, UT, LT) of infinite bipartite coloring problem on a set of colors C into a distributed 2DM game G such that:

Lemma 4.43. *There is a winning strategy in G iff there is an infinite bipartite coloring satisfying the constraints* (C_i, S, UT, LT).

In the rest of the section we describe the construction of the game G and then prove the next Lemma 4.43.

Game processes: In the game, there are two pools, T and B, each containing six processes. Each pool is divided into three pairs of processes, labeled as $0, 1$ and 2. Additionally, there are two oracles, O_0 and O_1. These are the non-deterministic processes that participate in every non-deterministic action.

Process Pairs: In each pool X, a pair X_i consists of processes X_{i0} and X_{i1}, where i ranges from 0 to 2. For example, in pool T, the pairs are $\{T_{00}, T_{01}\}, \{T_{10}, T_{11}\}$ and $\{T_{20}, T_{21}\}$.

Next, we describe the actions, states, and transitions shared among these processes. Fig. 4.44 illustrates the events of this game. A play corresponds to a maximal conflict-free set of events. Two events are in minimal conflict if they are not causally related, share at least one common participating process, and no other pair of events in their causal past is in conflict.

Deterministic Increment Synchronization: During each increment, two pairs of processes from a pool synchronize. On I_{Xi} , the pairs i and $i+1 \mod 3$ pair of pool X synchronize. Specifically, the synchronization on increment action I_{Xi} involves processes $X_{i0}, X_{i1}, X_{(i+1 \mod 3)0}$, and $X_{(i+1 \mod 3)1}$, where i is the pair index and X is the pool.

Teams and Oracle non-deterministic Synchronization: For each pair $i \in \{0,1,2\}$ and for each oracle $o \in \{o_0, o_1\}$, there is a team D_{io} consisting of three members: Processes T_{io}, B_{io} from the two pools and the oracle o. This team synchronizes on the action C_{io}.

This description clarifies the structure of the pools, the synchronization among the processes. We use X to denote pools, i to denote pairs, o to denote oracles and D_{io} to denote teams.

State space of each process in a pool contains the following information. An "increment allowed" bit is set to true for the pair $X_0 o$ and false for every other pair in pool X. This bit is flipped for each process on every increment. An increment I_{Xi} is allowed only if this bit is true in pair X_{i0}, X_{i1}.A round for pool X consists of three increments I_{X0}, I_{X1}, I_{X2}. Each process in the pool participates in 2 increments per round.

No two increments in a pool are concurrent. A pool X is in round x after $(I_{X0}I_{X1}I_{X2})^x I_{X0}$ $(I_{X0}I_{X1}I_{X2})^x I_{X0}I_{X1}$ and $(I_{X0}I_{X1}I_{X2})^x I_{X0}I_{X1}I_{X2}$. A " $\mod 4$ counter" is maintained with values in $\{0,1,2,3\}$ which is updated on each increment. This allows the processes to maintain a "round parity bit". Values $0,1$ of the counter indicate even round parity set to 0 and values $2,3$ of the counter indicate odd round parity set to 1.

The state space of each oracle contains the information of a color. When the environment challenges a team D_{io} for color it involves one process X_{io} from each pool and an oracle O_o. These then have the information of top round parity, bottom round parity, and a color. And the team is expected to respond with the current round edge color.

When a top pool process T_{io} is in round x and a bottom pool process B_{io} is in round y, the environment can ask team D_{io}- "what is the color of edge (x,y)?" using a check action C_{io}. After the check, the couple picks a color value c for $f(x,y)$ and stores the answer in its next state which is blocking. Intuitively, if the strategy of the players makes them cheat or describe a coloring f which does not satisfy the constraints, the environment can trigger one or two checks which make the system lose. After this any enabled increments may be scheduled for

other processes. Then a check with another couple is inevitably scheduled. The safety of the blocking states of the participating processes decides the winner of the play. Two such questions can be asked concurrently either in the same round or in consecutive rounds. After that another couple participates in a check action before the end of next round There are only two possible outcomes: The next state for both couples is either safe and the system wins or is unsafe and the system loses.

A typical play in this game is either a deterministic play product of $(I_{X_0}I_{X_1}I_{X_2})^*$ for $X = T$ and $X = B$ or we can have two non-deterministic actions. Any process participating in a non-deterministic action goes to a blocking state and the winner is determined by the safety of this state.

Before stating the winning conditions and its implications notice that whether or not two processes are in the same round can be known by keeping track of whether it is an odd or an even round for a given process. In both pools process X_1 and X_2 are always in the same round and process X_3 may be one round behind, or in the same round, but never ahead. Therefore, along with parity information, one can discern if process X_3 is in the same round or previous round. From the parity of the round and the process numbers, one can derive if they are in the "same round" or "who is ahead".

A team D_{io} after a synchronization is said to be in the round (x,y) if the pool T process (pool B process resp) it participated with is in round x (round y resp). Wlog we assume that the couple is in round (x,y) after its participation in a check. After another team $D_{i'o'}$ has also participated in some check action-

1. The two teams may be in the same round (x,y) and hence in *same round relation*.
2. If in second team $D_{i'o'}$ both pools are ahead i.e. $(x+1,y+1)$ of first team D_{io} we say that they are in *square round relation*.
3. If Top pool is in different rounds and Bottom pool is in same round, i.e. second couple $D_{i'o'}$ is in round $(x+1,y)$ we say that they are in *upper triangle relation*.
4. Similarly for lower triangle, if second couple is in round $(x,y+1)$, we say that they are in *lower triangle relation*.

After both checks we would want the two teams $D_{io}, D_{i'o'}$ should satisfy the constraints for

the round index relation they are in. We now state the *winning conditions* and its implications. Set of unsafe states is the union of the following sets. Any two teams $D_{io}, D_{i'o'}$-

1. in the same round relation with different colors is an unsafe state.
2. in square round relation, not satisfing the square constraint are in unsafe state.
3. in upper triangle(lower triangle resp) relation, not satisfing the upper triangle (lower triangle resp) constraint are in unsafe state.
4. Additionally, if both teams are in round 0 and their answer c is a color which is not initial $c \notin C_i$. The condition "has played a single increment " can be stored in the state space of the processes, and used to allow or not this transition.

Therefore the only way for system to lose is when the two couples choose inconsistent answers and land in an unsafe state or choose a wrong initial color. Losing can occur in four different ways. Three of them correspond to the constraints in the definition of the BIPARTITE COLORING PROBLEM. If no checks are made by the environment the play stays deterministic and is won by the system. But in general, a winning strategy should also react correctly to non-deterministic actions or checks.

Proof of Lemma 4.43. ($\Leftarrow$)We start with the converse implication that is if there is a bipartite coloring f satisfying the constraints then there is a winning strategy in G. Assume that there is an infinite bipartite coloring $f : \mathbb{N} \times \mathbb{N} \to C$ satisfying the constraints (C_i, S, UT, LT). A winning strategy for the system, when the environment does a check with team D_{io} is to find its round number (x, y) from the combined causal past of participating processes and answer with $f(x, y)$.

Since f is well defined the first unsafe set is never seen. Since f also satisfies the constraints, after two check actions, the participating processes are in safe blocking state. As no further transitions are possible the system wins.

($\Rightarrow$) We now prove the direct implication. Assume there is a winning strategy σ in G . We define a bipartite coloring f and prove that it satisfies the constraints.

Given a trace u, let $u_{x,0} = (I_{T,0}I_{T,1}I_{T,2})^x I_{T,0}$, $u_{x,1} = (I_{T,0}I_{T,1}I_{T,2})^x I_{T,0}I_{T,1}$, and $u_{x,2} = (I_{T,0}I_{T,1}I_{T,2})^x I_{T,0}I_{T,1}I_{T,2}$; similarly we define $v_{y,0}, v_{y,1}, v_{y,2}$ using the bottom process actions.

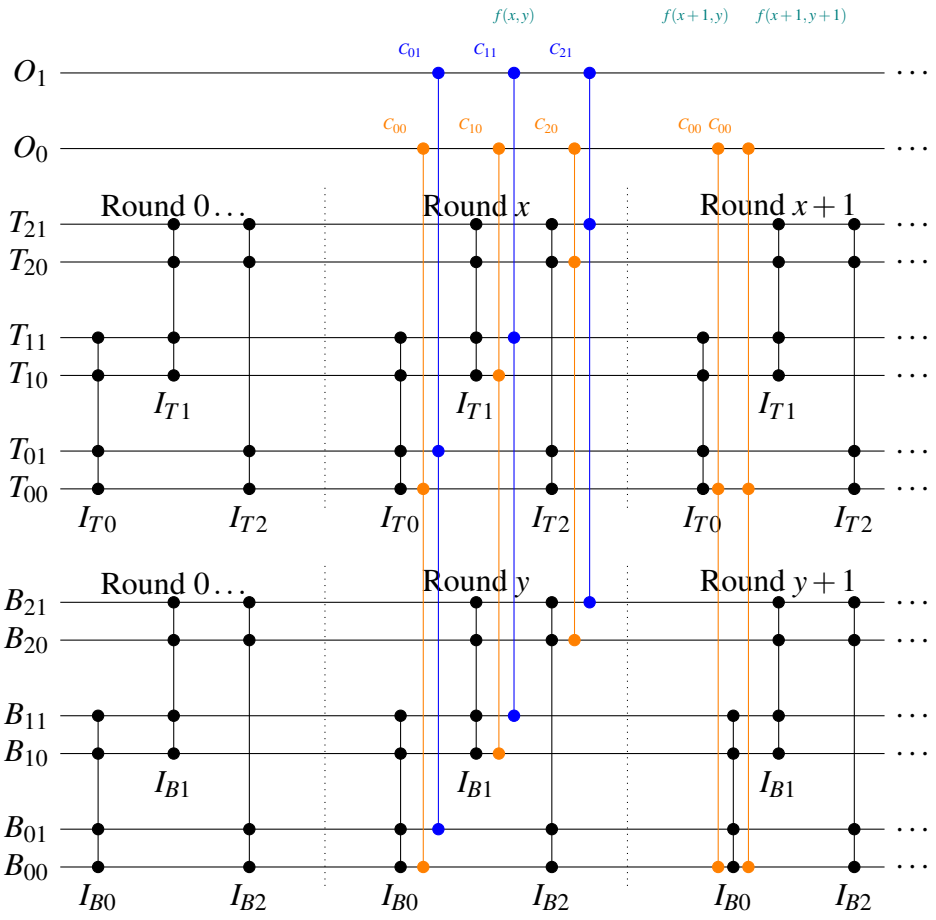


Figure 4.44: *Undecidability of 2 decision maker: A play- Each round is separated by dotted lines. Non-deterministic actions can occur at various occasions as pointed out by various check points in which one of the oracles participates. Notice that any two consecutive checkpoints are concurrent and thus can occur in the same play.*

Then we define $f(x,y) = \sigma(u_{x,0}v_{y,0}c_{00})_{O_0}$. We randomly fix couple 00 to see the color response when oracle 0 asks the color of that edge. We now show that f satisfies the constraints.

We will show that we have fixed the unsafe state so that if a constraint is not satisfied the environment can force a visit to an unsafe state. For any pool, in any round, checks can happen at 3 points that is after each increment by any one of the oracles. A check after the first increment of a round can happen in parallel with the last round of one or both pools and so should be consistent with the choice in last round. The check after the last increment of a round can happen in parallel with the next round and so should be consistent with the choice in last round. But both of these checks can happen in parallel with the check after the second increment which is why the choice of the color should be singular. More formally, we state the following.

An unsafe state is possible in case one of the following conditions hold: There are four ways to lose, which require that two checks occur in parallel, for two different teams $D_{io}, D_{i'o'}$ respectively. This leads to two parallel answers corresponding to colors c and c', respectively. The conditions for losing rely on the round index relation of the two checks.

We see that $\sigma(u_{x,0}v_{y,0}c_{00})_{O_0} = \sigma(u_{x,0}v_{y,0}c_{01})_{O_1} = \sigma(u_{x,1}v_{y,1}c_{10})_{O_0} = \sigma(u_{x,1}v_{y,1}c_{11})_{O_1} = \sigma(u_{x,2}v_{y,2}c_{20})_{O_0} = \sigma(u_{x,2}v_{y,2}c_{21})_{O_1}$.

As there is no conflicting action in each consecutive pair, they can occur in the same play and the first safe set ensures equality. Notice that, as the checks are concurrent the answers for a check do not change in different plays, yet their presence in different plays allows the environment to test their equality or related constraint.

The pair $(\sigma(u_{x,2}v_{y,2}c_{21})_{O_1}, \sigma(u_{x+1,0}v_{y+1,0}c_{00})_{O_0})$ satisfies the square constraint. As only process pair X_2 is blocked process pairs X_0 and X_1 can still move to the next round either in both pools or in one of the pools. And the check on team 0 is independent of check on couple 2. Therefore pair $(f(x,y), f(x+1,y+1))$ satisfy the square constraint.

Similarly because of the pair $(\sigma(u_{x,2}v_{y,2}c_{21})_{O_1}, \sigma(u_{x+1,0}v_{y,2}c_{00})_{O_0})$ being in safe set, the values $(f(x,y), f(x+1,y)$ satisfies the upper triangle constraint. And similarly because of the pair $(\sigma(u_{x,2}v_{y,2}c_{21})_{O_1},\ \sigma(u_{x,2}v_{y+1,0}c_{00})_{O_0})$ being in the safe set the values $(f(x,y), f(x,y+1))$ satisfies the lower triangle constraint. As $\sigma(u_{0,0}v_{0,0}c_{00})_{O_0}$ is a safe state $f(0,0)$ is an initial color. We then see that all required constraints are satisfied.

□

Chapter 5

Control games and their equivalence to ATS games

Control games have been extensively studied [Genest et al., 2013; Muscholl and Walukiewicz, 2014; Gimbert, 2017, 2022] and are recognized as undecidable. Moreover, ATS games exhibit equivalence to Control games, meaning that for every ATS game, a equivalent control game can be constructed. This equivalence ensures that if a system possesses a winning strategy in one type of game, it will possess it in the other as well. Conversely, an ATS game can be constructed for every control game, maintaining this symmetry.

In this chapter, we establish an initial connection between control games and asynchronous transition system (ATS) games by showing that they are equivalent in expressive power. Control games—well-studied and known to be undecidable—can be systematically transformed into equivalent ATS games, and conversely, ATS games can be encoded as control games. This correspondence highlights a close relationship between the two models and will eventually allow insights from one model to be translated to the other. For this we describe the control games more formally.

5.1 Control Games

We fix a distributed alphabet $\widetilde{\Pi}$ over a fixed team $\mathscr{P}$ of processes. Let $\Pi = \bigcup_{i \in \mathscr{P}} \Pi_i$ be the total alphabet, that is, the set of all actions. In a control game [Genest et al., 2012], Π is partitioned into two sets of actions: controllable actions, denoted as Π^{con}, and uncontrollable actions, denoted as Π^{ucon}. This partitioning $\Pi = \Pi^{con} \sqcup \Pi^{ucon}$ reflects the distinct roles played by the system and the environment in influencing the game dynamics.

Central to the study of control games is the concept of a deterministic automaton, represented as $\mathscr{B} = (\{Q_i\}, \{\xrightarrow[c]{a}\})$, where Q is the set of states and $\xrightarrow[c]{a}$ denotes the deterministic transition relation for each action a. Given an initial state, and a trace, the state of the automaton is uniquely determined by the deterministic transitions on each successive action. Recall that given an initial state s_0, $\mathscr{B}(u)$ denotes the final global state attained on the unique run of $\mathscr{B}$ on the finite trace u. Specifically the unique run ρ on a trace u from initial state s is given by $\rho(c) = \mathscr{B}(c)$ for all $c \in C_u$.

For our convenience, we state the abstract winning condition for control games as a set of traces $\mathbb{W}\text{in}$.

Definition 5.1. *A control game is of the form* $\mathbb{G} = (\mathscr{B}, s_0, \mathbb{W}\text{in})$ *where*

- $\mathscr{B} = (\{Q_i\}, \{\xrightarrow{a}\})$ *is a deterministic asynchronous transition system over* $\widetilde{\Pi}$.
- $q_0 \in Q_{\mathscr{P}}$ *is an initial global state of* $\mathscr{B}$.
- $\mathbb{W}\text{in}$ *is a specification of the winning condition.*

We consider only state based winning conditions. A state-based winning condition in a control game means that whether a play is considered winning or not depends only on the states visited during that play, rather than on the specific set of actions or their order.

A trace shall be used to represent the set of actions taken during the game and the partial order among them thus representing a play. The moves of a play is then a trace that has a run in $\mathscr{B}$ from initial state s_0. A run of ATS $\mathscr{B}$ is defined in the earlier section. The system moves of a play are given by a function that maps each finite trace to the next allowed actions.

Definition 5.2. *The trace u is a (control) play of the game* $\mathbb{G} = (\mathscr{B}, s_0, \mathbb{W}\text{in})$ *if,* $u = (E, \leq, \lambda)$ *is a trace over* $\widetilde{\Pi}$ *and has a run in* $\mathscr{B}$ *from* s_0*. i.e.*

$$\forall c, c' \in C_u : [c \xrightarrow{a}_u c' \implies \mathscr{B}(c) \xrightarrow{a} \mathscr{B}(c')]$$

A play u is said to be maximal if there is no play u' *such that u is a proper prefix of* u'*.*

To effectively navigate a control game $\mathbb{G}$, players employ strategies.

Definition 5.3. *A (control) strategy in* $\mathbb{G} = (\mathscr{B}, s_0, \mathbb{W}\text{in})$ *is a tuple* $\tau = (\tau_i)$ *where* τ_i *is a* strategy for process *i. A strategy for process i is a function* $\tau_i : \mathrm{TR}^*(\Pi) \to 2^{\Pi}$ *that associates with each finite trace u a "set of actions" such that:*

1. *all environment actions are allowed:* $\Pi^{ucon} \subseteq \tau_i(u)$
2. *the decision depends only on the view of the process:* $\tau_i(u) = \tau_i(\partial_i(u))$

The first condition ensures that any system move does not restrict the environment move. The second condition ensures that any process makes its choices based only on the causal past.

Definition 5.4. *Let* $\tau = (\tau_i)_{i \in \mathscr{P}}$ *be a control strategy in* $\mathbb{G}$*. A control play u in* $\mathbb{G}$ *from initial state s is said to* conform τ *if*

$$\forall c, c' \in C_u : [c \xrightarrow{a}_u c' \implies \forall i \in \mathrm{loc}(a) : a \in \tau_i(c)]$$

We denote the set of all plays that conform τ *as* $\mathrm{plays}(\mathscr{B}, \tau)$*. A control strategy* τ *is said to be winning if all maximal plays conforming it are in winning set* $\mathbb{W}$in*.*

This condition ensures that any configuration is derived from its immediate prefix and the system choice given by the strategy.

Recall that the decidability question for both games is: “Does there exist a winning strategy?” We describe the various classes of control games for which this question has been

addressed. One line of work studies *local reachability winning conditions* [Genest et al., 2013], where every process has a set of target states $F_i \subseteq S_i$. These target states are blocking, meaning they have no outgoing transitions. In particular, *tree architecture games*—where the communication architecture is acyclic, and each process communicates only with its parent and children—are shown to be decidable in non-elementary time, with matching lower bounds. Another setting considers *ω-regular local specifications* , [Muscholl and Walukiewicz, 2014], which are also shown to be decidable in tree architectures. A more general class is that of *decomposable games with termination conditions* [Gimbert, 2017], where each process is required to terminate its computation in finite time in a final state. Decomposable games subsume several known formulations, including tree architectures, series-parallel systems [Gastin et al., 2004], and connectedly communicating architectures [Madhusudan et al., 2005], and are proven to be decidable. Despite these positive results, there are strong *undecidability results*: even for simple objectives such as local reachability, termination, or deadlock-freeness, asynchronous games with as few as six processes are undecidable [Gimbert, 2022], underscoring the inherent complexity of distributed synthesis problems.

5.2 Equivalence of Control and ATS Games under State-Based Conditions

In control games actions are partitioned into controllable and uncontrollable sets. Each individual process can enable or disable a subset of controllable actions based on its own causal past; these choices are then scheduled, allowing processes to share and update their causal pasts. To construct an equivalent ATS game, we introduce nondeterministic local actions that let each process select a subset of controllable actions. Subsequently, deterministic transitions allow the environment to act as a scheduler, determining the next global state, after which the participating processes exchange their causal pasts.

We illustrate this in Fig. 5.5. Say $\mathrm{loc}(b) = \{i, j\}$ and s_b is a b-state such that $s_b(i) = s_i$ and $s_b(j) = s_j$. Then Part(a) shows an interaction in a control game in the presence of a controller. Process i and j based on their local pasts allow the set of actions $\{a, b\}$ and $\{a, b, c\}$ respectively. The control choices are depicted in **bold** inside circles in the Fig. 5.5. Then action b present

Figure 5.5: Control to ATS

Figure 5.6: ATS to control

in the chosen sets of all participating processes is scheduled. The next state is given by the deterministic transition system. In Part(b) the same interaction is played out in an ATS game. To construct an ATS game we introduce local non deterministic choice actions o_i for each process to allow each process to make a choice. A strategy in the ATS game capture the controller choices in the local choices made on these actions. The system choices here based on the controller are depicted in **bold**.

We next move to the construction of an equivalent control game from given a ATS game. In this translation, environment moves in the ATS game—where the environment plays actions—are modeled as uncontrollable actions in the control game. Controllable actions correspond to joint states for each action in the ATS game, enabling the system to choose its next move. In an ATS game, when the environment plays an action, all processes participating in that action know the entire causal past before deciding on the next state. To capture this in a control game, each environment move (modeled as an uncontrollable action) deterministically stores this move in the participating processes, effectively synchronizing their causal pasts. Each process can then enable or disable the next combined state of the participating processes, which is subsequently scheduled as a controllable action.

In Fig. 5.6 Part(a) an ATS game interaction is shown. Environment chooses an action b and participating processes share their histories and then choose b-state s'_b among the possible successors. The system choice is depicted in **bold**. To simulate this same interaction in a

control game we introduce joint states as controllable actions as shown in Part(b). In the control game, after b is scheduled, which is modeled as an uncontrollable action, the environment choice is deterministically stored in all participating processes, also allowing them to share histories. They then individually make their choice of the next joint state known in the form of a controllable action. This action, when scheduled, changes the joint state to the desired one deterministically. Here also, the control choices derived from the system choices in the ATS game are depicted in **bold** inside circles.

These constructions are formally stated in later sections and establish a correspondence between plays in the two game models, which in turn ensures that strategies and winning conditions are preserved.

5.2.1 Reduction from Control Games to ATS Games

In the following lemma, we state that for every control game we can construct an equivalent ATS game. It is equivalent in the sense that there exists a winning strategy for the Control game iff there exists a winning strategy for the corresponding ATS game. This section is dedicated to construct such a game and proving its equivalence.

Lemma 5.7. *[Con $<$ ATS] For any state based winning condition given an control game $\mathbb{G}$ over distributed alphabet $\Pi = \Pi^{con} \sqcup \Pi^{ucon}$ we can construct an ATS game $\mathbb{G}'$ over a distributed alphabet Π' with the same number of processes and comparable number of actions such that there is a wining strategy in $\mathbb{G}$ iff there is a winning strategy in $\mathbb{G}'$.*

Construction: Given an control game $\mathbb{G} = (\mathscr{B}, \mathbb{W}\text{in})$, where $\mathscr{B} = (\{Q_i\}, \{\ \xrightarrow[c]{a}\})$ is a deterministic ATS over $\Pi = \Pi^{con} \sqcup \Pi^{ucon}$, we construct an ATS game $\mathbb{G}' = (\mathscr{B}', \mathbb{W}\text{in}')$ where $\mathscr{B}' = (\{Q'_i\}, \{\overset{a}{\hookrightarrow}\})$ is a non deterministic ATS over Π' as follows. We use loc' to denote the location function in the ATS game.i.e. $i \in \text{loc}'(a)$ if $a \in \Pi'_i$:

1. **States:** The set of states Q'_i for any process consists of the set of pure states and impure states. The set of pure states is given by Q_i. The set of impure states are Q_i states augmented with a set of actions that the process can choose in that state, i.e., $Q_i \times 2^{\Pi}$.

Therefore,

$$Q_i' = Q_i \cup (Q_i \times 2^{\Pi}).$$

2. **Actions:** The set of actions Π' is the union of two types of actions. Actions in Π are deterministic. Another set of actions we call *choice actions* allows non-determinism. It contains an action o_i for each process i. These choice actions are enabled at every pure state and allow the system to specify which deterministic actions are permitted by the system in the ATS game.

$$\Pi' = \Pi \cup \{o_i \mid i \in \mathscr{P}\}.$$

The location of each choice action o_i is given by $\mathrm{loc}'(o_i) = \{i\}$, while for action $a \in \Pi$: $\mathrm{loc}'(a) = \mathrm{loc}(a)$. In other words,

$$\Pi_i' = \Pi_i \cup \{o_i\}$$

3. **Transitions on choice actions from pure states:** For each $o_i \in \Pi_i'$

$$q_i \overset{o_i}{\hookrightarrow} (q_i, X),$$

is a transition if $q_i \in Q$ and X is any set of actions in Π containing all uncontrollable actions i.e. $\Pi^{ucon} \subseteq X \subseteq \Pi$. In a pure state, only this choice action is allowed, enabling the system to select its controllable actions without interfering with uncontrollable ones.

4. **Transitions on deterministic actions from impure states:** Once the processes have chosen their action sets, the next state is determined by the chosen actions. For $a \in \Pi$ where $\mathrm{loc}(a) = \{i_1, i_2, \dots\}$,

$$((q_{i_1}, X_{i_1}), (q_{i_2}, X_{i_2}), \dots) \overset{a}{\hookrightarrow} q_a',$$

is a transition if $a \in X_{i_1} \cap X_{i_2} \cap \ldots$ and q'_a is the unique a successor of $q_a = (q_{i_1}, q_{i_2} \ldots)$ state i.e. $q_a \xrightarrow[c]{a} q'_a$. This allows the environment to choose the next action among all available options from the system and uncontrollable actions.

5. **Play correspondence:**

   We define a function $\text{ContoAts}_{Con} : \text{TR}^*(\Pi) \to \text{TR}^*(\Pi')$ that maps a play u in control game $\mathbb{G}$ to a unique trace t in $\mathscr{B}'$ in game $\mathbb{G}'$. For example, if $u = abc$ and $t = \text{ContoAts}_{Con}(u)$ then

   $$t = o_a a o_b b o_c c \quad \text{where} \quad o_a = o_{i_1} o_{i_2} \ldots o_{i_k} \text{ for } \text{loc}(a) = \{i_1, i_2, \ldots, i_k\},$$

   We then define the function $\text{ContoAts}_{Ats} : \text{TR}^*(\Pi') \to \text{TR}^*(\Pi)$ as the inverse of ContoAts_{Con} which maps a trace t to u iff u is the restriction of t to the action set Π.

6. **Winning Conditions:**‘ The winning set $\mathbb{W}\text{in}'$ in the ATS game is defined as the set of plays corresponding to the winning set $\mathbb{W}\text{in}$ in the control game. i.e. $\mathbb{W}\text{in}' = \{(t, \rho) \text{ a play in } \mathbb{G}' \mid \text{ContoAts}_{Ats}(t) \in \mathbb{W}\text{in}\}$. Note that if the winning condition $\mathbb{W}\text{in}$ is state based the winning condition $\mathbb{W}\text{in}'$ can also be expressed as a state based winning condition.

   Note that u has a run in $\mathscr{B}$ if and only if $\text{ContoAts}_{Con}(u)$ has a run in $\mathscr{B}'$.

Natural Strategy in $\mathbb{G}'$ derived from $\mathbb{G}$: Given a control strategy τ_i in $\mathbb{G}$ from state q_0, we construct a corresponding ATS strategy τ' from q_0 in $\mathbb{G}'$ by selecting the next set of allowed actions according to τ_i. For ε $\tau'(\varepsilon) = q_0$.

Let t be a trace pointed in local action o_i. Then the choice of τ is stored in the next state by strategy τ' along with the current state. The present state allows for the validity of transition in $\mathbb{G}$.

$$\tau'(t)_i = (\mathscr{B}(u)_i, \tau_i(u)) : u = \text{ContoAts}_{Ats}(t)$$

For t pointed in an action $a \in \Pi$, action a is deterministic and

$$\tau'(t)_a = \mathscr{B}(u)_a : u = \text{ContoAts}_{Ats}(t)$$

Note that control choices in $\mathbb{G}$ in control strategy τ are *non-deterministic*: the actions in a play are consistent with one of many possible control choices, and only one of them is scheduled. In contrast, in $\mathbb{G}'$, the system's choice in ATS strategy τ' is deterministic. Among the choices given by control one can be randomly chosen using the combined knowledge of the participating processes.

This resolution cannot be made deterministic in $\mathbb{G}$, since choices there are made by each process in isolation. Hence, a play u in $\mathbb{G}$ conforming τ corresponds to the unique play $(\text{ContoAts}_{Con}(u), \tau')$ in $\mathbb{G}'$. Moreover, although $\mathscr{B}'$ is non-deterministic, the states visited in a run, when restricted to Q_i, coincide with those in the corresponding run of $\mathscr{B}$.

Claim 5.8. *If τ is a control strategy in $\mathbb{G}$ from state q_0 then τ' is a distributed strategy in $\mathbb{G}'$ from q_0.*

Proof. We know that $\tau'(\varepsilon) = q_0$. Now we prove that τ' respects the transitions of $\mathscr{B}'$. Let $u, ua \in \text{plays}(\mathscr{B}, \tau)$ and $t = \text{ContoAts}_{Con}(u)$ be in the domain of τ'. We prove that $to_a a$ is in the domain of τ' and for $i \in \text{loc}(a)$, $\tau'(t) \overset{o_i}{\hookrightarrow} \tau'(to_i)$, $\tau'(to_a) \overset{a}{\hookrightarrow} \tau'(to_a a)$.

As $u = \text{ContoAts}_{Ats}(t)$ we have $\tau'(t) = \mathscr{B}(u)$. Then, for $i \in \text{loc}(a)$, $\tau'(t) \overset{o_i}{\hookrightarrow} \tau'(to_i)$ is established by Item 3 in the construction. $\tau'(to_a) \overset{a}{\hookrightarrow} \tau'(to_a a)$ is because of Item 4.

Recall that control choices in $\mathbb{G}$ in control strategy τ are non-deterministic. Say aDb and $ub, ua \in \text{plays}(\mathscr{B}, \tau)$. Then in strategy τ' both plays $(uo_a a, \tau')$ and $(uo_b b, \tau')$ are valid and the environment can play either a or b after $to_{\text{loc}(a) \cup \text{loc}(b)}$.

□

Claim 5.9. *τ' is winning in $\mathbb{G}'$ iff control strategy τ_i is winning in $\mathbb{G}$.*

Proof. A play (t, τ') in $\mathbb{G}'$ corresponds to the unique play $\text{ContoAts}_{Ats}(t)$ in $\mathbb{G}$ that conforms to τ . Moreover, although $\mathscr{B}'$ is non-deterministic, the states visited in a run, when restricted to

Q_i, coincide with those in the corresponding run of $\mathscr{B}$. Therefore, τ' is winning in $\mathbb{G}'$ iff control strategy τ_i is winning in $\mathbb{G}$.

□

Natural Strategy in $\mathbb{G}$ derived from $\mathbb{G}'$: Given a strategy σ in $\mathbb{G}'$ from initial state $q_0 \in Q_{\mathscr{P}}$, we construct a corresponding strategy σ'_i in $\mathbb{G}$ from initial state q_0 by selecting the next set of allowed actions based on σ. Note that for any valid strategy σ in $\mathbb{G}'$, $\sigma(\text{ContoAts}_{Con}(u)) = \mathscr{B}(u)$.

$$\sigma'_i(u) = X : \sigma(\text{ContoAts}_{Con}(u)o_i)_i = (q_i, X) \text{ for } q_i = \mathscr{B}(u)_i.$$

Note that given strategy τ in $\mathbb{G}$ let τ' be the derived strategy. Then, the strategy τ'' for game $\mathbb{G}$ derived from τ' is equivalent to τ in the sense that, whenever u is a play conforming to τ then $\forall i : \tau_i(u) = \tau''_i(u)$. This implies, $\text{plays}(\mathscr{B}, \tau) = \text{plays}(\mathscr{B}, \tau'')$. Thus, the following statements can be proved in exactly the same way as before in the construction of τ' from τ. For completeness we present the proofs as well.

Claim 5.10. *If σ is a distributed strategy in $\mathbb{G}'$ then σ' is a control strategy in $\mathbb{G}$.*

Proof. We know that $\sigma'_i(u) = X$ where $\sigma(\text{ContoAts}_{Con}(u)o_i)_i = (q_i, X)$ for some q_i.

As σ is a distributed strategy for any trace $\sigma(t)_i = \sigma(\partial_i(t))_i$. Because of the definition of ContoAts_{Con} we have if $\text{ContoAts}_{Con}(u) = t$ then $\text{ContoAts}_{Con}(\partial_i(u)) = \partial_i(t)$. Also as o_i is a local action, $\partial_i(uo_i) = \partial_i(u)o_i$. Therefore, $\sigma'_i(u) = \sigma'_i(\partial_i(u))$ and σ' is a control strategy.

□

Claim 5.11. *Play u conforms strategy σ' in game $\mathbb{G}$ iff $(\text{ContoAts}_{Con}(u), \sigma)$ is a play that is in game $\mathbb{G}'$ i.e. $\text{ContoAts}_{Con}(u)$ is in the domain of σ.*

Proof. Let play u conform to strategy τ_i. Let $t = \text{ContoAts}_{Con}(u)$. Then, for a configuration $c \in C_t$ pointed at o_i, $\sigma(c) = (\mathscr{B}(\text{ContoAts}_{Ats}(c))_i, \sigma'(\text{ContoAts}_{Ats}(c))) : c' = \text{ContoAts}_{Ats}(c) \in C_u$. For $c \in C_t$ pointed in an action $a \in \Pi$: $\tau'(c) = \mathscr{B}(\text{ContoAts}_{Ats}(c))$ where a is a deterministic action. Item 3,Item 4 in construction ensure the validity of the transitions thus enabling the

appropriate actions. Therefore, $\text{ContoAts}_{Con}(u)$ is in the domain of σ. The converse follows in the same lines.

□

As the states visited in a run in $\mathscr{B}'$, when restricted to Q_i, coincide with those in the corresponding run of $\mathscr{B}$ we have the following claim.

Claim 5.12. *Strategy σ is winning in $\mathbb{G}'$ iff σ'_i is winning in $\mathbb{G}$.*

Conservation of Winner in strategies: Due to the one to one correspondence of plays a control strategy is winning in a control game iff the corresponding distributed strategy is winning in the corresponding ATS game. Claim 5.9 and Claim 5.12 conclude the proof of Lemma 5.7.

Undecidability: The special case of a safety winning condition for six processes is known to be undecidable for control games. By defining the safety winning condition for the ATS game using the same set of unsafe states, we obtain an equivalent undecidable game.

5.2.2 Reduction from ATS Games to control Games

Further, to establish equivalence, we prove the following lemma. In the following lemma, we state that for every ATS game we can construct an equivalent ATS game. This section is dedicated to construct such a game and proving its equivalence.

Lemma 5.13 (ATS $<$ Con)**.** *For any state based winning condition given an ATS game $\mathscr{G}$ over distributed alphabet Σ we can construct a control game $\mathscr{G}^c$ over a distributed alphabet $\Sigma^c = \Sigma^{con} \sqcup \Sigma^{ucon}$ with the same number of processes and comparable number of actions such that there is a winning strategy in $\mathscr{G}$ iff there is a winning strategy in $\mathscr{G}^c$.*

Construction: Let $\mathscr{G} = (A, \text{Win})$ be an ATS game where $A = (S_i, \xrightarrow{a})$ is an asynchronous transition system over distributed alphabet Σ on the set of processes $\mathscr{P}$ and the set of winning plays is Win. We construct an control game $\mathscr{G}^c = (A', \text{Win}^c)$ where $A' = (S'_i, \overset{a}{\curvearrowright}_c)$ is a deterministic asynchronous transition system over distributed alphabet $\Sigma^c = \Sigma^{con} \sqcup \Sigma^{ucon}$ with the same set

of processes as follows. We use loc' to denote the location function in the control game i.e. $i \in \text{loc}'(a)$ if $a \in \Sigma_i^c$

1. **Actions:** The set Σ^{ucon} consists of the original alphabet Σ which allows the environment to schedule and play all actions.

$$\Sigma^{ucon} = \Sigma, \quad \text{loc}'(a) = \text{loc}(a).$$

On the other hand Σ^{con} consists of joint states for each action enabling the system to make its next move.

$$\Sigma^{con} = \{s_a \mid \exists a \in \Sigma : s_a \in S_a\}, \quad \text{loc}'(s_a) = \text{loc}(a).$$

2. **States:** The set of states for process i, denoted as Q_i, is defined as:

$$S_i' = S_i \cup \{(s_a, a) \mid a \in \Sigma_i \wedge s_a \in S_a\}.$$

We introduce the following terms:

- **Pure state:** We call an i-state $s_i \in S_i$, which is a single, well-defined state of process i in $\mathscr{G}$ as a pure process i state in game $\mathscr{G}^c$.
- **Impure state:** A state $(s_a, a) \in S_a \times \{a\}$ for $a \in \Sigma_i$, which includes additional information about the action a is an impure process i-state in game $\mathscr{G}^c$.

3. **Transitions from pure states:** In a pure state only environment actions are allowed. When environment schedules a, the next local state for each process is derived by this choice. Each participating process keeps a copy of the current a-state along with the environments choice of action a. $s_a \overset{a}{\curvearrowright}_c (s_a, a)^{|a|}$ is an a-transition in $\mathscr{G}^c$ if $\exists s_a'$ st $s_a \xrightarrow{a} s_a'$ is an a-transition in $\mathscr{G}$. Here, $(s_a, a)^{|a|}$ is an a-state where each process in $\text{loc}(a)$ has local state (s_a, a). That is the environment can play an action if it is enabled in the current state.

$$s_a \overset{a}{\curvearrowright}_c (s_a, a)^{|a|} \text{ if } \exists s_a' \text{ st } s_a \xrightarrow{a} s_a'$$

4. **Transitions from impure states:** In an impure a-state processes participating in a can locally choose the next a state and it is fired or allowed to be the next choice only if all participating processes agree on some choice and the new local pure states are assigned to the participating processes according to this choice. i.e. $(s_a, a)^{|a|} \overset{s'_a}{\curvearrowright}_c s'_a$ if $s_a \xrightarrow{a} s'_a$. If all processes participating in a choose the next state as s'_a this choice can be proceeded with. Note that in an impure state, all participating processes share the same information due to the last pure action.

$$(s_a, a)^{|a|} \overset{s'_a}{\curvearrowright}_c s'_a \text{ if } s_a \xrightarrow{a} s'_a$$

.

5. **Play correspondence:**

   We define a function AtstoCon_{Ats} : plays in $\mathscr{G} \rightarrow$ plays in $\mathscr{G}^c$ that maps a play in ATS game $\mathscr{G}$ to a unique play in $\mathscr{G}^c$ in this construction from an ATS game to a control game. Formally an empty ATS play mapsto to an empty control play.i.e. $\text{AtstoCon}_{Ats}((\varepsilon, \rho_\varepsilon)) = \varepsilon$. And when play (ta, ρ_{ta}) extends play (t, ρ_t) then, $\text{AtstoCon}_{Ats}(ta, \rho_{ta}) = \text{AtstoCon}_{Ats}(t, \rho_t).a.s_a$ where $s_a = \rho_{ta}(ta)_a$. For example, let $t = abc$, and ρ be a run on it. Then, $u = as_abs_bcs_c$ where $s_a = \rho(a)_a, s_b = \rho(ab)_b, s_c = \rho(abc)_c$.

   We now define AtstoCon_{Con} as the inverse of AtstoCon_{Ats}. This establishes a one-to-one correspondence between plays. By a slight abuse of notation when $\text{AtstoCon}_{Con}(u) = (t, \rho)$ we also use $\text{AtstoCon}_{Con}(u)$ to also denote trace t. Whether we are referring to t or (t, ρ) will be clear from context.

6. **Winning Conditions:**

   The winning set Win^c in the control game is defined as the set of plays corresponding to the winning set Win in the ATS game. i.e.$\text{Win}^c = \{u \mid \text{AtstoCon}_{Con}(u) \in \text{Win}\}$. Note that if the winning condition Win is state based the winning condition Win^c can also be expressed as a state based winning condition.

   Finally, we require that the controller produce a *deadlock-free* control strategy. In ATS games, a play is interpreted as winning or losing only if it is maximal, i.e., it cannot be extended. Therefore, whenever a play can be extended, the control strategy must extend

it.

We want to demonstrate that "there exists a winning strategy in $\mathscr{G}$ if and only if there exists a winning strategy in $\mathscr{G}^c$." We proceed by arguing in both directions:

Natural Strategy in $\mathscr{G}^c$ from $\mathscr{G}$: Given a winning strategy σ in $\mathscr{G}$ from $s_0 \in S_{\mathscr{P}}$, we define the corresponding strategy σ' from s_0 in $\mathscr{G}^c$ as follows:

- If a state s_P is a pure P-state $s_P \in S_P$, all P actions enabled in s_P in $\mathscr{G}$ are enabled in $\mathscr{G}^c$. All enabled actions are environment actions. A control strategy does not restrict the environment's move.
- If a process is in an impure state (s_a, a) after participating in action a, all processes participating in a choose the next pure a state given by σ as the next allowable action and deterministically move to it. For u pointed at a and $i \in \mathrm{loc}(a)$

$$\sigma'_i(u) = \sigma(t)_a : t = \mathrm{AtstoCon}_{Con}(u)$$

Conservation of Winning Condition : Since Win^c is defined such that a play is winning in $\mathscr{G}^c$ if and only if the corresponding play in $\mathscr{G}$ is winning, σ' is a winning strategy in $\mathscr{G}^c$.

The following statements can be proved in exactly the same way as in the construction of an ATS game from a control game. For completeness we present the proofs as well.

Claim 5.14. *If σ is a distributed strategy in $\mathscr{G}$ then σ' is a control strategy in $\mathscr{G}^c$.*

Proof. Due to locality of action, $A(t)_i = A(\partial_i(t))_i$. As $\sigma'_i(u) = \sigma(t)_a : t = \mathrm{AtstoCon}_{Con}(u)$, $\sigma'_i(u) = \sigma'_i(\partial_i(u))$. Therefore, the strategy is distributed.

□

Claim 5.15. *Play (t, σ) conforms to strategy σ in game $\mathscr{G}$ iff $\mathrm{AtstoCon}_{Ats}(t, \sigma)$ conforms strategy σ' in game $\mathscr{G}^c$.*

Proof. For the base case, we know that $\mathrm{AtstoCon}_{Con}(\varepsilon) = (\varepsilon, \rho_\varepsilon)$ and ε conforms σ' and $(\varepsilon, \rho_\varepsilon)$ conforms σ. As this is a pure state all enabled actions are environment actions. Let play

(t,σ) and (ta,σ) conform strategy σ. Let $u = \text{AtstoCon}_{Ats}(t,\sigma)$ conform strategy σ'. Then $u.a.s_a = \text{AtstoCon}_{Ats}(ta,\sigma)$ also conforms strategy σ' because of Item 3 for deterministic action a and Item 4 for non deterministic action s_a.

The converse also follows in the same lines.

□

Since the state sequence in a play restricted to pure states is identical in to the both games, a play in $\mathcal{G}^c$ is winning (i.e., in Win^c) if and only if the corresponding play in $\mathcal{G}$ is winning (i.e., in Win). Thus, the winning party is preserved under the correspondence of plays. As a direct implication of the this, we get the following-

Claim 5.16. *Strategy σ is winning in $\mathcal{G}$ iff σ'_i is winning in $\mathcal{G}^c$.*

Natural Strategy in $\mathcal{G}$ from $\mathcal{G}^c$: Suppose there exists a winning strategy τ in $\mathcal{G}^c$ from $s_0 \in S_{\mathscr{P}}$. We define the corresponding strategy τ' from s_0 in $\mathcal{G}$ inductively as follows. $\tau'(\varepsilon) = s_0$. For a trace ta pointed at a

$$\tau'(ta)_a = s_a \text{where } \forall i \in \text{loc}(a) : s_a \in \tau_i(\text{AtstoCon}_{Ats}(t,\tau').a)$$

Such an s_a always exists as we require a deadlock free strategy in $\mathcal{G}^c$.

Note that given strategy σ in $\mathcal{G}$ let σ' be the derived strategy in game $\mathcal{G}^c$. Then, the strategy σ'' for game $\mathcal{G}$ derived from σ' is equal to σ, i.e. t is in domain of σ iff it is in domain of σ'' and $\sigma(t) = \sigma''(t)$. Thus, the below statements follow.

Conservation of Winning State : Given that Win^c and Win are in one-to-one correspondence, the strategy τ' should be a winning strategy in $\mathcal{G}$ if τ is a winning strategy in $\mathcal{G}^c$.

Claim 5.17. *If τ is a control strategy in $\mathcal{G}^c$ then τ' is a distributed strategy in $\mathcal{G}$.*

Proof. We know that $\tau'(\varepsilon) = s_0$. Now we prove that τ' respects the transitions of A. Say u, uas_a conform strategy τ and $t = \text{AtstoCon}_{Con}(u)$ is in the domain of τ'. We prove that ta is in the

domain of τ' and $\tau'(t) \overset{a}{\curvearrowright}_c \tau'(ta)$.

As $A'(u)_a \overset{a}{\curvearrowright}_c (A'(u)_a, a)^a$ according to Item 3 and $(A'(u)_a, a)^a \overset{s_a}{\curvearrowright}_c s_a$ according to Item 4 are valid transitions only if ta is in the domain of τ' and $A'(u)_a \xrightarrow{a} s_a$ where according to definition of τ', forall $i \in \mathrm{loc}(a) : s_a \in \tau_i(ua)$. Then, $\tau'(t)_a = A'(u)_a$ and $\tau'(ta)_a = A'(uas_a)_a = s_a$. $A'(u)_a \xrightarrow{a} A'(uas_a)_a$.

□

Claim 5.18. *Play u conforms strategy τ_i in game $\mathcal{G}^c$ iff play* $\mathrm{AtstoCon}_{Con}(u)$ *conforms strategy τ' in game $\mathcal{G}$.*

Proof. For the base case, we know that $\mathrm{AtstoCon}_{Con}(\varepsilon) = (\varepsilon, \rho_\varepsilon)$ and ε conforms τ' and $(\varepsilon, \rho_\varepsilon)$ conforms τ. Let play u, uas_a conform to strategy τ. Let $(t, \tau') = \mathrm{AtstoCon}_{Con}(u)$ then t is in the domain of τ' and the play also conform τ'. Then from Item 3 and Item 4 we can see that ta is also in the domain of τ' and (ta, τ') also conforms τ' and is the same as $\mathrm{AtstoCon}_{Con}(uas_a)$.

□

Since the state sequence in a play restricted to pure states is identical to corresponding play in the corresponding game, a direct implication of the above two claims, we get the following-

Claim 5.19. *Strategy τ_i is winning in $\mathcal{G}^c$ iff τ' is winning in $\mathcal{G}$.*

Conservation of Winner in strategies: Thus, if there is a winning strategy in $\mathcal{G}$, there is a corresponding winning strategy in $\mathcal{G}^c$, and vice versa. We conclude that "there is a winning strategy in $\mathcal{G}$ if and only if there is a winning strategy in $\mathcal{G}^c$", validating that the games $\mathcal{G}$ and $\mathcal{G}^c$ are essentially equivalent in terms of winnability.

5.2.3 Implications of the Equivalence

The equivalence between control games and ATS games has significant implications. It extends the known undecidability results from control games to ATS games, revealing the inherent complexity of analyzing distributed systems under these models.

At the same time, this equivalence enables results established for control games to carry over to ATS games, and vice versa, thereby unifying the two frameworks. While carrying out such translations may need further investigation it allows insights from control games to directly enrich our understanding of ATS games, and vice versa, leading to a more cohesive view of distributed synthesis.

This page was intentionally left blank.

Chapter 6

Conclusion and Future work

In this work, we investigated distributed games in the framework of *asynchronous transition system (ATS) games*, focusing in particular on two special cases that naturally arise in this model: two-process ATS games and games with a *central decision maker (CDM)*. These cases are not only easily formulated within the ATS framework, but also capture practically relevant scenarios in distributed synthesis. Thanks to the equivalence we established between ATS games and control games, results obtained in one model can be systematically lifted to the other, thereby unifying the two frameworks. Further exploring the connections between ATS games and other models of distributed synthesis to unify disparate approaches is left for future.

For two-process ATS games, we studied objectives- global safety, local reachability, and global reachability. Our results address two aspects: first, the decidability of whether there exists a winning strategy; and second, the memory structure of such strategies, where we establish both upper and lower bounds. We provide fixpoint-based techniques for computing winning regions, inspired by classical solutions in two player full information games. Extending these results to more general objectives—such as parity conditions—offers a natural and interesting direction for future work.

For CDM games—where all non-deterministic choices are controlled by a single pro-

cess—we established two main results: first, we proved that deciding the existence of winning strategies for global safety and local parity objectives is EXPTIME-complete; and second, we described optimal memory structures for these strategies whenever they exist. Generalizing these results to ω-regular winning conditions is a natural next step. We expect that analogues of the Büchi–Landweber theorem should hold—namely, that whenever a winning strategy exists, there is also a finite one. Establishing this formally will be the subject of future work.

In summary, this work advances the theory of distributed synthesis by introducing a unified framework that captures both classical and asynchronous settings, while addressing decidability and implementability questions in practically meaningful cases. The ATS model, along with our results for two-process and CDM games, opens several directions for future work—including extending to full ω-regular objectives.

References

Adsul, B., Jain, N., 2025. Asynchronous transition system games for two processes and their analysis. In: 11th Indian Conference on Logic and Its Applications (ICLA 2025). Vol. 15402 of Lecture Notes in Computer Science. Springer, pp. 69–83.

Beutner, R., Finkbeiner, B., Hecking-Harbusch, J., 2019. Translating Asynchronous Games for Distributed Synthesis. In: Fokkink, W., van Glabbeek, R. (Eds.), 30th International Conference on Concurrency Theory (CONCUR 2019). Vol. 140 of Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 26:1–26:16.
URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CONCUR.2019.26

Calude, C. S., Jain, S., Khoussainov, B., Li, W., Stephan, F., 2022. Deciding parity games in quasi-polynomial time. SIAM Journal on Computing 51 (2), STOC17–152–STOC17–188.

Diekert, V., Métivier, Y., 1997. Partial commutation and traces. In: Rozenberg, G., Salomaa, A. (Eds.), Handbook of Formal Languages: Volume 3 Beyond Words. Springer, pp. 457–533.

Diekert, V., Rozenberg, G., 1995. The Book of Traces. World Scientific Publishing Co., Inc., USA.

Fijalkow, N., Horn, F., 2012a. The surprizing complexity of generalized reachability games.
URL https://arxiv.org/abs/1010.2420

Fijalkow, N., Horn, F., 2012b. The surprizing complexity of generalized reachability games.
URL https://arxiv.org/abs/1010.2420

Finkbeiner, B., Gieseking, M., Hecking-Harbusch, J., Olderog, E., 2021. Global winning conditions in synthesis of distributed systems with causal memory. CoRR abs/2107.09280.
URL https://arxiv.org/abs/2107.09280

Finkbeiner, B., Gölz, P., 2018. Synthesis in Distributed Environments. In: 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science

(FSTTCS 2017). Vol. 93 of Leibniz International Proceedings in Informatics (LIPIcs). pp. 28:1–28:14.

Finkbeiner, B., Olderog, E., 2017. Petri games: Synthesis of distributed systems with causal memory. Information and Computation 253, 181–203.

Gastin, P., Lerman, B., Zeitoun, M., 2004. Distributed games with causal memory are decidable for series-parallel systems. In: Lodaya, K., Mahajan, M. (Eds.), FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings. Vol. 3328 of Lecture Notes in Computer Science. Springer, pp. 275–286.
URL https://doi.org/10.1007/978-3-540-30538-5_23

Genest, B., Gimbert, H., Muscholl, A., Walukiewicz, I., 2012. Asynchronous games over tree architectures. CoRR abs/1204.0077.
URL http://arxiv.org/abs/1204.0077

Genest, B., Gimbert, H., Muscholl, A., Walukiewicz, I., 2013. Asynchronous games over tree architectures. In: Fomin, F. V., Freivalds, R., Kwiatkowska, M. Z., Peleg, D. (Eds.), Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II. Vol. 7966 of Lecture Notes in Computer Science. Springer, pp. 275–286.
URL https://doi.org/10.1007/978-3-642-39212-2_26

Gimbert, H., 2017. On the control of asynchronous automata.
URL https://arxiv.org/abs/1601.05176v12

Gimbert, H., 2022. Distributed asynchronous games with causal memory are undecidable. Log. Methods Comput. Sci. 18 (3).
URL https://doi.org/10.46298/lmcs-18(3:30)2022

Grädel, E., Thomas, W., Wilke, T. (Eds.), 2002. Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]. Vol. 2500 of Lecture Notes in Computer Science. Springer.
URL https://doi.org/10.1007/3-540-36387-4

Madhusudan, P., Thiagarajan, P. S., Yang, S., 2005. The MSO theory of connectedly communicating processes. In: Ramanujam, R., Sen, S. (Eds.), FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Hyderabad, India, December 15-18, 2005, Proceedings. Vol. 3821 of Lecture Notes in Computer Science. Springer, pp. 201–212.
URL https://doi.org/10.1007/11590156_16

McNaughton, R., 1993. Infinite games played on finite graphs. Annals of Pure and Applied Logic 65 (2), 149–184.

Mukund, M., 2012. Automata on distributed alphabets. In: Modern Applications of Automata Theory. pp. 257–288.
URL http://dx.doi.org/10.1142/9789814271059_0009

Mukund, M., Sohoni, M. A., 1997. Keeping track of the latest gossip in a distributed system. Distributed Comput. 10 (3), 137–148.
URL https://doi.org/10.1007/s004460050031

Muscholl, A., Walukiewicz, I., 2014. Distributed Synthesis for Acyclic Architectures. In: 34th International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014). Vol. 29 of Leibniz International Proceedings in Informatics (LIPIcs). pp. 639–651.

Pnueli, A., Rosner, R., 1990. Distributed reactive systems are hard to synthesize. In: 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II. IEEE Computer Society, pp. 746–757.
URL https://doi.org/10.1109/FSCS.1990.89597

Stockmeyer, L. J., Chandra, A. K., 1979. Provably difficult combinatorial games. SIAM Journal on Computing 8 (2), 151–174.
URL https://doi.org/10.1137/0208013

Thomas, W., 2008. Solution of church's problem: A tutorial. In: Apt, K. R., van Rooij, R. (Eds.), New Perspectives on Games and Interaction. Amsterdam Univ. Press, pp. 211–236.

Zielonka, W., 1987. Notes on finite asynchronous automata. RAIRO-Theoretical Informatics and Applications 21 (2), 99–135.

This page was intentionally left blank.

List of Publications

Published

- Bharat Adsul and Nehul Jain. *Distributed Games with a Central Decision Maker*. In: 45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2025), BITS Pilani, K. K. Birla Goa Campus, India, December 17–19, 2025. Leibniz International Proceedings in Informatics (LIPIcs), vol. 360, pp. 5:1–5:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.FSTTCS.2025.5

- Bharat Adsul and Nehul Jain. *Asynchronous Transition System Games for Two Processes and Their Analysis*. In: Logic and Its Applications — 11th Indian Conference, ICLA 2025, Kolkata, India, February 3–5, 2025, Proceedings. Lecture Notes in Computer Science, vol. 15402, pp. 69–83. Springer, 2025. doi:10.1007/978-3-031-89610-1_5

- Bharat Adsul and Nehul Jain, *Non-deterministic asynchronous automata games and their undecidability*, CoRR, vol. abs/2410.04420, 2024. https://doi.org/10.48550/arXiv.2410.04420