Università degli Studi di Udine
Dipartimento di Matematica e Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis

Dividing and Conquering the Layered Land

Candidate:
Massimo Franceschet

Supervisor:
Prof. Angelo Montanari

February 13, 2002

Contents

“Le par tute casete...” (Edda)

Abstract

Il *mondo piatto* è un mondo in cui il tempo scorre scandito da un unico orologio, in un'unica direzione, orizzontale. Un po' noioso, direte voi. Gli abitanti di questo mondo possono muoversi nel futuro, e, quelli più furbi, anche nel passato. Il futuro non ha fine, il passato ha invece un inizio, uno zero (ma qualcuno non ci crede). Tutto qui.

Il *mondo a strati* è un po' più eccitante. Ci sono molti strati, disposti uno sopra l'altro. Il tempo di ogni strato è scandito da un orologio. La particolarità è questa: ogni tick di un orologio di qualche strato corrisponde ad un certo numero di tick dell'orologio dello strato che sta sotto, ed è una frazione del tick dell'orologio della strato che sta sopra. Il bello del mondo a strati è che i loro abitanti, oltre a spostarsi avanti e indietro nello strato in cui si trovano, possono, sorpresa sorpresa, muoversi verticalmente, cambiare strato, e vivere più lentamente, prendendo le cose con più calma, se vanno verso su, oppure vivere in modo più frenetico, precipitando un po' le cose, se vanno verso giù (si dice che gli abitanti-giovani prediligano gli strati bassi, e, man mano che maturano, salgano verso l'alto). La dimensione dell'universo a strati non la conosce nessuno. Gli abitanti-scienziati fanno tre ipotesi: esiste un numero finito di strati, quindi, non si può salire per sempre, e neppure scendere per sempre. Oppure, esistono infiniti strati. In questo caso, due soluzioni vengono studiate: vi è uno strato in cima, e poi giù, all'infinito, senza mai fermarsi. Oppure, vi è uno strato in basso, e poi su, all'infinito, senza limite.

In the *flat land* time elapses according to a unique clock, in one direction, horizontally. A bit boring, you may say. The inhabitants of this land may move in the future, and, the smartest ones, in the past too. Future has no end, while past has a starting point, a zero (but someone does not believe it). That's all.

The *layered land* is a bit more exiting. The are several layers, one above the other. Each layer has its own clock. The peculiarity is the following: every tick of a clock of some layer corresponds to a certain number of ticks of the clock of the layer below, and is a fraction of the tick of the clock of the layer above. The beauty of the layered land is that people living here may move back and forth on some layer, and, surprise surprise, they may also move vertically, changing the layer, going upward, where life is slower and calmer, or going downward, where things get faster and frantic (they say that young people prefer living on low layers and, when they get older and wiser, they move upward). The dimension of the layered universe is unknown. Science people make three hypothesis: either there exists a finite number of layers, and hence one can move neither arbitrarily upward nor arbitrarily downward, or there are infinitely many layers. In this case, two solutions are studied: either there exists a coarsest layer on the top, and then downward unbounded, or there is a finest layer at the bottom, and then upward unbounded.

1

Introduction

The goal of the thesis is to find expressive, flexible and executable methods to tackle the problem of automatic verification of temporal specifications involving different time granularities. We follow a *divide and conquer* approach, according to which problems are split into sub-problems and these are delegated to the components. When dealing with real-world systems, organizing their descriptive and inferential requirements in a structured way is often the only way to master the complexity of the design, verification, and maintenance tasks. Formulated in the setting of combined logics, the basic issue underlying such an approach is: how can we guarantee that the logical properties of the component logics, such as axiomatic completeness and decidability, are inherited by the combined one? It has a natural analogue in terms of the associated methods and tools: can we reuse methods and tools developed for the component logics, such as deductive engines and model checkers, to obtain methods and tools for the combined one? We will try to answer this and other related questions, focusing on *granular temporal logics*, which are the direct motivation for this work.

The introduction is organized as follows. In Section 1.1 we describe the main representation and reasoning frameworks for time granularity. In Section 1.2 we relate time granularity to a number of different topics, including real-time logics, interval logics, and combined logics. In Section 1.3 we outline the contributions of the thesis.

1.1 Representing and reasoning with time granularity

The ability of providing and relating temporal representations at different 'grain levels' of the same reality is an important research theme in computer science and artificial intelligence. In particular, it is a major requirement for formal specifications, temporal databases, data mining, problem solving, and natural language understanding. As for *logical specifications*, there exists a large class of reactive systems whose components have dynamic behavior regulated by very different time constants (granular reactive systems). A good specification language must enable one to specify and verify the components of a granular reactive system and their interactions in a simple and intuitively clear way [17, 22, 23, 40, 84, 94, 95, 96, 97]. With regard to *temporal databases*, a standard way to incorporate time is to extend a schema to include some time attributes. Each time attribute takes value over some fixed granularity. Users and applications, however, may require the flexibility of viewing the temporal information contained in the corresponding relation in terms of different granularities. In particular, when information is collected from different sources which are not under the same control, differently-grained time-stamps are associated with different data. To guarantee consistency either the data must be converted into a uniform representation that is independent of time granularity or temporal operations must be generalized to cope with data associated with different temporal domains. In both cases, a precise semantics for time granularity is needed [4, 15, 21, 27, 30, 70, 71, 72, 93, 102, 109, 121, 122, 123]. With regard to *data mining*, a huge amount of data is collected every day in the form of event-time sequences. These sequences represent valuable sources of information, not only for what is explicitly registered, but also for deriving implicit information and predicting the future behaviour of the process that we are monitoring. The latter activity requires an analysis of the frequency of certain events, the discovery of their regularity, the identification ofsets of events that are linked by particular temporal relationships. Such frequencies, regularity, and relationships are very often expressed in terms of multiple granularities, and thus analysis and discovery tools must be able to deal with these granularities [1, 5, 7, 29, 88]. With regard to *problem solving*, several problems in scheduling, planning, and diagnosis can be formulated as temporal constraint satisfaction problems, often involving multiple time granularities. In a temporal constraint satisfaction problem, variables are used to represent event occurrences and constraints are used to represent their granular temporal relationships [6, 25, 38, 59, 82, 101, 105, 110]. Finally, shifts in the temporal perspective occur very often in natural language communication, and thus the ability of supporting and relating a variety of temporal models, at different grain sizes, is a relevant feature for the task of *natural language understanding* [10, 47, 53].

Any time granularity can be viewed as the partitioning of a temporal domain in groups of elements, where each group is perceived as an indivisible unit (a granule). The description of a fact can use these granules to provide it with a temporal qualification, at the appropriate abstraction level. However, adding the concept of time granularity to a formalism does not merely mean that one can use different temporal units to represent temporal quantities in a unique flat model, but it involves semantic issues related to the problem of assigning a proper meaning to the association of statements with the different temporal domains of a layered model and of switching from one domain to a coarser/finer one.

The frameworks to represent and reason about time granularity present in the literature can be classified into algebraic frameworks and logical frameworks. In an *algebraic* (or *operational*) framework, a *bottom granularity* is assumed, and a finite set of *calendar operators* are defined to generate new granularities from existing ones. A granularity is hence identified by an algebraic expression. In the algebraic framework, algorithms are provided to perform granule conversions, that is, to convert the granules in one granularity to those in another granularity, and to perform semantic conversion of statements associated to different granularities. The algebraic approach to time granularity has been mostly applied in the fields of databases, data mining, and temporal reasoning.

Algebraic frameworks for time granularities have been proposed by Foster, Leban, and McDonald [47], by Niezette and Stevenne [102] and by Ning, Jajodia, and Wang [103]. Foster, Leban, and McDonald propose the *temporal interval collection formalism*. A collection is a structured set of intervals, where the order of the collection gives a measure of the structure depth: a collection of order 1 is an ordered list of intervals, and a collection of order n, with $n > 1$, is an ordered list of collections of order $n-1$. Each interval denotes a set of contiguous moments of time. To manipulate collections, dicing and slicing operators are used. The former allows one to divide each interval of a collection into another collection, while the latter provide means to select intervals from collections. For instance, the application of the dicing operator `Week` : `during` : `January1998` divides the interval corresponding to `January1998` into the intervals corresponding to the weeks that are fully contained in the month. Moreover, the application of the slicing operator $[1,-1]/$`Week` : `during` : `January1998` selects the first and the last week from those identified by the dicing operator above. Niezette and Stevenne introduce a similar formalism, called the slice formalism.

Finally, Ning, Jajodia, and Wang introduce a *calendar algebra* consisting of a finite set of parametric calendar operations that can be classified into grouping-oriented operations and granule-oriented operations. The former operations group certain granules of a granularity together to form the granules of a new granularity. For instance, a typical group-oriented operation is $\texttt{Group}_n(G)$ that generates a new granularity G' by partitioning the granules of G into groups containing n granules and making each group a granule of the resulting granularity. The granule-oriented operations do not change the granules of a granularity, but rather select which granules should remain in the new granularity. A typical granule-oriented operation is $\texttt{Subset}^n_m(G)$ that generates a new granularity G' by taking all the granules of G between m and n. A comparison between the expressive power of the above described algebraic frameworks can be found in [9].

In the *logical* (or *descriptive*) framework for time granularity, the different granularities and their interconnections are represented by means of mathematical structures, called layered structures. A *layered structure* consists of a possibly infinite set of related differently-grained temporal domains. Such a structure identifies the relevant temporal domains and defines the relations between time points belonging to different domains. Suitable operators make it possible to move horizontally *within* a given temporal domain of the structure (displacement operators), and to move vertically *across* temporal domains of the structure (projection operators). These operators recall the slicing and dicing operators of the collection formalism. Both classical and temporal logics can be interpreted over the layered structure. Logical formulas allow one to specify properties involving different time granularities in a single formula by mixing displacement and projection operators. Algorithms are provided to verify whether a given formula is consistent (satisfiability problem) as well as to check whether a given formula is satisfied in a particular structure (model checking problem). The logical approach to represent time granularity has been mostly applied in the field of formal specification and verification of concurrent systems.

A logical approach to represent and reason about time granularity, based on a many-level view of temporal structures, has been proposed by Montanari in [92], and further investigated by Montanari, Peron, and Policriti in [94, 95, 97]. In the proposed framework, the *flat* temporal structure of standard temporal logics is replaced by a *layered* temporal universe consisting of a possibly infinite set of related differently-grained temporal domains. In [92], a metric and layered temporal logic for time granularity has been proposed. It is provided with temporal operators of displacement and projection, which can be arbitrarily combined, and it is interpreted over layered structures. However, only a sound axiomatic system for the temporal logic is given, and no decidability result is proved.

Layered structures with exactly $n \geq 1$ temporal domains such that each time point can be refined into $k \geq 2$ time points of the immediately finer temporal domain, if any, are called k-refinable n-layered structures (n-LSs for short). They have been investigated in [97], where a classical second-order language, with second-order quantification restricted to monadic predicates, has been interpreted over them. The language includes a total ordering $<$ and k projection functions $\downarrow_0, \ldots, \downarrow_{k-1}$ over the layered temporal universe such that, for every point x, $\downarrow_0(x), \ldots, \downarrow_{k-1}(x)$ are the k elements of the immediately finer temporal domain, if any, into which x is refined. The satisfiability problem for the monadic second-order language over n-LSs has been proved to be decidable by using a reduction to the emptiness problem for Büchi sequence automata. Unfortunately, the decision procedure has a nonelementary complexity.

Layered structures with an infinite number of temporal domains, ω-layered structures, have been studied in [95]. In particular, the authors investigated *k-refinable upward unbounded layered structures* (UULSs), that is, ω-layered structures consisting of a finest temporal domain together with an infinite number of coarser and coarser domains, and *k-refinable downward unbounded layered structures* (DULSs), that is, ω-layered structures consisting of a coarsest domain together with an infinite number of finer and finer domains. A classical monadic second-order language, including a total ordering $<$ and k projection functions $\downarrow_0, \ldots, \downarrow_{k-1}$, has been interpreted over both UULSs and DULSs. The decidability of the monadic second-order theories of UULSs and DULSs has been proved by reducing the satisfiability problem to the emptiness problem for systolic and Rabin tree automata, respectively. In both cases the decision procedure has a nonelementary complexity. Moreover, an expressively complete temporal logic counterpart of the *first-order* theory of UULSs has been proposed in [94].

A comparison of the algebraic and the logical frameworks is not immediate. The main reason is that, as pointed out above, these frameworks have been applied to different application fields calling for different requirements. For instance, in the database context, granule conversion plays a major role because it allows the user to view the temporal information contained in the database in terms of different granularities, while in the context of verification, decision procedures for consistency and model checking are unavoidable to validate the system. However, abstracting away from the application fields of the two frameworks, a comparison is possible. The main advantage of the algebraic framework is its naturalness: by applying user-friendly operations to existing standard granularities like 'days', 'weeks', and 'months', a quite large class of new granularities, like 'business weeks', 'business months', and 'years since 2000', can be easily generated. The major weakness of the algebraic approach is that reasoning methods basically reduce to granule conversions and semantic translations of statements. Scarce care has received the investigation on algorithms to check whether some relation holds between granularities, e.g., G_1 is finer than G_2 or G_1 is equivalent to G_2. Moreover, only a finite number of time granularities can be represented. On the contrary, reasoning methods have been extensively investigated in the logical framework, where both a finite and an infinite number of time granularities can be dealt with. Theorem provers make it possible to verify whether a granular requirement is consistent, while model checkers allow one to check whether a granular property is satisfied in a particular structure. To allow such computational properties, however, some assumptions have to be taken about the interconnections between granularities, e.g., in a layered structure granularities are totally ordered with respect to the finer-than relation.

A recent original approach to represent and reason about a finite number of time granularities has been proposed by Wijsen [124] and refined by Dal Lago and Montanari [26]. Wijsen models infinite periodic granularities as infinite strings over a suitable finite alphabet. The resulting string-based model is then used to formally state and solve problems of granularity equivalence and minimality. Dal Lago and Montanari gives an automata-theoretic counterpart of the string-based model. They use single string automata, that is, finite-state automata accepting a single infinite string, to represent in a compact way time granularities and to give an algorithmic solution to the problems of equivalence and classification of time granularities (the latter problem is strictly related to the granule conversion problem). Our approach in this thesis is close to the logical one proposed by Montanari, Peron and Policriti.

1.2 Related issues

The original motivation of our research was indeed the design of a temporal logic embedding the notion of time granularity, suitable for the specification of complex concurrent systems whose components evolve according to different time units. However, we established an interesting complementary point of view on time granularity: it can be regarded as an expressive setting to investigate the definability of meaningful timing properties over a single time domain. Moreover, layered structures and logics represent an embedding framework for flat real-time structures and logics. Furthermore, there exists a natural link between structures and theories of time granularity and those developed for representing and reasoning about time intervals. Finally, there are significant similarities between the problems we encountered in studying time granularity, and those addressed by current research on combining logics, theories, and structures. In the following, we briefly explain all these connections.

1.2.1 Granular reactive systems

As pointed out above, we were originally motivated by the design of a temporal logic embedding the notion of time granularity suitable for the specification of granular reactive systems. A *reactive system* is a concurrent program that maintains and interaction with the external environment and that ideally runs forever. Temporal logic has been successfully used for modeling and analyzing the behavior of reactive systems (a survey is [36]). It supports semantic model checking, which can be used to check specifications against system behaviors; it also supports pure syntactic deduction, which may be used to verify the consistency of specifications. Finite-state automata, such as Büchi sequence automata and Rabin tree automata (a survey is [114]), have been proved very useful in order to provide clean and asymptotically optimal satisfiability and model checking algorithms for temporal logics [79, 118] as well as to cope with the *state explosion problem* that frighten concurrent system verification [24, 73, 117]. Moreover, automata themselves can be directly used as a specification formalism, provided with a natural graphical interpretation [87].

A *granular reactive systems* is a reactive system whose components have dynamic behaviours regulated by very different time constants. As an example, consider a pondage power station consisting of a reservoir, with filling and emptying times of days or weeks, generator units, possibly changing state in a few seconds, and electronic control devices, evolving in microseconds or even less. A complete specification of the power station must include the description of these components and of their interactions. A natural description of the temporal evolution of the reservoir state will probably use days: "During rainy weeks, the level of the reservoir increases 1 meter a day", while the description of the control devices behaviour may use microseconds: "When an alarm comes from the level sensors, send an acknowledge signal in 50 microseconds". We say that systems of such a type have *different time granularities*. It is somewhat unnatural, and sometimes impossible, to compel the specifier to use a unique time granularity, microseconds in the previous example, to describe the behaviour of all the components. A good language must indeed allow the specifier to easily describe all simple and intuitively clear facts (naturalness of the notation). Hence, a specification language for granular reactive systems must support different time granularities to allow one (i) to maintain the specifications of the dynamics of differently-grained components as separate as possible (modular specifications), (ii) to differentiate the refinement degree of the specifications of different system components (flexible specifications), and (iii) to write complex specifications in an incremental way by refining higher-level predicates associated with a given time granularity in terms of more detailed ones at a finer granularity (incremental specifications).

1.2.2 Definability of meaningful timing properties

Time granularity can be viewed not only as an important feature of a representation language, but also as a formal tool to investigate the definability of meaningful timing properties, such as density and exponential grow/decay, over a *single* time domain [95]. In this respect, the number of layers (single vs. multiple, finite vs. infinite) of the underlying temporal structure, as well as the nature of their interconnections, play a major role: certain timing properties can be expressed using a single layer; others using a finite number of layers; others only exploiting an infinite number of layers. For instance, temporal logics over binary 2-layered structures suffice to deal with conditions like "P holds at all even times of a given temporal domain" that cannot be expressed using flat propositional temporal logics [125]. Moreover, temporal logics over ω-layered structures allow one to express relevant properties of infinite sequences of states over a single temporal domain that cannot be captured by using flat or n-layered temporal logics. For instance, temporal logics over k-refinable UULSs allow one to express conditions like "P holds at all time points k^i, for all natural numbers i, of a given temporal domain", which cannot be expressed by using either propositional or quantified temporal logics over a finite number of layers, while temporal logics over DULSs allow one to constrain a given property to hold true 'densely' over a given time interval (see Section 1.2.4).

1.2.3 On the relationship with real-time logics

Layered structures and logics can be regarded as an embedding framework for flat real-time structures and logics. A *real-time system* is a reactive system with well-defined fixed-time constraints. Processing must be done within the defined constraints or the system fails. Systems that control scientific experiments, industrial control systems, automobile-engine fuel-injection systems, and weapon systems are examples of real-time systems. Examples of quantitative specifications for real-time systems are periodicity, bounded responsiveness, and timing delays. In order to deal with real-time systems, the notion of state has been extended to that of *timed state*, that is, a state with an attribute specifying the corresponding time instant, and real-time logics have been interpreted over timed state sequences (see [2] for a survey).

Montanari et al. showed that the second-order theory of timed state sequences can be *properly* embedded into the second-order theory of binary UULSs as well as into the second-order theory of binary DULSs [96]. The increase in expressive power of the embedding frameworks makes it possible to express and check further timing properties of real-time systems, which cannot be dealt with by the classical theory. For instance, in the theory of timed state sequences, saying that a state s holds true at time i can be meant to be an abstraction of the fact that state s can be arbitrarily placed in the time interval $[i, i+1)$. The stratification of domains in layered structures naturally supports such an interval interpretation (see Section 1.2.4) and gives means for reducing the uncertainty involved in the abstraction process, allowing to express the more refined property saying that state s belongs to the first (respectively, second) half of the time interval $[i, i+1)$. More generally, the embedding of real-time logics into the granularity framework allows one to deal with *temporal indistinguishability* of states (two or more states having associated the same time) and *temporal gaps* between states (a nonempty time interval between the time associated to two contiguous states) in the uniform framework of time granularity. Temporal indistinguishability and temporal gaps can indeed be interpreted as phenomena due to the fact that real-time logics lack the ability to express properties at the right (finer) level of granularity: distinct states, having the same associated time, can always be ordered at the right level of granularity; similarly, time gaps represent intervals in which a state cannot be specified at a finer level of granularity. A finite number of layers is not sufficient to capture timed state sequences: it is not possible to fix a priori any bound on the granularity that a domain must have to allow one to temporally order a given set of states, and thus we need to have an infinite number of temporal domains at our disposal.

1.2.4 On the relationship with interval logics

There exists a natural link between structures and theories of time granularity and those developed for representing and reasoning about time intervals [92]. Differently-grained temporal domains can indeed be interpreted as different ways of partitioning a given discrete/dense time axis into consecutive disjoint intervals. According to this interpretation, every time point can be viewed as a suitable interval over the time axis and projection implements an intervals-subintervals mapping. More precisely, let us define *direct constituents* of a time point x, belonging to a given domain, the time points of the immediately finer domain into which x can be refined (if any) and *indirect constituents* the time points into which the direct constituents of x can be directly or indirectly refined (if any). The mapping of a given time point into its direct or indirect constituents can be viewed as a mapping of a given time interval into (a specific subset of) its subintervals. The existence of such a natural correspondence between interval and granularity structures hints at the possibility of defining a similar connection at the level of the corresponding theories. For instance, according to such a connection, temporal logics over DULSs allow one to constrain a given property to hold true densely over a given time interval, where P densely holds over a time interval w if P holds over w and there exists a direct constituent of w over which P densely holds.

Most interval temporal logics, such as, for instance, Moszkowski's Interval Temporal Logic (ITL) [100], Halpern and Shoham's Modal Logic of Time Intervals (HS) [63], Venema's CDT Logic [119], and Chaochen and Hansen's Neighborhood Logic (NL) [16], have been shown to be undecidable. Decidable fragments of these logics have been obtained by imposing severe restrictions on their expressive power. As an example, Moszkowski [100] proves the decidability of the fragment of Propositional ITL resulting from the introduction of a *locality* constraint. An ITL interval is a finite or infinite sequence of states. The locality property states that each propositional variable is true over an interval if and only if it is true at its first state. This property allows one to collapse all the intervals starting at the same state into a single interval of length zero, that is, the interval consisting of the first state only. By exploiting such a constraint, decidability of Local ITL can be easily proved by embedding it into Quantified Linear Temporal Logic.

We are currently working on the problem of establishing a connection between structures and logics for time granularity and those for time intervals in order to transfer decidability results from the granularity setting to the interval one. We expect that more expressive decidable fragments of interval logics can be obtained as counterparts of decidable theories of time granularity over n-layered and ω-layered structures. Preliminary results can be found in [120]. The authors propose a new interval temporal logic, called Split Logic (SL for short), which is equipped with operators borrowed from HS and CDT, but is interpreted over specific interval structures, called *split-frames*. The distinctive feature of a split-frame is that there is at most one way to chop an interval into two adjacent subintervals, and consequently it does not possess *all* the intervals. They prove the decidability of SL with respect to particular classes of split-frames which can be put in correspondence with the first-order fragments of the monadic theories of time granularity. In particular, *discrete* split-frames with maximal intervals correspond to finitely layered structures, discrete split-frames (with unbounded intervals) can be mapped into upward unbounded layered structures, and *dense* split-frames with maximal intervals can be encoded into downward unbounded layered structures.

1.2.5 The combining logic perspective

There are significant similarities between the problems we addressed in the time granularity setting and those dealt with by current research on logics that model changing contexts and perspectives. The design of these types of logics is emerging as a relevant research topic in the broader area of combination of logics, theories, and structures, at the intersection of logic with artificial intelligence, computer science, and computational linguistics [56]. The reason is that application domains often require rather complex hybrid description and specification languages, while theoretical results and implementable algorithms are at hand only for simple basic components [54].

As for granular reactive systems, their operational behavior can be naturally described as a suitable combination of temporal *evolutions* (sequences of component states) and temporal *refinements* (mapping of a component state into a finite sequence of states belonging to a finer component). According to such a point of view, the model describing the operational behavior of the system and the specification language can be obtained by *combining* simpler models and languages, respectively, and model checking/satisfiability procedures for combined logics can be used.

It turns out from the above discussion that the setting of time granularity is expressive and flexible enough in order to embed and uniformly study many interesting frameworks not directly related to time granularity. We think that this is a good reason to deepen our understanding the framework of time granularity. In the next section, we summarize the main contributions of the thesis.

1.3 Our contributions

The goal of the thesis is to find expressive, flexible and executable methods to tackle the problem of automatic verification of temporal specifications involving different time granularities. We mainly focus on three kind of layered structures, namely, n-layered structures, downward and upward layered structures. The relevance of these structures has been explained in Section 1.2.

In previous work [94, 95, 96, 97], layered structures have been studied according to the following approach. Classical monadic logics are defined over layered structures, and the resulting theories are reduced to theories over *collapsed* structures. In particular, the monadic theory of n-layered structures is reduced to the monadic theory of one successor over infinite sequences (known as $S1S$), that of downward unbounded layered structures is embedded into the monadic theory of k successors over infinite trees (known as SkS), and that of upward unbounded layered structures in translated into a proper extension of the monadic theory of one successor over infinite sequences (known as $S1S^k$). Since $S1S$ can be embedded into Büchi automata over infinite sequences, the satisfiability problem for the monadic theory of

n-layered structures can be effectively reduced to the emptiness problem for Büchi sequence automata, which is known to be decidable. Similarly, the satisfiability problem for the monadic theory of downward unbounded layered structures can be encoded into the emptiness problem for Rabin tree automata (the automata-theoretic counterpart of SkS), and that for the monadic theory of upward unbounded layered structures can be translated into the emptiness problem for systolic tree automata (the automata-theoretic counterpart of $S1S^k$). Monadic logics for time granularity are quite expressive, but, unfortunately, they have few computational appealing: their decision problem is indeed nonelementary. Moreover, the corresponding automata (Büchi sequence automata, Rabin tree automata, and systolic tree automata) are defined over collapsed structures, and do not directly work over layered structures. Hence, they do not represent natural and intuitive tools to express properties of time granularity.

In this thesis, we follow a different approach. We start by studying how to combine temporal logics in such a way that properties of the components are inherited by the combination. We do the same for automata. Then, we reinterpret layered structures as combined structures. This intuition reveals to be the keystone of our endeavor. Indeed, it allows us to define combined temporal logics and combined automata over layered structures, and to study their expressive power and computational properties by taking advantage of the transfer theorems for combined logics and combined automata. The outcome is appealing: the resulting combined temporal logics and automata directly work over layered structures. Moreover, they are expressively equivalent to monadic languages, and are elementarily decidable. The reader may be skeptical about the latter claim: how can an elementarily decidable logic be expressively equivalent to a nonelementarily decidable one? The elucidation of this mystery lies somewhere in the rest of this thesis.

In the following, we briefly sketch the contents of the thesis.

Chapter 2: we introduce structures, logics, and automata that we will use in the rest of the thesis. In particular, we formally define layered structures. We summarize well-known results about expressiveness and decidability of classical monadic logics, temporal logics, and automata.

Chapter 3: we describe the combining approach to temporal logics and we propose a similar approach for automata. We introduce three well-known modes for combining temporal logics: temporalization, independent combination, and join. We study the model checking problem for combined temporal logics and we propose an automata-theoretic counterpart of temporalized logics. This chapter is based on [50, 51].

Chapter 4: we define and study temporal logics and automata over n-layered structures and ω-layered structures. Taking advantage of the combining method introduced in Chapter 3, we define expressively complete and elementarily decidable temporal logic and automata counterparts of the monadic theories of layered structures. Finally, we apply the combining approach to model, specify, and verify granular reactive systems. This chapter is based on [48, 49].

Chapter 5: we try to extend the picture with new meaningful predicates while preserving decidability. We systematically explore several possibilities, and give a number of positive and negative results by reduction to/from a wide spectrum of decidable/undecidable problems. Interestingly, we find out that the monadic second-order language for time granularity in the signature with a total ordering $<$ and k projection functions $\downarrow_0, \ldots, \downarrow_{k-1}$ has a valid antagonist, which is still decidable and can express different properties. Finally, we propose a framework to represent and reason about time granularity that reconciles the algebraic and logical approaches. This chapter is based on [52].

2

Structures, logics and automata

In this chapter we introduce structures, logics, and automata that we will use in the rest of this thesis. In Section 2.1 we define sequences, trees, and layered structures. In Section 2.2 we introduce classical monadic logics and interpret them over previously defined structures. In Section 2.3 we introduce finite-state automata over sequences and trees, and compare their expressive power with that of the monadic theories of Section 2.2. Finally, in Section 2.4 we introduce temporal logics over sequences and trees, and link them with the monadic theories of Section 2.2. In particular, we define $\mathrm{CTL}^*_\mathrm{k}$, an extension of the popular Computational Tree Logic CTL* with k directed successors $\mathbf{X_0}, \ldots, \mathbf{X_{k-1}}$, and we study the complexity of its satisfiability and model checking problems.

2.1 Structures

In this section we introduce the relational structures that we will use in the rest of the thesis. We begin with some preliminary definitions. We denote by $\mathbb{N}$ the set $\{0, 1, \ldots\}$ of natural numbers and by $\mathbb{N}_+$ the set $\{1, 2, \ldots\}$ of positive natural numbers. An initial segment I of $\mathbb{N}$ is a set $\{0, 1, \ldots, n\}$, for some $n \in \mathbb{N}$. We will make use of O-notation and Θ-notation. Recall that $n = O(m)$ means that $n \leq c \cdot m$, for some constant $c > 0$, and $n = \Theta(m)$ means that $c_1 \cdot m \leq n \leq c_2 \cdot m$, for some constants $c_2 \geq c_1 > 0$. Let A, B be two sets. The *concatenation* of A and B, denoted by AB, is the set $\{xy \mid x \in A,\ y \in B\}$. For every $n \in \mathbb{N}$, we recursively define the set A^n of *strings* of length n over A as follows: $A^0 = \{\epsilon\}$, $A^{n+1} = AA^n$. The *Kleene closure* of A is the set $A^* = \bigcup_{n \in \mathbb{N}} A^n$ of strings of arbitrary finite length over A. We denote by $|x|$ the length of the string $x \in A^*$ defined as follows: $|\epsilon| = 0$, and $|xa| = |x| + 1$, for $x \in A^*$ and $a \in A$. Moreover, let $A^+ = A^* \setminus \{\epsilon\}$ and A^ω be the set of infinite strings (or ω-strings) over A. Given a string α, we denote by $\alpha(i)$ its i-th element and by $\alpha(i, j)$ the segment $\alpha(i) \ldots \alpha(j)$, for $i \leq j$. Let $x, y \in A^*$. We say that x is a *prefix* of y, denoted by $x <_{pre} y$, if $xw = y$ for some $w \in A^+$. Note that the prefix relation $<_{pre}$ is a partial ordering over A^*. Let $<$ be a total ordering over A. For every $x, y \in A^*$, we say that x *lexicographically precedes* y with respect to $<$, denoted by $x <_{lex} y$, if either $x <_{pre} y$ or there exist $z \in A^*$ and $a, b \in A$ such that $za \leq_{pre} x$, $zb \leq_{pre} y$ and $a < b$. Note that the lexicographical relation $<_{lex}$ is a total ordering over A^*.

We now introduce three binary predicates over the natural numbers that we are going to use in this thesis. Let $k \geq 2$. The binary predicate $\mathtt{flip}_k$ is as follows. Given two natural numbers x, y, $\texttt{flip}_k(x, y)$, also denoted by $\texttt{flip}_k(x) = y$, if $y = x - x'$, where x' is the least power of k with non-null coefficient in the k-ary representation of x. Formally, $\texttt{flip}_k(x) = y$ if $x = a_n \cdot k^n + a_{n-1} \cdot k^{n-1} + \ldots + a_m \cdot k^m$, $0 \leq a_i \leq k-1$, $a_m \neq 0$, and $y = a_n \cdot k^n + a_{n-1} \cdot k^{n-1} + \ldots + (a_m - 1) \cdot k^m$. For instance, $\texttt{flip}_2(18, 16)$, since $18 = 2^4 + 2^1$, $a_m = 1$, $m = 1$, and $16 = 2^4 + 02^1$. Moreover, $\texttt{flip}_2(16, 0)$, since $16 = 2^4$, $a_m = 1$, $m = 4$, and $0 = 0 \cdot 2^4$. Finally, there is no y such that $\texttt{flip}_2(0, y)$. The predicate $\texttt{adj}$ is defined as follows: $\texttt{adj}(x, y)$, also denoted by $\texttt{adj}(x) = y$, if $x = 2^{k_n} + 2^{k_{n-1}} + \ldots + 2^{k_0}$, with $k_n > k_{n-1} > \ldots > k_0 > 0$, and $y = x + 2^{k_0} + 2^{k_0 - 1}$. For instance, $\texttt{adj}(12, 18)$, since if $12 = 2^3 + 2^2$, $k_0 = 2$ and $18 = 12 + 2^2 + 2^1$. Moreover, there is no y such that $\texttt{adj}(13, y)$, since $13 = 2^3 + 2^2 + 2^0$ and $k_0 = 0$. Finally, the predicate $2\times$ is such that $2 \times (x, y)$ if $y = 2x$.

Let $\mathcal{P} = \{P, Q, \ldots\}$ be a *finite* set of monadic predicate symbols. A *finite sequence* is a relational structure $s = \langle I, < \rangle$, where I is an initial segment of $\mathbb{N}$ and $<$ is the usual ordering over natural numbers. A $\mathcal{P}$-labeled finite sequence is a relational structure $s = \langle I, <, (\overline{P})_{P \in \mathcal{P}} \rangle$, where I and $<$ are as above and, for every $P \in \mathcal{P}$, $\overline{P} \subseteq I$ is the set of elements labeled with the symbol P. Note that a single point may be labeled with more than one letter. For the sake of simplicity, in the following we will identify the symbol P with its interpretation $\overline{P}$, and, consequently, we will write $\langle I, <, (P)_{P \in \mathcal{P}} \rangle$ instead of $\langle I, <, (\overline{P})_{P \in \mathcal{P}} \rangle$. An *infinite sequence* (or ω-sequence) is a relational structure $s = \langle \mathbb{N}, < \rangle$ and a $\mathcal{P}$-labeled infinite sequence is an infinite sequence expanded with monadic predicates P, for $P \in \mathcal{P}$.

We now define finite and infinite trees. Let $k \geq 2$ and T_k be the set $\{0, \ldots, k-1\}^*$. A set $D \subseteq T_k$ is a k-ary *tree domain* if:

1. D is *prefix closed*, that is, $x \in D$ and $y <_{pre} x$ implies $y \in D$, for every $x, y \in T_k$;
2. for every $x \in T_k$, either $xi \in D$ for every $0 \leq i \leq k-1$ or $xi \notin D$ for every $0 \leq i \leq k-1$.

Note that the whole T_k is a tree domain. A k-ary *finite tree* is a relational structure $t = \langle D, (\downarrow_i)_{i=0}^{k-1}, <_{pre} \rangle$, where D is a k-ary finite tree domain, $\downarrow_i$ is the i-th *successor relation* over D such that $\downarrow_i(x, y)$, also denoted by $\downarrow_i(x) = y$, if $y = xi$, for every $0 \leq i \leq k-1$, and $<_{pre}$ is the prefix ordering over D defined as above. We call *nodes* the elements of D. If $\downarrow_i(x) = y$, then y is said the i-th *son* of x. The lexicographical ordering $<_{lex}$ over D is defined with respect to the natural ordering $<$ over $\{0, \ldots, k-1\}$ such that $0 < 1 < \ldots < k-1$. The *root* of t is the node ϵ. A *leaf* of t is an element $x \in D$ devoid of sons. A node which is not a leaf is called an *internal node*. The *depth* of a node $x \in D$ is the length of the (unique) path from the root ϵ to x. The *height* of t is the highest depth of a node in D. A set $X \subseteq D$ is said *downward closed* if, for every $x \in X$, either x is a leaf, or x has a son $y \in X$. A *full path* π in t is a subset of D whose nodes can be written as a finite sequence $x_0, \ldots, x_n$ such that, for every $0 \leq i \leq n$, $|x_i| = i$ and, for every $0 \leq i < n$, there exists $0 \leq j \leq k-1$ with $x_{i+1} = \downarrow_j(x_i)$. We will denote by $\pi(i)$ the i-th element x_i of the path π. A *path* is a downward closed subset of a full path. A *chain* is a subset of a path. The tree t is *complete* if every full path in t has the same length. A $\mathcal{P}$-labeled finite tree is a relational structure $t = \langle D, (\downarrow_i)_{i=0}^{k-1}, <_{pre}, (P)_{P \in \mathcal{P}} \rangle$, where D, $\downarrow_i$, and $<_{pre}$ are as above and, for every $P \in \mathcal{P}$, $P \subseteq D$ is the set of nodes labeled with the symbol P.

As for infinite trees, we are interested only in *complete* k-ary infinite trees over the tree domain T_k. A (complete) *infinite k-ary tree* is a relational structure $t = \langle T_k, (\downarrow_i)_{i=0}^{k-1}, <_{pre} \rangle$. A *full path* π in t is a subset of T_k whose nodes can be written as an infinite sequence $x_0, x_1, \ldots$ such that, for every $i \geq 0$, $|x_i| = i$ and there exists $0 \leq j \leq k-1$ with $x_{i+1} = \downarrow_j(x_i)$. A *path* is a downward closed subset of a full path. A *chain* is a subset of a path. A $\mathcal{P}$-labeled infinite k-ary tree is an infinite k-ary tree expanded with monadic predicates P, for $P \in \mathcal{P}$.

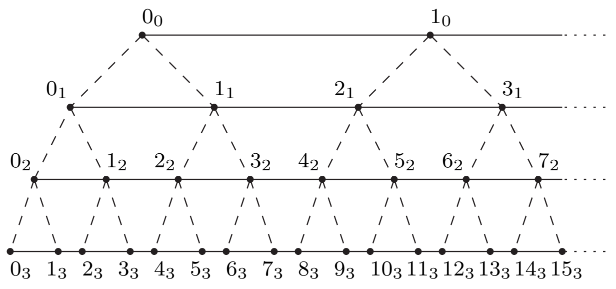

Figure 2.1: The 2-refinable 2-layered structure.

Finally, we define the most relevant structures in the context of this thesis, that is, layered structures. We begin by defining layered structures with a finite and fixed number of layers. Let $n \geq 1$ and $k \geq 2$. For every $i \geq 0$, let $T^i = \{j_i \mid j \geq 0\}$. Let $\mathcal{U}_n = \bigcup_{0 \leq i < n} T^i$ be the *n-layered temporal universe.* A k-refinable *n-layered structure* (n-LS) is a relational structure $\langle \mathcal{U}_n, (\downarrow_i)_{i=0}^{k-1}, < \rangle$, that intuitively represents an infinite sequence of complete k-ary trees, each one rooted at a point of T^0 and of height $n-1$ (see Figure 2.1). The sets $\{T^i\}_{0 \leq i < n}$ are the layers of the trees, $\downarrow_i$, with $i = 0, \ldots, k-1$, is a *projection relation* such that $\downarrow_i (x, y)$, also denoted by $\downarrow_i (x) = y$, if y is the i-th son of x, and $<$ is a total ordering of $\mathcal{U}_n$ given by the *preorder* (root-left-right) visit of the nodes (for elements belonging to the same tree) and by the total linear ordering of trees (for elements belonging to different trees). Formally, for $a_b, c_d \in \mathcal{U}_n$, $\downarrow_i (a_b) = c_d$ if $b < n-1$, $d = b+1$ and $c = a \cdot k + i$. Moreover, the ordering $<$ is axiomatically defined as follows. Let $\downarrow^0 (x) = \{x\}$ and, for every $i \geq 1$, $\downarrow^i (x) = \{\downarrow_j(y) \mid y \in \downarrow^{i-1} (x),\ 0 \leq j \leq k-1\}$.

1. if $x = a_0$, $y = b_0$, and $a < b$ over natural numbers, then $x < y$;
2. $x < \downarrow_j(y)$, for every $0 \leq j \leq k-1$;
3. $\downarrow_j(x) < \downarrow_{j+1}(x)$, for every $0 \leq j \leq k-2$;
4. if $x < y$ and $y \notin \bigcup_{i \geq 0} \downarrow^i (x)$, then $\downarrow_{k-1}(x) < y$;
5. if $x < z$ and $z < y$, then $x < y$.

A *full path* over an n-LS is a subset of $\mathcal{U}_n$ whose elements can be written as a sequence $x_0, x_1, \ldots, x_{n-1}$, such that, for every $i = 0, \ldots, n-1$, x_i belongs to the i-th domain T^i and, for every $i = 0, \ldots, n-2$, there exists $0 \leq j < k$ with $x_{i+1} = \downarrow_j(x_i)$. A *path* is a downward closed subset of a full path. A *chain* is any subset of a path. A $\mathcal{P}$-labeled k-refinable n-layered structure is a relational structure $\langle \mathcal{U}_n, (\downarrow_i)_{i=0}^{k-1}, <, (P)_{P \in \mathcal{P}} \rangle$, where $\downarrow_i$ and $<$ are defined as above and, for every $P \in \mathcal{P}$, $P \subseteq \mathcal{U}_n$ is the set of points in $\mathcal{U}_n$ labeled with symbol P.

We now introduce ω-layered structures, that is, layered structures with an infinite number of layers. Let $\mathcal{U} = \bigcup_{i \geq 0} T^i$ be the *ω-layered temporal universe.* A k-refinable *downward unbounded layered structure* (DULS) is a relational structure $\langle \mathcal{U}, (\downarrow_i)_{i=0}^{k-1}, < \rangle$, that intuitively represents an infinite sequence of complete infinite k-ary trees, each one rooted at a point of T^0 (Figure 2.2). The sets $\{T^i\}_{i \geq 0}$ are the layers of the trees, $\downarrow_i$, with $i = 0, \ldots, k-1$, is a *projection relation* such that $\downarrow_i (x, y)$, also denoted by $\downarrow_i (x) = y$, if y is the i-th son of x, and $<$ is the total ordering of $\mathcal{U}$ given by the *preorder* (root-left-right) visit of the nodes (for elements belonging to the same tree) and by the total linear ordering of trees (for elements belonging to different trees). Formally, for $a_b, c_d \in \mathcal{U}$, $\downarrow_i (a_b) = c_d$ if and only if $d = b+1$ and $c = a \cdot k + i$, and the ordering $<$ is defined as follows:

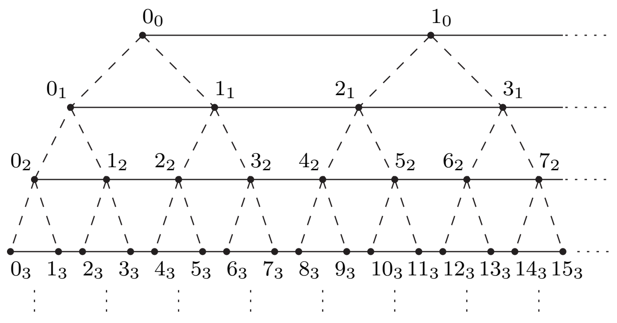

Figure 2.2: A 2-refinable downward unbounded layered structure.

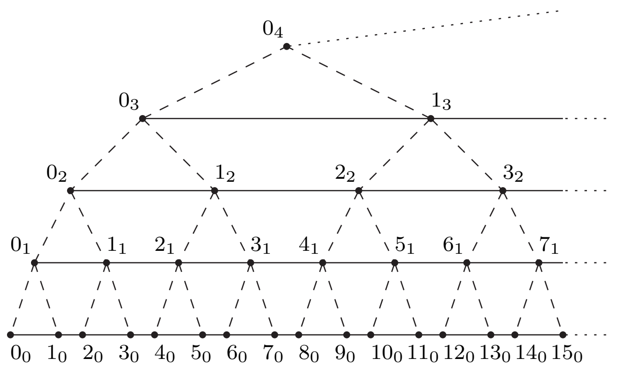

Figure 2.3: A 2-refinable upward unbounded layered structure.

1. if $x = a_0$, $y = b_0$, and $a < b$ over natural numbers, then $x < y$;
2. $x < \downarrow_j(y)$, for every $0 \leq j \leq k-1$;
3. $\downarrow_j(x) < \downarrow_{j+1}(x)$, for every $0 \leq j \leq k-2$;
4. if $x < y$ and $y \notin \bigcup_{i \geq 0} \downarrow^i (x)$, then $\downarrow_{k-1}(x) < y$;
5. if $x < z$ and $z < y$, then $x < y$.

Note that the orderings $<$ over DULSs and $<$ over n-LSs coincide. A *full path* over a DULS is a subset of the domain whose elements can be written as an infinite sequence $x_0, x_1, \ldots$ such that, for every $i \geq 0$, x_i belongs to the i-th domain T^i and there exists $0 \leq j < k$ such that $x_{i+1} = \downarrow_j(x_i)$. A *path* is a downward closed subset of a full path. A *chain* is any subset of a path. A $\mathcal{P}$-labeled k-refinable DULS is defined by augmenting a DULS with a set $P \subseteq \mathcal{U}$ for any $P \in \mathcal{P}$ (the elements of the structure labeled by P).

A k-refinable *upward unbounded layered structure* (UULS) is a relational structure $\langle \mathcal{U}, \langle \downarrow_i \rangle_{i=0}^{k-1}, < \rangle$, that intuitively represents a complete infinite k-ary tree generated from the leaves (Figure 2.3). The sets $\{T^i\}_{i \geq 0}$ are the layers of the tree, $\downarrow_i$, with $i = 0, \ldots, k-1$, is a *projection relation* such that $\downarrow_i (x, y)$, also denoted by $\downarrow_i (x) = y$, if y is the i-th son of x, and $<$ is the total ordering of $\mathcal{U}$ given by the *inorder* (left-root-right) visit of the treelike structure. Formally, for every $a_b, c_d \in \mathcal{U}$, $\downarrow_i (a_b) = c_d$ if and only if $b > 0$, $d = b - 1$ and $c = a \cdot k + i$, and the ordering $<$ is defined as follows:

1. if $x = a_0$, $y = b_0$ and $a < b$ over natural numbers, then $x < y$;

2. $\downarrow_j(x) < \downarrow_{j+1}(x)$, for every $0 \leq j \leq k-2$;
3. if $x < y$ and $y \notin \bigcup_{i \geq 0} \downarrow^i (x)$, then $\downarrow_{k-1}(x) < y$;
4. if $x < y$ and $x \notin \bigcup_{i \geq 0} \downarrow^i (y)$, then $x < \downarrow_0(y)$;
5. if $x < z$ and $z < y$, then $x < y$.

A *full path* over an UULS is a subset of the domain whose elements can be written as an infinite sequence $x_0, x_1, \ldots$ such that, for every $i \geq 0$, x_i belongs to the i-th domain T^i and there exists $0 \leq j < k$ such that $x_i = \downarrow_j(x_{i+1})$. A *path* is a downward closed subset of a full path. A *chain* is any subset of a path. Notice that every pair of full paths over an UULS may differ on a finite prefix only. A $\mathcal{P}$-labeled k-refinable UULS is defined by augmenting an UULS with a set $P \subseteq \mathcal{U}$ for any $P \in \mathcal{P}$ (the elements of the structure labeled by P).

Remark 2.1.1 We have introduced $\mathcal{P}$-labeled relational structures $\langle W, \tau, (P)_{P \in \mathcal{P}} \rangle$, where W is a set, τ is a vocabulary, and, for every $P \in \mathcal{P}$, $P \subseteq W$ is a monadic predicate. In Section 2.2, we will define classical monadic logics and we will interpret them over $\mathcal{P}$-labeled relational structures. Moreover, in Section 2.4, we will introduce temporal logics and we will interpret them over $\mathcal{P}$-labeled *Kripke structures* (W, τ, V), where $V : W \rightarrow 2^{\mathcal{P}}$. To link classical monadic logics and temporal logics, we need to link $\mathcal{P}$-labeled relational and Kripke structures. A $\mathcal{P}$-labeled relational structure $\langle W, \tau, (P)_{P \in \mathcal{P}} \rangle$ corresponds to a $\mathcal{P}$-labeled *Kripke structure* (W, τ, V), where $V : W \rightarrow 2^{\mathcal{P}}$ is such that $P \in V(i)$ iff $i \in P$. Similarly, a $\mathcal{P}$-labeled Kripke structure (W, τ, V) corresponds to a $\mathcal{P}$-labeled relational structure $\langle W, \tau, (P)_{P \in \mathcal{P}} \rangle$.

Furthermore, in Section 2.3, we will introduce finite-state automata accepting Σ-labeled Kripke structures (W, τ, V), where $\Sigma = \{a, b, \ldots\}$ is a finite set of symbols, and $V : W \rightarrow \Sigma$. To link logics and automata, we need to link $\mathcal{P}$-labeled Kripke structures and Σ-labeled Kripke structures. A $\mathcal{P}$-labeled Kripke structure (W, τ, V) can be represented as a Σ-labeled Kripke structure (W, τ, V'), where $\Sigma = 2^{\mathcal{P}}$ and $V' : W \rightarrow \Sigma$ is such that $V'(w) = X$ iff $V(w) = X$, for every $w \in W$. Moreover, a Σ-labeled Kripke structure (W, τ, V) can be represented as a $\mathcal{P}_\Sigma$-labeled Kripke structure (W, τ, V'), where $\mathcal{P}_\Sigma = \{P_a \mid a \in \Sigma\}$ and $V' : W \rightarrow 2^{\mathcal{P}_\Sigma}$ is such that $V'(w) = \{P_a\}$ iff $V(w) = a$, for every $w \in W$.

2.2 Monadic theories

In this section we introduce classical monadic logics and interpret them over the relational structures introduced in Section 2.1.

Definition 2.2.1 (*Monadic second-order logic*)

Let $\tau = c_1, \ldots, c_r, u_1, \ldots, u_s, b_1, \ldots, b_t$ be a finite alphabet of symbols, where $c_1, \ldots, c_r$ (resp. $u_1, \ldots, u_s$, $b_1, \ldots, b_t$) are constant symbols (resp. unary relational symbols, binary relational symbols). The second-order language with equality MSO$[\tau \cup \mathcal{P}]$, *simply denoted by* MSO$_{\mathcal{P}}[\tau]$, *is built up as follows:*

1. atomic formulas *are of the forms $x = y$, $x = c_i$, with $1 \leq i \leq r$, $u_i(x)$, with $1 \leq i \leq s$, $b_i(x, y)$, with $1 \leq i \leq t$, $x \in X$, $x \in P$, where x, y are individual variables, X is a set variable, and $P \in \mathcal{P}$;*

2. formulas *are built up from atomic formulas by means of the Boolean connectives $\neg$ and $\wedge$, and the quantifier $\exists$ ranging over both individual and set variables.*

The additional Boolean connectives $\vee$, $\rightarrow$, and $\leftrightarrow$ and the universal quantifier $\forall$ are defined as usual. We will assume that $\neg$ has priority over $\wedge$, and $\wedge$ has priority over $\rightarrow$ and $\leftrightarrow$. Moreover, $\rightarrow$ and $\leftrightarrow$ have priority over $\vee$. For instance $\neg A \wedge B$ reads $(\neg A) \wedge B$, $A \wedge B \rightarrow C$ reads $(A \wedge B) \rightarrow C$, and $A \rightarrow B \vee C$ reads $(A \rightarrow B) \vee C$.

We will take into consideration the first-order fragment $\text{MFO}_{\mathcal{P}}[\tau]$ of $\text{MSO}_{\mathcal{P}}[\tau]$ as well as its path (resp. chain) fragment $\text{MPL}_{\mathcal{P}}[\tau]$ (resp. $\text{MCL}_{\mathcal{P}}[\tau]$), which is obtained by interpreting second-order variables over paths (resp. chains). In particular, we will consider the following monadic languages interpreted over the following structures in the standard way:

1. $\text{MFO}_{\mathcal{P}}[<]$ and $\text{MSO}_{\mathcal{P}}[<]$ over finite and infinite sequences;
2. $\text{MFO}_{\mathcal{P}}[<, \texttt{flip}_k]$ and $\text{MSO}_{\mathcal{P}}[<, \texttt{flip}_k]$ over infinite sequences;
3. $\text{MSO}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ and its first-order, path, and chain fragments over finite and infinite trees;
4. $\text{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ and its first-order, path, and chain fragments over n-LSs, DULSs, and UULSs.

A model of the formula φ is a structure in which φ is true. We denote by $\mathcal{M}(\varphi)$ the set of models of the formula φ. We say that $\text{MSO}_{\mathcal{P}}[\tau_1]$ is *embeddable* into $\text{MSO}_{\mathcal{P}}[\tau_2]$, denoted by $\text{MSO}_{\mathcal{P}}[\tau_1] \rightarrow \text{MSO}_{\mathcal{P}}[\tau_2]$, if there is an *effective* translation τ of $\text{MSO}_{\mathcal{P}}[\tau_1]$-formulas into $\text{MSO}_{\mathcal{P}}[\tau_2]$-formulas such that, for every formula $\varphi \in \text{MSO}_{\mathcal{P}}[\tau_1]$, $\mathcal{M}(\varphi) = \mathcal{M}(\tau(\varphi))$. $\text{MSO}_{\mathcal{P}}[\tau_1]$ is *expressively equivalent* to $\text{MSO}_{\mathcal{P}}[\tau_2]$, written $\text{MSO}_{\mathcal{P}}[\tau_1] \rightleftarrows \text{MSO}_{\mathcal{P}}[\tau_2]$, if $\text{MSO}_{\mathcal{P}}[\tau_1] \rightarrow \text{MSO}_{\mathcal{P}}[\tau_2]$ and $\text{MSO}_{\mathcal{P}}[\tau_2] \rightarrow \text{MSO}_{\mathcal{P}}[\tau_1]$. Clearly, if $\text{MSO}_{\mathcal{P}}[\tau_1] \rightarrow \text{MSO}_{\mathcal{P}}[\tau_2]$ and $\text{MSO}_{\mathcal{P}}[\tau_2]$ is decidable, then $\text{MSO}_{\mathcal{P}}[\tau_1]$ is decidable too. Let β be a relational symbol. We say that β is *definable* in $\text{MSO}_{\mathcal{P}}[\tau]$ if $\text{MSO}_{\mathcal{P}}[\tau \cup \{\beta\}] \rightarrow \text{MSO}_{\mathcal{P}}[\tau]$. If the addition of β to a decidable theory $\text{MSO}_{\mathcal{P}}[\tau]$ makes the resulting theory $\text{MSO}_{\mathcal{P}}[\tau \cup \{\beta\}]$ undecidable, we can conclude that β is not definable in $\text{MSO}_{\mathcal{P}}[\tau]$. The opposite does not hold in general: the predicate β may be not definable in $\text{MSO}_{\mathcal{P}}[\tau]$, but extending $\text{MSO}_{\mathcal{P}}[\tau]$ with β may preserve decidability. In such a case, we obviously cannot reduce the decidability of $\text{MSO}_{\mathcal{P}}[\tau \cup \{\beta\}]$-formulas to that of $\text{MSO}_{\mathcal{P}}[\tau]$-formulas. All these definitions and results transfer to chain, path, and first-order fragments of $\text{MSO}_{\mathcal{P}}[\tau]$.

It is easy to see that

$$\text{MFO}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}] \rightarrow \text{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}] \rightarrow \text{MCL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}] \rightarrow \text{MSO}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$$

over trees. Indeed, the predicate 'X is a path' can be encoded in monadic chain logic, and the predicate 'X is a chain' can be expressed in monadic second-order logic. Moreover, monadic path logic over paths has the same expressive power than monadic path logic over full paths [114]. Similarly for layered structures.

$\text{MSO}_{\mathcal{P}}[<]$ over infinite sequences is better known as $S1S$. The decidability of $\text{MSO}_{\mathcal{P}}[<]$ over finite sequences has been proved in [12, 32], and that of $S1S$ has been shown in [13].

Theorem 2.2.2 $\text{MSO}_{\mathcal{P}}[<]$ *over finite (resp. infinite) sequences is decidable.*

$\text{MSO}_{\mathcal{P}}[<, \texttt{flip}_k]$ over infinite sequences is known as $S1S^k$, and it properly extends $S1S$ [98]. Moreover, the unary predicate $\texttt{pow}_k$ such that $\texttt{pow}_k(x)$ if x is a power of k can be easily expressed as $\texttt{flip}_k(x) = 0$. Hence, $\text{MSO}_{\mathcal{P}}[<, \texttt{flip}_k]$ is at least as expressive as the well-known extension of $\text{MSO}_{\mathcal{P}}[<]$ with the predicate $\texttt{pow}_k$ [33]. The decidability of $S1S^k$ has been proved in [98].

Theorem 2.2.3 $\mathrm{MSO}_{\mathcal{P}}[<, \mathtt{flip}_k]$ *over infinite sequences is decidable.*

$\mathrm{MSO}_{\mathcal{P}}[<, \mathtt{adj}]$ properly extends $\mathrm{MSO}_{\mathcal{P}}[<, \mathtt{flip}_2]$, but, unfortunately, it is undecidable [99].

Theorem 2.2.4 $\mathrm{MSO}_{\mathcal{P}}[<, \mathtt{adj}]$ *over infinite sequences is undecidable.*

$\mathrm{MSO}_{\mathcal{P}}[<, 2\times]$ is at least as expressive as $\mathrm{MSO}_{\mathcal{P}}[<, \mathtt{adj}]$ and hence its decision problem is undecidable [99].

Theorem 2.2.5 $\mathrm{MSO}_{\mathcal{P}}[<, 2\times]$ *over infinite sequences is undecidable.*

Moving to trees, $\mathrm{MSO}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ over infinite k-ary trees is well-known as SkS. The decidability of $\mathrm{MSO}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ over finite trees has been shown in [28, 113]. The decidability of $S2S$ has been proved in [107]. This result can be easily generalized to SkS over k-ary trees (and even to $S\omega S$ over countably branching trees) [114].

Theorem 2.2.6 $\mathrm{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ *over finite (resp. infinite) trees is decidable.*

As for layered structures, the decidability of the second-order theory of n-LSs has been proved in [97] by reducing it to $S1S$.

Theorem 2.2.7 $\mathrm{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ *over n-LSs is decidable.*

The decidability of the second-order theory of DULSs has been proved in [95] by embedding it into SkS.

Theorem 2.2.8 $\mathrm{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ *over DULSs is decidable.*

Finally, the decidability of the second-order theory of UULSs has been proved in [95] by exploiting a reduction to $S1S^k$.

Theorem 2.2.9 $\mathrm{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ *over UULSs is decidable.*

It is worth pointing out that the decision problem for every decidable monadic theory considered above has *nonelementary* complexity. Recall that a function $f : \mathbb{N} \to \mathbb{N}$ is *elementary* if there is $k \in \mathbb{N}$ such that, for every $n \in \mathbb{N}$, $f(n) \leq \kappa(k, n)$, where $\kappa(k, n)$ is an exponential tower of height k, i.e., $\kappa(0, n) = n$ and $\kappa(k + 1, n) = 2^{\kappa(k,n)}$.

2.3 Finite-state automata

In this section we introduce finite-state automata for sequences and trees and link them to monadic theories defined in Section 2.2.

We begin with some preliminary definitions. We say that a class $\mathcal{A}$ of automata is *closed* under an n-ary operation O over $\mathcal{A}$ if $O(A_1, \ldots, A_n) \in \mathcal{A}$ whenever $A_1, \ldots, A_n \in \mathcal{A}$. We say that $\mathcal{A}$ is *effectively closed* under O if $\mathcal{A}$ is closed under O and there is an algorithm that computes $O(A_1, \ldots, A_n)$. Given an automaton A recognizing objects in the set Ω, we denote by $\mathcal{L}(A) \subseteq \Omega$ the language accepted by A. We will often consider the Boolean operations of union, intersection, and complementation over automata. The union operation $\cup$ is a binary operation such that $A = A_1 \cup A_2$ iff $\mathcal{L}(A) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$. The intersection operation $\cap$ is a binary operation such that $A = A_1 \cap A_2$ iff $\mathcal{L}(A) = \mathcal{L}(A_1) \cap \mathcal{L}(A_2)$. Finally, the complementation operation is a unary operation such that $A = \overline{A_1}$ iff $\mathcal{L}(A) = \overline{\mathcal{L}(A_1)} = \Omega \setminus \mathcal{L}(A_1)$. We say that a class $\mathcal{A}$ of automata is *decidable* if, given $A \in \mathcal{A}$, the problem $\mathcal{L}(A) = \emptyset$ (the language accepted by A is the empty set) is decidable.

Let $\Sigma = \{a, b, \ldots\}$ be is a finite set of symbols. A Σ-labeled finite sequence is a *Kripke structure* $(I, <, V)$, where $(I, <)$ is a finite sequence and $V : I \rightarrow \Sigma$ is a valuation function. Similarly for infinite sequences, finite and infinite trees, and layered structures defined in Section 2.1. In the following, we define automata accepting Σ-labeled sequences and automata accepting Σ-labeled trees. We start by defining finite-state automata over sequences. The simplest automata class is that of finite sequence automata.

Definition 2.3.1 (*Finite sequence automata*)

A (nondeterministic) finite sequence automaton *A over the alphabet Σ is a quadruple (Q, q_0, Δ, F), where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, and $F \subseteq Q$ is a set of final states. Given a Σ-labeled finite sequence $w = (\{0, \ldots, n\}, <, V)$, a* run *of A on w is a function $\sigma : \{0, \ldots, n+1\} \rightarrow Q$ such that $\sigma(0) = q_0$ and $(\sigma(i), V(i), \sigma(i+1)) \in \Delta$, for every $0 \leq i \leq n$. The automaton A accepts w if there is a run σ of A on w such that $\sigma(n+1) \in F$. The language accepted by A, denoted by $\mathcal{L}(A)$, is the set of Σ-labeled finite sequences accepted by A.*

It is folklore knowledge that finite sequence automata are effectively closed under Boolean operations and are decidable in polynomial time [68]. Moreover, deterministic and nondeterministic finite sequence automata are expressively equivalent. By exploiting a 'subset construction', a nondeterministic finite sequence automaton can be converted into an equivalent deterministic one of size exponential in the size of the input automaton. We now introduce infinite sequence automata.

Definition 2.3.2 (*Büchi sequence automata*)

A (nondeterministic) Büchi sequence automaton *A over the alphabet Σ is a quadruple (Q, q_0, Δ, F), where the components are as for finite sequence automata. Given a Σ-labeled infinite sequence $w = (\mathbb{N}, <, V)$, a* run *of A on w is function $\sigma : \mathbb{N} \rightarrow Q$ such that $\sigma(0) = q_0$ and $(\sigma(i), V(i), \sigma(i+1)) \in \Delta$, for every $i \geq 0$. The automaton A accepts w if there is a run σ of A on w such that some final state $q \in F$ occurs infinitely often in σ. The language accepted by A, denoted by $\mathcal{L}(A)$, is the set of Σ-labeled infinite sequences accepted by A. Let $\mathcal{B}$ be the class of Büchi sequence automata.*

Büchi sequence automata are effectively closed under Boolean operations and are decidable in polynomial time (the emptiness problem is NLOGSPACE-complete) [114]. However, deterministic Büchi sequence automata are *not* closed under complementation, and hence are strictly less expressive than their nondeterministic counterpart. In the following, we define a class of *deterministic* sequence automata which is expressively equivalent to the class of nondeterministic Büchi sequence automata.

Definition 2.3.3 (*Rabin sequence automata*)

A (deterministic) Rabin sequence automaton *A over the alphabet Σ is a quadruple (Q, q_0, δ, Γ), where Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \rightarrow Q$ is a transition function, and $\Gamma = \{(L_1, U_1), \ldots, (L_m, U_m)\}$ is a set of accepting pairs (L_i, U_i) with $L_i, U_i \subseteq Q$. Given a Σ-labeled infinite sequence $w = (\mathbb{N}, <, V)$, the (unique)* run *of A on w is the function $\sigma : \mathbb{N} \rightarrow Q$ such that $\sigma(0) = q_0$ and $\delta(\sigma(i), V(i)) = \sigma(i+1)$, for every $i \geq 0$. The automaton A accepts w if there is $i \in \{1, \ldots, m\}$ such that the L_i-states occur*

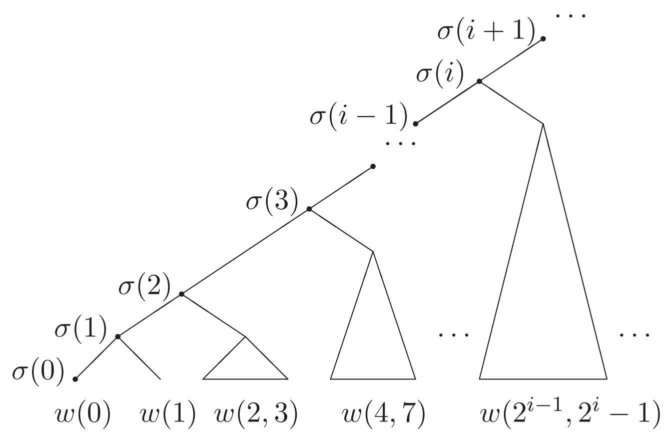

Figure 2.4: A systolic run σ on w.

only finitely often in σ and some U_i-state occurs infinitely often in σ. The language accepted by A, denoted by $\mathcal{L}(A)$, is the set of Σ-labeled infinite sequences accepted by A.

A remarkable result due to McNaughton is that Rabin and Büchi sequence automata have the same expressive power [90]. Moreover, there is an effective translation from Büchi to Rabin sequence automata and viceversa. By using Safra's construction [108], a Büchi sequence automaton with n states can be converted into a Rabin sequence automaton with $2^{O(n \cdot \log n)}$ states and $O(n)$ accepting pairs.

In the following we introduce systolic automata (see [60] for a survey on systolic computations). Systolic automata recognize labeled sequences working in a bottom-up fashion over a treelike structure. We will only consider systolic automata recognizing labeled infinite sequences [98, 99]. We will introduce three differently expressive classes of systolic automata: systolic tree automata, systolic Y-tree automata and systolic trellis automata. We first introduce systolic binary tree automata. The definition for the k-ary case, with $k > 2$, is similar. It is convenient to view a $\mathcal{P}$-labeled finite sequence as a finite string over the alphabet $2^{\mathcal{P}}$: given a finite sequence $s = (\{0, \ldots, n\}, <, (P)_{P \in \mathcal{P}})$, it corresponds to the finite string $x_0 \ldots x_n$ over $2^{\mathcal{P}}$, where, for every $i \in \{0, \ldots, n\}$, $x_i = \{P \in \mathcal{P} \mid i \in P\}$. For instance, the $\{P, Q\}$-labeled finite sequence $\langle \{0,1\}, <, P, Q \rangle$ such that $P = \{0,1\}$ and $Q = \{0\}$ corresponds to the finite string $\{P,Q\}\{P\}$ over $2^{\{P,Q\}}$. Accordingly, we will also write $s(i)$ to denote x_i, $s(i,j)$ to denote the segment $s(i) \ldots s(j)$, for $i \leq j$, and $|s|$ to denote the length of s. Similarly for infinite sequences.

Informally, a systolic binary tree computation works as follows. Consider a labeled infinite sequence w. At each computation step, the automaton processes a segment of w whose length increases by a factor of 2 step by step. In particular, an infinite sequence labeled with states in Q stores at the i-th position the state q resulting from processing the prefix $w(0, 2^i - 1)$ of w. The state resulting from processing the next prefix $w(0, 2^{i+1} - 1)$ is obtained from q and from the state q' output by the systolic automaton fed with $w(2^i, 2^{i+1} - 1)$, according to a transition relation. In Figure 2.4, we graphically describe the way in which a systolic binary tree automaton processes an infinite sequence w. The left-hand side edge of the tree structure consists of nodes associated with states obtained by processing prefixes of w whose length is a power of 2. Such a sequence of states is called a *systolic run*. Formally, a systolic binary tree automaton is defined as follows.

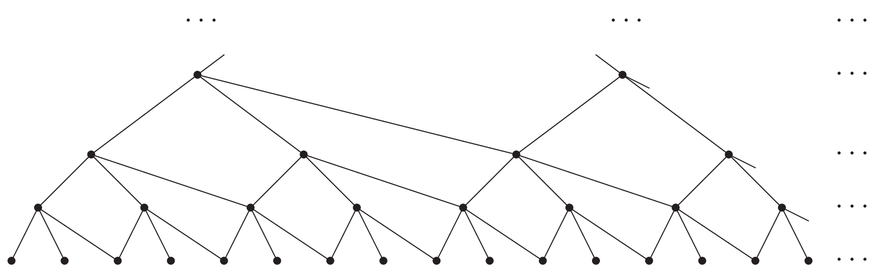
Figure 2.5: An upward unbounded Y-tree structure.

Definition 2.3.4 (*Systolic binary tree automata*)

A (nondeterministic) systolic binary tree automaton *A over the alphabet Σ is a quadruple (Q, in, Δ, F), where Q is a finite set of states, $in \subseteq \Sigma \times Q$ is an input relation, $\Delta \subseteq Q \times Q \times Q$ is a transition relation, and $F \subseteq Q$ is a set of final states. The computation of A over a Σ-labeled finite sequence $w = x_1 \dots x_{2^m}$ of length 2^m is a binary relation $O_A \subseteq \Sigma^* \times Q$ recursively defined as follows:*

- *if $|w| = 1$, then $(w, q) \in O_A$ if and only if $(w, q) \in in$;*
- *if $|w| = 2^m$, with $m > 0$, then let $w_1 = x_1 \dots x_{2^{m-1}}$ and $w_2 = x_{2^{m-1}+1} \dots x_{2^m}$. We have $(w, q) \in O_A$ if and only if $(q_1, q_2, q) \in \Delta$, where $(w_1, q_1) \in O_A$ and $(w_2, q_2) \in O_A$.*

Given a Σ-labeled infinite sequence $w = (\mathbb{N}, <, V)$, a systolic run of A on w is a function $\sigma : \mathbb{N} \to Q$ such that:

- *$(V(0), \sigma(0)) \in in$;*
- *$(\sigma(i-1), q, \sigma(i)) \in \Delta$, with $(w(2^{i-1}, 2^i - 1), q) \in O_A$, for every $i > 0$.*

The automaton A accepts w if there exists a systolic run σ of A on w such that some final state $q \in F$ occurs infinitely often in σ. The language recognized by A, denoted by $\mathcal{L}(A)$, is the set of Σ-labeled infinite sequences accepted by A.

The class of systolic k-ary tree automata is closed under Boolean operations and is decidable (the emptiness problem is PSPACE-complete) [98]. Moreover, systolic k-ary tree automata are strictly more expressive than Büchi sequence automata. Indeed, the language $\mathcal{L} = \{a^{2^i}\{b\}^\omega \mid i \geq 0\}$ is recognized by the following systolic binary tree automaton (Q, in, Δ, F) over the alphabet $\{a, b\}$, where

- $Q = \{q_1, q_2, q_3\}$ and $F = \{q_3\}$;
- $in = \{(a, q_1), (b, q_2)\}$;
- $\Delta = \{(q_1, q_1, q_1), (q_2, q_2, q_2), (q_1, q_2, q_3), (q_3, q_2, q_3)\}$.

However, $\mathcal{L}$ is not regular, and hence cannot be recognized by any Büchi sequence automaton [98].

There exist interesting connections between systolic tree automata and upward unbounded layered structures. A systolic computation corresponds to a labelling (over the

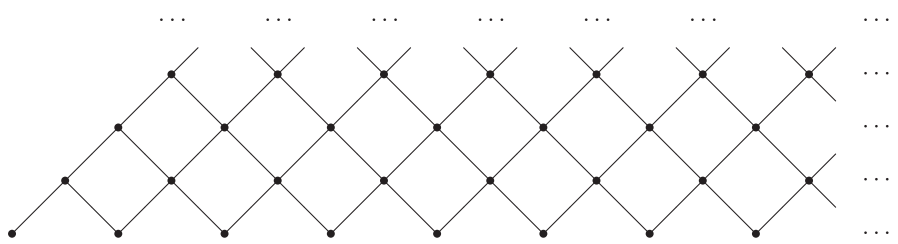

Figure 2.6: An upward unbounded trellis structure.

alphabet of states of the automaton) of an upward unbounded layered structure according to the transition relation of the automaton. The labeled infinite sequence to be processed is associated with the first layer (the finest one) of the upward unbounded layered structure, adjacent symbols being associated with adjacent nodes. The first layer is processed according to the initial relation. Then, the computation flows up, in parallel and synchronously, level by level, according to the transition relation. In order to accept a sequence, the states of the leftmost path of the layered structure are checked for a Büchi acceptance condition.

Similarly, *systolic Y-tree automata* and *systolic trellis automata* compute by labeling the first layer of a suitable layered structure with the infinite sequence to be processed and proceed bottom-up level by level. Nodes of a layer are connected to nodes of the previous layer drawing a regular topology that, in the case of systolic Y-tree automata, is an upward unbounded Y-tree (cf. Figure 2.5) and, in the case of systolic trellis automata, is an upward unbounded trellis (cf. Figure 2.6). Formal definitions for systolic Y-tree automata and systolic trellis automata over infinite sequences can be found in [99]. The connection between systolic automata and corresponding layered structures will be studied in Chapter 5. Systolic Y-tree automata are strictly more expressive than systolic tree automata. They are not closed under complementation and they are not decidable. Moreover, systolic trellis automata are strictly more expressive than Y-tree automata, and hence they are not decidable. The closure under complementation problem for systolic trellis automata is open and it is related to the same problem for the famous complexity class NP [99].

Finally, we introduce automata over trees.

Definition 2.3.5 (*Finite tree automata*)

A (nondeterministic bottom-up) finite binary tree automaton A *over the alphabet* Σ *is a quadruple* (Q, in, Δ, F) *where* Q *is a finite set of states,* $in \subseteq \Sigma \times Q$ *is an input relation,* $\Delta \subseteq Q \times Q \times \Sigma \times Q$ *is a transition relation, and* $F \subseteq Q$ *is a set of final states. Given a* Σ*-labeled finite binary tree* $t = (D, \downarrow_0, \downarrow_1, <_{pre}, V)$*, a* run *of* A *on* t *is a function* $\sigma : D \rightarrow Q$ *such that, for every leaf* u *of* t*,* $(V(u), \sigma(u)) \in in$ *and for every internal node* v *of* t*,* $(\sigma(\downarrow_0(v)), \sigma(\downarrow_1(v)), V(v), \sigma(v)) \in \Delta$*. The automaton* A *accepts* t *if there is a run* σ *of* A *on* t *such that* $\sigma(\epsilon) \in F$*. The language accepted by* A*, denoted by* $\mathcal{L}(A)$*, is the set of* Σ*-labeled finite trees accepted by* A*. Finite* k*-ary tree automata, for* $k > 2$*, are defined similarly. Let* $\mathcal{D}_k$ *be the class of finite* k*-ary tree automata.*

Finite tree automata are effectively closed under Boolean operations and are decidable in polynomial time [28, 113]. Moreover, by exploiting a 'subset construction', a nondeterministic bottom-up finite tree automaton can be converted into a deterministic one accepting the same language. The size of the resulting automaton is exponential in the size of the input automaton.

Infinite tree automata are defined as follows.

Definition 2.3.6 (*Büchi and Rabin tree automata*)

A (nondeterministic) Büchi binary tree automaton A *over the alphabet* Σ *is a quadruple* (Q, q_0, Δ, F)*, where* Q *is a finite set of states,* $q_0 \in Q$ *is an initial state,* $\Delta \subseteq Q \times \Sigma \times Q \times Q$ *is a transition function, and* F *is a set of final states. Given a* Σ*-labeled infinite binary tree* $t = (T_2, \downarrow_0, \downarrow_1, <_{pre}, V)$*, a* run *of* A *on* t *is a function* $\sigma : T_2 \to Q$ *such that* $\sigma(\epsilon) = q_0$ *and, for every node* v *of* t*,* $(\sigma(v), V(v), \sigma(\downarrow_0(v)), \sigma(\downarrow_1(v))) \in \Delta$*. Given a path* π *in* t*, we denote by* $\sigma|\pi$ *the infinite sequence* $\sigma(\pi(0)), \sigma(\pi(1)) \ldots$ *The automaton* A *accepts* t *if there is a run* σ *of* A *on* t *such that, for every full path* π *of* t*, there exists some final state* $q \in F$ *that occurs infinitely often in* $\sigma|\pi$*. The language accepted by* A*, denoted by* $\mathcal{L}(A)$*, is the set of* Σ*-labeled infinite binary trees accepted by* A*.*

A (nondeterministic) Rabin binary tree automaton A *over the alphabet* Σ *is a quadruple* (Q, q_0, Δ, Γ)*, where* Q *is a finite set of states,* $q_0 \in Q$ *is an initial state,* $\Delta \subseteq Q \times \Sigma \times Q \times Q$ *is a transition function, and* $\Gamma = \{(L_1, U_1), \ldots, (L_m, U_m)\}$ *is a set of accepting pairs* (L_i, U_i) *with* $L_i, U_i \subseteq Q$*. A* run *of* A *is defined as in the case of Büchi tree automata. The automaton* A *accepts* t *if there is a run* σ *of* A *on* t *such that, for every full path* π *of* t*, there exists* $i \in \{1, \ldots, m\}$ *such that the* L_i*-states occur only finitely often in* $\sigma|\pi$ *and some* U_i*-state occurs infinitely often in* $\sigma|\pi$*. The language accepted by* A*, denoted by* $\mathcal{L}(A)$*, is the set of* Σ*-labeled binary trees accepted by* A*. Rabin* k*-ary tree automata, for* $k > 2$*, are defined similarly. Let* $\mathcal{R}_k$ *be the class of Rabin* k*-ary tree automata.*

Büchi tree automata can be embedded into Rabin tree automaton: given a Büchi tree automaton (Q, q_0, Δ, F), an equivalent Rabin tree automaton is $(Q, q_0, \Delta, \{(\emptyset, F)\})$. The opposite embedding does not hold in general; indeed, Büchi tree automata are not closed under complementation while Rabin tree automata are effectively closed under Boolean operations. Both Büchi and Rabin tree automata are decidable (the emptiness problem is P-complete for Büchi tree automata, and NP-complete for Rabin tree automata) [114]. In particular, the emptiness problem for a Rabin tree automaton with n states and m accepting pairs is solvable in time $(n \cdot m)^{O(m)}$, and hence it is linear in the number of states [34].

In the following, we compare the expressive power of the above defined finite-state automata classes with that of the monadic theories introduced in Section 2.2. Recall that, as explained in Remark 2.1.1, labeled Kripke structures accepted by automata correspond to labeled relational structures over which monadic classical logics are interpreted, and vice versa. We say that an automata class $\mathcal{A}$ is *embeddable* into a classical logic $\mathbf{L}$, denoted by $\mathcal{A} \to \mathbf{L}$, if there is an *effective* translation τ of $\mathcal{A}$-automata into $\mathbf{L}$-formulas such that, for every $\mathcal{A}$-automaton A, $\mathcal{L}(A) = \mathcal{M}(\tau(A))$. We say that a classical logic $\mathbf{L}$ is *embeddable* into an automata class $\mathcal{A}$, denoted by $\mathbf{L} \to \mathcal{A}$, if there is an *effective* translation τ of $\mathbf{L}$-formulas into $\mathcal{A}$-automata such that, for every $\mathbf{L}$-formula φ, $\mathcal{L}(\tau(\varphi)) = \mathcal{M}(\varphi)$. Finally, $\mathcal{A}$ is *expressively equivalent* to $\mathbf{L}$, written $\mathcal{A} \rightleftarrows \mathbf{L}$, if both $\mathbf{L} \to \mathcal{A}$ and $\mathcal{A} \to \mathbf{L}$. The following results are well-known.

Theorem 2.3.7 (*Expressiveness of sequence and tree automata*)

1. $\mathrm{MSO}_{\mathcal{P}}[<]$ *is expressively equivalent to finite sequence automata over finite sequences* [12, 32]*;*

2. $\mathrm{MSO}_{\mathcal{P}}[<]$ *is expressively equivalent to Büchi automata over infinite sequences* [13, 90]*;*
3. $\mathrm{MSO}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ *is expressively equivalent to finite k-ary tree automata over finite k-ary trees* [113, 28]*;*
4. $\mathrm{MSO}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ *is expressively equivalent to Rabin k-ary tree automata over infinite k-ary trees* [107].

Some more recent results are the following ones.

Theorem 2.3.8 (*Expressiveness of systolic automata*)

1. $\mathrm{MSO}_{\mathcal{P}}[<, \mathtt{flip}_k]$ *is expressively equivalent to systolic k-ary tree automata over infinite sequences* [98]*;*
2. *Systolic Y-tree automata are embeddable into* $\mathrm{MSO}_{\mathcal{P}}[<, \mathtt{adj}]$ *over infinite sequences* [99]*;*
3. *Systolic trellis automata are embeddable into* $\mathrm{MSO}_{\mathcal{P}}[<, 2\times]$ *over infinite sequences* [99].

The converse of (2) in Theorem 2.3.8 does not hold, since Y-tree are not closed under complementation. The converse of (3) in the same theorem holds if and only if trellis automata are closed under complementation. As already pointed out, the latter holds if and only if the complexity class NP is closed under complementation.

2.4 Temporal logics

In this section we introduce temporal logics for sequences and trees and link them to monadic theories defined in Section 2.2.

Let $\mathcal{P} = \{P, Q, \ldots\}$ be the finite set of proposition letters. We will interpret temporal logics over $\mathcal{P}$-labeled *Kripke structures* (W, τ, V), where (W, τ) is a sequence or a tree, and $V : W \rightarrow 2^{\mathcal{P}}$ is a valuation function.

Definition 2.4.1 (*Linear and branching time logics*)

We inductively define a set of state *formulas and a set of* path *formulas:*

- *state formulas*

  (S1) any proposition letter $P \in \mathcal{P}$ is a state formula;

  (S2) if p, q are state formulas, then $p \wedge q$ and $\neg p$ are state formulas;

  (S3) if p is a path formula, then $\mathbf{A}p$ and $\mathbf{E}p$ are state formulas;

- *path formulas*

  (P0) any proposition letter $P \in \mathcal{P}$ is a path formula;

  (P1) any state formula is a path formula;

  (P2) if p, q are path formulas, then $p \wedge q$ and $\neg p$ are path formulas;

  (P3) if p, q are path formulas, then $\mathbf{X}p$, and $p\mathbf{U}q$ are path formulas;

  (P4) if p is a path formula, then $\mathbf{X_0}p, \ldots, \mathbf{X_{k-1}}p$ are path formulas;

  (P5) if p, q are path formulas, then $\mathbf{X^{-1}}p$ and $p\mathbf{S}q$ are path formulas.

(P6) if p, q are state formulas, then $\mathbf{X}p$, and $p\mathbf{U}q$ are path formulas;

The languages of branching time logics *are the smallest sets of state formulas generated by the above rules as follows:*

- Past Directed CTL* *(*$\mathrm{PCTL}^*_\mathrm{k}$ *for short): rules (S1-S3) and (P1-P5);*
- Directed CTL* *(*$\mathrm{CTL}^*_\mathrm{k}$*): rules (S1-S3) and (P1-P4);*
- Past CTL* *(*PCTL^**): rules (S1-S3) and (P1-P3,P5);*
- CTL**: rules (S1-S3) and (P1-P3);*
- CTL*: rules (S1-S3) and (P6).*

The languages of linear time logics *are the smallest sets of path formulas generated by the above rules as follows:*

- Past PLTL *(*PPLTL*): rules (P0,P2,P3,P5);*
- PLTL*: rules (P0,P2,P3).*

Finally, let QLTL *be* Quantified *Linear Temporal Logic. The language of* QLTL *is obtained by adding the following rule to the formation rules of* PLTL*: if φ is a* QLTL*-formula and $Q \in \mathcal{P}$ is a proposition letter occurring free in φ, then $\exists Q\varphi$ is a formula. Moreover, let* EQLTL *be the fragment of* QLTL *consisting of formulas of the form $\exists Q_1 \ldots \exists Q_n\varphi$, where φ is a* PLTL*-formula. Similarly* $\mathrm{QCTL}^*_\mathrm{k}$ *and* $\mathrm{EQCTL}^*_\mathrm{k}$ *can be defined.*

Formulas $\mathbf{F}p$, $\mathbf{G}p$, $\mathbf{P}p$ and $\mathbf{H}p$ are respectively defined as $\mathtt{true}\mathbf{U}p$, $\neg\mathbf{F}\neg p$, $\mathtt{true}\mathbf{S}p$ and $\neg\mathbf{P}\neg p$ as usual, where $\mathtt{true} = P \vee \neg P$, for some $P \in \mathcal{P}$.

We interpret (P)PLTL over $\mathcal{P}$-labeled finite and infinite sequences. The semantics of (P)PLTL over infinite sequences is as follows.

Definition 2.4.2 (*Semantics for linear time logics*)

Let $\mathcal{M} = (\mathbb{N}, <, V)$ be a $\mathcal{P}$-labeled infinite sequence and $i \in \mathbb{N}$. We now define the notion of truth of a PPLTL*-formula ψ in a model $\mathcal{M}$ with respect to a point i, denoted by $\mathcal{M}, i \models \psi$, as follows:*

$$\begin{array}{lll}
\mathcal{M}, i \models P & \textit{iff} & P \in V(i) \textit{ for } P \in \mathcal{P} \\
\mathcal{M}, i \models \phi \wedge \psi & \textit{iff} & \mathcal{M}, i \models \phi \textit{ and } \mathcal{M}, i \models \psi \\
\mathcal{M}, i \models \neg\phi & \textit{iff} & \textit{it is not the case that } \mathcal{M}, i \models \phi \\
\mathcal{M}, i \models \phi\mathbf{U}\psi & \textit{iff} & \mathcal{M}, j \models \psi \textit{ for some } j \geq i \textit{ and} \\
 & & \mathcal{M}, k \models \phi \textit{ for every } i \leq k < j; \\
\mathcal{M}, i \models \mathbf{X}\psi & \textit{iff} & \mathcal{M}, i+1 \models \psi; \\
\mathcal{M}, i \models \phi\mathbf{S}\psi & \textit{iff} & \mathcal{M}, j \models \psi \textit{ for some } j \leq i \textit{ and} \\
 & & \mathcal{M}, k \models \phi \textit{ for every } j < k \leq i; \\
\mathcal{M}, i \models \mathbf{X^{-1}}\psi & \textit{iff} & i > 0 \textit{ and } \mathcal{M}, i-1 \models \psi.
\end{array}$$

We say that $\mathcal{M}$ is a model of ψ if $\mathcal{M}, 0 \models \psi$. We denote by $\mathcal{M}(\psi)$ the set of models of ψ.

The semantics of (P)PLTL over finite sequences differs only in the definition of the operators $\mathbf{X}$ and $\mathbf{U}$. Let $\mathcal{M} = (I, <, V)$ be a $\mathcal{P}$-labeled finite sequence, with $I = \{0, \ldots, n\}$, and $i \in I$. We have that:

$$\begin{array}{lll} \mathcal{M}, i \models \phi \mathbf{U} \psi & \text{iff} & \mathcal{M}, j \models \psi \text{ for some } i \leq j \leq n \text{ and} \\ & & \mathcal{M}, k \models \phi \text{ for every } i \leq k < j; \\ \mathcal{M}, i \models \mathbf{X}\psi & \text{iff} & i < n \text{ and } \mathcal{M}, i+1 \models \psi. \end{array}$$

The semantics of quantified formulas of QLTL is the following. Given a $\mathcal{P}$-labeled infinite sequence $\mathcal{M} = (\mathbb{N}, <, V)$ and $i \in \mathbb{N}$, we have $\mathcal{M}, i \models \exists Q \phi$ iff there exists $\mathcal{M}' = (\mathbb{N}, <, V')$ such that $\mathcal{M}', i \models \phi$, where V' differs from V in at most the truth value of Q, that is, for every $i \in \mathbb{N}$, either $V(i) = V'(i)$ or $V'(i) \setminus V(i) = \{Q\}$. Similarly in the finite case.

We interpret (P)CTL* over $\mathcal{P}$-labeled finite and infinite trees. The semantics of (P)CTL* over infinite trees is as follows.

Definition 2.4.3 (*Semantics for branching time logics*)

Let $\mathcal{M} = (T_k, (\downarrow_i)_{i=0}^{k-1}, <, V)$ be a $\mathcal{P}$-labeled infinite tree and $w \in T_k$. We define the notion of truth of a state PCTL**-formula ψ in a model $\mathcal{M}$ with respect to a point w, denoted by $\mathcal{M}, w \models \psi$, as follows: for any proposition letter $P \in \mathcal{P}$:*

$$\mathcal{M}, w \models P \quad \textit{iff} \quad P \in V(w).$$

For any state formulas ϕ and ψ:

$$\begin{array}{lll} \mathcal{M}, w \models \phi \wedge \psi & \textit{iff} & \mathcal{M}, w \models \phi \textit{ and } \mathcal{M}, w \models \psi; \\ \mathcal{M}, w \models \neg\phi & \textit{iff} & \textit{it is not the case that } \mathcal{M}, w \models \phi. \end{array}$$

For any path formula ϕ:

$$\begin{array}{lll} \mathcal{M}, w \models \mathbf{E}\phi & \textit{iff} & \textit{there is a path } \pi \textit{ in } \mathcal{M} \textit{ starting at } w \textit{ such that } \mathcal{M}, \pi, 0 \models \phi; \\ \mathcal{M}, w \models \mathbf{A}\phi & \textit{iff} & \textit{for every path } \pi \textit{ in } \mathcal{M} \textit{ starting at } w \textit{ we have } \mathcal{M}, \pi, 0 \models \phi. \end{array}$$

Moreover, let π be a path in $\mathcal{M}$ and $i \in \mathbb{N}$. We define the notion of truth of a path PCTL**-formula ϕ in a model $\mathcal{M}$ with respect to a path π and a position i, denoted by $\mathcal{M}, \pi, i \models \phi$, as follows.*

For any state formula ψ:

$$\mathcal{M}, \pi, i \models \psi \quad \textit{iff} \quad \mathcal{M}, \pi(i) \models \psi.$$

For any path formulas ϕ and ψ:

$$\begin{array}{lll} \mathcal{M}, \pi, i \models \phi \mathbf{U} \psi & \textit{iff} & \mathcal{M}, \pi, j \models \psi \textit{ for some } j \geq i \textit{ and} \\ & & \mathcal{M}, \pi, k \models \phi \textit{ for every } i \leq k < j; \\ \mathcal{M}, \pi, i \models \mathbf{X}\psi & \textit{iff} & \mathcal{M}, \pi, i+1 \models \psi; \\ \mathcal{M}, \pi, i \models \phi \mathbf{S} \psi & \textit{iff} & \mathcal{M}, \pi, j \models \psi \textit{ for some } 0 \leq j \leq i \textit{ and} \\ & & \mathcal{M}, \pi, k \models \phi \textit{ for every } j \leq k < i; \\ \mathcal{M}, \pi, i \models \mathbf{X^{-1}}\psi & \textit{iff} & i > 0 \textit{ and } \mathcal{M}, \pi, i-1 \models \psi. \end{array}$$

We say that $\mathcal{M}$ is a model of ψ if $\mathcal{M}, \epsilon \models \psi$. We denote by $\mathcal{M}(\psi)$ the set of models of ψ.

The semantics of (P)CTL* over finite trees slightly differs in the semantics for $\mathbf{X}$ and $\mathbf{U}$ only, as for linear time logics. The semantic interpretation for (P)CTL$^*_{\mathrm{k}}$ coincides with that for (P)CTL*, except for path formulas of the form $\mathbf{X_j}p$, whose interpretation is defined as follows:

$$\mathcal{M}, \pi, i \models \mathbf{X_j}p \text{ iff } \pi(i+1) = \downarrow_j(\pi(i)) \text{ and } \mathcal{M}, \pi, i+1 \models p.$$

Finally, the semantics of quantified formulas in $\mathrm{QCTL}_\mathrm{k}^*$ is obtained as in the linear time case.

In the following, we compare the expressive power of the above defined temporal logics with that of the monadic theories introduced in Section 2.2, and hence also with that of automata defined in Section 2.3. Recall that, according to Remark 2.1.1, labeled Kripke structures over which temporal logics are interpreted correspond to labeled relational structures over which monadic classical logics are interpreted, and vice versa. We say that a temporal logic $\mathbf{T}$ is *embeddable* into a classical logic $\mathbf{L}$, denoted by $\mathbf{T} \rightarrow \mathbf{L}$, if there is an *effective* translation τ of $\mathbf{T}$-formulas into $\mathbf{L}$-formulas such that for every $\mathbf{T}$-formula φ, $\mathcal{M}(\varphi) = \mathcal{M}(\tau(\varphi))$. A similar definition holds for $\mathbf{L} \rightarrow \mathbf{T}$. Finally, $\mathbf{T}$ is *expressively equivalent* to $\mathbf{L}$, written $\mathbf{T} \rightleftarrows \mathbf{L}$, if both $\mathbf{L} \rightarrow \mathbf{T}$ and $\mathbf{T} \rightarrow \mathbf{L}$.

As for linear time logic, it is well-known that, when interpreted over the class of finite sequences as well as over the class of infinite ones, PLTL and PPLTL are expressively equivalent to MFO[<] [55, 75].

Theorem 2.4.4 (*Expressiveness of* PLTL *and* PPLTL)

PLTL *and* PPLTL *are expressively equivalent to* $\mathrm{MFO}_\mathcal{P}[<]$*, when interpreted over finite (resp. infinite) sequences.*

Both QLTL and EQLTL capture the full second-order expressiveness of MSO[<] [86].

Theorem 2.4.5 (*Expressiveness of* QLTL *and* EQLTL)

QLTL *and* EQLTL *are expressively equivalent to* $\mathrm{MSO}_\mathcal{P}[<]$*, when interpreted over finite (resp. infinite) sequences.*

The satisfiability problem for PLTL is PSPACE-complete [111]. While QLTL is nonelementarily decidable [118], EQLTL has the same complexity as PLTL: given an EQLTL-formula $\phi = \exists Q_1, \dots \exists Q_n \varphi$, we have that ϕ is satisfiable if and only if φ is satisfiable. A model for ϕ can be obtained from a model for φ by deleting proposition letters Q_i. It is worth recalling that there exists an elementary translation of PLTL-formulas into Büchi sequence automata: given a PLTL-formula φ over $\mathcal{P}$ of length n, one can construct an equivalent Büchi sequence automaton A_φ over $2^\mathcal{P}$ with $2^{O(n)}$ states [86, 118].

As for branching time logic, the expressive power of CTL* and PCTL* is equivalent to the one of monadic path logic over infinite binary trees [61].

Theorem 2.4.6 (*Expressiveness of* CTL* *and* PCTL*)

CTL* *and* PCTL* *are expressively equivalent to* $\mathrm{MPL}_\mathcal{P}[<_{pre}]$*, when interpreted over infinite binary trees.*

As pointed out by Hafer and Thomas [61], Theorem 2.4.6 can be generalized to $\mathrm{CTL}_\mathrm{k}^*$ and $\mathrm{PCTL}_\mathrm{k}^*$ with respect to $\mathrm{MPL}_\mathcal{P}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ by incorporating successors into both temporal and monadic path logics.

Theorem 2.4.7 (*Expressiveness of* $\mathrm{CTL}_\mathrm{k}^*$ *and* $\mathrm{PCTL}_\mathrm{k}^*$)

$\mathrm{CTL}_\mathrm{k}^*$ *and* $\mathrm{PCTL}_\mathrm{k}^*$ *are expressively equivalent to* $\mathrm{MPL}_\mathcal{P}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$*, when interpreted over finite (resp. infinite) k-ary trees.*

Moreover, both $\mathrm{QCTL}_\mathrm{k}^*$ and $\mathrm{EQCTL}_\mathrm{k}^*$ gain the full second-order power of $\mathrm{MSO}_\mathcal{P}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ [35]:

Theorem 2.4.8 (*Expressiveness of* $\mathrm{QCTL}^*_\mathrm{k}$ *and* $\mathrm{EQCTL}^*_\mathrm{k}$)

$\mathrm{QCTL}^*_\mathrm{k}$ and $\mathrm{EQCTL}^*_\mathrm{k}$ are expressively equivalent to $\mathrm{MSO}_\mathcal{P}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$, when interpreted over finite (resp. infinite) k-ary trees.

Emerson and Jutla proved that the satisfiability problem for CTL* is 2EXPTIME-complete [34]. Furthermore, a decision procedure for $\mathrm{CTL}^*_\mathrm{k}$ can be obtained by means of the following non trivial adaptation of the decision procedure for CTL* originally developed by Emerson and Sistla [35] and later refined by Emerson and Jutla [34].

We start by defining an auxiliary linear time logic, called *Directed* k-ary PLTL ($\mathrm{PLTL_k}$), whose language is the smallest set of path formulas generated by the rules (P0), (P2-P5) in Definition 2.4.1. $\mathrm{PLTL_k}$ is interpreted over $\mathcal{P}$-labeled full paths belonging to k-ary infinite trees, that is, Kripke structures $(\pi, <_{pre}, V)$, where $\pi \subset T_k$ is a full path, $<_{pre}$ is the prefix ordering and $V : \pi \to 2^\mathcal{P}$ is a valuation function. Note that $<_{pre}$ is a total ordering over π. Let us denote by $\pi(i)$ the i-th element of π according to the total ordering $<_{pre}$. The semantic interpretation for $\mathrm{PLTL_k}$ coincides with that for PLTL, except for formulas of the form $\mathbf{X_j}p$, whose interpretation is defined as follows. Given $\mathcal{M} = (\pi, <_{pre}, V)$ and $i \in \mathbb{N}$,

$$\mathcal{M}, i \models \mathbf{X_j}p \text{ iff } \pi(i+1) = \downarrow_j(\pi(i)) \text{ and } \mathcal{M}, i+1 \models p.$$

Let us assume $k = 2$ (the generalization to an arbitrary k is straightforward). As a preliminary step, we provide an embedding of $\mathrm{PLTL_k}$ into PLTL. To this end, we define a translation τ of $\mathrm{PLTL_k}$-formulas into PLTL-formulas as follows:

$$\begin{array}{lcl}
\tau(P) & = & P \text{ for } P \in \mathcal{P} \\
\tau(\alpha \wedge \beta) & = & \tau(\alpha) \wedge \tau(\beta) \\
\tau(\neg\alpha) & = & \neg\tau(\alpha) \\
\tau(\mathbf{X_0}\alpha) & = & Z \wedge \mathbf{X}\tau(\alpha) \\
\tau(\mathbf{X_1}\alpha) & = & U \wedge \mathbf{X}\tau(\alpha) \\
\tau(\alpha\mathbf{U}\beta) & = & \tau(\alpha)\mathbf{U}\tau(\beta)
\end{array}$$

We define an *isomorphism* ι from $\mathrm{PLTL_k}$-models $\langle \pi, <_{pre}, V\rangle$ over $\mathcal{P}$ to PLTL-models $\langle \mathbb{N}, <, V\rangle$ over $\mathcal{P} \cup \{Z, U\}$ that maps a node $\pi(i)$, labeled with $X \subseteq \mathcal{P}$ and such that $\pi(i+1) = \downarrow_0(\pi(i))$ (resp. $\pi(i+1) = \downarrow_1(\pi(i))$), to a node $i \in \mathbb{N}$, labeled with $X \cup \{Z\}$ (resp. $X \cup \{U\}$). The following lemma can be easily proved.

Lemma 2.4.9 *Let $\psi = \mathbf{G}((Z \vee U) \wedge \neg(Z \wedge U))$. For every formula φ of $\mathrm{PLTL_k}$, we have that $\iota(\mathcal{M}(\varphi)) = \mathcal{M}(\psi \wedge \tau(\varphi))$.*

As a second preliminary step, we transform $\mathrm{CTL}^*_\mathrm{k}$-formulas in a normal form suitable for subsequent manipulation. Such a normal form is a straightforward generalization of the normal form for CTL*-formulas proposed by Emerson and Sistla [35]. This result is formally stated by the following lemma, whose proof is similar to the one for CTL* and thus omitted.

Lemma 2.4.10 *For any given $\mathrm{CTL}^*_\mathrm{k}$-formula φ_0, there exists a corresponding formula φ_1 in a normal form composed of conjunctions and disjunctions of subformulas of the form* $\mathbf{A}p$, $\mathbf{E}p$, *and* $\mathbf{AGE}p$, *where p is a pure linear time formula of $\mathrm{PLTL_k}$, such that (i) φ_1 is satisfiable if and only if φ_0 is satisfiable, and (ii) $|\varphi_1| = \mathcal{O}(|\varphi_0|)$. Moreover, any model of φ_1 can be used to define a model of φ_0 and vice versa.*

Theorem 2.4.11 ($\mathrm{CTL}^*_\mathrm{k}$ *is elementarily decidable*)

The satisfiability problem for $\mathrm{CTL}^*_\mathrm{k}$ over infinite k-ary trees is 2EXPTIME-complete.

Proof.

Let us assume $k = 2$. The general case is similar. Hardness follows from 2EXPTIME-hardness of CTL* [116]. To show that it belongs to 2EXPTIME, we outline an algorithm for checking the satisfiability of $\mathrm{CTL}^*_\mathrm{k}$-formulas with deterministic doubly exponential time complexity. Given a $\mathrm{CTL}^*_\mathrm{k}$-formula φ_0, such an algorithm is as follows:

1. by exploiting Lemma 2.4.10, construct an equivalent formula φ_1 composed of conjunctions and disjunctions of subformulas of the form $\mathbf{A}p$, $\mathbf{E}p$, and $\mathbf{AGE}p$, where p is a PLTL_k-formula;

2. by exploiting Lemma 2.4.9, replace every maximal PLTL_k-formula p (over $\mathcal{P}$) in φ_1 by the PLTL-formula $\eta(p) = \psi \wedge \tau(p)$ (over $\mathcal{P} \cup \{Z, U\}$); then, construct a Büchi sequence automaton $\mathcal{A}_{\eta(p)}$ (over $2^{\mathcal{P} \cup \{Z,U\}}$) recognizing the models of $\eta(p)$, by using the technique described in [35];

3. for every subformula of the form $\mathbf{A}p$ of φ_1, determinize the Büchi sequence automaton $A_{\eta(p)}$ for $\eta(p)$, using Safra's construction [108], to obtain an equivalent *deterministic* Rabin sequence automaton $A'_{\eta(p)}$ (over $2^{\mathcal{P} \cup \{Z,U\}}$) for $\eta(p)$;

4. program a Rabin tree automaton A_{φ_1} (over $2^{\mathcal{P}}$), accepting the models of φ_1, which incorporates the sequence automata built in steps 2 and 3 in a suitable way (see below);

5. test whether A_{φ_1} recognizes the empty language using the algorithm proposed by Emerson and Jutla [34].

Step 4 is as follows. For every subformula $\mathbf{E}p$ of φ_1, let $A_{\eta(p)} = (Q, q_0, \Delta, F)$ be the Büchi sequence automaton for $\eta(p)$. We construct the Büchi tree automaton $A_{\mathbf{E}p} = (Q \cup \{q_*\}, q_0, \Delta', F \cup \{q_*\})$ for $\mathbf{E}p$, where Δ' is defined as follows:

$$\begin{array}{lll}
\Delta'(q, X, q', q_*) & \text{if and only if} & \Delta(q, X \cup \{Z\}, q'); \\
\Delta'(q, X, q_*, q') & \text{if and only if} & \Delta(q, X \cup \{U\}, q'); \\
\Delta'(q_*, X, q_*, q_*) & \text{if and only if} & X \in 2^{\mathcal{P}}.
\end{array}$$

For every subformula $\mathbf{A}p$ of φ_1, let $A_{\eta(p)} = (Q, q_0, \Delta, \Gamma)$ be the deterministic Rabin sequence automaton for $\eta(p)$. We construct the deterministic Rabin tree automaton $A_{\mathbf{A}p} = (Q, q_0, \Delta', \Gamma)$ for $\mathbf{A}p$, where Δ' is defined as follows:

$$\Delta'(q, X, q', q'') \quad \text{if and only if} \quad \Delta(q, X \cup \{Z\}, q') \text{ and } \Delta(q, X \cup \{U\}, q'').$$

For every subformula $\mathbf{AGE}p$ of φ_1, let $A_{\eta(p)} = (Q, q_0, \Delta, F)$ be the Büchi sequence automaton for $\eta(p)$. We construct the Büchi tree automaton $A_{\mathbf{AGE}p} = (Q \cup \{q^* \mid q \in Q\}, q_0, \Delta', F \cup \{q^* \mid q \in Q\})$ for $\mathbf{AGE}p$, where Δ' is defined as follows:

$$\begin{array}{lll}
\Delta'(q, X, q', q_1^*) & \text{if and only if} & \Delta(q, X \cup \{Z\}, q') \text{ and } \Delta(q_0, X \cup \{U\}, q_1); \\
\Delta'(q, X, q_1^*, q') & \text{if and only if} & \Delta(q, X \cup \{U\}, q') \text{ and } \Delta(q_0, X \cup \{Z\}, q_1); \\
\Delta'(q, X, q', q_0^*) & \text{if and only if} & \Delta(q, X \cup \{Z\}, q') \text{ and } \Delta(q_0, X \cup \{Z\}, q'); \\
\Delta'(q, X, q_0^*, q') & \text{if and only if} & \Delta(q, X \cup \{U\}, q') \text{ and } \Delta(q_0, X \cup \{U\}, q'); \\
\Delta'(q^*, X, q', q'') & \text{if and only if} & \Delta'(q, X, q', q'').
\end{array}$$

A tree automaton A_{φ_1} for φ_1 is obtained by taking the intersection and/or the union of the Rabin tree automata constructed for the subformulas of φ_1 (recall that a Büchi tree automaton is a particular kind of Rabin tree automaton). Notice that if φ_1 does not contain subformulas of the form $\mathbf{X_i}p$, then A_{φ_1} is symmetric.

We show that the proposed algorithm has deterministic doubly exponential time complexity. Let φ_0 be a $\mathrm{CTL}^*_\mathrm{k}$-formula of length n. The normalized formula φ_1 after step 1 above has length $O(n)$. The Büchi sequence automaton $A_{\eta(p)}$ for $\eta(p)$ after step 2 has $2^{O(n)}$ states and the Rabin sequence automaton $A'_{\eta(p)}$ for $\eta(p)$ after step 3 has $2^{n \cdot 2^{O(n)}}$ states and $2^{O(n)}$ accepting pairs. Step 4 does not change the asymptotic complexity. Finally, the emptiness checking of step 5 leads to a time complexity of $2^{n \cdot 2^{O(n)}}$. Hence the upper bound is proved. ■

With minor modifications, the above doubly exponential upper bound holds in the case of finite trees also. In particular, nondeterministic Büchi sequence automata are replaced by nondeterministic finite sequence automata, deterministic Rabin sequence automata are replaced by deterministic finite sequence automata, and Rabin tree automata are replaced by finite tree automata.

Theorem 2.4.12 ($\mathrm{CTL}^*_\mathrm{k}$ *is elementarily decidable*)

The satisfiability problem for $\mathrm{CTL}^*_\mathrm{k}$ *over finite* k*-ary trees is in 2EXPTIME.*

Finally, $\mathrm{QCTL}^*_\mathrm{k}$ is nonelementarily decidable (because already QLTL is nonelementary), and $\mathrm{EQCTL}^*_\mathrm{k}$ has the asymptotic complexity of $\mathrm{CTL}^*_\mathrm{k}$. Indeed, given an $\mathrm{EQCTL}^*_\mathrm{k}$-formula $\psi = \exists Q_1 \ldots \exists Q_n \varphi$, we have that ψ and φ are equi-satisfiable.

We switch to the *model checking* problem for linear and branching time logics. We will define the model checking problem for undirected logics with respect to Kripke structures $\mathcal{M} = (W, R, V)$, where W is a set of worlds, $R \subseteq W \times W$ is a binary relation over W, and $V : W \to 2^{\mathcal{P}}$ is a valuation function [36]. Moreover, we will define the model checking problem for directed k-ary logics with respect to *directed* k*-ary Kripke structures* $\mathcal{M}_k = (W, R, L, V)$, where W is a set of worlds and $R \subseteq W \times W$ is a binary relation over W. Furthermore, $L : R \to \{0, \ldots, k-1\}$ is a labelling of edges with integers from 0 to $k-1$; if $L((w_1, w_2)) = i$ we say that w_2 is the i-th son of w_1. Finally, $V : W \to 2^{\mathcal{P}}$ is a valuation function. In the undirected case, we assume R to be *total*, that is, for every $w \in W$, there exists $v \in W$ such that $R(w, v)$. In the directed case, we assume that, for every $w \in W$, there are exactly k elements $w_0, \ldots, w_{k-1}$ such that, for $i = 0, \ldots, k-1$, w_i is the i-th son of w. The size of a Kripke structure is $|W| + |R|$. A structure is finite if its size is finite. A path in $\mathcal{M}$ is an infinite sequence $w_0, w_1, \ldots$ such that $R(w_i, w_{i+1})$ for every $i \geq 0$. A directed path in $\mathcal{M}_k$ is an infinite sequence $w_0, j_0, w_1, j_1 \ldots$ such that w_{i+1} is the j_i-th son of w_i, for every $i \geq 0$. Note that the *unfolding* of $\mathcal{M}$ is an infinite complete tree, while the unfolding of $\mathcal{M}_k$ is an infinite complete k-ary tree. We (re)interpret (directed) linear time logics over (directed) paths belonging to (directed) Kripke structures and (directed) branching time logics over (directed) Kripke structures in the obvious way. The *linear time model checking problem* is as follows: given a finite (directed) Kripke structure $\mathcal{M} = (W, R, V)$, a world $w \in W$, and a (directed) linear time formula φ, is there a (directed) path π in $\mathcal{M}$, starting at w, such that $\pi, 0 \models \varphi$? The *branching time model checking problem* is as follows: given a finite (directed) Kripke structure $\mathcal{M} = (W, R, V)$, a world $w \in W$, and a (directed) branching time formula φ, does $\mathcal{M}, w \models \varphi$? Note that, technically speaking, the linear time model checking problem is a *satisfiability* problem with respect to a given class of structures (the paths of a Kripke structure). It is well-known that:

Theorem 2.4.13 (*Complexity of model checking*)

1. *The model checking problem for* PLTL *is PSPACE-complete* [111]*;*
2. *the model checking problem for* CTL* *is PSPACE-complete* [20]*;*
3. *the model checking problem for* CTL *is in P* [19].

In particular, given a model of size n and a formula of length k, model checking in PLTL and in CTL* can be performed in time $n \cdot 2^{O(k)}$ and space $O(k \cdot (k + \log n)^2)$ [79], and model checking in CTL can be done in time $O(k \cdot n)$ [18, 106] and space $O(k \cdot \log^2(k \cdot n))$ [79].

Moving to quantified temporal logics (with Kripke structure semantics), we have the following results:

Theorem 2.4.14 (*Complexity of model checking for quantified temporal logics*)

1. *The model checking problem for* EQLTL *is PSPACE-complete* [77]*;*
2. *the model checking problem for* EQCTL* *is PSPACE-complete* [77]*;*
3. *the model checking problem for* EQCTL *is NP-complete* [77].

Taking advantage of the embedding of PLTL$_k$ into PLTL described above, we can easily prove the following results for directed temporal logics.

Theorem 2.4.15 (*Complexity of model checking for directed temporal logic*)

1. *The model checking problem for* PLTL$_k$ *is PSPACE-complete;*
2. *the model checking problem for* $\mathrm{CTL}^*_{\mathrm{k}}$ *is PSPACE-complete;*
3. *the model checking problem for* $\mathrm{EQCTL}^*_{\mathrm{k}}$ *is PSPACE-complete.*

3 The combining approach

Logic combination is emerging as a relevant research topic at the intersection of mathematical logic with computer science [56]. It essentially provides a logical account of traditional computer science notions such as modularity and abstraction. When dealing with real-world systems, organizing their descriptive and inferential requirements in a structured way is often the only way to master the complexity of the design, verification, and maintenance tasks. Formulated in the setting of combined logics, the basic issue underlying such an approach is: how can we guarantee that the logical properties of the component logics, such as axiomatic completeness and decidability, are inherited by the combined one? This issue is known as the *transfer problem*. It has a natural analogue in terms of the associated methods and tools: can we reuse methods and tools developed for the component logics, such as deductive engines and model checkers, to obtain methods and tools for the combined one? In general, the answer depends on the amount of interaction among components: the transfer generally succeeds in the absence of interaction among the component logics. Such positive result is usually based on a *divide and conquer* strategy: split problems into sub-problems and delegate these to the components [41, 76]. However, the transfer can easily fail when the components are allowed to interact [45, 57].

In this chapter we describe the combining approach for temporal logics and we propose a similar approach for finite-state automata. In Section 3.1 we introduce three well-known modes for combining temporal logics: temporalization, independent combination and join. In Section 3.2 we propose an automata-theoretic counterpart of temporalized logics. Finally, in Section 3.3 we study the model checking problem for combined logics. In Chapter 4 we will use the results presented in the present chapter to provide monadic theories of time granularity with temporal logic and an automata-theoretic counterparts.

3.1 Combining methods

Various forms of logic combination have been proposed in the literature. Temporalization, independent combination (or fusion), and join (or product) are probably the most popular ones as well as the ones that have been studied most extensively [41, 44, 45, 54, 57, 76, 112]. They have been successfully applied in several areas, including databases [42, 43, 46], artificial intelligence [37, 39, 64, 91, 3, 131], and system specification and verification [51]. We are mainly interested in this last application of combined logics. In the following, we introduce syntax and semantics for temporalization, independent combination and join.

We will use the following general definition of temporal logic. The language of *temporal logic* is based on a set $\mathcal{P} = \{P, Q, \ldots\}$ of proposition letters and extends that of propositional logic with a set $OP = \{\mathbf{O}_1^{i_1}, \ldots, \mathbf{O}_n^{i_n}\}$ of *temporal operators* with arities $i_1, \ldots, i_n$, respectively. The language of temporal logic is the smallest set of formulas generated by the following rules:

(P1) every proposition letter $P \in \mathcal{P}$ is a formula;

(P2) if ϕ, ψ are formulas, then $\phi \wedge \psi$ and $\neg\phi$ are formulas;

(P3) if $\mathbf{O}_j^{i_j} \in OP$ and $\phi_1, \ldots, \phi_{i_j}$ are formulas, then $\mathbf{O}_j^{i_j}(\phi_1, \ldots, \phi_{i_j})$ is a formula.

Boolean connectives $\vee$, $\rightarrow$, $\leftrightarrow$ are defined as usual. Moreover, given $P \in \mathcal{P}$, `true` abbreviates $P \vee \neg P$ and `false` stands for $\neg$`true`. A *frame* for temporal logic is a pair $(W, \mathcal{R})$, where W is a set of *worlds*, or *states*, and $\mathcal{R}$ is a set of *accessibility relations* on W. We restrict ourselves to *binary* accessibility relations on W. A *model* for temporal logic is a Kripke structure $(W, \mathcal{R}, V)$, where $(W, \mathcal{R})$ is a frame and $V : W \rightarrow 2^{\mathcal{P}}$ is a *valuation function* mapping states into sets of proposition letters. The semantics of temporal logic extends that of propositional logic with clauses for the temporal operators in OP. For $n \geq 1$, *n-dimensional* temporal logics are interpreted at n-tuples of points (equivalently, n-dimensional temporal formulas can be embedded in classical logics with n-ary predicates). Examples of one-dimensional temporal logics has been provided in Section 2.4. Given an arbitrary logic $\mathbf{L}$, we use $\mathcal{L}_{\mathbf{L}}$ and $\mathsf{K}_{\mathbf{L}}$ to denote the language and the set of models of $\mathbf{L}$, respectively. We write $OP(\mathbf{L})$ to denote the set of operators of $\mathbf{L}$ different from Boolean ones.

Temporalization is the simplest of the three modes of combining logics that we will consider; here, the two component languages are only allowed to interact in a very restricted way [44]. Let $\mathbf{T}$ be a temporal logic and $\mathbf{L}$ an arbitrary logic. For simplicity we constrain $\mathbf{L}$ to be an extension of propositional logic. We partition the set of $\mathbf{L}$-formulas into *Boolean combinations* $BC_{\mathbf{L}}$ and *monolithic formulas* $ML_{\mathbf{L}}$: α belongs to $BC_{\mathbf{L}}$ if its outermost operator is a Boolean connective; otherwise it belongs to $ML_{\mathbf{L}}$. We assume that $OP(\mathbf{T}) \cap OP(\mathbf{L}) = \emptyset$.

Definition 3.1.1 (*Temporalization – Syntax*)

The language $\mathcal{L}_{\mathbf{T}(\mathbf{L})}$ of the temporalization $\mathbf{T}(\mathbf{L})$ *of* $\mathbf{L}$ *by means of* $\mathbf{T}$ *over the set of proposition letters $\mathcal{P}$ is obtained by replacing the following atomic formation rule of $\mathcal{L}_{\mathbf{T}}$:*

Every proposition letter $P \in \mathcal{P}$ is a formula,

by the following rule:

Every monolithic formula $\alpha \in \mathcal{L}_{\mathbf{L}}$ is a formula.

□

A *model* for $\mathbf{T}(\mathbf{L})$ is a triple $(W, \mathcal{R}, g)$, where $(W, \mathcal{R})$ is a frame for $\mathbf{T}$ and $g : W \rightarrow \mathsf{K}_{\mathbf{L}}$ a total function mapping worlds in W to models for $\mathbf{L}$.

Definition 3.1.2 (*Temporalization – Semantics*)

Given a model $\mathcal{M} = (W, \mathcal{R}, g)$ and a state $w \in W$, the semantics of the temporalized logic $\mathbf{T}(\mathbf{L})$ *is obtained by replacing the following semantic clause for proposition letters of* $\mathbf{T}$*:*

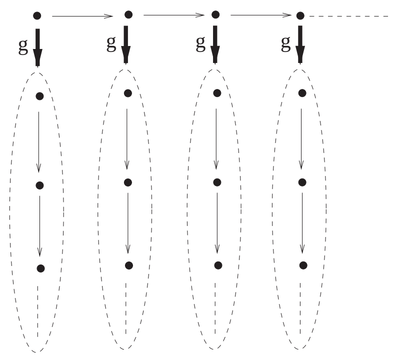

Figure 3.1: A temporalized model for PLTL(PLTL).

$$\mathcal{M}, w \models P \textit{ iff } P \in V(w), \textit{ whenever } P \in \mathcal{P},$$

by the following clause:

$$\mathcal{M}, w \models \alpha \textit{ iff } g(w) \models_{\mathbf{L}} \alpha, \textit{ whenever } \alpha \in ML_{\mathbf{L}}.$$

□

For instance, we consider the temporalization of PLTL$_2$ by means of PLTL$_1$, where, for $i \in \{1,2\}$, PLTL$_i$ is the propositional linear temporal logic PLTL over $\mathcal{P}$ where the temporal operators $\mathbf{X}$ and $\mathbf{U}$ has been renamed as $\mathbf{X}_i$ and $\mathbf{U}_i$, respectively. The language of PLTL$_1$(PLTL$_2$) is the smallest set of formulas generated by the following rules:

(P1) any monolithic formula in ML_{PLTL_2} is a formula;

(P2) if p, q are formulas, then $p \wedge q$ and $\neg p$ are formulas;

(P3) if p, q are formulas, then $\mathbf{X}_1 p$ and $p\mathbf{U}_1 q$ are formulas.

For instance, $\mathbf{X}_1\mathbf{X}_2 P$ is a PLTL$_1$(PLTL$_2$)-formula, but $\mathbf{X}_2\mathbf{X}_1 P$ is not. A temporalized model for PLTL$_1$(PLTL$_2$) is a triple $(\mathbb{N}, <, g)$, where g maps natural numbers into $\mathcal{P}$-labeled infinite sequences. An unlabeled model for PLTL$_1$(PLTL$_2$) is depicted in Figure 3.1. For $i \in \{1,2\}$, let $\models_i$ be the semantic relation of PLTL$_i$. Let $\mathcal{M} = (\mathbb{N}, <, g)$ be a temporalized model for PLTL$_1$(PLTL$_2$) and $i \in \mathbb{N}$. The semantic relation of PLTL$_1$(PLTL$_2$), denoted by $\models_{1(2)}$, is defined as follows:

$$\begin{array}{lll}
\mathcal{M}, i \models_{1(2)} \alpha & \text{iff} & g(i), 0 \models_2 \alpha \text{ whenever } \alpha \in ML_{\text{PLTL}_2} \\
\mathcal{M}, i \models_{1(2)} \phi \wedge \psi & \text{iff} & \mathcal{M}, i \models_{1(2)} \phi \text{ and } \mathcal{M}, i \models_{1(2)} \psi \\
\mathcal{M}, i \models_{1(2)} \neg\phi & \text{iff} & \text{it is not the case that } \mathcal{M}, i \models_{1(2)} \phi \\
\mathcal{M}, i \models_{1(2)} \phi\mathbf{U}_1\psi & \text{iff} & \mathcal{M}, j \models_{1(2)} \psi \text{ for some } j \geq i \text{ and} \\
 & & \mathcal{M}, k \models_{1(2)} \phi \text{ for every } i \leq k < j; \\
\mathcal{M}, i \models_{1(2)} \mathbf{X}_1\psi & \text{iff} & \mathcal{M}, i+1 \models_{1(2)} \psi.
\end{array}$$

The *independent combination* of two logics puts together all the expressive power of the two component logics in an unrestricted way [45]. Let $\mathbf{T_1}$ and $\mathbf{T_2}$ be two temporal logics defined over the same set of proposition letters $\mathcal{P}$, with $OP(\mathbf{T_1}) \cap OP(\mathbf{T_2}) = \emptyset$.

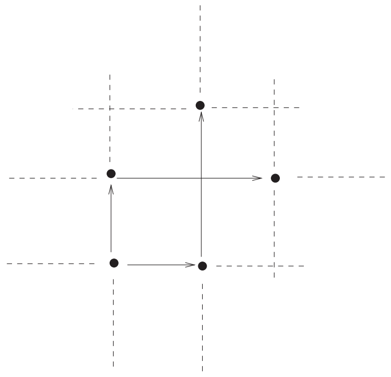

Figure 3.2: An independently combined model for PLTL $\oplus$ PLTL.

Definition 3.1.3 (*Independent Combination – Syntax*)

The language $\mathcal{L}_{\mathbf{T_1} \oplus \mathbf{T_2}}$ *of the* independent combination $\mathbf{T_1} \oplus \mathbf{T_2}$ *of* $\mathbf{T_1}$ *and* $\mathbf{T_2}$ *over* $\mathcal{P}$ *is obtained by taking the union of the formation rules for* $\mathbf{T_1}$ *and* $\mathbf{T_2}$*.* $\square$

To define the semantics of $\mathbf{T_1} \oplus \mathbf{T_2}$, we need the following notion. Given a binary relation R, we write R^* for its transitive closure, and R^{-1} for its converse. Let $(W, \mathcal{R})$ be a frame. A *connected component* $(W', \mathcal{R}')$ of $(W, \mathcal{R})$ is a frame with (i) $\emptyset \neq W' \subseteq W$ and $\mathcal{R}' = \{R|_{W'} \mid R \in \mathcal{R}\}$, (ii) $(W', \mathcal{R}')$ is *connected*, i.e., for every u and v in W', with $u \neq v$, we have $(u, v) \in [\bigcup\{(R \cup R^{-1}) \mid R \in \mathcal{R}\}]^*$, and (iii) $(W', \mathcal{R}')$ is *maximal*, i.e., there is no connected component $(W'', \mathcal{R}'')$ with $W' \subset W''$. Notice that an *isolated* point is a connected component. A *model* for the combined logic $\mathbf{T_1} \oplus \mathbf{T_2}$ is a 4-tuple $(W, \mathcal{R}_1, \mathcal{R}_2, V)$, where the connected components of $(W, \mathcal{R}_1, V)$ are in $\mathsf{K}_{\mathbf{T_1}}$ (models for $\mathbf{T_1}$) , the connected components of $(W, \mathcal{R}_2, V)$ are in $\mathsf{K}_{\mathbf{T_2}}$ (models for $\mathbf{T_2}$), and W is the (not necessarily disjoint) union of the sets of worlds that constitute each connected component. Finally, $V : W \rightarrow 2^{\mathcal{P}}$ is a valuation function.

Definition 3.1.4 (*Independent Combination – Semantics*)

The semantics of the independently combined logic $\mathbf{T_1} \oplus \mathbf{T_2}$ *is obtained by taking the union of the semantic clauses for* $\mathbf{T_1}$ *and* $\mathbf{T_2}$*.* $\square$

As an example, we consider the independent combination of PLTL$_1$ and PLTL$_2$ over $\mathcal{P}$. The language of PLTL$_1 \oplus$ PLTL$_2$ is the smallest set of formulas generated by the following rules:

(P1) any proposition letter $P \in \mathcal{P}$ is a formula;

(P2) if p, q are formulas, then $p \wedge q$ and $\neg p$ are formulas;

(P3) if p, q are formulas, then $\mathbf{X}_1 p$, $\mathbf{X}_2 p$, $p\mathbf{U}_1 q$, and $p\mathbf{U}_2 q$ are formulas.

Note that both $\mathbf{X}_1\mathbf{X}_2 P$ and $\mathbf{X}_2\mathbf{X}_1 P$ are PLTL$_1 \oplus$ PLTL$_2$-formulas. An independently combined model for PLTL$_1 \oplus$ PLTL$_2$ is a quadruple $(W, <_1, <_2, V)$, where the connected components of $(W, <_1, V)$ and those of $(W, <_2, V)$ are $\mathcal{P}$-labeled infinite sequences. An unlabeled model for PLTL$_1 \oplus$ PLTL$_2$ is depicted in Figure 3.2. We define two binary predicates $\mathtt{succ}_1$ and $\mathtt{succ}_2$ over W such that $\mathtt{succ}_1(w, v)$ iff $w <_1 v$ and there is no $z \in W$ such that $w <_1 z <_1 v$, and similarly for $\texttt{succ}_2(w, v)$. Let $\mathcal{M} = (W, <_1, <_2, V)$ be a model for PLTL$_1 \oplus$ PLTL$_2$ and $w \in W$. The semantics of PLTL$_1 \oplus$ PLTL$_2$ is the following:

$$\begin{array}{lll}
\mathcal{M}, w \models P & \text{iff} & P \in V(w) \text{ whenever } P \in \mathcal{P} \\
\mathcal{M}, w \models \phi \wedge \psi & \text{iff} & \mathcal{M}, w \models \phi \text{ and } \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \neg\phi & \text{iff} & \text{it is not the case that } \mathcal{M}, w \models \phi \\
\mathcal{M}, w \models \phi \mathbf{U}_1 \psi & \text{iff} & \mathcal{M}, v \models \psi \text{ for some } w \leq_1 v \text{ and} \\
 & & \mathcal{M}, z \models \phi \text{ for every } w \leq_1 z <_1 v; \\
\mathcal{M}, w \models \mathbf{X}_1 \psi & \text{iff} & \mathcal{M}, v \models \psi \text{ and } \texttt{succ}_1(w, v); \\
\mathcal{M}, w \models \phi \mathbf{U}_2 \psi & \text{iff} & \mathcal{M}, v \models \psi \text{ for some } w \leq_2 v \text{ and} \\
 & & \mathcal{M}, z \models \phi \text{ for every } w \leq_2 z <_2 v; \\
\mathcal{M}, w \models \mathbf{X}_2 \psi & \text{iff} & \mathcal{M}, v \models \psi \text{ and } \texttt{succ}_2(w, v).
\end{array}$$

It is worth noting that the formula $\mathbf{X}_1\mathbf{X}_2 P \leftrightarrow \mathbf{X}_2\mathbf{X}_1 P$, that allows to commute the two successor operators, is not *valid* in PLTL$_1 \oplus$PLTL$_2$, since, given a model $\mathcal{M} = (W, <_1, <_2, V)$ and $w \in W$, $\texttt{succ}_2(\texttt{succ}_1(w))$ is not necessarily equal to $\texttt{succ}_1(\texttt{succ}_2(w))$.

One may be tempted to view independent combination as an arbitrarily nesting of temporalizations. Let $L_0 = \mathbf{T_1}$, $\overline{L}_0 = \mathbf{T_2}$ and, for $i > 0$, let $L_i = \mathbf{T_1}(\overline{L}_{i-1})$ and $\overline{L}_i = \mathbf{T_2}(L_{i-1})$. Any formula in the independent combination $\mathbf{T_1} \oplus \mathbf{T_2}$ is a formula in a temporalized logic L_i, for some $i \geq 0$. Hence, one may be tempted to reduce the satisfiability/model checking problems for independent combination to the same problem for temporalization. However, this reduction would be erroneous. Informally, the reason is that there is a notable difference between the semantics of independent combination and that of (nested) temporalization. A model for independent combination is a *flat* structure, that is, all the (connected) components have the same nesting level. More importantly, a formula may 'visit' a component, leave it for a while, and later come back. This is not possible in temporalization. A model for temporalization is a *nested* structure: different components may have different nesting levels. Once a formula has 'entered' a component, it may only proceed forward, visiting the reached component or a component with a greater nesting level, but may not come back. More concretely, consider the following example. Consider PLTL over $\mathcal{P} = \{P\}$ interpreted over finite sequences. Take the PLTL$_1$(PLTL$_2$(PLTL$_1$))-formula $\varphi = \mathbf{F_1}(\mathbf{G_2}\mathbf{G_1}\neg P \wedge \mathbf{F_1} P)$ (as usual, to avoid confusion, we renamed temporal operators). We show that φ is satisfiable in PLTL$_1$(PLTL$_2$(PLTL$_1$)) but it is unsatisfiable in PLTL$_1 \oplus$ PLTL$_2$. We construct a PLTL$_1$(PLTL$_2$(PLTL$_1$))-model in which φ is true. Let $\mathcal{M}_P$ be a PLTL-model consisting of a single point labeled with letter P, and let $\mathcal{M}_{\neg P}$ be a PLTL-model consisting of a single point labeled with no letter. Let $\mathcal{M}_1 = (\{0\}, \emptyset, g_1)$ be a PLTL(PLTL) model such that $g_1(0) = \mathcal{M}_{\neg P}$, and let $\mathcal{M}_2 = (\{0\}, <, g_2)$ be a PLTL(PLTL) model such that $g_2(0) = \mathcal{M}_P$. Finally, let $\mathcal{M} = (\{0, 1\}, <, g)$ be a PLTL(PLTL(PLTL))-model such that $g(0) = \mathcal{M}_1$ and $g(1) = \mathcal{M}_2$. It is immediate to see that $\mathcal{M}$ is a model of φ. However, there is no model for independent combination PLTL$_1 \oplus$ PLTL$_2$ that satisfies φ. Indeed, suppose $\mathcal{M} = (W, <_1, <_2, V)$ is a model, $w \in W$, and $\mathcal{M}, w \models \varphi$. Hence, there is v such that $w \leq_1 v$ and $\mathcal{M}, v \models \mathbf{G_2}\mathbf{G_1}\neg P \wedge \mathbf{F_1} P$, that is, $\mathcal{M}, v \models \mathbf{G_2}\mathbf{G_1}\neg P$ and $\mathcal{M}, v \models \mathbf{F_1} P$. Since $\mathcal{M}, v \models \mathbf{G_2}\mathbf{G_1}\neg P$, for every z such that $v \leq_2 z$, we have $\mathcal{M}, z \models \mathbf{G_1}\neg P$. In particular, we have that $\mathcal{M}, v \models \mathbf{G_1}\neg P$, which is in contradiction with $\mathcal{M}, v \models \mathbf{F_1} P$.

The temporalization and the independent combination of two n-dimensional temporal logics is an n-dimensional temporal logic. For instance, formulas in the language of PLTL$_1$(PLTL$_2$) and PLTL$_1 \oplus$PLTL$_2$ are still evaluated at a single node in a model. The combining method of *join*, instead, produces higher-dimensional temporal logics by combining lower-dimensional

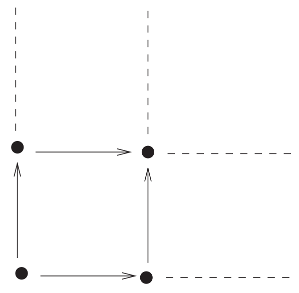

Figure 3.3: A joined model for PLTL $\otimes$ PLTL.

ones. For notational simplicity we assume that our component logics are one-dimensional. Let $\mathbf{T_1}$ and $\mathbf{T_2}$ be two (one-dimensional) temporal logics.

Definition 3.1.5 (*Join – Syntax*)

The language $\mathcal{L}_{\mathbf{T_1}\otimes\mathbf{T_2}}$ of the join $\mathbf{T_1}\otimes\mathbf{T_2}$ *of $\mathbf{T_1}$ and $\mathbf{T_2}$ over $\mathcal{P}$ is obtained by taking and the union of the formation rules for $\mathbf{T_1}$ and $\mathbf{T_2}$.* □

Note that the language of join coincides that of independent combination. However, the semantics is different. A *model* for $\mathbf{T_1}\otimes\mathbf{T_2}$ is a 5-tuple $(W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$, where $(W_1, \mathcal{R}_1)$ is a $\mathbf{T_1}$-frame and $(W_2, \mathcal{R}_2)$ is a $\mathbf{T_2}$-frame, and $V : W_1 \times W_2 \to 2^{\mathcal{P}}$ is a valuation mapping *pairs* of worlds to sets of proposition letters. Given a formula φ on the language of $\mathbf{T_1}\otimes\mathbf{T_2}$, we denote by $\varphi^{\uparrow}$ the $\mathbf{T_1}$-formula obtained from φ by replacing every maximal subformula α starting with a $\mathbf{T_2}$-operator with proposition letter P_α.

Definition 3.1.6 (*Join – Semantics*)

Truth of a formula φ in a model $\mathcal{M} = (W_1,\ \mathcal{R}_1,\ W_2,\ \mathcal{R}_2,\ V)$, at states $s_1 \in W_1$ and $s_2 \in W_2$, is defined as follows. If $\varphi = P$, with $P \in \mathcal{P}$, $\varphi = (\varphi_1 \wedge \varphi_2)$, or $\varphi = \neg\varphi_1$, then $\mathcal{M}, s_1, s_2 \models_{\mathbf{T_1}\otimes\mathbf{T_2}} \varphi$ is defined as usual. If $\varphi = \mathbf{O}(\varphi_1, \ldots, \varphi_n)$, with $\mathbf{O} \in OP(\mathbf{T_1})$, we define $\mathcal{M}, s_1, s_2 \models_{\mathbf{T_1}\otimes\mathbf{T_2}} \varphi$ by replacing every occurrence of $\mathcal{M}, x$ in the definition of $\mathcal{M}, s_1 \models_{\mathbf{T_1}} \varphi^{\uparrow}$ by $\mathcal{M}, x, s_2$. Similarly if $\mathbf{O} \in OP(\mathbf{T_2})$. □

In our presentation of the join of logics, we have followed [45]; in [57] a slightly different, but equivalent, construction is studied: the *product* of modal logics. As an example, we consider the join of PLTL$_1$ and PLTL$_2$ over $\mathcal{P}$. The language of PLTL$_1 \otimes$ PLTL$_2$ is as for PLTL$_1 \oplus$ PLTL$_2$. A joined model for PLTL$_1 \otimes$ PLTL$_2$ is a quintuple $(\mathbb{N}, <, \mathbb{N}, <, V)$. An unlabeled model for PLTL$_1 \otimes$ PLTL$_2$ is depicted in Figure 3.3. Let $\mathcal{M} = (\mathbb{N}, <, \mathbb{N}, <, V)$ be a model for PLTL$_1 \otimes$ PLTL$_2$, and $i, j \in \mathbb{N}$. The semantics of PLTL$_1 \otimes$ PLTL$_2$ is as follows.

$\mathcal{M}, i, j \models P$	iff	$P \in V(i,j)$ whenever $P \in \mathcal{P}$
$\mathcal{M}, i, j \models \phi \wedge \psi$	iff	$\mathcal{M}, i, j \models \phi$ and $\mathcal{M}, i, j \models \psi$
$\mathcal{M}, i, j \models \neg\phi$	iff	it is not the case that $\mathcal{M}, i, j \models \phi$
$\mathcal{M}, i, j \models \phi\mathbf{U}_1\psi$	iff	$\mathcal{M}, r, j \models \psi$ for some $i \leq r$ and $\mathcal{M}, k, j \models \phi$ for every $i \leq k < r$;
$\mathcal{M}, i, j \models \mathbf{X}_1\psi$	iff	$\mathcal{M}, i+1, j \models \psi$;
$\mathcal{M}, i, j \models \phi\mathbf{U}_2\psi$	iff	$\mathcal{M}, i, r \models \psi$ for some $j \leq r$ and $\mathcal{M}, i, k \models \phi$ for every $j \leq k < r$;
$\mathcal{M}, i, j \models \mathbf{X}_2\psi$	iff	$\mathcal{M}, i, j+1 \models \psi$;

It is easy to see that the formula $\mathbf{X}_1\mathbf{X}_2P \leftrightarrow \mathbf{X}_2\mathbf{X}_1P$, that allows to commute the two successor operators, is *valid* in $\mathrm{PLTL}_1 \otimes \mathrm{PLTL}_2$.

We summarize the main transfer results present in the literature for the combining methods introduced above. The independent combination of two decidable normal polyadic polymodal modal logics is decidable as well [129]. The same result holds for modal logics with the *converse* operator interpreted over transitive frames [126, 127, 128]. Moreover, properties of finite axiomatizability, soundness and completeness transfer through the independent combination of monomodal logics [41, 76]. As for 'Since and Until logics', it is known that PPLTL(PPLTL) and PPLTL$\oplus$PPLTL are decidable, and admit a sound and complete finite axiomatization [44, 45]. In the case of join, things get much harder. For instance, the modal logic $S5$ is NP-complete, $S5^2$ (that is $S5 \otimes S5$) is NEXPTIME-complete [89] and $S5^3$ is undecidable (and does not even have the finite model property) [81]. Moreover, $\mathrm{PLTL} \otimes K_m$ is decidable, but $\mathrm{PLTL} \otimes \mathrm{PLTL}$ is not even recursively enumerable [130].

3.2 Automata for combined temporal logics

The goal of this section is to provide combined temporal logics with an automata-theoretic counterpart. Such an equivalent characterization of combined logics as combined automata presents several advantages for automated system specification and verification.

First, turning a formula of (combined) temporal logic into an automaton allows us to give a uniform representation of systems and specifications as automata. The problem of establishing whether a system P behaves according to a given specification ϕ (model checking problem) can then be reduced to the language containment problem $\mathcal{L}(\mathcal{A}_P) \subseteq \mathcal{L}(\mathcal{A}_\phi)$, where $\mathcal{L}(\mathcal{A}_P)$ is the language recognized by the automaton $\mathcal{A}_P$, consisting of all and only the behaviors of P, and $\mathcal{L}(\mathcal{A}_\phi)$ is the language recognized by the automaton $\mathcal{A}_\phi$, consisting of all and only the models of ϕ [118]. Furthermore, if the considered class of automata is closed under Boolean operations, the language containment problem can be mapped into the emptiness one, that is, $\mathcal{L}(\mathcal{A}_P \cap \overline{\mathcal{A}_\phi}) = \emptyset$.

Second, (combined) automata can be directly used as a specification formalism, provided with a natural graphical interpretation [87]. Moreover, to avoid the costly complementation of the specification automaton $\mathcal{A}_\phi$, some model checkers constrain the user to directly provide the automaton $\mathcal{A}_{\neg\phi}$ for $\neg\phi$, that is, they constrain the user to specify the unacceptable behaviors of the system instead of the good ones [66, 67]. As an alternative, one can replace specification automata with a costly complementation by other, equally expressive, automata for which the complementation is much easier [80]. For instance, if the specification language is (quantified) linear temporal logic, we can replace Büchi automata by Muller automata.

Third, the specification automaton $\mathcal{A}_\phi$ can be used to cope with the *state explosion problem* by preventing the complete construction of the system automaton $\mathcal{A}_P$ from happening, whenever possible (on-the-fly model-checking) [24, 73, 117]. More precisely, some system states, which are incompatible with or irrelevant to the specification, may not be generated at all; furthermore, a counterexample for $\mathcal{A}_P \cap \overline{\mathcal{A}_\phi}$ can be detected before the completion of $\mathcal{A}_P$ construction, making such a completion no more necessary.

Finally, the automata-theoretic characterization of (combined) temporal logics can help in finding the temporal logic counterpart of monadic theories (cf. Figure 3.4). In many cases, this is a difficult task, involving a non-elementary blow up in the length of formulas. Ehrenfeucht games have been successfully exploited to deal with such a correspondence problem for first-order monadic theories [69] and well-behaved fragments of second-order ones, e.g. the path fragment of the monadic second-order theory of infinite binary trees [61].

$$\begin{tikzcd}
\mathbf{L} \arrow[rr, leftrightarrow, "?"] \arrow[dr, leftrightarrow] & & \mathbf{T} \arrow[dl, leftrightarrow] \\
& \mathbf{A} &
\end{tikzcd}$$

Figure 3.4: Monadic theories (L), temporal logics (T), and automata (A).

Unfortunately, these techniques do not naturally lift to the full second-order case. The existence of a correspondence between (combined) temporal logics and (combined) automata, satisfying the usual closure properties, allows one to reduce the task of finding a temporal logic counterpart of a (second-order) monadic theory to the often easier one of finding an automata counterpart of it. The mapping of monadic formulas into automata (the difficult direction) can indeed greatly benefit from automata closure properties.

In this thesis, we only focus on the case of temporalized logics (cf. Section 3.1). However, we believe that this is the first step towards a general automata-theoretic counterpart of combined temporal logics. We define a new class of combined automata, called *temporalized automata*, which can be viewed as the automata-theoretic counterpart of temporalized logics, and show that relevant properties, such as closure under Boolean operations, decidability, and expressive equivalence with respect to temporal logics, transfer from component automata to temporalized ones.

For the sake of simplicity, we first define automata and prove results over sequence structures; then, we generalize definitions and results to tree structures (we believe that our machinery can actually be extended to cope with more general structures, such as graphs). We will use the following general definition of sequence automata. Let $\mathcal{S}(\Sigma)$ be the set of Σ-labeled infinite sequences $(\mathbb{N}, <, V)$, with $V : \mathbb{N} \rightarrow \Sigma$.

Definition 3.2.1 (*Sequence automata*)

A sequence automaton *A over Σ consists of (i) a Labeled Transition System (LTS) $(Q, q_0, \Delta, M, \Omega)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, Ω is a finite alphabet, and $M \subseteq Q \times \Omega$ is a labeling of states, and (ii) an acceptance condition AC. Given a Σ-labeled infinite sequence $w = (\mathbb{N}, <, V)$, a* run *of A on w is function $\sigma : \mathbb{N} \rightarrow Q$ such that $\sigma(0) = q_0$ and $(\sigma(i), V(i), \sigma(i+1)) \in \Delta$, for every $i \geq 0$. The automaton A accepts w if there is a run σ of A on w such that $AC(\sigma)$, i.e., the acceptance condition holds on σ. The language accepted by A, denoted by $\mathcal{L}(A)$, is the set of Σ-labeled infinite sequences accepted by A.* □

A class of sequence automata $\mathcal{A}$ is a set of automata that share the acceptance condition AC. It is worth clarifying that we do not explicitly specify the acceptance condition for sequence automata since all the results do not rest on any particular acceptance condition.

Instances of sequence automata are Büchi and Rabin sequence automata (Definitions 2.3.2 and 2.3.3). In particular, a Büchi sequence automaton (Q, q_0, Δ, F) corresponds to a sequence automaton $(Q, q_0, \Delta, M, \Omega)$ such that $\Omega = \{\mathtt{final}\}$ and $M = \{(q, \mathtt{final}) \mid q \in F\}$ and a Rabin sequence automaton $(Q, q_0, \Delta, \{(L_1, U_1), \ldots, (L_m, U_m)\})$ corresponds to a sequence automaton $(Q, q_0, \Delta, M, \Omega)$ such that $\Omega = \{\mathtt{fin}_i, \mathtt{inf}_i \mid 1 \leq i \leq m\}$, $M = \bigcup_{i=1}^{m} \{(q, \mathtt{fin}_i) \mid q \in L_i\} \cup \{(q, \mathtt{inf}_i) \mid q \in U_i\}$.

In the following, we will always assume that $\mathcal{A}_2$ is a sequence automata class whose automata accept Σ-labeled infinite sequences in $\mathcal{S}(\Sigma)$. Moreover, let $\Gamma(\Sigma)$ be a finite alphabet whose symbols $A, B, \ldots$ represent automata in $\mathcal{A}_2$, and let $\mathcal{A}_1$ be a sequence automata class whose automata accept $\Gamma(\Sigma)$-labeled infinite sequences in $\mathcal{S}(\Gamma(\Sigma))$. Given $\mathcal{A}_1$ and $\mathcal{A}_2$ as above, we define a class of temporalized automata $\mathcal{A}_1(\mathcal{A}_2)$ that combines the two component automata classes in a suitable way. Let $\mathcal{S}(\mathcal{S}(\Sigma))$ be the set of infinite sequences of Σ-labeled infinite sequences, that is, temporalized models $(\mathbb{N}, <, g)$ where $g : \mathbb{N} \rightarrow \mathcal{S}(\Sigma)$ is a total function mapping worlds in $\mathbb{N}$ into Σ-labeled infinite sequences in $\mathcal{S}(\Sigma)$. Automata in the combined class $\mathcal{A}_1(\mathcal{A}_2)$ accept in $\mathcal{S}(\mathcal{S}(\Sigma))$.

Definition 3.2.2 (*Temporalized automata*)

A temporalized automaton *A over $\Gamma(\Sigma)$ is a quintuple $(Q, q_0, \Delta, M, \Omega)$ as for sequence automata (Definition 3.2.1). The* combined acceptance condition *for A is the following. Given $w = (\mathbb{N}, <, g) \in \mathcal{S}(\mathcal{S}(\Sigma))$, a* run *of A on w is function $\sigma : \mathbb{N} \rightarrow Q$ such that $\sigma(0) = q_0$ and, for every $i \geq 0$, $(\sigma(i), B, \sigma(i+1)) \in \Delta$ for some $B \in \Gamma(\Sigma)$ such that $g(i) \in \mathcal{L}(B)$. The automaton A accepts w if there is a run σ of A on w such that $AC(\sigma)$, where AC is the acceptance condition of $\mathcal{A}_1$. The language accepted by A, denoted by $\mathcal{L}(A)$, is the subset of $\mathcal{S}(\mathcal{S}(\Sigma))$ accepted by A. We denote by $\mathcal{A}_1(\mathcal{A}_2)$ the class of temporalized automata.* □

Given a temporalized automaton $A \in \mathcal{A}_1(\mathcal{A}_2)$, we denote by $A^{\uparrow}$ the automaton in $\mathcal{A}_1$ with the same labelling transition system as A and with the acceptance condition of $\mathcal{A}_1$. Hence, while A accepts in $\mathcal{S}(\mathcal{S}(\Sigma))$, its *abstraction* $A^{\uparrow}$ recognizes in $\mathcal{S}(\Gamma(\Sigma))$. Moreover, given an automaton $A \in \mathcal{A}_1$, we denote by $A^{\downarrow}$ the automaton in $\mathcal{A}_1(\mathcal{A}_2)$ with the same labelling transition system as A and with the combined acceptance condition of $\mathcal{A}_1(\mathcal{A}_2)$. Thus, while A accepts in $\mathcal{S}(\Gamma(\Sigma))$, its *concretization* $A^{\downarrow}$ recognizes in $\mathcal{S}(\mathcal{S}(\Sigma))$. With the aid of these notations, the combined acceptance condition for temporalized automata can be rewritten as follows: let $w = (\mathbb{N}, <, g) \in \mathcal{S}(\mathcal{S}(\Sigma))$. We say that a temporalized automaton A accepts w if and only if there is $v = (\mathbb{N}, <, V) \in \mathcal{S}(\Gamma(\Sigma))$ such that $v \in \mathcal{L}(A^{\uparrow})$ and, for every $i \in \mathbb{N}$, $g(i) \in \mathcal{L}(V(i))$. We will often use this alternative but equivalent definition of acceptance condition for temporalized automata.

We define the *transfer problem* for temporalized automata as follows: assuming that automata classes $\mathcal{A}_1$ and $\mathcal{A}_2$ enjoy some property, does $\mathcal{A}_1(\mathcal{A}_2)$ enjoy the same property? Recall that $\mathcal{A} \leftrightarrows \mathbf{T}$ denotes the fact that the automata class $\mathcal{A}$ is expressively equivalent to the temporal logic $\mathbf{T}$. We investigate the transfer problem with respect to the following properties of automata:

1. (Effective) *closure* under Boolean operations (union, intersection, and complementation): if $\mathcal{A}_1$ and $\mathcal{A}_2$ are (effectively) closed under Boolean operations, is $\mathcal{A}_1(\mathcal{A}_2)$ (effectively) closed under Boolean operations?

2. *Decidability*: if $\mathcal{A}_1$ and $\mathcal{A}_2$ are decidable, is $\mathcal{A}_1(\mathcal{A}_2)$ decidable?

3. *Expressive equivalence* with respect to temporal logic: if $\mathcal{A}_1 \leftrightarrows \mathbf{T_1}$ and $\mathcal{A}_2 \leftrightarrows \mathbf{T_2}$, does $\mathcal{A}_1(\mathcal{A}_2) \leftrightarrows \mathbf{T_1}(\mathbf{T_2})$?

The following lemma is crucial in the rest of this section. It shows that every temporalized automaton is equivalent to another temporalized automaton whose transitions are labeled with automata that form a partition of the set of $\mathcal{S}(\Sigma)$ of Σ-labeled sequences. Hence, *different* labels in the 'partitioned automaton' correspond to (automata accepting) *disjoint* sets of Σ-labeled sequences. Moreover, the partitioned automaton can be effectively constructed from the original one. A similar partition lemma holds for temporalized logics too (cf. Lemma 3.2.6 below).

Lemma 3.2.3 (*Partition Lemma for temporalized automata*)

Let A be a temporalized automaton in $\mathcal{A}_1(\mathcal{A}_2)$. If $\mathcal{A}_2$ is closed under Boolean operations (union, intersection, and complementation), then there exists a finite alphabet $\Gamma'(\Sigma) \subseteq \mathcal{A}_2$ and a temporalized automaton A' over $\Gamma'(\Sigma)$ such that $\mathcal{L}(A) = \mathcal{L}(A')$ and $\{\mathcal{L}(X) \mid X \in \Gamma'(\Sigma)\}$ forms a partition of $\mathcal{S}(\Sigma)$. Moreover, if $\mathcal{A}_2$ is effectively closed under Boolean operations and it is decidable, then A' can be effectively computed from A.

Proof.

To construct $\Gamma'(\Sigma)$ and $\mathcal{A}'$ we proceed as follows. Suppose $A = (Q, q_0, \Delta, M, \Omega)$, over $\Gamma(\Sigma) = \{X_1, \ldots, X_n\} \subseteq \mathcal{A}_2$. For every $1 \leq i \leq n$ and $j \in \{0,1\}$, let $X_i^j = X_i$ if $j = 0$, and $X_i^j = \mathcal{S}(\Sigma) \setminus X_i$ if $j = 1$. Given $(j_1, \ldots, j_n) \in \{0,1\}^n$, let $\texttt{Cap}_{(j_1,\ldots,j_n)} = \bigcap_{i=1}^n X_i^{j_i}$. We define $\Gamma_1(\Sigma)$ as the set of all and only $\texttt{Cap}_{(j_1,\ldots,j_n)}$ such that $(j_1, \ldots, j_n) \in \{0,1\}^n$. Since $\mathcal{A}_2$ is closed under Boolean operations, $\Gamma_1(\Sigma) \subseteq \mathcal{A}_2$. Moreover, let $\Gamma_2(\Sigma) = \{X \in \Gamma_1(\Sigma) \mid \mathcal{L}(X) \neq \emptyset\}$. We set $\Gamma'(\Sigma) = \Gamma_2(\Sigma)$, and, for $1 \leq i \leq n$, $\Gamma'_i(\Sigma) = \{X \in \Gamma'(\Sigma) \mid X \cap X_i \neq \emptyset\}$. Note that $\{\mathcal{L}(X) \mid X \in \Gamma'(\Sigma)\}$ forms a partition of $\mathcal{S}(\Sigma)$. Moreover, for every $1 \leq i \leq n$, $\{\mathcal{L}(X) \mid X \in \Gamma'_i(\Sigma)\}$ forms a partition of $\mathcal{L}(X_i)$. We define $A' = (Q, q_0, \Delta', M, \Omega)$ over $\Gamma'(\Sigma)$, where Δ' contains all and only the triples $(q_1, X, q_2) \in Q \times \Gamma'(\Sigma) \times Q$ such that $X \in \Gamma'_i(\Sigma)$ and $(q_1, X_i, q_2) \in \Delta$ for some $1 \leq i \leq n$. It is easy to see that $\mathcal{L}(A) = \mathcal{L}(A')$. ∎

We now prove the first transfer theorem: closure under Boolean operations transfers through temporalized automata.

Theorem 3.2.4 (*Transfer of closure under Boolean operations*)

Closure under Boolean operations (union, intersection, and complementation) transfers through temporalized automata: given two classes $\mathcal{A}_1$ and $\mathcal{A}_2$ of automata which are (effectively) closed under Boolean operations, the class $\mathcal{A}_1(\mathcal{A}_2)$ of temporalized automata is (effectively) closed under Boolean operations.

Proof.

Let $X, Y \in \mathcal{A}_1(\mathcal{A}_2)$.

Union We must provide an automaton $A \in \mathcal{A}_1(\mathcal{A}_2)$ that recognizes the language $\mathcal{L}(X) \cup \mathcal{L}(Y)$. Define $A = (X^{\uparrow} \cup Y^{\uparrow})^{\downarrow}$. We show that $\mathcal{L}(A) = \mathcal{L}(X) \cup \mathcal{L}(Y)$. Let $x = (\mathbb{N}, <, g) \in \mathcal{L}(A)$. Hence, there is $y = (\mathbb{N}, <, V) \in \mathcal{L}(A^{\uparrow}) = \mathcal{L}(X^{\uparrow}) \cup \mathcal{L}(Y^{\uparrow})$ such that, for every $i \in \mathbb{N}$, $g(i) \in \mathcal{L}(V(i))$. Suppose $y \in \mathcal{L}(X^{\uparrow})$. It follows that $x \in \mathcal{L}(X)$. Hence $x \in \mathcal{L}(X) \cup \mathcal{L}(Y)$. Similarly if $y \in \mathcal{L}(Y^{\uparrow})$.

We now show the opposite direction. Suppose that $x = (\mathbb{N}, <, g) \in \mathcal{L}(X) \cup \mathcal{L}(Y)$. If $x \in \mathcal{L}(X)$, then there is $y = (\mathbb{N}, <, V) \in \mathcal{L}(X^{\uparrow})$ such that, for every $i \in \mathbb{N}$, $g(i) \in \mathcal{L}(V(i))$. Hence, $y \in \mathcal{L}(X^{\uparrow}) \cup \mathcal{L}(Y^{\uparrow}) = \mathcal{L}(X^{\uparrow} \cup Y^{\uparrow}) = \mathcal{L}(A^{\uparrow})$. It follows that $x \in \mathcal{L}(A)$. Similarly if $x \in \mathcal{L}(Y)$.

Complementation We must provide an automaton $A \in \mathcal{A}_1(\mathcal{A}_2)$ that recognizes the language $\mathcal{S}(\mathcal{S}(\Sigma)) \setminus \mathcal{L}(X)$. Given the Partition Lemma 3.2.3, we may assume that $\{\mathcal{L}(Z) \mid Z \in \Gamma(\Sigma)\}$ forms a partition of $\mathcal{S}(\Sigma)$. We define $A = (\mathcal{S}(\Gamma(\Sigma)) \setminus X^{\uparrow})^{\downarrow}$. We show that $\mathcal{L}(A) = \mathcal{S}(\mathcal{S}(\Sigma)) \setminus \mathcal{L}(X)$. Let $x = (\mathbb{N}, <, g) \in \mathcal{L}(A)$. Hence, there exists $y = (\mathbb{N}, <, V) \in \mathcal{L}(A^{\uparrow}) = \mathcal{S}(\Gamma(\Sigma)) \setminus \mathcal{L}(X^{\uparrow})$ such that, for every $i \in \mathbb{N}$, $g(i) \in \mathcal{L}(V(i))$. Suppose, by contradiction, that $x \in \mathcal{L}(X)$. It follows that there exists $z = (\mathbb{N}, <, V') \in \mathcal{L}(X^{\uparrow})$ such that, for every $i \in \mathbb{N}$, $g(i) \in \mathcal{L}(V'(i))$. Hence, for every $i \in \mathbb{N}$, $g(i) \in \mathcal{L}(V(i)) \cap \mathcal{L}(V'(i))$. Since, for every $i \in \mathbb{N}$, $\mathcal{L}(V(i)) \cap \mathcal{L}(V'(i)) = \emptyset$ whenever $V(i) \neq V'(i)$, we conclude that $V(i) = V'(i)$. Hence $V = V'$ and thus $y = z$. This is a contradiction since y and z belong to disjoint sets. It follows that $x \in \mathcal{S}(\mathcal{S}(\Sigma)) \setminus \mathcal{L}(X)$.

We now show the opposite direction. Let $x = (\mathbb{N}, <, g) \in \mathcal{S}(\mathcal{S}(\Sigma)) \setminus \mathcal{L}(X)$. It follows that, for every $y = (\mathbb{N}, <, V) \in \mathcal{L}(X^{\uparrow})$, there exists $i \in \mathbb{N}$ such that $g(i) \notin \mathcal{L}(V(i))$. Suppose, by contradiction, that $x \in \mathcal{S}(\mathcal{S}(\Sigma)) \setminus \mathcal{L}(A)$. It follows that, for every $z = (\mathbb{N}, <, V) \in \mathcal{L}(A^{\uparrow}) = \mathcal{S}(\Gamma(\Sigma)) \setminus \mathcal{L}(X^{\uparrow})$, there exists $i \in \mathbb{N}$ such that $g(i) \notin \mathcal{L}(V(i))$. We can conclude that, for every $v = (\mathbb{N}, <, V) \in \mathcal{S}(\Gamma(\Sigma))$, there exists $i \in \mathbb{N}$ such that $g(i) \notin \mathcal{L}(V(i))$. This is a contradiction: since $\{\mathcal{L}(Z) \mid Z \in \Gamma(\Sigma)\}$ forms a partition of $\mathcal{S}(\Sigma)$, for every $i \in \mathbb{N}$, there is $Y_i \in \Gamma(\Sigma)$ such that $g(i) \in \mathcal{L}(Y_i)$. We have that $(\mathbb{N}, <, V')$, with $V'(i) = Y_i$, is an element of $\mathcal{S}(\Gamma(\Sigma))$ and, for every $i \in \mathbb{N}$, $g(i) \in \mathcal{L}(V'(i))$. We conclude that $x \in \mathcal{L}(A)$.

Intersection Follows from closure under union and complementation using De Morgan's laws. ■

Note that, if $A = (X^{\uparrow} \cap Y^{\uparrow})^{\downarrow}$, then $\mathcal{L}(A) \subseteq \mathcal{L}(X) \cap \mathcal{L}(Y)$, but equivalence does not hold in general. Here is a counterexample of $\mathcal{L}(A) \supseteq \mathcal{L}(X) \cap \mathcal{L}(Y)$. Let $\Gamma(\Sigma) = \{A, B\}$, $X^{\uparrow}$ be the automata accepting sequences starting with symbol A and $Y^{\uparrow}$ be the automata accepting strings starting with symbol B. Then, $\mathcal{L}(X^{\uparrow} \cap Y^{\uparrow}) = \emptyset$ and hence $\mathcal{L}(A) = \emptyset$. Given $\Sigma = \{a, b\}$, let A be an automaton accepting sequences with an odd number of symbols a and B be an automaton recognizing sequences with a prime number of symbols b. Thus $\mathcal{L}(X) \cap \mathcal{L}(Y)$ contains a combined structure starting with a sequences with exactly 13 symbols a, and hence it is not empty.

We now investigate the transfer problem for automata with respect to the decidability property. Given $A \in \mathcal{A}_1(\mathcal{A}_2)$, it is easy to see that a sufficient condition for $\mathcal{L}(A) = \emptyset$ is that $\mathcal{L}(A^{\uparrow}) = \emptyset$. However, this condition is not necessary, since A may be labeled with some $\mathcal{A}_2$-automaton accepting the empty language. Nevertheless, if we know that A is labeled with $\mathcal{A}_2$-automata recognizing non-empty languages, then the condition $\mathcal{L}(A^{\uparrow}) = \emptyset$ is both necessary and sufficient for $\mathcal{L}(A) = \emptyset$. In the following theorem, we apply these considerations to devise an algorithm that checks emptiness of a temporalized automaton.

Theorem 3.2.5 (*Transfer of decidability*)

Decidability transfers through temporalized automata: given two decidable classes $\mathcal{A}_1$ and $\mathcal{A}_2$ of automata, the class $\mathcal{A}_1(\mathcal{A}_2)$ of temporalized automata is decidable.

Proof.

Let A be a temporalized automaton in $\mathcal{A}_1(\mathcal{A}_2)$. We describe an algorithm that returns 1 if $\mathcal{L}(A) = \emptyset$ and returns 0 otherwise.

Step 1 Check whether $\mathcal{L}(A^{\uparrow}) = \emptyset$ using the emptiness algorithm for $\mathcal{A}_1$. If $\mathcal{L}(A^{\uparrow}) = \emptyset$, then return 1.

Step 2 For every $X \in \Gamma(\Sigma)$, if $\mathcal{L}(X) = \emptyset$ (this check is performed exploiting the emptiness algorithm for $\mathcal{A}_2$), then remove every transition of the form (q_1, X, q_2) from the transition relation of A.

Step 3 Let B be the temporalized automaton resulting from A after Step 2. Check, using the emptiness algorithm for $\mathcal{A}_1$, whether $\mathcal{L}(B^{\uparrow}) = \emptyset$. If $\mathcal{L}(B^{\uparrow}) = \emptyset$, then return 1, else return 0.

Note that the above algorithm always terminates returning either 1 or 0. We prove that the algorithm returns 1 if and only if $\mathcal{L}(A) = \emptyset$. Suppose that the algorithm returns 1. If $\mathcal{L}(A^{\uparrow}) = \emptyset$, then $\mathcal{L}(A) = \emptyset$. Suppose now that $\mathcal{L}(A^{\uparrow}) \neq \emptyset$ and $\mathcal{L}(B^{\uparrow}) = \emptyset$. Note that $\mathcal{L}(A) = \mathcal{L}(B)$, since B is obtained from A by cutting off automata accepting the empty language. Assume, by contradiction, that there is $x \in \mathcal{L}(A)$. Since $\mathcal{L}(A) = \mathcal{L}(B)$, we have that $x \in \mathcal{L}(B)$. Hence $\mathcal{L}(B)$ in not empty. Since $\mathcal{L}(B^{\uparrow}) = \emptyset$, we have that $\mathcal{L}(B)$ is empty. A contradiction. Hence $\mathcal{L}(A) = \emptyset$.

Suppose now that the algorithm returns 0. Then $\mathcal{L}(B^{\uparrow})$ contains at least one element, say $x = (\mathbb{N}, <, V)$. Since B uses only non-empty $\mathcal{A}_2$-automata as alphabet symbols, we have that, for every $i \in \mathbb{N}$, $\mathcal{L}(V(i)) \neq \emptyset$. Hence $y = (\mathbb{N}, <, g)$, with g such that, for every $i \in \mathbb{N}$, $g(i)$ equals to some element of $\mathcal{L}(V(i))$, is an element of $\mathcal{L}(A)$. Hence $\mathcal{L}(A) \neq \emptyset$ ■

Finally, we investigate the transfer of expressive equivalence with respect to temporal logics through temporalized automata. We first state a partition lemma for temporalized logics. The proof is similar to the one of the corresponding lemma for temporalized automata.

Lemma 3.2.6 (*Partition Lemma for temporalized logics*)

Let φ be a temporalized formula of $\mathbf{T_1}(\mathbf{T_2})$ and $\alpha_1, \ldots, \alpha_n$ be the maximal $\mathbf{T_2}$-formulas of φ. Then, there is a finite set Λ of $\mathbf{T_2}$-formulas such that:

1. *the set $\{\mathcal{M}(\alpha) \mid \alpha \in \Lambda\}$ forms a partition of $\bigcup_{i=1}^{n} \mathcal{M}(\alpha_i)$, and*
2. *the formula φ' obtained by replacing every $\mathbf{T_2}$-formula α_i in φ with $\bigvee\{\alpha \mid \alpha \in \Lambda$ and $\mathcal{M}(\alpha) \cap \mathcal{M}(\alpha_i) \neq \emptyset\}$ is equivalent to φ, i.e., $\mathcal{M}(\varphi) = \mathcal{M}(\varphi')$.*

We now show that expressive equivalence transfers through temporalized automata.

Theorem 3.2.7 (*Transfer of expressive equivalence w.r.t. temporal logic*)

Expressive equivalence w.r.t. temporal logic transfers through temporalized automata: let $\mathcal{A}_2$ be closed under Boolean operations. If $\mathcal{A}_1 \leftrightarrows \mathbf{T_1}$ and $\mathcal{A}_2 \leftrightarrows \mathbf{T_2}$, then $\mathcal{A}_1(\mathcal{A}_2) \leftrightarrows \mathbf{T_1}(\mathbf{T_2})$.

Proof.

We first prove that $\mathcal{A}_1(\mathcal{A}_2) \to \mathbf{T_1}(\mathbf{T_2})$. Let $A \in \mathcal{A}_1(\mathcal{A}_2)$ be a temporalized automaton over $\Gamma(\Sigma) = \{X_1, \ldots, X_n\} \subseteq \mathcal{A}_2$. We have to find a temporalized formula $\varphi_A \in \mathbf{T_1}(\mathbf{T_2})$ such that $\mathcal{L}(A) = \mathcal{M}(\varphi_A)$. Because of the Partition Lemma 3.2.3, we may assume that $\{\mathcal{L}(X_1), \ldots, \mathcal{L}(X_n)\}$ partitions $\mathcal{S}(\Sigma)$. Since $\mathcal{A}_1 \to \mathbf{T_1}$, we have a translation τ_1 from $\mathcal{A}_1$-automata to $\mathbf{T_1}$-formulas such that, for every $X \in \mathcal{A}_1$, $\mathcal{L}(X) = \mathcal{M}(\tau_1(X))$. Let $\varphi_{A^{\uparrow}} = \tau_1(A^{\uparrow})$. The formula $\varphi_{A^{\uparrow}}$ uses proposition letters in $\{P_{X_1}, \ldots, P_{X_n}\}$. Moreover, since $\mathcal{A}_2 \to \mathbf{T_2}$, we have a translation σ_1 from $\mathcal{A}_2$-automata to $\mathbf{T_2}$-formulas such that, for every $X \in \mathcal{A}_2$, $\mathcal{L}(X) = \mathcal{M}(\sigma_1(X))$. For every $1 \leq i \leq n$, let $\varphi_{X_i} = \sigma_1(X_i)$. For every proposition letter P_{X_i} appearing in $\varphi_{A^{\uparrow}}$, replace P_{X_i} with φ_{X_i} in $\varphi_{A^{\uparrow}}$ and let φ_A be the resulting formula. Note that $\varphi_A \in \mathbf{T_1}(\mathbf{T_2})$. We claim that $\mathcal{L}(A) = \mathcal{M}(\varphi_A)$.

($\subseteq$) Let $x = (\mathbb{N}, <, g) \in \mathcal{L}(A)$. Hence, there is $x^\uparrow = (\mathbb{N}, <, V) \in \mathcal{S}(\Gamma(\Sigma))$ such that $x^\uparrow \in \mathcal{L}(A^\uparrow)$ and, for every $i \in \mathbb{N}$, $g(i) \in \mathcal{L}(V(i))$. Since $\mathcal{L}(A^\uparrow) = \mathcal{M}(\varphi_{A^\uparrow})$, we have that $x^\uparrow \in \mathcal{M}(\varphi_{A^\uparrow})$. We claim that, for every $i \in \mathbb{N}$ and $j \in \{1, \ldots, n\}$, $x^\uparrow, i \models P_{X_j}$ iff $x, i \models \varphi_{X_j}$. We prove the claim. Let $i \in \mathbb{N}$ and $j \in \{1, \ldots, n\}$. We know that $x^\uparrow, i \models P_{X_j}$ iff $V(i) = X_j$. We prove that $V(i) = X_j$ iff $g(i) \in \mathcal{L}(X_j)$. The left to right direction of the previous claim follows since $g(i) \in \mathcal{L}(V(i))$. We prove the right to left direction by contradiction. Suppose $g(i) \in \mathcal{L}(X_j)$ and $V(i) = X_k \neq X_j$. Hence $g(i) \in \mathcal{L}(V(i)) = \mathcal{L}(X_k)$ and thus $g(i) \in \mathcal{L}(X_j) \cap \mathcal{L}(X_k)$. A contradiction, since $\mathcal{L}(X_j) \cap \mathcal{L}(X_k) = \emptyset$. Hence $V(i) = X_j$. Finally, $g(i) \in \mathcal{L}(X_j)$ iff $g(i) \in \mathcal{M}(\varphi_{X_j})$ iff $x, i \models \varphi_{X_j}$. Hence the claim is proved. Summing up, we have that $x^\uparrow \in \mathcal{M}(\varphi_{A^\uparrow})$ and, for every $i \in \mathbb{N}$ and $j \in \{1, \ldots, n\}$, $x^\uparrow, i \models P_{X_j}$ iff $x, i \models \varphi_{X_j}$. It follows that $x \in \mathcal{M}(\varphi_A)$.

($\supseteq$) Let $x = (\mathbb{N}, <, g) \in \mathcal{M}(\varphi_A)$. We define $x^\uparrow = (\mathbb{N}, <, V) \in \mathcal{S}(\Gamma(\Sigma))$ such that, for every $i \in \mathbb{N}$, $V(i) = X_j$ if and only if $g(i) \in \mathcal{M}(\varphi_{X_j}) = \mathcal{L}(X_j)$. Note that $V(i)$ is always defined, since $\{\mathcal{L}(X_1), \ldots, \mathcal{L}(X_n)\}$ partitions $\mathcal{S}(\Sigma)$. We claim that, for every $i \in \mathbb{N}$ and $j \in \{1, \ldots, n\}$, we have that $x^\uparrow, i \models P_{X_j}$ iff $x, i \models \varphi_{X_j}$. We prove the claim. Let $i \in \mathbb{N}$ and $j \in \{1, \ldots, n\}$. We know that $x^\uparrow, i \models P_{X_j}$ iff $V(i) = X_j$. We prove that $V(i) = X_j$ iff $g(i) \in \mathcal{L}(X_j)$. The left to right direction of the previous claim follows by definition of $x^\uparrow$.The right to left direction follows since $\mathcal{L}(X_j) \cap \mathcal{L}(X_k) = \emptyset$ whenever $k \neq j$. Finally, $g(i) \in \mathcal{L}(X_j)$ iff $g(i) \in \mathcal{M}(\varphi_{X_j})$ iff $x, i \models \varphi_{X_j}$. Hence the claim is proved. It follows that $x^\uparrow \in \mathcal{M}(\varphi_{A^\uparrow}) = \mathcal{L}(A^\uparrow)$. Moreover, for every $i \in \mathbb{N}$, $g(i) \in \mathcal{M}(\varphi_{X_j}) = \mathcal{M}(\varphi_{V(i)}) = \mathcal{L}(V(i))$. Therefore, $x \in \mathcal{L}(A)$.

We now prove that $\mathbf{T_1}(\mathbf{T_2}) \rightarrow \mathcal{A}_1(\mathcal{A}_2)$. Let $\varphi \in \mathbf{T_1}(\mathbf{T_2})$ be a temporalized formula. We have to find a temporalized automaton $A_\varphi \in \mathcal{A}_1(\mathcal{A}_2)$ such that $\mathcal{M}(\varphi) = \mathcal{L}(A_\varphi)$. Let $\alpha_1, \ldots, \alpha_n$ be the maximal $\mathbf{T_2}$-formulas of φ. Because of the Partition Lemma 3.2.6, we may assume that there is a finite set Λ of $\mathbf{T_2}$-formulas such that the set $\{\mathcal{M}(\alpha) \mid \alpha \in \Lambda\}$ forms a partition of $\bigcup_{i=1}^{n} \mathcal{M}(\alpha_i)$, and every maximal $\mathbf{T_2}$-formula α_i in φ has the form $\bigvee\{\alpha \mid \alpha \in \Lambda \text{ and } \mathcal{M}(\alpha) \cap \mathcal{M}(\alpha_i) \neq \emptyset\}$.

Let $\varphi^\uparrow$ be the formula obtained from φ by replacing every $\mathbf{T_2}$-formula $\alpha \in \Lambda$ appearing in φ with proposition letter P_α and by adding to the resulting formula the conjunct $P_\beta \vee \neg P_\beta$, where β is the $\mathbf{T_2}$-formula $\neg \bigvee_{i=1}^{n} \alpha_i$. Let $\mathcal{Q} = \{P_\alpha \mid \alpha \in \Lambda \cup \{\beta\}\}$ be the set of proposition letters of $\varphi^\uparrow$. Since $\mathbf{T_1} \rightarrow \mathcal{A}_1$, we have a translation τ_2 from $\mathbf{T_1}$-formulas to $\mathcal{A}_1$-automata such that, for every $\psi \in \mathbf{T_1}$, $\mathcal{M}(\psi) = \mathcal{L}(\tau_2(\psi))$. Let $A_{\varphi^\uparrow} = \tau_2(\varphi^\uparrow)$. The automaton $A_{\varphi^\uparrow}$ labels its transitions with symbols in $2^{\mathcal{Q}}$. Moreover, since $\mathbf{T_2} \rightarrow \mathcal{A}_2$, we have a translation σ_2 from $\mathbf{T_2}$-formulas to $\mathcal{A}_2$-automata such that, for every $\psi \in \mathbf{T_2}$, $\mathcal{M}(\psi) = \mathcal{L}(\sigma_2(\psi))$. For every $\alpha \in \Lambda \cup \{\beta\}$, let $A_\alpha = \sigma_2(\alpha)$. Finally, let A_φ be the automaton obtained by replacing every label $X \subseteq \mathcal{Q}$ on a transition of $A_{\varphi^\uparrow}$ with the $\mathcal{A}_2$-automaton $\bigcap_{P_\alpha \in X} A_\alpha = \sigma_2(\bigwedge_{P_\alpha \in X} \alpha)$. We have that $A_\varphi \in \mathcal{A}_1(\mathcal{A}_2)$ and $\mathcal{L}(A_\varphi) = \mathcal{M}(\varphi)$. The proof is similar to the case $\mathcal{L}(A) = \mathcal{M}(\varphi_A)$. Note that to prove this direction we do not use the hypothesis of closure under Boolean operations of $\mathcal{A}_2$. ■

As a corollary, we have the following:

Corollary 3.2.8 *If* $\mathbf{T_1} \rightarrow \mathcal{A}_1$, $\mathbf{T_2} \rightarrow \mathcal{A}_2$, *and both* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are decidable, then* $\mathbf{T_1}(\mathbf{T_2})$ *is decidable.*

Theorems 3.2.4, 3.2.5 and 3.2.7 hold for finite sequences as well. Moreover, they immediately generalize to finite and infinite trees. Corollary 3.2.8 permits us to show the decidability of many temporalized logics. For instance, we have seen in Chapter 2 that QLTL (and all its fragments) over infinite sequences can be embedded into Büchi sequence automata, $\mathrm{QCTL}^*_\mathrm{k}$

(and all its fragments) over infinite k-ary trees can be embedded into Rabin k-ary tree automata, and both Büchi sequence and Rabin k-ary tree automata are decidable. Moreover, QLTL (and all its fragments) over finite sequences can be embedded into finite sequence automata, $\mathrm{QCTL}^*_\mathrm{k}$ (and all its fragments) over finite k-ary trees can be embedded into finite k-ary tree automata, and both finite sequence and finite k-ary tree automata are decidable. It follows that any temporalized logic $\mathbf{T_1}(\mathbf{T_2})$, where $\mathbf{T_1}$ and $\mathbf{T_2}$ are (fragments of) QLTL or $\mathrm{QCTL}^*_\mathrm{k}$, interpreted over either finite or infinite structures, are decidable. In particular, $\mathrm{PLTL}(\mathrm{CTL}^*_\mathrm{k})$ and $\mathrm{EQLTL}(\mathrm{EQCTL}^*_\mathrm{k})$ over infinite sequences of infinite (resp. finite) trees are decidable. We will use these temporal logics in Chapter 4. Note that the decidability of PPLTL(PPLTL) over infinite sequences of infinite sequences has been previously proved in [44].

We conclude this section by showing how the model checking problem for temporalized logics can be reduced to the emptiness problem for temporalized automata. Let $\mathbf{T_1}$ and $\mathbf{T_2}$ be either linear or branching time logics as defined in Section 2.4 such that $\mathbf{T_1} \to \mathcal{A}_1$ and $\mathbf{T_2} \to \mathcal{A}_2$. The linear and branching time model checking problems for $\mathbf{T_1}(\mathbf{T_2})$ are defined as for component logics (cf. Section 2.4) with respect to temporalized Kripke structures. A *temporalized Kripke structure* is a triple (W, R, g), where W is a set of worlds, $R \subseteq W \times W$ is a binary relation on W, and g is a total function mapping worlds in W into Kripke structures (W', R', V), with $R' \subseteq W' \times W'$ and $V : W' \to 2^{\mathcal{P}}$. A similar definition holds for directed k-ary temporalized Kripke structures. Let $\mathcal{M}$ be a (directed k-ary) temporalized Kripke structure and φ be a $\mathbf{T_1}(\mathbf{T_2})$-formula. Let $A_\mathcal{M}$ be the $\mathcal{A}_1(\mathcal{A}_2)$-automaton recognizing the set of $\mathbf{T_1}(\mathbf{T_2})$-models encoded by $\mathcal{M}$, and let A_φ be the $\mathcal{A}_1(\mathcal{A}_2)$-automaton accepting the set of $\mathbf{T_1}(\mathbf{T_2})$-models of φ, whose existence is guaranteed by Theorem 3.2.7. Then, the model checking problem for $\mathcal{M}$ and φ is equivalent to the language containment problem $\mathcal{L}(A_\mathcal{M}) \subseteq \mathcal{L}(A_\varphi)$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are closed under Boolean operations, the latter problem is equivalent to the emptiness problem $\mathcal{L}(A_\mathcal{M} \cap \overline{A_\varphi}) = \emptyset$. Finally, if $\mathcal{A}_1$ and $\mathcal{A}_2$ are decidable, the latter problem is decidable too. In Section 3.3 we will propose an automata-free model checking algorithm for temporalization, independent combination and join of temporal logics.

3.3 Model checking combined temporal logics

In this section we study the combination of model checking procedures with respect to the three modes of combining logics presented above. We analyze the computational complexity of the proposed procedure and reports on our experiments with implementations. It will turn out that, in contrast to combining deductive engines, combinations of model checking procedures are well behaved, even in the presence of interaction, thus supporting the general believe that modularity is easier to achieve in model checking than in theorem proving approaches [62]. In particular, complexity upper bounds for model checking transfer from the components to the combination.

3.3.1 Combined model checkers

In this section we introduce model checkers for temporalization, independent combination and join.

We start with temporalization. We first define the global model checking problem for the temporalized logic $\mathbf{T}(\mathbf{L})$. Let $\mathcal{M} = (W, \mathcal{R}, g)$ be a $\mathbf{T}(\mathbf{L})$-model. We say that $\mathcal{M}$ is *finite* if W and $\mathcal{R}$ are finite and, for every $w \in W$, $g(w)$ is finite. Let $\mathcal{M} = (W, \mathcal{R}, g)$ be a finite $\mathbf{T}(\mathbf{L})$-model and ψ a formula in $\mathcal{L}_{\mathbf{T}(\mathbf{L})}$. The *global* model checking problem for $\mathbf{T}(\mathbf{L})$ is to

```
Function $\mathtt{MC}_{\mathbf{T(L)}}$
Input: a $\mathbf{T}(\mathbf{L})$-model $\mathcal{M} = (W, \mathcal{R}, g)$ and a formula $\psi \in \mathcal{L}_{\mathbf{T(L)}}$

compute $\mathrm{MML}_{\mathbf{L}}(\psi)$ and $\psi^{\uparrow}$
for every $\alpha \in \mathrm{MML}_{\mathbf{L}}(\psi)$
    for every $w \in W$
        if $\mathtt{MC}_{\mathbf{L}}(g(w), \alpha) = \mathtt{true}$ then
          $V(w) = V(w) \cup \{P_\alpha\}$
return $\mathtt{MC}_{\mathbf{T}}((W, \mathcal{R}, V), \psi^{\uparrow})$
```

Figure 3.5: Model checking procedure for temporalized logics.

check whether there exists $w \in W$ such that $\mathcal{M}, w \models_{\mathbf{T(L)}} \psi$. We use 'model checker' for a program that solves the global model checking problem. Let ψ be a $\mathbf{T}(\mathbf{L})$-formula and $\mathrm{MML}_{\mathbf{L}}(\psi)$ the set of *maximal* monolithic subformulas of ψ belonging to $\mathcal{L}_L$; $\psi^{\uparrow}$ denotes the $\mathbf{T}$-formula obtained from ψ by replacing every formula $\alpha \in \mathrm{MML}_{\mathbf{L}}(\psi)$ by a new proposition letter P_α. Moreover, let $\mathtt{MC}_{\mathbf{T}}$ and $\mathtt{MC}_{\mathbf{L}}$ be model checkers for $\mathbf{T}$ and $\mathbf{L}$, respectively. Given an appropriate model checking instance, these programs return `true` if the corresponding instance is a "yes" instance, `false` otherwise.

In Figure 3.5, we present the pseudo-code of a model checker $\mathtt{MC}_{\mathbf{T(L)}}$ for $\mathbf{T}(\mathbf{L})$ that exploits $\mathtt{MC}_{\mathbf{T}}$ and $\mathtt{MC}_{\mathbf{L}}$. Given a model $\mathcal{M} = (W, \mathcal{R}, g)$ and a formula ψ, the function $\mathtt{MC}_{\mathbf{T(L)}}$ first computes the set $\mathrm{MML}_{\mathbf{L}}(\psi)$ and the formula $\psi^{\uparrow}$. Then, for every maximal monolithic formula $\alpha \in \mathrm{MML}_{\mathbf{L}}(\psi)$ and every world $w \in W$, it invokes the model checker for logic $\mathbf{L}$ with input the $\mathbf{L}$-model $g(w)$ and the $\mathbf{L}$-formula α, and, accordingly to the result, it updates a valuation function V. Finally, it calls the model checker for $\mathbf{T}$ with input the $\mathbf{T}$-model $(W, \mathcal{R}, V)$ and the $\mathbf{T}$-formula $\psi^{\uparrow}$, and returns the output of this invocation.

It is easy to prove that:

Theorem 3.3.1 (*Termination, Soundness, and Completeness*)

Let $\mathcal{M} = (W, \mathcal{R}, g)$ be a finite model for $\mathbf{T}(\mathbf{L})$ and $\psi \in \mathcal{L}_{\mathbf{T(L)}}$. If $\mathtt{MC}_{\mathbf{L}}$ and $\mathtt{MC}_{\mathbf{T}}$ are terminating, sound and complete, then:

1. Termination*: the function $\mathtt{MC}_{\mathbf{T(L)}}$, with input $\mathcal{M}$ and ψ, terminates, returning either* `true` *or* `false`*;*

2. Soundness*: if $\mathtt{MC}_{\mathbf{T(L)}}$ returns* `true` *on input $\mathcal{M}$ and ψ, then there exists $w \in W$ such that $\mathcal{M}, w \models_{\mathbf{T(L)}} \psi$;*

3. Completeness*: if $\mathtt{MC}_{\mathbf{T(L)}}$ returns* `false` *on input $\mathcal{M}$ and ψ, then, for every $w \in W$, $\mathcal{M}, w \not\models_{\mathbf{T(L)}} \psi$.* ■

An alternative model checking algorithm for $\mathbf{T_1}(\mathbf{T_2})$ can be obtained via a reduction to temporalized automata as described in Section 3.2.

We now give a general algorithm for solving the global model checking problem for the independently combined logic $\mathbf{T_1} \oplus \mathbf{T_2}$. Let $\mathbf{T_1}$ and $\mathbf{T_2}$ be two temporal logics, and let $\mathcal{M} = (W, \mathcal{R}_1, \mathcal{R}_2, V)$ be a model for $\mathbf{T_1} \oplus \mathbf{T_2}$. We say that $\mathcal{M}$ is *finite* if W, $\mathcal{R}_1$, and $\mathcal{R}_2$ are finite, and, for every $w \in W$, $V(w)$ is finite. The global model checking problem for $\mathbf{T_1} \oplus \mathbf{T_2}$

```
Procedure $\mathtt{MC}_{\mathbf{T_1}\oplus\mathbf{T_2}}$
Input: a $\mathbf{T_1}\oplus\mathbf{T_2}$-model $\mathcal{M}=(W,\mathcal{R}_1,\mathcal{R}_2,V)$ and a formula $\psi\in\mathcal{L}_{\mathbf{T_1}\oplus\mathbf{T_2}}$

compute $\mathcal{C}^1_{\mathcal{M}}$, $\mathcal{C}^2_{\mathcal{M}}$, and $MSub(\psi)$
for every $w\in W$ let $\overline{V}(w)=V(w)$
for every $i=1,\ldots,|\psi|$
    for every $\varphi\in MSub(\psi)$ such that $|\varphi|=i$
        case on the form of $\varphi$
            $\varphi=P$, $P\in\mathcal{P}$: skip
            $\varphi=\varphi_1\wedge\varphi_2$: for every $w\in W$
                              if ($\varphi_1\in V(w)$ and $\varphi_2\in V(w)$) then
                                $V(w)=V(w)\cup\{\varphi\}$ ; $\overline{V}(w)=\overline{V}(w)\cup\{P_\varphi\}$
            $\varphi=\neg\varphi_1$: for every $w\in W$
                          if (not $\varphi_1\in V(w)$) then
                            $V(w)=V(w)\cup\{\varphi\}$ ; $\overline{V}(w)=\overline{V}(w)\cup\{P_\varphi\}$
            $\varphi=\mathbf{O}(\varphi_1,\ldots,\varphi_c)$, $\mathbf{O}\in OP(\mathcal{L}_{T_i})$, $i\in\{1,2\}$
            let $\Phi=\{\alpha\in Sub(\varphi)\cap MSub(\psi)\mid 1<|\alpha|<|\varphi|\}$ and $\varphi'=\varphi$
            for every $\alpha\in\Phi$ replace $\alpha$ in $\varphi'$ with $P_\alpha$
            for every $(U,\mathcal{S})\in\mathcal{C}^i_{\mathcal{M}}$
                for every $u\in U$ let $V'(u)=\overline{V}(u)$
                $\mathtt{MC}_{\mathbf{T}_i}((U,\mathcal{S},V'),\varphi')$
                for every $u\in U$
                    if $\varphi'\in V'(u)$ then
                        $V(u)=V(u)\cup\{\varphi\}$ ; $\overline{V}(u)=\overline{V}(u)\cup\{P_\varphi\}$
```

Figure 3.6: Model checking procedure for independently combined logics.

is defined just as for $\mathbf{T}(\mathbf{L})$. Let $\mathcal{C}^1_{\mathcal{M}}$ and $\mathcal{C}^2_{\mathcal{M}}$ be the sets of connected components of $(W,\mathcal{R}_1)$ and $(W,\mathcal{R}_2)$, respectively. Since $\mathcal{M}$ is a model for $\mathbf{T_1}\oplus\mathbf{T_2}$, every connected component in $\mathcal{C}^1_{\mathcal{M}}$ (resp. $\mathcal{C}^2_{\mathcal{M}}$) is a model for $\mathbf{T_1}$ (resp. $\mathbf{T_2}$). $Sub(\varphi)$ is the set of subformulas of φ, and $MSub(\varphi)\subseteq Sub(\varphi)$ is constructed as follows. Let $S=Sub(\varphi)\cap\mathcal{L}_{\mathbf{T_1}\oplus\mathbf{T_2}}$. Let $i\in\{1,2\}$. For every formula $\mathbf{O}(\varphi_1,\ldots,\varphi_n)$ in S, with $\mathbf{O}\in OP(\mathcal{L}_{\mathbf{T}_i})\cup\{\wedge,\vee,\neg\}$, if, for every $j=1,\ldots,n$, φ_j is a proposition letter or its main operator is in $OP(\mathcal{L}_{\mathbf{T}_i})\cup\{\wedge,\vee,\neg\}$, then delete formulas $\varphi_1,\ldots,\varphi_n$ from S; $MSub(\varphi)$ is the set S at the end of this procedure. Note that if $\varphi\in\mathcal{L}_{\mathbf{T}_i}$, then $MSub(\varphi)=\{\varphi\}$.

Below, we view model checkers as *procedures* that receive a model $(W,\mathcal{R},V)$ and a formula ψ as input, and that extend the valuation V (which maps a state to a set of proposition letters) to a valuation V' mapping states to sets of *subformulas* of ψ in the following way: for every subformula φ of ψ and every node w, $V'(w)$ contains φ iff φ is true at w in $(W,\mathcal{R},V)$. Let $\mathtt{MC}_{\mathbf{T_1}}$ and $\mathtt{MC}_{\mathbf{T_2}}$ be model checkers for $\mathbf{T_1}$ and $\mathbf{T_2}$, respectively. In Figure 3.6, we present the pseudo-code of a model checker for $\mathbf{T_1}\oplus\mathbf{T_2}$ that exploits the procedures $\mathtt{MC}_{\mathbf{T_1}}$ and $\mathtt{MC}_{\mathbf{T_2}}$. Given a model $\mathcal{M}=(W,\mathcal{R}_1,\mathcal{R}_2,V)$ and a formula ψ, the procedure $\mathtt{MC}_{\mathbf{T_1}\oplus\mathbf{T_2}}$ first computes the sets of connected components $\mathcal{C}^1_{\mathcal{M}}$ and $\mathcal{C}^2_{\mathcal{M}}$ and the set of formulas $MSub(\psi)$. Then, it model checks formulas in $MSub(\psi)$ in increasing order with respect to their lengths, and accordingly it extends the valuation V. In particular, propositional cases are easily solved, while cases of formulas $\varphi\in MSub(\psi)$, with main operator in the language of $\mathbf{T}_i$, $i\in\{1,2\}$, are resolved by taking advantage of the corresponding model checker for $\mathbf{T}_i$. Note that the traversal of $MSub(\psi)$ instead of $Sub(\psi)$ is not essential but it saves time. The following can

```
Procedure $\mathtt{MC}_{\mathbf{T}_1\otimes\mathbf{T}_2}$
Input: a $\mathbf{T}_1 \otimes \mathbf{T}_2$-model $\mathcal{M} = (W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$ and a formula $\psi \in \mathcal{L}_{\mathbf{T}_1\otimes\mathbf{T}_2}$
compute  $MSub(\psi)$
for every $w \in W_2$ compute $\widehat{\mathcal{R}}^1_w$;  for every $w \in W_1$ compute $\widehat{\mathcal{R}}^2_w$
for every  $(w_1, w_2) \in W_1 \times W_2$ let  $\overline{V}((w_1, w_2)) = V((w_1, w_2))$
for  every $i = 1, \ldots, |\psi|$
    for  every $\varphi \in Sub(\psi) \cap (\mathcal{L}_{T_1} \cup \mathcal{L}_{T_2})$ such that $|\varphi| = i$
        case on the form of $\varphi$
            $\varphi = P$, $P \in \mathcal{P}$: skip
            $\varphi = \varphi_1 \wedge \varphi_2$: for  every  $(w_1, w_2) \in W_1 \times W_2$
                            if $\varphi_1 \in V((w_1, w_2))$ and  $\varphi_2 \in V((w_1, w_2))$ then
                              $V((w_1, w_2)) = V((w_1, w_2)) \cup \{\varphi\}$
                              $\overline{V}((w_1, w_2)) = \overline{V}((w_1, w_2)) \cup \{P_\varphi\}$
            $\varphi = \neg\varphi_1$: for  every  $(w_1, w_2) \in W_1 \times W_2$
                        if  not  $\varphi_1 \in V((w_1, w_2))$ then
                          $V((w_1, w_2)) = V((w_1, w_2)) \cup \{\varphi\}$
                          $\overline{V}((w_1, w_2)) = \overline{V}((w_1, w_2)) \cup \{P_\varphi\}$
            $\varphi = \mathbf{O}(\varphi_1, \ldots, \varphi_c)$, $\mathbf{O} \in OP(\mathcal{L}_{T_i})$, $i \in \{1, 2\}$
            let  $\Phi = \{\alpha \in Sub(\varphi) \cap MSub(\psi) \mid 1 < |\alpha| < |\varphi|\}$ and $\varphi' = \varphi$
            for  every  $\alpha \in \Phi$ replace  $\alpha$ in $\varphi'$ with $P_\alpha$
            for  every  $w \in W_{\overline{i}}$
            if  $i = 1$ then  $D = W_1 \times \{w\}$ else  $D = \{w\} \times W_2$
                for  every  $(u, v) \in D$ let  $V'((u, v)) = \overline{V}((u, v))$
                $\mathtt{MC}_{\mathrm{T}_i}((D, \widehat{\mathcal{R}}^i_w, V'), \varphi')$
                for  every  $(u, v) \in D$
                    if  $\varphi' \in V'((u, v))$ then $V((u, v)) = V((u, v)) \cup \{\varphi\}$;
                                                   $\overline{V}((u, v)) = \overline{V}((u, v)) \cup \{P_\varphi\}$
```

Figure 3.7: Model checking procedure for joined logics.

be easily proved.

Theorem 3.3.2 (*Termination, Soundness, and Completeness*)

Let $\mathcal{M} = (W, \mathcal{R}_1, \mathcal{R}_2, V)$ be a finite model for $\mathbf{T_1} \oplus \mathbf{T_2}$ and $\psi \in \mathcal{L}_{\mathbf{T_1}\oplus\mathbf{T_2}}$. If $\mathtt{MC}_{\mathbf{T_1}}$ and $\mathtt{MC}_{\mathbf{T_2}}$ are terminating, sound, and complete, then:

1. Termination*: the procedure $\mathtt{MC}_{\mathbf{T_1}\oplus\mathbf{T_2}}$, with input $\mathcal{M}$ and ψ, terminates;*

2. Soundness *and* Completeness*: let V be the (extended) valuation function after termination of procedure $\mathtt{MC}_{\mathbf{T_1}\oplus\mathbf{T_2}}$, with input $\mathcal{M}$ and ψ. Then, for every subformula φ of ψ and every world $w \in W$, $\varphi \in V(w)$ if and only if $\mathcal{M}, w \models_{\mathbf{T_1}\oplus\mathbf{T_2}} \varphi$.* ■

Finally, we give a general algorithm that solves the global model checking problem for $\mathbf{T_1} \otimes \mathbf{T_2}$. Let $\mathbf{T_1}$ and $\mathbf{T_2}$ be temporal logics and $\mathcal{M} = (W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$ be a model for $\mathbf{T_1}\otimes\mathbf{T_2}$. We say that $\mathcal{M}$ is *finite* if W_1, W_2, $\mathcal{R}_1$ and $\mathcal{R}_2$ are finite, and, for every $(w_1, w_2) \in W_1 \times W_2$, $V((w_1, w_2))$ is finite. Let $\mathcal{M} = (W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$ be a finite $\mathbf{T_1} \otimes \mathbf{T_2}$-model and $\psi \in \mathcal{L}_{\mathbf{T_1}\otimes\mathbf{T_2}}$. The *global* model checking problem for $\mathbf{T_1} \otimes \mathbf{T_2}$ is to check whether there exist $w_1 \in W_1$ and $w_2 \in W_2$ such that $\mathcal{M}, w_1, w_2 \models_{\mathbf{T_1}\otimes\mathbf{T_2}} \psi$.

Given a $\mathbf{T_1} \otimes \mathbf{T_2}$-model $\mathcal{M} = (W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$, a binary relation $R \in \mathcal{R}_1$ on W_1 (resp. $R \in \mathcal{R}_2$ on W_2), and a world $w \in W_2$ (resp. $w \in W_1$), we use $\widehat{R}^1_w$ (resp. $\widehat{R}^2_w$) to denote the binary relation on $W_1 \times W_2$ (resp. $W_2 \times W_1$) such that $\widehat{R}^1_w((x_1, y_1), (x_2, y_2))$ if and only if $R(x_1, x_2)$ and $y_1 = y_2 = w$ (resp. $\widehat{R}^2_w((x_1, y_1), (x_2, y_2))$ if and only if $R(y_1, y_2)$ and $x_1 = x_2 = w$). Moreover, let $\widehat{\mathcal{R}}^1_w = \{\widehat{R}^1_w \mid R \in \mathcal{R}_1\}$ and $\widehat{\mathcal{R}}^2_w = \{\widehat{R}^2_w \mid R \in \mathcal{R}_2\}$. Finally, let $\bar{i} = 2$ if $i = 1$, and $\bar{i} = 1$ if $i = 2$.

In Figure 3.7, we present the pseudo-code for a model checker for $\mathbf{T_1} \otimes \mathbf{T_2}$ that exploits model checkers $\mathtt{MC}_{\mathbf{T_1}}$ and $\mathtt{MC}_{\mathbf{T_2}}$ for the component logics $\mathbf{T_1}$ and $\mathbf{T_2}$, respectively. The implementation is similar to that of the model checker for $\mathbf{T_1} \oplus \mathbf{T_2}$.

Theorem 3.3.3 (*Termination, Soundness, and Completeness*)

Let $\mathcal{M} = (W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$ be a finite model for $\mathbf{T_1} \otimes \mathbf{T_2}$ and $\psi \in \mathcal{L}_{\mathbf{T_1} \otimes \mathbf{T_2}}$. If $\mathtt{MC}_{\mathbf{T_1}}$ and $\mathtt{MC}_{\mathbf{T_2}}$ are terminating, sound, and complete, then:

1. Termination*: the procedure $\mathtt{MC}_{\mathbf{T_1} \otimes \mathbf{T_2}}$, with input $\mathcal{M}$ and ψ, terminates;*

2. Soundness *and* Completeness*: let V be the (extended) valuation function after termination of procedure $\mathtt{MC}_{\mathbf{T_1} \otimes \mathbf{T_2}}$, with input $\mathcal{M}$ and ψ. Then, for every subformula φ of ψ and every pair $w_1, w_2 \in W_1 \times W_2$, $\varphi \in V((w_1, w_2))$ if and only if $\mathcal{M}, w_1, w_2 \models_{\mathbf{T_1} \otimes \mathbf{T_2}} \varphi$.* ■

3.3.2 Computational Complexity

We now turn to an analysis of the computational complexity of the model checkers proposed in the previous section. An instance for the model checking problem has two components: a model $(W, \mathcal{R}, V)$ and a formula ψ. In our analysis, we will consider three main complexity parameters: the cardinality n of W, the sum m of the cardinalities of the relations in $\mathcal{R}$, and the length k of ψ, i.e., the number of operators and proposition letters in ψ. Given $w \in W$, we will heavily use the following operations on the (extended) valuation $V(w)$: checking whether a formula φ belongs to $V(w)$, and adding a formula φ to $V(w)$. Both operations can be efficiently implemented in constant time by representing V as a 2-dimensional bit array of size $n \times k$ [18].

We will express the complexity of the combined model checker in terms of that of the component model checkers. It will turn out that the complexity of the combined model checker is the sum of two factors: the communication overhead and the model checking cost. The *communication overhead* is the time spent for "packing" the inputs for the components and for "unpacking" their outputs; this represents the cost of the interaction between the components. The *model checking cost* represents the cost of performing the actual model checking of the component logics.

We first consider the case of temporalization. Let $\mathbf{L}$ be a logic and $\mathbf{T}$ a temporal logic. We write $C_{\mathbf{T(L)}}(\cdot, \cdot, \cdot)$ (resp. $C_{\mathbf{L}}(\cdot, \cdot)$, $C_{\mathbf{T}}(\cdot, \cdot, \cdot)$) for the complexity function of the model checker $\mathtt{MC}_{\mathbf{T(L)}}$ (resp. $\mathtt{MC}_{\mathbf{L}}$, $\mathtt{MC}_{\mathbf{T}}$). Note that $C_{\mathbf{L}}(\cdot, \cdot)$ has two parameters (the size of the model and the length of the formula).

Theorem 3.3.4 *Let $(W, \mathcal{R}, g)$ be a finite $\mathbf{T(L)}$-model and ψ a $\mathbf{T(L)}$-formula. The complexity of $\mathtt{MC}_{\mathbf{T(L)}}$ on input $\mathcal{M}$ and ψ is*

$$O(n) \cdot [O(k) \cdot C_{\mathbf{L}}(N, O(1)) + O(1) \cdot C_{\mathbf{L}}(N, O(k))] + C_{\mathbf{T}}(n, m, O(k)),$$

where $n = |W|$, $m = \sum_{R \in \mathcal{R}} |R|$, $k = |\psi|$ and $N = \max_{w \in W} |g(w)|$.

Proof.

The set of formulas $\mathrm{MML}_{\mathrm{L}}(\psi)$ and the formula $\psi^{\uparrow}$ can be computed in one pass through ψ, hence in $\mathcal{O}(k)$.

The subsequent nested **for** loop costs

$$\sum_{\alpha \in \mathrm{MML}_{\mathrm{L}}(\psi)} \sum_{w \in W} C_{\mathbf{L}}(|g(w)|, |\alpha|) \leq n \cdot \sum_{\alpha \in \mathrm{MML}_{\mathrm{L}}(\psi)} C_{\mathbf{L}}(N, |\alpha|).$$

To bound the last sum, notice that the set $\mathrm{MML}_{\mathrm{L}}(\psi)$ contains only subformulas of ψ and its cardinality is $O(k)$. Since a set of cardinality n can be partitioned either into $\Theta(n)$ sets of cardinality $\Theta(1)$ or into $\Theta(1)$ sets of cardinality $\Theta(n)$, the above sum is

$$O(n) \cdot [O(k) \cdot C_{\mathbf{L}}(N, O(1)) + O(1) \cdot C_{\mathbf{L}}(N, O(k))].$$

Finally, as the length of $\psi^{\uparrow}$ is $O(k)$, the unique call to $\mathtt{MC}_{\mathrm{T}}$ costs $C_{\mathbf{T}}(n, m, O(k))$. Summing up, the overall cost is

$$O(n) \cdot [O(k) \cdot C_{\mathbf{L}}(N, O(1)) + O(1) C_{\mathbf{L}}(N, O(k))] + C_{\mathbf{T}}(n, m, O(k)). \quad \blacksquare$$

The communication overhead is the cost of computing the set $\mathrm{MML}_{\mathbf{L}}(\psi)$ and the formula $\psi^{\uparrow}$. It equals $O(k)$ and is dominated by the model checking cost. For instance, if $\mathbf{T}$ is CTL (hence $C_{\mathbf{T}}(n, m, k) = O((n+m) \cdot k)$), and $\mathbf{L}$ is a logic such that $C_{\mathbf{L}}(n, k) = O(n \cdot k)$, then the model checking cost is $O(k \cdot (n \cdot N + m))$, hence still linear in the size of the model and in the length of the formula.

We now treat the independent combination of two temporal logics $\mathbf{T_1}$ and $\mathbf{T_2}$.

Theorem 3.3.5 *Let $\mathcal{M} = (W, \mathcal{R}_1, \mathcal{R}_2, V)$ be a finite $\mathbf{T_1} \oplus \mathbf{T_2}$-model and ψ a $\mathbf{T_1} \oplus \mathbf{T_2}$-formula. The complexity of $\mathtt{MC}_{\mathbf{T_1} \oplus \mathbf{T_2}}$ on input $\mathcal{M}$ and ψ is:*

$$\begin{array}{ll} O(m_1 + m_2 + n \cdot k) + & \\ \sum_{i=1}^{2} (& O(k) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(1)) + \\ & O(n) \cdot C_{\mathbf{T}_i}(O(1), O(1), O(k)) + \\ & O(1) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(k))) \end{array}$$

where $n = |W|$, $m_i = \sum_{R \in \mathcal{R}_i} |R|$, for $i = 1, 2$, and $k = |\psi|$.

Proof.

The set of connected components $\mathcal{C}^1_{\mathcal{M}}$ (resp. $\mathcal{C}^2_{\mathcal{M}}$) can be computed in time $O(n + m_1)$ (resp. $O(n + m_2)$) by means of a depth-first visit of the graph $(W, \mathcal{R}_1)$ (resp. $(W, \mathcal{R}_2)$). The cost of computing $MSub(\psi)$ is linear in the size of ψ, and hence it is $O(k)$.

The cost of the second part of the computation is: $\sum_{\varphi \in MSub(\psi)} C(\varphi)$, where the cost factor $C(\varphi)$ depends on the form of φ. In particular, if φ is a proposition letter, then $C(\varphi) = O(1)$. If $\varphi = \varphi_1 \wedge \varphi_2$, or $\varphi = \neg\varphi_1$, then $C(\varphi) = O(n)$. If $i \in \{1, 2\}$ and $\varphi = \mathbf{O}(\varphi_1, \ldots, \varphi_c)$, with $\mathbf{O} \in OP(\mathcal{L}_{T_i})$, then the cost $C(\varphi)$ is computed as follows. Let $\mathcal{C}^i_{\mathcal{M}} = \{(U^i_j, S^i_j) \mid j = 1, \ldots, c_i\}$, n^i_j and m^i_j be the cardinalities of U^i_j and S^i_j, respectively, for $j = 1, \ldots, c_i$. The replacement of subformulas α in φ' with letters P_α costs $O(|\varphi'|)$. Since $|\varphi'| = O(|\varphi|)$, the replacement costs $O(|\varphi|)$. Moreover, for every connected component (U^i_j, S^i_j) in $\mathcal{C}^i_{\mathcal{M}}$, the following steps are performed:

- the valuation V' is computed in $O(n^i_j)$;

- the formula φ' is model checked in $C_{\mathbf{T}_i}(n^i_j, m^i_j, O(|\varphi|))$;
- the valuation V is updated in $O(n^i_j)$.

It follows that, in this case, $C(\varphi)$ amounts to

$$O(|\varphi|) + \sum_{j=1}^{c_i} \left(O(n^i_j) + C_{\mathbf{T}_i}(n^i_j, m^i_j, O(|\varphi|))\right).$$

Since $\mathcal{C}^i_{\mathcal{M}}$ contains either $\Theta(n)$ connected components with $\Theta(1)$ nodes or $\Theta(1)$ connected components with $\Theta(n)$ nodes, $C(\varphi)$ is as follows:

$$O(|\varphi|) + O(n) + O(n) \cdot C_{\mathbf{T}_i}(O(1), O(1), O(|\varphi|)) + O(1) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(|\varphi|)).$$

Moreover, since the set $MSub(\psi)$ contains either $\Theta(k)$ formulas of length $\Theta(1)$ or $\Theta(1)$ formulas of length $\Theta(k)$, the cost of the second part of the computation is:

$$\begin{array}{lll} O(n \cdot k) & + \sum_{i=1}^{2} & (O(k) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(1)) + \\ & & \;O(n) \cdot C_{\mathbf{T}_i}(O(1), O(1), O(k)) + \\ & & \;O(1) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(k))\ ) \end{array}$$

and, hence, the overall complexity is:

$$\begin{array}{lll} O(m_1 + m_2 + n \cdot k) & + \sum_{i=1}^{2} & (O(k) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(1)) + \\ & & \;O(n) \cdot C_{\mathbf{T}_i}(O(1), O(1), O(k)) + \\ & & \;O(1) \cdot C_{\mathbf{T}_i}(O(n), O(m_i), O(k))\ ). \quad \blacksquare \end{array}$$

The communication overhead is the cost of computing the connected components, of preparing the valuation as input to the model checking procedure, and of updating the valuations when the procedure returns. It adds up to $\mathcal{O}(m_1 + m_2 + n \cdot k)$, which is more significant than in the case of temporalization. By way of example, if both $\mathbf{T_1}$ and $\mathbf{T_2}$ are CTL, and $m = m_1 = m_2$, then the communication overhead is $O(m + n \cdot k)$, which is proportional to the model checking cost of $O((n + m) \cdot k)$. So, the overall cost of the model checker for CTL $\oplus$ CTL is $O((n + m) \cdot k)$, which is linear in the size of the model and in the length of the formula. If both $\mathbf{T_1}$ and $\mathbf{T_2}$ are CTL*, then the model checking cost is exponential in the length k of the formula, and hence it dominates the communication overhead.

Finally, we consider the join of temporal logics $\mathbf{T_1}$ and $\mathbf{T_2}$.

Theorem 3.3.6 *Let $\mathcal{M} = (W_1, \mathcal{R}_1, W_2, \mathcal{R}_2, V)$ be a finite $\mathbf{T_1} \otimes \mathbf{T_2}$-model and ψ a $\mathbf{T_1} \otimes \mathbf{T_2}$-formula. Let $\overline{1} = 2$ and $\overline{2} = 1$. The complexity of $\mathtt{MC}_{\mathbf{T_1} \otimes \mathbf{T_2}}$ on input $\mathcal{M}$ and ψ is:*

$$\begin{array}{l} O(n_1 \cdot m_2 + n_2 \cdot m_1 + n_1 \cdot n_2 \cdot k) + \\ \sum_{i=1}^{2} O(n_{\overline{i}}) \cdot [O(k) \cdot C_{\mathbf{T}_i}(n_i, m_i, O(1)) + O(1) \cdot C_{\mathbf{T}_i}(n_i, m_i, O(k))], \end{array}$$

where $n_i = |W_i|$, $m_i = \sum_{R \in \mathcal{R}_i} |R|$, for $i = 1, 2$, and $k = |\psi|$.

Proof.

The sets $\widehat{\mathcal{R}}^1_w$ and $\widehat{\mathcal{R}}^2_w$ can be computed in $O(n_2 \cdot m_1)$ and $O(n_1 \cdot m_2)$, respectively. The cost of computing $MSub(\psi)$ is linear in the size of ψ, hence it is $O(k)$. The cost of the second part of the computation is: $\sum_{\varphi \in MSub(\psi)} C(\varphi)$, where the cost factor $C(\varphi)$ depends on the form of φ. In particular, if φ is a proposition letter, then $C(\varphi) = O(1)$. If $\varphi = \varphi_1 \wedge \varphi_2$, or $\varphi = \neg\varphi_1$, then $C(\varphi) = O(n_1 \cdot n_2)$. If $\varphi = \mathbf{O}(\varphi_1, \ldots, \varphi_c)$, with $\mathbf{O} \in OP(\mathcal{L}_{\mathbf{T_1}})$, the cost $C(\varphi)$ amounts to $O(|\varphi|)$ to replace subformulas α in φ with letters P_α, plus n_2 times the sum of the following factors:

- n_1 to compute V';
- $C_{\mathbf{T_1}}(n_1, m_1, O(|\varphi|))$ to model check φ;
- n_1 to update V.

That is, $C(\varphi)$ is

$$O(|\varphi|) + O(n_2) \cdot [O(n_1) + C_{\mathbf{T_1}}(n_1, m_1, O(|\varphi|))].$$

Similarly, if $\varphi = \mathbf{O}(\varphi_1, \ldots, \varphi_c)$, with $\mathbf{O} \in OP(\mathcal{L}_{T_2})$, the cost $C(\varphi)$ amounts to

$$O(|\varphi|) + O(n_1) \cdot [O(n_2) + C_{\mathbf{T_2}}(n_2, m_2, O(|\varphi|))].$$

It follows that the cost of the second part of the computation is

$$\begin{aligned}&O(n_1 \cdot n_2 \cdot k)+\\&\textstyle\sum_{i=1}^{2} O(n_{\bar{i}}) \cdot [O(k) \cdot C_{\mathbf{T}_i}(n_i, m_i, O(1)) + O(1) \cdot C_{\mathbf{T}_i}(n_i, m_i, O(k))],\end{aligned}$$

and, hence, the overall complexity is:

$$\begin{aligned}&O(n_1 \cdot m_2 + n_2 \cdot m_1 + n_1 \cdot n_2 \cdot k)+\\&\textstyle\sum_{i=1}^{2} O(n_{\bar{i}}) \cdot [O(k) \cdot C_{\mathbf{T}_i}(n_i, m_i, O(1)) + O(1) \cdot C_{\mathbf{T}_i}(n_i, m_i, O(k))].\end{aligned}$$

■

The communication overhead is the cost of computing the sets $\widehat{\mathcal{R}}^1_w$ and $\widehat{\mathcal{R}}^2_w$ plus the cost of preparing and updating the valuation functions before and after the invocation of the model checking procedure, respectively. It amounts to $\mathcal{O}(n_2 \cdot m_1 + n_1 \cdot m_2 + n_1 \cdot n_2 \cdot k)$. For instance, if both $\mathbf{T_1}$ and $\mathbf{T_2}$ are CTL, $n = n_1 = n_2$, and $m = m_1 = m_2$, then the communication cost is $O(n \cdot (m + n \cdot k))$, which is proportional to the model checking cost of $O(n \cdot (n+m) \cdot k)$. Hence, the overall cost of the model checker for CTL $\otimes$ CTL is $O(n \cdot (n+m) \cdot k)$. If $m = \Theta(n)$, the cost is $O(n^2 \cdot k)$, hence it is linear in the size of the model and the length of the formula, else if $m = \Theta(n^2)$, then the complexity is $O(n^3 \cdot k)$. As in the case of the independent combination, if both $\mathbf{T_1}$ and $\mathbf{T_2}$ are CTL*, the model checking dominates the communication overhead.

3.3.3 Experimental Results

We report on experimental results based on implementations of (combined) model checkers for CTL(CTL) and CTL $\oplus$ CTL. The model checkers have been implemented in C, and are available from `http://www.science.uva.nl/~mdr/ACLG/Software/`. Tests were carried on a Sun ULTRA II (300MHz) with 1Gb RAM, under Solaris 5.2.5.

First, we treat the case of temporalization of CTL by means of CTL. We tested the model checker on "linear" and "dense" models, varying the size of the model. Let $\mathcal{M}$ be a model for CTL(CTL) and φ a formula in the language of CTL(CTL). In the first test, we fixed φ to be $\mathbf{A_1 G_1 A_2}(P\,\mathbf{U_2}\,Q)$, and we adopted as model $\mathcal{M}_1 = (W, R, g)$, where (W, R) is a complete binary tree of height h_1 and, for every $w \in W$, $g(w)$ is a labeled complete binary tree of height h_2. Hence, this model contains $n = n_1 \cdot (n_2 + 1)$ nodes, and $m = n_1 \cdot n_2 - 1$ edges, where $n_1 = 2^{h_1+1} - 1$ and $n_2 = 2^{h_2+1} - 1$. Moreover, the trees $g(w)$ are labeled so that every node

h_1	h_2	*# nodes*	*# edges*	t_{ms}
4	4	992	960	10
5	5	4032	3968	30
6	6	16256	16128	110
7	7	65280	65024	380
8	8	261632	261120	1490
9	9	1047552	1046528	5850

Table 3.1: Trees and $\mathbf{A_1 G_1 A_2}(P\,\mathbf{U_2}\,Q)$

n_1	n_2	*# nodes*	*# edges*	t_{ms}
32	32	1056	33792	20
64	64	4160	266240	110
128	128	16512	2113536	820
256	256	65792	16842752	6010
512	512	262656	134479872	47970
1024	1024	1049600	1074790400	386760

Table 3.2: Complete graphs and $\mathbf{A_1 G_1 E_2}(P\,\mathbf{U_2}\,Q)$

and every edge is processed during the checking of $\mathbf{A_2}(P\,\mathbf{U_2}\,Q)$. The experimental outcomes we have obtained on this instance are summarized in Table 3.1, where t_{ms} represents the CPU time in milliseconds. Note that the time needed to perform the model checking grows linearly in the size of the model.

In the second test, we checked the formula $\varphi = \mathbf{A_1 G_1 E_2}(P\,\mathbf{U_2}\,Q)$, and we adopted as model $\mathcal{M}_2 = (W, R, g)$, where (W, R) is a complete graph of n_1 nodes and, for every $w \in W$, $g(w)$ is a complete graph of n_2 nodes. Hence, this model contains $n = n_1 \cdot (n_2 + 1)$ nodes, and $m = n_1 \cdot (n_2^2 + n_1)$ edges. The models $g(w)$ are labeled in an appropriate way in order to process every node and every edge during the checking of $\mathbf{E_2}(P\,\mathbf{U_2}\,Q)$. The outcomes are summarized in Table 3.2. Once again, the time needed to perform the model checking grows linearly in the size of the model. Moreover, as expected, the complexity of the model checker depends on the number of edges too. Indeed, if we compare the costs of the above two instances for the same number of nodes, we note that checking the second instance in harder. This is because $\mathcal{M}_2$ contains *dense* graphs, i.e., graphs in which the number of edges is quadratic in the number of nodes, while $\mathcal{M}_1$ is based on *linear* graphs, i.e., graphs in which the number of edges is linear in the number of nodes.

Next, we treat the case of the independent combination of CTL and CTL. We tested the model checker on "square grid" models, varying the size of the model (and fixing the formula) or varying the "degree of interaction" of the formula (and fixing the model). Let $\mathcal{M}$ be a model for CTL $\oplus$ CTL and φ a formula in the language of CTL $\oplus$ CTL.

In the first test, we fixed φ to be $\mathbf{A}_1\mathbf{G}_1 Q \wedge \mathbf{A}_2\mathbf{G}_2 Q$, and we adopted as our model $\mathcal{M} = (W, R_1, R_2, V)$ a square grid in which the rows are the connected components of (W, R_1) and the columns are the connected components of (W, R_2). We tested the model checker on square grids with a side of size l, hence with number of nodes $n = l^2$ and a number of edges $m = 2 \cdot l \cdot (l - 1)$. The outcomes are summarized in Table 3.3. Note that the time needed to perform the model checking grows linearly in the size of the model.

l	# *nodes*	# *edges*	t_{ms}
32	1024	1984	90
64	4096	8024	340
128	16384	32512	1400
256	65396	130560	5760
512	262144	532264	23480
1024	1048576	2095104	118980

Table 3.3: Square grids and $\mathbf{A}_1\mathbf{G}_1Q \wedge \mathbf{A}_2\mathbf{G}_2Q$

r	0	3	7	11	15	19
t_{ms}	1870	2540	2970	3860	4720	5590

Table 3.4: Fixed square grids and alternating formulas

In our second test, we fixed the model $\mathcal{M}$ to be a square grid with a side of size 256, and hence with 65396 nodes and 130560 edges, and we changed the degree of interaction of the formula. Define $f_0 = Q$, and $f_{k+1} = \mathbf{E}_i\mathbf{X}_i f_k$, for $k \geq 0$ and $i \in \{1, 2\}$. We call the token $\mathbf{E}_i\mathbf{X}_i$ in f_k a *quantifier* of f_k and the token $\mathbf{E}_i\mathbf{X}_i\mathbf{E}_j\mathbf{X}_j$, with $i \neq j$, an *alternation of quantifiers* of f_k. We tested the model checker on f_{20}, varying the number r of alternations of quantifiers of f_{20} from 0 (no interaction at all) to 19 (maximal interaction). The outcomes are summarized in Table 3.4. As expected, the greater the degree of interaction of the formula is, the longer the response time of the model checker becomes. Indeed, the communication overhead is higher when checking formulas with strong interaction, due to the time spent on packing the input and unpacking the output during the switches between the main model checker and the component's ones.

3.4 Discussion

One of the main reasons for the relative ease with which we can make combinations work in this chapter, is that the modes of combining that we consider require no synchronization between the components. Although there may be *interaction* between the components—as we have seen with the join—, this is only a very loose kind of interaction. This is in contrast with *modular model checking*, which has been proposed as a way to address the so-called state-explosion problem. In modular verification, the specification of a module consists of two parts. One part describes the guaranteed behavior of the module. The other part describes the assumed behavior of the environment with which the module is interacting; this is called the assume-guarantee paradigm [74, 83]. The level of interaction between the module and its environment is far more intricate than the kinds of interaction we have been discussing, and, hence, in modular model checking the computational overhead for the combination is much more significant than in our setting [78].

It is worth remarking that we do not consider in this thesis the (nontrivial) problem of decomposing a complex system into simpler components. We simply assume that the target system is already expressible as a combination of components according to some combining method. We showed that in some cases, for instance for granular reactive systems and for mobile reactive systems, this assumption is satisfied. However, we are aware that there are models that cannot be obtained as a composition of simpler components. For instance, not every $S5_2$-model is a join of two models for $S5$. For systems modelled by such models, our modular approach cannot be adopted.

4

Temporal logics and automata for time granularity

In this chapter we define and study temporal logics and automata over downward unbounded layered structures (Section 4.1), n-layered structures (Section 4.2) and upward unbounded layered structures (Section 4.3). To this end, we take advantage of the combining method introduced in Chapter 3. We first reinterpret layered structure as models for temporalization that embeds a vertical flow of time (temporal refinement) inside an horizontal flow of time (temporal evolutions). Accordingly, we define temporalized logics and temporalized automata over layered structures, and study their expressive power as well as their computational complexity. Temporalized automata will be particularly useful: they will provide decision algorithms for temporalized logics, by embedding formulas into automata. Moreover, they will be exploited to define more expressive temporal logics for time granularity, preserving nice computational properties. Finally, we apply the combining approach to model, specify and verify granular reactive systems (Section 4.4).

4.1 Downward unbounded layered structures

In this section we define temporal logics and finite-state automata for DULSs. We investigate both the expressive power and the complexity of the defined tools, and relate them with the monadic theories interpreted over DULSs. In Section 4.1.1 we define a temporal logic counterpart for the path fragment of the monadic second-order theory of DULSs. We show that the satisfiability problem for the defined temporal logic is elementarily decidable and we precisely characterize its complexity. In Section 4.1.2 we define finite-state automata for the second-order theory of DULSs. Moreover, we take advantage of the defined automata class to devise an elementarily decidable temporal logic counterpart of the full second-order theory of DULSs.

4.1.1 Temporal logics for DULSs

In this section we define a temporal logic for DULSs, and study its expressive power and complexity.

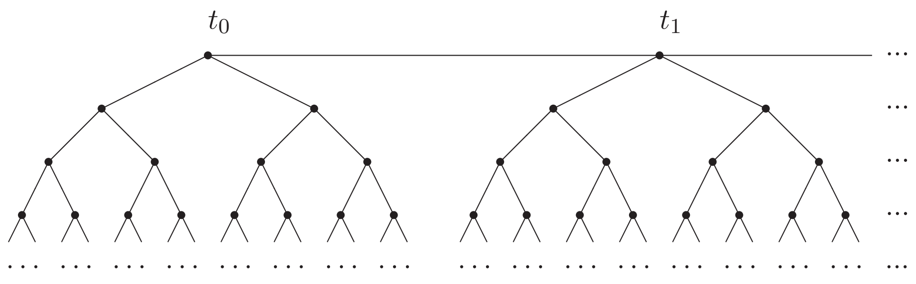

Figure 4.1: A tree sequence.

We start by giving an alternative characterization of DULSs in terms of tree sequences. Let $\mathcal{T}_k(\mathcal{P})$ be the set of $\mathcal{P}$-labeled infinite k-ary trees $(T_k, (\downarrow_i)_{i=0}^{k-1}, <_{pre}, V)$ with $V : T_k \to 2^{\mathcal{P}}$. Let $\mathcal{S}(\mathcal{T}_k(\mathcal{P}))$ be the set of infinite sequences of $\mathcal{P}$-labeled infinite k-ary trees, that is, temporalized models $(\mathbb{N}, <, g)$ where $g : \mathbb{N} \to \mathcal{T}_k(\mathcal{P})$ is a total function mapping worlds in $\mathbb{N}$ into $\mathcal{P}$-labeled infinite k-ary trees in $\mathcal{T}_k(\mathcal{P})$. We show that $\mathcal{P}$-labeled DULSs correspond to tree sequences in $\mathcal{S}(\mathcal{T}_k(\mathcal{P}))$, and vice versa. A $\mathcal{P}$-labeled DULS t can be viewed as an infinite sequence of $\mathcal{P}$-labeled infinite k-ary trees, whose i-th tree, denoted by t_i, is the $\mathcal{P}$-labeled tree rooted at the i-th point i_0 of the coarsest domain T^0 of t (cf. Figure 4.1). Such a sequence can be represented as the temporalized model $(\mathbb{N}, <, g) \in \mathcal{S}(\mathcal{T}_k(\mathcal{P}))$ such that, for every $i \in \mathbb{N}$, $g(i) = t_i$. Similarly, a $\mathcal{P}$-labeled DULS can be viewed as a $\mathcal{P}$-labeled tree sequence.

Since DULSs correspond to temporalized models, we can use temporalized logics $\mathbf{T_1}(\mathbf{T_2})$, where $\mathbf{T_1}$ is a linear time logic and $\mathbf{T_2}$ is a branching time logic, to express properties of DULSs. In the following, we show that the temporalized logics PLTL(CTL$^*_\mathrm{k}$) and PPLTL(PCTL$^*_\mathrm{k}$), interpreted over (temporalized models corresponding to) DULSs, are expressively equivalent to the monadic path theory of DULSs. To avoid confusion we rename linear temporal operators of PLTL and PPLTL as follows: we write $\triangle$, $\triangle^{-1}$, $\bigcirc$, $\bigcirc^{-1}$, $\Box$, and $\Diamond$ instead of $\mathbf{U}$, $\mathbf{S}$, $\mathbf{X}$, $\mathbf{X}^{-1}$, $\mathbf{G}$, and $\mathbf{F}$, respectively.

It will be convenient to work with a different but equivalent monadic path logic that replaces the total ordering $<$ with two partial orderings $<_1$ and $<_2$ defined as follows. Let $t = \langle \mathcal{U}, (\downarrow_i)_{i=0}^{k-1}, < \rangle$ be a DULS. According to the above reinterpretation of DULSs as tree sequences, given $x, y \in \mathcal{U}$, we define $x <_1 y$ iff x is the root of some tree t_i of t, y is the root of some tree t_j of t, and $i < j$ over natural numbers. Moreover, $x <_2 y$ iff x and y belong to the same tree, say t_i, of t, and $x <_{pre} y$ over t_i. We have the following equivalence.

Proposition 4.1.1 $\mathrm{MPL}_\mathcal{P}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}] \leftrightarrows \mathrm{MPL}_\mathcal{P}[<, (\downarrow_i)_{i=0}^{k-1}]$ *over DULSs.*

Proof.

Let $\mathtt{T}^0(x)$ be a shorthand for $\neg\exists y \bigvee_{i=0}^{k-1} \downarrow_i(y) = x$. We first encode $<$ in $\mathrm{MPL}_\mathcal{P}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$. We have that $x < y$ if and only if either x and y belong to the same tree and x lexicographically precedes y, or x belongs to a tree that precedes the tree y belongs to. Hence, $x < y$ if and only if

$$\begin{array}{l} x <_2 y \vee \exists z(\bigvee_{0\le i<j\le k}(\downarrow_i(z) \le_2 x \wedge \downarrow_j(z) \le_2 y)) \vee \\ \exists r_1 \exists r_2 (T^0(r_1) \wedge T^0(r_2) \wedge r_1 <_1 r_2 \wedge r_1 \le_2 x \wedge r_2 \le_2 y). \end{array}$$

We now encode $<_1$ and $<_2$ in $\mathrm{MPL}_\mathcal{P}[<, (\downarrow_i)_{i=0}^{k-1}]$. We have $x <_1 y$ if and only if

$$\mathtt{T}^0(x) \wedge \mathtt{T}^0(y) \wedge x < y.$$

Furthermore, $x <_2 y$ if and only if

$$\exists X(x \in X \wedge y \in X) \wedge x < y.$$

■

Note that Proposition 4.1.1 generalizes to monadic chain and second-order logics. However, we conjecture that it does not hold in monadic first-order logic. In particular, we have that $\mathrm{MFO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}] \rightarrow \mathrm{MFO}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$, since, the encoding of $<$ is the proof of Proposition 4.1.1 is at first-order. We conjecture that the opposite embedding does not hold.

In the following we will focus on $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$ interpreted over $\mathcal{P}$-labeled DULSs $\langle \mathcal{U}, (\downarrow_i)_{i=0}^{k-1}, <_1, <_2, (P)_{P\in\mathcal{P}} \rangle$. We show that the temporalized logic PLTL($\mathrm{CTL}^*_{\mathrm{k}}$) is expressively equivalent to $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$ over $\mathcal{P}$-labeled DULSs. One direction of the proof is easier: there exists a standard embedding of PLTL($\mathrm{CTL}^*_{\mathrm{k}}$) into $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$.

Theorem 4.1.2 PLTL($\mathrm{CTL}^*_{\mathrm{k}}$) *can be embedded into* $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$*, when interpreted over DULSs.*

Proof.

We know that PLTL $\rightarrow \mathrm{MFO}_{\mathcal{P}}[<]$. Hence, there exists a standard translation τ_x from PLTL-formulas into $\mathrm{MFO}_{\mathcal{P}}[<]$-formulas with one free variable x such that, for every PLTL-formula φ, every PLTL-model $\mathcal{M}$ and every point x of $\mathcal{M}$, it holds that $\mathcal{M}, x \models \varphi$ iff $\mathcal{M}, x \models \tau_x(\varphi)$. Let $\hat{\tau}_x$ be the embedding τ_x in which symbol $<$ is replaced by symbol $<_1$. Moreover, we know that $\mathrm{CTL}^*_{\mathrm{k}} \rightarrow \mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$. Thus, there exists a standard translation σ_x from $\mathrm{CTL}^*_{\mathrm{k}}$-formulas into $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$-formulas with one free variable x such that, for every $\mathrm{CTL}^*_{\mathrm{k}}$-formula φ, every $\mathrm{CTL}^*_{\mathrm{k}}$-model $\mathcal{M}$ and every point x of $\mathcal{M}$, it holds that $\mathcal{M}, x \models \varphi$ iff $\mathcal{M}, x \models \sigma_x(\varphi)$. Let $\hat{\sigma}_x$ be the embedding σ_x in which symbol $<_{pre}$ is replaced by symbol $<_2$. An embedding of PLTL($\mathrm{CTL}^*_{\mathrm{k}}$)-formulas into $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$ is as follows. Given $\varphi \in$ PLTL($\mathrm{CTL}^*_{\mathrm{k}}$), let $\varphi^{\uparrow}$ be the PLTL-formula obtained by replacing in φ every maximal monolithic $\mathrm{CTL}^*_{\mathrm{k}}$-subformula α of φ by a new proposition letter P_α. Let $\varphi_1(x) = \hat{\tau}_x(\varphi^{\uparrow})$. Let $\varphi_2(x)$ be the $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$-formula obtained by replacing in φ_1 every atomic formula of the form $y \in P_\alpha$ by the formula $\hat{\sigma}_y(\alpha)$. We have that φ is equivalent to $\varphi_2(0_0)$ modulo the above described isomorphism between DULSs and tree sequences. ■

In order to prove the opposite direction, that is, to show that PLTL($\mathrm{CTL}^*_{\mathrm{k}}$) is expressively complete with respect to $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$, we follow a 'decomposition method' similar to that exploited by Hafer and Thomas in [61] to prove the expressive completeness of CTL^* with respect to MPL[$<$]. We first decompose the model checking problem for an $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$-formula and a DULS into a *finite* number of model checking subproblems for formulas and structures that do not refer to the whole tree sequence anymore, but only to certain disjoint components of it. Then, taking advantage of such a decomposition step, we map every $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$-formula into an equivalent (but sometimes much longer) PLTL($\mathrm{CTL}^*_{\mathrm{k}}$)-formula. We will focus on monadic path logic over *full paths*. As previously pointed out, monadic path logic over paths has the same expressive power than monadic path logic over full paths.

As a preliminary step, we show that the addition of past operators to PLTL($\mathrm{CTL}^*_{\mathrm{k}}$) does not increase its expressive power.

Lemma 4.1.3 PPLTL($\mathrm{PCTL}^*_{\mathrm{k}}$) *is expressively equivalent to* PLTL($\mathrm{CTL}^*_{\mathrm{k}}$).

Proof.

We know that $\mathrm{PCTL}^*_{\mathrm{k}}$ can be embedded into $\mathrm{CTL}^*_{\mathrm{k}}$. Hence, any $\mathrm{PPLTL}(\mathrm{PCTL}^*_{\mathrm{k}})$-formula p can be replaced by an equivalent $\mathrm{PPLTL}(\mathrm{CTL}^*_{\mathrm{k}})$-formula q. Let $q_1, \ldots, q_n$ be the maximal monolithic $\mathrm{CTL}^*_{\mathrm{k}}$-subformulas of q. Regarding $q_1, \ldots, q_n$ as additional atomic propositions within q, we may consider q as a PPLTL-formula. We know that PPLTL can be embedded into PLTL. Hence q can be replaced by an equivalent PLTL-formula that contains $q_1, \ldots, q_n$ as subformulas. Thus the thesis. ■

Let $\equiv_n$ be a relation over $\mathcal{P}$-labeled DULSs such that $t \equiv_n t'$ if and only if t and t' satisfy the same sentences of $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$ of quantifier depth n. It is possible to show that $\equiv_n$ is an equivalence relation of *finite* index. Its equivalence classes are called *n-types* and are described by path formulas called n-types descriptors.

Definition 4.1.4 (*n-types descriptors*)

Let t be a $\mathcal{P}$-labeled DULS, $\overline{a} = a_1 \ldots, a_r$ be a sequence of r elements of t, $\overline{P} = P_1 \ldots, P_s$ be a sequence of s full paths of t, and $n \geq 0$. We denote by $\Pi(t)$ the set of full paths of t. An n-type descriptor $\psi^n_{t,\overline{a},\overline{P}}$ is a path formula defined as follows:

$$\begin{array}{rcl}
\psi^0_{t,\overline{a},\overline{P}} & = & \bigwedge\{\varphi(x_1 \ldots x_r, X_1 \ldots X_s) \mid \varphi \textit{ atomic or negated atomic and } t, \overline{a}, \overline{P} \models \varphi\} \\
\psi^{n+1}_{t,\overline{a},\overline{P}} & = & \bigwedge_{a \in t} \exists x_{r+1} \psi^n_{t,\overline{a}a,\overline{P}} \wedge \bigvee_{a \in t} \forall x_{r+1} \psi^n_{t,\overline{a}a,\overline{P}} \wedge \\
& & \bigwedge_{P \subseteq \Pi(t)} \exists X_{r+1} \psi^n_{t,\overline{a},\overline{P}P} \wedge \bigvee_{P \subseteq \Pi(t)} \forall X_{r+1} \psi^n_{t,\overline{a},\overline{P}P}.
\end{array}$$

□

The relation $\equiv_n$ can be characterized by an Ehrenfeucht game $G_n(t, t')$ as follows (basics on Ehrenfeucht games can be found, for instance, in [31]). A play of this game is played by two players Spoiler and Duplicator on $\mathcal{P}$-labeled DULSs t and t' and consists of n rounds. At each round, Spoiler chooses an element or a full path either from t or from t'; Duplicator reacts by choosing an object of the same kind in the other structure. After n rounds, elements $a_1, \ldots, a_r$ ($\overline{a}$ for short) and full paths $P_1, \ldots, P_s$ ($\overline{P}$) in t (with $n = r + s$), and the corresponding elements $b_1, \ldots, b_r$ ($\overline{b}$) and full paths $Q_1, \ldots, Q_s$ ($\overline{Q}$) in t', have been chosen. Duplicator wins if the map $\overline{a} \to \overline{b}$ is a *partial isomorphism* from $(t, \overline{P})$ to $(t', \overline{Q})$, i.e., it is injective and respects $<_1$, $<_2$, $\downarrow_i$, for $i = 0, \ldots, k-1$, as well as membership in P, for every $P \in \mathcal{P}$. The game can be naturally extended to $G_n((t, \overline{P}), \overline{a}, (t', \overline{Q}), \overline{b})$, where $\overline{a}$ and $\overline{P}$ (resp. $\overline{b}$ and $\overline{Q}$) are a finite sequence of elements and a finite sequence of full paths in t (resp. in t'), respectively.

Let $\sim_n$ be a relation such that, for any pair of structures $(t, \overline{a}, \overline{P})$ and $(t', \overline{b}, \overline{Q})$, $(t, \overline{a}, \overline{P}) \sim_n (t', \overline{b}, \overline{Q})$ if and only if Duplicator wins $G_n((t, \overline{P}), \overline{a}, (t', \overline{Q}), \overline{b})$. The following result easily follows from the well-known Ehrenfeucht-Fraïssé Theorem.

Theorem 4.1.5 *Given $\mathcal{P}$-labeled DULSs t and t', element sequences $\overline{a}$ in t and $\overline{b}$ in t', full path sequences $\overline{P}$ in t and $\overline{Q}$ in t', the following are equivalent conditions:*

1. $(t, \overline{a}, \overline{P}) \sim_n (t', \overline{b}, \overline{Q})$;

2. $t', \overline{b}, \overline{Q} \models \psi^n_{t,\overline{a},\overline{P}}$;

3. *$\overline{a}, \overline{P}$ satisfy in t the same formulas of* $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$ *of quantifier depth less than or equal to n as $\overline{b}, \overline{Q}$ in t'.*

Corollary 4.1.6 *Given $n \geq 0$ and an $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$-formula $\varphi(\overline{x}, \overline{X})$ of quantifier depth less than or equal to n, φ is equivalent to a* finite *disjunction of formulas $\psi^n_{t,\overline{a},\overline{P}}$ such that $t, \overline{a}, \overline{P} \models \varphi$.*

Similar definitions and results hold for k-ary tree structures and infinite as well as finite word structures. In the former case, n-type descriptors are path formulas in the language of k-ary trees $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$. In the latter case, the rules of the game are that Spoiler and Duplicator can only pick *elements* from the given pair of words; hence, n-type descriptors are formulas in the *first-order* language of sequences $\mathrm{MFO}_{\mathcal{P}}[<]$.

In the following it is convenient to view a DULS as a Kripke structure $t = (\mathcal{U}, (\downarrow_i)_{i=0}^{k-1}, <_1, <_2, V)$, with $V : \mathcal{U} \rightarrow 2^{\mathcal{P}}$. Let $k_0 \in \mathcal{U}$ be an element belonging to the i-th tree of t, $n \geq 0$, and m be the index of $\equiv_n$. We enlarge the alphabet $\mathcal{P}$ to $\mathcal{P}_1 = \mathcal{P} \cup \{Z_j \mid j \in \{1, \ldots, m\}\}$, where Z_j serves as index for the j-th n-type. We denote by $v_1(t, k_0)$ the finite $\mathcal{P}_1$-labeled sequence whose l-th element has label $X \cup \{Z_j\}$, $X \subseteq \mathcal{P}$, if $V(l_0) = X$ and the tree rooted at it has n-type j. Moreover, we denote by $v_3(t, k_0)$ the infinite $\mathcal{P}_1$-labeled sequence whose l-th element has label $X \cup \{Z_j\}$, $X \subseteq \mathcal{P}$, if $V((i+1+l)_0) = X$ and the tree rooted at it has n-type j. Similarly, we enlarge the alphabet $\mathcal{P}$ to $\mathcal{P}_2 = \mathcal{P} \cup \{Z_j \mid j \in (\{0, \ldots, k-1\} \times \{1, \ldots, m\})^{k-1}\}$, and we denote by P the finite path from 0_i up to and excluding k_0. We denote by $v_2(t, k_0)$ the $\mathcal{P}_2$-labeled finite sequence whose l-th element has label $X \cup \{Z_{(a_1,b_1),\ldots,(a_{k-1},b_{k-1})}\}$, $X \subseteq \mathcal{P}$, if $V(P(l)) = X$, and, for $r = 1, \ldots, k-1$, the a_r-th son of it that does not belong to P roots a tree of n-type b_r.

We need to prove the following auxiliary lemma, which states that combining *local* winning strategies on disjoint parts of two tree sequences it is possible to obtain a *global* winning strategy on the two tree sequences. Given a node x of a DULS t, we denote by t_x the tree in t rooted at x.

Lemma 4.1.7 *For arbitrary P-labeled DULSs $t = (\mathcal{U}, (\downarrow_i)_{i=0}^{k-1}, <_1, <_2, V)$ and $t' = (\mathcal{U}, (\downarrow_i)_{i=0}^{k-1}, <_1, <_2, V')$, and arbitrary elements $k_0, k_0' \in \mathcal{U}$, if $v_i(t, k_0) \sim_n v_i(t', k_0')$, for $i = 1, 2, 3$, $t_{\downarrow_i(k_0)} \sim_n t'_{\downarrow_i(k_0)}$, for $i = 0, \ldots, k-1$, and $V(k_0) = V'(k_0')$, then $(t, k_0) \sim_n (t', k_0')$.*

Proof.

Suppose that Spoiler picks an element k (different from k_0) in t. If k belongs to some path $v_i(t, k_0)$ or to some $t_{\downarrow_i(k_0)}$, then Duplicator chooses k' in t' according to the corresponding local winning strategies. If k belongs to the tree k_0 belongs to, and neither $k <_2 k_0$ nor $k_0 <_2 k$, then Duplicator looks for the last node k_1 belonging to the path $v_2(t, k_0)$ such that $k_1 <_2 k$. Let k_2 be the son of k_1 such that there is a path from k_2 to k. Suppose that k_2 is the i-th son of k_1. Duplicator chooses k_1' in the path $v_2(t', k_0')$ according to his winning strategy on $v_2(t, k_0), v_2(t', k_0')$. Let k_2' be the i-th son of k_1'. Hence, k_1 and k_1' have the same label from $2^{\mathcal{P}_2}$ and thus the subtrees rooted at their successors k_2 and k_2' have the same n-type. Hence, by Theorem 4.1.5 (its variant for k-ary trees), Duplicator has a winning strategy on these subtrees and can use this strategy to choose an element k' corresponding to k in the subtree rooted at k_2'. Finally, if k and k_0 belong to different trees, then Duplicator chooses k' in t' exploiting, in a similar way, his winning strategy on $v_1(t, k_0), v_1(t', k_0')$ or on $v_3(t, k_0), v_3(t', k_0')$.

Suppose now that Spoiler picks a full path P_0 in t. Let A (resp. A') be the finite path from the root of the tree k_0 (resp. k_0') belongs to up to (and excluding) k_0 (resp. k_0'). If $k_0 \in P_0$, then P_0 has the form A, k_0, B, where B is a path on $t_{\downarrow_i(k_0)}$, for some i. Hence, Duplicator chooses a path B' in $t'_{k_0',i}$ according to his local strategy on $t_{\downarrow_i(k_0)}$, $t'_{k_0',i}$, and he responds to Spoiler with the full path A', k'_0, B'. If $k_0 \notin P_0$ and P_0 belongs to the tree k_0 belongs to, then $P_0 = A_1 k_1 C$, where k_1 is the last node of P_0 which belongs to A and C is a path on $t_{k_1,i}$, for some i. Duplicator first chooses k'_1 according to his local strategy on $v_2(t, k_0), v_2(t', k'_0)$. Let A'_1 be the finite path from the root of the tree k'_0 belongs to up to (and excluding) k'_1. Then, Duplicator picks a path C' on $t'_{k'_1,i}$ exploiting his winning strategy on $t_{k_1,i}$, $t'_{k'_1,i}$, and, finally, he responds to Spoiler with the full path A'_1, k'_1, C' . The case in which P_0 does not belong to the tree k_0 belongs to is similar to the previous one, and thus its analysis is omitted. ∎

The following lemma states that checking a path formula $\varphi(x)$ at a node k_0 belonging to the DULS t corresponds to verifying a number of sentences that do not refer to properties of x relative to the whole structure anymore, but only relative to certain disjoint components of it. In particular, these disjoint substructures are as follows: suppose k_0 belong to the i-th tree of t. We consider the sequence of roots from 0_0 up to 0_{i-1} and the sequence of roots $0_{i+1}, 0_{i+2}, \ldots$, with the trees rooted at them, as well as the path rooted at 0_i up to and excluding k_0, with the trees rooted at successor nodes that are not in it, the node k_0, and finally the k trees rooted at the k successors of k_0.

Lemma 4.1.8 (*First Decomposition Lemma*)

For every $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$*-formula* $\varphi(x)$ *of quantifier depth* n*, there exists a finite set* Φ *of elements of the form* $(\psi_1, \psi_2, X, \beta_0, \ldots, \beta_{k-1}, \psi_3)$*, where* ψ_1 *and* ψ_3 *are* n*-type descriptors in* $\mathrm{MFO}_{\mathcal{P}_1}[<]$*,* ψ_2 *is an* n*-type descriptor in* $\mathrm{MFO}_{\mathcal{P}_2}[<]$*,* $X \in 2^{\mathcal{P}}$*, and* β_i*, for* $i = 0, \ldots, k-1$*, are* n*-type descriptors in* $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$*, such that, for every* $\mathcal{P}$*-labeled DULS* $t = (\mathcal{U}, (\downarrow_i)_{i=0}^{k-1}, <_1, <_2, V)$ *and every element* k_0 *in* $\mathcal{U}$*, it holds that:*

> $t, k_0 \models \varphi(x)$ *if and only if there exists* $(\psi_1, \psi_2, X, \beta_0, \ldots, \beta_{k-1}, \psi_3)$ *in* Φ *such that* $v_i(t, k_0) \models \psi_i$*, for* $i = 1, 2, 3$*,* $V(k_0) = X$*, and* $t_{\downarrow_i(k_0)} \models \beta_i$*, for* $i = 0, \ldots, k-1$*.*

Proof.

By Corollary 4.1.6, $\varphi(x)$ is equivalent to a finite disjunction $\bigvee\{\psi^n_{t',k'_0} \mid t', k'_0 \models \varphi(x)\}$. Hence, $t, k_0 \models \varphi(x)$ if and only if there exist t', k'_0 such that $t', k'_0 \models \varphi(x)$ and $t, k_0 \models \psi^n_{t',k'_0}(x)$. By Theorem 4.1.5, this holds true if and only if there exist t', k'_0 such that $t', k'_0 \models \varphi(x)$ and $(t, k_0) \sim_n (t', k'_0)$. We claim that this is equivalent to the existence of t'', k''_0 such that $t'', k''_0 \models \varphi(x)$, $v_i(t, k_0) \sim_n v_i(t'', k''_0)$, for $i = 1, 2, 3$, $V(k_0) = V''(k''_0)$ and $t_{\downarrow_i(k_0)} \sim_n t''_{\downarrow_i(k''_0)}$, for $i = 0, \ldots, k-1$. We show the claim. The implication from right to left follows from Lemma 4.1.7 by setting $t' = t''$ and $k'_0 = k''_0$. To prove the opposite implication, take $t'' = t$ and $k''_0 = k_0$. Since $\sim_n$ is reflexive, we only have to show that $t, k_0 \models \varphi(x)$. Observe that, by hypothesis, $(t, k_0) \sim_n (t', k'_0)$ and $(t', k'_0) \models \varphi(x)$. Since $\varphi(x)$ is of quantifier depth n, by applying Theorem 4.1.5, we have that $t, k_0 \models \varphi(x)$.

We proceed by invoking the analogous of Theorem 4.1.5 for sequences and k-ary trees, to obtain, respectively, appropriate n-type descriptors $\psi_i = \psi^n_{v_i(t'',k''_0)}$, for $i = 1, 2, 3$, and $\beta_i = \psi^n_{t''_{\downarrow_i(k''_0)}}$, for $i = 0, \ldots, k-1$, such that $v_i(t, k_0) \models \psi_i$, for $i = 1, 2, 3$, and $t_{\downarrow_i(k_0)} \models \beta_i$, for $i = 0 \ldots k-1$. By collecting all such n-type descriptors, we obtain a set Φ as required. Since, for every $n \geq 0$, the equivalence relation $\equiv_n$ has finite index, by virtue of Theorem 4.1.5, there is a finite number of non equivalent n-type descriptors. From this, it follows that the set Φ is finite. ∎

It is possible to prove a similar decomposition lemma for the second-order case. To state it, we need the following definition. Let t be a $\mathcal{P}$-labeled DULS and P_0 be a full path in t. We denote by $v_2(t, P_0)$ the $\mathcal{P}_2$-labeled infinite sequence whose l-th element has label $X \cup \{Z_{(a_1,b_1),\ldots,(a_{k-1},b_{k-1})}\}$, $X \subseteq \mathcal{P}$, if $V(P_0(l)) = X$, and, for $r = 1, \ldots, k-1$, the a_r-th son of it that does not belong to P_0 roots a tree of n-type b_r. The proof of Lemma 4.1.9 is similar to that of Lemma 4.1.8.

Lemma 4.1.9 (*Second Decomposition Lemma*)

For every $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$*-formula* $\varphi(X)$ *of quantifier depth* n*, there exists a finite set* Φ *of elements of the form* (ψ_1, ψ_2, ψ_3)*, where* ψ_1 *and* ψ_3 *are* n*-type descriptors in* $\mathrm{MFO}_{\mathcal{P}_1}[<]$ *and* ψ_2 *is an* n*-type descriptor in* $\mathrm{MFO}_{\mathcal{P}_2}[<]$ *such that, for every* $\mathcal{P}$*-labeled DULS* t *and every full path* P_0 *in* t*, it holds that:*

> $t, P_0 \models \varphi(X)$ *if and only if there exists* (ψ_1, ψ_2, ψ_3) *in* Φ *such that* $v_1(t, k_0) \models \psi_1$*,* $v_2(t, P_0) \models \psi_2$ *and* $v_3(t, k_0) \models \psi_3$*.*

We are now ready to prove the following result.

Theorem 4.1.10 (*Expressiveness of* $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$)

$\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$ *is expressively equivalent to* $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$*, when interpreted over DULSs.*

Proof.

The embedding of temporal formulas into monadic ones has been proved in Theorem 4.1.2. We prove that every $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$-sentence corresponds to an equivalent $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$-formula. We focus on the two relevant cases: $\varphi = \exists x \phi(x)$ and $\varphi = \exists X \phi(X)$.

Let $\varphi = \exists x \phi(x)$. By Lemma 4.1.8, checking $\phi(x)$ in (t, k_0) is equivalent to checking certain sentences ψ_1, ψ_2, $\beta_0, \ldots, \beta_{k-1}$, and ψ_3, and a label $X \in 2^{\mathcal{P}}$, taken from a finite set Φ, in particular substructures of t. It suffices to consider the case in which $|\Phi| = 1$. By Theorem 2.4.4, the first-order sentence ψ_1 can be mapped into an equivalent PLTL-formula h_1. Given the formula h_1, we construct the dual formula $h_1^{-1} \in$ PPLTL, that is, a formula such that, for every finite sequence w of length l, $w, 0 \models h_1$ if and only if $w, l-1 \models h_1^{-1}$. The formula h_1^{-1} contains atomic propositions $P \in \mathcal{P}_1$, which must be replaced by suitable $\mathrm{CTL}^*_{\mathrm{k}}$-formulas q_P. Hereinafter, we will denote by p_j the $\mathrm{CTL}^*_{\mathrm{k}}$-formula equivalent to the j-th n-type descriptor, whose existence is guaranteed by Theorem 2.4.7. Let $q_P = P$ if $P \in \mathcal{P}$, and $q_P = p_j$ if $P = Z_j$. Let $(h_1^{-1})'$ be the $\mathrm{PPLTL}(\mathrm{PCTL}^*_{\mathrm{k}})$-formula obtained from h_1^{-1} by replacing propositions $P \in \mathcal{P}_1$ by formulas q_P (and using the symbols $\bigcirc$, $\triangle$, $\bigcirc^{-1}$, and $\triangle^{-1}$ for the linear time operators that occur in h_1^{-1}).

In a similar way, the first-order sentence ψ_3 can be mapped into a PLTL-formula h_3. The formula h_3 can be turned into a $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$-formula $(h_3)'$ by replacing propositions $P \in \mathcal{P}_1$ by $\mathrm{CTL}^*_{\mathrm{k}}$-formulas q_P as in the previous case (notice that, in this case, linear past operators are not needed).

Finally, the first-order sentence ψ_2 can be mapped into an equivalent PLTL-formula h_2, whose dual version h_2^{-1} is obtained as already explained in the case of h_1. The formula h_2^{-1} contains atomic propositions $P \in \mathcal{P}_2$, which must be replaced by suitable $\mathrm{CTL}^*_{\mathrm{k}}$-formulas q_P. Let $q_P = P$ if $P \in \mathcal{P}$, and $q_P = \bigwedge_{r=1}^{k-1} \mathbf{EX_{a_r}} p_{b_r}$ if $P = Z_{((a_1,b_1),\ldots,(a_{k-1},b_{k-1}))}$. The $\mathrm{PCTL}^*_{\mathrm{k}}$-formula $(h_2^{-1})'$ is obtained by replacing propositions $P \in \mathcal{P}_2$ by formulas q_P.

As for the path sentences β_i, for $i = 0, \ldots, k-1$, let b_i be the index of the n-type descriptor β_i. By exploiting once more Theorem 2.4.7, we obtain a $\mathrm{CTL}^*_{\mathrm{k}}$-formula p_{b_i}, for each $i = 0, \ldots, k-1$.

Merging together the above results, we have that the given $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$-formula φ is equivalent to the PPLTL($\mathrm{PCTL}^*_\mathrm{k}$)-formula:

$$p_\varphi = \Diamond(\bigcirc^{-1}(h_1^{-1})' \wedge \mathbf{EF}p \wedge \bigcirc(h_3)'),$$

where

$$p = \mathbf{X}^{-1}(h_2^{-1})' \wedge \bigwedge_{P \in X} P \wedge \bigwedge_{i=0}^{k-1} \mathbf{EX_i} p_{b_i}.$$

Theorem 4.1.3 guarantees that there exists a PLTL($\mathrm{CTL}^*_\mathrm{k}$)-formula p'_φ, devoid of past operators, which is equivalent to p_φ, and, thus, equivalent to φ.

Let $\varphi = \exists X \phi(X)$. By Lemma 4.1.9, checking $\phi(X)$ in (t, P_0) is equivalent to checking certain sentences ψ_1, ψ_2, and ψ_3 in particular substructures of t. In analogy to the case of first-order quantification, ψ_1, ψ_2, and ψ_3 can be replaced by a PPLTL($\mathrm{PCTL}^*_\mathrm{k}$) formula $(h_1^{-1})'$, a $\mathrm{CTL}^*_\mathrm{k}$ formula $(h_2)'$, and a PLTL($\mathrm{CTL}^*_\mathrm{k}$) formula $(h_3)'$, respectively. It is easy to check that the $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$-formula φ is equivalent to the PPLTL($\mathrm{PCTL}^*_\mathrm{k}$)-formula:

$$p_\varphi = \Diamond(\bigcirc^{-1}(h_1^{-1})' \wedge \mathbf{E}(h_2)' \wedge \bigcirc(h_3)').$$

Again, by Theorem 4.1.3, there exists a PLTL($\mathrm{CTL}^*_\mathrm{k}$)-formula p'_φ which is equivalent to p_φ, and, thus, equivalent to φ. ■

We show some examples of properties that can be expressed in PLTL($\mathrm{CTL}^*_\mathrm{k}$). Let us consider the following property over DULSs: we say that proposition P *densely* holds at node x if, for every $i \geq 0$, there is $y \in \downarrow^i(x)$ such that P holds at y. We recall that, for $i \geq 0$, $\downarrow^i(x)$ is the i-th layer of the tree rooted at x. The property 'P densely holds at some x' can be written in monadic path logic as follows:

$$\exists X(x \in X \wedge \forall y(y \in X \rightarrow x \leq_2 y) \wedge \forall y(y \in X \rightarrow y \in P)).$$

The same property can be (more concisely) expressed in PLTL($\mathrm{CTL}^*_\mathrm{k}$) by the formula

$$\Diamond\mathbf{EFEG}P.$$

Moreover, the property 'P holds along the leftmost path', where the leftmost path is the path $0_0, 0_1, \ldots$, is encoded in monadic path logic as

$$\exists X(0_0 \in X \wedge \forall y(y \in X \rightarrow \downarrow_0(y) \in X) \wedge \forall y(y \in X \rightarrow y \in P)),$$

where 0_0 is first-order definable as follows: $x = 0_0$ iff $\forall y(x \leq_1 y)$. The same property is (more elegantly) expressible in temporal logic as follows:

$$\mathbf{E}(P \wedge \mathbf{GX_0}P).$$

However, things are not always that easy. For instance, formulas of the form 'there is exactly one point in which P holds' can be easily encoded in first-order logic as $\exists x(x \in P \wedge \forall y(y \neq x \rightarrow y \notin P))$, while it is not easy at all to express them in temporal logic. Moreover, since $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$ is nonelementarily decidable, while PLTL($\mathrm{CTL}^*_\mathrm{k}$) is elementarily decidable, the translation τ of path formulas into temporal formulas in nonelementary. In other words, for every $n \in \mathbb{N}$, there is a path formula φ such that $\tau(\varphi)$ has length greater than $\kappa(n, |\varphi|)$ (an exponential tower of height n).

We consider the decidability and the complexity of the combined temporal logic $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$. Since $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$ can be embedded into $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$ over DULSs, and since the latter is decidable, we have that $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$ over DULSs is decidable. Unfortunately, $\mathrm{MPL}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$ is *nonelementarily* decidable. On the contrary, $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$ is elementarily decidable, and its exact complexity is shown in the following result.

Theorem 4.1.11 (*Complexity of* $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$)

The satisfiability problem for $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$ *over DULSs is 2EXPTIME-complete.*

Proof.

Hardness follows from 2EXPTIME-hardness of the component logic CTL* [116]. Recall that $\mathcal{B}$ is the class of Büchi sequence automata (Definition 2.3.2) and $\mathcal{R}_k$ is the class of Rabin k-ary tree automata (Definition 2.3.6). To show that the satisfiability problem for $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$ belongs to 2EXPTIME, we give a doubly exponential time algorithm that first embeds temporalized formulas in $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$ into equivalent temporalized automata in $\mathcal{B}(\mathcal{R}_k)$, and then checks $\mathcal{B}(\mathcal{R}_k)$-automata for emptiness. The embedding of temporal formulas into automata is described in the proof of Theorem 3.2.7. Recall that PLTL-formulas can be embedded into $\mathcal{B}$-automata with a singly exponential blow-up in the length of the formula, and $\mathrm{CTL}^*_{\mathrm{k}}$-formulas can be embedded into $\mathcal{R}_k$-automata with a doubly exponential number of states and a singly exponential number of accepting pairs in the length of the formula. Given a $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$-formula ϕ of length n, the equivalent formula ψ resulting from the partition step in the proof of Theorem 3.2.7 is obtained by replacing $\mathrm{CTL}^*_{\mathrm{k}}$-formulas in ϕ by finite disjunctions of $\mathrm{CTL}^*_{\mathrm{k}}$-formulas taken from a set Λ. Since the cardinality of Λ is $O(2^n)$, the formula ψ has length $O(2^n)$. The formula $\psi^{\uparrow}$ has the same length and hence the corresponding Büchi sequence automaton has $O(2^{2^n})$ states. Each $\mathrm{CTL}^*_{\mathrm{k}}$-formula in ψ belonging to Λ has length $O(n)$ and is associated with a Rabin tree automata with $O(2^{n \cdot 2^n})$ states and $O(2^n)$ accepting pairs. Hence, the resulting $\mathcal{B}(\mathcal{R}_k)$-automaton A_ψ associated with ψ has $O(2^{2^n})$ states, and is labeled with $O(2^n)$ Rabin tree automata with $O(2^{n \cdot 2^n})$ states and $O(2^n)$ accepting pairs. We check the emptiness of A_ψ by using the algorithm described in the proof of Theorem 3.2.5. That algorithm checks for emptiness the Rabin tree automata labelling A_ψ, and then it checks for emptiness the Büchi sequence automaton $A_\psi^{\uparrow}$. Since Rabin tree automata can be checked for emptiness in polynomial time in the number of states and singly exponential time in the number of accepting pairs, and the emptiness problem for Büchi sequence automata is polynomial in the number of states, we conclude that A_ψ can be checked for emptiness in time doubly exponential. ■

In Section 4.1.2 we will introduce an elementarily decidable extension of $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$ that is expressively equivalent to the full second-order theory of DULSs.

4.1.2 Automata for DULSs

In this section we define finite-state automata accepting labeled DULSs. We study both the expressive power and the complexity of the defined automata class. Moreover, we exploit automata to devise a temporal logic counterpart of the full second-order theory of DULSs.

Recall that DULSs correspond to tree sequences. Since tree sequences are temporalized models embedding trees into sequences, we can use temporalized automata in the class $\boldsymbol{\mathcal{A}_1(\mathcal{A}_2)}$, where $\mathcal{A}_1$ is a sequence automata class and $\mathcal{A}_2$ is a tree automata class, to express properties of DULSs. In particular, we will consider the temporalized automata class $\mathcal{B}(\mathcal{R}_k)$ embedding Rabin k-ary tree automata (cf. Definition 2.3.6) into Büchi sequence automata (cf. Definition 2.3.6). We call an automaton in $\mathcal{B}(\mathcal{R}_k)$ an *infinite tree sequence automaton.* Since both $\mathcal{B}$ and $\mathcal{R}_k$ are effectively closed under Boolean operations and are decidable, so is the class $\mathcal{B}(\mathcal{R}_k)$ of infinite tree sequence automata (cf. Theorems 3.2.4 and 3.2.5). The complexity of the emptiness problem for infinite tree sequence automata is the following. Let $A \in \mathcal{B}(\mathcal{R}_k)$. Let n be the number of states of A and N (resp. M) be the maximum number of states (resp. accepting pairs) of a Rabin tree automaton labelling transitions in A. Recall that the emptiness of Büchi sequence automata can be checked in polynomial time in the number of states and that of Rabin tree automata can be verified in time polynomial in the number of states and exponential in the number of accepting pairs. Hence, accordingly to the algorithm proposed in the proof of Theorem 3.2.5 for checking the emptiness of temporalized automata, the complexity to verify if A accepts the empty language is polynomial in n and N, and exponential in M.

Theorem 4.1.12 (*Complexity of infinite tree sequence automata*)

The emptiness problem for infinite tree sequence automata is decidable in polynomial time in the number of states, and exponential time in the number of accepting pairs.

The following theorem proves that infinite tree sequence automata are expressively equivalent to the monadic second-order theory of DULSs.

Theorem 4.1.13 (*Expressiveness of infinite tree sequence automata*)

Infinite tree sequence automata are expressively equivalent to the monadic second-order logic over DULSs.

Proof.

We prove that:

1. for every automaton $A \in \mathcal{B}(\mathcal{R}_k)$ over $\Gamma(\Sigma)$ there is a formula $\varphi_A \in \text{MSO}_{\mathcal{P}_\Sigma}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$ over $\mathcal{P}_\Sigma$ such that $\mathcal{L}(A) = \mathcal{M}(\varphi_A)$;
2. for every formula $\varphi \in \text{MSO}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$ over $\mathcal{P}$ there is an automaton $A_\varphi \in \mathcal{B}(\mathcal{R}_k)$ over some $\Gamma(2^{\mathcal{P}})$ such that $\mathcal{M}(\varphi) = \mathcal{L}(A_\varphi)$.

We follow a standard proof technique for embedding automata into monadic formulas and viceversa (see, for instance, the proof of Büchi's Theorem in [114]). We first introduce some shorthands that are easily definable in the monadic second-order logic over DULSs. Let $+1$ be a binary predicate such that $+1(x, y)$ iff x and y belong to the first domain and y is the immediate successor of x. We will write $x + 1 \in X$ for $\exists y(+1(x, y) \wedge y \in X)$. Moreover, let $\mathtt{T}^0(x)$ be a shorthand for "x belongs to the first domain", $0_0 \in X$ be a shorthand for "the first element of the first domain belongs to X", and $\mathtt{Path}(X, x)$ be a shorthand for "X is a path rooted at x".

We prove point 1 for $k = 2$. The generalization to $k > 2$ is straightforward. Let $A = (Q, q_0, \Delta, F)$ be a $\mathcal{B}(\mathcal{R}_2)$-automaton over $\Gamma(\Sigma) \subset \mathcal{R}_2$ accepting tree sequences in $\mathcal{S}(\mathcal{T}_2(\Sigma))$. We will construct a formula $\varphi_A \in \text{MSO}_{\mathcal{P}_\Sigma}[<_1, <_2, \downarrow_0, \downarrow_1]$ using monadic predicates in $\mathcal{P}_\Sigma = \{P_a \mid a \in \Sigma\}$ and interpreted over $\mathcal{S}(\mathcal{T}_2(\Sigma))$ such that $\mathcal{L}(A) = \mathcal{M}(\varphi_A)$. We assume $Q = \{0, \ldots, m\}$ and $q_0 = 0$. For every $Z \in \Gamma(\Sigma)$, let $Z = (Q_Z, q_Z^0, \Delta_Z, \Gamma_Z)$ over Σ, with $Q_Z = \{0, \ldots, m_Z\}$, $q_Z^0 = 0$, and $\Gamma_Z = \{(L_i^Z, U_i^Z) \mid 1 \leq i \leq r_Z\}$.

The formula φ_A in $\text{MSO}_{\mathcal{P}_\Sigma}[<_1, <_2, \downarrow_0, \downarrow_1]$ corresponding to automata A encodes the combined acceptance condition for $\mathcal{B}(\mathcal{R}_2)$-automata. The outermost part of the sentence expresses the existence of an accepting run over the first layer of the tree sequence for the Büchi sequence automaton $A^{\uparrow}$. It uses second-order variable X_i for the set of positions of the run that assume state i, and monadic predicate Q_Z for the set of positions of the run that are labeled with Rabin tree automaton Z. The innermost part $\mathtt{RAC}(x, Z)$ captures the existence of an accepting run over the tree rooted at x for the Rabin tree automaton Z. It uses second-order variables Y_i for the set of positions of the run that assume state i. The formula φ_A is the following:

$$\begin{aligned}&(\exists Q_Z)_{Z\in\Gamma(\Sigma)}(\exists X_i)_{i=0}^{m}(\textstyle\bigwedge_{i=0}^{m}\forall x(x\in X_i\rightarrow \mathtt{T}^0(x))\wedge\bigwedge_{Z\in\Gamma(\Sigma)}\forall x(x\in Q_Z\rightarrow \mathtt{T}^0(x))\wedge\\ &0_0\in X_0\wedge\textstyle\bigwedge_{i\neq j}\neg\exists y(y\in X_i\wedge y\in X_j)\wedge\\ &\forall x(\mathtt{T}^0(x)\rightarrow\textstyle\bigvee_{(i,Z,j)\in\Delta}(x\in X_i\wedge x\in Q_Z\wedge x+1\in X_j))\wedge\\ &\textstyle\bigvee_{i\in F}\forall x(\mathtt{T}^0(x)\rightarrow\exists y(\mathtt{T}^0(y)\wedge x<_1 y\wedge y\in X_i))\wedge\\ &\textstyle\bigwedge_{Z\in\Gamma(\Sigma)}\forall x(x\in Q_Z\rightarrow\mathtt{RAC}(x,Z))\end{aligned}$$

where $\mathtt{RAC}(x, Z)$ stands for:

$$\begin{aligned}&(\exists Y_i)_{i=0}^{m_Z}(\textstyle\bigwedge_{i=0}^{m_Z}\forall y(y\in Y_i\rightarrow x\leq_2 y)\wedge x\in Y_0\wedge\bigwedge_{i\neq j}\neg\exists y(y\in Y_i\wedge y\in Y_j)\wedge\\ &\forall y(x\leq_2 y\rightarrow\textstyle\bigvee_{(i,a,j_0,j_1)\in\Delta_Z}(y\in Y_i\wedge y\in P_a\wedge\downarrow_0(y)\in Y_{j_0}\wedge\downarrow_1(y)\in Y_{j_1}))\wedge\\ &\forall W(\mathtt{Path}(W,x)\rightarrow\textstyle\bigvee_{i=0}^{r_Z}(\bigwedge_{j\in L_i^Z}\exists u(u\in W\wedge\forall v(v\in W\wedge u<_2 v\rightarrow v\notin Y_j))\wedge\\ &\textstyle\bigvee_{j\in U_i^Z}\forall u(u\in W\rightarrow\exists v(v\in W\wedge u<_2 v\wedge v\in Y_j)))))\end{aligned}$$

We now prove point 2. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$. We prove this direction for $\mathrm{MSO}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}, +1]$. It is immediate to see that the latter is equivalent to $\mathrm{MSO}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$. Given a formula in $\mathrm{MSO}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}, +1]$ using monadic predicates in $\mathcal{P}$ and interpreted over $\mathcal{P}$-labeled tree sequences in $\mathcal{S}(\mathcal{T}_k(\mathcal{P}))$, we will construct an automaton $A_\varphi \in \mathcal{B}(\mathcal{R}_k)$ over some $\Gamma(2^{\mathcal{P}})$ and accepting in $\mathcal{S}(\mathcal{T}_k(\mathcal{P}))$ such that $\mathcal{L}(A_\varphi) = \mathcal{M}(\varphi)$.

We first remove the ordering relations $<_1$ and $<_2$ as follows. We replace $x <_1 y$ by

$$\mathtt{T}^0(x)\wedge\mathtt{T}^0(y)\wedge\forall X(x+1\in X\wedge\forall z(z\in X\rightarrow z+1\in X)\rightarrow y\in X)),$$

and $x <_2 y$ by

$$\forall X(\bigwedge_{i=0}^{k-1}\downarrow_i(x)\in X\wedge\forall z(z\in X\rightarrow\bigwedge_{i=0}^{k-1}\downarrow_i(z)\in X)\rightarrow y\in X).$$

Hence $\mathrm{MSO}_{\mathcal{P}}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}, +1]$ is as expressive as $\mathrm{MSO}_{\mathcal{P}}[(\downarrow_i)_{i=0}^{k-1}, +1]$. In the following, we introduce an equivalent variant of $\mathrm{MSO}_{\mathcal{P}}[(\downarrow_i)_{i=0}^{k-1}, +1]$, which will be denoted by $\mathrm{MSO}[(\downarrow_i)_{i=0}^{k-1}, +1]$, which uses free set variables X_i in place of predicate symbols P_i and is interpreted over $\{0,1\}^n$-labeled tree sequences in $\mathcal{S}(\mathcal{T}_k(\{0,1\}^n))$. The idea is to encode a set $X \subseteq \mathcal{P}$ with the string $i_1 \ldots, i_n \in \{0,1\}^n$ such that, for $j = 1, \ldots, n$, $i_j = 1$ iff $P_j \in X$. We now reduce $\mathrm{MSO}[(\downarrow_i)_{i=0}^{k-1}, +1]$ to a simpler formalism $\mathrm{MSO}_0[(\downarrow_i)_{i=0}^{k-1}, +1]$, where *only* second-order variables X_i occur and atomic formulas are of the forms $X_i \subseteq X_j$ (X_i is a subset of X_j), $\mathtt{Proj}_m(X_i, X_j)$, with $m = 0, \ldots, k-1$ (X_i and X_j are the singletons $\{x\}$ and $\{y\}$, respectively, and $\downarrow_m(x) = y$), and $\mathtt{Succ}(X_i, X_j)$ (X_i and X_j are the singletons $\{x\}$ and $\{y\}$, respectively, and $x + 1 = y$). This step is performed as in the proof of Büchi's Theorem.

Finally, given a $\mathrm{MSO}_0[(\downarrow_i)_{i=0}^{k-1}, +1]$-formula $\varphi(X_1, \ldots, X_n)$, we prove, by induction on the complexity of φ, that there exists a temporalized automaton A_φ accepting in $\mathcal{S}(\mathcal{T}_k(\{0,1\}^n))$ such that $\mathcal{M}(\varphi) = \mathcal{L}(A_\varphi)$. A corresponding automaton accepting in $\mathcal{S}(\mathcal{T}_k(\mathcal{P}))$ can be obtained in the obvious way. As for atomic formulas, let $\alpha_{i,j}$ be the Rabin tree automaton over $\{0,1\}^n$ for $X_i \subseteq X_j$. The temporalized automaton for $X_i \subseteq X_j$ is depicted in Figure 4.2

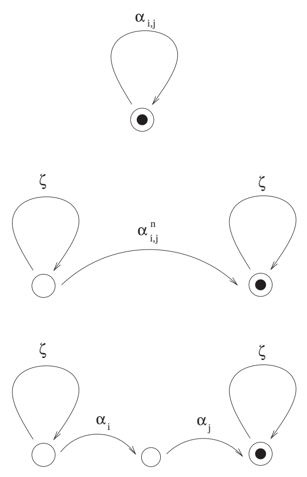

Figure 4.2: Temporalized automata for atomic formulas.

(top). Moreover, let ζ be the Rabin tree automaton over $\{0,1\}^n$ acceptance the singleton set containing a tree labeled with 0^n everywhere, and let $\alpha^m_{i,j}$ be the Rabin tree automaton over $\{0,1\}^n$ for $\texttt{Proj}_m(X_i, X_j)$. The temporalized automaton for $\texttt{Proj}_m(X_i, X_j)$ is depicted in Figure 4.2 (middle). Finally, let α_i be the Rabin tree automaton over $\{0,1\}^n$ acceptance the singleton set containing a tree labeled with $0^{i-1}10^{n-i}$ at the root, and labeled with 0^n elsewhere. The combined automaton for $\texttt{Succ}(X_i, X_j)$ is depicted in Figure 4.2 (bottom). The induction step is clear from the closure of $\mathcal{B}(\mathcal{R}_k)$ automata under Boolean operations and projection. It is easy to see that $\mathcal{B}(\mathcal{R}_k)$-automata are closed under projection: given a $\mathcal{B}(\mathcal{R}_k)$ automaton A, the corresponding projected $\mathcal{B}(\mathcal{R}_k)$ automaton is obtained by projecting every Rabin automaton that labels some transition of A. ■

With the aid of infinite tree sequence automata, we are able to extend the result proved in Theorem 4.1.10 to the full second-order theory of DULSs. We know that $\mathcal{B} \leftrightarrows$ QLTL and $\mathcal{B} \leftrightarrows$ EQLTL (cf. Theorem 2.3.7 point 2 and Theorem 2.4.5). Moreover, $\mathcal{R}_k \leftrightarrows \text{QCTL}^*_\text{k}$ and $\mathcal{R}_k \leftrightarrows \text{EQCTL}^*_\text{k}$ (cf. Theorem 2.3.7 point 4 and Theorem 2.4.8). Since Rabin tree automata are closed under Boolean operations, by applying Theorem 3.2.7, we have that $\text{QLTL}(\text{QCTL}^*_\text{k}) \leftrightarrows \mathcal{B}(\mathcal{R}_k)$ and $\text{EQLTL}(\text{EQCTL}^*_\text{k}) \leftrightarrows \mathcal{B}(\mathcal{R}_k)$[1]. From Theorem 4.1.13, we conclude that $\text{QLTL}(\text{QCTL}^*_\text{k}) \leftrightarrows \text{MSO}_\mathcal{P}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$ and $\text{EQLTL}(\text{EQCTL}^*_\text{k}) \leftrightarrows \text{MSO}_\mathcal{P}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$. Hence we have proved the following.

Theorem 4.1.14 (*Expressiveness of* $\text{QLTL}(\text{QCTL}^*_\text{k})$ *and* $\text{EQLTL}(\text{EQCTL}^*_\text{k})$)

$\text{QLTL}(\text{QCTL}^*_\text{k})$ *and* $\text{EQLTL}(\text{EQCTL}^*_\text{k})$ *are expressively equivalent to* $\text{MSO}_\mathcal{P}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$, *when interpreted over DULSs.*

By way of example, the property 'P holds everywhere on every even tree' can be encoded in $\text{EQLTL}(\text{CTL}^*_\text{k})$ as follows:

$$\exists Q(Q \wedge \bigcirc \neg Q \wedge \Box(Q \leftrightarrow \bigcirc \bigcirc Q) \wedge \Box(Q \rightarrow \mathbf{AG}P)).$$

[1]It is worth noting that the partition step in Theorem 3.2.7 for temporal formulas in EQCTL^*_k generates formulas of the form $\neg\exists Q_1 \ldots \exists Q_n \varphi$, with φ a CTL^*_k-formula, which are not in the language of EQCTL^*_k, since the latter is not closed under negation. However, formulas of the form $\neg\exists Q_1 \ldots \exists Q_n \varphi$ can still be embedded into Rabin tree automata. The Rabin tree automaton for $\neg\exists Q_1 \ldots \exists Q_n \varphi$ is obtained by taking the complementation of the projection, with respect to $Q_1, \ldots, Q_n$, of the Rabin tree automaton for φ.

It is worth noting that this property cannot be expressed in PLTL(CTL$^*_\mathrm{k}$), since PLTL cannot express the property 'P holds on every even point' [125]. Similarly, the property 'P holds everywhere on every even domain' can be encoded in PLTL(EQCTL$^*_\mathrm{k}$) as follows:

$$\Box\exists Q(Q \wedge \mathbf{AX}\neg Q \wedge \mathbf{AG}(Q \leftrightarrow \mathbf{AXAX}Q) \wedge \mathbf{AG}(Q \rightarrow P)).$$

It is worth noting that this property cannot be expressed in PLTL(CTL$^*_\mathrm{k}$).

Since MSO$_\mathcal{P}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$ is decidable, both QLTL(QCTL$^*_\mathrm{k}$) and EQLTL(EQCTL$^*_\mathrm{k}$) are decidable. Moreover, the latter is *elementarily* decidable.

Theorem 4.1.15 (*Complexity of* EQLTL(EQCTL$^*_\mathrm{k}$))

The satisfiability problem for EQLTL(EQCTL$^*_\mathrm{k}$) *over DULSs is in ELEMENTARY.*

Proof.

The proof is similar to that of Theorem 4.1.11. An embedding into $\mathcal{B}(\mathcal{R}_k)$-automata, as described in the proof of Theorem 3.2.7, is used to decide EQLTL(EQCTL$^*_\mathrm{k}$). Note that EQLTL can be elementarily embedded into Büchi sequence automata. Indeed, given an EQLTL-formula $\exists Q_1 \ldots \exists Q_n \varphi$, the PLTL-formula φ can be converted into a Büchi sequence automaton A_φ of size $O(2^{|\varphi|})$. A Büchi sequence automaton for $\exists Q_1 \ldots \exists Q_n \varphi$ can be obtained by taking the projection of A_φ with respect to letters $Q_1, \ldots, Q_n$, that is, by deleting letters $Q_1, \ldots, Q_n$ from the transitions of A_φ. The resulting automaton is of size $O(2^{|\varphi|})$. Similarly, EQCTL$^*_\mathrm{k}$-formulas can be embedded into Rabin tree automata with a doubly exponential number of states and a singly exponential number of accepting pairs in the length of the formula. Moreover, as previously pointed out, the partition step in Theorem 3.2.7 for temporal formulas in EQCTL$^*_\mathrm{k}$ generates formulas of the form $\neg\exists Q_1 \ldots \exists Q_n \varphi$ which are not in the language of EQCTL$^*_\mathrm{k}$. However, a Rabin tree automaton for $\neg\exists Q_1 \ldots \exists Q_n \varphi$ is obtained by taking the complementation of the projection, with respect to $Q_1, \ldots, Q_n$, of the Rabin tree automaton for φ. The resulting automaton for $\neg\exists Q_1 \ldots \exists Q_n \varphi$ has elementary size. Hence, any EQLTL(EQCTL$^*_\mathrm{k}$) can be converted into an equivalent $\mathcal{B}(\mathcal{R}_k)$-automaton of elementary size. Since $\mathcal{B}(\mathcal{R}_k)$-automata are elementarily decidable, we have the thesis. ■

Summing up, we have that EQLTL(EQCTL$^*_\mathrm{k}$) is elementarily decidable and expressively equivalent to MSO$_\mathcal{P}[<_1, <_2, (\downarrow_i)_{i=0}^{k-1}]$ over DULSs.

4.2 n-layered structures

In this section we define temporal logics (Section 4.2.1) and finite-state automata (Section 4.2.2) for n-LSs. We study their expressive power with respect to that of monadic theories over n-LSs as well as their decidability and complexity.

4.2.1 Temporal logics for n-LSs

Temporal logics for n-LSs can be defined as done for DULSs. Let $\mathcal{F}_k(\mathcal{P})$ be the set of $\mathcal{P}$-labeled finite k-ary trees $(D, (\downarrow_i)_{i=0}^{k-1}, <_{pre}, V)$ with D a finite k-ary tree domain and $V : D \rightarrow 2^\mathcal{P}$. Moreover, let $\mathcal{F}_k^n(\mathcal{P})$ be the set of $\mathcal{P}$-labeled finite complete k-ary trees of height n. Let $\mathcal{S}(\mathcal{F}_k(\mathcal{P}))$ be the set of infinite sequences of $\mathcal{P}$-labeled finite k-ary trees, that is, temporalized models $(\mathbb{N}, <, g)$ where $g : W \rightarrow \mathcal{F}_k(\mathcal{P})$ is a total function mapping worlds in W into $\mathcal{P}$-labeled finite k-ary trees in $\mathcal{F}_k(\mathcal{P})$. Similarly $\mathcal{S}(\mathcal{F}_k^n(\mathcal{P}))$ is defined. It is immediate to see that $\mathcal{P}$-labeled n-LSs correspond to tree sequences in $\mathcal{S}(\mathcal{F}_k^n(\mathcal{P}))$, and vice versa. Accordingly, we can use temporalized logics $\mathbf{T_1}(\mathbf{T_2})$, where $\mathbf{T_1}$ is a linear time logic and $\mathbf{T_2}$ is a branching time logic, to express properties of n-LSs. In particular, the temporalized logic PLTL($\mathrm{CTL}^*_\mathrm{k}$), interpreted over (temporalized models corresponding to) n-LSs, is expressively equivalent to the monadic path logic theory of n-LSs. Here, we give a strong interpretation to the next operators of $\mathrm{CTL}^*_\mathrm{k}$: $\mathbf{X}\varphi$ holds true on a node u iff u has a successor v and φ holds on v. Similarly for $\mathbf{X_i}$, for $i = 0, \ldots, k-1$. Consequently, $\mathbf{X}\texttt{true}$ holds true on u iff u has a successor, and $\neg\mathbf{X}\texttt{true}$ holds true on u iff u is a leaf. The proof of Theorem 4.2.1 is similar to that of Theorem 4.1.10.

Theorem 4.2.1 (*Expressiveness of* PLTL($\mathrm{CTL}^*_\mathrm{k}$))

PLTL($\mathrm{CTL}^*_\mathrm{k}$) *is expressively equivalent to* $\mathrm{MPL}_\mathcal{P}[<, (\downarrow_i)_{i=0}^{k-1}]$, *when interpreted over n-LSs.*

By way of example, consider a 2-layered binary structure. The property 'P holds on every even point of the finest domain T^1' can be encoded in PLTL($\mathrm{CTL}^*_\mathrm{k}$) as follows:

$$\Box\mathbf{E}\mathbf{X_0}P.$$

However, the property 'P holds on every even point of the coarsest domain T^0' cannot be expressed in PLTL($\mathrm{CTL}^*_\mathrm{k}$), since PLTL cannot express the property 'P holds on every even point' [125].

Since $\mathrm{MPL}_\mathcal{P}[<, (\downarrow_i)_{i=0}^{k-1}]$ over n-LSs is decidable, we have that PLTL($\mathrm{CTL}^*_\mathrm{k}$) over n-LSs is decidable too. In Section 4.2.2 we will study the complexity of PLTL($\mathrm{CTL}^*_\mathrm{k}$) and we will introduce a decidable extension of PLTL($\mathrm{CTL}^*_\mathrm{k}$) that is expressively equivalent to the full second-order theory of n-LSs.

4.2.2 Automata for n-LSs

In Section 4.2.1 we have showed that n-LSs correspond to (particular) temporalized models embedding trees into sequences. Hence, we can use temporalized automata in the class $\mathcal{A}_\mathbf{1}(\mathcal{A}_\mathbf{2})$, where $\mathcal{A}_1$ is a sequence automata class and $\mathcal{A}_2$ is a tree automata class, to express properties of n-LSs. Consider the temporalized automata class $\mathcal{B}(\mathcal{D}_k)$ embedding finite k-ary tree automata (cf. Definition 2.3.5) into Büchi sequence automata (cf. Definition 2.3.6). We call an automaton in $\mathcal{B}(\mathcal{D}_k)$ a *finite tree sequence automaton*. Since both $\mathcal{B}$ and $\mathcal{D}_k$ are effectively closed under Boolean operations and are decidable, so is the class $\mathcal{B}(\mathcal{D}_k)$ of finite tree sequence automata (cf. Theorems 3.2.4 and 3.2.5). However, finite tree sequence automata do not fit our purposes, since they accept tree sequences in $\mathcal{S}(\mathcal{F}_k(\Sigma))$, while n-LSs correspond to tree sequences in $\mathcal{S}(\mathcal{F}_k^n(\Sigma))$. Nevertheless, $\mathcal{S}(\mathcal{F}_k^n(\Sigma)) \subset \mathcal{S}(\mathcal{F}_k(\Sigma))$, and automata in $\mathcal{B}(\mathcal{D}_k)$ work over trees sequences in $\mathcal{S}(\mathcal{F}_k^n(\Sigma))$ as well. We show that $\mathcal{B}(\mathcal{D}_k)$-automata are closed under Boolean operations over the set $\mathcal{S}(\mathcal{F}_k^n(\Sigma))$. More precisely, let $A, B \in \mathcal{B}(\mathcal{D}_k)$. We show that:

- Complementation: there is $C \in \mathcal{B}(\mathcal{D}_k)$ such that

$$\mathcal{L}(C) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma)) = \mathcal{S}(\mathcal{F}_k^n(\Sigma)) \setminus \mathcal{L}(A);$$

- Union: there is $C \in \mathcal{B}(\mathcal{D}_k)$ such that

$$\mathcal{L}(C) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma)) = (\mathcal{L}(A) \cup \mathcal{L}(B)) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma));$$

- Intersection: there is $C \in \mathcal{B}(\mathcal{D}_k)$ such that

$$\mathcal{L}(C) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma)) = (\mathcal{L}(A) \cap \mathcal{L}(B)) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma)).$$

As for complementation, let $C = \overline{A}$. We have:

$$\begin{array}{ll}
\mathcal{L}(C) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma)) & = \\
\mathcal{L}(\overline{A}) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma)) & = \\
(\mathcal{S}(\mathcal{F}_k(\Sigma)) \setminus \mathcal{L}(A)) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma)) & = \\
(\mathcal{S}(\mathcal{F}_k(\Sigma)) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma))) \setminus (\mathcal{L}(A) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma))) & = \\
\mathcal{S}(\mathcal{F}_k^n(\Sigma)) \setminus (\mathcal{L}(A) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma))) & = \\
\mathcal{S}(\mathcal{F}_k^n(\Sigma)) \setminus \mathcal{L}(A) &
\end{array}$$

Similarly, $C = A \cup B$ in the case of union and $C = A \cap B$ in the case of intersection.

The following theorem proves that finite tree sequence automata and monadic second-order logic have the same expressive power over n-LSs.

Theorem 4.2.2 (*Expressiveness of finite tree sequence automata*)

Finite tree sequence automata are expressively equivalent to monadic second-order logic over n-LSs.

Proof.

We prove that:

1. for every automaton $A \in \mathcal{B}(\mathcal{D}_k)$ over $\Gamma(\Sigma)$ there is a formula $\varphi_A \in \text{MSO}_{\mathcal{P}_\Sigma}[<, (\downarrow_i)_{i=0}^{k-1}]$ such that $\mathcal{L}(A) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma)) = \mathcal{M}(\varphi_A)$;

2. for every formula $\varphi \in \text{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ there is an automaton $A_\varphi \in \mathcal{B}(\mathcal{D}_k)$ over some $\Gamma(2^{\mathcal{P}})$ such that $\mathcal{M}(\varphi) = \mathcal{L}(A_\varphi) \cap \mathcal{S}(\mathcal{F}_k^n(\mathcal{P}))$.

The proof is similar to that of Theorem 4.1.13. The embedding of automata into formulas is performed by encoding the combined acceptance condition for $\mathcal{B}(\mathcal{D}_k)$-automata into $\text{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$. The embedding of formulas into automata takes advantage of the closure properties of $\mathcal{B}(\mathcal{D}_k)$-automata over n-LSs. ■

We now study the decidability of the emptiness problem for $\mathcal{B}(\mathcal{D}_k)$-automata over $\mathcal{S}(\mathcal{F}_k^n(\Sigma))$. More precisely, given a $\mathcal{B}(\mathcal{D}_k)$-automaton A, we study the following problem: is there a tree sequence $t \in \mathcal{S}(\mathcal{F}_k^n(\Sigma))$ such that $t \in \mathcal{L}(A)$? Equivalently, does $\mathcal{L}(A) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma)) \neq \emptyset$? By virtue of Theorem 4.2.2, the described problem can be reduced to the satisfiability problem for the monadic second-order logic over n-LSs, which is known to be nonelementarily decidable. A finer result is the following.

Theorem 4.2.3 *The emptiness problem for finite tree sequence automata over n-LSs is decidable in polynomial time.*

Proof.

We give a polynomial reduction of the the emptiness problem for $\mathcal{B}(\mathcal{D}_k)$-automata over $\mathcal{S}(\mathcal{F}_k^n(\Sigma))$ to the the emptiness problem for $\mathcal{B}(\mathcal{D}_k)$-automata over $\mathcal{S}(\mathcal{F}_k(\Sigma))$. Since Büchi sequence automata and finite tree automata are decidable in polynomial time, the latter is decidable in polynomial time too (cf. Theorem 3.2.5).

Let A be a finite tree sequence automaton. The problem is to verify if $\mathcal{L}(A) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma)) \neq \emptyset$ We build a finite tree sequence automaton F_n of size n recognizing the set $\mathcal{S}(\mathcal{F}_k^n(\Sigma))$ of Σ-labeled sequences of complete k-ary trees of height n. The temporalized automaton $F_n = (\{q\}, q, \Delta, \{q\})$, where $\Delta(q, X, q)$ for every $X \in \Gamma(\Sigma)$. The alphabet $\Gamma(\Sigma) = \{Y_n\}$, where Y_n is a finite tree automaton accepting $\mathcal{F}_k^n$. The automaton $Y_n = (Q, in, \Delta, F)$, where $Q = \{q_1, \ldots, q_n\}$, $in(a, q_1)$ for every $a \in \Sigma$, $\Delta((q_i)^k, a, q_{i+1})$ for every $i = 1, \ldots, n-1$ and every $a \in \Sigma$, and $F = \{q_n\}$. Clearly, Y_n accept a finite tree t iff t is complete, k-ary and of height n. Moreover, F_n accepts the set of infinite sequences of complete, k-ary trees of height n. Note that the size of F_n is n. Hence the problem $\mathcal{L}(A) \cap \mathcal{S}(\mathcal{F}_k^n(\Sigma)) \neq \emptyset$ reduces to the question $\mathcal{L}(A \cap F_n) \neq \emptyset$. ■

A consequence is that PLTL(CTL$_\mathrm{k}^*$) over n-LSs is elementarily decidable. Given a PLTL(CTL$_\mathrm{k}^*$)-formula φ, we construct a $\mathcal{B}(\mathcal{D}_k)$-automaton A_φ as done in the proof of Theorem 3.2.7, and check whether A_φ accepts some tree sequence in $\mathcal{S}(\mathcal{F}_k^n(\Sigma))$ as described in the proof of Theorem 4.2.3. An easy analysis of the complexity of the described algorithm yields the following bound.

Theorem 4.2.4 (*Complexity of* PLTL(CTL$_\mathrm{k}^*$))

The satisfiability problem for PLTL(CTL$_\mathrm{k}^*$) *over n-LSs is in 2EXPTIME.*

As done for DULSs, Theorem 4.2.1 can be extended to the full second-order case by taking advantage of automata. The proof of the following result is similar to that of Theorem 4.1.14

Theorem 4.2.5 (*Expressiveness of* QLTL(QCTL$_\mathrm{k}^*$) *and* EQLTL(EQCTL$_\mathrm{k}^*$))

QLTL(QCTL$_\mathrm{k}^*$) *and* EQLTL(EQCTL$_\mathrm{k}^*$) *are expressively equivalent to* MSO$_\mathcal{P}[<, (\downarrow_i)_{i=0}^{k-1}]$, *when interpreted over n-LSs.*

For instance, consider a 2-layered binary structure. The property 'P holds on every even point of the coarsest domain T^0' can be expressed in EQLTL(CTL$_\mathrm{k}^*$) as follows:

$$\exists Q(Q \wedge \bigcirc\neg Q \wedge \Box(Q \leftrightarrow \bigcirc\bigcirc Q) \wedge \Box(Q \rightarrow P)).$$

Finally, we have the following result.

Theorem 4.2.6 (*Complexity of* EQLTL(EQCTL$_\mathrm{k}^*$))

The satisfiability problem for EQLTL(EQCTL$_\mathrm{k}^*$) *over n-LSs is in 3EXPTIME.*

Proof.

The proof is similar to that of Theorem 4.1.15. EQLTL(EQCTL$_\mathrm{k}^*$)-formulas are embedded into $\mathcal{B}(\mathcal{D}_k)$-automata of size triple exponential in the length of the input formula. This bound holds since nondeterministic finite tree automata can be complemented with a singly exponential blow-up. Finally, the emptiness problem for $\mathcal{B}(\mathcal{D}_k)$-automata over n-LSs can be decided in polynomial time (cf. Theorem 4.2.3). ■

We conclude that EQLTL(EQCTL$_\mathrm{k}^*$) is elementarily decidable and expressively equivalent to MSO$_\mathcal{P}[<, (\downarrow_i)_{i=0}^{k-1}]$ over n-LSs.

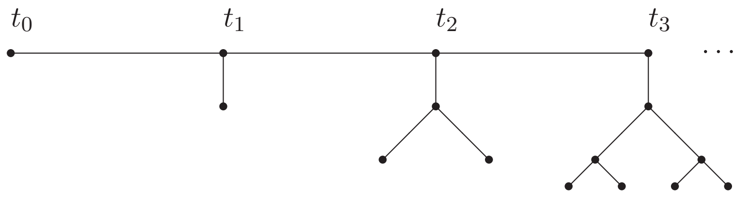

Figure 4.3: A increasing binary tree sequence.

4.3 Upward unbounded layered structures

In this section we define temporal logics and finite-state automata for UULSs. We investigate both the expressive power and the complexity of the defined tools, and relate them with the monadic theories interpreted over UULSs. In Section 4.3.1 we define a temporal logic counterpart for the path fragment of the monadic second-order theory of UULSs. In Section 4.3.2 we define finite-state automata for the second-order theory of UULSs. Moreover, we take advantage of the defined automata class to devise an elementarily decidable temporal logic counterpart of the full second-order theory of DULSs.

4.3.1 Temporal logics for UULSs

In this section we define a temporal logics for UULSs and study its expressive power. The complexity of the temporal logic for UULS will be studied in Section 4.3.2.

We start by giving an alternative characterization of UULSs in terms of tree sequences. To this end, we need to introduce the notions of almost k-ary tree and of increasing tree sequence. For $k \geq 2$, an *almost k-ary finite tree* is a complete finite tree whose root has exactly $k-1$ sons $0, \ldots, k-2$, each of them is the root of a finite (complete) k-ary tree. Let $\mathcal{H}_k(\mathcal{P})$ be the set of $\mathcal{P}$-labeled almost k-ary finite trees. A $\mathcal{P}$-labeled *increasing k-ary tree sequence* (ITS, for short) is a tree sequence such that, for every $i \in \mathbb{N}$, the i-th tree of the sequence is a $\mathcal{P}$-labeled almost k-ary tree of height i (cf. Figure 4.3). A $\mathcal{P}$-labeled ITS can be represented as a temporalized model $(\mathbb{N}, <, g)$ such that, for every $i \in \mathbb{N}$, $g(i)$ is the i-th tree of the sequence. Let $ITS_k(\mathcal{P})$ be the set of $\mathcal{P}$-labeled k-ary ITSs. It is worth noting that $ITS_k(\mathcal{P})$ is *not* the temporalized class of models embedding almost k-ary finite trees into infinite sequences. Indeed, $ITS_k(\mathcal{P}) \subsetneq \mathcal{S}(\mathcal{H}_k(\mathcal{P}))$: an increasing tree sequence is a particular sequence of almost k-ary finite trees, but a sequence of almost k-ary finite trees is not necessary increasing.

We show that a $\mathcal{P}$-labeled UULS corresponds to a $\mathcal{P}$-labeled ITS and vice versa. Recall that an UULS can be viewed as an infinite complete k-ary tree generated from the leaves. Intuitively, the tree sequence is obtained starting from the first point of the first layer of the UULS and climbing up along the leftmost path of the structure. The i-th tree in the sequence is obtained by taking the tree rooted at the i-th point of the leftmost path, and by deleting from it the subtree rooted at the leftmost son of its root. More precisely, let t be a k-ary UULS. For every node x in t, we define t_x to be the finite complete k-ary tree rooted at x. For every $i \geq 0$, let $\hat{t}_{0_i}$ be the almost k-ary finite tree obtained from t_{0_i} by deleting, whenever $i > 0$, the subtree $t_{0_{i-1}}$ from it. The ITS $(\mathbb{N}, <, g)$ associated with the UULS t is obtained by defining, for every $i \geq 0$, $g(i) = \hat{t}_{0_i}$. The embedding of a binary UULS into a binary ITS is depicted in Figure 4.4. Similarly, an UULS can be viewed as an ITS.

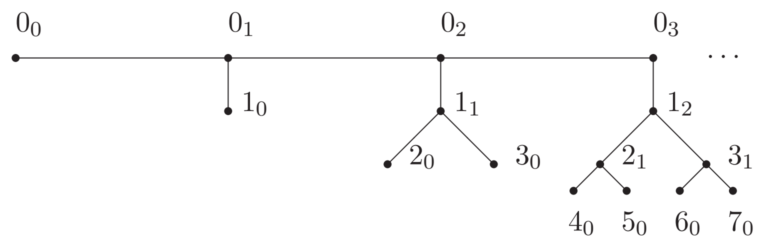

Figure 4.4: Mapping an UULS into an increasing tree sequence.

Since UULSs correspond to (particular) temporalized models embedding trees into sequences, we can use temporalized logics $\mathbf{T_1}(\mathbf{T_2})$, where $\mathbf{T_1}$ is a linear time logic and $\mathbf{T_2}$ is a branching time logic, to express properties of UULSs. More precisely, we interpret $\mathbf{T_1}(\mathbf{T_2})$ over $\mathcal{S}(\mathcal{H}_k(\mathcal{P}))$ but, since we are interested in increasing tree sequences, we study the properties of $\mathbf{T_1}(\mathbf{T_2})$ such as expressiveness and decidability with respect to the set $ITS_k(\mathcal{P})$. In particular, we consider the temporalized logics PLTL(CTL$^*_\mathrm{k}$) and PPLTL(PCTL$^*_\mathrm{k}$). We show that, PLTL(CTL$^*_\mathrm{k}$) and PPLTL(PCTL$^*_\mathrm{k}$) are expressively equivalent to monadic path logic over increasing tree sequences, or, equivalently, over UULSs. To avoid confusion, as done in the case of DULSs, we rename linear temporal operators of PLTL and PPLTL as follows: we write $\triangle$, $\triangle^{-1}$, $\bigcirc$, $\bigcirc^{-1}$, $\Box$, and $\Diamond$ instead of $\mathbf{U}$, $\mathbf{S}$, $\mathbf{X}$, $\mathbf{X}^{-1}$, $\mathbf{G}$, and $\mathbf{F}$, respectively.

It will be convenient to work with a different but equivalent monadic path logic that replaces the total ordering $<$ with a partial ordering $<_{pre}$ defined as follows. Let $t = \langle \mathcal{U}, (\downarrow_i)_{i=0}^{k-1}, < \rangle$ be an UULS. Given $x, y \in \mathcal{U}$, we define $x <_{pre} y$ iff y is different from x and y belongs to the tree t_x rooted at x. The following proposition proves that both $<$ and $<_{pre}$ are redundant in monadic path logic over UULSs.

Proposition 4.3.1 *Both* $\mathrm{MPL}_\mathcal{P}[<, (\downarrow_i)_{i=0}^{k-1}]$ *and* $\mathrm{MPL}_\mathcal{P}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ *are expressively equivalent to* $\mathrm{MPL}_\mathcal{P}[(\downarrow_i)_{i=0}^{k-1}]$ *over UULSs.*

Proof.

We first show how to encode $<_{pre}$ into $\mathrm{MPL}_\mathcal{P}[(\downarrow_i)_{i=0}^{k-1}]$. We have that $x <_{pre} y$ if and only if

$$(x \neq y) \wedge \exists X(x, y \in X \wedge \exists z(\bigvee_{i=0}^{k-1} \downarrow_i(z) = x \wedge z \notin X)).$$

We now show how to write $<$ into $\mathrm{MPL}_\mathcal{P}[(\downarrow_i)_{i=0}^{k-1}]$. We have that $x < y$ if and only if

$$x \in t_{\downarrow_0(y)} \vee y \in \bigcup_{i=1}^{k-1} t_{\downarrow_i(x)} \vee \exists z(x \in t_{\downarrow_0(z)} \wedge y \in \bigcup_{i=1}^{k-1} t_{\downarrow_i(z)}),$$

where $y \in t_{\downarrow_i(x)}$ stands for $\downarrow_i(x) \leq_{pre} y$, and $<_{pre}$ can be encoded into $\mathrm{MPL}_\mathcal{P}[(\downarrow_i)_{i=0}^{k-1}]$ as shown above. ■

It follows that $\mathrm{MPL}_\mathcal{P}[<, (\downarrow_i)_{i=0}^{k-1}]$ and $\mathrm{MPL}_\mathcal{P}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ are expressively equivalent. The same result holds for monadic chain and second-order logics: in the proof, it is sufficient to constrain second-order variables to range over paths. However, we conjecture that it does not hold in monadic first-order logic. In particular, we have that $\mathrm{MFO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}] \rightarrow \mathrm{MFO}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$, since $<$ can be encoded in $\mathrm{MFO}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ as shown in Proposition 4.3.1. We conjecture that the opposite embedding does not hold.

We can reason as in Lemma 4.1.3 to prove that $\mathrm{PLTL}(\mathrm{CTL}^*_\mathrm{k})$ and $\mathrm{PPLTL}(\mathrm{PCTL}^*_\mathrm{k})$ are equivalently expressive. This helps us in proving the expressive equivalence of $\mathrm{PLTL}(\mathrm{CTL}^*_\mathrm{k})$ with respect to $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ over UULSs.

Theorem 4.3.2 (*Expressiveness of* $\mathrm{PLTL}(\mathrm{CTL}^*_\mathrm{k})$)

$\mathrm{PLTL}(\mathrm{CTL}^*_\mathrm{k})$ *is expressively equivalent to* $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ *over UULSs.*

Proof.

The embedding of $\mathrm{PLTL}(\mathrm{CTL}^*_\mathrm{k})$ into $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ is standard and is similar to that of Theorem 4.1.2. The only difference in that PLTL-formulas have to be translated over the leftmost path of the UULS, while $\mathrm{CTL}^*_\mathrm{k}$-formulas have to be translated over almost k-ary trees rooted at nodes in the leftmost path of the UULS. Let us define in $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ the binary predicate $<_1$ as follows: $x <_1 y$ iff $D^0(x) \wedge D^0(y) \wedge y <_{pre} x$, where $\mathtt{D}^0(x)$ iff x belongs to the leftmost path. We know that there exists a standard translation τ_x from PLTL-formulas into equivalent $\mathrm{MFO}_{\mathcal{P}}[<]$-formulas with one free variable x. Let $\hat{\tau}_x$ be the embedding τ_x in which symbol $<$ is replaced by symbol $<_1$. Moreover, we know that there exists a standard translation σ_x from $\mathrm{CTL}^*_\mathrm{k}$-formulas into equivalent $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$-formulas with one free variable x. Let $\hat{\sigma}_x$ be the embedding σ_x in which (i) every atomic formula $x = \downarrow_i(y)$, with $i \in \{0, \ldots, k-2\}$, is replaced by

$$(D^0(y) \rightarrow x = \downarrow_{i+1}(y)) \wedge (\neg D^0(y) \rightarrow x = \downarrow_i(y)),$$

(ii) every atomic formula $x = \downarrow_{k-1}(y)$ is replaced by

$$(D^0(y) \rightarrow x = \perp) \wedge (\neg D^0(y) \rightarrow x = \downarrow_{k-1}(y)),$$

where $\perp$ stands for *undefined* and is a shorthand for $\downarrow_0(0_0)$, and (iii) every atomic formula $x <_{pre} y$ is replaced by

$$(D^0(x) \rightarrow \bigvee_{i=1}^{k-1} \downarrow_i(x) \leq_{pre} y) \wedge (\neg D^0(y) \rightarrow x <_{pre} y).$$

The embedding of $\mathrm{PLTL}(\mathrm{CTL}^*_\mathrm{k})$ into $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ is obtained by combining the embeddings $\hat{\tau}_x$ and $\hat{\sigma}_x$ as in the proof of Theorem 4.1.2.

The opposite direction is more involved but the proof is similar to that of Theorem 4.1.10. We invite the reader to recall proof of Theorem 4.1.10 before proceeding. The proof exploits the same decomposition method we used to prove Theorem 4.1.10: we decompose the model checking problem for an $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$-formula and an increasing tree sequence into a *finite* number of model checking subproblems for formulas and structures that do not refer to the whole tree sequence anymore, but only to certain disjoint components of it. Then, taking advantage of such a decomposition step, we map every $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$-formula into an equivalent (but sometimes much longer) $\mathrm{PLTL}(\mathrm{CTL}^*_\mathrm{k})$-formula.

Let t be an increasing tree sequence, whose trees are denoted by $t_0, t_1, \ldots$, and let r be a node in t. Let t_i be the tree r belongs to. We can decompose the structure t with respect to r into the following disjoint substructures:

(S1) the finite increasing tree sequence from t_0 up to t_{i-1};

(S2) the finite path π from the root of t_i up to and excluding r, accompanied with the finite trees rooted at the sons, not belonging to π, of nodes in π;

(S3) the node r;

(S4) the k finite trees rooted at the k sons of r;

(S5) the infinite increasing tree sequence $t_{i+1}, t_{i+2}, \ldots$.

It is possible to define Ehrenfeucht games over increasing tree sequences and show that two increasing tree sequences satisfy the same formulas in $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ if and only if Duplicator has a winning strategy over them (as done in Theorem 4.1.5). Moreover, it is possible to prove that combining *local* winning strategies on the above defined disjoint parts of two increasing tree sequences it is possible to obtain a *global* winning strategy on the two increasing tree sequences (as done in Lemma 4.1.7). This game-theoretic result permits us to prove two decomposition lemmas for UULSs, corresponding to Lemmas 4.1.8 and 4.1.9 in the case of DULSs. In particular, the first lemma states that, given the formula $\phi(x)$ in $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$, checking $\phi(x)$ in (t, r) is equivalent to verify certain sentences ψ_1 on substructure (S1) of t, ψ_2 on substructure (S2) of t, a label $X \subseteq \mathcal{P}$ on substructure (S3) of t, $\beta_0, \ldots, \beta_{k-1}$ on substructures (S4) of t, and ψ_3 on substructure (S5) of t.

The final step of the proof is the mapping of $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$-formulas into equivalent $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$-formulas. We focus on the two relevant cases: $\varphi = \exists x \phi(x)$ and $\varphi = \exists X \phi(X)$. Let $\varphi = \exists x \phi(x)$. Exploiting the above decomposition lemma, checking $\phi(x)$ in (t, r) is equivalent to verify sentences ψ_1 on (S1), ψ_2 on (S2), $X \subseteq \mathcal{P}$ on (S3), $\beta_0, \ldots, \beta_{k-1}$ on (S4), and ψ_3 on (S5). As done in the proof of Theorem 4.1.10, formula ψ_1 can be mapped into an equivalent $\mathrm{PPLTL}(\mathrm{CTL}^*_{\mathrm{k}})$-formula $(h_1^{-1})'$, formula ψ_2 can be turned into an equivalent $\mathrm{PCTL}^*_{\mathrm{k}}$-formula $(h_2^{-1})'$, formulas β_i can be encoded into equivalent $\mathrm{CTL}^*_{\mathrm{k}}$-formulas p_{b_i}, and formula ψ_3 can be turned into an equivalent $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$-formula $(h_3)'$. The only difference with respect to the proof of Theorem 4.1.10 is that here we need to invoke a version of Theorem 2.4.7 for *finite* trees. Merging together the above results, we have that the given $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$-formula φ is equivalent to the $\mathrm{PPLTL}(\mathrm{PCTL}^*_{\mathrm{k}})$-formula:

$$\Diamond(\bigcirc^{-1}(h_1^{-1})' \wedge \mathbf{EF}p \wedge \bigcirc(h_3)'),$$

where

$$p = \mathbf{X}^{-1}(h_2^{-1})' \wedge \bigwedge_{P \in X} P \wedge \bigwedge_{i=0}^{k-1} \mathbf{EX_i} p_{b_i}.$$

Since $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$ and $\mathrm{PPLTL}(\mathrm{PCTL}^*_{\mathrm{k}})$ are equivalent, we have the proof of this part. The case $\varphi = \exists X \phi(X)$ is similar. ■

For instance, consider a binary UULS. The property 'P holds on every exponential point 0_{2^i} for $i \in \mathbb{N}$ of the finest domain T^0' can be easily encoded in $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$ as follows:

$$\bigcirc \Box \mathbf{EX_1G}((\mathbf{X}\mathtt{true} \rightarrow \mathbf{X_0}\mathtt{true}) \wedge (\neg \mathbf{X}\mathtt{true} \rightarrow P)).$$

It is worth noting that the similar property 'P holds on every point 2^i for $i \in \mathbb{N}$' cannot be expressed in QLTL.

The temporal logic $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$ over UULSs is decidable: given a formula p in $\mathrm{PLTL}(\mathrm{CTL}^*_{\mathrm{k}})$, we translate it into an equivalent formula φ_p of $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$, and we check whether

φ_p is satisfiable using the algorithm for deciding $\mathrm{MPL}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$. However, the complexity of this procedure is nonelementary in the size of the checked formula. In Section 4.3.2, we will give an algorithm for deciding PLTL($\mathrm{CTL}^*_\mathrm{k}$) with elementary complexity. Moreover, we will introduce an elementarily decidable extension of PLTL($\mathrm{CTL}^*_\mathrm{k}$) that is expressively equivalent to the full second-order theory of UULSs.

4.3.2 Automata for UULSs

In this section we define finite-state automata accepting labeled UULSs. We study both the expressive power and the complexity of the defined automata class. Moreover, we exploit automata to devise a elementarily decidable temporal logic counterpart of the full second-order theory of UULSs.

We start by introducing the following definition of finite tree. A set $D \subseteq \mathbb{N}^*$ is a *tree domain* if:

1. D is *prefix closed*, that is, $x \in D$ and $y <_{pre} x$ implies $y \in D$, for every $x, y \in \mathbb{N}^*$;
2. for every $x \in \mathbb{N}^*$, $xi \in D$ implies $xj \in D$ for every $i, j \in \mathbb{N}$ and $j < i$.

A *finite tree* is a relational structure $\langle D, <_{pre}\rangle$, where D is a finite tree domain.

Definition 4.3.3 (*Finite tree automata*)

A (nondeterministic bottom-up) finite tree automaton A *over the alphabet* Σ *is a quadruple* (Q, in, Δ, F) *where* Q *is a finite set of states,* $in \subseteq \Sigma \times Q$ *is an input relation,* $\Delta \subseteq Q^+ \times \Sigma \times Q$ *is a transition relation, and* $F \subseteq Q$ *is a set of final states. Given a* Σ*-labeled finite tree* $t = (D, <_{pre}, V)$*, a* run *of* A *on* t *is a function* $\sigma : D \to Q$ *such that, for every leaf* u *of* t*,* $(V(u), \sigma(u)) \in in$*. Moreover, for every internal node* v *of* t *with* m *sons* $v0, \ldots, v(m-1)$*,* $(\sigma(v0), \ldots, \sigma(v(m-1)), V(v), \sigma(v)) \in \Delta$*. The automaton* A *accepts* t *if there is a run* σ *of* A *on* t *such that* $\sigma(\epsilon) \in F$*. The language accepted by* A*, denoted by* $\mathcal{L}(A)$*, is the set of* Σ*-labeled finite trees accepted by* A*.* □

Finite tree automata are closed under Boolean operations and are decidable in polynomial time [58]. This notion of finite tree is too general for our purposes, since our treelike structures have always a bounded branching degree. Therefore, we introduce the following definition of at most k-ary finite tree. A set $D \subseteq T_k$ is an *at most k-ary tree domain* if:

1. D is *prefix closed*, that is, $x \in D$ and $y <_{pre} x$ implies $y \in D$, for every $x, y \in T_k$;
2. for every $x \in T_k$, $xi \in D$ implies $xj \in D$ for every $i, j \in T_k$ and $j < i$.

An *at most k-ary finite tree* is a relational structure $\langle D, (\downarrow_i)_{i=0}^{k-1}, <_{pre}\rangle$, where D is an at most k-ary finite tree domain, and the rest is as for k-ary finite trees. Let $\mathcal{G}_k(\Sigma)$ the set of at most k-ary finite trees. We introduce the notion of at most k-ary finite tree automaton.

Definition 4.3.4 (*At most k-ary finite tree automata*)

A (nondeterministic bottom-up) at most binary finite tree automaton A *over the alphabet* Σ *is a quadruple* (Q, in, Δ, F) *where* Q *is a finite set of states,* $in \subseteq \Sigma \times Q$ *is an input relation,* $\Delta \subseteq (Q \cup Q^2) \times \Sigma \times Q$ *is a transition relation, and* $F \subseteq Q$ *is a set of final states. Given a* Σ*-labeled at most binary finite tree* $t = (D, \downarrow_0, \downarrow_1, <_{pre}, V)$*, a* run *of* A *on* t *is a function* $\sigma : D \to Q$ *such that, for every leaf* u *of* t*,* $(V(u), \sigma(u)) \in in$*. Moreover, for every internal* *node v of t with two sons $v0$ and $v1$, $(\sigma(v0), \sigma(v1), V(v), \sigma(v)) \in \Delta$, and for every internal node v of t with one son $v0$, $(\sigma(v0), V(v), \sigma(v)) \in \Delta$. The automaton A accepts t if there is a run σ of A on t such that $\sigma(\epsilon) \in F$. The language accepted by A, denoted by $\mathcal{L}(A)$, is the set of Σ-labeled at most binary finite trees accepted by A. At most k-ary finite tree automata, for $k > 2$, can be defined similarly. Let $\mathcal{C}_k$ the class of at most k-ary finite tree automata* □

Note that a k-ary finite tree is a particular at most k-ary finite tree in which the internal nodes have exactly k sons. Moreover, an at most k-ary finite tree is a particular finite tree in which the branching degree is bounded by k. Furthermore, a k-ary finite tree automaton is a particular at most k-ary finite tree automaton such that $\Delta \subseteq Q^k \times \Sigma \times Q$, and an at most k-ary finite tree automaton is a particular finite tree automaton in which $\Delta \subseteq (\bigcup_{i=1}^{k} Q^i) \times \Sigma \times Q$. In particular, there exists a finite tree automaton that accepts the set of Σ-labeled at most k-ary finite trees. Therefore, closure under Boolean operations and decidability of at most k-ary finite tree automata easily follows from the same properties for finite tree automata.

Recall that UULSs correspond to increasing tree sequences. Since increasing tree sequences are (particular) temporalized models embedding trees into sequences, we can use temporalized automata in the class $\mathcal{A}_1(\mathcal{A}_2)$, where $\mathcal{A}_1$ is a sequence automata class and $\mathcal{A}_2$ is a tree automata class, to express properties of UULSs. More precisely, we consider automata in $\mathcal{A}_1(\mathcal{A}_2)$ accepting in $\mathcal{S}(\mathcal{G}_k(\Sigma))$ but, since we are interested in increasing tree sequences, we study the properties of $\mathcal{A}_1(\mathcal{A}_2)$, such as closure under Boolean operations, expressiveness and decidability, with respect to the set $ITS_k(\Sigma)$. In particular, we will consider the temporalized automata class $\mathcal{B}(\mathcal{C}_k)$ embedding at most k-ary finite tree automata (cf. Definition 4.3.4) into Büchi sequence automata (cf. Definition 2.3.6). We call an automaton in $\mathcal{B}(\mathcal{C}_k)$ a *finite at most k-ary tree sequence automaton.*

Since both $\mathcal{B}$ and $\mathcal{C}_k$ are effectively closed under Boolean operations and are decidable, so is the class $\mathcal{B}(\mathcal{C}_k)$ (cf. Theorems 3.2.4 and 3.2.5). We show that $\mathcal{B}(\mathcal{C}_k)$-automata are closed under Boolean operations over the set $ITS_k(\Sigma)$ as well. More precisely, let $A, B \in \mathcal{B}(\mathcal{C}_k)$. We show that:

- Complementation: there is $C \in \mathcal{B}(\mathcal{C}_k)$ such that
$$\mathcal{L}(C) \cap \mathcal{S}(ITS_k(\Sigma)) = \mathcal{S}(ITS_k(\Sigma)) \setminus \mathcal{L}(A);$$
- Union: there is $C \in \mathcal{B}(\mathcal{C}_k)$ such that
$$\mathcal{L}(C) \cap \mathcal{S}(ITS_k(\Sigma)) = (\mathcal{L}(A) \cup \mathcal{L}(B)) \cap \mathcal{S}(ITS_k(\Sigma));$$
- Intersection: there is $C \in \mathcal{B}(\mathcal{C}_k)$ such that
$$\mathcal{L}(C) \cap \mathcal{S}(ITS_k(\Sigma)) = (\mathcal{L}(A) \cap \mathcal{L}(B)) \cap \mathcal{S}(ITS_k(\Sigma)).$$

It is sufficient to set $C = \overline{A}$ in case of complementation, $C = A \cup B$ in the case of union and $C = A \cap B$ in the case of intersection. The following theorem proves that finite at most k-ary tree sequence automata are as expressive as the monadic second-order theory of UULSs.

Theorem 4.3.5 (*Expressiveness of finite at most k-ary tree sequence automata*)

Finite at most k-ary tree sequence automata are expressively equivalent to monadic second-order logic over UULSs.

Proof.

We prove that:

1. for every automaton $A \in \mathcal{B}(\mathcal{C}_k)$ over $\Gamma(\Sigma)$ there is a formula $\varphi_A \in \mathrm{MSO}_{\mathcal{P}_\Sigma}[<, (\downarrow_i)_{i=0}^{k-1}]$ such that $\mathcal{L}(A) \cap ITS_k(\Sigma) = \mathcal{M}(\varphi_A)$;

2. for every formula $\varphi \in \mathrm{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ there is an automaton $A_\varphi \in \mathcal{B}(\mathcal{C}_k)$ over some $\Gamma(2^{\mathcal{P}})$ such that $\mathcal{M}(\varphi) = \mathcal{L}(A_\varphi) \cap ITS_k(\mathcal{P})$.

The proof is similar to that of Theorem 4.1.13. The embedding of automata into formulas is performed by encoding the combined acceptance condition for $\mathcal{B}(\mathcal{C}_k)$-automata into $\mathrm{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$. As done in the proof of Theorem 4.3.2 for temporalized logics, the Büchi acceptance condition have to be implemented over the leftmost path of the structure, and the finite tree automata acceptance condition have to be constrained to hold over almost k-ary trees rooted at nodes in the leftmost path of the structure. The embedding of formulas into automata takes advantage of the closure properties of $\mathcal{B}(\mathcal{C}_k)$-automata over UULSs. ■

With the aid of automata, we are able to extend the results proved in Theorem 4.3.2 to the full second-order theory of UULSs. We know that $\mathcal{B} \leftrightarrows$ QLTL and $\mathcal{B} \leftrightarrows$ EQLTL (cf. Theorem 2.3.7 point 2 and Theorem 2.4.5). Moreover, $\mathcal{C}_k \leftrightarrows \mathrm{QCTL}^*_\mathrm{k}$ and $\mathcal{C}_k \leftrightarrows \mathrm{EQCTL}^*_\mathrm{k}$ (the proof is similar to that of Theorem 2.3.7 point 4 and Theorem 2.4.8). Since finite at most k-ary tree automata are closed under Boolean operations, by applying Theorem 3.2.7, we have that $\mathrm{QLTL}(\mathrm{QCTL}^*_\mathrm{k}) \leftrightarrows \mathcal{B}(\mathcal{C}_k)$ and $\mathrm{EQLTL}(\mathrm{EQCTL}^*_\mathrm{k}) \leftrightarrows \mathcal{B}(\mathcal{C}_k)$ over sequences of at most k-ary finite trees. Since increasing k-ary tree sequences are sequences of at most k-ary finite trees, the above equivalences hold over increasing k-ary tree sequences as well. From Theorem 4.3.5, we conclude that $\mathrm{QLTL}(\mathrm{QCTL}^*_\mathrm{k}) \leftrightarrows \mathrm{MSO}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ and $\mathrm{EQLTL}(\mathrm{EQCTL}^*_\mathrm{k}) \leftrightarrows \mathrm{MSO}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$. Hence we have proved the following.

Theorem 4.3.6 (*Expressiveness of* $\mathrm{QLTL}(\mathrm{QCTL}^*_\mathrm{k})$ *and* $\mathrm{EQLTL}(\mathrm{EQCTL}^*_\mathrm{k})$)

$\mathrm{QLTL}(\mathrm{QCTL}^*_\mathrm{k})$ *and* $\mathrm{EQLTL}(\mathrm{EQCTL}^*_\mathrm{k})$ *are expressively equivalent to* $\mathrm{MSO}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ *over UULSs.*

For example, the property 'P holds on every even point of the leftmost path' can be encoded in $\mathrm{EQLTL}(\mathrm{CTL}^*_\mathrm{k})$ as follows:

$$\exists Q(Q \wedge \bigcirc \neg Q \wedge \square(Q \leftrightarrow \bigcirc \bigcirc Q) \wedge \square(Q \rightarrow P)).$$

It is worth noting that this property cannot be expressed in $\mathrm{PLTL}(\mathrm{CTL}^*_\mathrm{k})$, since PLTL cannot express the property 'P holds on every even point' [125]. However, it seems not to be immediate to codify the property 'P holds on every even point of the finest domain T^0'.

The decidability of $\mathrm{QLTL}(\mathrm{QCTL}^*_\mathrm{k})$ and $\mathrm{EQLTL}(\mathrm{EQCTL}^*_\mathrm{k})$ immediately follows from that of $\mathrm{MSO}_{\mathcal{P}}[<_{pre}, (\downarrow_i)_{i=0}^{k-1}]$ over UULSs.

We focus now on the decidability and complexity of the emptiness problem for finite at most k-ary tree sequence automata over increasing k-ary tree sequences. Such a problem can be formulated as follows: given an automaton $A \in \mathcal{B}(\mathcal{C}_k)$, is there a increasing k-ary tree sequence accepted by A? Equivalently, does $\mathcal{L}(A) \cap ITS_k(\Sigma) \neq \emptyset$? Its (nonelementary) decidability immediately follows from Theorem 4.3.5, since, given an automaton A, we can build an equivalent monadic formula φ_A and check its satisfiability over UULSs. In the following, we give a necessary and sufficient condition that solves the problem in an *elementary* way.

Let $A = (Q, q_0, \Delta, F)$ be an automaton in $\mathcal{B}(\mathcal{C}_k)$ over the alphabet $\Gamma(\Sigma) \subset \mathcal{C}_k$. Clearly, $\mathcal{L}(A) \neq \emptyset$ is necessary for $\mathcal{L}(A) \cap ITS_k(\Sigma) \neq \emptyset$. However, it is not sufficient. By definition of combined acceptance condition for A, we have that $\mathcal{L}(A) \neq \emptyset$ if and only if there is a finite sequence $q_0, q_1, \ldots, q_m$ of distinct states in Q, a finite sequence $X_0, X_1, \ldots, X_m$ of $\mathcal{C}_k$-automata and $j \in \{0, \ldots, m\}$ such that:

1. $\Delta(q_i, X_i, q_{i+1})$, for every $i = 0, \ldots, m-1$, and $\Delta(q_m, X_m, q_j)$;
2. $q_j \in F$;
3. $\mathcal{L}(X_i) \neq \emptyset$, for every $i = 0, \ldots, m$

To obtain a necessary and sufficient condition for $\mathcal{L}(A) \cap ITS_k(\Sigma) \neq \emptyset$, we have to strengthen condition (3) as follows. Let $T^i_k(\Sigma)$ be the set of almost k-ary finite trees of height i:

3'. (3'a) $\mathcal{L}(X_i) \cap T^i_k(\Sigma) \neq \emptyset$, for every $i = 0, \ldots, j-1$, and (3'b) $\mathcal{L}(X_i) \cap T^{i+y\cdot l}_k(\Sigma) \neq \emptyset$, for every $i = j, \ldots, m$ and $y \geq 0$, where $l = m - j + 1$.

The conjunction of conditions (1,2,3') is a necessary and sufficient condition for $\mathcal{L}(A) \cap ITS_k(\Sigma) \neq \emptyset$. We show that conditions (1,2,3') are elementarily decidable. Clearly, there are elementarily many runs in A satisfying conditions (1,2). The following nontrivial Lemma 4.3.7 shows that condition 3' is elementarily decidable.

Lemma 4.3.7 *Let X be a finite at most k-ary tree automaton, and $a, l \geq 0$. Then, the problem $\mathcal{L}(X) \cap T^{a+y\cdot l}_k(\Sigma) \neq \emptyset$, for every $y \geq 0$, is elementarily decidable.*

Proof.
Let $X = (Q, q_{in}, \Delta, F)$ over $\Gamma(\Sigma)$. If $l = 0$, then the problem reduces to checking $\mathcal{L}(X) \cap T^a_k(\Sigma) \neq \emptyset$, for some $a \geq 0$. For every $a \geq 0$, the set T^a_k is finite and hence regular. Since finite at most k-ary tree automata are elementarily closed under Boolean operations and are elementarily decidable, we conclude that in this case the condition is elementarily effective.

Suppose now $l > 0$. For the sake of simplicity, we first give the proof for finite *sequence* automata, and then we discuss how to modify it to cope with the case of finite at most k-ary tree automata. Hence, let X be a finite sequence automaton. We have to give an elementarily effective procedure that checks whether X recognizes at least one sequence of length a, at least one of length $a + l$, at least one of length $a + 2l$, and so on. Without loss of generality, we may assume that the set of final states of X is the singleton containing $q_{fin} \in Q$. Hence, the problem reduces to check, for every $y \geq 0$, the existence of a path from q_{in} to q_{fin} of length $a + y \cdot l$ in the state-transition graph associated with X. We thus need to solve the following problem of Graph Theory, which we call the *Periodic Path Problem* (PPP for short):

> Given a finite directed graph $G = (N, E)$, two nodes $q_1, q_2 \in N$, and two natural numbers $a, l \geq 0$, the question is: for every $y \geq 0$, is there a path in G from q_1 to q_2 of length $a + y \cdot l$?

In the following, we further reduce the PPP to a problem of Number Theory. Let $\Pi_{q_1,q_2}(G)$ be the set of paths from q_1 to q_2 in the graph G. Given $\pi \in \Pi_{q_1,q_2}(G)$, we denote by $\pi^{\circlearrowleft}$ the path obtained by eliminating cyclic subpaths from π. That is, if π is acyclic, then $\pi^{\circlearrowleft} = \pi$. Else, if $\pi = \alpha q'\beta q'\gamma$, then $\pi^{\circlearrowleft} = \alpha^{\circlearrowleft} q'\gamma^{\circlearrowleft}$. Let $\sim_{q_1,q_2}$ be the relation on $\Pi_{q_1,q_2}(G)$ such that $\pi_1 \sim_{q_1,q_2} \pi_2$ if and only if $\pi_1^{\circlearrowleft} = \pi_2^{\circlearrowleft}$. Note that $\sim_{q_1,q_2}$ is an equivalence relation of finite index. For every equivalence class $[\pi]_{\sim_{q_1,q_2}}$, we need a formula expressing the length of a generic path in the class. Note that every path in $[\pi]_{\sim_{q_1,q_2}}$ differs from any other path in the same class only because of some cyclic subpaths. More precisely, let μ be the shortest path in $[\pi]_{\sim_{q_1,q_2}}$, let $C_1, \dots, C_n$ be the the cycles intersecting π, and let $w_1, \dots, w_n$ be their lengths, respectively. Note that μ does not cycle through any C_i. Every path in $[\pi]_{\sim_{q_1,q_2}}$ starts from q_1, cycles an arbitrary number of times (possibly zero) through every C_i, and finally reaches q_2. It is easy to see that the length of an arbitrary path $\sigma \in [\pi]_{\sim_{q_1,q_2}}$ is given by the parametric formula:

$$|\sigma| = |\mu| + \sum_{i=1}^{n} x_i \cdot w_i,$$

where $x_i \geq 0$ in the number of times the path σ cycles through C_i.
Let $[\pi_1]_{\sim_{q_1,q_2}}, \dots, [\pi_m]_{\sim_{q_1,q_2}}$ be the equivalence classes of $\sim_{q_1,q_2}$. For every $j = 1, \dots, m$, let μ_j be the shortest path in $[\pi_j]_{\sim_{q_1,q_2}}$, let $C_1^j, \dots, C_n^j$ be the the cycles intersecting π_j, and let $w_1^j, \dots, w_n^j$ be their lengths, respectively. Moreover, let

$$Y_j = \{y \geq 0 \mid \exists x_1, \dots, x_n \geq 0\,(|\mu_j| + \sum_{i=1}^{n} x_i \cdot w_i^j = a + y \cdot l)\}.$$

The PPP reduces to the following problem of Number Theory:

Do the sets $Y_1, \dots, Y_m$ cover the natural numbers? That is, does $\bigcup_{j=1}^m Y_j = \mathbb{N}$?

We now solve the latter problem. Let $w_i \geq 0$, for $i = 1, \dots, n$. We are interested in the form of the set $S = \{\sum_{i=1}^n x_i \cdot w_i \mid x_i \geq 0\}$. Let $W = (w_1, \dots, w_n)$ and let $d = GCD(W)$ (the greatest common divisor of $\{w_1, \dots, w_n\}$). We distinguish the cases $d = 1$ and $d \neq 1$. If $d = 1$, then it is easy to see that:

$$S = E \cup \{j \mid j \geq k\},$$

where E is a finite set of *exceptions* such that $max(E) < k$, and $k = (w_r - 1) \cdot (w_s - 1)$, with $w_r = min(W)$ (the minimum of $\{w_1, \dots, w_n\}$) and $w_s = min(W \setminus w_r)$. If $d \neq 1$, then consider the set $S' = \{\sum_{i=1}^n x_i \cdot w_i/d \mid x_i \geq 0\}$. Clearly, $GCD(w_1/d, \dots, w_n/d) = 1$ and hence, as above, $S' = E' \cup \{j \mid j \geq k'\}$ for some finite set E' and some $k' \in \mathbb{N}$. Therefore, in this case,

$$S = E' \cdot d \cup \{j \mid j \geq k' \cdot d \,\wedge\, d \,\texttt{div}\, j\},$$

where $a \,\texttt{div}\, b$ stands for a divides b. Summing up, in any case, the set S can be described as follows:

$$S = E \cup \{k + j \cdot d \mid j \in \mathbb{N}\},$$

for some finite (computable) set E, some (computable) $k \in \mathbb{N}$, and $d = GCD(W)$. In other words, the set S is the union of a finite and computable set of exceptions and an arithmetic progression.

Now we consider the equation $\sum_{i=1}^n x_i \cdot w_i = y \cdot l$. Our aim in to describe the set $Y = \{y \geq 0 \mid \exists x_1, \dots, x_n \geq 0\,(\sum_{i=1}^n x_i \cdot w_i = y \cdot l)\}$ in a similar way. Let $e = GCD(d, l)$, $l = l' \cdot e$ and $d = d' \cdot e$. We have that:

$$\begin{array}{ll}
y \in Y & \text{iff} \\
y \cdot l \in S & \text{iff} \\
y \cdot l \in E \,\vee\, y \cdot l \geq k \,\wedge\, d \,\texttt{div}\, y \cdot l & \text{iff} \\
y \cdot l \in E \,\vee\, y \geq \lceil k/l \rceil \,\wedge\, d' \cdot e \,\texttt{div}\, y \cdot l' \cdot e & \text{iff} \\
y \cdot l \in E \,\vee\, y \geq \lceil k/l \rceil \,\wedge\, d' \,\texttt{div}\, y &
\end{array}$$

Therefore, the set Y is the union of a finite and computable set and an arithmetic progression, i.e.,

$$Y = E' \cup \{k' + j \cdot d' \mid j \in \mathbb{N}\},$$

for some finite (computable) set E', some (computable) $k' \in \mathbb{N}$, and $d' = d/GCD(d,l)$. The set $Y = \{y \geq 0 \mid \exists x_1, \ldots, x_n \geq 0\,(\sum_{i=1}^{n} x_i \cdot w_i = a + y \cdot l)\}$, with $a \in \mathbb{N}$, can be described in the same way.

We have shown that, for $i = 1, \ldots, m$, every Y_i has the form $E_i \cup \{k_i + y \cdot d_i \mid y \geq 0\}$ for some finite E_i, and some $k_i, d_i \in \mathbb{N}$. We now give a solution to the problem $\bigcup_{i=1}^{m} Y_i = \mathbb{N}$. Let $k_r = min\{k_1, \ldots, k_m\}$ and $D = LCM(d_1, \ldots, d_m)$ (the lowest common multiple of $\{d_1, \ldots, d_m\}$). The algorithm works as follows: for every $k < k_r$, we check whether $k \in Y_i$ for some $i = 1, \ldots, m$. If this is not the case, the problem has no solution. Otherwise, we verify whether, for every $j = 0, \ldots, D-1$, $k_r + j \in Y_i$ for some $i = 1, \ldots, m$. If this is the case, then we have a solution, otherwise, there is no solution. Note that a solution can be described in terms of an ultimately periodic word $w = uv^{\omega}$, with $u, v \in \{1, \ldots, m\}^*$, such that, for every $i \geq 0$, $w(i) = j$ means that a path from q_1 to q_2 in the graph G belongs to the j-th equivalence class $[\pi_j]_{\sim_{q_1,q_2}}$.

The above algorithm solves the periodic path problem in doubly exponential time in the number n of nodes of the graph G. The number of equivalence classes of the relation $\sim_{q_1,q_2}$ over the set of paths from q_1 to q_2 in G may be exponential in n. Thus, we have m sets $Y_1, \ldots, Y_m$, each associated to a relevant equivalence class, and $m = \mathcal{O}(2^n)$. Every set Y_i can be represented in polynomial time as $E_i \cup \{k_i + y \cdot d_i \mid y \geq 0\}$ for some finite E_i, and some $k_i, d_i \in \mathbb{N}$. Note that the cardinality of E_i is bounded by k_i, $k_i = \mathcal{O}(n^2)$ and $d_i = \mathcal{O}(n)$. The final step of the procedure makes $k_0 + D$ membership tests with respect to some set Y_i, where $k_0 = min\{d_1, \ldots, d_m\}$, and $D = LCM(d_1, \ldots d_m)$. Each test is performed in $\mathcal{O}(1)$. Moreover, D is bounded by ${d_0}^m$, where $d_0 = max\{d_1, \ldots, d_m\}$, and hence $D = \mathcal{O}(2^{2^n})$. Hence, the procedure works in time doubly exponential.

The general case of finite trees is similar. Let X be a finite at most k-ary tree automaton. A path from q_1 to q_2 corresponds to a run of X such that the run tree is complete and k-ary, the root of the run tree is labeled with state q_1 and the leaves of the run tree are labeled with state q_2. A cycle is a path from q to q. The problem is to find, for every $y \geq 0$, a path from the initial state q_{in} to the final state q_{fin} of length $a + y \cdot l$. The rest of the proof follows the same reasoning done for sequence automata. ■

It follows that, given a $\mathcal{B}(\mathcal{C}_k)$-automaton A, we have an algorithm to solve the problem $\mathcal{L}(A) \cap ITS_k(\Sigma) \neq \emptyset$ in time doubly exponential in the size of A.

Theorem 4.3.8 *The emptiness problem for finite at most k-ary tree sequence automata over UULSs is in 2EXPTIME.*

Since PLTL($\mathrm{CTL}^*_{\mathrm{k}}$)-formulas (resp. EQLTL($\mathrm{EQCTL}^*_{\mathrm{k}}$)-formulas) can be converted into $\mathcal{B}(\mathcal{C}_k)$-automata with a doubly (resp. triply) exponential blow-up (as done for $\mathcal{B}(\mathcal{D}_k)$-automata in Section 4.2.2), we have the following.

Theorem 4.3.9 (*Complexity of* $\mathrm{PLTL}(\mathrm{CTL}^*_\mathrm{k})$ *and* $\mathrm{EQLTL}(\mathrm{EQCTL}^*_\mathrm{k})$)

The satisfiability problem for $\mathrm{PLTL}(\mathrm{CTL}^*_\mathrm{k})$ *over UULSs is in 4EXPTIME. The satisfiability problem for* $\mathrm{EQLTL}(\mathrm{EQCTL}^*_\mathrm{k})$ *over UULSs is in 5EXPTIME.*

We conclude that $\mathrm{EQLTL}(\mathrm{EQCTL}^*_\mathrm{k})$ is elementarily decidable and expressively equivalent to $\mathrm{MSO}_\mathcal{P}[<, (\downarrow_i)_{i=0}^{k-1}]$ over UULSs.

A contribution of this section is an alternative characterization of systolic binary tree automata (cf. Definition 2.3.4). Systolic binary tree automata have been proved to be the counterpart of $\mathrm{MSO}_\mathcal{P}[<, \mathtt{flip}]$ over infinite sequences: for every monadic formula there is an equivalent systolic automaton and vice versa. Moreover, $\mathrm{MSO}_\mathcal{P}[<, \mathtt{flip}]$ over infinite sequences is equivalent to $\mathrm{MSO}_\mathcal{P}[<, (\downarrow_i)_{i=0}^{k-1}]$ over UULSs in the following sense: there exists an isomorphism ι between labeled infinite sequences and labeled UULSs such that: (i) for every formula φ in $\mathrm{MSO}_\mathcal{P}[<, \mathtt{flip}]$, there exists a formula ψ in $\mathrm{MSO}_\mathcal{P}[<, (\downarrow_i)_{i=0}^{k-1}]$ with $\iota(\mathcal{M}(\varphi)) = \mathcal{M}(\psi)$, (ii) for every formula φ in $\mathrm{MSO}_\mathcal{P}[<, (\downarrow_i)_{i=0}^{k-1}]$, there exists a formula ψ in $\mathrm{MSO}_\mathcal{P}[<, \mathtt{flip}]$ with $\mathcal{M}(\varphi) = \iota(\mathcal{M}(\psi))$. Therefore, φ and ψ are equivalent modulo an isomorphism between the corresponding classes of models. Finally, we have proved in this section that $\mathcal{B}(\mathcal{C}_k)$-automata are the counterpart of $\mathrm{MSO}_\mathcal{P}[<, (\downarrow_i)_{i=0}^{k-1}]$ over UULSs: for every monadic formula there is an equivalent temporalized automaton and vice versa. Merging together all these results, we have that any systolic binary tree automaton can be converted into an equivalent temporalized automaton and vice versa (modulo an isomorphism between corresponding structures). In particular, any systolic binary tree automaton can be regarded as a *combined automaton* that embeds finite tree automata, working over finite trees rooted at nodes in the leftmost path of a layered structure, into a Büchi sequence automaton working over the leftmost path of a layered structure.

4.4 Model checking granular reactive systems

In this section we apply the combining approach to model, specify and verify granular reactive systems. In particular, we show how to encode (the state-transition representation of) granular reactive systems into combined models. Accordingly, combined temporal logics and combined automata may be adopted for the specification of requirements for such systems. This makes it possible to apply combined model checking/satisfiablity procedures presented in this thesis to verify granular reactive systems. The main advantage of the combining approach to system analysis is *separability*: different components are treated as different entities during both the modeling and specifying tasks. Other advantages include *modularity* and *flexibility*. Well-known modules, consisting of modeling and specification languages, as well as verification techniques, are combined in such a way that the resulting framework inherits nice features of the components. This improves the reuse of tools and software. Moreover, the result of the combination forms a bag of solutions, instead of a specific tailored one, where each possible solution is obtained by 'plugging' the preferred modules and 'playing' the combined solution.

A *reactive system* (*RS*) consists of a set of processes that execute concurrently and interact with the system environment; the system is supposed to run forever. Temporal logics have been used successfully for modeling and analyzing the behavior of reactive systems [104]. A *granular reactive system* (*GRS*) is a reactive system whose components (processes) have dynamic behaviors regulated by very different—even by orders of magnitude—time constants [92]. As an example, consider a pondage power station consisting of a reservoir, with filling and emptying times of days or weeks, generator units, possibly changing state in a few seconds, and electronic control devices, evolving in milliseconds or even less [17]. A complete specification of the power station must include the description of these components and of their interactions. A natural description of the temporal evolution of the reservoir state will probably use days: "During rainy weeks, the level of the reservoir increases 1 meter a day", while the description of the control devices behavior may use milliseconds: "When an alarm comes from the level sensors, send an acknowledge signal in 50 milliseconds". It is somewhat unnatural, and sometimes impossible, to force the specifier of these systems to use a unique time granularity when describing the behavior of all the components. Indeed, a good specification language must allow the specifier to easily describe all simple and intuitively clear facts. Therefore, a specification language for granular reactive systems must support *different time granularities.* Granular logics, interpreted over many-level temporal structures, have been extensively studied in this Chapter.

A GRS can be viewed as a layered set of system components, each one evolving at a different time granularity. Each *system component* can be regarded as a (flat) reactive system: *component states* are represented as sets of propositions that hold at the state; *component computations* are modeled as infinite sequences of component states. The *component behavior* is a set of component computations, and it is usually represented as a Kripke structure. The set of time granularities of the system (and thus the set of system components) is totally ordered on the basis of the degree of fineness (coarseness) of its elements. As an example, consider the set of time granularities including `weeks`, `days`, `seconds`, and `milliseconds`. Time granularities are ordered as follows: `weeks` $\prec$ `days` $\prec$ `seconds` $\prec$ `milliseconds`, where $\mathtt{g}_1 \prec \mathtt{g}_2$ means that $\mathtt{g}_1$ is coarser than $\mathtt{g}_2$. If $\mathtt{g}_1 \prec \mathtt{g}_2$, then there exists a *conversion factor* between the $\mathtt{g}_1$ and $\mathtt{g}_2$ that allows one to map each time point of $\mathtt{g}_1$ into a set of time points of $\mathtt{g}_2$. For instance, the conversion factor between `weeks` and `days` is 7. The behavior of the whole GRS can be described in terms of the synchronized behavior of its components. A *system state* corresponds to the set of states of all system components at/during an instant of the coarsest time granularity; it can be represented as a tree such that the elements of its i-th layer are the states of the i-th system component. Accordingly, a *system computation* can be defined as an infinite sequence of system states (this is the state-based view). Equivalently, a system computation can be expressed by a layered set of component computations, one computation for each component (this is the component-based view). Finally, the *system behavior* is a set of system computations. We consider the case of GRSs composed of a *finite* number, say n, of system components. Hence, any system computation of such a GRS is an n-layered structure. However, the generalization to the case of GRSs with an infinite number of system components, either downward unbounded (there exists a coarsest component) or upward unbounded (there exists a finest component) is immediate.

The operational behavior of a GRS can be naturally described as a suitable combination of temporal *evolutions* (sequences of component states) and temporal refinements (mapping of a component state into a finite sequence of states belonging to a finer component). According to such a point of view, the model describing the operational behavior of the system and the specification language can be obtained by combining simpler models and languages, respectively, and model checking procedures for combined logics (cf. Section 3.3) can be used. For instance, a temporalized model $\mathcal{M} = (W, R, g)$ can be used in both a component-based and a state-based fashion for model checking purposes. In the *component-based framework*, the frame (W, R) models the granular relationships among the different components: W is a set of components, and $R \subseteq W \times W$ is a linear relation such that $(c_1, c_2) \in R$ if component c_2 is a refinement of component c_1. Moreover, for $c \in W$, the model $g(c)$ is a Kripke structure

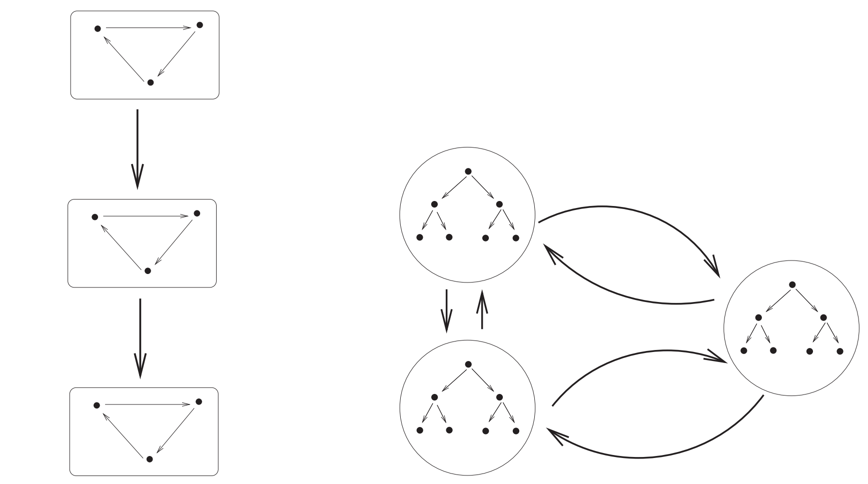
(a) component-based (b) state-based

Figure 4.5: Frameworks for a GRS

that captures the internal behavior of the component c. (Figure 4.5(a)). The language of the temporalized logic $\mathbf{T_1}(\mathbf{T_2})$ is the specification language, where $\mathbf{T_1}$ is a linear time logic and $\mathbf{T_2}$ is a branching time logic. *Component-based properties*, that is, formulas that predicate over temporal evolutions, are easily expressible in this framework. A typical component-based property is the following one: "there exists a computation of a system component during which P always holds". This condition can be expressed in PLTL(CTL) by means of the formula: $\mathbf{F_1 E_2 G_2} P$ (we distinguish PLTL and CTL operators by using different indexes, namely, 1 for PLTL and 2 for CTL).

In the *state-based framework*, the interpretation of the model is different: the frame (W, R) models the evolution of the system states: W is a set of system states, $R \subseteq W \times W$ is a relation such that $(s_1, s_2) \in R$ if state s_2 is reachable from state s_1 in the system. Moreover, for $s \in W$, the model $g(s)$ is the tree representing the system state s (Figure 4.5(b)). Specifications can be written in the language of the temporalized logic $\mathbf{T_1}(\mathbf{T_2})$, where $\mathbf{T_1}$ and $\mathbf{T_2}$ are either linear or branching time logics. As for expressiveness of such a framework, *state-based properties*, which refer to temporal refinements, are immediately expressible. A typical state-based requirement is the following one: "there exists a system state such that every component state belonging to it satisfies P". This condition can be captured in CTL(CTL) by means of the formula: $\mathbf{E_1 F_1 A_2 G_2} P$.

As for the expressive power, both the component-based framework and the state-based one suffer from some limitations. In the component-based framework, it is generally impossible, to specify state-based properties. The reason is that such a framework focuses on the temporal evolution of system components. The case of the state-based framework is dual: it focuses on the temporal refinement of system states; as a consequence, component-based properties are difficult, and often impossible, to capture. To overcome the limitations of both frameworks, it is possible to combine them into a more expressive *bi-temporal framework*. The bi-temporal framework exploits independent combination both to build a model $\mathcal{M} = (W, R_1, R_2, V)$ of the behavior of GRSs, and to obtain a logical specification language $\mathbf{T_1} \oplus \mathbf{T_2}$ to express granular requirements. The connected components of (W, R_1), which are models of $\mathbf{T_1}$, capture the temporal evolution of the system components, while the connected components of (W, R_2), which are models of $\mathbf{T_2}$, represent the temporal refinement of the

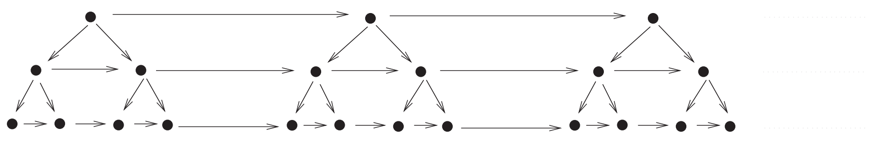
Figure 4.6: Bi-temporal framework for a deterministic GRS

system states. For simplicity, we restrict ourselves to the simplified situation in which all system components are deterministic, and hence the connected components of (W, R_1) are linear structures (Figure 4.6). In the bi-temporal framework, both component-based and state-based requirements can be easily specified. Let us consider the previously introduced examples. The (component-based) property "there exists a computation of a system component during which P always holds" can be specified in PPLTL $\oplus$ CTL as $\mathbf{E_2F_2}(\mathbf{H_1}P \wedge \mathbf{G_1}P)$, while the (state-based) property "there exists a system state such that every component state belonging to it satisfies P" is expressible in PLTL $\oplus$ CTL as $\mathbf{F_1A_2G_2}P$. Obviously, the increase in expressiveness obtained by moving from the simpler temporalized frameworks to the more complex bi-temporal one does not come for free. The task of encoding the system behavior in the bi-temporal framework is indeed much harder than the task of encoding it in the simpler temporalized setting. The reason is that the model used in the bi-temporal framework admits a higher degree of interaction between system components.

One of the motivations for this thesis has been the need to develop model checking frameworks for granular reactive systems and corresponding logics. We have shown that this is indeed possible, using a *divide and conquer* strategy: we first isolate the orthogonal 'simple' entities in which a system can be decomposed. Then, we apply well-known structures and logics to the component entities. Finally, we combine the component entities according to a combining method and we use combined tools to analyze the system. We feel that this divide and conquer approach can be useful to model and analyze many other complex systems, which, inherently, are the composition of simpler entities. One example are mobile reactive systems [14]. A *mobile reactive system* (*MRS*) is a reactive system whose processes (may) reside within a hierarchy of locations (*ambient structure*) and modify it. The system may temporally evolve in two ways: a process may execute one step of computation, or it can move (with its ambient and subambients) inside or outside another ambient. Moreover, a process may dissolve the ambient of another process. The execution of a computation step modifies the program state of the process that executes it, whereas the movement or dissolution of an ambient modifies the ambient structures. Hence, the evolution of an MRS takes place along three orthogonal dimensions: the *temporal evolution* of processes, modeled by sequences of (program) states, the temporal evolution of ambients, modeled by sequences of ambient structures, and the *spatial distribution* of processes within the ambient structures. In [50], we adopted combined models and logics to model the behavior and the requirements of MRSs, respectively, and we used combined model checking techniques to verify them.

4.5 Discussion

In this chapter we have often approached the problem of finding the temporal logic counterpart $\mathbf{T}$ of a monadic theory $\mathcal{L}$. We solved the problem with two different techniques: (1) the *decomposition method*; (2) the *automata reduction method*. The decomposition method consists in dividing the model checking problem for $\mathcal{L}$-formula with respect to $\mathcal{L}$-structures into a finite number of model checking subproblems for $\mathcal{L}$-formulas with respect to certain disjoint substructures of the original structure. Such a result is obtained by exploiting Ehrenfeucht games: local strategies, played over the disjoint substructures, can be composed to form a global strategy over the whole structure. Taking advantage of such a decomposition step, every $\mathcal{L}$-formula is converted into an equivalent $\mathbf{T}$-formula.

Instead of trying to establish a direct correspondence between monadic theories and temporal logics, the automata reduction method connects them via automata. The first step is to find the automata-theoretic counterpart $\mathcal{A}$ of temporal logic $\mathbf{T}$. This step is relatively easy, since, in general, automata and temporal logics represent two elementarily comparable *normal forms* of monadic theories. The second step is to prove that $\mathcal{A}$ is expressively equivalent to $\mathcal{L}$. The mapping of monadic formulas into automata (the difficult direction) is greatly benefit from automata closure properties. Composing these two steps, one obtains that $\mathcal{T}$ is expressively equivalent to $\mathcal{L}$.

The decomposition and the automata reduction methods are orthogonal: the former works well for monadic first order or well-behaved second-order (like path) theories, but does not naturally extend to the full second-order case. The reason is that full second-order quantification ranges over arbitrary sets, which can spread all over the structure without any control. This is in contrast with the decomposition strategy. The latter fits well to monadic full second-order theories, since automata correspond to second-order formulas.

5 Extending the picture

In the previous chapters, the language for time granularity we focused on is the monadic second-order logic in the signature with a total ordering $<$ and k projection functions $\downarrow_0, \ldots, \downarrow_{k-1}$. The reader may wonder about the motivations for such a choice. In particular, is this language expressive enough to model interesting properties of time granularity? Moreover, can this language, or some of its fragments, be extended with other significant predicates while preserving decidability? In this chapter we try to answer these natural questions. We will consider rather natural predicates, such as the equi-level (resp. equi-column) predicate constraining two time points to belong to the same layer (resp. column) as well as the horizontal (resp. vertical) successor predicate relating a time point to its successor within a given layer (resp. column). We call them *global* predicates, because they do not refer to any particular layer or column. We will consider also the corresponding *local* predicates, which, on the contrary, refer to a fixed layer or column.

In [92], among the many possible relations between time points belonging to the layered temporal universe, the above local predicates are identified as the *primitives* for time granularity, that is, relations that any specification language for time granularity should be able to express. Global predicates are a natural generalization of local ones. In the context of time granularity, the utility of the equi-level is clear: it is a contextualization predicate that permits to constrain the validity of a formula over a single (unspecified) domain. On the other hand, the equi-column predicate may be useful to express time constraints such as the uniform occurrence of a give event after the same number of ticks of the component clocks.

It is worth mentioning that the problem of the decidability of monadic theories extended with equi-level and vertical successor predicates has been already studied in the literature. In particular, the decidability of the first-order theory of two successors, devoid of free set variables, over infinite binary trees, extended with the equi-level predicate, was first proved by Elgot and Rabin in [33]. Thomas extended this result by showing that monadic chain logic extended with the equi-level predicate over infinite k-ary trees is decidable [115]. Finally, Läuchli and Savoiz proved the undecidability of (weak) monadic second-order theory of k successors over infinite k-ary trees extended with either the equi-level or the vertical successor predicate [85]. This chapter extends the above results in two ways: first, new predicates are introduced, such as the horizontal successor and the equi-column predicates; second, monadic theories are interpreted over more general structures, such as DULSs and UULSs.

We will show that every local predicate is definable in $\mathrm{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ over layered

$$\begin{tikzcd}
& \mathrm{MSO}_{\mathcal{P}} \arrow[dl, shift right] & & & \\
\mathrm{MSO} \arrow[ur, shift right] & & \mathrm{MCL}_{\mathcal{P}} \arrow[ul] & & \\
& \mathrm{MCL} \arrow[ul] \arrow[ur] & & \mathrm{MPL}_{\mathcal{P}} \arrow[ul] & \\
& & \mathrm{MPL} \arrow[ul] \arrow[ur] & & \mathrm{MFO}_{\mathcal{P}} \arrow[ul] \\
& & & \mathrm{MFO} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Figure 5.1: A hierarchy of monadic theories.

structures. Since local predicates correspond to the *primitives* for time granularity, that is, relations that any specification language for time granularity should be able to express [92], we claim that $\mathrm{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ is expressive enough. Moreover, we will show that global predicates are not definable in $\mathrm{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ over DULSs and UULSs, and, even worse, their addition to such theories immediately leads to undecidability. We also point out the different status of the equi-level and equi-column predicates with respect to the theory of n-layered structures: while the first one is definable, the second one is undefinable and its addition yields undecidability. These negative results do not prevent the possibility of studying the effect of the addition of global predicates to *fragments* of the monadic second-order theories of time granularity, such as their first-order, path, and chain fragments, possibly devoid of monadic predicates in $\mathcal{P}$. Indeed, a predicate may be undefinable in a decidable logic, but its addition to the logic may preserve decidability. We systematically explore all the possibilities, and give a number of positive and negative results. From a technical point of view, (un)definability and (un)decidability results are obtained by reduction from/to a wide spectrum of undecidable/decidable problems. Some of the resulting decidability problems are open. However, the achieved results suffice to formulate some general statements. We prove that equi-level, horizontal successor, and vertical successor predicates can be added to monadic first-order, path, and chain fragments, devoid of monadic predicates in $\mathcal{P}$, preserving decidability. We do not know yet whether the same holds for the equi-column predicate or not. However, we know that the addition of the equi-column predicate to monadic first-order fragments over ω-layered structures, with monadic predicates in $\mathcal{P}$, makes the resulting theory undecidable. As for the equi-level predicate, we know that adding it to the monadic path fragment over DULSs, with monadic predicates in $\mathcal{P}$, leads to undecidability. Even though we do not have yet a formal argument, these results induce us to conjecture that the complexity of the equi-column predicate is higher than the complexity of the equi-level predicate. Finally, as far as the monadic second-order theory over UULSs is concerned, we establish an interesting connection between its extension with the equi-level (resp. equi-column) predicate and systolic Y-tree (resp. trellis) automata.

5.1 Local and global predicates

Given a finite set $\mathcal{P}$ of monadic predicates, recall that $\mathrm{MSO}_{\mathcal{P}}[\tau]$ is defined as $\mathrm{MSO}[\tau \cup \mathcal{P}]$ (cf. Definition 2.2.1). We denote by $\mathrm{MSO}[\tau]$ the monadic second-order language devoid of monadic predicates in $\mathcal{P}$. Moreover, let $\mathrm{MFO}[\tau]$, $\mathrm{MPL}[\tau]$ and $\mathrm{MCL}[\tau]$ be the first-order, path and chain fragments of $\mathrm{MSO}[\tau]$. Figure 5.1 summarizes the relationships between the expressive power of all the defined monadic languages over layered structures (an arrow

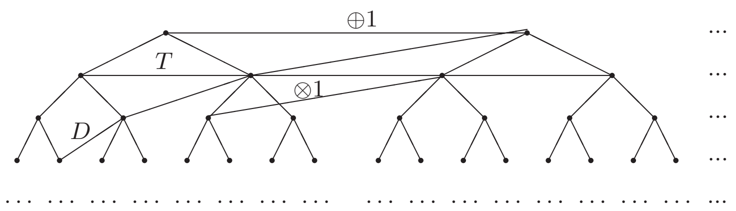

Figure 5.2: The global predicates for time granularity.

from A to B means that $A \to B$). Note that $\mathrm{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ is expressively equivalent to $\mathrm{MSO}[<, (\downarrow_i)_{i=0}^{k-1}]$. Indeed, any monadic predicate $P \in \mathcal{P}$ may be regarded as a free set variable. Since $\mathrm{MSO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ over layered structures is decidable, all the theories in Figure 5.1, when interpreted over layered structures, are decidable.

In the following, we investigate the possibility of defining meaningful binary predicates within the monadic theories, and, whenever this is not possible, the possibility of adding such predicates preserving decidability. Let $\mathcal{U}$ be the layered temporal universe. We focus on the following predicates over $\mathcal{U}$. Let $n_r, m_s \in \mathcal{U}$:

1. *equi-level predicate* T, such that $T(n_r, m_s)$ iff $r = s$;
2. *i-th equi-level predicate* T^i, such that $T^i(n_r, m_s)$ iff $r = s = i$;
3. *equi-column predicate* D, such that $D(n_r, m_s)$ iff $n = m$;
4. *i-th equi-column predicate* D^i, such that $D^i(n_r, m_s)$ iff $n = m = i$;
5. *horizontal successor* $+1$, such that $+1(n_r, m_s)$ iff $r = s$ and $m = n + 1$;
6. *i-th horizontal successor* $+_i1$, such that $+_i1(n_r, m_s)$ iff $r = s = i$ and $m = n + 1$;
7. *vertical successor* $\oplus 1$, such that $\oplus 1(n_r, m_s)$ iff $n = m$ and $s = r + 1$;
8. *i-th vertical successor* $\oplus_i 1$, such that $\oplus_i 1(n_r, m_s)$ iff $n = m = i$ and $s = r + 1$.

A layered structure extended with the global predicates is depicted in Figure 5.2. We will sometimes use the functional notation for the above predicates. For instance, we will write $T(x) = y$ for $T(x, y)$. Moreover, we shall write $T^i(x)$ as a shorthand for $T^i(x, x)$ (x belongs to the i-th layer). Predicates $+_i1$ and T^i are inter-definable in $\mathrm{MSO}[<, (\downarrow_i)_{i=0}^{k-1}]$ over layered structures:

$$\begin{array}{lcl} +_i1(x,y) & = & x < y \wedge T^i(x,y) \wedge \forall z((T^i(x,z) \wedge x < z) \to y \leq z) \\ T^i(x,y) & = & \exists w(+_i1(x,w)) \wedge \exists w(+_i1(y,w)) \end{array}$$

Similarly, $+1$ and T are inter-definable in $\mathrm{MSO}[<, (\downarrow_i)_{i=0}^{k-1}]$ as follows:

$$\begin{array}{lcl} +1(x,y) & = & x < y \wedge T(x,y) \wedge \forall z((T(x,z) \wedge x < z) \to y \leq z); \\ T(x,y) & = & \forall X(x \in X \wedge \forall z(z \in X \to \exists w(+1(z,w) \wedge w \in X)) \to y \in X) \vee \\ & & \forall X(y \in X \wedge \forall z(z \in X \to \exists w(+1(z,w) \wedge w \in X)) \to x \in X) \end{array}$$

As a consequence, we have that $+_i1$ (resp. $+1$) is definable in $\mathrm{MSO}[<, (\downarrow_i)_{i=0}^{k-1}]$ if and only if T^i (resp. T) is definable in $\mathrm{MSO}[<, (\downarrow_i)_{i=0}^{k-1}]$. In a similar way, it can be shown the inter-definability of the pairs of predicates $(\oplus_i 1, D^i)$, and $(\oplus 1, D)$.

We proceed as follows. For any predicate β, we try to define it in MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$ over layered structures. In the positive case (β is definable), we investigate its definability into MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$ fragments, possibly extended with monadic predicates $P \in \mathcal{P}$. In the negative case (β is not definable), we study the decidability of the extensions of MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$ and of its fragments with β.

5.2 Definability and decidability over n-LSs

In this section we explore definability and decidability of local and global predicates with respect to the monadic theories of n-LSs. We start by defining the local predicates. Let $\downarrow(x) = y$ be a shorthand for $\bigvee_{j=0}^{k-1} \downarrow_j(x) = y$. The i-th equi-level predicate T^i can be inductively defined as follows:

$$\begin{array}{lcl} T^0(x,y) & = & \neg\exists z_1(\downarrow(z_1) = x) \wedge \neg\exists z_2(\downarrow(z_2) = y); \\ T^{i+1}(x,y) & = & \exists z_1 \exists z_2(T^i(z_1,z_2) \wedge \downarrow(z_1) = x \wedge \downarrow(z_2) = y). \end{array}$$

The equi-level predicate $T(x,y)$ can be defined as $\bigvee_{i=0}^{n-1} T^i(x,y)$. Horizontal successors $+_i1$ and $+1$ are definable in terms of T^i and T, respectively. We consider now the equi-column predicate D^i. Let $+_ij(x,y)$ be a shorthand for $(+_i)^j(x) = y$. Given $w \in \{0, \ldots, k-1\}^*$, we inductively define $\downarrow_w(x)$ as follows: if $|w| = 0$, then $\downarrow_w(x) = x$, otherwise, for $w = av$, we define $\downarrow_w(x) = \downarrow_a(\downarrow_v(x))$. Let 0_0 be the first-order definable origin of layer zero. We define

$$D^0(x,y) = \bigvee_{i=0}^{n-1} (\downarrow_{0^i}(0_0) = x) \wedge \bigvee_{i=0}^{n-1} (\downarrow_{0^i}(0_0) = y).$$

Then, for $i > 0$, we define

$$\begin{array}{lcl} D^i(x,y) & = & \bigvee_{j=0}^{n-1} \exists z(T^j(z) \wedge D^0(z) \wedge +_ji(z,x)) \wedge \\ & & \bigvee_{j=0}^{n-1} \exists z(T^j(z) \wedge D^0(z) \wedge +_ji(z,y)). \end{array}$$

The vertical successor predicate $\oplus_i 1$ can be defined in terms of D^i. Since all the above definitions do not use second-order quantification, we can conclude that local predicates and global predicates T and $+1$ are definable in MFO$[<, (\downarrow_i)_{i=0}^{k-1}]$ over n-LSs.

We now turn to the global predicates equi-column D and vertical successor $\oplus 1$. We show that D is not definable in MSO$[<, \downarrow_0, \downarrow_1]$ over binary 2-layered structures, and, even worse, the addition of D to MSO$[<, \downarrow_0, \downarrow_1]$ yields undecidability. Since D and $\oplus 1$ are inter-definable, we have that $\oplus 1$ is not definable in MSO$[<, \downarrow_0, \downarrow_1]$ over binary 2-layered structures as well, and that its addition leads to undecidability. Moreover, it is easy to show that MSO$[<, \downarrow_0, \downarrow_1]$ over binary 2-layered structures is embeddable in MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$ over k-ary n-layered structures, and thus all the above results generalize to MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$ over k-ary n-layered structures. We begin with the following auxiliary lemma. Let us define the predicate D over $\mathbb{N}$ as the reflexive and symmetric closure of the following set:

$$\{(3k, \frac{3k+2}{2} \mid k \text{ even}\} \cup \{(3k, \frac{3k+1}{2}) \mid k \text{ odd}\}.$$

We show that MSO$[<]$ over natural numbers cannot be extended with such a predicate preserving decidability.

Lemma 5.2.1 MSO$[<, D]$ *over* $\langle \mathbb{N}, < \rangle$ *is undecidable.*

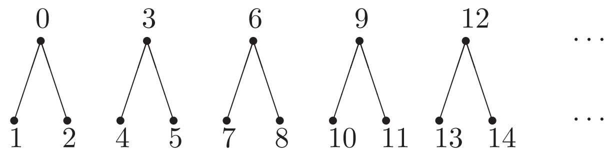

Figure 5.3: The binary 2-layered structure over natural numbers.

Proof.

Let $P_0 = \{3n \mid n \geq 0\}$, $P_1 = \{3n+1 \mid n \geq 0\}$ and $P_2 = \{3n+2 \mid n \geq 0\}$ be three unary predicates over natural numbers representing the congruence classes modulo 3. These predicates can be easily defined in MSO[$<$]. For instance $P_0(x)$ is defined as follows:

$$\exists X(x \in X \land 0 \in X \land \forall y, v, z, w(y \in X \rightarrow ((+1(y,v) \rightarrow v \notin X) \land \\ (+2(y,z) \rightarrow z \notin X) \land (+3(y,w) \rightarrow w \in X))),$$

where 0 is the first-order definable constant representing the natural number 0 and $+1$, $+2$ and $+3$ are the first-order definable predicates defining the first, second and third successor of a point, respectively. By exploiting the relation D, we are able to define the relation $2\times$ such that $2 \times (x,y)$ iff $y = 2x$ as follows:

$$\begin{aligned} 2 \times (x,y) \quad = \quad & (x = 0 \rightarrow y = 0) \land \exists z, w \\ & ((P_0(x) \land +1(x,w) \land D(w,z) \land w \neq z) \rightarrow y = z \land \\ & (P_1(x) \land +2(z,w) \land D(x,z) \land x \neq z) \rightarrow y = w \land \\ & (P_2(x) \land +1(z,w) \land D(x,z) \land x \neq z) \rightarrow y = w). \end{aligned}$$

It is well known that MSO[$<, 2\times$] over natural numbers is undecidable, since it allows one to interpret full first-order arithmetic. We conclude that MSO[$<, D$] over natural numbers is undecidable. ■

To prove our thesis, it suffices to show that MSO[$<, D$] over $\langle \mathbb{N}, < \rangle$ can be embedded into MSO[$<, \downarrow_0, \downarrow_1, D$] over binary 2-LSs.

Theorem 5.2.2 (*Undefinability of equi-column and vertical successor*)

The predicates D and $\oplus 1$ are not definable in MSO[$<, (\downarrow_i)_{i=0}^{k-1}$] *over k-ary n-LSs. Moreover, the extension of* MSO[$<, (\downarrow_i)_{i=0}^{k-1}$] *with D or $\oplus 1$ is undecidable.*

Proof.

We first show that MSO[$<, D$] over $\langle \mathbb{N}, < \rangle$ can be embedded into MSO[$<, \downarrow_0, \downarrow_1, D$] over 2-layered binary structures (note that, for notational simplicity, we are overloading symbols $<$ and D). Let us consider the bijection $\tau : \mathcal{U}_2 \rightarrow \mathbb{N}$ depicted in Figure 5.3. It is formally defined as follows: $\tau(n_0) = 3n$, $\tau(n_1) = (3n+2)/2$ if n is even, and $\tau(n_1) = (3n+1)/2$ if n is odd. It is easy to see that τ is an isomorphism between $\langle \mathcal{U}_2, <, D \rangle$ and $\langle \mathbb{N}, <, D \rangle$. It follows that, for every $\varphi \in$ MSO[$<, D$], φ is satisfiable over $\langle \mathbb{N}, <, D \rangle$ if and only if φ is satisfiable over $\langle \mathcal{U}_2, <, D \rangle$. From Lemma 5.2.1 it immediately follows that MSO[$<, \downarrow_0, \downarrow_1, D$] over binary 2-LSs is undecidable. Moreover, it is easy to show that MSO[$<, \downarrow_0, \downarrow_1, D$] over binary 2-LSs is embeddable in MSO[$<, (\downarrow_i)_{i=0}^{k-1}, D$] over k-ary n-LSs. Hence, MSO[$<, (\downarrow_i)_{i=0}^{k-1}, D$] and, thus, MSO[$<, (\downarrow_i)_{i=0}^{k-1}, \oplus 1$] are undecidable. Since MSO[$<, (\downarrow_i)_{i=0}^{k-1}$] over k-ary n-LSs is decidable (Theorem 2.2.7), we have that D and $\oplus 1$ are not definable in MSO[$<, (\downarrow_i)_{i=0}^{k-1}$]. ■

Theory	T	$+1$	D	$\oplus 1$
MFO	Decidable	Decidable	?	Decidable
MPL	Decidable	Decidable	?	Decidable
MCL	Decidable	Decidable	?	Decidable
$\text{MFO}_{\mathcal{P}}$	Decidable	Decidable	?	?
$\text{MPL}_{\mathcal{P}}$	Decidable	Decidable	?	?
$\text{MCL}_{\mathcal{P}}$	Decidable	Decidable	?	?
MSO	Decidable	Decidable	Undecidable	Undecidable

Table 5.1: Decidability over n-layered structures

Finally, it is possible to show that the extensions of the chain, path, and first-order fragments of $\text{MSO}[<, (\downarrow_i)_{i=0}^{k-1}]$ with the undefinable predicate $\oplus 1$ are decidable. It can indeed be proved that $\text{MCL}[<, (\downarrow)_{i=0}^{k-1}, \oplus 1]$ is decidable, and thus so are also $\text{MPL}[<, (\downarrow)_{i=0}^{k-1}, \oplus 1]$ and $\text{MFO}[<, (\downarrow)_{i=0}^{k-1}, \oplus 1]$. This result follows from Theorem 5.4.4, since n-LSs can be easily embedded into DULSs. The decidability results for n-layered structures are summarized in Table 5.1 (a question mark stands for an open problem).

5.3 Definability and decidability over UULSs

In this section, we explore definability and decidability issues for monadic theories of UULSs. The local predicate i-th equi-level T^i can be defined as follows:

$$\begin{array}{lcl} T^0(x,y) & = & \neg\exists z_1(\downarrow_0(x) = z_1) \wedge \neg\exists z_2(\downarrow_0(y) = z_2); \\ T^{i+1}(x,y) & = & \exists z_1 \exists z_2 (T^i(z_1,z_2) \wedge \downarrow_0(x) = z_1 \wedge \downarrow_0(y) = z_2). \end{array}$$

The horizontal successor $+_i$ can be defined in terms of T^i. As for D^i, we first define D^0 (0_0 is the first-order definable origin of layer zero):

$$\begin{array}{lcl} \mathtt{D}^0(x,y) & = & \exists X(x \in X \wedge y \in X \wedge 0_0 \in X \wedge \\ & & \forall z((T^0(z) \wedge z \neq 0_0) \rightarrow z \notin X) \wedge \\ & & \forall z(z \in X \rightarrow \exists w(\downarrow_0(w) = z \wedge w \in X))) \wedge \\ & & \forall z((z \in X \wedge z \neq 0_0) \rightarrow \exists w(\downarrow_0(z) = w \wedge w \in X)))). \end{array}$$

Let $a_n k^n + \ldots a_0 k^0$ be the k-ary representation of i, for any $i > 0$. D^i can be defined as follows:

$$D^i(x,y) \;=\; \exists z(D^0(z) \wedge \downarrow_{a_0 \ldots a_n}(z) = x) \wedge \exists z(D^0(z) \wedge \downarrow_{a_0 \ldots a_n}(z) = y).$$

The vertical successor $\oplus_i$ can be defined in terms of D^i. Notice that second-order quantification comes into play only in the definition of D^0. Moreover, the semantics of D^0 does not change if we interpret the second-order variable X as a path. Hence, i-th equi-column D^i and i-th vertical successor $\oplus_i 1$ can be encoded in $\text{MPL}[<, (\downarrow_i)_{i=0}^{k-1}]$, and i-th equi-level T^i and i-th horizontal successor $+_i 1$ can be encoded in $\text{MFO}[<, (\downarrow_i)_{i=0}^{k-1}]$.

Consider now the global predicates. We start by showing that the addition of the vertical predicate $\oplus 1$ or D to the first-order theory $\text{MFO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ makes it undecidable. The proof reduces a suitable undecidable version of the tiling problem to the satisfiability problem for $\text{MFO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$-formulas and essentially exploits monadic predicates in $\mathcal{P}$.

Theorem 5.3.1 $\text{MFO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}, \oplus 1]$ *and* $\text{MFO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}, D]$ *over k-ary UULSs are undecidable.*

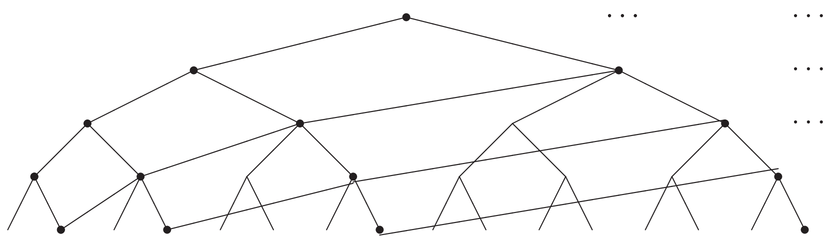

Figure 5.4: Shaping an octant over an UULS.

Proof.

We give the proof for binary UULSs. The general result follows. We show that $\mathrm{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, \oplus 1]$ over binary UULSs is undecidable by embedding into it the *octant tiling problem* [65]. Since $\mathrm{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, \oplus 1] \to \mathrm{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, D]$, we have that $\mathrm{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, D]$ is undecidable too.

Recall that the octant tiling problem asks: given a finite set of tile types $\mathcal{T}$, can $\mathcal{T}$ tile the octant (half of a quadrant) $\mathcal{O} = \bigcup_{i \geq 0}\{(i,j) \mid 0 \leq j \leq i\}$? For every tile type $t \in \mathcal{T}$, let $\mathtt{right}(t)$, $\mathtt{left}(t)$, $\mathtt{up}(t)$ and $\mathtt{down}(t)$ be the colors or the corresponding sides of t. The octant tiling problem is to find a function $f : \mathcal{O} \to \mathcal{T}$ such that $\mathtt{right}(f(n,m)) = \mathtt{left}(f(n{+}1,m))$ and, whenever $m < n$, $\mathtt{up}(f(n,m)) = \mathtt{down}(f(n,m+1))$. We will reduce the octant tiling problem to the satisfiability problem for $\mathrm{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, \oplus 1]$ over binary UULSs. Suppose that $\mathcal{T} = \{T_1, \ldots, T_k\}$ is the given set of tile types. We will construct a $\mathrm{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, \oplus 1]$-formula $\varphi_{\mathcal{T}}$ such that $\mathcal{T}$ tiles $\mathcal{O}$ if and only if $\varphi_{\mathcal{T}}$ is satisfiable over binary UULSs.

The first step is forcing the octant grid over a binary UULS $\langle \mathcal{U}, \downarrow_0, \downarrow_1, < \rangle$. The octant grid domain is the set $\mathcal{G} = \bigcup_{i \geq 0}\{(2^{(i-j)} - 1)_j \mid 0 \leq j \leq i\} \subset \mathcal{U}$. Notice that $x \in \mathcal{G}$ if and only if x is reachable along a rightmost path rooted at some point in $\{0_i \mid i \geq 0\}$. The horizontal grid successor is $\mathtt{s_0}$ such that, for every $n_r \in \mathcal{G}$, $\mathtt{s_0}(n_r) = n_{r+1}$ and the vertical grid successor is $\mathtt{s_1}$ such that, for every $n_r \in \mathcal{G}$, $r > 0$, $\mathtt{s_1}(n_r) = (2n+1)_{r-1}$ (cf. Figure 5.4). Note that, for every $n_r \in \mathcal{G}$, $r > 0$, $\mathtt{s_0}(\mathtt{s_1}(n_r)) = \mathtt{s_1}(\mathtt{s_0}(n_r))$. We can define in $\mathrm{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, \oplus 1]$ a monadic predicate $\mathtt{grid}$ such that $\mathtt{grid}(x)$ if and only if x belongs to the octant grid domain $\mathcal{G}$. Let $P_{lp}, Q_{grid} \in \mathcal{P}$. We have $\mathtt{grid}(x)$ if and only if

$$
\begin{aligned}
&x \in Q_{grid} \wedge 0_0 \in P_{lp} \wedge \\
&\forall y((y \in P_{lp} \to \exists z(\downarrow_0(z) = y \wedge z \in P_{lp} \wedge \downarrow_1(z) \notin P_{lp})) \wedge \\
&(y \notin P_{lp} \wedge \neg T^0(y) \to \downarrow_0(y) \notin P_{lp} \wedge \downarrow_1(y) \notin P_{lp})) \wedge \\
&\forall y((y \in P_{lp} \to y \in Q_{grid}) \wedge \\
&(y \in Q_{grid} \wedge y \in P_{lp} \wedge \neg T^0(y) \to \downarrow_0(y) \in Q_{grid} \wedge \downarrow_1(y) \in Q_{grid}) \wedge \\
&(y \in Q_{grid} \wedge y \notin P_{lp} \wedge \neg T^0(y) \to \downarrow_0(y) \notin Q_{grid} \wedge \downarrow_1(y) \in Q_{grid}) \wedge \\
&(y \notin Q_{grid} \wedge \neg T^0(y) \to \downarrow_0(y) \notin Q_{grid} \wedge \downarrow_1(y) \notin Q_{grid}))
\end{aligned}
$$

Moreover, the horizontal successor $\mathtt{s_0}$ is $\oplus 1$, and the vertical successor $\mathtt{s_1}$ is $\downarrow_1$. We now have to write the tiling constraints on the grid. To this end, we make use of monadic predicates in $\{P_1, \ldots, P_k\} \subset \mathcal{P}$ corresponding to the tile types in $\{T_1, \ldots, T_k\}$.

1. Exactly one tile is placed at each node:

$$\phi_1(x) = \bigvee_{i=1}^{i=k} x \in P_i \bigwedge_{1 \leq i < j \leq k} \neg(x \in P_i \wedge x \in P_j)$$

2. Colors match going right:

$$\phi_2(x) = \bigvee_{right(T_i)=left(T_j)} x \in P_i \wedge \oplus 1(x) \in P_j$$

3. Colors match going up:

$$\phi_3(x) = \neg T^0(x) \rightarrow \bigvee_{up(T_i)=down(T_j)} x \in P_i \wedge \downarrow_1(x) \in P_j$$

We define

$$\varphi_{\mathcal{T}} = \forall x(\mathtt{grid}(x) \rightarrow \phi_1(x) \wedge \phi_2(x) \wedge \phi_3(x)).$$

It is easy to see that $\mathcal{T}$ tiles $\mathcal{O}$ if and only if $\varphi_{\mathcal{T}}$ is satisfiable over binary UULSs. ■

We do not know whether the same effect is produced by the addition of $+1$ (resp. T) to MFO$_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$. The decidability of MFO$_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}, +1]$ (resp. MFO$_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}, T]$) over UULSs is indeed an open problem. However, we have a proof of the undecidability of MSO$[<, (\downarrow_i)_{i=0}^{k-1}, +1]$ and MSO$[<, (\downarrow_i)_{i=0}^{k-1}, T]$. The proof rests on the undecidability of the monadic second-order theory of binary UULSs extended with a binary predicate `adj` such that $\mathtt{adj}(x, y)$ holds if and only if y is the *adoptive* son of x, that is, y is the horizontal successor of the right son of x.

Theorem 5.3.2 MSO$[<, (\downarrow_i)_{i=0}^{k-1}, +1]$ *and* MSO$[<, (\downarrow_i)_{i=0}^{k-1}, T]$ *over k-ary UULSs are undecidable.*

Proof.

We prove the thesis for binary UULSs. The general case follows. To show that MSO$[<, \downarrow_0, \downarrow_1, +1]$ is undecidable, we embed the theory MSO$[<, \mathtt{adj}]$ over $\langle \mathbb{N}_+, < \rangle$, which is known to be undecidable (cf. Theorem 2.2.4), into MSO$[<, \downarrow_0, \downarrow_1, +1]$ over binary UULSs. Since T and $+1$ are inter-definable, the undecidability holds for MSO$[<, \downarrow_0, \downarrow_1, T]$ also. Recall that the predicate `adj` over positive natural numbers is defined as follows: $\mathtt{adj}(x, y)$ iff $x = 2^{k_n} + 2^{k_{n-1}} + \ldots + 2^{k_0}$, with $k_n > k_{n-1} > \ldots > k_0 > 0$, and $y = x + 2^{k_0} + 2^{k_0 - 1}$. We define the binary predicate `adj` over UULSs as follows (we are overloading the symbol `adj`): for every $n_r, m_s \in \mathcal{U}$, we have $\mathtt{adj}(n_r, m_s)$ iff $r > 0$, $s = r - 1$, and $m = 2(n + 1)$. Note that $\mathtt{adj}(x, y)$ iff y is the *adoptive* son of x, that is, y is the horizontal successor of the right son of x. Hence, `adj` is definable in MSO$[<, \downarrow_0, \downarrow_1, +1]$ over 2-refinable UULSs as follows:

$$\mathtt{adj}(x, y) \quad = \quad \exists z(\downarrow_1(x) = z \wedge +1(z, y)).$$

Consider now the bijection $\tau : \mathcal{U} \rightarrow \mathbb{N}_+$ depicted in Figure 5.5. Formally, τ is defined as follows: for every $n_r \in \mathcal{U}$, $\tau(n_r) = 2^r + n2^{r+1}$. It is easy to see that τ is an isomorphism between $\langle \mathcal{U}, <, \mathtt{adj} \rangle$ and $\langle \mathbb{N}_+, <, \mathtt{adj} \rangle$. Since MSO$[<, \mathtt{adj}]$ over $\langle \mathbb{N}_+, < \rangle$ is undecidable, we have that MSO$[<, \mathtt{adj}]$ over UULSs is undecidable. Finally, since MSO$[<, \mathtt{adj}] \rightarrow$ MSO$[<, \downarrow_0, \downarrow_1, +1]$, we have the thesis. ■

Taking advantage of the bijection used in the proof of Theorem 5.3.2, it is possible to embed the undecidable theory MSO$[<, 2\times]$ over $\langle \mathbb{N}_+, < \rangle$ into MSO$[<, \downarrow_0, \downarrow_1, \oplus 1]$ over binary UULSs (the double predicate $2\times$ is encoded as the vertical successor $\oplus 1$). Hence, MSO$[<, \downarrow_0, \downarrow_1]$ extended with either $\oplus 1$ or D is undecidable (this result is however implied by the stronger Theorem 5.3.1). We thus have the following.

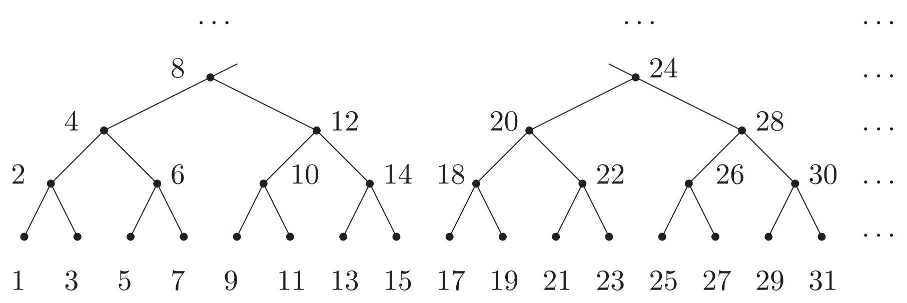

Figure 5.5: The upward unbounded layered structure over natural numbers.

Theorem 5.3.3 (*Undefinability of global predicates*)

Global predicates are not definable in MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$ *over k-ary UULSs. Moreover, the extension of* MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$ *with any global predicate is undecidable.*

To complete the picture, let us focus on the decidability problem for the extensions of MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$ fragments with global predicates. Surprisingly, the addition of T, $+1$, and $\oplus 1$ to the chain fragment MCL$[<, (\downarrow_i)_{i=0}^{k-1}]$ preserves decidability (these global predicates are obviously *not* definable in such a fragment). The proof exploits a reduction of MCL$[<, (\downarrow_i)_{i=0}^{k-1}, T, +1, \oplus 1]$ to the decidable theory MSO$[<]$ over $\langle \mathbb{N}, < \rangle$.

Theorem 5.3.4 MCL$[<, (\downarrow_i)_{i=0}^{k-1}, T, +1, \oplus 1]$ *over k-ary UULSs is decidable.*

Proof.

The proof is given for $k = 2$ and it can be easily extended to the general case of $k > 2$.

First note that MCL$[<, \downarrow_0, \downarrow_1, T, +1, \oplus 1] \rightarrow$ MCL$[\downarrow_0, \downarrow_1, T, \oplus 1]$. Indeed, the horizontal successor $+1$ is first-order definable in terms of T and $<$. Moreover, $<$ is proved to be redundant as shown in Proposition 4.3.1. Furthermore, it is easy to show that MCL$[\downarrow_0, \downarrow_1, T, \oplus 1]$ is equivalent to a version of chain logic in which only second-order variables occur and atomic formulas are of the form $X_1 \subseteq X_2$ (chain X_1 is included in chain X_2), $\mathtt{Sing}(X)$ (chain X is a singleton), $\mathtt{proj}_i(X_1, X_2)$ (chain $X_1 = \{x_1\}$, chain $X_2 = \{x_2\}$ and $\downarrow_i(x) = y$), $\mathtt{equiL}(X_1, X_2)$ (chain $X_1 = \{x_1\}$, chain $X_2 = \{x_2\}$, and $T(x, y)$), and $\mathtt{vsucc}(X_1, X_2)$ (chain $X_1 = \{x_1\}$, chain $X_2 = \{x_2\}$, and $\oplus 1(x, y)$).

By induction on the formulas of the latter version of chain logic, it is possible to prove that every formula of chain logic can be encoded into an equi-satisfiable formula in MSO$[<]$ over natural numbers, a theory known to be decidable. The idea is the following, and is borrowed from [115]. Any second-order variable X interpreted as a chain is encoded by a pair of set variables Ch_X and Lv_X over the natural numbers. Ch_X is interpreted as a set of natural numbers encoding the leftmost upward unbounded path (starting form the first layer) containing the chain X, i.e., $i \in Ch_X$ iff the element of the i-th layer of the mentioned path is a right-hand side son. Lv_X is interpreted as a set of natural numbers describing the elements of the path actually belonging to the chain, i.e., $i \in Lv_X$ iff the element of the i-th layer of the path belongs to the chain X. To guarantee that a chain X corresponds to a unique pair Ch_X and Lv_X, we have to use the following condition

$$\mathtt{unique}(Ch_X, Lv_X) = \forall y (y \in Ch_X \rightarrow \mathtt{min}(Lv_X) \leq y),$$

where $\mathtt{min}(X)$ is the minimum of the set X of natural numbers with respect to the usual ordering relation $<$.

As for atomic formulas,

`Theory`	T	$+1$	D	$\oplus 1$
MFO	Decidable	Decidable	?	Decidable
MPL	Decidable	Decidable	?	Decidable
MCL	Decidable	Decidable	?	Decidable
$\text{MFO}_{\mathcal{P}}$	?	?	Undecidable	Undecidable
$\text{MPL}_{\mathcal{P}}$	?	?	Undecidable	Undecidable
$\text{MCL}_{\mathcal{P}}$	?	?	Undecidable	Undecidable
MSO	Undecidable	Undecidable	Undecidable	Undecidable

Table 5.2: Decidability for upward unbounded layered structures

- $X_1 \subseteq X_2$ is encoded as $Ch_{X_1} \subseteq Ch_{X_2} \wedge Lv_{X_1} \subseteq Lv_{X_2}$;
- $\texttt{Sing}(X)$ in encoded as 'Lv_X is a singleton', that is,

$$\exists Y(Y \subseteq Lv_X \wedge Lv_X \neq Y \wedge \neg\exists Z(Z \subseteq Lv_X \wedge Z \neq Lv_X \wedge Z \neq Y));$$

- $\texttt{proj}_0(X_1, X_2)$ is encoded as $Ch_{X_1} = Ch_{X_2}$, 'Lv_{X_1} is a singleton $\{x_1\}$', 'Lv_{X_2} is a singleton $\{x_2\}$', and $x_1 = x_2 + 1$;
- $\texttt{proj}_1(X_1, X_2)$ is encoded as 'Lv_{X_1} is a singleton $\{x_1\}$', 'Lv_{X_2} is a singleton $\{x_2\}$', $x_1 = x_2 + 1$, and $Ch_{X_1} = Ch_{X_2} \cup \{x_2\}$;
- $\texttt{equiL}(X_1, X_2)$ is encoded as 'Lv_{X_1} is a singleton $\{x_1\}$', 'Lv_{X_2} is a singleton $\{x_2\}$', and $x_1 = x_2$;
- $\texttt{vsucc}(X_1, X_2)$ is encoded as 'Lv_{X_1} is a singleton $\{x_1\}$', 'Lv_{X_2} is a singleton $\{x_2\}$', $x_1 = x_2 + 1$ and $Ch_{X_2} = \{n + 1 \mid n \in Ch_{X_1}\}$.

The induction cases for $\wedge$ and $\neg$ are trivial. Finally, the second-order existentially quantified chain formula $\exists X \phi(X)$ is translated into the formula

$$\exists Ch_X \exists Lv_X(\texttt{unique}(Ch_X, Lv_X) \wedge \phi^\tau(Ch_X, Lv_X)),$$

where ϕ^τ is the translation of ϕ. ■

The decidability results for UULSs are summarized in Table 5.2. Interestingly, and with some surprise, the horizontal successor predicate $+1$ can be defined in terms of the equi-column predicate D.

Theorem 5.3.5 $\text{MFO}[<, \downarrow_0, \downarrow_1, +1]$ *can be embedded into* $\text{MFO}[<, \downarrow_0, \downarrow_1, D]$ *over binary UULSs.*

Proof.

We have that $+1(x, y)$ iff

$$(\exists z(\downarrow_0(z) = x) \rightarrow \exists z(\downarrow_0(z) = x \wedge \downarrow_1(z) = y)) \wedge$$
$$(\exists z(\downarrow_1(z) = x) \rightarrow \exists z(\downarrow_1(z) = x \wedge \texttt{adj}(z, y))),$$

where $\texttt{adj}(z, y)$ says that y is the *adoptive* son of z whenever z does not belong to the first layer of the structure. Moreover, y is the adoptive son of z means that y is the horizontal successor of the right son of z. We need to encode the predicate `adj` into MFO$[<, \downarrow_0, \downarrow_1, D]$. We have that $\texttt{adj}(x, y)$ iff $\phi(x, y)$, where $\phi(x, y)$ iff:

$$\begin{aligned}
&\neg T^0(x) \rightarrow \exists z_1 \exists z_2 \exists z_3 \exists z_4 \exists z_5\\
&(T^0(z_1) \wedge D(z_1, x) \wedge\\
&+_0 1(z_1, z_2) \wedge\\
&\oplus 1(z_2, z_3) \wedge\\
&\downarrow_0(z_3) = z_4 \wedge\\
&D(z_4, z_5) \wedge \forall w(D(z_4, w) \wedge x \leq w \rightarrow z_5 \leq w) \wedge\\
&(D^0(x) \wedge x \neq 0_1 \rightarrow \oplus 1(z_5, y)) \wedge\\
&((\neg D^0(x) \vee x = 0_1) \rightarrow z_5 = y))
\end{aligned}$$

We prove that the above definition captures the predicate `adj`. Let $x = n_r$, $y = m_s$ such that x does not belong to the first layer of the structure, i.e., $r \geq 1$. Note that $\texttt{adj}(x, y)$ if and only if $r \geq 1$, $s = r - 1$ and $m = 2(n + 1)$. Suppose that $\phi(x, y)$ holds. Then, there exist z_i, for $i = 1, \ldots, 5$, such that $z_1 = n_0$, $z_2 = (n + 1)_0$, $z_3 = (n + 1)_1$, $z_4 = (2(n + 1))_0$, and $z_5 = \min\{w \mid w = (2(n + 1))_i \text{ and } i \geq 0 \text{ and } n_r \leq w\}$. We claim that $z_5 = y = (2(n + 1))_{r-1}$, and thus $\texttt{adj}(x, y)$ holds, whenever $(\neg D^0(x) \vee x = 0_1)$, and $z_5 = (2(n+1))_{r-2}$, $y = (2(n+1))_{r-1}$, and thus $\texttt{adj}(x, y)$ holds, whenever $(D^0(x) \wedge x \neq 0_1)$. We prove the claim. Suppose $\neg D^0(x)$, that is, $n \geq 1$. Since, for every $i, j \geq 0$, $i_j < i_{j+1}$, we only have to prove that $(2(n + 1))_{r-2} < n_r < (2(n + 1))_{r-1}$. To see that $(2(n + 1))_{r-2} < n_r$, note that $(2(n+1))_{r-2} \leq (4n)_{r-2}$. This holds since $2(n+1) \leq 4n$ whenever $n \geq 1$. Moreover, $(4n)_{r-2} < n_r$, since $(4n)_{r-2} = \downarrow_0(\downarrow_0(n_r))$, and, for every point v, $\downarrow_0(v) < v$. We now show that $n_r < (2(n+1))_{r-1}$. It holds since $(2(n+1))_{r-1} = +1(\downarrow_1(n_r))$, and, for every v, $v < \downarrow_1(v)$ and $v < +1(v)$. The cases $x = 0_1$ and $(D^0(x) \wedge x \neq 0_1)$ are easier. Similarly, if $\texttt{adj}(x, y)$ holds, then $\phi(x, y)$ holds. Hence the thesis. ■

Since $+1$ and T are inter-definable in the monadic second-order theory of UULSs, and the same holds for $\oplus$ and D, we have the following corollary.

Corollary 5.3.6 *Let $o \in \{+1, T\}$ and $v \in \{\oplus 1, D\}$.* MSO$[<, \downarrow_0, \downarrow_1, o]$ *can be embedded into* MSO$[<, \downarrow_0, \downarrow_1, v]$ *over binary UULSs.*

5.4 Definability and decidability over DULSs

We conclude this chapter by exploring definability and decidability issues for monadic theories of DULSs. The local predicates T^i and $+_i 1$ can be expressed as in the case of n-LSs (Section 5.2), while the definition of the local predicate D^i given in the case of n-LSs does not work anymore since we have to cope with an infinite number of layers. We first define $\mathtt{D}^0(x, y)$ as follows:

$$\begin{aligned}
&\exists X(x \in X \wedge y \in X \wedge 0_0 \in X \wedge \forall z(T^0(z) \wedge z \neq 0_0 \rightarrow z \notin X) \wedge\\
&\forall z(z \in X \rightarrow (\downarrow_0(z) \in X \wedge \textstyle\bigwedge_{i=1}^{k-1} \downarrow_i(z) \notin X) \wedge z \notin X \rightarrow \textstyle\bigwedge_{i=0}^{k-1} \downarrow_i(z) \notin X)),
\end{aligned}$$

where 0_0 is the first-order definable origin of layer zero. For $i > 0$, let $a_n k^n + \ldots + a_0 k^0$ be the k-ary representation of i. We define $D^i(x, y)$ as follows:

$$\begin{aligned}
&(\textstyle\bigvee_{j=0}^{\lfloor \log_k(i) \rfloor} \exists z(D^0(z) \wedge T^j(z) \wedge +_j i(z) = x) \vee \exists z(D^0(z) \wedge \downarrow_{a_0 \ldots a_n}(z) = x)) \wedge\\
&(\textstyle\bigvee_{j=0}^{\lfloor \log_k(i) \rfloor} \exists z(D^0(z) \wedge T^j(z) \wedge +_j i(z) = y) \vee \exists z(D^0(z) \wedge \downarrow_{a_0 \ldots a_n}(z) = y)).
\end{aligned}$$

As already shown, $\oplus_i 1$ can be defined in terms of D^i. As in the case of UULSs, second-order quantification is only used to define D^0. Moreover, the semantics of D^0 does not change

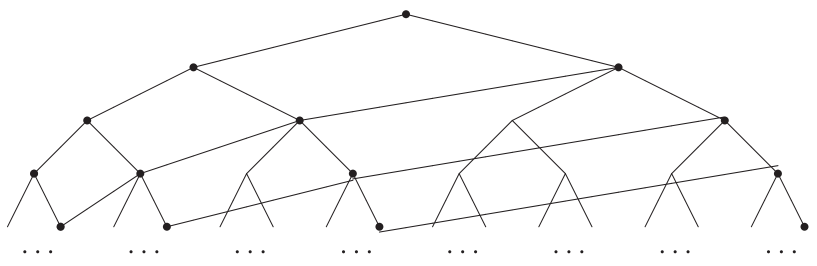

Figure 5.6: Shaping an $\mathbb{N} \times \mathbb{N}$-grid over a binary tree.

if we interpret the second-order variable X as a path. Hence i-th equi-column D^i and i-th vertical successor $\oplus_i 1$ can be encoded in $\text{MPL}[<, (\downarrow_i)_{i=0}^{k-1}]$, and i-th equi-level T^i and i-th horizontal successor $+_i 1$ can be encoded in $\text{MFO}[<, (\downarrow_i)_{i=0}^{k-1}]$.

Consider now global predicates. As in the case of UULSs, the addition of the vertical predicates $\oplus 1$ or D to $\text{MFO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ leads to undecidability. The proof takes advantage of a reduction of the $\mathbb{N}\times\mathbb{N}$ tiling problem to the satisfiability problem for $\text{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, \oplus 1]$ and essentially exploits the monadic predicates in $\mathcal{P}$.

Theorem 5.4.1 $\text{MFO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}, \oplus 1]$ *and* $\text{MFO}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}, D]$ *over k-ary DULSs are undecidable.*

Proof.

We show that both the theories are undecidable over binary infinite trees. Since binary infinite trees are embeddable into k-ary DULSs, we have the thesis. We show that $\text{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, \oplus 1]$ over binary infinite trees is undecidable by a reduction of the $\mathbb{N} \times \mathbb{N}$ tiling problem [11]. Since $\text{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, \oplus 1] \rightarrow \text{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, D]$, we have that $\text{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, D]$ is undecidable too. Recall that the $\mathbb{N} \times \mathbb{N}$ tiling problem asks: given a finite set of tile types $\mathcal{T}$, can $\mathcal{T}$ tile $\mathbb{N} \times \mathbb{N}$? For every tile type $t \in \mathcal{T}$, let $\texttt{right}(t)$, $\texttt{left}(t)$, $\texttt{up}(t)$ and $\texttt{down}(t)$ be the colors or the corresponding sides of t. The $\mathbb{N} \times \mathbb{N}$ tiling problem is to find a function $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathcal{T}$ such that $\texttt{right}(f(n,m)) = \texttt{left}(f(n+1,m))$ and $\texttt{up}(f(n,m)) = \texttt{down}(f(n,m+1))$. We will reduce the $\mathbb{N}\times\mathbb{N}$ tiling problem to the satisfiability problem for $\text{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, \oplus 1]$ over binary infinite trees. Suppose that $\mathcal{T} = \{T_1, \ldots, T_k\}$ is the given set of tile types. We will construct a monadic $\text{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, \oplus 1]$-formula $\varphi_{\mathcal{T}}$ such that $\mathcal{T}$ tiles $\mathbb{N} \times \mathbb{N}$ if and only if $\varphi_{\mathcal{T}}$ is satisfiable over binary infinite trees.

The first step is forcing the grid over a binary infinite tree $\langle \{0,1\}^*, <_{pre} \rangle$. The grid is given by the domain $\{0^*1^*\}$, the horizontal successor in $\texttt{s}_0(x) = x1$ and the vertical successor in $\texttt{s}_1(x) = 0x$ (cf. Figure 5.6). Note that, for every $x \in \{0^*1^*\}$, $\texttt{s}_0(\texttt{s}_1(x)) = \texttt{s}_1(\texttt{s}_0(x))$. We define in $\text{MFO}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, \oplus 1]$ a monadic predicate $\texttt{grid}$ such that $\texttt{grid}(x)$ if and only if x belongs to the grid domain $\{0^*1^*\}$. Note that $\{0^*1^*\} = \bigcup_{i \geq 0} \{0^i 1^*\}$. Hence, $\texttt{grid}(x)$ if and only if x is reachable along a rightmost path rooted at some point in $\{0^*\}$. Let $P_{lp}, Q_{grid} \in \mathcal{P}$. We have $\texttt{grid}(x)$ if and only if

$$\begin{aligned}
&x \in Q_{grid} \wedge \epsilon \in P_{lp} \wedge \\
&\forall y((y \in P_{lp} \rightarrow {\downarrow_0}(y) \in P_{lp} \wedge {\downarrow_1}(y) \notin P_{lp}) \wedge \\
&(y \notin P_{lp} \rightarrow {\downarrow_0}(y) \notin P_{lp} \wedge {\downarrow_1}(y) \notin P_{lp})) \wedge \\
&\forall y((y \in P_{lp} \rightarrow y \in Q_{grid}) \wedge \\
&(y \in Q_{grid} \wedge y \in P_{lp} \rightarrow {\downarrow_0}(y) \in Q_{grid} \wedge {\downarrow_1}(y) \in Q_{grid}) \wedge \\
&(y \in Q_{grid} \wedge y \notin P_{lp} \rightarrow {\downarrow_0}(y) \notin Q_{grid} \wedge {\downarrow_1}(y) \in Q_{grid}) \wedge \\
&(y \notin Q_{grid} \rightarrow {\downarrow_0}(y) \notin Q_{grid} \wedge {\downarrow_1}(y) \notin Q_{grid}))
\end{aligned}$$

Once we have shaped the grid, we can encode the horizontal and the vertical successors as $\downarrow_1$ and $\oplus 1$, respectively, and we can write the tiling constraints on the grid. To this end, we make use of monadic predicates in $\{P_1, \ldots, P_k\} \subset \mathcal{P}$ corresponding to the tile types in $\{T_1, \ldots, T_k\}$.

1. Exactly one tile is placed at each node:

$$\phi_1(x) = \bigvee_{i=1}^{i=k} x \in P_i \bigwedge_{1 \leq i < j \leq k} \neg(x \in P_i \wedge x \in P_j).$$

2. Colors match going right:

$$\phi_2(x) = \bigvee_{right(T_i)=left(T_j)} x \in P_i \wedge \downarrow_1 (x) \in P_j$$

3. Colors match going up:

$$\phi_3(x) = \bigvee_{up(T_i)=down(T_j)} x \in P_i \wedge \oplus 1(x) \in P_j.$$

We define

$$\varphi_{\mathcal{T}} = \forall x(\texttt{grid}(x) \rightarrow \phi_1(x) \wedge \phi_2(x) \wedge \phi_3(x)).$$

It is easy to see that $\mathcal{T}$ tiles $\mathbb{N} \times \mathbb{N}$ if and only if $\varphi_{\mathcal{T}}$ is satisfiable over binary infinite trees. ■

As a matter of fact, the proof of Theorem 5.4.1 does not exploit the whole DULS, but only the first tree of it. It follows that the same result holds for binary infinite trees. As for the equi-level predicate T, we have that its addition to $\text{MPL}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ does not preserve decidability. The proof rests of an embedding of the $\mathbb{N} \times \mathbb{N}$ tiling problem, and essentially uses the monadic predicates in $\mathcal{P}$.

Theorem 5.4.2 $\text{MPL}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}, T]$ *over k-ary DULSs is undecidable.*

Proof.

We prove the theorem for binary DULSs. The general case follows. We show that $\text{MPL}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, T]$ over binary DULSs is undecidable by reducing to it the $\mathbb{N} \times \mathbb{N}$ tiling problem (see proof of Theorem 5.4.1). Suppose that $\mathcal{T} = \{T_1, \ldots, T_k\}$ is the given set of tile types. We will construct a $\text{MPL}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, T]$-formula $\varphi_{\mathcal{T}}$ such that $\mathcal{T}$ tiles $\mathbb{N} \times \mathbb{N}$ if and only if $\varphi_{\mathcal{T}}$ is satisfiable over binary DULSs. The first step is forcing the grid over a binary DULS $\langle \mathcal{U}, \downarrow_0, \downarrow_1, < \rangle$. We define the grid domain as the set $\mathcal{G} = \bigcup_{i \geq 0} \{(i2^j)_j \mid j \geq 0\} \subset \mathcal{U}$, the horizontal successor $\mathtt{s_0}(n_r) = (n + 2^r)_r$, and the vertical successor $\mathtt{s_1}(n_r) = (2n)_{r+1}$ (cf.

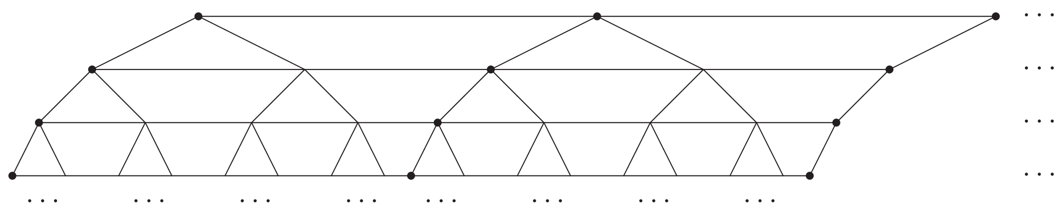

Figure 5.7: Shaping an $\mathbb{N} \times \mathbb{N}$-grid over a DULS.

Figure 5.7). Note that, for every $n_r \in \mathcal{G}$, $\mathtt{s_0}(\mathtt{s_1}(n_r)) = \mathtt{s_1}(\mathtt{s_0}(n_r))$. Moreover, it is easy to define in $\mathrm{MPL}_{\mathcal{P}}[<, \downarrow_0, \downarrow_1, T]$ a monadic predicate $\mathtt{grid}$ such that $\mathtt{grid}(x)$ if and only if x belongs to the grid domain $\mathcal{G}$. This predicate is true over x if and only if x is reachable along a leftmost path rooted at some point belonging to the first layer of the structure. We have

$$\mathtt{grid}(x) = \exists y (T^0(y) \wedge \mathtt{LP}(y, x)),$$

where $\mathtt{LP}(y, x)$ iff x and y belongs to the same leftmost path. Formally, $\mathtt{LP}(y, x))$ iff

$$\exists X (x \in X \wedge y \in X \wedge \forall z (z \in X \rightarrow \downarrow_0(z) \in X)).$$

Moreover, the vertical successor can be defined as $\downarrow_0$ and the horizontal successor as $\rightarrow$, where

$$\rightarrow (x, y) = \exists v, w (T(x, y) \wedge \mathtt{LP}(v, x) \wedge T^0(v) \wedge +_0(v, w) \wedge \mathtt{LP}(w, y)).$$

The proof proceeds as in the proof of Theorem 5.4.1. ■

It is worth noting that the proof of Theorem 5.4.2 involves the whole DULS, and not only its first tree, as it happens in the proof of Theorem 5.4.1. We do not know whether the addition of the horizontal successor $+1$ to $\mathrm{MPL}_{\mathcal{P}}[<, (\downarrow_i)_{i=0}^{k-1}]$ has the same effect. However, since T and $+1$ are inter-definable in the monadic second-order theory, we know that $\mathrm{MSO}[<, (\downarrow_i)_{i=0}^{k-1}, +1]$ is undecidable. Summing up, we have the following theorem.

Theorem 5.4.3 (*Undefinability of global predicates*)
Global predicates are not definable in $\mathrm{MSO}[<, (\downarrow_i)_{i=0}^{k-1}]$ *over k-ary DULSs. Moreover, the extension of* $\mathrm{MSO}[<, (\downarrow_i)_{i=0}^{k-1}]$ *with any global predicate is undecidable.*

The only positive result is the following, whose proof is similar to that of Theorem 5.3.4.

Theorem 5.4.4 $\mathrm{MCL}[<, (\downarrow_i)_{i=0}^{k-1}, T, +1, \oplus 1]$ *over k-ary DULSs is decidable.*

As for UULSs, we have that $\mathrm{MSO}[<, (\downarrow_i)_{i=0}^{k-1}]$ and $\mathrm{MCL}[<, (\downarrow_i)_{i=0}^{k-1}, T, +1, \oplus 1]$ over DULSs are orthogonal in their expressiveness.

The decidability results for DULSs are summarized in Table 5.3. As in the case of UULSs, the horizontal successor $+1$ can be encoded in terms of the equi-column D.

Theorem 5.4.5 $\mathrm{MFO}[<, \downarrow_0, \downarrow_1, +1]$ *can be embedded into* $\mathrm{MFO}[<, \downarrow_0, \downarrow_1, D]$ *over binary DULSs.*

`Theory`	T	$+1$	D	$\oplus 1$
MFO	Decidable	Decidable	?	Decidable
MPL	Decidable	Decidable	?	Decidable
MCL	Decidable	Decidable	?	Decidable
$\mathrm{MFO}_{\mathcal{P}}$	?	?	Undecidable	Undecidable
$\mathrm{MPL}_{\mathcal{P}}$	Undecidable	?	Undecidable	Undecidable
$\mathrm{MCL}_{\mathcal{P}}$	Undecidable	?	Undecidable	Undecidable
MSO	Undecidable	Undecidable	Undecidable	Undecidable

Table 5.3: Decidability over downward unbounded layered structures

Proof.

We have that $+1(x, y)$ iff

$$\begin{array}{l}(\exists z(\downarrow_0(z) = x) \rightarrow \exists z(\downarrow_0(z) = x \wedge \downarrow_1(z) = y)) \wedge \\ (\exists z(\downarrow_1(z) = x) \rightarrow \exists z(\downarrow_1(z) = x \wedge \mathtt{adj}(z, y))) \wedge \\ (T^0(x) \rightarrow +_0 1(x, y)),\end{array}$$

where $\mathtt{adj}(z, y)$ says that y is the adoptive son of z, that is, y is the horizontal successor of the right son of z. We need to encode the predicate `adj` into $\mathrm{MFO}[<, \downarrow_0, \downarrow_1, D]$. We have that $\mathtt{adj}(x, y)$ iff $\phi(x, y)$, where $\phi(x, y)$ iff:

$$\begin{array}{l}\exists z_1 \exists z_2 \exists z_3 \exists z_4 \\ (T^0(z_1) \wedge D(z_1, x) \wedge \\ +_0 1(z_1, z_2) \wedge \\ \downarrow_0(z_2) = z_3 \wedge \\ D(z_3, z_4) \wedge \forall w(D(z_3, w) \wedge x \leq w \rightarrow z_4 \leq w) \wedge \\ ((D^0(x) \vee D^1(x)) \rightarrow \oplus 1(y, z_4)) \wedge \\ (\neg D^0(x) \wedge \neg D^1(x) \rightarrow z_4 = y))\end{array}$$

We prove that the above definition captures the predicate `adj`. Let $x = n_r$, $y = m_s$. Note that $\mathtt{adj}(x, y)$ if and only if $s = r + 1$ and $m = 2(n + 1)$. Suppose that $\phi(x, y)$ holds. Then, there exist z_i, for $i = 1, \ldots, 4$, such that $z_1 = n_0$, $z_2 = (n + 1)_0$, $z_3 = (2(n + 1))_1$, and $z_4 = \min\{w \mid w = (2(n + 1))_i \text{ and } i \geq 0 \text{ and } n_r \leq w\}$. We claim that $z_4 = y = (2(n + 1))_{r+1}$, and thus $\mathtt{adj}(x, y)$ holds, whenever $\neg(D^0(x) \vee D^1(x))$, and $z_4 = (2(n+1))_{r+2}$, $y = (2(n+1))_{r+1}$, and thus $\mathtt{adj}(x, y)$ holds, whenever $(D^0(x) \vee D^1(x))$. Suppose $\neg(D^0(x) \vee D^1(x))$, that is, $n > 1$. Since, for every $i, j \geq 0$, $i_j < i_{j+1}$, we only have to prove that $(2(n + 1))_{r+2} < n_r < (2(n + 1))_{r+1}$. To see that $(2(n + 1))_{r+2} < n_r$, consider the set $\{i_{r+2} \mid i \geq 0\}$. It is easy to verify that, for every $i \geq 4n$, $n_r < i_{r+2}$, and, for every $i < 4n$, $i_{r+2} < n_r$. We hence have to show that $2(n + 1) < 4n$, and this is true for $n > 1$. We now show that $n_r < (2(n + 1))_{r+1}$. It holds since $(2(n + 1))_{r+1} = +1(\downarrow_1(n_r))$, and, for every v, $v < \downarrow_1(v)$ and $v < +1(v)$. The case $(D^0(x) \vee D^1(x))$ is easier. Similarly, if $\mathtt{adj}(x, y)$ holds, then $\phi(x, y)$ holds. Hence the thesis. ∎

Since $+1$ and T are inter-definable in the monadic second-order theory of DULSs, and the same holds for $\oplus$ and D, we have the following corollary.

Corollary 5.4.6 *Let $o \in \{+1, T\}$ and $v \in \{\oplus 1, D\}$.* $\mathrm{MSO}[<, \downarrow_0, \downarrow_1, o]$ *can be embedded into* $\mathrm{MSO}[<, \downarrow_0, \downarrow_1, v]$ *over binary DULSs.*

5.5 Reconciling the algebraic and the logical frameworks

In Section 1.1, we described two different approaches to represent and reason about time granularity, namely the algebraic and the logical framework, and we pointed out the advantages and the disadvantages of both of them. In this section, we propose a unifying approach to represent and reason about a *finite* number of time granularities that captures both the naturalness of the algebraic framework and the effectiveness of the logical one.

We aim at modelling time granularity according to the following definition.

Definition 5.5.1 *A* granularity *is a mapping G from an index set $\mathbb{N}$ to subsets of a time domain $\mathbb{N}$ such that: (1) if $i < j$ then each element of $G(i)$ is less than all elements of $G(j)$, (2) if $i < j$ and $G(j)$ is not empty, then $G(i)$ is not empty, and (3) $G(i)$ is a convex interval for every $i \in \mathbb{N}$.*

The first condition states that granules in a granularity do not overlap and that their index order is the same as their time domain order. The second condition states that the subset of the index set that maps to nonempty granules forms an initial segment. The third conditions avoids granularities with gaps inside the granules. With respect to the original definition given in [8], we have fixed the index set and the time domain to be $(\mathbb{N}, <)$. Moreover, empty granules may not be followed by nonempty granules. As a consequence, empty granules, if any, form the tail of the granularity. These choices do not reduce the generality of the definition. Finally, only granularities with no gap inside the granules are treated (we will remove this assumption later). A granularity G is said:

- *externally continuous*, if there are no gaps between nonempty granules of G;
- *internally continuous*, if there are no gaps inside the granules of G;
- *total*, if the granules of G cover all the time domain;
- *uniform*, if all the granules of G have the same cardinality.

For instance, the granularity ‘days’ is internally and externally continuous, uniform and total. Granularities ‘months’ and ‘years’ are not uniform, ‘years since 2000’ is not total, ‘business week’ is not externally continuous, and ‘business month’ is not internally continuous. A number of meaningful relationships can be established among granularities. In particular, a granularity G is *finer than* a granularity H if, for every index i, there exists an index j such that $G(i) \subseteq H(j)$. A granularity G is *coarser than* a granularity H is H is finer than G. For instance, days is finer than weeks, days is finer than months, while weeks and months are incomparable.

In the following, we present an approach that uses labeled infinite sequences to represent time granularities. Let $\mathcal{G} = \{G_1, \ldots, G_n\}$ be a *finite* set of granularities (we will refer to $\mathcal{G}$ as a *calendar*) . Let $\mathcal{P}_\mathcal{G} = \{P_{G_i}, Q_{G_i} \mid 1 \leq i \leq n\}$ be a set of proposition letters associated to the calendar $\mathcal{G}$. We use $\mathcal{P}_\mathcal{G}$-labeled infinite sequences $(\mathbb{N}, <, V)$, where $V : \mathbb{N} \rightarrow 2^{\mathcal{P}_\mathcal{G}}$ is a valuation function, to represent the granularities in $\mathcal{G}$. The idea is to use, for every $G \in \mathcal{G}$, the letter P_G (resp. Q_G) to label the starting (resp. ending) point of an arbitrary granule of G. More precisely:

Definition 5.5.2 *An infinite sequence $(\mathbb{N}, <, V)$ is G-consistent whenever:*

- *if $P_G \in V(i)$ for some $i \in \mathbb{N}$, then either $Q_G \in V(i)$ or $Q_G \in V(j)$ for some $j > i$ such that $P_G \notin V(k)$ for every $i < k \leq j$ and $Q_G \notin V(k)$ for every $i < k < j$;*
- *if $Q_G \in V(i)$ for some $i \in \mathbb{N}$, then either $P_G \in V(i)$ or $P_G \in V(j)$ for some $j < i$ such that $Q_G \notin V(k)$ for every $j \leq k < i$ and $P_G \notin V(k)$ for every $j < k < i$.*

The above conditions say that every starting point of a granule of G (a point labeled with P_G) has to match with a unique ending point of that granule (a point labeled with Q_G), and every ending point of a granule of G has to match with a unique starting point of that granule. It is easy to see that every G-consistent infinite sequence induces a granularity G according to Definition 5.5.1, and vice versa. Let $\mathcal{M} = (\mathbb{N}, <, V)$ be G-consistent. For $i \in \mathbb{N}$, the i-th granule of $\mathcal{M}$ with respect to G is the set of the natural numbers in the interval $[r, s]$, for some $r, s \in \mathbb{N}$ such that $P_G \in V(r)$, $Q_G \in V(s)$ and there are exactly i natural numbers less that r such that P_G holds on them. The infinite sequence $\mathcal{M}$ induces the granularity G such that, for every $i \in \mathbb{N}$, $G(i)$ is the i-th granule of $\mathcal{M}$ with respect to G, if it exists, and $G(i) = \emptyset$ otherwise. For instance,

- the structure $(\mathbb{N}, <, V)$ such that $V(i) = \{P_G\}$ iff i is even, and $V(i) = \{Q_G\}$ iff i is odd, induces the uniform, continuous and total granularity G such that $G(i) = \{2 \cdot i, 2 \cdot i + 1\}$ for every $i \in \mathbb{N}$;
- the structure $(\mathbb{N}, <, V)$ such that $V(0) = \{P_G\}$, $V(1) = \{Q_G\}$, $V(3) = \{P_G\}$, $V(5) = \{Q_G\}$ induces the nonuniform, noncontinuous, nontotal granularity G such that $G(0) = \{0, 1\}$, $G(1) = \{3, 4, 5\}$, and $G(i) = \emptyset$ for every $i \geq 2$;
- the structure $(\mathbb{N}, <, V)$ such that $V(0) = \{P_G\}$, $V(1) = \{Q_G, P_G\}$, $V(2) = \{Q_G\}$ does not induce any granularity, since it is not G-consistent (indeed, the granules $G(0) = \{0, 1\}$ and $G(1) = \{1, 2\}$ intersect).

Similarly, a granularity G induces a G-consistent infinite sequence. Since consistent infinite sequences induce granularities, we can use a linear time logic, interpreted over infinite sequences, to define (sets of) granularities. Our choice here is Past Propositional Linear Temporal Logic (PPLTL). For instance, the following PPLTL-formula $\eta(P_G, Q_G)$ encodes the set of all granularities (according to Definition 5.5.1):

$$\eta(P_G, Q_G) = \mathbf{G}((P_G \rightarrow \alpha) \wedge (Q_G \rightarrow \beta)),$$

where

$$\begin{aligned} \alpha &= Q_G \vee \mathbf{X}(\neg(P_G \vee Q_G)\mathbf{U}(\neg P_G \wedge Q_G)) \\ \beta &= P_G \vee \mathbf{X^{-1}}(\neg(P_G \vee Q_G)\mathbf{S}(P_G \wedge \neg Q_G)). \end{aligned}$$

Moreover, the formula

$$\alpha = P_G \wedge \neg Q_G \wedge \mathbf{X}\neg P_G \wedge \mathbf{X}Q_G \wedge \mathbf{G}((P_G \leftrightarrow \mathbf{XX}P_G) \wedge (Q_G \leftrightarrow \mathbf{XX}Q_G))$$

encodes the singleton set containing the granularity G such that $G(i) = \{2 \cdot i, 2 \cdot i + 1\}$, for every $i \in \mathbb{N}$, while the formula

$$\mathbf{F}(\alpha \wedge \mathbf{X^{-1}}\texttt{true} \rightarrow \mathbf{X^{-1}}\mathbf{H}\neg(P_G \vee Q_G))$$

captures the infinite set of granularities $\{G_0, G_1, \ldots\}$ such that $G_0 = G$ and, for every $i > 0$, G_i is obtained by shifting G_0 by i positions.

Different granularities may be addressed in the same formula (but only a finite number). For instance, given a calendar $\mathcal{G} = \{G_1, \ldots, G_n\}$, the formula $\bigwedge_{i=1}^{n} \eta(P_{G_i}, Q_{G_i})$ defines the set of all calendars with n granularities. Relations between granularities belonging to the calendar $\mathcal{G}$ may be captured too. For example, the relation finer-than between two granularities G_1 and G_2 is expressed by the formula

$$\varphi(P_{G_1}, Q_{G_1}, P_{G_2}, Q_{G_2}) = \eta(P_{G_1}, Q_{G_1}) \wedge \eta(P_{G_2}, Q_{G_2}) \wedge \mathbf{G}((P_{G_1} \rightarrow \alpha) \wedge (Q_{G_1} \rightarrow \beta)),$$

where

$$\begin{aligned} \alpha &= \neg(P_{G_2} \vee Q_{G_2})\mathbf{U}(Q_{G_1} \wedge (Q_{G_2} \vee \mathbf{X}(\neg P_{G_2}\mathbf{U}Q_{G_2}))) \\ \beta &= \neg(P_{G_2} \vee Q_{G_2})\mathbf{S}(P_{G_1} \wedge (P_{G_2} \vee \mathbf{X^{-1}}(\neg Q_{G_2}\mathbf{S}P_{G_2}))) \end{aligned}$$

Hence, the formula

$$\bigwedge_{i=1}^{n} \eta(P_{G_i}, Q_{G_i}) \wedge \bigwedge_{i=1}^{n-1} \varphi(G_i, G_{i+1})$$

defines all the calendars with n granularities that are totally ordered with respect to the finer-than relation.

Besides representing (sets of) granularities and relations among them, our framework permits to effectively solve several interesting problems concerning time granularity. The *consistency problem* is the problem of deciding whether a granularity representation corresponds to a well-defined granularity (with respect to a given definition). The algorithmic solution of the consistency problem is important to avoid the definition of inconsistent granularities that may produce unexpected failures in the system. Given a PPLTL-formula $\varphi(P_G, Q_G)$, one may verify whether it encodes a set of well-defined granularities (according to Definition 5.5.1) by checking the validity of the formula $\varphi(P_G, Q_G) \rightarrow \eta(P_G, Q_G)$. The *equivalence problem* is the problem of deciding whether two different representations define the same granularity. The decidability of the equivalence problem implies the possibility of effectively testing the semantic equivalence of two different time granularity representations, making it possible to use the smaller and more tractable one. Given PPLTL formulas $\varphi_1(P_{G_1}, Q_{G_1})$ and $\varphi_2(P_{G_2}, Q_{G_2})$ representing sets of time granularities G_1 and G_2, respectively, one may verify whether G_1 and G_2 are the same by checking whether $\varphi_1(P_{G_1}, Q_{G_1}) \leftrightarrow \varphi_2(P_{G_2}, Q_{G_2})$ is valid. Finally, the *classification problem* is as follows: given a natural number n and a granularity G, is there a granule of G containing n? The classification problem is strictly related to the granule conversion problem which allows to relate granules of a given granularity to granules of another one. Let $\varphi(P_G, Q_G)$ be a PPLTL-formula representing a set of granularities G. We inductively define the temporal operator $\mathbf{X^n}p$ as follows: $\mathbf{X^0}p = p$, and, for $n > 0$, $\mathbf{X^n}p = \mathbf{X}\mathbf{X^{n-1}}p$. We have that $n \geq 0$ is contained is some granule of every granularity in G if the formula $\varphi(P_G, Q_G) \rightarrow \alpha_n(P_G, Q_G)$ is valid, where

$$\alpha_n(P_G, Q_G) = \mathbf{X^n}(P_G \vee Q_G) \vee \mathbf{X^n}(\neg(P_G \vee Q_G)\mathbf{S}P_G \wedge \neg(P_G \vee Q_G)\mathbf{U}Q_G).$$

We recall that checking the validity for a PPLTL-formula φ is equivalent to checking the nonsatisfiability for $\neg\varphi$, and the latter is decidable in polynomial space.

The above framework does not consider granularities with gaps inside the granules (only internally continuous granularities are treated). However, it can be easily extended to cope with internal gaps. We first extend the definition of granularity to cope with internal gaps.

Definition 5.5.3 *A* granularity *is a mapping G from $\mathbb{N}$ to subsets of $\mathbb{N}$ such that: (1) if $i < j$ then each element of $G(i)$ is less than all elements of $G(j)$, and (2) if $i < j$ and $G(j)$ is not empty, then $G(i)$ is not empty.*

$$\begin{array}{ccccc}
\mathcal{SL}_\omega(\text{Trellis}) & \subseteq & \text{MSO}[<, 2\times] & = & \text{MSO}[<, \downarrow_0, \downarrow_1, D] \\
\rotatebox{90}{$\subsetneq$} & & \rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} \\
\mathcal{SL}_\omega(\text{Y-Trees}) & \subsetneq & \text{MSO}[<, \texttt{adj}] & = & \text{MSO}[<, \downarrow_0, \downarrow_1, T] \\
\rotatebox{90}{$\subsetneq$} & & \rotatebox{90}{$\subsetneq$} & & \rotatebox{90}{$\subsetneq$} \\
\mathcal{SL}_\omega(\text{Binary Trees}) & = & \text{MSO}[<, \texttt{flip}] & = & \text{MSO}[<, \downarrow_0, \downarrow_1]
\end{array}$$

Figure 5.8: Systolic ω-languages and monadic second-order theories

We still model granularities as labeled infinite sequences over the extended alphabet $\mathcal{P}_\mathcal{G} = \{P_G, Q_G, P_{H_G}, Q_{H_G} \mid G \in \mathcal{G}\}$. The idea is to use letters P_G and Q_G to delimit the granules of G as before, and letters P_{H_G} and Q_{H_G} to bound the gaps inside the granules of G. Let us define a new granularity H_G whose granules are the internal gaps of the granularity G. Note that H is strictly finer than G. Indeed, every internal gap of G (a granule of H) is a proper subset of some granule of G. We extend the definition of G-consistency as follows:

Definition 5.5.4 *An infinite sequence $\mathcal{M} = (\mathbb{N}, <, V)$ is G-gap-consistent whenever:*

1. *$\mathcal{M}$ is G-consistent;*
2. *$\mathcal{M}$ is H_G-consistent;*
3. *every granule of $\mathcal{M}$ with respect to H_G is a subset of some granule of $\mathcal{M}$ with respect to G;*
4. *no granule of $\mathcal{M}$ with respect to G is the union of some granules of $\mathcal{M}$ with respect to H_G.*

It is easy to show that a G-gap-consistent infinite sequence induces a granularity (according to Definition 5.5.3) and vice versa. The set of well-defined granularities (according to Definition 5.5.3) can be encoded by a temporal formula similar to the above defined formula $\varphi(P_{H_G}, Q_{H_G}, P_G, Q_G)$ saying that G and H_G are granularities, H_G is finer than G, and no granule of G is the union of some granules of H_G.

As for the expressive power of the proposed framework, it can express all the *regular* (sets of) granularities according to the following definition. Given a granularity G, we denote by $\mathcal{M}(G)$ the G-gap-consistent infinite sequence induced by G as shown above. Moreover, given a set $\mathcal{G}$ of granularities, let $\mathcal{M}(\mathcal{G}) = \{\mathcal{M}(G) \mid G \in \mathcal{G}\}$.

Definition 5.5.5 *A granularity G is* regular *if $\{\mathcal{M}(G)\} = \mathcal{L}(A)$ for some Büchi automaton A. A set of granularities $\mathcal{G}$ is* regular *if $\mathcal{M}(\mathcal{G}) = \mathcal{L}(A)$ for some Büchi automaton A.*

From correspondence theorems linking Büchi automata to monadic theories and temporal logics, we have that a granularity is regular if and only if it is definable in MSO[<] if and only if it is definable in QLTL. Similarly for sets of granularities. Since every regular set of infinite words contains an ultimately periodic word [114], we have that every regular set of granularities contains a regular granularity.

5.6 Discussion

The results for UULSs presented in this chapter allow us to connect monadic second-order theories over binary UULSs (Figure 5.8, right column) to both monadic second-order theories over positive natural numbers (Figure 5.8, middle column) and ω-languages recognized by systolic binary tree automata, systolic Y-tree automata, and systolic trellis automata (Figure 5.8, left column). In this chapter, we established a connection between MSO$[<, 2\times]$ (resp. MSO$[<, \texttt{adj}]$) over positive natural numbers and MSO$[<, \downarrow_0, \downarrow_1, D]$ (resp. MSO$[<, \downarrow_0, \downarrow_1, T]$) over binary UULSs. Moreover, we have proved that MSO$[<, \downarrow_0, \downarrow_1, T]$ is a proper extension of MSO$[<, \downarrow_0, \downarrow_1]$ and it can be embedded into MSO$[<, \downarrow_0, \downarrow_1, D]$. The other relationships depicted in Figure 5.8 have been shown in [98, 99]. One advantage of such a connection is a different and more intuitive characterization of systolic Y-tree automata: every systolic Y-tree automaton A can be associated to a MSO$[<, \downarrow_0, \downarrow_1, T]$-formula φ_A interpreted over binary UULSs such that the models of φ_A are, modulo an isomorphism, all and only the infinite sequences accepted by A. Notice that the opposite embedding does not hold, since systolic Y-tree automata are not closed under complementation. Similarly, systolic trellis automata can be embedded into MSO$[<, \downarrow_0, \downarrow_1, D]$-formulas. In this case, the opposite embedding is an open problem interestingly related to the closure under complementation of the well-known computational complexity class NP [99].

Moreover, the results of Theorems 5.3.4 and 5.4.4 open an interesting discussion: since the global predicates T, $+1$, and $\oplus 1$ are not definable in MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$ over UULSs and DULSs, we have that MCL$[<, (\downarrow_i)_{i=0}^{k-1}, T, +1, \oplus 1]$ can express new properties that are not specifiable in MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$. On the other hand, since MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$ admits full second-order quantification, while MCL$[<, (\downarrow_i)_{i=0}^{k-1}, T, +1, \oplus 1]$ restricts second-order quantification to chains, there are properties that can be expressed in MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$ but not in MCL$[<, (\downarrow_i)_{i=0}^{k-1}, T, +1, \oplus 1]$. Of course, the intersection of the definable properties in the two theories is not empty: for instance, every MCL$[<, (\downarrow_i)_{i=0}^{k-1}]$-definable property can be expressed in both MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$ and MCL$[<, (\downarrow_i)_{i=0}^{k-1}, T, +1, \oplus 1]$. We conclude that MSO$[<, (\downarrow_i)_{i=0}^{k-1}]$ and MCL$[<, (\downarrow_i)_{i=0}^{k-1}, T, +1, \oplus 1]$ are incomparable from an expressiveness point of view. Since both of them are (nonelementary) decidable, they can be regarded as orthogonal languages for time granularity.

6 Conclusions and open problems

To goal of this thesis was to find expressive, flexible and effective tools to reason about time granularity in a logical framework. We applied a divide and conquer methodology: problems are split into sub-problems and these are delegated to well-known modules. The main advantage of this solution is the reuse of tools like proofs of algorithms to obtain new theorems and software. It is worth noting that the divide and conquer approach proposed in this thesis is not tailored to time granularity. We feel that it can be useful in other frameworks. One example is mobile computing, where the interaction of temporal and spatial constraints is a major requirement.

The main contributions of this thesis can be summarized as follows:

- we studied various combinations of temporal logics and we proposed an approach to combine automata which can be considered as the automata-theoretic counterpart of the combining method for temporal logics known as temporalization. We studied the transfer of properties like closure under Boolean operations, decidability, and expressive equivalence from component to combined automata.
- we devised and implemented model checking procedures for combined temporal logics and we analyzed their computational complexity;
- we provided combined temporal logics and combined automata for finitely and infinitely layered structures. We studied their expressive power and their computational complexity by using transfer theorems for temporalized automata;
- we extended classical monadic logics over layered structures with new meaningful predicates and studied the decidability of the resulting theories.

The main open problems related to this thesis are listed in the following:

- we mostly used temporalized logics and hence we developed an automata-theoretic counterpart of temporalization. However, as pointed out in Chapter 3, there are at least two other popular ways of combining logics: independent combination and join. It would be interesting to study automata-theoretic counterparts of independent combination and join as well. More generally, one can study forms of automata combination in which there is a stronger interaction between components (there is few component interaction in temporalization).

- The decomposition method have been successfully used to prove expressive completeness results for monadic path theories over trees [61], over upward unbounded layered structures (Theorem 4.1.10), and over downward unbounded layered structures (Theorem 4.3.2). These proofs resemble a lot, and there should be a general method underlying them. There are two (related) key issues in understanding this method. The first is the kind of binary predicates that $\mathcal{L}$ has in its signature, the second in the type of quantification (first-order, path, chain, second-order, ...) that $\mathcal{L}$ allows. Why is the prefix ordering predicate over trees well-behaved with respect the decomposition strategy, while the lexicographical ordering one is not? Why does the decomposition method work with path quantification, but it does not naturally extend to second-order quantification?

- In Chapter 4 we have proposed automata over upward unbounded layered structures. An automaton over UULSs embeds finite tree automata, working over finite trees rooted at the nodes of the leftmost path of the UULS, into an infinite sequence automaton, that scans the left most path of the UULS. We have shown that the emptiness problem for these automata is in 2EXPTIME, but we did not prove that this complexity bound is tight. We feel that the complexity lower bound for this problem is lower.

- As an alternative, one may consider the following notion of automata accepting UULSs. Consider bottom-up finite tree automata as defined in Chapter 2. They accept finite trees working bottom-up, from the leaves to the root of the tree. Recall that an UULS can be viewed as a rootless infinite tree generated from the leaves. Hence, we can define bottom-up *infinite* tree automata working over UULSs. The acceptance condition is a Büchi acceptance condition over the leftmost path of the UULS. We conjecture that this notion of automata over UULSs is expressively equivalent to the notion defined in Chapter 4. Moreover, it would be interesting to compare bottom-up infinite tree automata over UULSs and systolic tree automata over infinite sequences. The two automata classes essentially work in the same way: they process an UULS in a bottom-up way starting from the leaves and accepting over the leftmost path. They differ because they accept different structures (UULSs and infinite sequences). We feel that many problems for systolic tree automata over infinite sequences, like emptiness and closure under Boolean operations, can be reduced to similar problems for bottom-up infinite tree automata.

- In Chapter 5 we tried to extend different monadic theories with new meaningful predicates while preserving decidability. A number of resulting decidability problems are still open. In particular, a beautiful positive result states that monadic chain logic, extended with equi-level, horizontal successor and vertical successor is still decidable both over DULSs and UULSs. This logic can express new properties that cannot be captured in monadic second-order logic over DULSs and UULSs. The main open problem in this chapter concerns the decidability of monadic chain logic extended with the equi-column predicate. We conjecture that the resulting theory is undecidable. Moreover, the extension of monadic chain logic with equi-level, horizontal successor and vertical successor is nonelementarily decidable. It would be useful to devise an expressively complete and elementarily decidable temporal logic counterpart. A possible candidate is the independent combination of PLTL and $\mathrm{CTL}^*_{\mathrm{k}}$: the former runs horizontally over the layers of the structures, the latter works vertically over the refinement trees of the structures. Moreover, the decision procedure for $\mathrm{MCL}[<, (\downarrow_i)_{i=0}^{k-1}, T, +1, \oplus 1]$ exploits an embedding into MSO[$<$] over infinite sequences. However, we know that MSO[$<$, `flip`] properly extends MSO[$<$] over infinite sequences and that it is still decidable. This suggests that MCL[$<, (\downarrow_i)_{i=0}^{k-1}, T, +1, \oplus 1$] can be further extended without losing the decidability property.

Bibliography

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In P. S. Yu and A. L. P. Chen, editors, *Proceedings of the International Conference on Data Engineering*, pages 3–14. IEEE Press, 6–10 March 1995.

[2] R. Alur and T. A. Henzinger. Logics and models of real time: A survey. *Lecture Notes in Computer Science*, 600:74–106, 1992.

[3] F. Baader and H. Ohlbach. A multidimensional terminological knowledge representation language. *Applied NonClassical Logic*, 5:153–197, 1995.

[4] C. Bettini, A. Brodsky, S. Jajodia, and X. S. Wang. Logical design for temporal databases with multiple granularities. *ACM Transactions on Database Systems*, 22(2):115–170, June 1997.

[5] C. Bettini, S. Jajodia, J. Lin, and X. S. Wang. Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):222–237, 1998.

[6] C. Bettini, S. Jajodia, and X. Wang. A general framework and reasoning models for time granularity. In *Proceedings of the International Workshop on Temporal Representation and Reasoning*, pages 104–111. IEEE Computer Society Press, 1996.

[7] C. Bettini, S. Jajodia, and X. S. Wang. Testing complex temporal relationships involving multiple granularities and its application to data mining. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, volume 15, pages 68–78. ACM Press, 1996.

[8] C. Bettini, S. Jajodia, and X. S. Wang. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer, 2000.

[9] C. Bettini and R. De Sibi. Symbolic representation of user-defined time granularities. *Annals of Mathematics and Artificial Intelligence*, 30(1-4):53–92, 2000.

[10] P. Blackburn and J. Bos. *Representation and Inference for Natural Language*. Studies in Logic, Language, and Information. CSLI Press. Forthcoming.

[11] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer, Berlin, 1997.

[12] J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlag. Math.*, 6:66–92, 1960.

[13] J. R. Büchi. On a decision method in restricted second-order arithmetic. In *Proceedings of the International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[14] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, June 2000.

[15] R. Chandra, A. Segev, and M. Stonebraker. Implementing calendars and temporal rules in next generation databases. In A. K. Elmagarmid and E. Neuhold, editors, *Proceedings of the International Conference on Data Engineering*, pages 264–273, Houston, TX, February 1994. IEEE Computer Society Press.

[16] Z. Chouchen and M. R. Hansen. An adequate first order interval logic. *Lecture Notes in Computer Science*, 1536:584–608, 1998.

[17] E. Ciapessoni, E. Corsetti, A. Montanari, and P. San Pietro. Embedding time granularity in a logical specification language for synchronous real-time systems. *Science of Computer Programming*, 20:141–171, 1993.

[18] E. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[19] E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In D. Kozen, editor, *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, Yorktown Heights, New York, May 1981. Springer.

[20] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. In *Conference Record of the Annual ACM Symposium on Principles of Programming Languages*, pages 117–126. ACM, ACM, January 1983.

[21] C. Combi, F. Pinciroli, and G. Pozzi. Managing time granularity of narrative clinical information: The temporal data model TIME-NESIS. In *Proceedings of the International Workshop on Temporal Representation and Reasoning*, pages 88–93. IEEE Computer Society Press, 1996.

[22] E. Corsetti, E. Crivelli, D. Mandrioli, A. Montanari, A. Morzenti, P. San Pietro, and E. Ratto. Dealing with different time scales in formal specifications. In *Proceedings of the International Workshop on Software Specification and Design*, pages 92–101. IEEE Computer Society Press, 1991.

[23] E. Corsetti, A. Montanari, and E. Ratto. Dealing with different time granularities in formal specifications of real-time systems. *The Journal of Real-Time Systems*, 3:191–215, 1991.

[24] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. In Edmund M. Clarke and Robert P. Kurshan, editors, *Proceedings of Computer-Aided Verification*, volume 531 of *Lecture Notes in Computer Science*, pages 233–242, Berlin, Germany, 1991. Springer.

[25] D. Cukierman and J. Delgrande. Expressing time intervals and repetition within a formalization of calendars. *Computational Intelligence*, 14(4):563–597, 1998.

[26] U. Dal Lago and A. Montanari. Calendars, time granularities, and automata. In *Proceedings of the International Symposium on Spatial and Temporal Databases*, volume 2121 of *Lectures Notes on Computer Science*, pages 279–298, Los Angeles, CA, USA, 2001.

[27] T. Dean and D. V. McDermott. Temporal data base management. *Artificial Intelligence*, 32:1–55, 1987.

[28] J. E. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.

[29] W. Dreyer, A. K. Dittrich, and D. Schmidt. Research perspectives for time series management systems. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(1):10–15, March 1994.

[30] C. E. Dyreson and R. T. Snodgrass. Temporal granularity. In R. T. Snodgrass, editor, *The TSQL2 Temporal Query Language*, pages 347–385. Kluwer Academic Press, 1995.

[31] H. D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 1995.

[32] C. C. Elgot. Decision problems for finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.

[33] C. C. Elgot and M. O. Rabin. Decidability and undecidability of second (first) order theory of generalized successor. *Journal of Symbolic Logic*, 31:169–181, 1966.

[34] E. A. Emerson and C. S. Jutla. The Complexity of tree Automata and Logics of Programs. In *In Proceedings op the Annual IEEE-CS Symposium in Foundations of Computer Science*, pages 328–337, 1988.

[35] E. A. Emerson and A. P. Sistla. Deciding full branching time logic. *Information and Control*, 61(3):175–201, June 1984.

[36] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 995–1072. Elsevier Science Publishers B.V., 1990.

[37] J. Engelfriet. Minimal temporal epistemic logic. *Notre Dame Journal of Formal Logic*, 37:233–259, 1996.

[38] J. Euzenat. An algebraic approach for granularity in qualitative space and time representation. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 894–900. Morgan Kaufmann, 1995.

[39] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge.* MIT Press, Cambridge, MA, 1995.

[40] J. L. Fiadeiro and T. Maibaum. Sometimes ”tomorrow” is ”sometime”: Action refinement in a temporal logic of objects. In Dov M. Gabbay and Hans Jürgen Ohlbach, editors, *Proceedings of the International Conference on Temporal Logic*, volume 827 of *Lectures Notes on Artificial Intelligence*, pages 48–66, Berlin, July 1994. Springer.

[41] K. Fine and G. Schurz. Transfer theorems for multimodal logics. In J. Copeland, editor, *Logic and Reality: Essays on the Legacy of Arthur Prior*, pages 169–213. Oxford University Press, Oxford, 1996.

[42] M. Finger. Handling database updates in two-dimensional temporal logic. *Journal of Applied Non-Classical Logics*, 2(2):201–224, 1992.

[43] M. Finger. *Changing the past: database applications of two-dimensional executable temporal logics*. PhD thesis, Imperial College, Department of Computing, 1994.

[44] M. Finger and D. M. Gabbay. Adding a temporal dimension to a logic system. *Journal of Logic Language and Information*, 1:203–233, 1992.

[45] M. Finger and D. M. Gabbay. Combining temporal logic systems. *Notre Dame Journal of Formal Logic*, 37:204–232, 1996.

[46] M. Finger and M. Reynolds. Two-dimensional executable temporal logic for bitemporal databases. In *Proceedings of Advances in Temporal Logic*, pages 393–411. Kluver Academic, 2000.

[47] D. Foster, B. Leban, and D. McDonald. A representation for collections of temporal intervals. In *Proceedings of the American National Conference on Artificial Intelligence*, pages 367–371, 1986.

[48] M. Franceschet and A. Montanari. Branching within time: an expressively complete and elementarily decidable temporal logic for time granularity. *Journal of Language and Computation*. To appear.

[49] M. Franceschet and A. Montanari. A combined approach to temporal logics for time granularity. In *Proceedings of the Workshop Methods for Modalities*, 2001.

[50] M. Franceschet and A. Montanari. Towards an automata-theoretic counterpart of combined temporal logics. In *Proceedings of the International Workshop on Verification and Computational Logic*, pages 55–74, 2001.

[51] M. Franceschet, A. Montanari, and M. de Rijke. Model checking for combined logics. In *Proceedings of the International Conference on Temporal Logic*, pages 65–73, 2000.

[52] M. Franceschet, A. Montanari, A. Peron, and G. Sciavicco. Definability and decidability of binary predicates for time granularity. Technical Report 2/2002, Department of Mathematics and Computer Science, University of Udine – Italy, 2002.

[53] D. Fum, G. Guida, A. Montanari, and C. Tasso. Using levels and viewpoints in text representation. In *Proceedings of the International Conference on Artificial Intelligence and Information-Control Systems of Robots*, pages 37–44, 1989.

[54] D. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyaschev. *Many-dimensional modal logics: theory and applications*. Forthcoming.

[55] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 163–173, 1980.

[56] D. M. Gabbay and M. de Rijke, editors. *Frontiers of Combining Systems 2*, volume 7 of *Studies in Logic and Computation.* Research Studies Press/Wiley, 2000.

[57] D. M. Gabbay and V. Shehtman. Products of modal logics, part I. *Logic Journal of the IGPL*, 6:73–146, 1998.

[58] F. Gécseg and M. Steinby. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.

[59] J. Greer and G. McCalla. A computational framework for granularity and its application to educational diagnosis. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 477–482. Morgan Kaufmann, 1989.

[60] J. Gruska. Synthesis, structure and power of systolic computations. *Theoretical Computer Science*, 71(1):47–77, March 1990.

[61] T. Hafer and W. Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In T. Ottmann, editor, *Proceedings of the International Colloquium Automata, Languages and Programming*, volume 267 of *Lecture Notes in Computer Science*, pages 269–279, Karlsruhe, Germany, 1987. Springer.

[62] J. H. Halpern and M. Y. Vardi. Model checking vs. theorem proving: A manifesto. In *Proceedings of the Conference on Principles of Knowledge Representation and Reasoning*, pages 325–334, 1991.

[63] J. Y. Halpern and Y. Shoham. A propositional modal logic of time intervals. *Journal of the ACM*, 38(4):935–962, October 1991.

[64] J. Y. Halpern and M. Y. Vardi. The complexity of reasoning about knowledge and time I: Lower bounds. *Journal of Computer and System Sciences*, 38(1):195–237, 1989.

[65] D. Harel. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *Journal of the ACM*, 33(1):224–248, January 1986.

[66] G. J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, Englewood Cliffs, NJ, 1991.

[67] G. J. Holzmann and D. Peled. The state of spin. In *Proceedings of the International Conference on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 385–389, New Brunswick, NJ, USA, 1996. Springer.

[68] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, N. Reading, MA, 1980.

[69] N. Immerman and D. Kozen. Definability with bounded number of bound variables. *Information and Computation*, 83(2):121–139, 1989.

[70] S. Jajodia, W. Litwin, and G. Wiederhold. Dealing with granularity of time in temporal databases. In R. Andersen, J. A. Bubenko jr., and A. Solvberg, editors, *Advanced Information Systems Engineering*, pages 124–140. Springer, Berlin, 1991.

[71] S. Jajodia, W. Litwin, and G. Wiederhold. Integrating temporal data in a heterogeneous environment. In A. Tansel et al., editor, *Temporal Databases: Theory, Design and Implementation*, pages 563–579. Database Systems and Applications Series, Benjamin/Cummings Pub. Co., Redwood City, CA, 1993.

[72] S. Jajodia, V. S. Subrahmanian, and X. S. Wang. Temporal modules: An approach toward federated temporal databases. *Information Sciences*, 82:103–128, 1995.

[73] C. Jard and T. Jeron. On-line model checking for finite linear temporal logic specifications. In *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science, pages 189–196, Grenoble, France, 1989. Springer.

[74] C.B. Jones. Specification and design of (parallel) programs. In *Information Processing 83: Proceedings IFIP World Congress*, pages 321–332, 1983.

[75] H. Kamp. *Tense logic and the theory of linear order.* PhD thesis, University of California, Los Angeles, 1968.

[76] M. Kracht and F. Wolter. Properties of independently axiomatizable bimodal logics. *Journal of Symbolic Logic*, 56(4):1469–1485, 1991.

[77] O. Kupferman. Augmenting branching temporal logics with existential quantification over atomic propositions. *Journal of Logic and Computation*, 9(2):135–147, 1999.

[78] O. Kupferman and M. Y. Vardi. Modular model checking. In *COMPOS'97*, 1997.

[79] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, March 2000.

[80] R. Kurshan. *Computer-aided Verification of Coordinating Processes: The Automata-Theoretic Approach.* Princeton University Press, 1994.

[81] A. Kurucz. $S5^3$ lacks the finite model property. In *Proceedings of the International Conference on Temporal Logic*, 2000.

[82] P. Ladkin. The completeness of a natural system for reasoning with time intervals. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 462–467. Morgan Kaufmann, 1987.

[83] L. Lamport. Specifying concurrent program modules. *ACM Transaction on Programming Language and Systems*, 5:190–222, 1983.

[84] L. Lamport. On interprocess communication. Technical Report Research Report 8, SRC, Palo Alto, CA, 1985.

[85] H. Läuchli and C. Savoiz. Monadic 2nd-order definable relations on the binary tree. *Journal of Symbolic Logic*, 52:219–226, 1987.

[86] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In R. Parikh, editor, *Proceedings of the Conference on Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218, Brooklyn, NY, June 1985. Springer.

[87] Z. Manna and A. Pnueli. Specification and verification of concurrent programs by $\forall$ automata. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, pages 1–12, 1987.

[88] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering Frequent Episodes in Sequences. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, Montreal, Canada, August 1995. AAAI Press.

[89] M. Marx. Complexity of products of modal logics. *Journal of Logic and Computation*, 9:221–238, 1999.

[90] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

[91] J. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Science*. Cambridge University Press, 1995.

[92] A. Montanari. *Metric and Layered Temporal Logic for Time Granularity*. ILLC Dissertation Series 1996-02, Institute for Logic, Language and Computation, University of Amsterdam, 1996.

[93] A. Montanari and B. Pernici. Temporal reasoning. In A. Tansell, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal Databases: Theory, Design and Implementation*, Database Systems and Applications Series, chapter 21, pages 534–562. Benjamin/Cummings Pub. Co., Redwood City, CA, 1993.

[94] A. Montanari, A. Peron, and A. Policriti. Extending Kamp's theorem to model time granularity. *Journal of Logic and Computation*. To appear.

[95] A. Montanari, A. Peron, and A. Policriti. Decidable theories of ω-layered metric temporal structures. *Logic Journal of the IGPL*, 7(1):79–102, 1999.

[96] A. Montanari, A. Peron, and A. Policriti. The taming (timing) of the states. *Logic Journal of the IGPL*, 8(5):681–699, 2000.

[97] A. Montanari and A. Policriti. Decidability results for metric and layered temporal logics. *Notre Dame Journal of Formal Logic*, 37:260–282, 1996.

[98] A. Monti and A. Peron. Systolic tree ω-languages: the operational and the logical view. *Theoretical Computer Science*, 23:1–17, 2000.

[99] A. Monti and A. Peron. Logical definability of Y-tree and trellis systolic ω-languages. *Acta Cybernetica*, 15:75–100, 2001.

[100] B. Moszkowski. *Reasoning about digital circuits*. PhD thesis, Department of Computer Science, University of Stanford, 1983.

[101] E. Mota and D. Robertson. Representing interaction of agents at different time granularities. In *Proceedings of the International Workshop on Temporal Representation and Reasoning*, pages 72–79. IEEE Computer Society Press, 1996.

[102] M. Niezette and J. Stevenne. An efficient symbolic representation of periodic time. In *Proceeding of the International Conference on Information and Knowledge Management*, volume 752 of *Lecture Notes in Computer Science*, pages 161–168, Baltimore, Maryland, November 1993.

[103] P. Ning, S. Jajodia, and X. S. Wang. An algebraic representation of calendars. *Annals of Mathematics and Artificial Intelligence*. To appear.

[104] A. Pnueli. The temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[105] M. Poesio and R. J. Brachman. Metric constraints for maintaining appointments: Dates and repeated activities. In *Proceedings of the National Conference on Artificial Intelligence*, pages 253–259. MIT Press, July 1991.

[106] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the International Symposium in Programming*, pages 195–220, 1981.

[107] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[108] S. Safra. On the complexity of ω-automata. In *In Proceedings of the Annual Symposium on Foundations of Computer Science*, pages 319–327, White Plains, New York, 24–26 October 1988. IEEE.

[109] A. Segev and R. Chandra. A data model for time-series analysis. *Lecture Notes in Computer Science*, 759:191–212, 1993.

[110] Y. Shahar. Dynamic temporal interpretation contexts for temporal abstraction. In *Proceedings of the International Workshop on Temporal Representation and Reasoning*, pages 64–71. IEEE Computer Society Press, 1996.

[111] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[112] E. Spaan. *Complexity of Modal Logics*. PhD thesis, Department of Mathematics and Computer Science, University of Amsterdam, 1993.

[113] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of 2nd-order logic. *Mathematical Systems Theory*, 2(1):57–81, March 1968.

[114] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 133–191. Elsevier Science Publishers, 1990.

[115] W. Thomas. Infinite trees and automaton definable relations over ω-words. In *Proceedings Annual Symposium on Theoretical Aspects of Computer Science*, pages 263–277. Springer, 1990.

[116] M. Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *In Proceedings of the ACM Symposium on Theory of Computing*, pages 240–251, Baltimore, USA, May 1985. ACM Press.

[117] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *In Proceedings of the Symposium on Logic in Computer Science*, pages 332–345, Washington, D.C., USA, 1986. IEEE Computer Society Press.

[118] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[119] Y. Venema. A Modal Logic for Chopping Intervals. *Journal of Logic and Computation*, 1(4):453–476, 1991.

[120] N. Vitacolonna. Decidability of interval temporal logics over split-frames via granularity. Technical Report 1/2002, Department of Mathematics and Computer Science, University of Udine – Italy, 2002.

[121] J. Wijsen. Design of temporal relational databases based on dynamic and temporal functional dependencies. In S. Clifford and A. Tuzhilin, editors, *Recent Advances in Temporal Databases*, pages 61–76, Zurich, Switzerland, September 1995. Springer.

[122] J. Wijsen. Reasoning about qualitative trends in databases. *Information Systems*, 23(7):469–493, 1998.

[123] J. Wijsen. Temporal FDs on complex objects. *ACM Transactions on Database Systems*, 24(1):127–176, March 1999.

[124] J. Wijsen. A string based-model for infinite granularities. In *Proceedings of the AAAI Workshop on Spatial and Temporal Granularity*, pages 9–16. AAAI Press, 2000.

[125] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, January/February 1983.

[126] F. Wolter. The finite model property in tense logic. *The Journal of Symbolic Logic*, 60(3):757–774, 1995.

[127] F. Wolter. A counterexample in tense logic. *Notre Dame Journal of Formal Logic*, 37(2):167–173, Spring 1996.

[128] F. Wolter. Completeness and decidability of tense logics closely related to logics above K4. *The Journal of Symbolic Logic*, 62(1):131–158, March 1997.

[129] F. Wolter. Fusions of modal logics revisited. In M. Kracht, M. de Rijke, H. Wansing, and M. Zakharyaschev, editors, *Advances in Modal Logic.* CSLI, Stanford, CA, 1998.

[130] F. Wolter. The product of converse pdl and polymodal K. *Journal of Logic and Computation*, 10(2):223–251, 2000.

[131] F. Wolter and M. Zakharyaschev. Satisfiability problem in description logics with modal operators. In *Proceedings of the Conference on Principles of Knowledge Representation and Reasoning*, pages 512–523. Moragan Kaufman, 1998.