Laboratoire
d'Informatique
Algorithmique:
Fondements et
Applications

Université Paris Diderot
École Doctorale Paris Centre

Thèse de doctorat

Discipline : Informatique

présentée par

Antoine Durand-Gasselin

Automata Based Logics for Program Verification

dirigée par Peter Habermehl et Ahmed Bouajjani

Soutenue publiquement le 3 décembre 2013 devant un jury composé de:

M. Jean-Éric Pin	C.N.R.S.	Président du jury
M. Peter Habermehl	Université Paris Diderot	Directeur de thèse
M. Ahmed Bouajjani	Université Paris Diderot	Co-Directeur de thèse
M^{me} Orna Kupferman	Université Hébraïque de Jérusalem	Rapporteur du jury
M. Dietrich Kuske	Université Technique d'Ilmenau	Membre du jury
M. Jean-François Raskin	Université libre de Bruxelles	Rapporteur du jury
M. Igor Walukiewicz	C.N.R.S.	Membre du jury

Laboratoire d'Informatique Algorithmique: Fondements et Applications
CNRS UMR 7089, Univ Paris Diderot
75205 Paris Cedex 13

École Doctorale de Paris Centre
4 place Jussieu
75 252 Paris cedex 05

Remerciements

Je tiens tout d'abord à remercier Ahmed et Peter, grâce à qui j'ai pu effectuer cette thèse. Leur encadrement, au début, puis leurs conseils, par la suite, puis leur supervision quand il a fallu rédiger, ont été excellents et m'ont permis d'aboutir à ce manuscrit.

Merci Ahmed aussi de m'avoir envoyé à Philly!

Je tiens également à remercier Noëlle Delgado, pour tout le support administratif et logistique qu'elle a pu nous apporter.

Je tiens également à remercier François Laroussinie, et Claire David qui m'ont donné des petits coups de pouce, à un moment où j'en avais bien besoin.

Enfin, je tiens à remercier tous les chercheurs du LIAFA et de PPS (stagiaires, doctorants, post-doctorants ou permanents) que j'ai cotoyés et avec lesquels j'ai eu l'occasion de travailler (ou pas), en tout cas avec lesquels j'ai eu d'intéressantes discussions, et grâce à qui le laboratoire a toujours été un endroit où je me plaisais.

Résumé

Résumé

La vérification formelle de programmes consiste à raisonner rigoureusement, à l'aide de logiques mathématiques sur des programmes informatiques afin de démontrer leur validité vis-à-vis d'une certaine spécification. Cela permet d'obtenir une très forte assurance d'absence de bug.

Le choix du cadre logique dans lequel effectuer la vérification d'un programme est dicté par deux impératifs: premièrement la logique doit être suffisamment expressive pour exprimer la spécification voulue et rendre compte de chacune des instructions du programme, deuxièmement elle doit être suffisamment peu complexe, afin de pouvoir automatiser autant que possible (et en temps raisonable), le processus de vérification.

Dans cette thèse, nous étudions deux types de logiques et comment les manipuler pour la vérification formelle. D'abord la logique du premier ordre sur les structures automatiques, par exemple l'Arithmétique de Presburger, qui permet d'exprimer des propriétés sur les entiers. Nous montrons qu'elles peuvent être efficacement représentées et manipulées avec des automates, typiquement nous détaillons un algorithme générique basé sur les automates qui permet de décider entre autres l'Arithmétique de Presburger avec une complexité proche de sa difficulté.

Ensuite, nous présentons des transducteurs finis avec des registres en écriture seule, et étudions leur pouvoir expressif. Nous verrons qu'ils sont une extension élégante des automates finis pour définir des transformations de mots, étant un modèle calculatoire simple et intuitif, et nous établissons qu'ils permettent d'implémenter les transformations définissables par la logique monadique du second ordre.

Mots-clefs

vérification formelle, automates, logique, structures automatiques, transducteurs finis, Ehrenfeucht-Fraïssé

Automata-based Techniques for Program Verification

Abstract

Formal software verification consists in a rigorous analysis of programs using mathematical logic, in order to demonstrate they meet their specification, and ensure with strong confidence in the absence of bugs.

The choice of the logical framework to perform the analysis is crucial, and must address two issues: not only must the logic be expressive enough to express the specification and reflect the action of each instruction, but also the logic formalism must be simple enough in order to automate efficiently as much as possible the verification process.

In this thesis we will study two logical frameworks and show how to manipulate those. First we will focus on first-order logics over automatic structures, such as Presburger Arithmetics which allows to express properties about integers. We will show that formulas can be effectively represented as automata, and we will detail a generic automata-based algorithm that allow to decide such first-order logics, whose complexity closely matches the hardness of those logics.

Then we will present streaming string transducers, which are automata with a finite number of write-only registers and we will inspect their expressiveness. We will present this elegant extension of finite state automata to define string transformations as an intuitive computational model and we will establish their expressive power is precisely express transformations definable by monadic second-order logics.

Keywords

formal verification, automata, logics, automatic structures, streaming transducers, Ehrenfeucht-Fraïssé

Contents

Introduction

Computers are a recent invention and with more than half century dedicated to their constant improvement, they have become faster and cheaper, consequently software systems now play an ubiquitous role in modern society. They are used for a great variety of purposes, and often dedicated to some very critical tasks.

As computers have gotten faster and smaller, and programs larger and more and more complex, the latter are usually to be blamed for dysfunctions of software systems. Indeed a computer carries out the sequence of instructions it was programmed to, with no big picture of the task it is intended to perform, thus if this sequence is faulty it will be executed nonetheless, which usually results in failing the task.

Program verification is ensuring that a program is correct with respect to a specification of its intended purpose. There are different possible approaches, for instance dynamic analysis, which consists of intensively testing the software by running under realistic conditions: if no unexpected behavior is exhibited, then surely the software should be correct. Though this method may prove efficient to exhibit some bugs, it does not allow to ensure the same level of confidence as static analysis, which consists in proving that a program satisfies its specification, and hence deduce that no unexpected behaviour shall occur.

In order to achieve strong confidence into a program's behaviour, a rigorous analysis is required. The specification is expressed in a formal logic, and program verification consists in proving that the program satisfies its specification. Hoare [Hoa69] presented a very generic method to perform such proofs, based on the knowledge of the properties of each elementary operation, to state that a given sequence of instructions, some precondition, will yield some postcondition.

We present in Figure 1 an example of verification of some program. First, we express the specification of the program in terms of a precondition at the beginning of the program (here the precondition that x_1, x_2, y_1, y_2 contain the values with which the function was called), will yield some postcondition (here that the returned value is $|x_1^i - x_2^i| + |y_1^i - y_2^i|$; $x_1^i, x_2^i, y_1^i, y_2^i$ designating the integer values with which the function was called). Then we annotate each intermediate step of the program. Finally we check that each annotation is compatible with each intermediate step of the program. The inference rules are given in Figure 2.

This method however requires a lot of effort, and is hardly feasible by hand even for very small programs (dozens of lines of code).

However part of this process can be automatized, but to do so, one must first fix a logical framework, obviously expressive enough so that is allows to express the desired specification (precondition and postcondition) of the program. The logical framework has to be decidable, so that we can perform the last step of the verification

```
int distance(int x1, int y1, int x2, int y2) {
    $x_1 = x_1^i \land x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$
    int d = 0;
    $d = 0 \land x_1 = x_1^i \land x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$ $\mathsf{pre}_1^i$
    if (x1 < x2) $t_1^i$ {
        $\mathsf{pre}_1^i \land t_1^i \implies d = x_1 - x_1^i \land x_1^i \le x_1 \le x_2^i \land x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$ $I_1$
        while (x1 < x2) $t_1^w$ {
            $t_1^w \land I_1 \implies x_1 < x_2 \land d = x_1 - x_1^i \land x_1^i \le x_1 \le x_2^i \land x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$
            x1++; d++;
            $x_1 - 1 < x_2 \land d - 1 = x_1 - 1 - x_1^i \land x_1^i \le x_1 - 1 \le x_2^i \land x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$
            $\implies d = x_1 - x_1^i \land x_1^i \le x_1 \le x_2^i \land x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$
        }
        $\neg t_1^w \land I_1 \implies \neg x_1 < x_2 \land d = x_1 - x_1^i \land x_1^i \le x_1 \le x_2^i \land x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$
        $\implies d = |x_1^i - x_2^i| \land x_1 = x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$
    } else {
        $\mathsf{pre}_1^i \land \neg t_1^i \implies d = x_1^i - x_1 \land x_1^i \ge x_1 \ge x_2^i \land x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$ $I_2$
        while (x1 > x2) $t_2^w$ {
            $t_2^w \land I_2 \implies x_1 > x_2 \land d = x_1^i - x_1 \land x_1^i \ge x_1 \ge x_2^i \land x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$
            x1--; d++;
            $x_1 + 1 > x_2 \land d - 1 = x_1^i - (x_1 + 1) \land x_1^i \ge x_1 + 1 \ge x_2^i \land x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$
            $\implies d = x_1^i - x_1 \land x_1^i \ge x_1 \ge x_2^i \land x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$
        }
        $\neg t_2^w \land I_2 \implies d = |x_1^i - x_2^i| \land x_1 = x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$
    } $d = |x_1^i - x_2^i| \land x_1 = x_2 = x_2^i \land y_1 = y_1^i \land y_2 = y_2^i$ $\mathsf{pre}_2^i$
    if (y1 < y2) $t_2^i$ {
        $\mathsf{pre}_2^i \land t_2^i \implies d = |x_1^i - x_2^i| + y_1 - y_1^i \land x_1 = x_2 = x_2^i \land y_1^i \le y_1 \le y_2^i \land y_2 = y_2^i$ $I_3$
        while (y1 < y2) $t_3^w$ {
            $t_3^w \land I_3 \implies y_1 < y_2 \land d = |x_1^i - x_2^i| + y_1 - y_1^i \land y_1^i \le y_1 \le y_2^i \land x_1 = x_2 = x_2^i \land y_2 = y_2^i$
            y1++; d++;
            $y_1 - 1 < y_2 \land d - 1 = |x_1^i - x_2^i| + (y_1 - 1) - y_1^i \land y_1^i \le y_1 - 1 \le y_2^i \land x_1 = x_2 = x_2^i \land y_2 = y_2^i$
            $\implies d = |x_1^i - x_2^i| + y_1 - y_1^i \land y_1^i \le y_1 \le y_2^i \land x_1 = x_2 = x_2^i \land y_2 = y_2^i$
        }
        $\neg t_3^w \land I_3 \implies d = |x_1^i - x_2^i| + |y_1^i - y_2^i| \land x_1 = x_2 = x_2^i \land y_1 = y_2 = y_2^i$
    } else {
        $\mathsf{pre}_2^i \land \neg t_2^i \implies d = |x_1^i - x_2^i| + y_1^i - y_1 \land x_1 = x_2 = x_2^i \land y_1^i \ge y_1 \ge y_2^i \land y_2 = y_2^i$ $I_4$
        while (y1 > y2) $t_4^w$ {
            $t_4^w \land I_4 \implies y_1 > y_2 \land d = |x_1^i - x_2^i| + y_1^i - y_1 \land y_1^i \ge y_1 \ge y_2^i \land x_1 = x_2 = x_2^i \land y_2 = y_2^i$
            y1--; d++;
            $y_1 + 1 > y_2 \land d - 1 = |x_1^i - x_2^i| + y_1^i - (y_1 + 1) \land y_1^i \ge y_1 + 1 \ge y_2^i \land x_1 = x_2 = x_2^i \land y_2 = y_2^i$
            $\implies d = |x_1^i - x_2^i| + y_1^i - y_1 \land y_1^i \ge y_1 \ge y_2^i \land x_1 = x_2 = x_2^i \land y_2 = y_2^i$
        } $\neg t_4^w \land I_4 \implies d = |x_1^i - x_2^i| + |y_1^i - y_2^i| \land x_1 = x_2 = x_2^i \land y_1 = y_2 = y_2^i$
    }
    return d; $d = |x_1^i - x_2^i| + |y_1^i - y_2^i| \land x_1 = x_2 = x_2^i \land y_1 = y_2 = y_2^i$
}
```

Figure 1: Exemple of an annotated program with Hoare triples.

- Consequence rule: $\dfrac{\boxed{P_1} \implies \boxed{P_2},\quad \boxed{P_2}\ \texttt{S}\ \boxed{Q_2},\quad \boxed{Q_2} \implies \boxed{Q_1}}{\boxed{P_1}\ \texttt{S}\ \boxed{Q_1}}$

  This rule allows to weaken (resp. strenghten) the post-condition (resp. pre-condition).

- Rule of sequential composition: $\dfrac{\boxed{P}\ \texttt{S}\ \boxed{Q},\quad \boxed{Q}\ \texttt{T}\ \boxed{R}}{\boxed{P}\ \texttt{S;T}\ \boxed{R}}$

- Assignment axiom schema: $\dfrac{}{\boxed{P[E/x]}\ \texttt{x:= E}\ \boxed{P}}$

  The logic must be expressive enough to express values of terms.

- If rule: $\dfrac{\boxed{B \wedge P}\ \texttt{S}\ \boxed{Q},\quad \boxed{\neg B \wedge P}\ \texttt{T}\ \boxed{Q}}{\boxed{P}\ \textbf{if}\ \texttt{(B)}\ \texttt{S}\ \textbf{else}\ \texttt{T}\ \boxed{Q}}$

  The logic must be expressive enough to express validity of terms.

- While rule: $\dfrac{\boxed{I \wedge B}\ \texttt{S}\ \boxed{I}}{\boxed{I}\ \textbf{while}\ \texttt{(B)}\ \texttt{S}\ \boxed{\neg B \wedge I}}$

  The main difficulty of this rule is to find the appropriate rule invariant I that holds as pre-condition and such that its conjunction with $\neg B$ expresses an interesting property about the loop execution.

Figure 2: Inferences rules of Hoare logic.

process, i.e. checking the compatibility of each intermediate specification with the instructions of the program.

The intermediate step, which consists in annotating the program is the most challenging, and mechanizing it can follow different approaches, for example propagating upwards the postcondition, by computing at each step the weakest precondition, and hoping that at the begining of the program a consequence of its specified precondition has been propagated. The mechanization of this approach however can hardly deal with loops: first because an invariant has to be inferred, and also because it is not always the case that the most accurate loop invariant can be expressed in the fixed logical framework.

The expressivity and decidability of the logics is crucial for this case: even if the most general loop invariant cannot be automatically computed or even be expressed in the logics, it is crucial that appropriate invariants, implying the correctness of the program, can be expressed (and ideally be effectively and efficiently generated).

An example of a very successful logical framework was developped by Büchi [Büc60]. He showed that the monadic second-order logic over the scattered linear order –which is a very expressive logic to express properties over words– was effectively decidable. The method which he developped in his proof is a major contribution to the topic of automata theory: indeed the decidability result relies on the very intuitive computational model of finite-state automata. By showing the equi-expressiveness of this powerful logical framework with automata, he pioneered the use of automata to analyse so-called regular properties over words.

His work over words lead to many applications as representing objects, values or traces as words is natural. And such representations allows to formally state some properties over words that faithfully reflect properties of these objects. Noticeably, representing proofs as such was a cornerstone in addressing the *Entscheidungsproblem*. In the setting of regular word analysis, we only allow ourselves to express properties over these words under the restriction of the powerful monadic second order logic, which allows to use the theory of automata. Instead of manipulating formulas stating properties about these objects, one can use the equally expressive model of automata to deal with these properties over these objects. The logical equivalence allows one to carry on the automata all the logical operations.

We recall here some interesting primitives over automata which allow to carry efficiently the logical connectives on the automata, and also to perform efficiently some tests which correspond to high complexity problems in the logical framework.

- Complementation: an automaton is a transition system, which enjoys the property that if it is not deterministic one can build an equivalent deterministic automaton whose set of states is the set of subsets of the states of the initial one. Building this automaton through its graph traversal allows to build only the set of accessible states which is not necessarily exponential. Then complementation can be easily performed on deterministic automata by flipping the acceptance of each state. The negation of a formula is easily performed with the appropriate connective, and is also easily handled with deterministic automata.

- Product of automata: this is how one handles the boolean connectives over automata, given two automata, one can build a product automaton (its set of states is the cartesian product of the sets of state of each automata), then depending on the definition of the final states, any boolean combination of the two languages can be obtained.

- Emptiness problem: it consists in checking if some automaton accepts some word, which is the analogous of checking whether a formula, a property is satisfiable. The fact that an automaton is a finite-state transition system, makes this problem as simple as checking the accessibility of some final state: it is no harder than a graph traversal, that is NLOGSPACE. Let us remark that for an MSO formula, this problem is non-elementary.

- Minimization: Any automaton has a deterministic canonical minimal form, which can be build in quasi-linear time from a deterministic automaton. This allows to check very efficiently the equivalence of two automata (with a traversal of the two automata). As equivalence is expressible in the logics, the decidability of equivalence is no surprise but the complexity with automata is much better than the non-elementary complexity in the logical framework.

However, automata are limited in the sense that they only allow to express qualitatives properties over words. If we specify some values, we might also be interested in expressing some relations between these objects, and it is crucial to have both an expressive logical formalism to express relations between words and also an efficient way to manipulate these relations, for example for composing them, or to perform regular tests over these relations.

Automata are the cornerstone of regular model checking [BJNT00], who addresses verification by showing that the set of reachable configurations satisfies some property. The meaningful properties of a program are represented, typically by integers, or words, and it deals with propagating sets of those. Atomic instruction thus define some very simple operations over those sets, and if-branching is also easily addressed. Though these relations that atomic instructions yield are easily computable, their transitive closure is not necessarily, as iterating these relations might not yield a fixpoint. There are different approaches to handle this problem:

- Acceleration techniques [BFLP03] which rely on the possiblity to compute transitive closure of some relations to perform the computation of transition closure with bigger steps, which sometimes yield a fixpoint.

- Overapproximations [BHV04], which consists of considering more reachable configurations to ease the computation of the fixpoint. These overapproximations come at the price that the larger set of configurations must all satisfy the desired properties of the program. Some suitable overapproximations can be determined for example by using a counter-example refinement technique: if the approximation contains some configurations which conflicts with the specification, the approximation is refined by removing them.

In this dissertation, I will study two kinds of relations over words. First, a rather direct extension of automata theory to define relations (i.e. relations such that the synchronized product of the words form a regular language), which is the idea that led to the study of so-called automatic structures, over which the primary result is that their first-order theory is decidable.

Then, as functionality is a very natural concept to describe properties over computation, I will study a class of tranformations over words, defined in MSO logic, which were introduced by Courcelle [Cou94] in the more general framework of graph to graph transformations. In the setting of word input, I give a direct reduction from the logical definition of this class of transformations to the equi-expressive computational model of streaming string transducers. This computational model will yield decision procedures for some non-trivial problems such as functional equivalence or typechecking.

These two kinds of relations differ from the well-studied transducers. Transducers are automata whose transitions are further labeled by words over an output alphabet, and put in relation the input word with the concatenation of those words over each run. Their typical use is to specify some transformations over words, which usually reflect the effect of some atomic operation (for instance a single transition of a Turing machine over some tape). The challenging problem is to compute (provided the input and output alphabets are the same) the transitive closure of such transducers, which in the general case is recursively enumerable. This has an application to generate loops invariants, and due to the non-closure of transducers by transitive closure one must deal with under- and over-approximations, which are fine enough so that they reflect some crucial properties of the loop (e.g. the termination of the loop).

The first kind of relations that we will study –those defined by automatic structures– don't enjoy transitive closure, but are closed under first-order logic. The second one, on the other hand, allows to define a much more expressive class of transformations.

Automatic Structures

Presburger Arithmetic, is the first-order logic over integers with addition, it allows to express integer linear constraints, and is a distinctive choice as a logic to state properties and relations between integers (such as in the example given in figure 1), as it is decidable, more precisely complete for 2EXPTIME with linearly many alternations. It was shown decidable by Presburger [Pre30], relying on a quantifier elimination procedure improved by Cooper [Coo72], thus working on the very syntactic objects that formulas are as intermediate objects.

However, when representing integers as words (e.g. using the base 2 encoding), Büchi [Büc60] remarked that the addition could be described by a (simple) finite-state automaton. Furthermore, it is possible to relate any (first-order) logical connective with a corresponding natural operation on automata; therefore, for any first-order formula we can build inductively an automaton accepting the language of solutions of this formula relying on this correspondance between logical connectives and automata operations. Instead of an ad-hoc algorithm manipulating formulas, one can use automata to decide Presburger Arithmetic, reducing the satisfiability of a formula to checking emptiness of the automaton describing its language of solutions.

Manipulating automata allows to address much more efficiently some problems than they would have been addressed manipulating formulas. For example, they provide a canonical and minimal form for each first-order relation over integers. This canonical form trivialises equivalence. Also implication can be decided by checking language inclusion (with a classical construction of a product automaton, or with the technique of antichains [DDHR06]) instead of model-checking some boolean combination of formulas.

This automata-based decision procedure is not specific to Presburger Arithmetic, in fact in his original work, Büchi [Büc60], not only gave a normal form for monadic second order (MSO) formulas over the linear order to show the equivalence between MSO and finite-state automata but also showed that the first-order logic over integers with addition, equality and "is a power of two and appears in the base 2 decomposition of" also effectively had the same expressive power as MSO. The direction of interpretability of this first-order logic –strictly more expressive than Presburger Arithmetic– in MSO suffices to give an automata-based decision procedure for the first-order theory of this more general structure.

Hodgson [Hod82] sketched the definition of an automatic structure: it is a relational structure whose domain can be encoded as a regular language such that the relations are also regular languages. He announces the decidability of first-order theories over such structures by an inductive construction of an automaton accepting any first-order definable relation, by induction over formulas using the effective closure properties (complementation, intersection and projection) of regular languages.

Khoussainov and Nerode [KN94] coined the name *automatic structure* and initiated a systematic study of which structures could be automatically presentable. This notion was further extended by Blumensath [BG00] to *tree automatic structures*, which allowed to capture more logical structures, at the small cost of using tree automata instead of string automata, both enjoying the nice closure properties required to ensure regularity of any first-order relation.

Recently Kuske [Kus09] gave general complexity results on automatic structures: in the general case, the model checking is non elementary, which is reflected by the automaton construction: each language projection (the operation corresponding to quantification) would lead to an exponential blow-up of the size of the automaton if we use deterministic automata. If we use non-deterministic automata it is negations that each lead to an exponential blow-up of the size of the automaton.

This automata-based technique in the special setting of Presburger Arithmetic was implemented in the tools LASH [z2], PRESTAF [z3] and MONA [1] [z1] but the complexity of this automata-based decision procedure was left unknown, as the general complexity result, (which can be obtained by a rapid analysis) leads to a non-elementary upper-bound. Then Klaedtke [Kla08] showed that for Presburger Arithmetic, the size of the minimal automaton accepting the solution of a formula was at most triply exponential w.r.t. the size of that formula. His proof relied on the complexity results of the quantifier elimination procedure, and from that

1. Unlike LASH and PRESTAF which use automata to represent Presburger sets, MONA implements a decision procedure for the weak second order theory of one successor, and allows to define Presburger formulas as a special construct.

derives a 4ExpTime upper bound on the complexity of these algorithms. The first contribution of this thesis is to show that this automaton construction lies in 3ExpTime, a posteriori giving a theorical explanation to some experimental remark: that these tools were working "too well" for tools with an alleged non-elementary complexity.

The non-elementary completeness result brought little appeal to study the complexity of automatic structures, but we strove to understand the deep theoretical reasons that had this automata construction be elementary for Presburger Arithmetic, and give a general criterion that allows to bound the size and time of construction of these automata. This criterion is generic enough so that we can apply it to some other automatic structures with elementary first-order theory such as automatic structures of bounded degree, nested pushdown graphs and Skolem Arithmetic and give a fine upper bound on this automata construction which is close to the complexity of these logics.

This exhibits a class of structures over which first-order logic is decidable with a generic and efficient decision procedure.

Streaming Transducers

Automatic structures enjoy a decidable first-order theory, at the cost that all the relations have to be synchronously regular. However this is a strong restriction on the kind of relations which can be defined, and incindently many structures cannot be automatically presented, as their relations cannot be possibly described by automata.

Most programming languages have a deterministic operational semantics, meaning that their behaviour is usually accurately determined by a function, so instead of working in a logical framework that expresses relations over words, we will focus on a class of functions over words.

MSO transducers were introduced by Courcelle and give a nice logical framework to define transformations from graphs to graphs. Indeed they enjoy nice closure properties by composition, and their MSO-based definition allows them to deal with MSO properties.

We will study their restriction over word input, as they seem a natural extension of the theory of automata which allow to state qualitative properties over words, to a theory of regular transformations which would allow to use these properties to build new objects. In the case of finite string to finite string transformations, in a similar manner as automata have the same expressive power as MSO, there exists a computational model introduced by Alur et al. [AČ10], namely *streaming string transducers*, that captures exactly the expressive power of these transformations. The closure properties of the MSO transducers (restricted on string input) for instance by composition, or by regular choice (i.e. if the input is in regular language L_1, apply transformation T_1, else apply transformation T_2) admit corresponding algorithms in the computational model.

A key feature of this streaming transducer model is their operational semantics. First and foremost, they are deterministic. They are deterministic automata with a finite set of write-only registers. As deterministic automata, they work in a single left-to-right pass on the input, updating deterministically their configuration: both their control state and the values of the registers, the newer being obtained as concatenations of values of older ones. The main idea behind streaming transducers, (from which derives their name: a single pass by a deterministic finite state machine with finitely many write-only registers), can be generalized to any kind of output: we only need to fix a domain of valuation of registers and a set of allowed operation to combine values of these registers: the image of the input word is the value of some determined register at the end of the run of the transducer. Hence they are not limited to string to graph transformation, but can for example also compute quantitative functions over words if we fix the domain of registers to integers or reals and allow some numeric functions to combine values of registers.

When we think about register machines, we usually think that we can perform some tests on the content of registers, and by definition, no test can be carried out during the run to update values of registers: this is why we can consider that these registers are write-only, their value cannot influence the sequential processing, their values are manipulated blindly. In that sense they are very similar to automata which accept languages definable with memoryless Turing Machines.

The genericity of this model makes it impossible to decide for instance equivalence of two streaming transducers, for example if we choose undecidable functions to update the values of registers. However for some classes of allowed updates of registers, decidability was established. We might refer to [ADD$^+$13] for an overview of results for different sets of register domains and allowed operations, noticeably using numeric domains, which allows to define classes of numeric functions over words which can be qualified as “regular”.

In the case of finite string to finite string transformations, the proof that transformations defined by these write-only register automata can all be expressed in the logical framework of Courcelle’s MSO transducers relies on the ease of expressing runs using MSO. Similarly as for finite-state automata, to show equivalence with the logical definition, the difficult part of the equivalence is that this finite transition system model can express any definable string-to-string MSO transducer.

The first proof of equivalence went through the intermediate model of two-way transducers [EH01], obfuscating the connection between the two models [2]. Our main contribution here is to provide a direct reduction from the logical framework to the computational model of streaming transducers. The generic logic-driven approach for this reduction allows to carry this reduction to more general cases, for instance ω-word input, and tree ouptut.

Handling trees as domain of valuation for registers is very motivating, as it allows to represent any other domain and operations for registers, as combinations of registers during a run correspond to a tree. However a term is produced, which needs then to be interpreted to obtain the values of the transformation. Hence though we show equivalence for string-to-tree transformations, we do not solve it in the general case, as the interpretation of two distinct terms can be the same.

This computational model motivates the use of this logical framework to specify transformations, as from the computational model one deduces some decision pro-

2. Just as if equivalence between MSO and automata was shown going through memoryless Turing Machines!

cedures for some non-trivial properties over this class transformations, whose great expressiveness derives from their MSO based definition. Indeed the decidability of functionnal equivalence between two logically-defined string-to-tree transformations relies on the effective equivalence with streaming string-to-tree transducers over which functional equivalence can be performed. Also regular typechecking, that is given a regular property stated on the input word whether its implies another property on the output word which can be decided in the logical framework can be more efficiently relying on the finite transition system structure of the transducer model.

Contributions and Plan of Thesis

We first present in Chapter 1 the notions of string and trees and the notions of automata that accept languages of those. Then we present first-order and monadic second order logics and exhibit indexed families of formulas which will allow us to carry the complexity proofs.

Chapter 2 is dedicated to show that building inductively from a Presburger formula an automaton accepting solutions of that formula represented using a base 2 least significant digit first notation, can be done in 3EXPTIME, establishing a previously unknown upper bound matching closely the 2EXPTIME with linearly many alternation hardness of Presburger Arithmetic. This result was published in [DGH10].

Then, in Chapter 3, we present a generalization of this result, addressing the previously unexplored problem of giving complexity lower bounds for the inductive construction of automata accepting solutions of formulas for automatic structures. We give a generic approach to derive such bound and apply it to automatic structures with elementary first-order theory and obtain closely matching bounds. Incidently we also show that using the most significant digit first for Presburger Arithmetic yields the same result. These results were partly published in [DGH12].

Finally, in Chapter 4, we study the more expressive functional relations (expressible as MSO string-to-graph transformations) and we give a logic-based reduction to show the equi-expressiveness of the computational model of streaming infinite string to tree transducers with Courcelle's MSO transducers, giving a more direct proof for the cases of finite string to tree, and infinite string to string transducers and establishing equivalence for the previously unexplored case of infinite string to trees transformations. These results were partly published in [ADGT13].

Chapter 1

Preliminaries

In this section, we will briefly recall some classical notations over words, trees, and how to accept languages with finite state automata.

We will then present the logical frameworks that we will use, namely first-order logics and monadic second-order logic.

1.1 Languages and Automata

Languages are sets of words or of trees, and we will essentially deal with so-called regular languages, which can be seen as the subclass of languages which can be accepted by finite automata. What makes this class interesting, is that it can be also equivalently defined logically, or in an algebraic manner.

1.1.1 Words

A finite word (resp. ω-word) over a finite alphabet Σ (i.e. a finite set of letters) is defined as a finite, possibly empty, (resp. infinite) sequence of letters. We denote ε the empty word. The set of finite words over Σ are denoted Σ^*, the set of infinite words over Σ are denoted Σ^ω, and $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$.

In a more logical setting, a finite word w (of length, denoted $|w| = n$ for some $n \in \mathbb{N}$) over alphabet Σ can be seen as an interpreted structure whose domain is $[1, n]$, and with predicates $<$ which denotes the order over integers and $(P_\alpha)_{\alpha\in\Sigma}$ such that for any integer x in the domain, exactly one P_α holds, we say that α is the letter at position x. We also write $w[x]$ to denote the letter at position x in w. In Chapter 4 we will also need to consider subwords that is the sequence of letters between two positions in the word: for any i, j such that $i < j \leq |w|$, we write $w[i{:}j], w(i{:}j), w[i{:}j)$ and $w(i{:}j]$, to denote subwords of w respectively starting at i and ending at j, starting just after i, ending just before j, and so on. For instance, $w[1{:}x)$ denotes the prefix ending just before x (it is ε if $x = 1$), while $w(x{:}|w|]$ denotes the suffix starting just after x.

Automata are finite state machines, that define languages: they process words in a single left-to-right pass and the last state (or set of infinitely occurring states in the case of infinite words) characterizes whether or not the word is in the language. We first define automata over finite words.

Definition 1.1. A finite-state word automaton over alphabet Σ is a 4-tuple (Q, δ, Q_0, F)

- Q is a finite set of states
- δ is the transition relation: $\delta \subseteq Q \times \Sigma \times Q$
- $Q_0 \subseteq Q$ is the set of initial states
- $F \subseteq Q$ is the set of accepting states

We now define the semantics of those automata, with the notion of run, which formalizes the notion of processing a word by an automaton.

A run of an automaton $A = (Q, Q_0, F, \delta)$ over a finite string $w \in \Sigma^*$ is a finite sequence of $|w| + 1$ states $q_0, q_1, \ldots, q_{|w|}$ such that $q_0 \in Q_0$ and for any $0 \leq i < |w|$ we have $(q_i, w[i+1], q_{i+1}) \in \delta$. Word w reaches a state $q \in Q$ in automaton A if there exists a run that ends with q. A run is accepting if it leads to a state $q \in F$. The automaton A defines the language $L(A)$ which contains exactly the words that have an accepting run in A.

If an automaton is such that $|Q_0| = 1$ (we denote q_0 the unique element in Q_0) and for any state q and any letter α, we have$|\{q'|(q, \alpha, q') \in \delta\}| \leq 1$, the automaton is deterministic. Indeed any word then has at most one run in the automaton, (hence it reaches at most one state) and δ can be seen as a partial function from $Q \times \Sigma$ to Q. We can naturally extend this partial function to finite words over Σ to $\widehat{\delta}$ as follows: $\widehat{\delta}(\varepsilon) = q_0$ and $\widehat{\delta}(w\alpha) = \delta(\widehat{\delta}(w), \alpha)$. The image of a word by $\widehat{\delta}$ is the state it reaches in the automaton.

The semantics of automata relies on runs, hence non-accessible states do not influence the behaviour of the automaton, they can therefore be safely removed: trimming an automaton consists in removing thoses states.

It is an important result to note that for any automaton, there exists a deterministic automaton accepting the same language. There is an effective way to build this automaton using the subset construction. This construction may lead to an exponentially bigger deterministic automaton, but if the automaton is built on the fly (that is by a traversal), this construction will produce a trim automaton, in time polynomial w.r.t. this output automaton.

The notions of runs are easily extended to ω-words –yielding infinite runs, that is infinite sequences of states satisfying the same compatibility w.r.t. the transition relation– only the acceptance condition needs another definition.

Büchi automata, have the same syntactical definition, their semantics differ in that an ω-word is accepted by a Büchi automaton iff there exists a run of that word which visits infinitely often a final state. We won't use this definition, as there does not always exist a deterministic Büchi automaton accepting the same language.

We will however rely on another equi-expressive model, which will also enjoy determinism.

Definition 1.2. A Muller automaton over alphabet Σ is a 4-tuple $(Q, Q_0, \mathcal{F}, \delta)$ where: Q, Q_0 and δ are respectifively a finite set of states (Q), a set of initial states ($Q_0 \subseteq Q$) and the transition relation ($\delta \subseteq Q \times \Sigma \times Q$); and $\mathcal{F} \subseteq 2^Q$ is the set of accepting sets of states.

A Muller automaton accepts an infinite word iff there exists a run of that word in this automaton whose set of infinitely occuring states is an accepting set of states. Formally, a word $w \in \Sigma^\omega$ is accepted by $A = (Q, Q_0, \mathcal{F}, \delta)$ if there exists a an infinite run $\rho = q_0, \dots, q_i, \dots$ of w in A, such that $\{q \in Q \mid q \text{ appears infinitely often in } \rho\} \in \mathcal{F}$.

The notion of determinism does only depend on Q, Q_0, δ, the notion of determinism is thus the same for Muller automata, and as for automata over finite words, for any Muller automaton there exists a deterministic Muller automaton accepting the same language, its construction is more involved than for finite-word automata and the complexity of construction is $2^{O(n \log n)}$ w.r.t. the number of states [Saf88].

This determinizability comes at the cost that the size of a Muller automaton is not necessarily polynomial w.r.t. it number of states, indeed, the size of the acceptance condition can be exponential. There are other definitons of acceptance conditions, such as parity, that allow to maintain the size of the automaton polynomial w.r.t. its set of states.

1.1.2 Trees

In this dissertation, we will consider only binary trees, but every technique detailed maps well to any trees with arbitrary fixed finite arity.

Definition 1.3. A *binary tree* over alphabet Σ is a mapping $t : \text{dom}(t) \to \Sigma$, where $\text{dom}(t) \subseteq \{1, 2\}^*$ is prefix closed i.e. for all $w \in \{1, 2\}^*$ if $\{w1, w2\} \cap \text{dom}(t) \neq \emptyset$, then also $w \in \text{dom}(t)$. The tree t is finite iff $\text{dom}(t)$ is finite.

With T^*_Σ (resp. T^ω_Σ), we denote the set of all finite (resp. infinite) binary trees over Σ.

A node of a tree t is an element of $\text{dom}(t)$, ε (the empty word) is also called the root (every non-empty tree has a root), and every node w such that $\{w1, w2\} \cap \text{dom}(t) = \emptyset$ is called a leaf. If the domain is empty the tree is called the empty tree and is denoted η.

Equivalently a tree can also be seen as a term, where we see letters of the alphabet as binary functions, we also need a special constant symbol to reflect that some node does not have the corresponding successor. Also, a (binary) tree is a special case of directed labeled graphs: nodes are labelled by letters of the alphabet, and edges by either 1 or 2, which are acyclic, connected, and such that any node has at most two outgoing edges (necessarily with different labels) and at most one incoming edge.

We will also use the notion of trees over Σ with holes. These are binary trees over the alphabet $\Sigma \cup \{?\}$ (where ? is a special symbol not in Σ) such that any ?-labeled node is a leave (i.e. it has no successor) and we call these nodes holes. If a tree has exactly one hole we call it a context (and we denote C_Σ the set of all context over alphabet Σ).

A tree over Σ with $n \in \mathbb{N}$ holes can be seen as a function from $(T^*_\Sigma)^n$ to T^*_Σ. This function is defined as follows: let t a tree with n holes, denote $l_1, \dots, l_n$ the set of ?-labeled leaves sorted in the lexicographic order, then applying this tree with holes to trees $t_1, \dots, t_n \in T^*_\Sigma$ (denoted $t[t_1, \dots, t_n]$) will produce the tree whose domain is $(\text{dom}(t) \setminus \{l_1, \dots, l_n\}) \cup \bigcup_{i=1}^{n} \{w \mid w = l_i w', w' \in \text{dom}(t_i)\}$. The elements in

$\text{dom}(t)\backslash\{l_1,\ldots,l_n\}$ are labeled by the same label as in t, elements of the form l_iw will be labeled by $t_i(w)$. With more generality this function can be applied to trees with holes and will produce trees with holes.

Also if we consider a letter $a \in \Sigma$, we will write $a[X,Y]$ as a shorthand for the application of the tree with two holes with a root labeled by a and with two ?-labeled children; to the trees (possibly with holes) X, Y. We will also denote ? the tree with exactly one node and whose label is ?. Thus $a[t_1,t_2]$ will denote the tree whose root is a and with children t_1 and t_2.

As for words, there is also a notion of automata over trees, that accept tree languages. The definition that we give here can be seen as a special case of the definition (over terms) given in [CDG+07].

Definition 1.4. Tree automata

- A *bottom-up tree automaton* over Σ is a tuple $A = (Q, \Delta, I, F)$ where Q is a finite set of states, $I \in Q$ is the set of initial states and $F \subseteq Q$ is the set of final states and $\Delta \subseteq (Q \times Q \times \Sigma \times Q)$ is the nonempty transition relation.

- A *successful run* of A on a non-empty tree t is a mapping $\rho : \text{dom}(t) \to Q$ such that for every $w \in \text{dom}(t)$, there exists $q, q' \in I$ such that:

  - if $w0, w1 \in \text{dom}(t)$, we have $(\rho(w0), \rho(w1), t(w), \rho(w)) \in \Delta$,
  - if $w0 \in \text{dom}(t), w1 \notin \text{dom}(t)$ we have $(\rho(w0), q, t(w), \rho(w)) \in \Delta$,
  - if $w1 \in \text{dom}(t), w0 \notin \text{dom}(t)$ we have $(q, \rho(w1), t(w), \rho(w)) \in \Delta$,
  - and if $w0, w1 \notin \text{dom}(t)$ we have $(q, q', t(w), \rho(w)) \in \Delta)$.

- We say that a tree t *reaches* a state q in an automaton A, if t is empty and $q \in I$, or t is not empty and there exists a run ρ of t on the automaton A such that $\rho(\varepsilon) = q$.

- $L(A)$ denotes the language *accepted* by A, that is the set of all finite binary trees t that reach a final state of A. Notice that the empty tree η is in the language iff $I \cap F \neq \emptyset$.

- A set $L \subseteq T_\Sigma$ is called *regular* if there exists a finite tree automaton A with $L = L(A)$.

- Given a tree automaton $A = (Q, \Delta, I, F)$ over Σ, a state $q \in Q$ is said to be *reachable* iff there exists a tree $t_q \in T_\Sigma$ that reaches q.

- A bottom-up tree automaton is called *deterministic* iff $|I| = 1$ and for all $q, q' \in Q$ and for all $\alpha \in \Sigma$, $|\Delta \cap \{q\} \times \{q'\} \times \{\alpha\} \times Q| \leq 1$. More intuitively, this means that the transition relation Δ can be seen as a partial function from $Q \times Q \times \Sigma$ to Q.

As for words, tree automata can be determinized, that is if a tree langage L is accepted by a tree automaton A, then there exists a deterministic tree automaton A' accepting the same language. Furthermore, such an automaton can be explicitely built from any automaton accepting L using the standard subset construction, and very similarly to word automata it is easier to build only the trim subset automaton (often called on-the-fly subset determinisation). Building this (trim) automaton can be exponential w.r.t. the size of the input (non-deterministic) automaton, however if there is an upper bound c on the number of states in the trim subset automaton, it can be built in time polynomial w.r.t. both c and the size of the input automaton.

1.1.3 DFA and NFA as special cases of tree automata

In the case of finite input, trees and tree automata generalize words and word automata. Indeed, words can be seens as a special case of trees whose nodes only have at most one child. And tree automata accepting languages of such trees, can be seen as string automata.

To formalize this equivalence, we will identify the empty word with the empty tree and a word $w \in \Sigma^*$ to the tree t (over Σ) whose domain is $\{0^k | k < |w|\}$ (if $|w| = 1$ the domain will be $\{\epsilon\}$) and such that $t(0^i)$ is the $(|w| - i)$-th letter of w.

We can easily obtain from a (deterministic) bottom-up tree automaton accepting only trees whose domains are in 0^* a (deterministic) finite automaton accepting exactly strings corresponding to trees accepted by the tree automaton: the set of states, the initial and final states are the same, and the transition relation is obtained from transitions (in the tree automaton) whose second component is an initial state (by discarding this component).

Let w be a finite word and t the corresponding tree, let us show that t is accepted by the tree automaton A iff w is accepted by the corresponding string automaton. We start by showing the "only if" direction. Denote ρ an accepting run of t in A. By definition, we have that $(q_0, q_0', t(0^{|w|-1}), \rho(0^{|w|-1})) \in \Delta)$ for some initial states q_0, q_0' of A. Denote for all $1 \leq i \leq |w|$, $q_i = \rho(0^{|w|-i})$. We can easily show that $q_0, q_1, \dots, q_{|w|}$ is an accepting run of w in the word automaton as for any $0 \leq i < w$, $(q_i, q, t(0^{|w|-i}), q_{i+1}) \in \Delta$ for some $q \in Q_0$, hence $(q_i, w[i], q_{i+1})$ is a transition in the string automaton (and $q_{|w|} = \rho(\epsilon)$ is an accepting state). Conversely, from an accepting run of a word w in the word automaton, we can easily show that the run ρ which associate to any 0^i (for $0 \leq i < |w|$) $q_{|w|-i}$ is also an accepting run in A.

To conclude this equivalence, we finally exhibit that given a DFA $A = (Q, q_0, F, \delta)$, one can obtain a corresponding bottom-up deterministic automaton, by duplicating the initial state (if it is reachable by a non-empty word): $(Q \cup \{q_0'\}, \{q_0'\}, \{q \mid q \in F \vee (q = q_0' \wedge q_0 \in F)\}, \{(q, q_0', a, q') \mid \delta(q, a) = q'\} \cup \{(q_0', q_0', a, q) \mid \delta(q_0, a) = q\})$.

1.2 Logics

1.2.1 Structures and First Order Logic

A *signature* is a finite set $\mathcal{S}$ of *predicate* symbols. Each predicate $P \in \mathcal{S}$ has a fixed arity denoted by ar_P. A *relational structure* of signature $\mathcal{S}$ is a couple $\mathcal{A} = (A, (P^{\mathcal{A}})_{P \in \mathcal{S}})$, with A a set called *domain* and $P^{\mathcal{A}} \subseteq A^{ar_P}$, the *interpretation of the predicate symbol* P. When the structure is clear from the context, we will identify a predicate P with its *interpretation* $P^{\mathcal{A}}$. We say that P holds for $(a_1, \dots, a_{ar_P}) \in A^{ar_P}$ (also formulated $P(a_1, \dots, a_{ar_P})$ holds) if $(a_1, \dots, a_{ar_P}) \in P^{\mathcal{A}}$.

To define first-order logical formulas, we fix a countably infinite set V of variables, which evaluate to elements of the domain of structures. *First-order logical formulas over the signature $\mathcal{S}$* –or just (first-order) formulas as the signature will be clear from the context– are constructed from the atomic formulas $x = y$ and $P(x_1, \ldots, x_{ar_P})$, where $P \in \mathcal{S}$ and $x, y, x_1, \ldots, x_{ar_P} \in V$, using the boolean connectives $\wedge$ and $\neg$ and the existential quantification over any variable x from V (denoted $\exists x.$). The connective $\vee$ and the universal quantification ($\forall x$) can be derived from these operators in the usual way.

The *quantifier depth* of a formula φ is the maximal nesting depth of quantifiers in φ. Inductively, the quantifier depth of an atomic formula is 0, negation does not modify the quantifier depth of a formula, the quantifier depth of a conjunction is the maximal quantifier depth of the two subformulas and the quantifier depth of $\exists x \cdot \varphi$ is one plus the quantifier depth of φ. The notion of free variable is defined as usual, so is the notion of subformula. We denote $FO^r_m(\mathcal{S})$ (or just FO^r_m if $\mathcal{S}$ is clear from the context) the set of formulas over the signature $\mathcal{S}$, that have at most r free variables and quantifier depth at most m. Notice that any subformula of a formula $\varphi \in FO^r_m$ is a formula in FO^{r+k}_{m-k} for some $k \in [0, m]$.

For example, $\varphi(x) = \forall y(\neg(\exists z \cdot R_1(x, y, z)) \vee \forall t \cdot \exists a \cdot \neg R_2(x, y, a) \vee R_3(x, y, t))$ has one free variable, 4 quantifiers but only quantifier depth 3.

Given a formula $\varphi(\bar{x}_r)$ over r free variables, we denote by $\mathcal{A} \vDash \varphi(\bar{a}_r)$ (with $\bar{a}_r \in A^r$) that the formula φ is valid (in the usual sense) when we substitute the variables with the corresponding constants. We can associate to any formula its set of solutions (in the structure $\mathcal{A}$ which will always be clear from the context), that is the set of assignments of the free variables, seen as r-tuples of elements of A, that satisfy (in the usual sense) the formula. Thus first-order (r-variables) formulas define (r-ary) *first-order relations* over the domain.

1.2.2 Monadic Second Order logic over the linear order

Monadic Second Order logic (MSO) is a strictly more expressive logic than first-order logic. In addition, we have another set of (monadic second-order) variables W, which will represent sets of elements of the domain. There is the additional atomic formula $x \in X$ (with x a first-order variable and X a second-order variable), and we allow existential quantification over those second-order variables. Thanks to first-order, the usual set predicates and operations (inclusion, intersection, union, complement, etc.) can be expressed. The notion of quantifier depth is defined identically as for first-order formulas, as for that we won't make any distinction between first-order and second-order quantifier.

We will only work with MSO over scattered linear orders, that is structures of the form $(N, <, (P_i)_{i \in I})$ where N is either $\mathbb{N}$ or some $[1, n]$ for some integer n, the $(P_i)_{i \in I}$ are a finite set of monadic predicates, and $<$ is a binary predicate whose interpretation is its usual interpretation over the set of integers N.

A word over alphabet Σ will define a structure $(N, <, (P_\alpha)_{\alpha \in \Sigma})$, where $N = [1, |w|]$ (or $N = \mathbb{N}$ if w is an infinite word), and $P_\alpha(k)$ will hold iff $w[k] = \alpha$, and we will confound such structures and words. A sentence (i.e. formulas without any free variables) defines a language, as the set of structures in which the sentence holds.

Equivalent definitions of regular languages

These MSO defined languages are exactly those which are accepted by a finite-state automaton. This class of languages is called the set of regular languages as it is the smallest set of languages which contains $\{\varepsilon\}$, and $\{\alpha\}$ (for all $\alpha \in \Sigma$) and which is closed under concatenation, union, and Kleene star. These languages are also exactly those defined using regular expression. Another more algebraic characterization is that they can be separated by a finite monoid: there exists a finite monoid M and a monoid morphism f from the monoid $(\Sigma^*, \cdot)$ to M, such that the language is exactly the preimage by f of some subset of the monoid M.

1.2.3 Finiteness of formulas

We now detail a crucial remark on formulas, which is clear when those are presented in the predicate calculus but is hindered with the inductive definition of formulas. We fix a finite signature $\mathcal{S}$, the distinct number of first-order formulas over that signature is unbounded. By distinct, we mean that two formulas are distinct, if there exists an interpretation of the structure $\mathcal{S}$ such that the two formulas have different solutions.

Now, if we fix the quantifier depth and the number of free-variables then there are only finitely many distinct formulas. To make this fact clear we will present such sets of formulas as a finitely generated boolean algebra. Informally, we are dealing with boolean algebras so to state that φ, $\neg\neg\varphi$ and $\varphi \wedge \varphi$ are not distinct, as these formulas hold for the same interpretations.

Proof. For any finite signature, by induction over k, we show that for any r the number of distinct formulas with quantifier depth k and r free variables is bounded. The initialization consists of showing that there are finitely many distinct quantifier free formulas, these are the boolean algebra of formulas generated from atomic formulas which are bounded as the number of free-variables is fixed.

For the inductive step, we have to show that for any k, r, the number of distinct formulas with quantifier depth k and r free variables is finite, to show that we can exploit the induction hypothesis that for any $k' < k$ and any r', the number of distinct formulas with quantifier depth k' and r' free variables is finite. Specifically we fix $k' = k - 1$ and $r' = r + 1$. Indeed formulas of quantifier depth at most k and at most r free variables are boolean combinations of formulas of the form $\exists x.\varphi$ for some (first or second-order) variable x and some formula φ with quantifier depth at most $k - 1$ and $r + 1$ free variables. These formulas are build in a finite manner from a finite set of formulas, hence the set of formulas of quantifier depth k and r free variables is also a finitely generated boolean algebra, which concludes the induction. □

We will exploit this result in a different manner for first order to define relations on tuples of elements of the domain and for MSO to define the notion of types.

MSO k-types

This finiteness leads to the finiteness of MSO sentences of quantifier depth at most k (the special case where $r = 0$), which motivates the following definition:

Definition 1.5. For a string $s \in \Sigma^\infty$, we define its k-type (denoted $[s]_{\cong_k}$) as the set of MSO sentences with quantifier depth at most k which hold for this string.

We write Θ_k the set of k-types and $s \cong_k s'$ when two strings s, s' have the same k-type.

For each k, the set of k-types is finite, moreover, every k-type can be represented by an MSO sentence with quantifier depth exactly k. For instance by the conjunction of all formulas with quantifier depth at most k that hold for elements of this k-type together with the conjunction of all negations of all formulas that do not hold for elements of this k-type. A more elegant way of providing such formulas was given by Hintikka [Hin53], these formulas will be referred to as Hintikka formulas.

The next fundamental result is due to Shelah's extension [She75] of Feferman-Vaught composition theorem [FV59] to the context of monadic second-order logic. It follows from the fact that k-types of any two strings uniquely determine the k-type of their concatenation and there is an effective procedure to compute the resulting k-type.

Proposition 1.6 (Composition Theorem [She75])**.**
Let $\cong_k$ be the k-type equivalence relation for MSO formulas over strings.

1. *$\cong_k$ is a monoid congruence, i.e. for strings $u, u' \in \Sigma^*$ s.t. $u \cong_k u'$ and strings $v, v' \in \Sigma^\infty$ s.t. $v \cong_k v'$, we have that $uv \cong_k u'v'$.*
2. *For strings $u, u' \in \Sigma^+$ s.t. $u \cong_k u'$, we have $u^\omega \cong_k u'^\omega$.*

Chapter 2

Automata-based decision procedure for Presburger Arithmetic

Presburger [Pre30] showed that the first-order theory of natural numbers with addition admitted a decision procedure by giving a quantifier elimination procedure. His procedure, improved by Cooper [Coo72] to a 3EXPTIME upper bound, is however quite complex.

Integers are easily represented as word using a base 2 encoding, and Presburger sets (solutions of Presburger formulas) represented as such, enjoy the property forming regular languages. Furthermore there exists a simple algorithm allowing to build inductively an automaton that accepts exactly the language of solutions of formula.

We establish in section 2.3 the construction of size known to be triply exponential [Kla08] admits a 3EXPTIME upper-bound.

2.1 The additive theory of natural numbers

Presburger Arithmetic is the first-order theory over integers with addition, that is the structure $(\mathbb{N}, +, >, 0, 1)$. It was shown decidable by Presburger in 1929 [Pre30], using a quantifier-elimination procedure over an extension of the structure, obtained by adding as atomic formulas all formulas of the form $\sum_{i=1}^{n} a_i x_i \sim c$, where a_i and c are integer constants, x_i integer variables, and $\sim$ is an arithmetic operator among $\{=, \neq, <, >, \leq, \geq, \equiv_m\}$ (with $\equiv_m$ the congruence modulo m, for any $m \geq 2$).

Its decidability, together with the fact it allows to express addition, makes it a good choice when one wants to express relations between integers, for example to express some constraints over the values of some integer variables in the verification of a program, for instance program distance in Figure 1 in which all intermediate specifications are expressed as Presburger formulas. The complexity of Presburger Arithmetic was shown to be exactly doubly exponential in time with linearly many alternations [Ber77]. The lower bound derives from the reduction given by Fischer and Rabin [FR74], who build a formula whose validity depends on the halting of a 2EXPSPACE Turing Machine. However the quantifier elimination procedure given by Presburger had non-elementary worst-case complexity, and Cooper [Coo72] tuned the quantifier elimination to obtain a triply exponential deterministic time upper bound which was exhibited by Oppen [Opp78]. Ferrante and Rackoff [FR79] showed the existence of a non deterministic algorithm to solve Presburger Arithmetic, with the optimal complexity.

In Cooper's decision procedure, given a Presburger formula, one manipulates formulas in order to obtain an equivalent quantifier free formula. However, formulas are descriptive objects, and not computational ones. In the 60's Büchi [Büc60] showed that Presburger Arithmetic could be MSO-interpreted, meaning that deciding the validity of Presburger formula could be reduced to the decision of an MSO formula, hence the emptiness of an automaton. In the mid 90's, Wolper, Boigelot [WB95], Boudet and Comon [BC96] gave an explicit construction of automata representing an atomic formula, in that its language is the set of solutions of the formula represented in a base 2 encoding. Then noticing that logical connectives mapped very well to operations on automata (negation to language complementation, conjunction to language intersection and existential quantification to language projection), they gave an inductive construction of an automaton accepting the set of solutions of any Presburger formula and hence exhibited a purely automata-based algorithm to decide Presburger Arithmetic (satisfiability is reduced to checking language emptiness).

This automata-based decision procedure has been implemented in the tools Lash [z2] and PresTAF [z3], but the complexity of this automata-based algorithm was unknown, or at best a non-elementary upper-bound could be derived. However these tools gave experimental evidence that the complexity was not so stiff, inviting for a much refined proof of complexity.

Recently Klaedtke [Kla08] showed that the size of the automaton for a Presburger formula was at most triply exponential w.r.t. the size of the formula and that this automaton could be built in triple exponential time. Furthermore he showed these bounds tight exhibiting formulas with large automata. However his construction relies on the quantifier elimination procedure, hence the complexity for the generic inductive construction could only be bounded by quadruply exponential deterministic time.

In this section, first we will present the construction of automata accepting solution of atomic constraints. Then we will inspect the size of those automata in order to finally establish an upper bound on the time of inductive construction of an automaton for a Presburger formula. We will detail this inductive construction, and by a careful inspection show that it is achieved in triply exponential time, giving an optimal complexity upper bound, which closes the complexity of this inductive automata-based decision procedure for Presburger Arithmetic.

2.2 The Automatic representation of Presburger Arithmetic

We consider Presburger atomic formulas to be of the form $\sum_{i=1}^{n} a_i x_i \sim c$ where $\sim \in \{=, >, \equiv_m\}$ and $\gcd(a_1, \ldots, a_n) = 1$ (as an equivalent formula satisfying this criterion can be obtained by dividing all a_i by their greatest common dividor), and also consider the two trivial formulas $\top$ and $\bot$ (respectively the tautology and the unsatisfiable proposition) as atomic formulas and as boolean operators we use $\neg$, $\wedge$ and $\vee$, these operators sufficing to express all others, and with a simple linear preprocessing, any formula can be equivalently expressed with only those operators.

A Presburger formula is defined as an object of this first-order theory. The length of a Presburger formula is defined in the usual way (see [Kla08]). Remark that size of the formulas only depends on the log of the values of constants which reflects that these are written using a binary notation. Also if we would impose writting such formulas in the relational structure (i.e. $+$ as a ternary predicate, and $>$ as a binary) the use of additional quantifiers would allow to write an equivalent formula with a linear size increase.

We use the vectorial notation for the atomic formulas, i.e. $\mathbf{a}.\mathbf{x} \sim c$. Let φ be a Presburger formula with r free variables. $[\![\varphi]\!]$ is defined as the set of solutions of φ, that is the set of assignments (seen as a subset of $\mathbb{Z}^r$) of the free variables of φ validating φ (with the usual semantics of arithmetic). The set of solutions (a set of integer vectors) of any Presburger formula is called a Presburger set. Two Presburger formulas φ and φ' are called semantically equivalent iff $[\![\varphi]\!] = [\![\varphi']\!]$. Two n-tuples of Presburger formulas $(\varphi_1, \dots, \varphi_n)$ and $(\varphi'_1, \dots, \varphi'_n)$ are called semantically equivalent iff $[\![\varphi_i]\!] = [\![\varphi'_i]\!]$ for all $i \in \{1, \dots, n\}$. For $1 \leq i \leq n$, we let $\mathbf{e}_i^n$ be the n-dimensional vector which has a 1 as its i-th component and 0 elsewhere.

We represent integer vectors as finite words, so we can map Presburger sets to languages. We use a vectorial least significant bit first coding. For $r > 0$ we define $\Sigma_r = \{0, 1\}^r$. Note that one typically uses 0 and 1 as (positive and negative) signs, but we will use the separate sign alphabet $S_r = \{+, -\}^r$ (indicating if the corresponding integer is positive or negative), which will simplify the constructions, as these letters will carry the information that they are the last one. However, the results that we give will still hold even if we don't take a separate sign alphabet. Words of $\Sigma_r^* S_r$ represent r-dimensional integer vectors. A word $w_0 \dots w_n s \in \Sigma_r^* S_r$ represents the integer vector we denote $\langle w_0 \dots w_n s\rangle$, each component of which is computed as follows (where $\pi_i(x)$ representing the value of the i-th component of x). If $s_i = +$, then $\pi_i(\langle w_0 \dots w_n s\rangle) = \sum_{j=0}^{n} 2^j.\pi_i(w_j)$ and if $s_i = -$, then $\pi_i(\langle w_0 \dots w_n s\rangle) = -2^{n+1} + \sum_{j=0}^{n} 2^j.\pi_i(w_j)$. In particular, $\langle +\rangle = 0$ and $\langle -\rangle = -1$. We also define the notation $\langle . \rangle_+$ over Σ_r^* as $\langle w\rangle_+ = \langle w+^r\rangle$.

Remark 2.1. Let $w', w \in \Sigma_r^*, s \in S_r$. We have $\langle w'ws\rangle = \langle w'\rangle_+ + 2^{|w'|}\langle ws\rangle$.

This representation is clearly surjective and provides an infinite number of representations for each vector. Indeed for any word $w_0 \dots w_n s \in \Sigma_r^* S_r$, any $w_0 \dots w_n (s')^* s$ (with $s'_i = 0$ if $s_i = +$ or $s'_i = 1$ if $s_i = -$) represents the same vector.

Given a Presburger formula $\varphi(\mathbf{x})$ (with $\mathbf{x}$ the vector of free variables of φ, and r its dimension), we say it defines the language $L_\varphi = \{w \in \Sigma_r^* S_r \mid \langle w\rangle \in [\![\varphi]\!]\}$. Such languages are called Presburger-definable and meet the following saturation property: if a representation of a vector is in the language then any other representation of that vector is also in the language.

In order to present explicitly the construction of automata for Presburger atomic constraints, we need to recall a concept of automata theory.

Definition 2.2. Residual languages

- Given a language L, and a word w, we call the residual of L (by w) denoted $w^{-1}L$ the set of words u such that $wu \in L$.
- Given an automaton $A = (Q, Q_0, F, \delta)$, and a state q of A, we call the residual of state q denoted $L(A_q)$, the set of words accepted by the automaton $(Q, \{q\}, F, \delta)$, that is the set of words having an accepting run from q in A.

The set of residuals of language L is finite if and only if the language L is regular. If the automaton is deterministic, the set of residuals of A is exactly the set of residual of the language $L(A)$.

Least significant digit first coding enjoys the following important property [Ler08].

Proposition 2.3. *Any residual of a saturated Presburger-definable language is either a saturated Presburger-definable language, or the empty word language.*

We not only have a residual closure property on (languages representing) Presburger-definable sets, but we also have an effective way to characterise them.

Lemma 2.4. *In an automaton accepting all solutions of a Presburger formula $\varphi(\mathbf{x})$ with r free variables, a word $w \in \Sigma_r^*$ leads from the initial state to a state accepting exactly all solutions of $\varphi(2^{|w|}\mathbf{x} + \langle w \rangle_+)$.*

Proof. This lemma relies on the following property of the least significant digit first coding used. Let $w \in \Sigma_r^*$ and $u \in \Sigma_r^* S_r$, $\langle wu \rangle = 2^{|w|}\langle u \rangle + \langle w \rangle_+$. Hence for any word $w \in \Sigma_r^*$, if a word u encodes a solution of $\psi_w(\mathbf{x}) = \varphi(2^{|w|}\mathbf{x} + \langle w \rangle_+)$, wu encodes a solution of φ, hence the state reached by w accepts all solutions of ψ_w. Conversely, if u does not encode a solution of ψ_w, either u is not a valid coding, in that case wu is not (and hence is not accepted by the automaton for φ), or u is a valid coding and not a solution of ψ_w, in which case uw does not encode a solution of φ, and is therefore not accepted by the automaton for φ. □

This allows us to detail explicitely the construction of automata accepting solutions of atomic Presburger formulas.

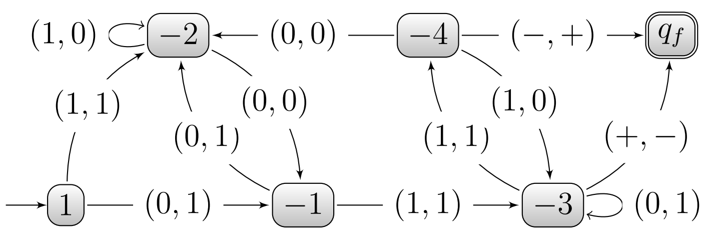
Automata representing atomic Presburger formula

We study here automata that accept Presburger-definable languages. Notice that all final states of such automata are equivalent (there is only one residual that contains the empty word). All other states have a residual that is Presburger definable, i.e. definable by a Presburger formula. Thus, in the following, automata have as states a unique final state F and elements of $\mathcal{F}$, the set of Presburger formulas. Each state $\varphi \in \mathcal{F}$ characterises its residual, for example a state with empty residual will be represented by $\perp$.

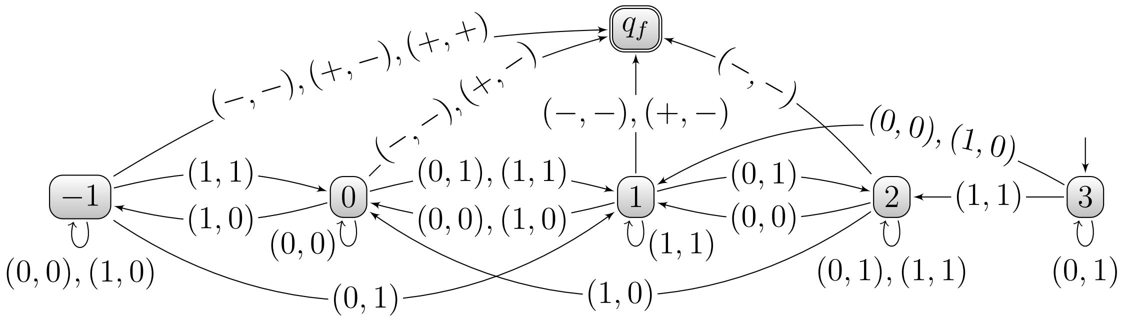
We recall here the construction of the DFA for atomic Presburger constraints. The construction for equations and inequations with a least significant bit first coding was given by Boudet and Comon [BC96]; as they worked with natural numbers, they had not to handle the sign letters. Wolper and Boigelot [WB95] worked with integers but used a most significant bit first coding. They gave a backwards construction of the automaton yielding an NFA for inequations which they then had to determinise. Since we work with a least significant bit first coding we are spared this determinisation procedure. We also give the construction of the automaton for modular constraints directly, rather than using Klaedtke's [Kla08] which is also based on a most significant bit first coding. In figure 2.1 we give some automata built for some atomic constraints.

- The automaton $A_{\mathbf{a}.\mathbf{x}=c}$ for the formula $\mathbf{a}.\mathbf{x} = c$ is given by the following forward construction starting from the initial state $\mathbf{a}.\mathbf{x} = c$:
  - if $b \in \Sigma_r$, $\delta(\mathbf{a}.\mathbf{x} = \gamma, b) = \begin{cases} \mathbf{a}.\mathbf{x} = \gamma' \text{ with } \gamma' = \frac{\gamma - \mathbf{a}.\mathbf{b}}{2} \text{ when } 2 \mid \gamma - \mathbf{a}.\mathbf{b} \\ \perp \text{ otherwise} \end{cases}$
  - if $s \in S_r$, $\delta(\mathbf{a}.\mathbf{x} = \gamma, s) = F$ when $\mathbf{a}.\langle s \rangle = \gamma$ and $\delta(\mathbf{a}.\mathbf{x} = \gamma, s) = \perp$ otherwise
  - for $\alpha \in S_r \cup \Sigma_r, \delta(\perp, \alpha) = \delta(F, \alpha) = \perp$
- The automaton $A_{\mathbf{a}.\mathbf{x}>c}$ for the formula $\mathbf{a}.\mathbf{x} > c$ is given by the following forward construction starting from the initial state $\mathbf{a}.\mathbf{x} > c$:
  - if $b \in \Sigma_r$, $\delta(\mathbf{a}.\mathbf{x} > \gamma, b) = \mathbf{a}.\mathbf{x} > \gamma'$ with $\gamma' = \left\lfloor \frac{\gamma - \mathbf{a}.\mathbf{b}}{2} \right\rfloor$.
  - if $s \in S_r$, $\delta(\mathbf{a}.\mathbf{x} > \gamma, s) = F$ when $\mathbf{a}.\langle s \rangle > \gamma$ and $\delta(\mathbf{a}.\mathbf{x} > \gamma, s) = \perp$ otherwise
  - for $\alpha \in S_r \cup \Sigma_r, \delta(\perp, \alpha) = \delta(F, \alpha) = \perp$
- The automaton $A_{\mathbf{a}.\mathbf{x} \equiv_{2^m(2n+1)} c}$ for the formula $\mathbf{a}.\mathbf{x} \equiv_{2^m(2n+1)} c$ (with some $m, n \geq 0$) is given by the following forward construction from $\mathbf{a}.\mathbf{x} \equiv_{2^m(2n+1)} c$:
  - if $b \in \Sigma_r$ and $p \geq 1$,
    $\delta(\mathbf{a}.\mathbf{x} \equiv_{2^p(2n+1)} \gamma, b) = \begin{cases} \mathbf{a}.\mathbf{x} \equiv_{2^{p-1}(2n+1)} \gamma' \text{ with } \gamma' = \frac{\gamma - \mathbf{a}.\mathbf{b}}{2} \text{ when } 2 \mid \gamma - \mathbf{a}.\mathbf{b} \\ \perp \text{ otherwise} \end{cases}$
  - if $b \in \Sigma_r$ and $p = 0$,
    $\delta(\mathbf{a}.\mathbf{x} \equiv_{2n+1} \gamma, b) = \mathbf{a}.\mathbf{x} \equiv_{2n+1} \gamma'$ with $\gamma' = \begin{cases} \frac{\gamma - \mathbf{a}.\mathbf{b}}{2} \text{ when } 2 \mid \gamma - \mathbf{a}.\mathbf{b} \\ \frac{\gamma + 2n + 1 - \mathbf{a}.\mathbf{b}}{2} \text{ when } 2 \nmid \gamma - \mathbf{a}.\mathbf{b} \end{cases}$
  - if $s \in S_r$, $\delta(\mathbf{a}.\mathbf{x} \equiv_{2^p(2n+1)} \gamma, s) = \begin{cases} F \text{ when } \mathbf{a}.\langle s \rangle \equiv_{2^p(2n+1)} \gamma \\ \perp \text{ otherwise} \end{cases}$
  - for $\alpha \in S_r \cup \Sigma_r, \delta(\perp, \alpha) = \delta(F, \alpha) = \perp$

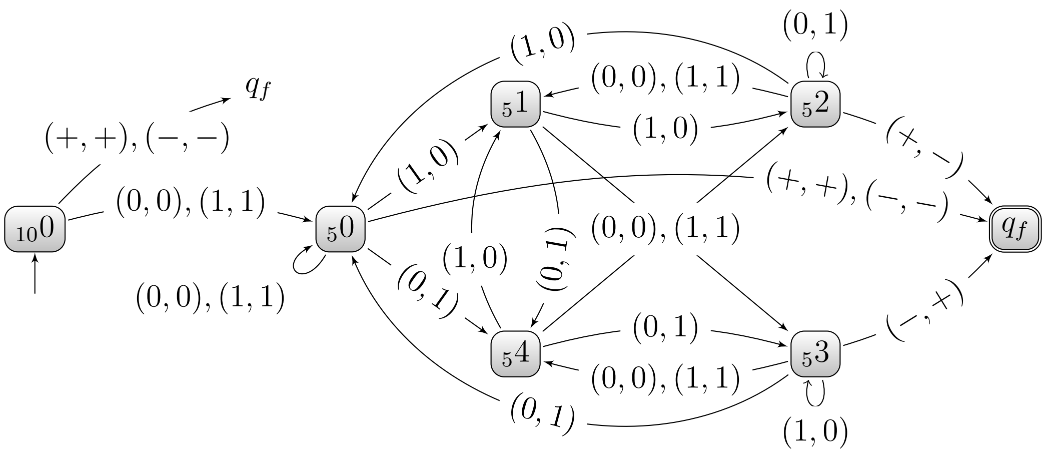
The correctness of this construction derives from Proposition 2.3. For all states φ and all $b \in \Sigma_r$ we have $\delta(\varphi(\mathbf{x})) = \varphi(2\mathbf{x} + b)$ showing that the transitions labeled by Σ_r are correct. The transitions labelled by S_r are correct by definition, as the last letter is always from S_r. The definitions are clearly exhaustive. Furthermore, as $gcd(\mathbf{a}) = 1$, all states are non-equivalent, indeed $[\![\mathbf{a}.\mathbf{x} \sim \gamma]\!] \neq [\![\mathbf{a}.\mathbf{x} \sim \gamma']\!]$ when $\gamma \neq \gamma'$. Thus our construction provides us the *minimal DFA*. In the following we detail the number of states of the automata constructed. Those results were given by Klaedtke [Kla08] on automata for most significant bit first coding. For a vector $\mathbf{a}$ we define $\|\mathbf{a}\|_+ = \sum_{\{i \mid a_i \geq 0\}} a_i$ and $\|\mathbf{a}\|_- = \sum_{\{i \mid a_i \leq 0\}} |a_i|$.



(a) Automaton for $2x + 3y = 1$



(b) Automaton for $x - 3y > 3$



(c) Automaton for $7x + 3y \equiv_{10} 0$

Figure 2.1: Automata for various atomic constraints

Theorem 2.5. *Let $gcd(\mathbf{a}) = 1$.*

1. *For the case of an equation $\mathbf{a}.\mathbf{x} = c$, the states $\{\mathbf{a}.\mathbf{x} = \gamma \mid -\|\mathbf{a}\|_+ < \gamma < \|\mathbf{a}\|_-\}$ are reachable and form a maximal strongly connected component (SCC) and all other reachable states are in $\{\mathbf{a}.\mathbf{x} = \gamma \mid \gamma = c \vee \min(c, -\|\mathbf{a}\|_+) < \gamma < \max(c, \|\mathbf{a}\|_-)\}$.*

2. *In the case of an inequation $\mathbf{a}.\mathbf{x} > c$, the states $\{\mathbf{a}.\mathbf{x} > \gamma \mid -\|\mathbf{a}\|_+ \leq \gamma < \|\mathbf{a}\|_-\}$ are reachable and form a maximal SCC and all other reachable states are in $\{\mathbf{a}.\mathbf{x} > \gamma \mid \gamma = c \vee \min(c, -\|\mathbf{a}\|_+) \leq \gamma < \max(c, \|\mathbf{a}\|_-)\}$.*

3. *For modulo constraints $\mathbf{a}.\mathbf{x} \equiv_{2^n(2p+1)} c$ the states in $\{\mathbf{a}.\mathbf{x} \equiv_{2p+1} \gamma \mid \gamma \in [0, 2p]\}$ are reachable and form a maximal SCC and all other reachable states are in $\{\mathbf{a}.\mathbf{x} \equiv_{2^m(2p+1)} \gamma \mid (m = n \wedge \gamma = c) \vee (m < n \wedge \gamma \in [0, 2^m(2p+1) - 1])\}$.*

The SCCs only depend on $\mathbf{a}$ and not on the constant c. Before proving the theorem we give in Figure 2.1 the automaton for a simple inequation. The reachable maximal SCC is formed by the states $\{-1, 0, 1, 2\}$. The transitory state 3 exists only because $3 > \|\mathbf{a}\|_-$.

Proof. Klaedtke [Kla08] studied the number of reachable states for equations and inequations. Automata for equations are by construction backwards deterministic, thus we built the reverse automaton of Klaedtke's and all states in $\{\mathbf{a}.\mathbf{x} = \gamma \mid -\|\mathbf{a}\|_+ < \gamma < \|\mathbf{a}\|_-\}$ are reachable and form a maximal SCC. We get the same reachable states forming a maximal SCC for an automaton for an inequation $\mathbf{a}.\mathbf{x} > \gamma$, as it just contains more transitions than the automaton for $\mathbf{a}.\mathbf{x} = \gamma$. For modulo constraints, for our coding, if the modulus is even, it is halved by the next transition, so all states are reached only for odd modulus. Thus, the automaton is possibly smaller compared to the automaton constructed for the most significant bit first coding which is as big as the modulus. $\square$

2.3 Inspecting the size of automata for Presburger Arithmetic

In this section, we will establish the size of the automata for atomic constraints, and show that for these formulas, the minimal automaton which is obtained by our construction is also the minimal non-deterministic automaton.

Then, we will inspect the size of boolean combinations of atomic formulas, in order to achieve, in the next section, the claimed time upper bound for the inductive construction of the automaton accepting solutions of a Presburger formula.

2.3.1 Automata for atomic constraints

We exhibit here that the canonical automata built for atomic constraint do no admitt a smaller equivalent non-deterministic automaton.

Theorem 2.6. *For equations, inequations or modulo constraints with an odd modulus, no non-determinist automaton is smaller than the minimal DFA.*

The proof of this theorem consists essentially in considering a class of languages for which the minimal DFA is already a minimal NFA.

We will use the following result from [LLRT06] for a class of regular languages, namely biRFSA languages.

Definition 2.7. An automata A is an RFSA (Residual Finite State Automaton) if any of its residuals is a residual of the language $L(A)$.

Notice that deterministic automata are obviously RFSA, but this does not necessarily hold for non-deterministic automata. We are now ready to present the notion of biRFSA languages.

Definition 2.8. biRFSA automata accept biRFSA languages:

- An automaton A is a biRFSA if it is an RFSA, and its reverse, denoted $\widetilde{A}$ (obtained by reversing its transitions and swap its initial states for its final), is still an RFSA.
- A language is biRFSA iff it is accepted by a biRFSA.

The biRFSA languages are thus languages that can be accepted by a very special kind of RFSA: its reverse must still be an RFSA. RFSA also enjoy a canonical minimal form:

Proposition 2.9. *Given an RFSA $A = (Q, Q_0, \delta, F)$, we denote $(L_q)_{q \in Q}$ its set of residual languages: it is canonical if*

- *Any L_q is prime: it cannot be obtained exactly as an union of some other $L_{q'}$.*
- *For any $a \in \Sigma$, $q, q' \in Q$, $aL_q \subseteq L_{q'}$ iff $(q', a, q) \in \delta$.*
- *$q \in F$ iff $L_q \subseteq L(A)$.*

We are now ready to use the central result of biRFSA [LLRT06]:

Proposition 2.10. *The canonical RFSA of a biRFSA language is a minimal NFA.*

We will now show that the languages of solutions for constraints described in Theorem 2.6 are biRFSA, and that their canonical RFSA is not smaller than the minimal DFA. Therefore it is a minimal NFA as well. For that we characterize a larger class of biRFSA languages.

Lemma 2.11. *Given a language L with residuals $L_0 = L, L_1, \ldots, L_n$, such that:*

1. *Given a residual, residuals languages included in it form a strictly increasing chain w.r.t. inclusion:*

$$\forall i, j, k \in [1, n] \cdot (L_j \subseteq L_i) \wedge (L_k \subseteq L_i) \implies (L_j \subseteq L_k) \vee (L_k \subseteq L_j)$$

2. *Two residuals are either disjoint or one is included in the other:*

$$\forall i, j \in [1, n] \cdot (L_i \cap L_j) \neq \emptyset \implies (L_i \subseteq L_j) \vee (L_j \subseteq L_i)$$

Then any NFA accepting this language is at least as big (in term of number of states) as the minimal DFA.

Proof. Let L be a regular language satisfying the lemma's hypotheses. The minimal DFA has as many states as the canonical RFSA as all the residuals are prime. This is because for $L_0, \dots, L_m$ residuals of L, if $L_0 = \bigcup_{i \geq 1} L_i$ then $\forall i. L_i \in L_0$ thus the L_i form an increasing chain, thus $\exists i \geq 1. L_0 = L_i$.

Let $\mathcal{A} = \langle \Sigma, (q_i)_{i \leq m}, Q_0, F, \delta \rangle$ be the canonical RFSA of L. We show that L is biRFSA by observing that the reversed automaton of the canonical RFSA is a RFSA. With the two hypotheses we have on the residuals, it is easy to find for each residual L_i (suppose of the state q_i) a word w_i that is only in L_i and the residuals containing L_i.

Let us show that $L(A_{q_i}) = \widetilde{w_i}^{-1}\widetilde{L}$. By definition $\widetilde{w_i}^{-1}\widetilde{L} = \{v \mid \widetilde{w_i}.v \in \widetilde{L}\}$, then $v \in L(\widetilde{\mathcal{A}}_{q_i})$ implies $\widetilde{w}$ can reach q_i in $\mathcal{A}$, which implies that $\widetilde{v} \cdot w_i \in L$ so $\widetilde{w_i} \cdot v \in \widetilde{L}$. We have the first inclusion, $L(\widetilde{\mathcal{A}}_{q_i}) \subseteq \widetilde{w_i}^{-1}\widetilde{L}$ and now show the other inclusion, $\widetilde{w_i} \cdot v \in \widetilde{L} \implies v \in L(\widetilde{\mathcal{A}}_{q_i})$. If $\widetilde{w_i}.v \in \widetilde{L}$, then by definition, there is an accepting path labelled by $\widetilde{w_i}.v$ in $\widetilde{\mathcal{A}}$. We can extract from this path a path labelled by $\widetilde{w_i}$, which will reach a state, namely q. By definition of w_i, $L(\mathcal{A}_q) \supseteq L(\mathcal{A}_{q_i})$, from which we deduce that words reaching q in $\mathcal{A}$ also reach q_i, therefore $L(\widetilde{\mathcal{A}}_q) \subseteq L(\widetilde{\mathcal{A}}_{q_i})$. We have shown that the $L(\widetilde{\mathcal{A}}_{q_i})$ are residuals of $\widetilde{L}$. Thus $\widetilde{\mathcal{A}}$ is an RFSA, so $\mathcal{A}$ is a biRFSA, and L a biRFSA language. There is no smaller NFA than the canonical RFSA which has as many states as the minimal DFA. □

Now, Theorem 2.6 follows by observing that the residuals of the corresponding languages verify the conditions of Lemma 2.11.

2.3.2 Automata for quantifier-free formula

We first consider general boolean combinations of atomic constraints. A boolean combination of formulas $\varphi_1, \dots, \varphi_n$ is a formula generated by $\top, \bot, \varphi_1, \dots, \varphi_n, \neg, \vee$ or $\wedge$. We denote by $\mathcal{C}(\varphi_1, \dots, \varphi_n)$ such a boolean combination. We define the notion of *simple* boolean combination as follows. The underlying propositional formula corresponding to $\mathcal{C}(\varphi_1, \dots, \varphi_n)$ is $\mathcal{C}(b_1, \dots, b_n)$ where $b_1, \dots, b_n$ are propositional variables. We say that $\mathcal{C}(b_1, \dots, b_n)$ is simple, if the truth value of the formula depends on all propositional variables. Then, a boolean combination $\mathcal{C}(\varphi_1, \dots, \varphi_n)$ is simple, if its underlying propositional formula $\mathcal{C}(b_1, \dots, b_n)$ is simple. From any boolean combination $\mathcal{C}(\varphi_1, \dots, \varphi_n)$ we can always obtain a simple one by removing some atomic formulas if needed.

To build an automaton for a boolean combination of atomic formulas, we build a product automaton whose states are Presburger formulas (not tuples of formulas).

Definition 2.12. Given a boolean combination of atomic formulas $\mathcal{C}(\varphi_1(\mathbf{x}), \dots, \varphi_n(\mathbf{x}))$, the product automaton $A_{\mathcal{C}(\varphi_1(\mathbf{x}), \dots, \varphi_n(\mathbf{x}))}$ is given by:

- Q is the set of Presburger formulas and the designated final state q_f,
- $q_0 = \mathcal{C}(\varphi_1(\mathbf{x}), \dots, \varphi_n(\mathbf{x}))$
- δ, the transition relation is defined:

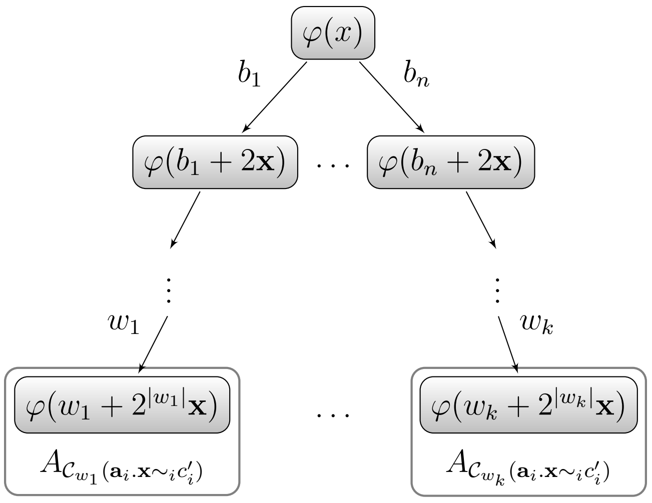

$\varphi(x)$
b_1
b_n
$\varphi(b_1 + 2\mathbf{x})$
$\cdots$
$\varphi(b_n + 2\mathbf{x})$
w_1
w_k
$\varphi(w_1 + 2^{|w_1|}\mathbf{x})$
$A_{\mathcal{C}_{w_1}(\mathbf{a}_i.\mathbf{x} \sim_i c'_i)}$
$\cdots$
$\varphi(w_k + 2^{|w_k|}\mathbf{x})$
$A_{\mathcal{C}_{w_k}(\mathbf{a}_i.\mathbf{x} \sim_i c'_i)}$

Figure 2.2: Automaton built for a formula, by unfolding first up to depth doubly exponential.

- for all $b \in \Sigma_r$, $\delta(\mathcal{C}(\psi_1(\mathbf{x}), \ldots, \psi_n(\mathbf{x})), b) = \mathcal{C}(\psi'_1(\mathbf{x}), \ldots, \psi'_n(\mathbf{x}))$ each $\psi_i(\mathbf{x})$ being a state, possibly $\perp$, of $\mathcal{A}_{\varphi_i}$ (the automaton of φ_i), and $\psi'_i(\mathbf{x}) = \delta_{\varphi_i}(\psi_i(\mathbf{x}), b)$.
- If $s \in S^r$, then $\delta(\mathcal{C}(\psi_1(\mathbf{x}), \ldots, \psi_n(\mathbf{x})), s) = q_f$, when $\langle s \rangle \in [\![\mathcal{C}(\psi_1(\mathbf{x}), \ldots, \psi_n(\mathbf{x}))]\!]$ and $\delta(\mathcal{C}(\psi_1(\mathbf{x}), \ldots, \psi_n(\mathbf{x})), s) = \perp$ otherwise.

It is clear that this construction provides us a deterministic finite automaton. This construction for an automaton for a quantifier free formula, together with the construction of automata for atomic constraints, will be crucial in the next section dedicated to establish a time upper bound on the inductive construction of a Presburger formula.

More precisely we will use this construction to build a large automaton accepting solutions of a formula, and show that on this large automaton, because of its structure, projection can be performed "efficiently", and show that the same argument carries on the minimized automaton, hence this gedenken automaton is built only to establish the upper bound.

2.3.3 Complexity of automaton construction

The well-known decision procedure for Presburger arithmetic using automata is based on recursively constructing an automaton accepting solutions of a Presburger formula by using automata constructions for handling logical connectives and quantifiers. Automata for basic formulas can be easily constructed (see section 2.2). Each logical connective ($\wedge$,$\vee$, $\neg$) corresponds then naturally to an operation on automata. Furthermore to get an automaton for $\exists y.\varphi(y, \mathbf{x})$ given an automaton for $\varphi(y, \mathbf{x})$ one *projects away*[1] the component for y and obtains a *non-deterministic* automaton. Then, this automaton is determinised to be able to continue the recursive

1. Since the automaton should accept all encodings of the solutions, one has to sometimes add additional transitions with a sign letter going to the final state.

construction and minimised. Given a formula of size n, Klaedtke has shown [Kla08] — by eliminating the quantifiers on the logical level and translating the obtained quantifier-free formula to an automaton — that the minimal DFA obtained *at the end* of this procedure and the minimal DFA for subformulas obtained *during* the procedure have triple-exponential size. However, the DFA obtained after determinising the NFA which is a result of a projection might be of quadruple exponential size, as an automaton of triple exponential size is determinised. In this section we show that *all* automata obtained during the construction are in fact of at most triple exponential size solving an open problem from [Kla08]. We do this by carefully inspecting the structure of the NFA which is determinised. Notice that our upper bound is obtained using the least significant bit first coding of integer vectors which allows to reason conveniently about states of the automata corresponding to formulas.

The following theorem on quantifier elimination is from Klaedtke [Kla08]. We use here a simplified version, where the only parameter is the length of the formula. In [Kla08], other parameters (e.g. alternation depth) are used. Given a quantifier free Presburger formula ψ, let d_ψ be the number of different atomic modulo constraints of the form $\mathbf{a}.\mathbf{x} \equiv_m \beta$ and $max_{div}(\psi)$ the biggest value of m appearing in them. Let t_ψ be the number of *different* vectors $\mathbf{a}$ appearing in atomic formulas of the form $\mathbf{a}.\mathbf{x} \sim \gamma$ with $\sim \in \{=, \neq, <, >, \leq, \geq,\}$ in ψ, $max_{coef}(\psi)$ the biggest absolute value of their coefficients and $max_{const}(\psi)$ the biggest absolute value of the constants γ appearing in them. We use the abbreviations $exp2(x) = 2^{2^x}$ and $exp3(x) = 2^{2^{2^x}}$.

Theorem 2.13 ([Kla08], Theorem 4.5)**.** *For every Presburger formula φ of length n, there is a semantically equivalent quantifier free formula ψ such that:*

- $t_\psi \leq exp2(cn \log n)$,
- $d_\psi \leq exp2(cn \log n)$,
- $max_{coef}(\psi) < exp2(cn)$,
- $max_{div}(\psi) < exp2(cn)$ *and*
- $max_{const}(\psi) < exp3(cn \log n)$,

where c is a constant independent of n.

We can suppose w.l.o.g. that ψ is a boolean combination of modulo constraints of the form $\mathbf{a}.\mathbf{x} \equiv_m \beta$ and of atomic formulas of the form $\mathbf{a}.\mathbf{x} > \gamma$ only. The following theorem gives a bound on the size of the minimal DFA accepting solutions of a Presburger formula. Klaedtke gives a corresponding theorem for the most significant bit first coding. His proof is simpler due to the fact that only one automaton has to be constructed for all inequations with the same coefficients.

Theorem 2.14. *The size of the minimal DFA accepting solutions of a Presburger formula φ of length n is at most $exp3(cn \log n)$ for some constant c.*

Proof. Let r be the number of free variables of φ. Let ψ be the quantifier free formula obtained from φ using Theorem 2.13. We have $t_\psi \leq exp2(c_1 n \log n)$, $d_\psi \leq exp2(c_1 n \log n)$, $max_{coef}(\psi) < exp2(c_1 n)$, $max_{div}(\psi) < exp2(c_1 n)$ and $max_{const}(\psi) <$

$exp3(c_1 n \log n)$ for some constant c_1. If we build the product automaton for the quantifier free formula ψ equivalent to φ according to Definition 2.12, a naive analysis of its size gives a quadruply exponential automaton, since there are possibly $t_\psi^{2max_{const}(\psi)}$ distinct inequations in ψ. However a closer inspection reveals a triple exponential bound, due to the special structure of automata for inequations. Here, we give a slightly different construction of the automaton A_ψ accepting solutions of ψ which we will use in the rest of the section.

Let $\mathbf{a}_1, \ldots, \mathbf{a}_{t_\psi}$ be the different vectors appearing in the atomic inequations of ψ and $\psi_1, \ldots, \psi_{l_\psi}$ an enumeration of all atomic formulas of the form $\mathbf{a}_i.\mathbf{x} > \gamma_j$ for all $1 \le i \le t_\psi$ and γ_j with $|\gamma_j| \in [-\|\mathbf{a}_i\|_+ - 1, \|\mathbf{a}_i\|_-]$. Clearly, $l_\psi \le exp2(c_2 n \log n)$ for some constant c_2. Let $\phi_1, \ldots, \phi_{d_\psi}$ be an enumeration of all the modulo constraints appearing in ψ and $\mathcal{BC}$ be the set of boolean combinations of the form $\mathcal{C}(\psi_1, \ldots, \psi_{l_\psi}, \phi_1, \ldots, \phi_{d_\psi})$. For each member of $\mathcal{BC}$ an automaton can be built with the product construction of Definition 2.12.

We describe now informally the automaton A_ψ we construct from ψ, represented in Figure 2.2. It has first the form of a complete tree starting at the initial state. Its branching factor is the size of the alphabet Σ_r and its depth is $exp2(c_1 n \log n)$. Each of the states in the tree recognises the solutions of the formula $\psi(2^{|w|}\mathbf{x} + \langle w \rangle_+)$ where $w \in \Sigma_r^*$ with $|w| \le exp2(c_1 n \log n)$ is the word leading to the state from the initial state. Then, at level $exp2(c_1 n \log n)$ there are separate automata accepting solutions of the corresponding formulas reached after reading the word leading to them. All these automata correspond to boolean combinations of $\mathcal{BC}$. Indeed, for any atomic formula $\zeta(\mathbf{x}) = \mathbf{a}.\mathbf{x} > \gamma$ of ψ and any word $w \in \Sigma_r^*$ with $|w| = exp2(c_1 n \log n)$ we have $\zeta(2^{|w|}\mathbf{x} + \langle w \rangle_+) \Leftrightarrow \mathbf{a}.\mathbf{x} > \gamma'$ for some $\gamma' \in [-\|a\|_+ - 1, \|a\|_-]$. Therefore, for any atomic subformula $\zeta(\mathbf{x})$ of ψ, $\zeta(2^{|w|}\mathbf{x} + \langle w \rangle_+)$ is equivalent to a ψ_i, so $\psi(2^{|w|}\mathbf{x} + \langle w \rangle_+)$ is equivalent to a formula of $\mathcal{BC}$. Notice that in any member of $\mathcal{BC}$ *all* atomic formulas of a given form appear. That is not a restriction, since we can just expand each boolean combination to be of this form. Let $W = \{w \in \Sigma_r^* \mid |w| = exp2(c_1 n \log n)\}$. For any $w \in W$, let $\mathcal{C}_w \in \mathcal{BC}$ be the boolean combination equivalent to $\psi(2^{|w|}\mathbf{x} + \langle w \rangle_+)$. For each $\mathcal{C}_w$ we can construct an automaton $A_{\mathcal{C}_w} = (Q_w \cup \{F\}, q_{w,0}, \{F\}, \delta_w)$ according to Definition 2.12. Notice that the automata $A_{\mathcal{C}_w}$ only differ in the transitions going to the final state, since the atomic formulas composing them are all the same. The final state F is the same in each automaton.

We can now give the definition of the automaton for the formula ψ formally, i.e. $A_\psi = (Q, q_\epsilon, \{F\}, \delta)$ where $Q = Q_1 \cup Q_2 \cup \{F\}$ with $Q_1 = \{q_w \mid w \in \Sigma_r^* \wedge |w| < exp2(c_1 n \log n)\}$ and $Q_2 = \bigcup_{w \in W} Q_w$. Furthermore, $\delta(q_w, b) = \{q_{wb}\}$ for all $b \in \Sigma_r$ and $|w| < exp2(c_1 n \log n) - 1$, $\delta(q_w, b) = \{q_{wb,0}\}$ for all $b \in \Sigma_r$ and $|w| = exp2(c_1 n \log n) - 1$ and $\delta(q, b) = \delta_w(q, b)$ for all $b \in \Sigma_r$ and $q \in Q_2$. Clearly, the number of states (and also the size) of the automaton A_ψ is smaller than $exp3(cn \log n)$ for some constant c. □

We can now prove the main theorem of the section which shows that elimination of a variable does not lead to an exponential blow-up in the size of the automaton.

Theorem 2.15. *Let $\exists y.\varphi(y, \mathbf{x})$ be a Presburger formula of size n, A the minimal DFA accepting the solutions of $\varphi(y, \mathbf{x})$ and A' the automaton obtained by projecting*

A on $\mathbf{x}$. Then, the automaton A'' obtained by determinising A' with the standard on-the-fly subset construction is of size at most $exp3(cn\log n)$ for some constant c.

Proof. Theorem 2.13 yields a quantifier free formula ψ semantically equivalent to $\varphi(y,\mathbf{x})$ with $t_\psi \leq exp2(c_1 n\log n)$, $d_\psi \leq exp2(c_1 n\log n)$, $max_{coef}(\psi) < exp2(c_1 n)$, $max_{div}(\psi) < exp2(c_1 n)$ and $max_{const}(\psi) < exp3(c_1 n\log n)$ for some constant c_1. For ψ, according to Theorem 2.14, there is an automaton $A_\psi = (Q, q_0, \{F\}, \delta)$ of triple-exponential size (not necessarily minimal) accepting the solutions of ψ. We use the same notation as in the proof of Theorem 2.14 for the parts of the automaton. A is the minimal automaton corresponding to A_ψ. Obviously, A_ψ might be bigger than A. We show that the size of the automaton A''_ψ obtained by determinising (using the standard on-the-fly subset construction) A'_ψ which is the automaton obtained from A_ψ by projecting away y is bounded by $exp3(cn\log n)$. Then the bound on A'' (whose states are sets of states of A') follows, as two different states in A'' correspond to two different states in A''_ψ.

Since we use for the determinisation of A'_ψ the standard subset construction, states of A''_ψ are sets of states of A'_ψ which are exactly the states of A_ψ.

We first introduce some notations. Let r be the number of free variables of $\varphi(y,\mathbf{x})$ and ψ. For any word $w \in \Sigma_r^*$ we denote by $w{\downarrow_1} \in \Sigma_{r-1}^*$ the word obtained from w by projecting away the first component and by $w{\downarrow_2} \in \{0,1\}^*$ the word obtained from w by projecting on the first component. For any $w \in \Sigma_{r-1}^*$ we define $w{\uparrow} = \{w' \in \Sigma_r^* \mid w'{\downarrow_1} = w\}$. For any $w \in \Sigma_{r-1}^*$ and $z \in [0, 2^{|w|}-1]$ we define $w{\uparrow^z} = w'$, if $\langle w'{\downarrow_2}\rangle_+ = z$ and $w'{\downarrow_1} = w$.

Let $S = \{\widehat{\delta}(w{\uparrow}, \{q_0\}) \mid w \in \Sigma_{r-1}^*\}$. Our goal is to show that the size of S is bounded by a triple-exponential. This implies that the number of states of A''_ψ has the same bound. We split S into two sets $S_<$ and $S_\geq$, where:

$S_< = \{\widehat{\delta}(w{\uparrow}, \{q_0\}) \mid w \in \Sigma_{r-1}^* \wedge |w| < exp2(c_1 n\log n)\}$ and

$S_\geq = \{\widehat{\delta}(w{\uparrow}, \{q_0\}) \mid w \in \Sigma_{r-1}^* \wedge |w| \geq exp2(c_1 n\log n)\}$.

It is obvious that $|S_<| \leq exp3(c_2 n\log n)$ for some constant c_2. We now show a bound on $|S_\geq|$. We first enumerate all words $w \in \Sigma_{r-1}^*$ of size exactly $exp2(c_1 n\log n)$ as $w_1, \ldots, w_m$ where $m \leq exp3(c_3 n\log n)$ for some constant c_3.
We have $S_\geq = \bigcup_{i=1}^m S_i$ where $S_i = \{\{\widehat{\delta}(w_i w{\uparrow}, \{q_0\}) \mid w \in \Sigma_{r-1}^*\}$. We will show that $|S_i| \leq exp3(c_4 n\log n)$ for some constant c_4 which implies that $S_\geq$ is bounded by a triple-exponential as well. We have:

$$S_i = \{\widehat{\delta}(w{\uparrow}, \widehat{\delta}(w_i{\uparrow}, \{q_0\})) \mid w \in \Sigma_{r-1}^*\} = \bigcup_{z\in[0,(exp3(c_1 n\log n)-1)]} \{\widehat{\delta}(w{\uparrow}, \widehat{\delta}(w_i{\uparrow^z}, \{q_0\})) \mid w \in \Sigma_{r-1}^*\}$$

Let $S_i^0 = \{\widehat{\delta}(w{\uparrow}, \widehat{\delta}(w_i{\uparrow^0}, \{q_0\})) \mid w \in \Sigma_{r-1}^*\}$. We have $|S_i| = |S_i^0|$, as $\widehat{\delta}(w_i{\uparrow^0}, \{q_0\})) = \mathcal{C}_{w_i\uparrow^0}$ and for all $0 < z \leq exp3(c_1 n\log n) - 1$ we have $\widehat{\delta}(w_i{\uparrow^z}, \{q_0\}) = \mathcal{C}_{w_i\uparrow^z}$ and all automata corresponding to $\mathcal{C}_{w_i\uparrow^z}$ for $0 \leq z \leq exp3(c_1 n\log n) - 1$ are the same except for the transitions leading to the final state. That means that there is a one-to-one correspondence between each state in sets of states of S_i^0 and each state in sets of states of the other S_i^z.

Now, we derive a bound of $|S_i^0|$. The word $w_i{\uparrow^0}$ leads to the state $\mathcal{C}_{w_i\uparrow^0}$ in A_ψ which is the initial state of an automaton, call it A_0, recognising all solutions of

$\mathcal{C}_{w_i\uparrow^0}$. A_0 is obtained as a product of automata for atomic inequations and modulo constraints. Let $\psi_1, \ldots, \psi_{l_\psi}$ be the atomic formulas which are inequations and $\phi_1, \ldots, \phi_{d_\psi}$ the atomic formulas which are modulo constraints of the boolean combination $\mathcal{C}_{w_i\uparrow^0}$. In the following it is convenient to consider states of A_0 to be $(l_\psi + d_\psi)$-tuples of states instead of considering them as formulas. That is, a state $\mathcal{C}(\psi'_1, \ldots, \psi'_{l_\psi}, \phi'_1, \ldots, \phi'_{d_\psi})$ is considered to be the tuple $(\psi'_1, \ldots, \psi'_{l_\psi}, \phi'_1, \ldots, \phi'_{d_\psi})$. For $1 \leq i \leq l_\psi$ let $\psi_i(y, \mathbf{x}) = a^i_1 y + \mathbf{a}^i.\mathbf{x} > \gamma_i$ and for $1 \leq i \leq d_\psi$ let $\phi_i(y, \mathbf{x}) = b^i_1 y + \mathbf{b}^i.\mathbf{x} \equiv_{m_i} \beta_i$.

Let us fix a $w \in \Sigma^*_{r-1}$ and let $w' \in w\uparrow$. Let $y' = \langle w'\downarrow_2\rangle_+$ and $l = |w| = |w'|$. It is clear that $0 \leq y' < 2^l$. For each $1 \leq i \leq t_\psi$, the state reached in $A_{a^i_1 y + \mathbf{a}^i.\mathbf{x} > \gamma_i}$ by w' is the state semantically equivalent to $a^i_1(2^l y + y') + \mathbf{a}^i.(2^l \mathbf{x} + \langle w\rangle_+) > \gamma_i$ which is equivalent to $a^i_1 y + \mathbf{a}^i.\mathbf{x} > (\gamma_i - a^i_1 y' - \mathbf{a}.\langle w\rangle_+)/2^l$.

Therefore, the first t_ψ components of the states reached in A_0 by words $w' \in w\uparrow$ are semantically equivalent to $(a^1_1 y + \mathbf{a}^1.\mathbf{x} > (\gamma_1 - a^1_1 y' - \mathbf{a}.\langle w\rangle_+)/2^l, \ldots, a^{t_\psi}_1 y + \mathbf{a}^{t_\psi}.\mathbf{x} > (\gamma_i - a^{t_\psi}_1 y' - \mathbf{a}^{t_\psi}.\langle w\rangle_+)/2^l)$. There are 2^l different values for y'. However there are at most $\sum_{i=1}^{t_\psi}(|a^i_1| + 1)$ semantically different corresponding t_ψ-tuples of formulas, since $0 \leq y' < 2^l$ and therefore the semantics of the i-th atomic formula changes at most a^i_1 times in a monotone fashion for increasing y'. Therefore if we consider the first t_ψ components of states reached by words of $w\uparrow$ in A_0, we get only $\sum_{i=1}^{t_\psi}(|a^i_1| + 1)$ semantically different ones, since the automata for basic formulas are minimal.

Now we consider the set of words $V = \{w' \in \Sigma^*_r \mid w'\downarrow_1 = w\}$ which lead to the *same* first t_ψ components of states in A_0 and consider the other components (corresponding to the modulo constraints) they can reach. The words in V differ only in the component corresponding to y. Clearly, the set $\{y' \mid y' = \langle w'\downarrow_2\rangle_+$ and $w' \in V\}$ is an interval of the form $[p, q]$ where $0 \leq p \leq q < 2^l$.

A state (formula) reached in $A_{b^i_1 y + \mathbf{b}^i.\mathbf{x} \equiv_{m_i} \beta_i}$ after reading a word w' of V with $y' = \langle w'\downarrow_2\rangle_+$ is semantically equivalent to $2^l(b^i_1 y + \mathbf{b}^i.\mathbf{x}) \equiv_{m_i} \beta_i - b^i_1 y' - \mathbf{b}^i.\langle w'\rangle_+$. It is clear that there are at most m_i semantically different formulas of this kind. Furthermore, we can order them starting from $y' = 0$ until $y' = m_i - 1$. Then it is clear that the set of states (formulas) reached by words of V (whose corresponding y' form intervals) must be an interval of states respecting this order. There are at most m_i^2 such intervals.

Finally, we can give an upper bound on the number of subsets of states of $S^0_i = \{\widehat{\delta}(w\uparrow, \{q_{w_i,0}\} \mid w \in \Sigma^*_{r-1}\}$ which are subsets of states of A_0. Given any word $w \in \Sigma^*_{r-1}$ we know from the above that words of $w\uparrow$ lead to at most $s := \sum_{i=1}^{t_\psi}(|a^i_1| + 1) \leq exp2(c_5 n \log n)$ (for some constant c_5) different tuples of the first t_ψ components of A_0. Furthermore, we know that the number of subsets of states of A_{ϕ_i} which can be reached simultaneously by words of subsets V of $w\uparrow$ such that all $w' \in V$ lead to the same tuples of the first t_ψ components is at most m_i^2. Therefore overall, S^0_i has at most $|A_0|^s \Pi_{i=1}^{d_\psi} m_i^2 \leq exp3(cn \log n)$ states for some constant c and $|A_0|$ being the number of states of A_0. From this follows in turn a triple-exponential bound on $|S_i|$, $|S_\geq|$, the number of states and size of A''_ψ and A''. □

The number of transitions of an automaton is bounded by $|Q||\Sigma|$ for a DFA and possibly $|Q|^2|\Sigma|$ for an NFA. As Σ is at most simply exponential w.r.t. the size of the formula, the sizes of the automata build have all a triple-exponential upper bound as well. Therefore the following is an easy consequence of Theorem 2.15.

Corollary 2.16. *The automata based decision procedure for Presburger arithmetic takes triple-exponential time in the size of the formula.*

2.4 Conclusion

In this section, we showed that the inductive construction of the automaton accepting solutions of a Presburger formula, using a special least significant digit first takes time triply exponential. First we can remark that encoding the sign bit in a separate alphabet is not crucial for the upper bound to hold: remark that the encoding of the sign can be performed in Σ, at the small cost of remembering on each Q_2 state the transition that lead to that state (then acceptance of these new states can easily be decided).

It seems natural to wonder whether using a most-significant digit first coding of integers yields the same time complexity. Notice that reversing the coding gives us at the beginning of the word the information of the sign of each of the component. In the next section, we describe a generic way to give complexity upper bounds on the time of inductive construction of automata for automatic structures, and incidently prove that for Presburger Arithmetic a 3EXPTIME upper bound also holds using the most significant digit first encoding.

Chapter 3

Automatic structures

The triply exponential time upper bound on the inductive construction of an automaton for a Presburger formula motivates the use of this generic algorithm. The proof of complexity presented in the previous chapter (and only the proof, not the algorithm) however relied on results on Presburger quantifier elimination procedure.

In this chapter we establish a generic criterion on automatic presentations in order to establish upper bounds on the generic inductive algorithm to build automata accepting solutions of first-order formulas.

We then apply this criterion on some automatic presentations of structures with an elementary first-order theory (Presburger and Skolem arithmetics, automatic structures of bounded degree and nested pushdown graphs), to establish upper bounds on the time of construction of automata accepting solutions of formulas, and these bound closely match the lower bounds of complexity of the first-order logics of these structures.

3.1 Presenting structures with automata

The automata based decision procedure for Presburger Arithmetic arises essentialy from the fact that addition can be presented as an automaton. The main idea lying behind automatic structures is that some structures –and not just Presburger Arithmetic– are regular, in the sense that the domain is regular (it can be seen as a regular tree language), and the predicates are regular relations (they can be seen as synchronoulsy regular tree languages). In this section we first detail the notion of synchronoulsy regular relations over finite trees, and what automata accept those, and then how the regularity of the structures implies the regularity of any first-order relation.

In this section, we will present a generic criterion that we can check for some automatic presentation to derive an upper bound on the time of inductive construction of automata for first-order formulas. We show we can apply this criterion to some automatically presented structures (with elementary first-order theory) and establish upper bounds on the deterministic time of construction of automata, that closely match the complexity of those structures.

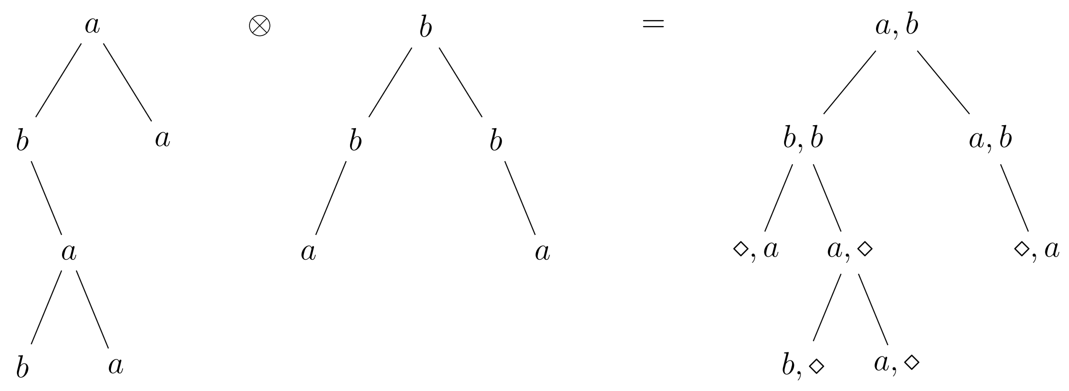


Figure 3.1: Example of convolution of trees.

Coding (tuples of) elements of the domain.

In order to represent "regular" relations over trees, we need to represent tuples of trees as trees, so that we can state that a relation is regular if the language of those tuple of trees is a regular tree language.

The idea is to encode an r-tuple of trees as a tree over r-tuples of letters.

Definition 3.1. Let Γ be a finite alphabet and let $\diamond \notin \Gamma$ be an additional padding symbol. Let $t_1, \dots, t_r \in T_\Gamma$, the *convolution* $t = t_1 \otimes \dots \otimes t_r$, is a finite binary tree over the alphabet $\Gamma^{\otimes r} = (\Gamma \cup \{\diamond\})^r \backslash \{\diamond\}^r$, defined as follows :

- $\text{dom}(t) = \bigcup_{i=1}^{r} \text{dom}(t_i)$ and for all $w \in \text{dom}(t)$,
- $t(w) = (a_1, \dots, a_m)$, where $a_i = t_i(w)$ if $w \in \text{dom}(t)$ and $a_i = \diamond$ otherwise.

Essentially convolution of r trees of T^*_Σ is an injective way to present them as a single tree of $T^*_{\Sigma^r}$. We give an example of convolution in Figure 3.1. Notice that $\diamond^r \notin \Gamma^{\otimes r}$, as this symbol would indicate that no tree has a node at that position, we reflect that simply with no node.

We define $T_\Gamma^{\otimes r}$ the set of finite binary trees in $T_{\Gamma^{\otimes r}}$ that are convolutions of r trees in T_Γ.

This set is regular and accepted by a deterministic automaton namely A_r with $2^r + 1$ states. A_r is defined as follows:

- Its alphabet is $\Gamma^{\otimes r}$
- $Q_r = \{0,1\}^r \cup \{\bot\}$, its initial state is $\{0\}^r$
- $F = \{0,1\}^r$
- For any $(q'_1, \dots, q'_r) \in Q_r$, $(q''_1, \dots, q''_r) \in Q_r$, and $(a_1, \dots, a_r) \in \Gamma^{\otimes r}$,
  - $\delta(\bot, q, \alpha) = \delta(q, \bot, \alpha) = \bot$ for any $q \in Q_r$ and any $\alpha \in \Gamma^{\otimes r}$
  - $\delta((q'_1, \dots, q'_r), (q''_1, \dots, q''_r), (a_1, \dots, a_r)) = \bot$
    if for some $i \in [1, r]$ $q'_i = 0$ and $q''_i = 1$, or $q'_i = 1$ and $a_i = \diamond$
    else $\delta((q'_1, \dots, q'_r), (q''_1, \dots, q''_r), (a_1, \dots, a_r)) = (q_1, \dots, q_r)$ with $q_i = 0$ if $q'_i = q''_i = 0$ and $a_i = \diamond$, else $q_i = 1$.

Clearly this automaton is built within time polynomial w.r.t. $|\Gamma|^r$.

Remark 3.2. This notion of synchronizing trees obviously maps to words (as words are trees), though this means synchronizing words on their last letter (the last letter of the words is the root of the corresponding tree), as in [DGH12], and not on the first letter as it is usually done in string automatic structures.

There is a bijection between (the regular language) $T_\Gamma^{\otimes r}$ and $(T_\Gamma)^r$ (r-tuples of trees over Γ). In the sequel we will manipulate automata, that accept languages included in $T_\Gamma^{\otimes r}$ (for some r). These automata will therefore accept (r-ary) relations over trees in T_Γ; an automaton over $\Gamma^{\otimes r}$ acccepts the r-ary relation $R(A) = \{(t_1, \dots, t_r) \mid t_1 \otimes \dots \otimes t_r \in L(A)\}$.

It is now important to remark that from an automaton acepting a r-ary relation, there are some operation that can be easily performed:

- permuting the relation with a permutation σ: it suffices to apply the permutation on each letter in each transition of the automaton.
- enforcing equality between the i-th and j-th components of the relations: it suffices to remove any transition labeled by a letter whose i-th and j-th component differ.

Automatic presentation.

Definition 3.3. A structure is tree automatic iff it has an automatic presentation. An (injective) automatic presentation is a tuple $(\Gamma, A_D, (A_P)_{P\in\mathcal{S}})$ such that

- Γ is a finite alphabet
- A_D is a tree automaton over Γ, whose language $L(A_D)$ is isomorphic to the domain of the structure. Hence each tree in $L(A_D)$ represents an element of the domain.
- $(A_P)_{P\in\mathcal{S}}$ are tree automata over $\Gamma^{\otimes ar_P}$ that accept the relations induced by each of the predicate (using the representation of elements of the domain as trees in $L(A_D)$).

A structure is word automatic if we can further impose that A_D accepts only words (i.e. trees with domains in 0^*).

The definition that we gave here is usually considered to be that of injective automatic presentation, non injective automatic presentation are defined as follows:

Definition 3.4. A non-injective automatic presentation of a structure S with signature $\mathcal{S}$ is an automatic presentation $AP = (\Gamma, A_D, (A_P)_{P\in\mathcal{S}})$ together with an automaton $A_=$ over $\Gamma^{\otimes 2}$ that accepts a reflexive symetric transitive relation over $L(A_D)$ and defines an equivalence relation, which is a congruence for each of the $R(A_P)$, such that S is isomorphic to the quotient of the automatic presentation AP by the equivalence relation $R(A_=)$.

It is a well-known fact that we can build from a non-injective automatic presentation an automatic presentation, by restricting the domain to the set of smallest representative of each equivalence class of $L(A_P)$. As the lexicographical (linear) order (denoted $<_{lex}$) can be expressed by an automaton, it suffices to build an automaton accepting $\varphi(x) = R(A_D)(x) \wedge \neg\exists y \cdot R(A_=)(x, y) \wedge (y <_{lex} x)$. Notice that even if the presentation is with non-deterministic automata, the automaton accepting φ is at most exponential w.r.t. the size of the automatic presentation.

Theorem 3.5. *[KN94, BG00]*
Given an automatic presentation of some $\mathcal{S}$-structure, any first order relation is accepted by some tree automaton.

The proof of this theorem is constructive: given a first order formula its automaton can be built inductively. This construction will be inspected in section 3.2.2.

3.2 Ehrenfeucht-Fraïssé to bound the time of construction

It is not possible to provide an elementary upper bound on the size (and hence time of construction) of an automaton accepting solutions of a first-order formula of an automatic structure, since for some structures this automaton is known to be of non-elementary size, for example for the structure $(\mathbb{N}, +, \epsilon_2)$ (where $\epsilon_2(x, y)$ holds when x is a power of two appearing in the base 2 decomposition of y) to given by Büchi [Büc60], which yields non-elementary automata because of the non-elementary lower bound exhibited by [SM73]. For some structures however this is possible. We detail here what properties we need to ensure to state such an upper bound.

3.2.1 Ehrenfeucht-Fraïssé back-and-forth relations

Definition 3.6. Given a structure $\mathcal{A} = (A, (P^{\mathcal{A}}))$, we say that $u, v \in A^r$ are FO^r_m-*undistinguishable* iff for any formula φ with quantifier depth m and r free variables, we have $\mathcal{A} \models \varphi(u) \iff \mathcal{A} \models \varphi(v)$; we write $u \equiv^r_m v$.

Remark 3.7. The number of equivalence classes of $\equiv^r_m$ is finite, because of the finiteness of FO^r_m relations.

The idea, introduced by Ferrante and Rackoff [FR79] is to provide for a given structure, a family of relations that refine $\equiv^r_m$. They exhibit that their relations have a small representative in each equivalence class, that is a representative that can be represented on a Turing tape within an appropriate space, and hence they deduce a space constrained algorithm to decide first-order theory over that structure. That allowed to establish optimal upper bounds for many structures, noticeably Presburger and Skolem Arithmetic, and the theory of real addition, and also many other. They rely on a back-and-forth property of the relation to prove their relation refining $\equiv^r_m$ which is essentially an induction over m and formalized in the following theorem:

Theorem 3.8. *Given a structure $\mathcal{A}$ and $(E^r_m)_{m\geq 0, r\geq 1}$ a family of (symetric, reflexive and transitive) relations such that for any m, r:*

- *E^r_0 refines $\equiv^r_0$*
- *if $uE^r_{m+1}v$ then for any u', there exists a v' such that $(u,u')E^{r+1}_m(v,v')$.*

Then uE^r_mv implies $u \equiv^r_m v$.

Proof. The proof is by induction over m. Assume $m = 0$, then by hypothesis, we have uE^r_0v implies $u \equiv^r_0 v$. Assume that for some m, for any r, uE^r_mv implies $u \equiv^r_m v$, we have to show that for any r, uE^r_{m+1} implies $u \equiv^r_m v$. We show that by contradiction: assume $uE^r_{m+1}v$, and a formula $\varphi \in FO^r_{m+1}$ such that $\mathcal{A} \vDash \varphi(u)$ and $\mathcal{A} \vDash \varphi(v)$, without loss of generality, we can assume that no subformula of φ distinguishes u and v, so $\varphi = \exists y.\psi(y, \bar{x}_r)$ for some formula $\psi \in FO^{r+1}_m$. As $\mathcal{A} \vDash \varphi(u)$ then there exists u' such that $\mathcal{A} \vDash \psi(u, u')$. According to the second hypothesis, there exists a v' such that $(u,u')E^{r+1}_m(v,v')$, by hypothesis of induction this implies $(u,u') \equiv^{r+1}_m (v,v')$, which means (u,u'), and (v,v') satisfy the same formulas in FO^{r+1}_m, so $\mathcal{A} \vDash \psi(v,v')$, which means that $\mathcal{A} \vDash \varphi(v)$ which contradicts our assumption. Thus E^r_{m+1} refines $\equiv^r_{m+1}$ which concludes the proof. □

3.2.2 Polynomial time language projection

Ladner [Lad77] was the first to use model theoretic games to give an upper bound on the size of automata for an MSO sentence. Then Klaedtke [Kla10] and Eisinger [Eis08] used this to bound the size of the minimal automaton accepting solutions of a formula in FO^r_m by finding a family of equivalence relations over words (they worked over string automata) that refines both the Myhill-Nerode equivalence (in the language of solutions of any formula in FO^r_m) and $\equiv^r_m$. Hence the bound on the index of this relation is also a bound of the number of residuals in the language of solutions of a formula in FO^r_m, hence a bound on the size of the minimal deterministic automaton. But if we further impose that this family of relations also has a back-and-forth property, then we also get that the projection of a language of solutions can be performed within time polynomial w.r.t. the index of this relation.

We now formally present the criterion (parametrized by a function f) that will allow to establish an upper bound on the size of automata and its time of construction w.r.t. the quantifier depth an the number of free variables of the corresponding formula. As the quantifier depth and the number of free variables are smaller than the size of the formula, this establishes the same upper bound w.r.t. the size of the formula.

Theorem 3.9. *Given an automatic presentation AP, and E^r_m a family of binary reflexive symetric and transitive relations over $T_{\Gamma^{\otimes r}}$ such that:*

1. *For any m, the set of trees that are not convolution of r trees in T_Γ are alone in a same E^r_m equivalence class. Also the empty tree is always alone in its E^r_m equivalence class.*
2. *For any $u, v \in T_{\Gamma^{\otimes r}}$, if $uE^r_m v$, and u is a convolution of trees in D, then so is v and the r-tuples presented by u and v satisfy the same atomic formulas in the structure presented by AP.*
3. *For any $u, v \in T_{\Gamma^{\otimes r}}$, if $u \in T_\Gamma^{\otimes r}$, and $uE^r_{m+1}v$, then for any $u_{r+1} \in T_\Gamma$ there exists a $v_{r+1} \in T_\Gamma$ such that $u \otimes u_{r+1} E^{r+1}_m v \otimes v_{r+1}$.*
4. *For any $u, v \in T_{\Gamma^{\otimes r}}$, if $uE^r_m v$ then for any context c in $C_{\Gamma^{\otimes r}}$, $c[u]E^r_m c[v]$.*
5. *The index of E^r_m is bounded by $f(m + r)$ for some function f.*

Then the inductive construction of a deterministic bottom-up tree automaton accepting solutions of a formula $\varphi \in FO^r_m$ leads to an automaton with a number of states bounded by $f(m+r)$ and can be done within time bounded by $c_1|\varphi|((|AP|^{m+r}f(m+r))^{c_2}$, for some constants c_1 and c_2 independant of the presentation.

$|\varphi|$ denotes the number of symbols in the formula φ, which will also be referred to as the size of the formula φ. Similarily $|AP|$ denotes the size of the automatic presentation, the sum of the sizes of all the automata. Clearly $|AP|$ is always greater than the number of states in any of the automaton in the automatic presentation; also $|AP|$ is always greater than the number of symbols in the alphabet Γ. To prove the complexity bound stated in the theorem, we will rely on the well-known complexity results over tree automata [1].

The rest of this subsection is dedicated to the proof of this theorem. We would like to insist on two aspects of this theorem: first that such a family of relations implies an upper bound on the number of states in the minimal tree automaton accepting solutions of a formula, but also on all the automata inductively built (it is shown by exhibiting that two trees equivalent –for some equivalence class– will reach the same state in that automaton), from that we will deduce the time of the inductive automaton construction is polynomial w.r.t. the size of the formula and hence the upper bound on the number of states –provided this upper bound is (super)exponential.

To achieve the proof of the theorem, we now have to detail a careful complexity analysis of the inductive automaton construction. Given a formula φ (with $|\varphi|$ symbols), quantifier depth m, and r free variables, we will inductively build for every subformula ψ of φ an automaton (namely A_ψ) accepting exactly solutions of ψ.

1. We may need to be a bit careful about the low-level representation of automata: their states have to be writeable in space logarithmic w.r.t. the size of the automaton, and letters of the alphabet also have to be writeable in space logarithmic w.r.t. the size of the alphabet. It should be clearly the case as the alphabets we manipulate are $(\Gamma^{\otimes r})_{r\geq 1}$ whose letters can be written in logarithmic space provided letters of Γ can.

Lemma 3.10. *Assume that E^r_m satisfies the hypotheses of theorem 3.9. Given a formula $\varphi \in FO^r_m$, let ψ a subformula at depth k in φ (that is ψ is under the scope of exactly k quantifiers in φ), then the automaton (inductively) built for the subformula ψ (which is in FO^{r+k}_{m-k}) has the property that any two trees in the same E^{r+k}_{m-k} equivalence class reach the same state in that automaton.*

Proof. This lemma is best proved by induction over ψ. We need however to be careful about our statement that a subformula at depth k in φ is in FO^{r+k}_{m-k}, indeed there can be less than r free variables syntactically occuring in ψ. There are two approaches in the construction: either we treat all subformulas at some depth k having exactly $r+k$ free variables, which will be the way this proof is done, we will need to carefully handle the case of atomic formulas which can have many more variables than only those syntactically occuring; or we can build for each subformula an automaton having no more tracks (that is accepting trees in $\Gamma^{\otimes r}$ with r no greater) than different free variables names in the formula. In the second case, the conjunction should be treated carefully and an operation of *cylindrification* should be performed on each of the automata representing solutions of the two conjuncts. Cylindrification consists of building from an automaton A_φ accepting trees representing r-tuples that are solutions of φ, an automaton A'_φ accepting trees representing r'-tuples ($r' > r$) whose r first components are solutions of φ (note that we consider here the r first components, but reordering the components can easily be performed by a permutation over letters on the transitions of the automaton, which preserves the property stated by the lemma). We however choose here to only perform cylindrification on atomic formulas, as it requires a minimization of the obtained automaton to ensure the property of the lemma.

We now present how to perform cylindrification of an r-ary automaton accepting solutions of formula φ to an r'-ary automaton which accept r'-tuples of trees in the domain of the structure and whose first r components represent solutions of φ. We make the product of the set of states of A_φ with $(r'-r)$ many times the set of states of A_D, the transition relation $\Delta_{A'_\varphi}$ of A'_φ is define as follows:

$$\Delta((q, q_1, \dots, q_{r'-r}), (q', q'_1, \dots, q'_{r'-r}), (a_1, \dots, a_{r'})) = (q'', q''_1, \dots, q''_{r'-r})$$

where q'' is either $\Delta_{A_\varphi}(q, q', (a_1, \dots, a_r))$ if $(a_1, \dots, a_r) \in \Gamma^{\otimes r}$ or the initial state of A_φ if q is the initial state of A_φ and $(a_1, \dots, a_r) = (\diamond, \dots, \diamond)$; and for any $1 \leq i \leq r'-r$, $q'_i = \delta_{A_D}(q_i, a_{r+i})$ if $a_{r+i} \in \Gamma$ and the initial state of A_D if q_i is the initial state of A_D and $a_{r+i} = \diamond$. The final states of A'_φ are the tuples of final states, thus if a tree of $\Gamma^{\otimes r'}$ is accepted, the fact that the first component of the final state is a final state in A_φ imposes that the r first components represent a solution of φ. Every other components of the final state being a final state in A_D impose that every other components are trees in the domain.

For **atomic subformulas**, we will show that the minimal automata are built within time polynomial w.r.t. $|AP|^{m+r}$. Then from their minimiality and the hypotheses on (E^r_m) we will deduce that they have the property that two words E^{r+k}_{m-k} reach the same state.

- If ψ is of the form $x = x$, from the tree automata A_{r+k} (accepting convolutions of $r+k$ trees) and A_D, we will build an automaton (namely $A_{D^{r+k}}$) that only accepts trees that accepts convolutions of words of the domain. This is achieved from A_{r+k} with which we make $r+k$ times a product with A_D, for each track.

- If ψ is of the form $x = y$ (with x and y two different variables), we start from the automaton A_{r+k} and we make a language intersection with the automaton that accepts equality between the components corresponding to x and y.

- If ψ is of the form $P(x_{i_1}, \ldots, x_{i_{ar_P}})$, with $(i_j)_{1\leq j\leq ar_P}$ a ar_P-tuple of elements in $[1, r+k]$. We start by picking the automaton accepting the predicate P in the automatic structure. For any j, j' such that $i_j = i_{j'}$, we make a language intersection of this language and the automaton accepting the equality between the j-th and the j'-th variable. Then we project away the redundant tracks [2], which consists in projecting these redundancies on each transition, which will lead to a deterministic automaton as we cast away redundancies. Finally we need to perform the appropriate cylindrification cylindrification and minimize the obtained automaton.

It should be clear that these can be built (minimal) within time polynomial w.r.t. $|AP|^{m+r}$, we have to show that any two trees in the same E^{r+k}_{m-k} equivalence class reach the same state in any of these minimal automata. Assume $t_1 E^r_m t_2$, then for any context c such that $c[t_1]$ is a solution of the atomic formula, $c[t_2]$ also is according to hypotheses 2 and 4 of the theorem, hence t_1 and t_2 reach the same state in the automaton because it is minimal. We strongly emphasize that here we need the automaton to be minimal, typically when the domain is not subtree-closed, the synchronized product of A_{r+k} with $r+k$ times A_D is not necessarily minimal and we cannot apply hypothesis 4 without minimizing.

If ψ is a **conjunction** of two formulas ψ_1 and ψ_2 then clearly as the automaton for ψ is built as the language intersection of the language of solutions of ψ_1 and ψ_2, the property that two E^{r+k}_{m-k} equivalent trees reach the same state is obviously preserved. Furthermore, as by definition $|\psi| = 1 + |\psi_1| + |\psi_2|$; by induction hypothesis, we just need to show that A_ψ can be built from A_{ψ_1} and A_{ψ_2}, within time bounded by $c_1|\psi|(|AP|^{m+r} f(m+r))^{c_2}$ for some constant c_1 and c_2, which is a well-known result about the complexity of language intersection.

If ψ is a **negation** of a formula ψ', one should be careful how the automaton accepting solutions of the formula ψ is built, first we complement the automaton for ψ' (which by induction also satisfies the property), and we make a language intersection with the minimal automaton $A_{D^{r+k}}$. So the automaton for ψ is built as a product automaton from two automata that both satisfy the property that two E^{r+k}_{m-k} equivalent words reach the same state (the complement of $A_{\psi'}$ satisfies this property by induction, and the automaton accepting trees that represent $r+k$-tuples of elements of the domain satisfy the property for $m = 0$, hence for any m), hence the automaton for ψ clearly satisfy the property. A_ψ is built from $A_{\psi'}$ within time

2. to build an automaton accepting $P(x, x, y)$ we essentially build an automaton for $\exists z \cdot P(x, z, y) \wedge x = z$. With a simple syntactic rewriting, we could impose that all variables names appearing inside an atomic predicate must be different. However the complexity proof allows to treat any atomic predicate as such.

bounded by $c_1|\psi|(|AP|^{m+r}f(m+r))^{c_2}$, hence A_ψ is built from scratch within time bounded by $c_1(1+|\psi'|)(|AP|^{m+r}f(m+r))^{c_2}$.

The hard case of the induction is the **quantification over a formula**. $\psi = \exists y.\psi'(\bar{x}_r, y)$, where ψ' has $r+k+1$ free variables and quantifier-depth at most $m-k-1$. Its automaton $A_{\psi'} = (\Gamma^{\otimes r+k+1}, Q_{\psi'}, F_{\psi'}, \delta_{\psi'})$ has the property that two trees E^{r+k+1}_{m-k-1} equivalent reach the same state. Without loss of generality, we assume that the track corresponding to the variable y in $A_{\psi'}$ is the $(r+k+1)$-th. Let $A'_\psi = (\Gamma^{\otimes r+k}, Q_\psi, F_\psi, \delta_\psi)$ be the (non-deterministic) automaton constructed as follows:

Let us denote $Q_\diamond$ the set of states in $A_{\psi'}$ that are reachable by a tree in $T_{\{\diamond\}^{r+k}\times\Gamma}$ (remark that $\{\diamond\}^{r+k}\times\Gamma \subset \Gamma^{\otimes r+k+1}$).

- $Q_\psi = Q_{\psi'}$
- $F_\psi = F_{\psi'}$
- $(a,q) \in \delta_\psi$ iff $\exists b \in \Gamma\cup\{\diamond\}, ((a,b),q) \in \delta_{\psi'}$ or $\exists q_1 \in Q_\diamond, (q_1,(a,b),q) \in \delta_{\psi'}$ or $\exists q_2 \in Q_\diamond, (q_1,q_2,(a,b),q) \in \delta_{\psi'}$.
- $(q_1,a,q) \in \delta_\psi$ iff $\exists b \in \Gamma\cup\{\diamond\}, (q_1,(a,b),q) \in \delta_{\psi'}$ or $\exists q_2 \in Q_\diamond, (q_1,q_2,(a,b),q) \in \delta_{\psi'}$.
- $(q_1,q_2,a,q) \in \delta_\psi$ iff $\exists b \in \Gamma\cup\{\diamond\}, (q_1,q_2,(a,b),q) \in \delta_{\psi'}$.

Let us show that the (non-deterministic) automaton A'_ψ accepts exactly the set of solutions of ψ. Assume that a tree t reaches a state q in A'_ψ (let us denote σ a run of t that reaches q in A'_ψ). We will show that there exists a tree $t' \in T_{\Gamma\cup\{\diamond\}}$ such that there exists a run σ' of $t\otimes t'$ in $A_{\psi'}$ that reaches q. This is proved by induction over the depth of t: if the depth of t is zero ($\mathrm{dom}(t)$ is empty), then t reaches a state in $Q_\diamond$, and by definition of $Q_\diamond$, there exists $t' \in T_\Gamma$ such that $t\otimes t'$ reaches q in $A_{\psi'}$. If $\mathrm{dom}(t)$ is not empty then $\varepsilon \in \mathrm{dom}(t)$, we need to distinguish the three following cases:

1. If t is a leaf (i.e. we have $\mathrm{dom}(t) = \{\varepsilon\}$), then there exists $q_1, q_2 \in Q_\diamond$, such that $(q_1,q_2,t(\varepsilon),q) \in \delta_\psi$. $(q_1,q_2,(t(\varepsilon),a),q) \in \delta_{\psi'}$ for some $a \in \Gamma\cup\{\diamond\}$, in which case $t' = a[t'_1,t'_2]$ (where t'_1 (resp. t'_2) is the inductive tree for the empty tree reaching q_1 (resp. q_2) in A'_ψ) is a tree such that $t\otimes t'$ reaches q in $A_{\psi'}$.
2. If $0 \in \mathrm{dom}(t)$ and $1 \notin \mathrm{dom}(t)$, then let $q_1 = \sigma(0)$ (we can easily extract from σ a run over the left successor of t, which reaches q_1 and hence by induction hypothesis there is a t'_1). Because σ is a valid run in A'_ψ, there exists $q_2 \in Q_\diamond$ such that $(q_1,q_2,t(\varepsilon),q) \in \delta_\psi$, hence there exists $a \in \Gamma\cup\{\diamond\}$, $(q_1,q_2,(t(\varepsilon),a),q \in \delta_{\psi'}$, let t'_2 the inductive tree for the empty tree reaching q_2 in A'_ψ, then $t' = a[t'_1,t'_2]$ is such that $t\otimes t'$ reaches q in $A_{\psi'}$.
3. If $\{0,1\} \subseteq \mathrm{dom}(t)$, then let $q_1 = \sigma(0)$ and $q_2 = \sigma(1)$, from σ we can extract a run over each of the left and right successors of t which reach respectively q_1 and q_2 in A'_ψ, hence the induction hypothesis provides us t'_1 and t'_2 for these two subtrees. σ is a valid run in A'_ψ, hence there exists $a \in \Gamma\cup\{\diamond\}$, $(q_1,q_2,(t(\varepsilon),a),q) \in \delta_{\psi'}$, clearly $t' = a[t'_1,t'_2]$ is such that $t\otimes t'$ reaches q in $A_{\psi'}$.

We now deduce that if a tree t reaches a final state (namely q) in a'_ψ, then t is a solution of $\exists y.\psi'(x,y)$. we can find t' such that $t \otimes t'$ reaches the final state q in $a_{\psi'}$, that means $t \otimes t'$ represents a solution of ψ', hence t represents a solution of ψ.

Assume a tree t represents a solution (x) of ψ, then there exists an element of the domain y such that (x,y) is a solution of ψ', hence the tree representing (x,y) reaches a final state in $a_{\psi'}$, it should be clear that from the run of that tree in $a_{\psi'}$, we deduce a run of the tree representing x in a'_ψ that reaches the same final state.

According to the induction hypothesis, all trees in $T_{\Gamma^{\otimes r+k+1}}$ that reach $\perp$ in A_{r+k+1} (that is trees that are not convolutions of $r+k+1$ trees in T_Γ) reach the same state in $A_{\psi'}$, let us denote q_s that state.

Let us show that for any trees $t_1, t_2 \in T_{\Gamma^{\otimes r+k}}$, if $t_1 E^{r+k}_{m-k} t_2$, then t_1 and t_2 reach the same set of states in A'_ψ.

If t_1 is empty, then by hypothesis t_2 also is, and they both reach exactly $Q_\diamond$.

If t_1 is not empty, and t_1 reaches q_s (in A'_ψ), then t_2 is not empty (because it is equivalent to a non empty tree). It should be clear that $t_2 \otimes \diamond[a]$ is a tree in $T_{\Gamma^{\otimes r+k+1}}$ (notice we need to have a non-empty tree otherwise the root would be labelled by $(\diamond,\ldots,\diamond)$) but is not a convolution of words in T_Γ hence it reaches q_s in $A_{\psi'}$ hence it reaches q_s in A'_ψ.

If t_1 reaches a state $q \neq q_s$ (in A'_ψ), then as it does not reach q_s it is a convolution of trees in T_Γ, and there exists a tree $t'_1 \in T_{\Gamma\cup\{\diamond\}}$ such that $t_1 \otimes t'_1$ reaches q in $A_{\psi'}$. As $q \neq q_s$, we have that $t_1 \otimes t'_1$ is a convolution of trees in T_Γ, so we can apply the back-and-forth hypothesis of the theorem and deduce the existence of a tree $t'_2 \in T_\Gamma$ such that $t_1 \otimes t'_1 E^{r+k+1}_{m-k-1} t_2 \otimes t'_2$, by the induction hypothesis, we deduce that $t_2 \otimes t'_2$ also reaches q in $A_{\psi'}$ from which we deduce that t_2 reaches q in A'_ψ.

Then we build A_ψ with an on-the-fly subset construction over A'_ψ. We just showed that any two E^{r+k}_{m-k} equivalent trees reached the same set of states in A'_ψ, so our on-the-fly construction will lead to an automaton with a number of states bounded by the index of E^{r+k}_{m-k}), which will have the property that two E^{r+k}_{m-k} equivalent trees reach the same state A_ψ. Also this construction will have time bounded by a fixed polynomial of $|\Gamma|^{r+k} f(m-k+r+k)$, since both automata have a number of states bounded by $f(m+r)$. □

3.2.3 Inspecting expressiveness of all states

Theorem 3.9 allows to check for some properties of a relation and establish an upper bound on the complexity of construction of automata for an automatic presentation, however it gives little intuition on how to exhibit such a relation. We present in this section a generic way to give such a relation that satisfies all the hypotheses of the theorem. Conspicuously we stated in Proposition 2.3 that the residuals of a language representing a Presburger set were respresenting Presburger sets, when using least-significant digit first base 2 representation; and this remark still holds when we use the most significant digit first representation.

This remark, we will see, not only ensures the regularity of Presburger sets, but also yields a 3EXPTIME bound on the inductive construction of the automaton for a formula. Relations accepted by each state in the automatic presentation of a structure play a central role in the complexity of the inductive construction of automata accepting first-order relations.

Definition 3.11. Given an automatic presentation AP over a signature $\mathcal{S}$, $AP = (\Gamma, A_D, (A_P)_{P\in\mathcal{S}})$ (such that the automata are all minimal), we denote AP_{sat} the structure over the domain T_Γ with predicates:

- P_ε (unary) that accepts the empty tree,
- $(P_D^q)_{q\in A_D}$ a family of (unary) predicates (one for each state in A_D). P_D^q accepts exactly the set of trees that reach the state q in A_D,
- $(R_P^q)_{P\in\mathcal{S},q\in Q_{A_P}}$ a family of predicates (one for each state in each automaton $(A_P)_{P\in\mathcal{S}}$). R_P^q is an ar_P-ary predicate that accepts exactly ar_P-tuples of trees whose convolution reach the state q in A_P.

We can remark that AP_{sat} is clearly automatic, with size atmost quadratic w.r.t. $|AP|$.

Theorem 3.12. *If there exists a function f such that for any m, r, the number of FO_m^r-undistinguishable r-tuples of trees in AP_{sat} is bounded by $f(m+r)$, then there exists a family of relations (E_m^r) that satisfies the hypotheses of Theorem 3.9 (over both AP and AP_{sat}), with index bounded by $f(m+r)+1$.*

Proof. The proof is in two steps. First we build E_m^r from the FO_m^r-undistinguishability relations and show that it satisfies the hypotheses of Theorem 3.9 for the automatic structure AP_{sat}, and then we show that this implies it also satisfies the hypotheses of Theorem 3.9 for the automatic structure AP.

E_m^r is defined as follows: uE_m^rv iff

- both u and v are not convolutions of trees in T_Γ
- or u and v are both convolutions of r trees in T_Γ and the tuple of trees u represents is FO_m^r-undistinguishable in AP_{sat} from the tuple of trees v represents.

It should be clear that the empty tree is always alone in its equivalence class (it is characterizable in AP_{sat} with a formula in FO_0^r), so hypothesis 1 holds.

By definition of AP_{sat} (its domain is T_Γ) and E_m^r (that relates only tuples of trees FO_m^r-undistinguishable), it should be clear it satisfies Hypothesis 2.

Hypothesis 3 relies on the fact that FO_m^r-undistinguishability is a back-and-forth relation. Though the back-and-forth property for a family of relations implies it refines FO_m^r-undistinguishability, the converse is not necessarily true. Assume (the r-tuple) u is FO_m^r-undistinguishable from v. Let $u' \in T_\Gamma$, and take $\varphi(x_1,\ldots,x_{r+1})$ the conjunction of all (semantically distinct) FO_{m-1}^{r+1} formulas which hold true for u, u', clearly $AP_{sat} \vDash \varphi(u,u')$ and by definition of φ, the existential quantification $\exists y.\varphi(u,y)$ only holds true for $y = u'$; as u and v are FO_m^r-undistinguishable, the existential quantification $\exists y.\varphi(v,y)$ holds true for some value of y, let us denote v' such a y, it should be clear that the construction of φ proves that (u,u') is FO_{m-1}^{r+1}-undistinguishable from (v,v') which concludes the proof of the back-and-forth property of FO_m^r.

By definition of the structure, we have that FO_0^r-undistinguishability is a congruence modulo contexting. This property will propagate for any m. We prove hypothesis 4 by induction over m. Assume that for some m, for any r, for any u, v, $uE_m^r v$ implies that for any $c \in C_{\Gamma^{\otimes r}}$, $c[u]E_m^r c[v]$, let us show it holds for $m+1$.

Assume $uE_{m+1}^r v$, we perform some case analysis over u and $c[u]$:

1. If $u \notin T_\Gamma^{\otimes r}$, then $v \notin T_\Gamma^{\otimes r}$, and for any $c \in C_{\Gamma^{\otimes r}}$ $c[u], c[v] \notin T_\Gamma^{\otimes r}$, hence $c[u]E_{m+1}^r c[v]$.

2. If $u \in T_\Gamma^{\otimes r}$ (then $v \in T_\Gamma^{\otimes r}$) and $c[u] \notin T_\Gamma^{\otimes r}$; let us show that u and v reach the same state in A_r, indeed, a tree t in $T_\Gamma^{\otimes r}$ (hence $t = t_1 \otimes \ldots \otimes t_r$, for some $t_1, \ldots, t_r \in T_\Gamma$) reaches a state $(q_1, \ldots, q_r)$ in A_r, where for any $i \in [1, r]$, $q_i = 1$ if t_i is not empty, and $q_i = 0$ if t_i is empty. Because AP_{sat} has an empty-tree predicate, this means that u and v beeing FO_{m+1}^r-undistinguishable implies that they have the same empty components, and hence reach the same state in A_r, so do $c[u]$ and $c[v]$, so if $c[u]$ is not in $T_\Gamma^{\otimes r}$, then neither is $c[v]$ so $c[u]E_{m+1}^r c[v]$.

3. Let $c \in C_{\Gamma^{\otimes r}}$ and denote l_c its ?-labeled node. If $u, c[u] \in T_\Gamma^{\otimes r}$ (then $v, c[v] \in T_\Gamma^{\otimes r}$), let us show that for any $t \in T_\Gamma$, we can find a $t' \in T_\Gamma$ such that $c[u] \otimes tE_m^{r+1} c[v] \otimes t'$.

   Assume first that $l_c \notin \mathrm{dom}(t)$ then let us show that $c[v] \otimes t$ is FO_m^{r+1}-undistinguishable from $c[u] \otimes t$. Because E_m^r is back-and-forth, we know that $u \otimes \eta E_m^{r+1} v \otimes \eta$ (where η denotes the empty tree over Γ), now let's build the context $c \otimes t$ as a convolution and we label node at position l_c by ?: as $l_c \notin \mathrm{dom}(t)$ we have that l_c is a leaf in $c \otimes t$. By induction, we deduce that $c[u] \otimes t = (c \otimes t)[u \otimes \eta]E_m^{r+1}(c \otimes t)[v \otimes \eta] = c[v] \otimes t$.

   Assume that $l_c \in \mathrm{dom}(t)$, let us denote $t_{\ngeq l}$ the subtree of t such that $\mathrm{dom}(t_{\ngeq l}) = \{w \in \mathrm{dom}(t) | w \text{ does not start with } l_c\}$, and $t_{\geq l} = \{w | l_c w \in \mathrm{dom}(t)\}$, $t_{\geq l}(w) = t(l_c w)$. By construction of $t_{\ngeq l}$ we can define similarly the context $c \otimes t_{\ngeq l}$ as a convolution and with its node l_c labeled by ? (and this node is guaranteed to be a leaf as $l_c \notin \mathrm{dom}(t_{\ngeq l})$) We have that $c[u] \otimes t = (c \otimes t_{\ngeq l})[u \otimes t_{\geq l}]$, since E_m^r is back-and-forth, we can find a t' such that $u \otimes t_{\geq l} E_m^{r+1} v \otimes t'$, from which we conclude that $c[u] \otimes tE_m^{r+1} c[v] \otimes (t_{\ngeq l}, l)[t']$.

By construction, E_m^r has one more equivalence class than FO_m^r-undistinguishability: which is $T_{\Gamma^{\otimes r}} \backslash T_\Gamma^{\otimes r}$. So the index of E_m^r is bounded by $1 + f(m + r)$.

So E_m^r satisfies the hypotheses of Theorem 3.9 for the automatic structure AP_{sat}, we now have to show that it also satisfies the hypotheses of the theorem for the automatic structure AP. It should be clear to the reader that we now have only to ensure Hypothesis 2, and we can deduce from the construction of AP_{sat}, that any tree that represents a tuple of trees in the domain of AP is only equivalent to trees that represent tuples of trees in the domain of AP (since this property is expressible by a quantifier free formula of AP_{sat}), and the $FO_m^r(AP)$-undistinguishability can be deduced from the fact that any formula of $FO_m^r(AP)$ can be rewritten as a formula in $FO_m^r(AP_{sat})$. □

3.3 Examples

We are now in a position to establish upper bounds for some automatic structures with elementary first-order theories.

3.3.1 (Tree-)Automatic structures of bounded degree

Automatic structures of bounded degree were introduced by Kuske and Lohrey [KL11], they devised a decision algorithm to show that their first-order theory could be decided in 2EXPSPACE. For string-automatic structures, their algorithm consists of presenting representatives of each FO^r_m-undistinguishability class within doubly exponential space (w.r.t. m and r), and ensuring properties about these representatives within 2EXSPACE as well. Model checking then reduces to non-deterministically or universally guessing those representatives to ensure the validity of a given formula.

They managed to establish a 2EXPSPACE lower bound, by exhibiting an automatic structure of bounded degree that is essentially the graph of 2^m computations of a 2EXPSPACE Turing machine and provided a formula that could check whether this machine would accept on a given input.

Their results extended nicely to tree-automatic structures, over which they could also show the 3EXPTIME-completeness of their first-order. The higher upper bound is due to the fact that checking emptiness of a tree automaton is PTIME whereas it is only NLOGSPACE for a string automaton. The lower bound could also be increased as they managed to encode alternating Turing machine.

In the following we address the question of what is the complexity of this inductive construction and will establish that not only automata are of size triply exponential (w.r.t the size of the formula) but also that they can be built within 3EXPTIME, thus establishing an upper bound on the generic algorithm that closely matches the complexity of this theory.

Definition 3.13. The **Gaifman Graph** of a structure $\mathcal{A} = (D, (P)_{P\in\mathcal{S}})$ is a graph whose set of nodes is D, and there exists an edge between $a, b \in D$, iff there exists $P \in \mathcal{S}$, $x_1, \dots, x_{ar_P} \in D$ such that $P(x_1, \dots, x_{ar_P})$ holds and $x_i = a$ and $x_j = b$ for some $i, j \in [1, ar_P]$.

Definition 3.14. A structure has **bounded degree** iff its Gaifman Graph has bounded degree, i.e. there exists an integer δ such that every element in the domain can be in relation with at most δ other elements.

Theorem 3.15. *Let us consider a (deterministic) automatic structure AP that has bounded degree δ, denote a the maximal arity appearing in AP. The number of $FO^r_m(AP_{sat})$-undistinguishable elements is bounded by:*

$$r^r(|AP|a^2r\delta)^{3^m}2^{|AP|(|AP|a^2r\delta)^{a3^m}}$$

We can assume without loss of generality that all the automata in AP are minimal. Let us consider the (automatic) structure $\mathcal{B}$ which is the same as AP_{sat} except it doesn't have the predicates (R^s_P) (for each $P \in \mathcal{S}$ and s is the sink state of A_P). As each (R^s_P) can be expressed as a quantifier free formula in $\mathcal{B}$, ($R^s_P \equiv \neg \bigvee_{q\in A_P, q\neq s} R^q_P$),

$FO^r_m(\mathcal{B})$-undistinguishability is equivalent to $FO^r_m(AP_{sat})$-undistinguishability. We will now show the bound on the index of $FO^r_m(\mathcal{B})$-undistinguishability.

Lemma 3.16. *If AP has bounded degree δ, then $\mathcal{B}$ has degree bounded by $a^2|AP|\delta$*

Proof. Assume by contradiction that $t \in T_\Gamma$ is in relation (in $\mathcal{B}$) with $1 + a^2|AP|\delta$ distinct trees. Then one of the $\sum_{P\in\mathcal{S}}(|Q_{A_P}|-1)$ relations defined from the $(A_P)_{P\in\mathcal{S}}$, namely R^q_P, lets t be in relation with more than δa^2 distinct trees, and we can find $j, j'(j \neq j')$ such that there are at least $1 + \delta$ (as $a \geq ar_P$) distinct trees in T_Γ (namely $t_1, \ldots, t_{\delta+1}$) such that R^q_P holds with such a tree in the j'-th position and t in the j-th position.

So there are at least $\delta + 1$ trees $(u_k)_{k\in[1,\delta+1]}$, in $T_\Gamma^{\otimes ar_P}$, such that for each k, u_k is a convolution of trees in T_Γ who reaches the state q in A_P, whose j-th component is t, and whose j'-th component is t_k. Then, as q is not a sink state, there exists a context $c \in C_{\Gamma^{\otimes ar_P}}$, such that $c[t']$ reaches a final state in A_P for each tree t' that reaches the state q in A_P.

As each $c[u_k]$ reaches a final state in A_P, each $c[u_k]$ is a convolution of trees that represents elements of the domain of $\mathcal{A}$. Furthermore the j-th tree is these convolutions is always the same, and by definition of the t_k, the j'-th are pairwise distinct (and also all distinct from the j-th component), hence the element of the domain of $\mathcal{A}$ represented by the j-th component is in relation with $\delta + 1$ different elements of the domain in $\mathcal{A}$, which contradicts δ to be the degree of the Gaifman graph of $\mathcal{A}$.

Therefore $\mathcal{B}$ also has bounded degree and its degree is bounded by $a^2|AP|\delta$. □

In order to establish a bound on the number of FO^r_m-undistinguishable elements, we need the following definition:

Definition 3.17. With the Gaifman graph, we can define a distance over elements of the structure, which is the distance in the Gaifman graph between the two elements: the length of the smallest path linking the two elements, or infinity if there exists no such path.

A neighborhood of radius k around a tuple of elements of the structure is defined as the set of elements which are at distance smaller or equal to k to one of the elements of the tuple.

Neighborhoods of radius k around r-tuples $(x_1, \ldots, x_r)$ and $(y_1, \ldots, y_r)$ are said to be isomorphic if there exists a bijection ξ between the two neighborhoods, such that $\xi(x_i) = y_i$ and such that for any $P \in \mathcal{S}$ and any $u_1, \ldots, u_{ar_P}$ in the neighborhood of radius k around $(x_1, \ldots, x_r)$ we have $P(u_1, \ldots, u_{ar_P})$ holds iff $P(\xi(u_1), \ldots, \xi(u_{ar_P}))$ holds.

We now establish that only the exponential neighborhood around r-tuples determine the formulas they validate.

Lemma 3.18. *Two tuples are $FO^r_m(\mathcal{B})$-undistinguishable if their neighbourhoods of radius 3^m are isomorphic.*

This result was first established by Gaifman [Gai82] with neighborhoods of radius 7^m, he also established conversely that neighborhoods of radius 2^{m-2} had to be considered. This result was then improved by Keisler and Lotfallah [KL04] to only considering neighborhoods of radius 4^m. We further improved this result, in the special case of structures of bounded degree in [DGH12] to 3^m. The proof that we present here crucially relies on the fact that the degree of the structure is bounded (hence all neighborhoods of finite radius are finite).

Proof. This can be simply shown by an Ehrenfeucht-Fraïssé game, provided we consider only neighborhood of radius $\frac{3^m-1}{2}$. We therefore prove this by induction over m. It is true when $m = 0$ by definition of isomorphism (in the special case of radius 0). If for some $m > 1$ and any r we have for any $u_1, v_1, \dots, u_r, v_r \in \Sigma^*$ such that $(u_1, \dots u_r)$ and $(v_1, \dots, v_r)$ have neighborhoods of radius $\frac{3^m-1}{2}$ isomorphic, for any $u_{r+1} \in \Sigma^*$ the existence of a word v_{r+1} such that the neighborhoods of radius $\frac{3^{m-1}-1}{2}$ around $(u_1, \dots, u_{r+1})$ and $(v_1, \dots, v_{r+1})$ are isomorphic, then we have the $FO^r_m(\mathcal{B})$-undistinguishability of tuples with $\frac{3^m-1}{2}$-isomorphic neighborhood, and as $\frac{3^m-1}{2} \leq 3^m$, the lemma holds.

We will need to distinguish three cases, informally:

1. If u_{r+1} is close to $(u_1, \dots, u_r)$, we will pick the corresponding v_{r+1} in the neighborhood of $(v_1, \dots, v_r)$
2. If u_{r+1} is far from both $(u_1, \dots, u_r)$ and $(v_1, \dots, v_r)$, we may also pick v_{r+1}
3. If u_{r+1} is far from $(u_1, \dots, u_r)$ and close to $(v_1, \dots, v_r)$, we will show how to find in the neighborhood of $(u_1, \dots, u_r)$, a corresponding element, that is also far from $(v_1, \dots, v_r)$.

We now formalize these remoteness and closeness concepts, let us denote explicitly ξ_m the morphism from the $\frac{3^m-1}{2}$-neighborhood around $(u_1, \dots, u_r)$ to the $\frac{3^m-1}{2}$-neighborhood around $(v_1, \dots, v_r)$ that maps each u_i to each v_i, and let us build ξ_{m-1}.

1. If $\min_{i \leq r} d_{\mathcal{A}(AP)_{sat}}(u_{r+1}, u_i) \leq 3^{m-1}$.

   We choose $v_{r+1} = \xi_m(u_{r+1})$. Notice that the $\frac{3^{m-1}-1}{2}$ neighborhood around u_{r+1} is inside the $\frac{3^m-1}{2}$ neighborhood around $(u_1, \dots, u_r)$, hence this remark conversely holds for v_{r+1}. ξ_{m-1} is thus a restriction of ξ_m, and the isomorphism of the $\frac{3^{m-1}-1}{2}$-neighborhoods holds.

2. If $\min_{i \leq r} d_{\mathcal{A}(AP)_{sat}}(u_{r+1}, u_i) > 3^{m-1}$ and $\min_{i \leq r} d_{\mathcal{A}(AP)_{sat}}(u_{r+1}, v_i) > 3^{m-1} + 1$, we take $v_{r+1} = u_{r+1}$.

   As the $\frac{3^{m-1}-1}{2}$ neighborhoods around $(u_1, \dots, u_r)$, $(v_1, \dots, v_r)$ are both not only disjoint from the $\frac{3^{m-1}-1}{2}$-neighborhood around u_{r+1}, but also no edge connects either of those two neighborhoods with the $\frac{3^{m-1}-1}{2}$-neighborhood around u_{r+1}, thus we can build ξ_{m-1} as the restriction of ξ_m over the $\frac{3^{m-1}-1}{2}$-neighborhood around $(u_1, \dots, u_r)$ and as the identity over the $\frac{3^{m-1}-1}{2}$ neighborhood around u_{r+1}. The absence of connecting edges, ensures that this is an isomorphism.

3. Finally, if $\min_{i\leq r} d_{\mathcal{A}(AP)_{sat}}(u_{r+1}, u_i) > 3^{m-1}$ and $\min_{i\leq r} d_{\mathcal{A}(AP)_{sat}}(u_{r+1}, v_i) \leq 3^{m-1}$.

   As u_{r+1} is in the domain of ξ_m^{-1}, so we can define $v^{(1)} = \xi_m^{-1}(u_{r+1})$. While $\min_{i\leq r} d_{\mathcal{A}(AP)_{sat}}(v^{(n)}, v_i) \leq 3^{m-1}$, (thus $v^{(n)}$ is in the domain of ξ_m^{-1}) we define $v^{(n+1)} = \xi_m^{-1}(v^{(n)})$. Let us show that there is always an n such that $\min_{i\leq r} d_{\mathcal{A}(AP)_{sat}}(v^{(n)}, v_i) > 3^{m-1}$. First we show that for any n, $u_{r+1} \neq v^{(n)}$. As there is an i such that $d_{\mathcal{A}(AP)_{sat}}(v^{(n)}, v_i) \leq 3^{m-1}$ and $v^{(n)}$, v_i and the shortest path from v_i to $v^{(n)}$ are in the domain of ξ_m^{-1}, we have $d_{\mathcal{A}(AP)_{sat}}(v^{(n+1)}, u_i) \leq 3^{m-1}$, which is not the case for u_{r+1}, thus clearly $u_{r+1} \neq v^{(n)}$. As ξ_m^{-1} is a bijection, it should be clear that all the $v^{(n)}$ are pairwise distinct. Since spheres of finite radius are finite, $v^{(n)}$ can only take finitely many values.

   Thus $(v^{(n)})_n$ is a finite sequence, let us denote n_c the greatest n such that $v^{(n)}$ is defined, we take $v_{r+1} = v^{(n_c)}$. By definition $\min_{i\leq r} d_{\mathcal{A}(AP)_{sat}}(v_{r+1}, v_i) > 3^{m-1}$, so the $\frac{3^{m-1}-1}{2}$ neighborhood around $(v_1, \ldots, v_r)$ and the $\frac{3^{m-1}-1}{2}$ neighborhood around v_{r+1} not only are disjoint but are also not connected by any edge. The same holds for $(u_1, \ldots, u_r)$ and u_{r+1}.

   We can furthermore show that $\xi_m^{-n_c}$ is defined on the $\frac{3^{m-1}-1}{2}$ neighborhood around u_{r+1}, thus, we can define ξ_{m-1} as ξ_m on the $\frac{3^{m-1}-1}{2}$ neighborhood around $(u_1, \ldots, u_r)$ and as $\xi_m^{-n_c}$ (which is still an isomorphism, as it is preserved by composition) on the $\frac{3^{m-1}-1}{2}$ neighborhood around u_{r+1}. As those two and their images do not touch, we have that ξ_{m-1} is an isomorphism.

□

Lemma 3.19. *The number of non-isomorphic neighbourhood of radius* 3^m *is bounded by* $r^r(|AP|a^2r\delta)^{3^m}2^{|AP|(|AP|a^2r\delta)^{a3^m}}$

Proof. This lemma is shown by a counting argument: for two tuples u and v, they are not equivalent if equality on u and v differ, or they don't have the same number of elements in their 3^m neighbourhood, or the neighbourhoods around u and v despite them having the same number of elements are not isomorphic.

There are as many behaviours of equality over u as there are partitions of $[1, r]$ which we bound by r^r.

The structure $\mathcal{B}$ has a degree bounded by $|AP|a^2\delta$, hence the number of elements in the neighbourhood of radius 3^m around u is bounded by $\sum_{i=1}^{3^m} r(|AP|a^2\delta)^i$.

The number of non-isomorphic sequences of k elements is bounded by $\prod_{P\in\mathcal{B}} 2^{k^{ar_P}} \leq 2^{|AP|k^a}$.

Therefore the number of non-isomorphic neighbourhood of radius 3^m around r-tuples is bounded by $r^r \sum_{i=1}^{3^m} r(|AP|a^2\delta)^i 2^{|AP|i^a} \leq r^r(|AP|a^2r\delta)^{3^m}2^{|AP|(|AP|a^2r\delta)^{a3^m}}$.
□

Therefore whatever the (deterministic) automatic presentation AP (with all automata minimal), the number of $FO_m^r(AP_{sat})$-undistinguishable elements are bounded by $r^r(|AP|a^2r\delta)^{3^m}2^{|AP|(|AP|a^2r\delta)^{a3^m}}$.

As delta here is a parameter, we deduce that the inductive construction of an automaton accepting solutions of a first-order formula $\varphi \in FO(\mathcal{A})$ is 3EXPTIME.

As a corrolary, we will now treat the uniform cases. The problem of uniform model-checking of automatic structures of bounded degree consists of taking as input both an automatic structure and a first-order formula and decide whether that formula holds in the structure. We will consider independantly string- and tree-automatic structures, with an injective presentation or not, with a presentation with deterministic or non-deterministic automaton. The major difficulty will reside in that we have to establish an upper bound on δ and on the size of the injective deterministic automatic presentation, w.r.t. the size of the input presentation.

Corollary 3.20. *Bounds on the uniform model-checking problem*

- *The uniform model-checking of string-automatic structures of bounded degree is 2EXPSPACE-hard and in 3EXPTIME.*
- *The uniform model-checking of injective tree-automatic structures of bounded degree is 3EXPTIME-complete.*
- *The uniform model-checking of non-injective tree-automatic structures of bounded degree in in 4EXPTIME.*

Proof. The complexity lower bounds were achieved by Kuske and Lohrey [KL11].

Let us first prove the claim over string-automatic structures. We consider as input a formula ψ and an automatic presentation P. Notice that P can be non-injective, and can be presented with non-deterministic automata. We will first reduce this case to an injective deterministic automatic structure P'. The automaton accepting the domain will be denoted A'_D, and we use the standard procedure to obtain an injective automatic structure, by giving as domain, the minimal representative (in the lexicographical order) as injective coding of an element. A'_D can be build from the formula $\varphi_D(x) = x \in D \wedge \neg\exists y \cdot y \equiv x \wedge y <_l x$. As existential quantification can be performed over a non-deterministic automaton resulting in an exponential deterministic automaton, A'_D is built as the product of the determinized A_D with the automaton representing the right conjunct. Both automata are exponential w.r.t. the size of the presentation P, so is A'_D. Similary the automata for relations, A'_R, can be obtained from the formula $\varphi_R(x_1, \dots, x_r) = R(x_1, \dots, x_r) \wedge \bigwedge_{i=1}^{r} \neg\exists y \cdot y \equiv x_i \wedge y <_l x_i$, this automaton is carefully built as a product of the deterministic automaton for each term of the conjunction, each being at most exonential w.r.t. $|P|$, (and as the arity is smaller than $|P|$), we have that A'_R is atmost exponential w.r.t. P.

We next exploit the result on synchronous transducers [Web90] that states that if the degree is bounded, it is bounded by an exponential of the size of the automaton, to deduce a doubly exponential (w.r.t. the size of P') upper bound on the degree of P'.

Theorem 3.15, together with the aformentioned bounds for the degree and the size of the (determinized injectived) automatic structure, yields a triply exponential time upper bound depending on the size of P and the length of ψ to build a deterministic automaton accepting the smallest representatives (in the lexicographic order) of its set of solutions. We can easily build from that automaton a non-deterministic automaton accepting all representatives of all solutions of the formula ψ.

This result improves the 3EXPSPACE upper bound established in [KL11] by Kuske and Lohrey, for the uniform model-checking of non injective string-automatic structures of bounded degree, but is not as tight as the 2EXPSPACE upper bound they obtained for injective string-automatic structures of bounded degree.

In the case of injective tree-automatic structures of bounded degree, we close the gap between the 3EXPSPACE upper bound and the 3EXPTIME lower bound [KL11]. For that we just rely on the known doubly exponential upper bound on the degree of a tree-automatic structure of bounded degree [Sei92] (the bound on the degree holds even in the case where the presentation is non-deterministic). As in the bound given in Theorem 3.15, the degree (and the size of the deterministic presentation) only appears exponentially, we can safely determinize and minimize the automata in the presentation (which does not modify the degree), and obtain a triply exponential upper bound on the time of construction.

The construction of an injective equivalent automatic structure, potentially yields an exponential size increase of the size of the presentation, which only allows us to give a triply exponential upper bound on the degree, which results in a quadruple exponential upper bound w.r.t. the size of the presentation for the uniform model checking of non-injective tree-automatic structures of bounded degree. This however improves the results from [KL11] where only a 5EXPTIME upper bound could be derived, hence narrowing the gap w.r.t. the 3EXPTIME lower bound. □

3.3.2 Some decidable arithmetics

Presburger Arithmetic with most significant digit first encoding

We showed in section 2, that building the automaton accepting solutions of a Presburger formula, using the least significant digit first representation can be done in 3EXPTIME, we now show that using the converse coding, most-significant digit first yields also a construction in 3EXPTIME.

Note that we work here in the framework of automatic structures, as opposed to section 2 where no padding symbol was required, as padding was performed by repeating the most significant bit. Sparing the padding symbol, reduces the size of the alphabet, and also the sizes of the automata. Indeed the automata accepting respectively addition and order have respectively three and two states.

The upper bound on the size of the automaton using most siginificant digit first was established by Eisinger [Eis08] using Ehrenfeucht-Fraïssé relations, and essentially by checking hypotheses of Theorem 3.9, however he did not establish the bound on the time of construction, and only a 4EXPTIME upper bound could be derived.

An automatic presentation for Presburger Arithmetic We detail here an automatic presentation of Presburger Arithmetic $(\mathbb{N}, <_{/2}, +_{/3})$, which we call $AP_{PA} = (\{0,1\}, A_D, A_<, A_+)$. We use a most significant digit first base 2 notation, that is a word w represents the integer $\sum_{i=1}^{|w|} 2^{i-1} w(i)$. To have an injective representation of $\mathbb{N}$, we impose that words do not start with a 0. $L(A_D) = \varepsilon + 1\{0,1\}^*$. This automatic presentation is exhibited in Figure 3.2.

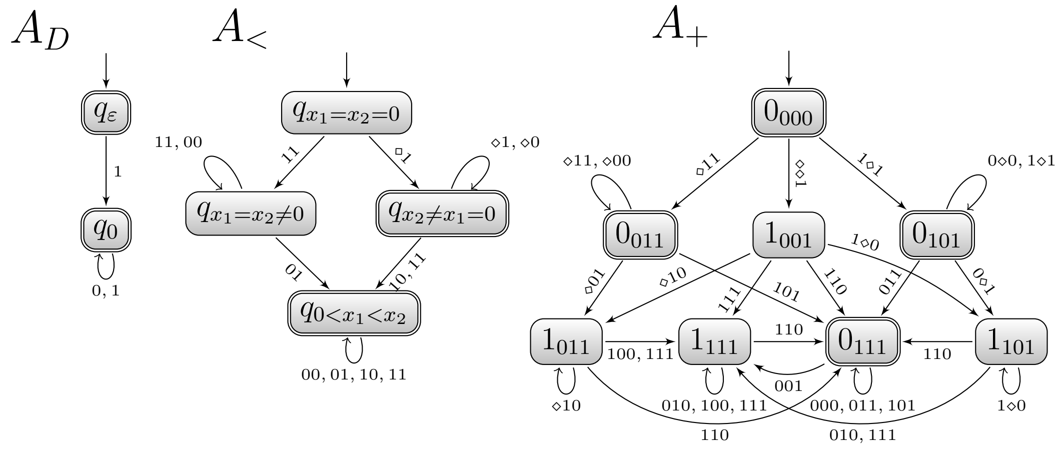


Figure 3.2: An automatic presentation of Presburger Arithmetic

””	$\to 0$		”0”	$\to 2\times$ ”1” $-1=1$
”1”	$\to 2$	(2×1)	”00”	$\to 2\times$ ”11” $-1=5$
”10”	$\to 4$	(2×2)	”01”	$\to 2\times$ ”10” $-1=3$
”11”	$\to 6$	(2×3)	”000”	$\to 2\times$ ”111” $-1=13$
”100”	$\to 8$	(2×4)	”001”	$\to 2\times$ ”110” $-1=11$

Figure 3.3: The isomorphism between $\{0,1\}^*$ and $\mathbb{N}$ to show the FO-interpretability of $AP_{PA,sat}$ as Presburger Arithmetic.

Lemma 3.21. *$AP_{PA,sat}$ is FO-interpretable as Presburger Arithmetic.*

Proof. The proof will be as follows: First we give an explicit one-to-one mapping between the elements of the two structures. Then we give a Presburger formula for each predicates of $AP_{PA,sat}$ (with a number of free variables equal the arity of that predicate) such that a tuple validates the formula iff the corresponding tuple of $AP_{PA,sat}$ validates the predicate.

We can essentially distinguish two kinds of elements in the domain of $AP_{PA,sat}$: words that start with a 0, and words in the domain of AP_{PA} (that is ε and words that start with a 1).

We canonically associate words in the domain of AP_{PA} to even integers: ε is mapped to 0, $1w$ is mapped to $2\times\left(2^{|w|}+\Sigma_{i=1}^{|w|}2^{i-1}w(i)\right)$. We associate words starting with 1 to odd integers as the predecessor of the double of its bitwise inverse (most-significant digit first). This defines a one-to-one mapping between words and integers (shown in Figure 3.3).

We now show that the predicates of $AP_{PA,sat}$ are all FO-interpretable in PA, we rely on two properties of the automatic presentation PA: first the domain of AP_{PA} is prefix closed, hence all words that are not in the domain of AP_{PA} reach the sink state in A_D, therefore words that do not reach the sink state in $A_<$ or A_+ are convolutions of words interpreted as even integers. Furthermore, it is a well-known fact that when using a most-significant bit first representation of integers, solutions of a Presburger formula form a regular language, and in the minimal deterministic automaton for that formula, the set of words that reach a given (non-sink) state is the set of solutions of a Presburger formula. We just have to ensure that we can interpret PA formulas over integers as PA formulas over the double of those integers which is the case as PA allow us to express evenness (which will allow to restrict quantifications) and with the fact that the interpretation of addition, order and equality are the same. □

Therefore we can deduce the FO-interpretability of $AP_{PA,sat}$ as Presbureger Arithmetic, from which we deduce that there exists an integer k (this integer corresponds to the largest quantifier depth in the PA interpretation of $AP_{PA,sat}$), such that the number of $FO^r_m(AP_{PA,sat})$-undistinguishable elements is bounded by the number of $FO^r_{m+k}(PA)$-undistinguishable elements, which is bounded by a triple exponential in $m+r$ [FR79].

An automatic presentation of Skolem Arithmetic

Skolem arithmetic is the first order theory over integers with multiplication. Its decidability relies on the fact that it is a Feferman-Vaught weak power of countably many times Presburger Arithmetic because of the uniqueness of the prime decomposition of integers. Each integer of Skolem Arithmetic is seen as the set of multiplicity of all prime numbers smaller than its greatest prime divisor. Multiplying two integers then only consists in adding these multiplicities.

The tree automaticity of Skolem Arithmetic hence relies on the string automaticity of Presburger Arithmetic, an integer of Skolem Arithmetic is represented as a finite comb, its i-th tooth corresponding to the base 2 representation of the multiplicity of the i-th prime number. Then three trees are in relation with $\times$ if either of the operands is 0 and the result is 0 or each i-th teeth (seen as words) are in relation for $+$. An example is given in figure 3.4.

We detail explicitely this idea to give a tree automatic presentation of Skolem Arithmetic: the set of trees that represent the domain are the empty tree (it represents 0) or and trees whose domain contains only words of the form $0^* \cup 0^*(10^*)$, more precisely 1 is represented by the tree whose domain is $\{\varepsilon\}$ and the root is labelled by 0; an integer $n \geq 2$ whose greatest prime divisor is p_i (p_i is the i-th prime number), is represented by a tree t_n whose leaves are $\{0^i\} \cup \{0^{j-1}10^{\lfloor \log_2(1+m_j)\rfloor} | j \leq i\}$ (m_j denotes the multiplicity of p_j in n), that is $\mathrm{dom}(t_n) = \{0^j | j \leq i\} \cup \{0^{j-1}10^{k_j} | j \leq i, k_j \leq \lfloor \log_2(1+m_j)\rfloor\}$; $t_n(0^j) = 0$ for all $j \leq i$, and the $t_n(0^{j-1}10^k)$ are defined uniquely such that for all $j \leq i$, $m_j = \sum_{k=0}^{\lfloor \log_2(1+m_j)\rfloor} 2^k.t_n(0^{j-1}10^k)$.

We give the minimal automaton accepting this regular language of trees: $Q_D = \{q_i, q_1, q_{Pres}, q_{Sko}\}, I_D = \{q_i\}, F_D = \{q_i, q_1, q_{Sko}\}$,
$\Delta_D = \{(0, q_i, q_i, q_1), (1, q_i, q_i, q_{Pres}), (1, q_{Pres}, q_i, q_{Pres}), (0, q_{Pres}, q_i, q_{Pres}),$
$(0, q_1, q_{Pres}, q_{Sko}), (0, q_{Sko}, q_i, q_{Sko}), (0, q_{Sko}, q_{Pres}, q_{Sko})\}$

The tree automaton accepting multiplication is defined from A_+ in Figure 3.2, as it has to ensure that all the pins of the combs are in relation for addition. $Q_\times = Q_+ \cup \{q_{111}, q_{100}, q_{010}, q_{101}, q_{011}, q_{SSS}, q_{S0S}, q_{0SS}, q_{S1S}, q_{1SS}, q_{P00}, q_{0P0}\}$,
$I_\times = \{0_{000}\}, F_\times = \{q_{111}, q_{100}, q_{010}, q_{SSS}, q_{S1S}, q_{1SS}\}$,
$\Delta_\times = \Delta_+ \cup \{(000, 0_{000}, 0_{000}, q_{111}), (0\diamond0, 0_{000}, 0_{000}, q_{101}), (\diamond00, 0_{000}, 0_{000}, q_{011}),$
$(0\diamond0, q_{101}, 0_{101}, q_{S0S}), (0\diamond0, q_{S0S}, 0_{101}, q_{S0S}), (0\diamond0, q_{S0S}, 0_{000}, q_{S0S}),$
$(\diamond00, q_{011}, 0_{011}, q_{0SS}), (\diamond00, q_{0SS}, 0_{011}, q_{0SS}), (\diamond00, q_{0SS}, 0_{000}, q_{0SS}),$

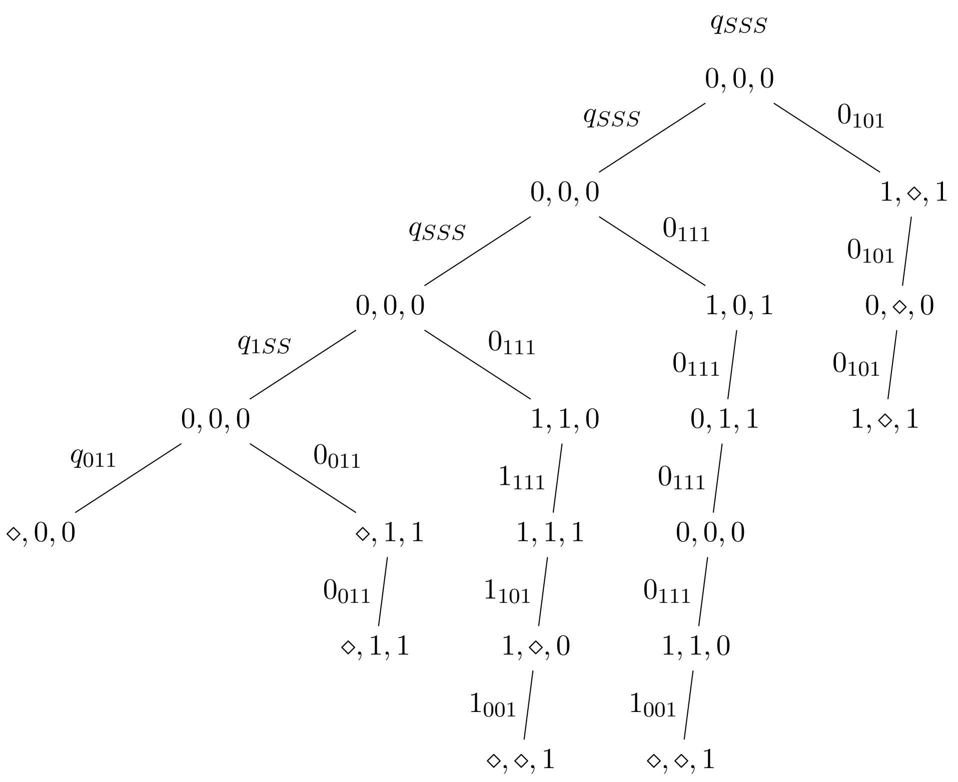


Figure 3.4: Example of multiplication: $2^5 \times 3^9 \times 5^7 = 49207500000$ times $3^{10} \times 5^3 \times 7^3 = 2531725875$ equals $2^5 \times 3^{19} \times 5^{10} \times 7^3 = 124579900994062500000$.

$(000, q_{101}, 0_{101}, q_{S1S}), (000, q_{S1S}, 0_{111}, q_{SSS}), (000, q_{S1S}, 0_{011}, q_{SSS}),$
$(000, q_{011}, 0_{011}, q_{1SS}, (000, q_{1SS}, 0_{111}, q_{SSS}), (000, q_{1SS}, 0_{101}, q_{SSS}),$
$(000, q_{SSS}, 0_{000}, q_{SSS}), (000, q_{SSS}, 0_{101}, q_{SSS}), (000, q_{SSS}, 0_{011}, q_{SSS}), (000, q_{SSS}, 0_{111}, q_{SSS}),$
$(1\diamond\diamond, 0_{000}, 0_{000}, q_{P00}), (0\diamond\diamond, q_{P00}, 0_{000}, q_{P00}), (1\diamond\diamond, q_{P00}, 0_{000}, q_{P00}),$
$(0\diamond\diamond, 0_{000}, 0_{000}, q_{100}), (0\diamond\diamond, q_{100}, q_{P00}, q_{S00}), (0\diamond\diamond, q_{S00}, 0_{000}, q_{S00}), (0\diamond\diamond, q_{S00}, q_{P00}, q_{S00}),$
$(\diamond 1\diamond, 0_{000}, 0_{000}, q_{0P0}), (\diamond 0\diamond, q_{P00}, 0_{000}, q_{0P0}), (\diamond 1\diamond, q_{0P0}, 0_{000}, q_{0P0}),$
$(\diamond 0\diamond, 0_{000}, 0_{000}, q_{010}), (\diamond 0\diamond, q_{010}, q_{0P0}, q_{0S0}), (\diamond 0\diamond, q_{0S0}, 0_{000}, q_{0S0}), (\diamond 0\diamond, q_{0S0}, q_{0P0}, q_{0S0})\}$

Lemma 3.22. *$AP_{SA,sat}$ is FO-interpretable as the sum of $(\mathbb{N}^*, \times)$ (i.e. Skolem Arithmetic over the strictly positive integers), $AP_{PA,sat}$ and the structure whose domain is $\mathbb{N}$ with a tautologic monadic predicate.*

Proof. First we detail an isomorphism between the domains of the two structures. Words, (i.e. trees whose domain are a possibly empty subset of 0^*) are seen as their corresponding element in $AP_{PA,sat}$, non-empty trees that are in the domain of AP_{SA}, are seen as their corresponding (non-zero) integer, and the rest of trees (which is infinite and countable, hence isomorphic to $\mathbb{N}$) correspond to elements of the trivial structure.

We now detail how we express the predicates in $AP_{SA,sat}$ in this sum structure. Clearly the predicates for states in Q_+ are trivially interpreted as their corresponding predicate in $AP_{PA,sat}$, and the predicate for each $q_{m_1 m_2 m_3}$ (for $m_1, m_2, m_3 \in \{0, 1, S, P\}$) are interpreted as follows:

- if $m_3 = 0$, and $m_2 = 0$ in which case the predicate corresponding to that state is interpreted as $m_3 = 0 \wedge m_2 = 0 \wedge \varphi$ where φ is $x_1 \in AP_{PA,sat}$ if m_1 is P, or φ is $x_1 \in (\mathbb{N}^*, \times) \wedge \times(x_1, x_1, x_1)$ if m_1 is 1, or φ is $x_1 \in (\mathbb{N}^*, \times) \wedge \neg \times (x_1, x_1, x_1)$ if m_1 is S. The case where $m_3 = 0$ and $m_1 = 0$ is defined analogously.

- if $m_3 \neq 0$ and $m_1 = 0$ (which implies $m_2 \notin \{0, P\}$ and $m_2 = m_3$), then the predicate corresponding to that state is interpreted as $x_3, x_2 \in (\mathbb{N}^*, \times) \wedge x_3 = x_2 \wedge \varphi$ with φ being $\times(x_3, x_3, x_3)$ if m_3 is 1 or φ being $\neg \times (x_3, x_3, x_3)$ if m_3 is S. The case where $m_3 \neq 0$ and $m_2 = 0$ is analogous.

- if none of m_1, m_2, m_3 is 0, then $m_1, m_2, m_3 \in \{1, S\}$, the predicate is interpreted as $m_1, m_2, m_3 \in (\mathbb{N}^*, \times) \wedge \bigwedge_{i=1}^{3} \varphi_i$ with φ being $\times(x_i, x_i, x_i)$ if m_i is 1, or $\neg \times (x_i, x_i, x_i)$ if m_i is S.

□

Corollary 3.23. *The inductive construction of an automaton accepting solutions of a Skolem Formula, is performed in 4EXPTIME.*

We recall that the theory of Skolem arithmetic is 3EXPSPACE, which was shown by Ferrante and Rackoff [FR79] by exhibiting a quadrifold exponential number of FO^r_m-undistinguishability equivalence classes. Using this result, together with the fact that $AP_{SA,sat}$ can be interpreted as a sum of three structures whose hardest is Skolem Arithmetic, we deduce that this sum structure has only a quadrifold exponential number of FO^r_m-undistinguishability equivalence classes. Therefore the inductive construction is performed in 4EXPTIME, which matches closely the 3EXPSPACE known upper bound.

3.3.3 Nested pushdown trees

Nested pushdown trees are the unfoldings of the configuration graphs of pushdown systems with an added jump relation which connects every push with the corresponding pop operation. Kartzow showed that the first-order model checking on nested pushdown trees is complete for doubly exponential time with linearly many alternation [Kar09] (the input being the nested pushdown system). The proof of upper bound relies on the existence of a small solution (i.e. a run of size doubly exponential) for any formula.

Definition 3.24. A *pushdown system* is a 5-tuples $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, (q_0, \bot))$ with a finite set of states Q, a transition alphabet Σ, a finite set of stack symbols Γ containing $\bot$, an initial configuration $(q_0, \bot) \in Q \times \Gamma$ and a transition function $\delta : Q \times \Sigma \times \Gamma \to Q \times (\{\mathbf{pop}, \mathbf{id}\} \cup \{\mathbf{push}_\gamma \mid \gamma \in \Gamma \backslash \{\bot\}\})$.

Definition 3.25. Let $\mathcal{P}$ a pushdown system. The *nested tree* generated by $\mathcal{P}$ is the structure $NPT(\mathcal{P}) = (R, (\xrightarrow{\alpha})_{\alpha \in \Sigma}, \hookrightarrow)$ defined as follows:

- R is the set of runs starting in $(q_0, \bot)$,

- The binary predicate $\xrightarrow{\alpha}$ connects a run ρ_1 with a run ρ_2 if ρ_2 extends ρ_1 by the transition α, and

- the *jump-relation* $\hookrightarrow$ connects $\rho_1 \in R$ with $\rho_2 \in R$ if ρ_2 extends ρ_1 with a run that starts with a $\mathbf{push}_\gamma$ for some γ, and such that exactly the last configuration of that run has the same stack as the last configuration in ρ_1.

Any pushdown nested tree is tree automatic. Furthermore the size of the automatic presentation is polynomial w.r.t. the pushdown system. The next subsection is dedicated to presenting in detail an automatic presentation, so that in the subsection after that we can give a precise description of the correponding saturated structure. Finally I will show how to use Kartzow's results to apply Theorem 3.9 to deduce a triply exponential bound on the size and time of construction of the automaton accepting solutions of a first-order formula.

Explicit automatic presentation

The run of a pushdown system $\mathcal{P}$ over a word $w \in \Sigma^*$ can be naturally encoded as a binary tree t over Σ, whose depth first traversal is w and a node has a right successor if it induced a push, the right successor is the node who lead to the corresponding pop.

However, I will use a representations that will embed more information on these nodes, that is also the originating state and the top-most stack symbol (before the transition is fired). A run shall therefore be represented as a tree over the alphabet $\Lambda = \Sigma \times Q \times \Gamma$.

The set of runs of a pushdown system form a regular tree language. We detail explicitely a tree automaton accepting these runs, whose set of states, namely Q_D consists of 3 kinds of states:

- $\{q_0^{/1}\}$, the initial state, which accepts exactly empty trees.

- R-states, labeled by $\{0,1\} \times Q \times \Gamma \times Q$. Such a state $r^i_{q,\gamma,q'}$ will accept runs from the state q with a visible stack symbol γ (that will not be popped) to a state q' with the same stack if $i = 0$, with a bigger stack if $i = 1$.

- P-states, labeled by $\{0,1\} \times \Gamma \times Q \times \Gamma \times Q$. Such a state $p^{i,\gamma'}_{q,\gamma,q'}$ will accept runs from a state q with the top of the stack containing $\gamma\gamma'$ that will start by popping γ', never pop the γ and end to the state q' with visible stack symbol that γ if $i = 0$, a bigger stack if $i = 1$.

The set of accepting states are the $r^i_{q,\perp,q'}$ such that q is the initial state of the pushdown system, $i \in \{0,1\}$ and $q' \in Q$. Since we use Λ as alphabet, this automaton is deterministic as the letter labeling a node characterizes uniquely the possible states of its children.

$\xrightarrow{\alpha}$ is clearly automatic with this representation as two trees are in relation with $\xrightarrow{\alpha}$, if they both represent runs of the pushdown system, and they only differ in that the second one has one additional node that appears at the end of its depth-first traversal. Tracking the difference of one additional node in the depth-first traversal can be easily done by a 3-state deterministic automaton:

$$\begin{aligned}
Q =& \{q_0^{/2}, i, d_\alpha\} \\
\Delta =& \{((\diamond, (\alpha, q, \gamma)), q_0^{/2}, q_0^{/2}, d_\alpha) \mid q \in Q, \gamma \in \Gamma\} \\
& \cup \{((\lambda, \lambda), q, q', i) \mid \lambda \in \Lambda; q, q' \in \{q_0^{/2}, i\}\} \\
& \cup \{((\lambda, \lambda), d_\alpha, q_0^{/2}, d_\alpha) \mid \lambda \in \Lambda\} \\
& \cup \{((\lambda, \lambda), q, d_\alpha, d_\alpha) \mid \lambda \in \Lambda, q \in \{q_0^{/2}, i\}\}
\end{aligned}$$

with initial state $q_0^{/2}$ and final state d. It then suffices to make a product automaton with an automaton accepting convolutions of elements of the domain (that is trees representing a run in P).

$\hookrightarrow$ is also automatic, to show that we define a more general relation $\hat{\hookrightarrow}$, which holds for trees such the last node in the depth-first traversal of the second tree is a right successor, and the first tree is identical except that it doesn't have the subtree rooted at the predecessor of that node. $\hat{\hookrightarrow}$ can be accepted by a 5-state automaton, with:

- an initial state $q_0^{/2}$,
- a state l that accepts leaves labeled in $\{\diamond\} \times \Lambda$,
- a state j that accepts (non-leafy) trees labeled in $\{\diamond\} \times \Lambda$ whose right subtree reaches a l state,
- a state k which accepts all the other trees labeled by $\{\diamond\} \times \Lambda$,
- i that accepts convolutions of the same tree
- and f the final state (which is obtained from a (β, β)-labeled node with either a j or f state on the left side and a $q_0^{/2}$ state on the right side, or with a i or $q_0^{/2}$ state on the left side and a j or f state on the right side).

Both j and f are final. The automaton accepting $\hookrightarrow$ is obtained by product of this 5-state automaton with the automaton accepting convolutions of elements of the domain.

Saturation

Now that we have an explicit automatic presentation of the nested-pushdown graph of P, we want to characterize its expressiveness. In this section we will show that the saturated structure is essentially a sum of nested pushdown graphs.

First we can remark that if we suppress all non reachable and all non co-reachable states in the automata $A_D, A_{\xrightarrow{\alpha}}, A_{\hookrightarrow}$, then only trees that represent a valid subrun in the pushdown system reach states. By subrun, I mean here that there is a state q and a stack symbol γ such that the tree represent a run in P starting from state q with a stack containing $\bot\gamma$ that will not pop the γ. These runs can be characterized by runs in some other pushdown systems.

We define a family of pushdown systems: $(P_{q,\gamma})_{q\in Q,\gamma\in\Gamma}$, such that $P_{q,\gamma}$ will characterize subruns of P starting from state q with stack $\perp\gamma$ and not popping that γ. The set of states and the alphabets of $P_{q,\gamma}$ will be the same as P. The initial state will be changed to q, and the transition relation will be changed to reflect the fact that we start the run with a γ (which will not be popped) as the topmost symbol. For any stack symbol $\gamma \in \Gamma\backslash\{\perp\}$, the transition function of $P_{q,\gamma}$, namely $\delta_{q,\gamma}$ will not differ; $\delta_{q,\gamma}(q', \alpha, \perp) = \delta(q', \alpha, \gamma)$ to reflect that we start the run with γ as the topmost stack symbol.

Now, it is easy to remark that the label of the root of a tree in T^*_Λ that reaches a state in some automaton of the automatic presentation characterizes which are the starting state and top-most stack symbol of the subrun it represents. Therefore a tree representing a subrun can be charactarized uniquely (thanks to the label of its root) as a run in some $P_{q,\gamma}$. Furthermore, a pair of non-empty trees are in relation only if they are both valid subruns which start from the same state in P and with the same top-most stack symbol.

Lemma 3.26. *The domain of the saturated structure can be partitioned into the empty tree, trees that do not represent valid subruns in P, and for each $(q, \gamma) \in Q\times\Gamma$, runs in the PDS $P_{q,\gamma}$.*

The saturated structured is thus a direct sum of the structures of nested pushdown graphs for each $P_{q,\gamma}$.

Kartzow [Kar09] gave a doubly exponential bound on the size of a representative for each FO^r_m-undistinguishability class of a nested pushdown graph, from that we deduce the same bound for the saturated structure, which allows to give a triply exponential bound on the size and time of construction of the automaton, when using this presentation.

3.4 Conclusion

In Chapter 2 we showed that the inductive construction of an automaton for Presburger formula (using least significant digit first coding) could be performed in 3EXPTIME, which is the optimal deterministic time complexity class for which an algorithm to decide Presburger is known. In Chapter 3 we showed that this result also holds when using most significant digit first coding and could be generalized to other automatic structures with elementary first-order theory for which the generic inductive construction of an automaton accepting solutions of a formula has a complexity close to the first-order theory of the structure. To ensure the lower bound on the construction of automata, we inspect the relations defined by all states of the automata of the presentation, and in the cases we studied, their first-order theory is no harder than the first-order theory of the structure presented.

The optimality of these results is limited in that we establish deterministic time upper bounds despite the complexity of these logics being complete for space or alternating time complexity classes. It is still a major open question whether these deterministic time and space or alternating time complexity classes are separated or not.

These results call for a more general result:

Conjecture 3.27. *Given an automatic structure, the inductive construction of an automaton accepting solutions of a formula is no harder than the first-order theory of the logic presented by the automatic structure.*

This result would be very strong, but so far we have not exhibited any counter-example to this conjecture. Even if we could exhibit one, perhaps this conjecture could be weakened in the following manner, making it much more difficult to either validate or invalidate it:

Conjecture 3.28. *Given an automatic structure, there exists an automatic presentation of that structure, such that the inductive construction of an automaton accepting solutions of a formula is no harder than the first-order theory of the logic presented by the automatic structure.*

Chapter 4

Streaming String Transducers

Synchronously regular relations, those definable in the context of automatic structures, which we studied in the previous chapter, are a very restricted class of relations and their first-order theory is still decidable.

A usual generalization of those relations consists of allowing ε on some tracks in transitions, inducing a desynchronization between the different tracks: this model is equivalently seen as multi-tape automata. Determinism of those machines is a crucial feature, as in the non-deterministic case, even equivalence is not decidable [Gri68]. Because of determinism, these relations become functional, and even injective. And this model of deterministic multi-tape automata does not have a nice equivalent logical definition, though they enjoy nice properties, noticeably closure under composition, or regularity preservation.

In this chapter, we will focus on string transformations, that is functional relations. We will present a more general framework of logically (using monadic second-order logic) defined transformations from graphs to graphs, which we will restrict to word input. We will show that this class of transformations is exactly captured by the equivalent model of Streaming String Transducers introduced by Alur and Černý [AČ11].

As we will see, it is possible to restrict the domain of these transformations to MSO definable classes of graphs, for example finite and/or infinite strings, or trees. Their restriction over string input appeals for an automaton looking model, here are some key features of automata that we would like to have over such transformations:

- An automaton allows to check the validity of the corresponding formula over a word by traversing that word in a sequential left-to-right single pass, using a fixed amount of memory.

  They also have exactly the same expressive power as non-deterministic Turing Machines with constant memory. It is interesting that for this class of languages, not only can non-determinism be removed, but also the processing can be forced to a sequential processing.

  This left-to-right single pass processing, using a finite number of control states will be a property that our computational model for string-to-tree transformations will also enjoy.

- Equivalence of automata is decidable, thanks to an effectively computable canonical minimal form. Though the model that we will present does not have a clear notion of canonical nor minimal form, equivalence is decidable, and relies on the computational model.
- The sequential processing of automata allows a generalization to infinite word input, where the acceptance is defined by the set of infinitely occuring memory states.
- Tree automata lose this property of being sequential deterministic fixed-memory machines. But they need either parallel processing (with the notion of bottom-up automaton), or an unbounded stack (using a pushdown automaton, which processes sequentially the in-order traversal of the tree)

First we will present a class of graphs transformations defined using MSO [Cou94]. We will work with such transformations over string input and with string or tree output. Then we will present the computational framework of streaming string-to-tree transducers: an extension of finite state automata, which process deterministically their input in a single left-to-right pass. They furthermore have a finite state of registers to compute some output value hence superseed automata as the processing of an input word yields a value and not just a boolean value.

The central result that we will show in this chapter is that these streaming string-to-tree transducers (SSTTs) implement exactly the string to tree transformations that can be defined logically using MSO. We will provide an explicit construction of a streaming string transducer implementing an MSO transformation.

Theorem 4.1. *An infinite string-to-tree transformation is MSO-definable if and only if it is* SSTT*-definable, and the reduction from* SSTT *to MSO-transducers and vice-versa is effective.*

Finally, relying on the equivalence between these logically defined transformation and the computational framework of streaming stransducers, we will present some decision procedures over streaming transducer to establish the decidability of some problems, such as functional eqquivalence, over MSO defined string transformations.

4.1 À la Courcelle transformations

Courcelle [Cou94] proposed a way to use monadic second-order logic to define directed labeled graph transformations $R \subset GR(\Sigma, \Sigma') \to GR(\Gamma, \Gamma')$. The main idea is to define a transformation $(G, G') \in R$ by defining the graph G' using a finite set of copies of the graph G. The existence of nodes, edges, and node-labels in G' is then given as MSO(Σ, Σ') formulas.

By MSO(Σ, Σ'), we mean MSO over the signature which contains a monadic predicate for each letter in Σ and a binary predicate for each letter in Σ'. A graph $G \in GR(\Sigma, \Sigma')$ will be interpreted as the structure whose domain is the set of nodes of G, the predicate corresponding to some letter $\alpha \in \Sigma$ will hold exactly for α-labeled nodes, the predicate corresponding to α' will hold for nodes x, y iff there is an α'-labeled edge from x to y in G.

Definition 4.2. [Cou94] An *MSO graph transducer* or *MSO graph transformation* is a tuple $T = (\Sigma, \Sigma', \Gamma, \Gamma', \phi_{\text{dom}}, C, \phi_{\text{nodes}}, \phi_{\text{edges}})$ where:

- Σ and Γ (resp. Σ' and Γ') are finite sets of input and output node labels (resp. edge labels) alphabets;
- ϕ_{dom} is a closed MSO(Σ, Σ') formula characterizing the domain of the transformation;
- $C = \{1, 2, \ldots, n\}$ is a finite set of copies of the nodes of the input graph;
- $\phi_{\text{nodes}} = \left\{\phi^c_\gamma(x) : c \in C \text{ and } \gamma \in \Gamma\right\}$ is a finite set of MSO(Σ, Σ') formulas with a free first-order variable x;
- $\phi_{\text{edges}} = \left\{\phi^{c,d}_{\gamma'}(x, y) : c, d \in C \text{ and } \gamma' \in \Gamma'\right\}$ is a finite set of MSO(Σ, Σ') formulas with two free first-order variables x and y.

The graph transformation $[\![T]\!]$ characterized by T is defined as follows. A graph $G = (V, (E_b)_{b\in\Sigma'}, (L_a)_{a\in\Sigma}) \in GR(\Sigma, \Sigma')$ is in the domain of $[\![T]\!]$ if $G \models \phi_{\text{dom}}$ and the output is the graph $G' = (V', (E'_b)_{b\in\Gamma'}, (L'_a)_{a\in\Gamma}) \in GR(\Gamma, \Gamma')$ such that:

- V' is the set of nodes v^c such that $v \in V$, $c \in C$ and there is a unique $a \in \Gamma$ such that $G \models \phi^c_a(v)$; notice that we follow the convention that a node v^c is absent if $G \models \neg\phi^c(v)$ where $\phi^c(v) \overset{\text{def}}{=} \bigvee_{a\in\Gamma} \phi^c_a(v)$;
- $(E'_b)_{b\in\Gamma'}$ is the set of b-labeled edges such that for $v, u \in V$ and $c, d \in C$ we have that $(v^c, u^d) \in E'_b$ if $G \models \phi^{c,d}_b(v, u)$;
- $(L'_a)_{a\in\Gamma}$ is the set of a-labeled nodes such that $v^c \in L'_a$ if $G \models \phi^c_a(v)$.

Note that as the output is unique, MSO graph transformations implement functions. It is a well-known [Cou94] fact that such MSO graph transducers are closed under functional composition.

An MSO string-to-tree transducer is an MSO graph transducer such that its domain is restricted to strings, while the output is restricted to binary trees. Such restriction can be imposed by composing two graph transducers where the first one defines the required transformation, while the second one verifies whether the output is a tree. Though string-to-tree transducers cannot be composed, the subclass of string-to-string transducers are closed under functional composition.

The input edge-labeled alphabet has thus only one symbol, and the output edge-label alphabet will have two symbols $\{1, 2\}$ (or just one symbol for the subclass of string-to-string transducers), and outputs will have the property that they are connected acyclic graphs such that each node has at most two outgoing edges (with different labels) and at most one ingoing edge. We write MSOT for the set of string-to-tree transformations expressible by MSO transducers.

Example 4.3. We now present an example of an infinite string-to-tree transformation $T_\#$ that we can define in this logical framework. In Figure 4.1, we describe the transformation depending on whether the input string contains finitely or infinitely

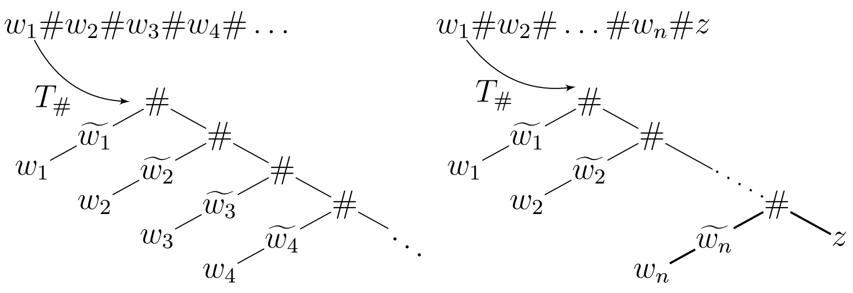


Figure 4.1: An infinite string-to-tree transformation $T_\#$.

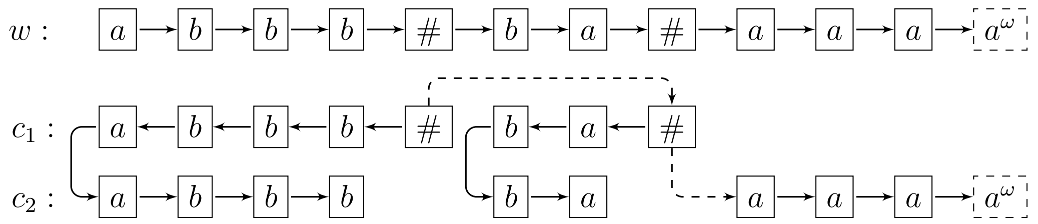


Figure 4.2: The result of MSO transformation $T_\#$ on the string $abbb\#ba\#a^\omega$.

many #'s (The w's are finite #-free strings, while z is a #-free ω-string. The string $\widetilde{w}$ denotes the reverse of string w as a left-branch tree-string).

To give the logical definition of this transformation, we consider the following MSO formulas with their intuitive meaning: $\text{reach}_\#(x)$ (holds if from x one can reach a node labeled #), $\text{first}(x)$ (holds if x is first position of the string) and $\text{path}(x, y)$ (holds if there is a path from x to y). Figure 4.2 presents the way we choose to implement this tranformation:

Let $T = (\Sigma, \Gamma, \phi_{\text{dom}}, C, \phi_{\text{nodes}}, \phi_{\text{edges}})$ the MSO transducer implementing $T_\#$:

- Σ=Γ= $\{a, b, \#\}$, $C = \{1, 2\}$, and
- $\phi_{\text{dom}} = \mathit{true}$,
- $\phi^1_\gamma(x) = L_\gamma(x) \wedge (L_\#(x) \vee \text{reach}_\#(x))$

  as shown in Figure 4.2, nodes in the first copy are those before the last #-labeled node (they have the same label as in the input).
- $\phi^2_\gamma(x) = L_\gamma(x) \wedge (\neg L_\#(x))$

  nodes in the second copy are those who are not #-labeled.
- $\phi^{1,1}_1(x, y)$=$\phi^1(x) \wedge \phi^1(y) \wedge \text{reach}_\#(x) \wedge \neg L_\#(y) \wedge E(y, x)$

  there exists a 1-edge from nodes x^1 and y^1 if there exists an edge from y to x in the input. Furthermore both x^1 and y^1 must exist in copy 1 and y (thus y^1) must not be labeled by #.
- $\phi^{1,1}_2(x, y) = L_\#(x) \wedge L_\#(y) \wedge \text{path}_\#(x, y) \wedge \forall z((\text{path}(x, z) \wedge \text{path}(z, y)) \rightarrow \neg L_\#(z))$

this formula states that there exists a 2-edge from each two consecutive # in copy 1.

- Other formulas $\phi_1^{1,2}, \phi_2^{1,2}$ and $\phi_1^{2,2}$ can also be expressed in MSO according to Figure 4.2.

  Nodes from the first copy that are connected with a 1-edge to nodes the second copy at the same position if these nodes are at the first position or just after a # which is not the last one. This can be expressed in MSO by $\phi_1^{1,2}$.

  If the input word contains finitely many #, there is only one 2-edge from the first copy to the second copy, it connects the last # in the first copy to the first node after that in the second copy. If the input has infinitely many hashed there is no such 2 edge. This can be expressed in MSO by $\phi_2^{1,2}$.

  Nodes in the second copy are connected to each other with a 1-edge if they were connect in the input: $\phi_1^{2,2}(x, y)$ just need to check whether x^2 and y^2 exist and whether y is the immediate successor of x.

- Finally $\phi_2^{2,2}$, $\phi_1^{2,1}$, $\phi_2^{2,1}$ are always false as there are neither 2-edge from nodes in the second nor 1-edge from the second copy to the first copy.

4.2 The Streaming String Transducer model

We will present in this section the computational model for the case of string-to-tree transformations, which we call streaming string(-to-tree) transducers –SS(T)T's.

Streaming string-to-tree transducers are deterministic finite state machines that read the input string once in a left-to-right pass and manipulate a finite set of registers containing trees with marked positions (called holes) where further trees can be appended.

If the input word is finite, the output will be defined as the value of some register (depending on the last state). In the case of infinite input word, The set of infinitely fired transitions determines the output register that defines the output as the limit of the successive values of this register.

4.2.1 Registers updates

Before we formally introduce streaming string-to-tree transducer in section 4.2.3 we discuss the set of allowable register updates. Let X be a finite set of registers and Γ be an alphabet. Let $H_\Gamma \subseteq \mathcal{T}^{\infty}_{\Gamma\cup\{?\}}$ be the set of trees with holes, that is the set of trees over Γ with a special symbol $? \notin \Gamma$ (called the *hole*) that appears only at leaf positions. Streaming string-to-tree transducers use registers to store and manipulate trees with holes using the following register expressions:

$$e_1, \ldots, e_n ::= \eta \mid ? \mid x \mid a(e_1, e_2) \mid x[e_1, \ldots, e_n]$$

where $x \in X$ and $a \in \Gamma$. We write $E(\Gamma, X)$ for the set of *register expressions* over the alphabet Γ and register set X. A *register update* s of the set of registers X is defined as a (partial) mapping $s \in [X \to E(\Gamma, X)]$.

Given a register valuation $\nu : X \to H_\Gamma$ we say that the update s is *compatible* with ν if and only if, each $x \in X$ appearing in an image of s has a value given by ν, and if $\nu(x)$ has n holes (?-labeled nodes) then x can only appear in the form x or $x[e_1, \dots, e_n]$ in the images of s. If s is compatible with ν, we define the new valuation $\nu' = s(\nu)$ as follows: $\nu'(x)$ is defined iff $s(x)$ is defined and $\nu'(x) = [\![s(x)]\!]_\nu$. This evaluation $[\![.]\!]_\nu$ is a standard conversion from terms to trees with the following difference: x is evaluated as the tree $\nu(x)$ (which must be defined because of compatibility), and $x[e_1, \dots, e_n]$ is evaluated as the tree $\nu(x)$ in which each of the n ?-labeled leaves (considered ordered as their order of occurrence in the left-to-right traversal of the tree $\nu(x)$) has been replaced (in that order) by $[\![e_1]\!]_\nu, \dots, [\![e_n]\!]_\nu$.

Restriction on the update function

As MSO transducers use only a fixed finite number of copies, they only implement transformations with linear size increase. We need to add further restriction on the register updates for SSTTs so that they do not allow more general transformations.

The main idea is to forbid that the content of some register at some point in the processing of the word appears an unbounded number of times in the output [1].

In their original paper, Alur and Černý [AČ11] ensured this with the notion of *copyless*ness: A register expression $u \in E(\Gamma, X)$ is *copyless* (or linear) if each $x \in X$ occurs at most once in u. Similarly, an update of registers s is copyless if each expression in the image is copyless and each $x \in X$ appears in at most one image. This restriction derives from their proof of equi-expressiveness of two-way transducers (which were known to have the same expressive power as MSO transducers [EH01]) and (finite) string-to-string tansducers: the two-way transducer is reduced to a one-way heap-based transducer (with a fixed number of pointers). Then this heap is flattened into a set of registers: it is represented as a bounded number of strings and some (finitely flavored) information of how these are combined to reflect the structure of the heap. The operations required by the heap based model are naturally expressed using copyless updates of registers.

To extend this construction to infinite strings-to-string transformations [AFT12], a more relaxed restriction on registers update was required. First for infinite strings, the notion of two-way transducers needed to be extended to that of two-way transducers with regular lookaround to capture the expressive power of MSO. Then these two-way transducers with look-around can be implemented by non-deterministic streaming string transducers with look-around, which enjoy the strong property of being functional. The look-around is then showed not to extend the expressive power of these functional non-deterministic transducers: the look-back can be remembered by the states as this model is one-way, and the look-ahead can be non-deterministically guessed thanks to the non-determinism. The crucial step of the proof is to then remove non-determinism, while still maintaining some restriction on the update function (so that the model only implements MSO-expressible transformations). This purpose is achieved with a more relaxed update rule called

1. Though this is not a necessary condition for the output to be linear w.r.t. the input, this condition is sufficient

K-bounded alive copy, which ensures that the transition system and the update of registers is such that over any run the number of register values that have been duplicated and that are still alive (i.e. present in some register) are bounded by some constant K. This restriction is finally shown equivalent to that of copylessness.

Copylessness of a transducer is very syntactically ensured, on the other hand K-bounded copy require some analysis of the behaviour of the transducer. However the former restriction requires usually more registers to implement a transformation and this additional number of registers is likely to obfuscate the transformation implemented by a transducer. Figure 4.3 is a convincing example of a 1-bounded transducer, and while copylessness can be ensured by adding a single additional register (which we would call y'), it would be very burdensome to do so.

We can achieve the best of both restrictions: a syntactic one which still allows some (safe) copying, by means of *restricted copy*: under restricted copy update rule, a register is allowed to be copied in multiple registers, however these registers cannot later be combined together. Restricted copy rule is syntactically imposed by defining a symmetric and reflexive (but not transitive) *conflict relation* over the set of registers. The content of one register can be duplicated in two (or more) conflicting registers, but any two conflicting registers cannot be combined together, neither directly, nor indirectly by ensuring that any two non-conflicting registers do not receive values from conflicting registers. Restricted copy clearly ensures that at any point the content of some register will not appear more than once in the output (or even in any other register).

It is easy to check this restriction provided the conflict relation is given, and the conflict relation can anyhow be computed inspecting only the register update function (i.e. without inspecting the transition system). Lemma 4.5 shows that this restriction suffices to forbid non MSO-definable transformations.

4.2.2 Output definition

There is a natural way to define the output in the case of a finite string input, the final state of the run will determine which register contains the image of the input word. If we restrict to copyless updates, we need to allow a bit more processing and define the output as the interpretation of a register term (determined by the final state). Thus for a finite string to tree transducers, the output definition is a function mapping states to registers (or states to registers expressions).

The case of infinite word input is more complex as there is no end to the run. Furthermore, the output can be infinite, and the registers will always contain finite trees. The first issue will be handled similarly as for automata over infinite words: we will inspect the infinitely occuring states. To obtain an infinite tree from finite trees, we will use the simple notion of limit.

We cannot just use the same output definition for infinite string to trees transducers, as there would not be any natural way to define uniquely the output tree. To capture precisely the infinitely occuring states, we will use an à la Muller output definition: the output will be defined as the limit value[2] of some register determined

2. With a natural definition of limit of trees, using the following metric: two trees t_1, t_2 are at distance 2^{-k} for the largest k such that the trees are identical up to level k

by the set of infinitely occuring states.

For convenience we will detail an à la Muller output definition based on the set of infinitely occuring transition (as this can be easily reduced to an inspection of the set of infinitely occuring states – simply by remembering in the state the last fired transition).

In order to ensure the convergence of the successive values of the "output" register, we will impose some further restriction on the update of registers: any register that is an output register for some set of transition will by updated by those transitions only by inserting trees with holes in its holes. Though the value of those registers will not always have a limit, this restriction on the updates will always ensure the convergence of the register that will happen to be the outputting one; since eventually, it will only be updated by inserting trees in its holes.

In order to ensure convergence towards an hole-free tree, we can either consider that each hole is replaced by η (the empty tree), or that there will be a transition in the corresponding set of infinitely occuring transition that will fill that hole with a non-empty tree.

As we will, in the following, prove a reduction from à la Courcelle infinite string-to-tree MSO transformation to SSTTs, we will not worry so much about these rather syntactic restriction which ensure the convergence of the output, as by definition, we will have the convergence. Indeed our construction will inspect the value of some register (determined by the set of infinitely occuring transitions), only when a specific transition is fired. It will happen that the other infinitely occuring transitions will move the content of that register from registers to registers, updating that content only by appending some trees in its holes. Allowing such thing in the definition of SSTTs would only help proving the reduction at the cost of a much more complex restriction on updates and output definition.

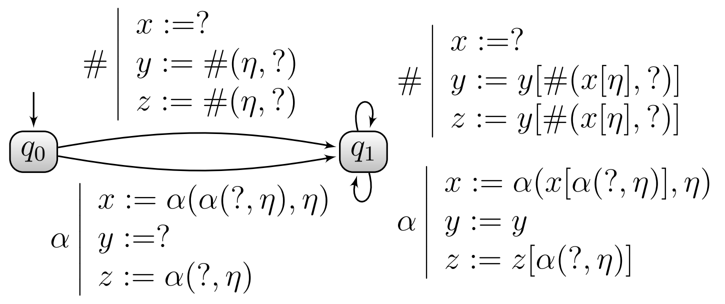
As there is no fundamental difficulty in imposing these restriction to the SSTTs that we will build from MSOT, this last part of the reduction will be omitted.

4.2.3 Definition of SST's

Now we are in a position to define the streaming transducer model.

Definition 4.4. A streaming string-to-tree transducer (SSTT) T is a tuple $(\Sigma, \Gamma, Q, q_0, \delta, X, \kappa, \rho, F, F')$ where:

- Σ and Γ are finite input and output alphabets,
- Q is a finite set of states whose initial state is q_0,
- $\delta : Q \times \Sigma \to Q$ is a transition function,
- X is a finite set of registers,
- κ is the conflict relation over X,
- $\rho : Q \times \Sigma \times X \to E(\Gamma, X)$ is the restricted-copy (with respect to the conflict relation κ) registers update function,


q_0
q_1
$\#$ | $x :=?$, $y := \#(\eta, ?)$, $z := \#(\eta, ?)$
α | $x := \alpha(\alpha(?, \eta), \eta)$, $y :=?$, $z := \alpha(?, \eta)$
$\#$ | $x :=?$, $y := y[\#(x[\eta], ?)]$, $z := y[\#(x[\eta], ?)]$
α | $x := \alpha(x[\alpha(?, \eta)], \eta)$, $y := y$, $z := z[\alpha(?, \eta)]$

$$F(\{(q_1, \alpha)\}) = z$$
$$F(\{(q_1, \alpha), (q_1, \#)\}) = y$$
$$F(\{(q_1, \#)\}) = y$$
$$F'(q_1) = y[\#(x[\eta], \eta)]$$
$$\kappa = \{(x, z), (y, z)\}$$

Figure 4.3: An SSTT implementing transformation $T_\#$.

- $F : 2^{Q \times \Sigma} \to X$ is the Muller (over transitions) output function for infinite input such that given for all S such that $F(S)$ is defined we have that the update of register $F(S)$ by transitions in S are of the form $F(S)[e_1, \dots, e_n]$ where $e_1, \dots, e_n \in E(\Gamma, X)$,

- $F' : Q \to E(\Gamma, X)$ is the output function for finite input.

We now detail the semantics of this model: the unique run $\varrho(T, w)$ of the SSTT T over the string w is the sequence $(q_i, \nu_i)_{i \le |w|}$ where $q_i \in Q$, and ν_i is a valuation of the set of registers X, such that $q_{i+1} = \delta(q_i, w[i])$, and ν_{i+1} is defined as $x \mapsto [\![\rho(q_i, w[i], x)]\!]_{\nu_i}$ (initially ν_0 is assumed to be the empty valuation). If w is finite, then the output is defined as the evaluation of the term $F'(q_{|w|})$ by $\nu_{|w|}$ (provided compatibility). If w is infinite, let Δ be the set of infinitely often fired transition (that is a set of tuples of $Q \times \Sigma$) in the run $\varrho(T, w)$. The output of T for the string w is defined as limit of sequence of values appearing in the register $F(\Delta)$ if it converges towards a tree of $\mathcal{T}_\Gamma^\infty$. The syntactic restriction on the update of the output register enforces that if the Muller set is defined then the output always converges towards a tree. We give in Figure 4.3 a transducer implementing transformation $T_\#$.

The central result (Theorem 4.1) is that SSTTs capture precisely the same class of transformations as MSOTs. The proof of this fact follows form the following two lemmas.

Lemma 4.5 (SSTT $\subseteq$ MSOT)**.** *Every* SSTT*-definable transformation is* MSOT*-definable.*

Lemma 4.6 (MSOT $\subseteq$ SST)**.** *Every MSO definable transformation from strings to trees is* SSTT*-definable.*

Lemma 4.5 is the easy direction. Informally, when considering string or tree automata, the expressiveness of MSO allows easily to capture the behaviour of these finite transition systems thanks to second-order quantification: a run is "guessed" with an existantial quantification, and its validity is "checked" locally with first-order universal quantification. The extension of this proof to MSO transducers follows from a straightforward extension of the reduction [AFT12] from SST to MSO string to string transformations.

Proof of Lemma 4.5. First remark that we can write for each state of the SSTT an MSO formula φ such that $w \vDash \varphi(x)$ if at position x in w the SSTT is in that state[3].

With these formulas, we can also write for each register an MSO formula which holds at position x if its content at that point will appear in the output. Note that if it does, we know from the restriction on registers update that it will appear only once. A register X at position x in w will appear in the ouptut iff X appears in the image by the (appropriate) update relation of some other register Y at position $x+1$ which will appear in the output.

Now, we define the copies of the MSO transducer: we define a copy for each letter appearing in some register expression given by our registers update function.

The copies will be thus be indexed by a transition, a register and the index of occurence of the letter in the in-order traversal of the associated register term. The existence of a node in the image will yield its node-label.

A node x of copy c will be in the image of the transformation, if the processing of position x by the SSTT resulted in the update of registers corresponding to c to be activated. Formulas that define the nodes of the image are thus easy to build.

Remark that in the case of finite words, we would need to add additional nodes to reflect the fact that the output is defined with a register expression (which can contain letters) depending on the final state of the run. This can be done by adding additional copies which might only contain nodes in the image at the last position. The edges of those nodes can be defined similarly as for the other nodes.

Now that we have characterized the nodes of our tree, we will explicit how edge formulas will be written. Consider a node in a register update fired at position x, it either have no successor (thus the edge formula is false for this node), a letter as successor (the edge formula is given by the existence formula) or a hole as successor.

In the latter case, we need to “fetch” the position y and register Y where the root node of the tree that will be inserted is defined. For that we will guess (using an existential quantification) the position z where the register update “connected” the two nodes.

To relate x and z, we need to track both the register that contains that hole and the relative position of that hole among other holes in the tree (that is its index among the holes in the in-order traversal) along the run up to position z. Remark that we can easily write formulas (indexed by a register and an integer; with one free variable) that track the number of holes in the value of a given register –whose value will be part of the output– has at some position, as this number is bounded (by some integer b) from the definition of the SSTT. We will have formulas indexed by a tuple of registers and a tuple of integers smaller than b and with two free first-order variables.

However at position z a tree consisting of just one hole can be inserted, we need to disregard that (we can easily track at which positions which registers contain such dummy values) and “keep going” (formally by an existential second order quantification of all such updates).

To relate y and z we just need to write formulas (indexed by a tuple of registers, and with two free variables) that track the successive registers containing at the

3. This is essentially the proof that the language of a DFA can be defined by an MSO formula.

root the content at y of register Y. Similarly we need to ignore this content being inserted in a trivial hole tree.

We conclude this proof by giving the property of our construction that leads to the equality of this tree with the limit in the infinite input word case. At any position, the graph (consisting of nodes before that position) contains the content of registers at that position that will appear in the output. Therefore, at any position once no more finitely occuring transition will be fired, the content of the output register will be part of the output and is in the graph. Therefore the MSO transformation that we defined, by relying on second order existential quantification to simulate its run, implements the SSTT. □

The proof of Lemma 4.6 is covered in the following two sections. In the next section we first introduce an intermediate model, SSTT with regular lookaround, and show a reduction from MSOT to this model. In Section 4.3.4 we complete the proof of Theorem 2.15 by showing that SSTTs are closed under regular lookaround.

4.3 Equi-expressiveness of the two models

The logical definition of the transformations, relies on MSO which is decidable. Therefore, there exists a straightforward non-deterministic algorithm to implement transformations of finite strings: guessing an annotated output and checking the existence of each edge. For the finite word input case, the first proof [AČ11] went very similarly as for the equivalence between string automata and read-only Turing Machines, that the forth and back motion of the reading head could be restricted to a single pass with a finite amount of memory. A clear intuition is given by the concept of regular look-ahead: knowing approximately what is left to be read ensures a correct construction of the image. The computational model allows, in the finte word case, to handle simultaenously all cases (whose number is finite) of the reminder of the word, and select at the end of the processing the correct one.

In the case of infinite word input, the approximate knowledge of the suffix is also sufficient for a correct construction, in the following sections, we will first present this notion of (finite case) approximate knowledge of the suffix, then we will show how this knowledge allows us to perform a correct construction of the image, and finally we will see how all cases of the suffix can be handled simultaneously, and why this handling (unsurprisingly) relies on the infinitary behaviour of the input word.

4.3.1 MSOT and SSTT with regular Look-Around

We consider an extension of SSTT where the transducer can make transitions based on regular properties of the string read so far (look-behind) as well as ω-regular property of the remaining ω-string (look-ahead). We call this extension SSTT with look-around. Notice that due to aforementioned look-around capabilities such transducer can be state-free since the state information can be encoded in the look-around queries.

Informally an SSTT_{rla} is an SSTT such with some regular languages as guards over transitions which check whether the the prefix and the suffix belong to that language. We therefore allow multiple transitions with the same label outgoing from one state, but we impose that the guards are mutually exclusive. Notice that thanks to the regular test on the prefix read so far (the look-behind), we don't need states anymore, as they can be encoded by guards.

Formally, we define SSTT with look-around as the following.

Definition 4.7 ($\mathrm{SSTT}_{\mathrm{rla}}$). A streaming string-to-tree transducer with look-around is a tuple $(\Sigma, \Gamma, X, \Lambda_b, \Lambda_a, \rho, F)$ where

- Σ, Γ, and X are the same as in definition 4.4,
- Λ_b and Λ_a are two finite sets of regular languages over Σ respectively called regular look-behind and look-ahead. Languages in Λ_b (resp. Λ_a) are pairwise disjoint.
- $\rho : \Lambda_b \times \Sigma \times \Lambda_a \times X \to E(\Gamma, X)$ is the (look-around based) copyless register-update function.
- $F : \Lambda_b \times \Sigma \times \Lambda_a \to X$ is the output function.

The semantics of $\mathrm{SSTT}_{\mathrm{rla}}$ is defined as follows. A run over a word w is a finite or infinite sequence of partial valuation of registers X, $(\nu(i))_{1 \leq i \leq |w|}$ that is compatible with the update function, that is for any $i \leq |w|$, we have $\lambda_b^i \in \Lambda_b$, $\lambda_a^i \in \Lambda_a$, such that $w[1{:}i) \in \lambda_b^i$, $w(i{:}|w|] \in \lambda_a^i$ and the update of registers $x \mapsto \rho(\lambda_a^i, w[i], \lambda_b^i, x)$ is compatible with the valuation ν_{i-1} (when $i = 1$, we consider ν_0 to be the empty valuation), and $\nu_i(x) = [\![\rho(\lambda_b^i, w[i], \lambda_a^i, x)]\!]_{\nu_{i-1}}$ when it is defined. The output of the string w is then defined as the limit of the sequence $\nu_i(F(\lambda_a^i, w[i], \lambda_b^i))$ if it exists.

In the sequel, this convergence will be ensured by the following property on the update of registers: Upon processing a word w, if at some position i, the register $O_i^w = F(\lambda_a^i, w[i], \lambda_b^i)$ holds a value, then $O_{i+1}^w = F(\lambda_a^{i+1}, w[i+1], \lambda_b^{i+1})$, will also hold a value and $\rho(\lambda_a^{i+1}, w[i+1], \lambda_b^{i+1}, O_{i+1}^w)$ will be of the form $O_i^w[e_1, \ldots, e_k]$ for some $k, e_1, \ldots, e_k$.

In the remainder of the section we prove the following Theorem:

Theorem 4.8 ($\mathrm{MSOT} \subseteq \mathrm{SSTT}_{\mathrm{rla}}$).
Every MSOT*-definable transformation is* $\mathrm{SSTT}_{\mathrm{rla}}$*-definable.*

4.3.2 Bounded-Crossing Property of MSOT

To show the reduction form MSOT to $\mathrm{SSTT}_{\mathrm{rla}}$ we first establish so-called bounded-crossing property of MSOT that allows us to capture this class by a computational model that manipulates only a bounded number of registers. Informally, this property states that at any position in the input the number of crossings in the image, i.e. the number of edges from one side of the position to the other side of the position, is bounded by a number that depends only on the number of copies and the quantifier depth of the formulas used in the MSOT.

Given a (possibly infinite) string w, a position x of w and an MSO transformation $T = (\Sigma, \Gamma, \phi_{\mathrm{dom}}, C, \phi_{\mathrm{nodes}}, \phi_{\mathrm{edges}})$, a *crossing* at position x is a pair $(x_1, c_1), (x_2, c_2) \in \mathbb{N}_{|s|} \times C$ such that $x_1 \leq x < x_2$, and there is an edge in $[\![T]\!](s)$ between nodes $x_1^{c_1}$ and $x_2^{c_2}$.

Theorem 4.9 (Bounded-Crossing Property)**.** *For every MSO-definable (infinite) string-to-tree (or string-to-string) transducer T, the number of crossings at any position is bounded by $3|C| + 2|C||\Sigma||\Theta_k|^2$, where k is the maximal quantifier depth of MSO formulas appearing in T and C is the copy set of T.*

The proof of this theorem will rely on the following result on MSO logics:

Theorem 4.10. *Given two strings w, w' each of those with two marked positions (namely $(x_1, x_2), (x'_1, x'_2)$, which are not necessarily distinct and need not to appear in that order in w, w'), they satisfy the same MSO formulas with 2 free first-order variables and quantifier depth at most k if and only if:*

- $w[x_1] = w'[x'_1]$, $w[x_2] = w'[x'_2]$
- $w[1{:}x_1) \cong_k w'[1{:}x'_1)$, $w[1{:}x_2) \cong_k w'[1{:}x'_2)$
- $w(x_1{:}|w|] \cong_k w'(x'_1{:}|w|]$, $w(x_2{:}|w|] \cong_k w'(x'_2{:}|w|]$
- $x_1 = x_2$ *and* $x'_1 = x'_2$,
  or $x_1 < x_2$, *and* $x'_1 < x'_2$ *and* $w(x_1{:}x_2) \cong_k w'(x'_1{:}x'_2)$,
  or $x_1 > x_2$, *and* $x'_1 > x'_2$ *and* $w(x_2{:}x_1) \cong_k w'(x'_2{:}x'_1)$.

Proof. The "if" part is a direct consequence of the composition theorem, the hypotheses imply that the two strings are built as a sum of some (finite) string of some k-type, then a letter (which can be seen as some rather trivial structure), then another (finite) string of some k-type, another letter, and a string of some k-type. We can also use an Ehrenfeucht-Fraïssé k-round game to show this result, the strategy of duplicator is built by composing the strategies in each of the substrings (for which he has a winning strategy for they have the same k-type).

The "only if" part is achieved by giving an explicit formula to distinguish the two strings if they are decomposed differently: the order of appearance of positions can be expressed by a quantifier free formula: they are necessarily the same; formulas holding for the substrings (and hence the k-types of the substrings) can be expressed by restricting quantifications in the formulas, which does not involve more quantifications, hence the k-types of the substrings are deduced from the formulas (with quantifier-depth k, and exactly 2 free first-order variable) validated by $(w, (x_1, x_2))$, $(w', (x'_1, x'_2))$. □

Now we are ready to present the proof of Theorem 4.9.

Proof of Theorem 4.9. Note that there are atmost $3|C|$ edges from or to nodes at position x. We will now bound edges that do not connect position x.

Given a transducer T and a string w, let us first consider the case (see Figure 4.4) of an edge from a position $y < x$ and copy c to a position $y' > x$ and copy d. For a given copy c, k-type of string $w[1, y)$ (substring u in Fig 4.4), letter $w[y]$, and k-type of string $w(y, x)$ (substring v in Figure 4.4), we show that only one such edge may exist.

For the sake of contradiction assume that there exists two distinct such edges, and denote them by $((y_a, c), (y'_a, c_a))$ and $((y_b, c), (y'_b, c_b))$. We first show that this

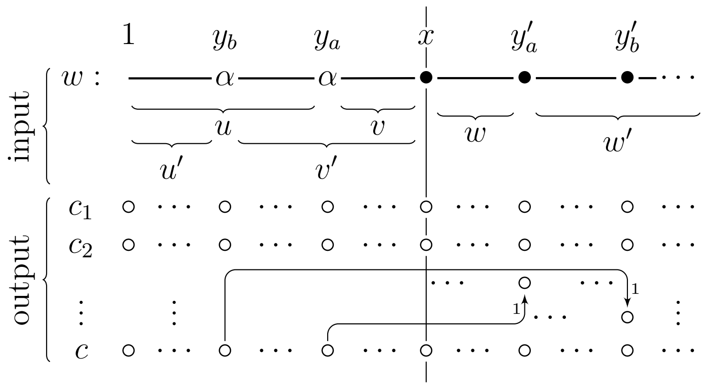


Figure 4.4: If $w[y_a] = w[y_a]$, $u \cong_k u'$, $v \cong_k v'$, and y_a and y_b are left-hand of two crossing edges, then if y'_a is a right-hand of the first crossing-edge, there is also an edge from y_b to y'_a: the output is not a tree.

assumption implies that $((y_a, y'_a), w)$ is undistinguishable from $((y_b, y'_a), w)$ by MSO formulas with 2 free first-order variable and quantifier depth k (that, we show in the next paragraph, yields a contradiction). Let us apply Theorem 4.10, see Fig 4.4, to this case: notice that $y_a < y'_a$ and $y_b < y'_a$; by assumption $w[1{:}y_a) \cong_k w[1{:}y_b)$. Now undistinguishability can be shown by exploiting monoid congruence of k-types: $w(y_a{:}x) \cong_k w(y_b{:}x)$ so $w(y_a{:}y'_a) = w(y_a{:}x) \cdot w[x] \cdot w(x{:}y'_a) \cong_k w(y_b{:}x) \cdot w[x] \cdot w(x{:}y'_a) = w(y_b, y'_a)$, thus all the hypotheses of Theorem 4.10 are satisfied hence the undistinguishability.

We have an edge from y^c_a to $y'^{c_a}_a$, so $(w, (y_a, y'_a)) \vDash \varphi^{c,c_a}_i$ (for some $i \in \{1, 2\}$), and since it was shown undistinguishable, we also have $(w, (y_b, y'_a)) \vDash \varphi^{c,c_a}_i$, this means that there are two disjoint nodes in the output mapping to the same node (with an edge with the same label), which is not possible as the output is by definition a tree (or a string). This contradiction ensures the uniqueness of crossing edges at some position once we fixed the copy of the originating node, the k-type of prefix from its position, the label of the position in the input string w and the k-type of the string between that position and the position at which the crossing occurs. Since there are $|C||\Sigma||\Theta_k|^2$ such choices for this case, and an equal number of choice for the other symmetric case when $y > x$ and $y' < x$, the theorem follows. □

Corollary 4.11. *The image of an infinite string by a string-to-tree MSO transducer with quantifier depth at most k, and $|C|$ copies, has at most $|C|(1+|\Sigma||\Theta_k|^2)$ infinite branches.*

Proof. The statement of the corollary gives a clear idea behind the proof: infinite branches will at some point yield simultaneously a crossing. However the formal proof is quite technical.

First we need to give a formal definition of the very clear informal notion of infinite branch. An infinite branch of a tree t is a word $w \in \{0, 1\}^\omega$ such that any prefix of w is in $\mathrm{dom}(t)$. A tree t has n infinite branches if there are n distinct infinite words that are infinite branches of t.

We won't detail the fact that from these n branches we can extract n prefixes incomparable w.r.t. the 2-successor order. We call the n prefixes buds (as from each one of those will shoot exactly one infinite branch).

Now we are ready to prove *ad reductio absurdum* the boundedness of the branches. Assume there exists a word w that has n infinite branches, with $n > |C|(1+|\Sigma||\Theta_k|^2)$, then there are n buds in the image of w, denote $x_1^{c_1}, \ldots, x_n^{c_n}$ the positions and copies at which they appear in the $|C|$ copies of w. Take $y > x_k$ for any $1 \leq k \leq n$. Then using the pigeon hole principle, as there are only finitely many position x^c before y, each infinite branch has nodes appearing after y. Pick one such node for each branch $(z_1^{c'_1}, \ldots, z_n^{c'_n})$. It is easy to show that $z_k > y > x_k$ implies the existence of an edge, linking two nodes below the k-th bud crossing position y in a left-to-right manner. Because those edge are each below a different bud in the image, this implies they are different, thus the n crossings edge are different. Which contradicts the boundedness of crossings at position y. □

4.3.3 MSOT $\subseteq$ SSTT$_{\text{rla}}$

In this section we exploit Theorem 4.9 to build an equivalent SSTT$_{\text{rla}}$ from an MSOT. Given an MSOT T, an input string w (for which $T(w)$ is defined), and a position x in w, we define $T(w)\downarrow_{<x}$ as a subgraph of $T(w)$, whose set of nodes is the set of all nodes y^c (corresponding to position y and copy $c \in C$) of $T(w)$, such that $y < x$. We define $T(w)\downarrow^?_{<x}$ from $T(w)\downarrow_{<x}$ as follows: for each node in $T(w)\downarrow_{<x}$, if that node had a 1-successor (resp. 2-successor) in $T(w)$ (namely y^c), but no such successor in $T(w)\downarrow_{<x}$, (which means $y \geq x$) then we add to this node a 1-successor (resp. 2 successor) that is a ?-labeled node. We say that this ?-labelled node corresponds to y^c. Intuitively, while processing a string at position x the SSTT needs to store each $T(w)\downarrow^?_{<x}$ tree in a register. To access these trees and their holes so as to perform necessary updates, we need a way to refer to them in a unique way. The following lemma gives us exactly that.

Lemma 4.12. *Let T be an MSO transducer with quantifier depth at most k, w be an input string in the domain of T, and x be a position in w. The following observations hold.*

- *For $\tau_1, \tau_2 \in \Theta_k$, $a \in \Sigma$, $c \in C$, there exists at most one tree in $T(w)\downarrow^?_{<x}$ that is rooted on a node y^c, $y < x$, and $w[y] = a$, $[w[1{:}y)]_{\cong_k} = \tau_1$, $[w(y{:}x)]_{\cong_k} = \tau_2$.*
- *For $\tau_1, \tau_2 \in \Theta_k$, $a \in \Sigma$, $c \in C$, there exists at most one ?-labeled node in $T(w)\downarrow^?_{<x}$ that corresponds to a node y^c, such that $y{>}x$, and $w[y]{=}a$, $[w(x{:}y)]_{\cong_k}{=}\tau_1$, $[w(y{:}|w|]]_{\cong_k}{=}\tau_2$.*
- *For a copy $c \in C$ there exists at most one ?-labeled node in $T(w)\downarrow^?_{<x}$ that corresponds to a node x^c.*

The trees $T(w)\downarrow^?_{<x}$ and their holes can thus be characterized uniquely by an element of $\Theta_k \times \Sigma \times \Theta_k \times C$.

Proof. This lemma is shown similarly as Theorem 4.9, assume there are to trees in $T(w)\downarrow^?_{<x}$ rooted at two positions y^c and y'^c such that $w[y] = w[y']$, $w[1{:}y) \cong_k w[1{:}y')$ and $w(y{:}x) \cong_k w(y'{:}x)$. One of these two trees has a root that has an

ancestor in $T(w)$, say a 1-ancestor (namely $z^{c'}$). Then as $(w,(y,z))$ and $(w,(y',z))$ are undistinguishable, according to Theorem 4.10, we deduce that $z^{c'}$ is also a 1-ancestor which contradicts the fact that $T(w)$ is a tree.

Similarly, if two ?-labeled nodes correspond to the same quadruple (τ_1, a, τ_2, c), then by undistinguishability, the node in $T(w)$ corresponding to one ?-labeled node should also correspond in $T(w)$ to the other ?-labeled node which once again contradicts the fact that $T(w)$ is a tree.

The case where two ?-labeled nodes correspond to the same x^c is also forbidden by the tree structure of $T(w)$.

Any ?-labeled in $T(w)\Downarrow^?_{<x}$ is thus uniquely characterized by an element of $\Theta_k \times \Sigma \times \Theta_k \times C \cup C$, as by definition of holes, it corresponds to a node occurring in $T(w)$ but not in $T(w)\Downarrow_x$ which means to a node y^c for some c and some $y \geq x$. □

We will next define registers, look-around, updates and the output function of the resulting SSTT$_{\text{rla}}$.

Registers The set of registers will be $\Theta_k \times \Sigma \times \Theta_k \times C$. During the processing of a string w, (for which $T(w)$ is defined), the register will satisfy the following invariant:

Invariant 1. While processing position x the register (τ_1, a, τ_2, c) contains the tree in $T(w)\Downarrow^?_{<x}$ (if it exists, otherwise the register is empty) that is rooted at some node y^c such that $w[y] = a$, $[w[1{:}y)]_{\cong_k} = \tau_1$, $[w(y{:}x)]_{\cong_k} = \tau_2$.

Regular look-around The set of regular look-ahead and look-behind will be K-types where $K = k + |C| + 4$ (more accurately the set, for all K-type, of the language of strings that has this K-type). So given a letter $a \in \Sigma$ and two K-types λ_b (regular look-behind) and λ_a (regular look-ahead), we need to describe the update of registers corresponding to $(\lambda_b, a, \lambda_a)$.

Update of registers Given a string w for which $T(w)$ is defined, and a position x in that string, such that $w[x] = a$, $[w[1{:}x)]_{\cong_K} \lambda_b$ and $[w(x{:}|w|]]_{\cong_K} = \lambda_a$, provided the registers satisfy Invariant 1, we will now describe how to compute $T(w)\Downarrow^?_{<x+1}$ using these registers, and store such trees in the appropriate registers such that the invariant remains preserved.

Lemma 4.13. *There exists MSO formulas with one free-variable of quantifier depth at most K that, given a string w and a position x, characterize that while processing position x*

- *whether a register is non-empty,*
- *the holes (characterized according to lemma 4.12) of the tree stored in the register, and*
- *the order these holes appear in an in-order traversal of the content of that register.*

Proof. By definition, a register (τ_1, a, τ_2, c) contains a value if there exists in $T(w)\Downarrow^?_{<x}$ a tree rooted at some node y^c, such that $y < x$, and either y^c is the root of $T(w)$ or the ancestor of y occurs at or after x. The challenge is to show that we can characterize such a y by a formula of depth bounded by K. The main difficulty is to characterize a y that satisfies the specifications that $[w[1{:}y)]_{\cong_k} = \tau_1$ and $[w(y{:}x)]_{\cong_k} = \tau_2$. We rely on the fact that a k-type can be determined by a Hintikka formula which has quantifier-depth k, and if we restrict the quantifications in this formula (that is $\forall p \cdot \varphi$ is translated in $\forall p \cdot p < y \to \varphi$) we can ensure the k-types of subwords of w. Thus the existence of such a y can be written:

$$\exists y \cdot y < x \wedge \mathscr{H}^{<y}_{\tau_1} \wedge P_a(y) \wedge \mathscr{H}^{>y,<x}_{\tau_2}$$

Notice that this formula has quantifier-depth $k+1 \leq K$. We further need to impose that such a y has an image in the copy c, and has no ancestor in $T(w)\Downarrow_{<x}$, which leads to formula $E_{(\tau_1,a,\tau_2,c)}$ in Figure 4.5, which holds at position x if and only if register (τ_1, a, τ_2, c) holds a value just before processing position x. Hence the regular look around determines uniquely which registers contain a value.

We now address the case to determine which holes a register contains. Say we want to determine whether register (τ_1, a, τ_2, c) contains the hole corresponding to $(\tau_1', a', \tau_2', c')$. For that we existentially quantify a path (which is a common thing to do in MSO) that we check is in the register (τ_1, a, τ_2, c) (that is, in a tree of $T(w)\Downarrow_{<x}$, whose root satisfies the specification imposed by the name of the register), and check that a node satisfying the specification $(\tau_1', a', \tau_2', c')$, hence appearing after x, is a direct successor in $T(w)$ of a node in the path. This is checked by the formula $H_{(\tau_1',a',\tau_2',c') \text{ in } (\tau_1,a,\tau_2,c)}$ in Figure 4.5. Determining whether the hole corresponding to $x^{c'}$ is in the register (τ_1, a, τ_2, c), is performed similarly by $H_{c' \text{ in } (\tau_1,a,\tau_2,c)}$.

Finally, to determine the relative order of appearance of those holes in the content of some register, one can notice that it just suffices to determine the relative order of appearance of these nodes in $T(w)$, which is rather straightforward to implement with an MSO formula, see for instance $G_{(\tau_1,a,\tau_2,c) \text{ then } (\tau_1',a',\tau_2',c')}$ and $G_{(\tau_1,a,\tau_2,c) \text{ then } c'}$ is Figure 4.5.

Notice that all these formulas have quantifier depth at most $K = k + |C| + 4$, hence the regular look-around indeed determines the properties stated in lemma 4.13, which concludes this proof. □

The regular look-around determines the validity of these K-depth formulas and thus describes the content of registers according to lemma 4.13. It is now straightforward to build $T(w)\Downarrow^?_{<x+1}$ and to store these trees in the register while still ensuring the invariant on the content of registers. Observe that $T(w)\Downarrow_{<x}$ is a subgraph of $T(w)\Downarrow_{<x+1}$, where $T(w)\Downarrow_{<x+1}$ has as additional nodes the x^c that are in $T(w)$ (which are determined, together with their label, by a quantifier depth $k+1$ formula, hence by $(\lambda_b, a, \lambda_a)$). The trees in $T(w)\Downarrow^?_{<x+1}$ are obtained by adding a Γ-labeled node for each of those x^c, and combining those nodes with the trees in $T(w)\Downarrow^?_{<x}$.

We describe the outgoing edges for these x^c nodes. The look-around determines which of those nodes have a successor occurring before, at x or after x. For those who have a successor after x, we will give a fresh ?-labeled node as corresponding successor, for those who have as successor a x^c, we will connect it to x^c accordingly,

$$E_{(\tau_1,a,\tau_2,c)}(x) = \exists y,\ y < x \wedge \mathscr{H}_{\tau_1}^{<y} \wedge P_a(y) \wedge \mathscr{H}_{\tau_2}^{>y,<x} \wedge \\ \left(\bigvee\nolimits_{\gamma\in\Gamma} \varphi_\gamma^c(y)\right) \wedge \forall z < x,\ \bigwedge\nolimits_{d\in C} \neg\left(\varphi_1^{d,c}(z,y) \vee \varphi^{d,c}(z,y)\right)$$

$$\mathit{ContainedIn}_{(\tau_1,a,\tau_2,c)}((X_d)_{d\in C}, x) = \\ \bigwedge_{d\in C, d\neq c}\left(\forall y \in X_d,\ y < x \wedge \bigvee_{d'\in C} \exists z \in X_{d'},\ \varphi_1^{d',d}(z,y) \vee \varphi_2^{d',d}(z,y)\right) \\ \wedge \forall y \in X_c, y < x \wedge \left((\mathscr{H}_{\tau_1}^{<y} \wedge P_a(y) \wedge \mathscr{H}_{\tau_2}^{>y,<x}) \vee \bigvee_{d\in C} \exists z \in X_d,\ \varphi_1^{d,c}(z,y) \vee \varphi_2^{d,c}(z,y)\right)$$

$$H_{c' \text{ in } (\tau_1,a,\tau_2,c)}(x) = \\ (\exists X_d)_{d\in C}\ \mathit{ContainedIn}_{(\tau_1,a,\tau_2,c)}((X_d)_{d\in C}), x) \wedge \bigvee_{d\in C} \exists y \in X_d,\ \varphi_1^{d,c'}(y,x) \vee \varphi_2^{d,c'}(y,x)$$

$$H_{(\tau_1',a',\tau_2',c') \text{ in } (\tau_1,a,\tau_2,c)}(x) = (\exists X_d)_{d\in C}\ \mathit{ContainedIn}_{(\tau_1,a,\tau_2,c)}((X_d)_{d\in C}), x) \\ \wedge \exists z,\ z > x \wedge \mathscr{H}_{\tau_1'}^{>x,<z} \wedge P_{a'}(z) \wedge \mathscr{H}_{\tau_2'}^{>z} \wedge \bigvee_{d\in C} \exists y \in X_d,\ \varphi_1^{d,c'}(y,z) \vee \varphi_2^{d,c'}(y,z)$$

$$\mathit{Subtree}_{i,c_s,c_e}(x_s,x_e) = \\ \varphi_i^{c_s,c_e}(x_s,x_e) \vee (\exists X_d)_{d\in C}\ \left(\bigvee_{d\in C} \exists y \in X_d,\ \varphi_1^{d,c_e}(y,x_e) \vee \varphi_2^{d,c_e}(y,x_e)\right) \\ \wedge \bigwedge_{d\in C} \forall y \in X_d,\ \left(\varphi_i^{c_s,d}(x_s,y) \vee \bigvee_{d'\in C} \exists z \in X_{d'},\ \varphi_1^{d',d}(z,y) \vee \varphi_2^{d',d}(z,y)\right)$$

$$G_{(\tau_1,a,\tau_2,c) \text{ then } (\tau_1',a',\tau_2',c')}(x) = \\ \bigvee_{d\in C} \exists p,\ \left(\exists y > x,\ (\mathscr{H}_{\tau_1}^{>x,<y} \wedge P_a(y) \wedge \mathscr{H}_{\tau_2}^{>y}) \wedge \mathit{Subtree}_{1,d,c}(p,y)\right) \\ \wedge \left(\exists y > x,\ (\mathscr{H}_{\tau_1'}^{<x,>y} \wedge P_{a'}(y) \wedge \mathscr{H}_{\tau_2'}^{>y}) \wedge \mathit{Subtree}_{2,d,c'}(p,y)\right)$$

$$G_{(\tau_1,a,\tau_2,c) \text{ then } c'}(x) = \\ \bigvee_{d\in C} \exists p,\ \left(\exists y > x,\ (\mathscr{H}_{\tau_1}^{>x,<y} \wedge P_a(y) \wedge \mathscr{H}_{\tau_2}^{>y}) \wedge \mathit{Subtree}_{1,d,c}(p,y)\right) \wedge \mathit{Subtree}_{2,d,c'}(p,x)$$

Figure 4.5: MSO queries to determine how to update the registers. $\mathscr{H}_\tau^{<x}$ denotes the Hintikka formula for τ with restricted quantification.

and for those who have successor before x, we will give as corresponding successor the tree of $T(w)\Downarrow^{?}_{<x}$ whose root is that successor. A $k+1$ depth formula determines that register (as the characterization of the root is the name of the register).

Now for the incoming edges, notice that once again the regular look-around determines where the ancestor occurs w.r.t. x. The case where the predecessor occurs after x leads to no incoming transition (hence we have a tree rooted at some x^c), the case where the predecessor is some $x^{c'}$ has been taken care of already, and for the case where the predecessor occurs before x, we insert this tree rooted at x^c instead of the ?-labeled node corresponding to x^c.

Finally we now detail how to store those trees appropriately in the registers. For those trees which are rooted at some x^c, we put them in the corresponding register $(\tau_b, w[x], [\varepsilon]_{\cong_k}, c)$ (where τ_b is the uniquely determined k-type of λ_b). A tree which was in register (τ_1, a', τ_2, c) whose root (namely y^c) was not connected to its parent, is relocated in register $(\tau_1, a', \tau_2 \cdot [w[x]]_{\cong_k}, c)$. Notice that $\tau_2 \cdot [w[x]]_{\cong_k}$ indeed corresponds to the k-type of $w(y{:}x] = w(y{:}x+1)$. This update of registers we described here can be encoded as a substitution as described in the previous section, and this substitution is copyless.

The output function At each step during the processing, the regular look-around tells us whether the root of the image is in the prefix of the position we are processing (this property being expressible as a quantifier depth $k+1$ MSO formula). If it is, we just output the corresponding register. Thus at each step we output the tree of $T(w)\Downarrow^{?}_{<x}$ that contains the root of $T(w)$, this sequence of trees necessarily converges towards the image $T(w)$.

The construction is now complete. We now have an explicit construction of an SSTT with regular look-around implementing an MSO transducer, which proves Theorem 4.8. We now have to show that we can remove regular look-around by introducing finitely many states, and allowing restricted copy. The rest of the section is dedicated to removing regular look-around.

4.3.4 Closure Under Regular Look-Around

In this section we show that SSTT are closed under regular look-around by showing the following theorem. The other direction is trivial, and with some effort, one easily shows that the syntactic insurance of convergence of output can be achieved in the $\mathrm{SSTT}_{\mathrm{rla}}$ from the syntactic insurance of convergence in the SSTT.

Theorem 4.14 ($\mathrm{SSTT}_{\mathrm{rla}} \subseteq \mathrm{SSTT}$)**.** *Every* $\mathrm{SSTT}_{\mathrm{rla}}$*-definable transformations is* SSTT*-definable.*

We prove this theorem in two parts over the next two subsections. In the first subsection we show how to remove the regular look-ahead from the update function at the cost of introducing states and relaxing copylessness to restricted copy. Note that this step is enough for SSTTs on finite strings. For infinite strings we need to also remove the regular look-ahead in the output function. In the second subsection we show how to shift from output guarded by regular look-ahead to output determined by the set of infinitely occurring transitions.

4.3.5 Removing Look-around from Register Updates

Regular look-behind can be easily removed by introducing states in the $\mathrm{SSTT}_{\mathrm{rla}}$. In the state-free $\mathrm{SSTT}_{\mathrm{rla}}$ the set of regular look-behind is essentially the set of K-types. Notice that the set of K-types covers the set of finite strings and is a finite monoid. This allows to build a finite state-transition structure, each state of which corresponds to a K-type and can be reached from initial state by strings with the same K-type. In any such finite state-transition structure, the current state stores all necessary information about the regular look-behind.

We now show how to remove regular look-ahead from the registers updates. We synchronize registers with all possible regular look-aheads: The set of registers is the cartesian product of registers in the $\mathrm{SSTT}_{\mathrm{rla}}$ with the set of K-types of ω-strings.

Given a configuration (q, ν) of a run over a string w, the register (r, λ) (r denoting some register in the $\mathrm{SSTT}_{\mathrm{rla}}$, λ some K-type) is meant to contain what r would contain should the regular look-ahead be λ. When processing some letter a (in some state q), it is updated similarly as r would (with regular look-behind corresponding to q and regular look-ahead λ) from the registers with second component being $a{\cdot}\lambda$. Formalizing this syntactic trick, if the update function in the one-state transducer with regular look-around is ρ, and we define ρ' as: $\rho'(q, a, (r, \lambda)) = \rho(\lambda_b, a, \lambda, r)[r_i/(r_i, a{\cdot}\lambda)]$ (we substitute each register r_i by $(r_i, a{\cdot}\lambda)$) where λ_b denote the K-type of strings reaching state q.

Removing look-ahead in this fashion introduces copies in the register updates, since a look-ahead λ can be obtained by prepending same letter to more than one K-type. However such register update follows restricted copy rule as it is straightforward to see that any two registers with different K-type component conflict: given a K-type, all registers with that K-type will be updated in a copyless manner with the values of registers all with the same K-type.

4.3.6 Removing Look-around from Output function

In this section we detail how to shift from look-ahead based output to an output based on Muller condition. Removing look-ahead based output in case of finite string setting—where regular look-ahead are K-types of finite strings—is trivial. In this case while processing the last letter of the input string, we precisely know that the regular look-ahead is the K-type of the empty string, and hence we can (for each state, that is each possible regular look-behind) output the corresponding register $(r, [\varepsilon]_{\cong_K})$. In the case of infinite strings, we do not have such a privileged position where we know the regular look-ahead, furthermore we need to output an infinite sequence of trees converging toward the image. So we need to correctly guess infinitely often the regular look-ahead, that needs to be correct eventually always, so that we output a sequence of trees that converges towards the image.

The rest of this section is dedicated to show that by introducing more states and registers, we can guess correctly and infinitely often the regular look-ahead from the set of infinitely occurring transitions. We use the concept of merging relation [She75] to correctly guess such look-ahead.

Definition 4.15 (Merging Relation)**.** Given a string $w{\in}\Sigma^\infty$ and $K{\in}\mathbb{N}$, we say that two positions x and y *merge* at some position z (written $x{\sim}^w_K y(z)$, we thereafter omit K and w) if $w(x{:}z] \cong_K w(y{:}z]$. We write $x \sim y$ if x and y merge at some position.

Proposition 4.16. *Let $\sim$ be the merging relation.*

- *$(\sim(z))_{z\in w}$ and $\sim$ are equivalence relations of index bounded by the number of K-types (of finite strings).*
- *If w is an ω-string and $x \sim y$ then $w(x{:}\infty] \cong_K w(y{:}\infty]$.*

Proof. The reflexivity of these relations is clear by definition, the transitivity can be shown relying on the K-types being a monoid congruence: if $x \sim y(z)$, then $x \sim y(z')$ for any position z' after z.

From this remark, we can easily deduce the bound on the index for finite strings: two positions merge if and only if they merge at the last position of the string, each equivalence class is characterized by a unique K-type corresponding to the K-type of suffixes from those positions.

For an infinite string, we show that for any finite set of positions, the number of merging equivalence classes is bounded by the number of K-types: take a prefix containing all these positions, merging in that prefix refines (implies) merging in the infinite string, which gives us the bound for any finite set of positions, hence the number of merging equivalence classes in the infinite string is bounded by the number of K-types. □

Given an infinite string, there is an equivalence class that has infinitely many representatives, and at all these positions the (K-type of the) regular look-ahead is the same. We need to determine those positions so that we can output the value of the register associated to this regular look-ahead K-type.

Tracking these merging equivalence classes

We will define a (Σ-labeled) transition system which tracks the following properties: the number of equivalence classes of merging at current position, and for each of those equivalence classes, the K-type of the string between the first occurring representative of any two of those equivalence classes. We can store this information in a triangular matrix with values in the set of K-types (of finite strings), of size bounded by the number of K-types (of finite strings). We choose thereafter to formally represent this "triangular matrix" by a square matrix with $|\Theta_K|$ rows and columns, and with values in the set of (finite-string) K-types with an additional dummy symbol, say $\perp$.

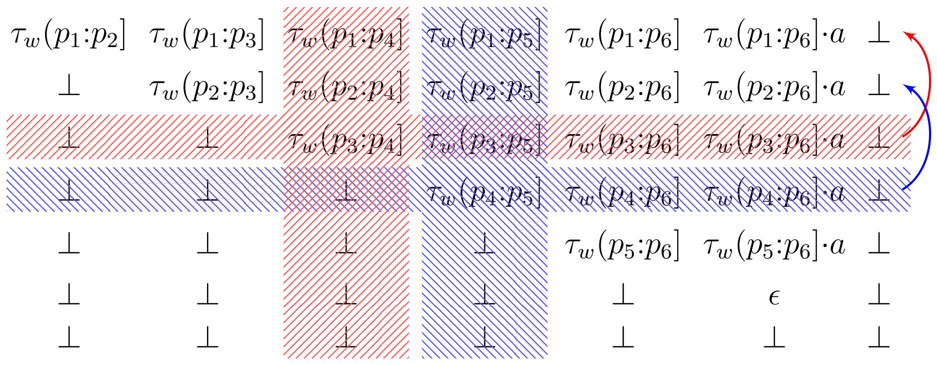
Given an infinite string w, a position x, let n denote the number of equivalence classes of $\sim(x)$, let $i < j \leq n$, let p_i denote the first occurrence in w of the i-th appearing equivalence class of $\sim(x)$, and p_j the first occurrence of the j-th appearing equivalence class. The element at position (i,j) in this matrix, is the K-type of $w(p_i{:}p_j]$. The rest of the matrix is filled with $\perp$. This construction is shown in Figure 4.6.

$$\begin{array}{ccccccc}
\tau_w(p_1{:}p_2] & \tau_w(p_1{:}p_3] & \tau_w(p_1{:}p_4] & \tau_w(p_1{:}p_5] & \tau_w(p_1{:}p_6] & \bot & \bot \\
\bot & \tau_w(p_2{:}p_3] & \tau_w(p_2{:}p_4] & \tau_w(p_2{:}p_5] & \tau_w(p_2{:}p_6] & \bot & \bot \\
\bot & \bot & \tau_w(p_3{:}p_4] & \tau_w(p_3{:}p_5] & \tau_w(p_3{:}p_6] & \bot & \bot \\
\bot & \bot & \bot & \tau_w(p_4{:}p_5] & \tau_w(p_4{:}p_6] & \bot & \bot \\
\bot & \bot & \bot & \bot & \tau_w(p_5{:}p_6] & \bot & \bot \\
\bot & \bot & \bot & \bot & \bot & \bot & \bot \\
\bot & \bot & \bot & \bot & \bot & \bot & \bot
\end{array}$$

(a) before processing position x; $\tau_w[p_i{:}p_j)$ denotes the K-type of $w[p_i{:}p_j)$

$$\begin{array}{ccccccc}
\tau_w(p_1{:}p_2] & \tau_w(p_1{:}p_3] & \tau_w(p_1{:}p_4] & \tau_w(p_1{:}p_5] & \tau_w(p_1{:}p_6] & \tau_w(p_1{:}p_6]{\cdot}a & \bot \\
\bot & \tau_w(p_2{:}p_3] & \tau_w(p_2{:}p_4] & \tau_w(p_2{:}p_5] & \tau_w(p_2{:}p_6] & \tau_w(p_2{:}p_6]{\cdot}a & \bot \\
\bot & \bot & \tau_w(p_3{:}p_4] & \tau_w(p_3{:}p_5] & \tau_w(p_3{:}p_6] & \tau_w(p_3{:}p_6]{\cdot}a & \bot \\
\bot & \bot & \bot & \tau_w(p_4{:}p_5] & \tau_w(p_4{:}p_6] & \tau_w(p_4{:}p_6]{\cdot}a & \bot \\
\bot & \bot & \bot & \bot & \tau_w(p_5{:}p_6] & \tau_w(p_5{:}p_6]{\cdot}a & \bot \\
\bot & \bot & \bot & \bot & \bot & \epsilon & \bot \\
\bot & \bot & \bot & \bot & \bot & \bot & \bot
\end{array}$$

(b) adding an extra column ($w[x] = a$)



τ_w(p_1:p_2] τ_w(p_1:p_3] τ_w(p_1:p_4] τ_w(p_1:p_5] τ_w(p_1:p_6] τ_w(p_1:p_6]·a ⊥
⊥ τ_w(p_2:p_3] τ_w(p_2:p_4] τ_w(p_2:p_5] τ_w(p_2:p_6] τ_w(p_2:p_6]·a ⊥
⊥ ⊥ τ_w(p_3:p_4] τ_w(p_3:p_5] τ_w(p_3:p_6] τ_w(p_3:p_6]·a ⊥
⊥ ⊥ ⊥ τ_w(p_4:p_5] τ_w(p_4:p_6] τ_w(p_4:p_6]·a ⊥
⊥ ⊥ ⊥ ⊥ τ_w(p_5:p_6] τ_w(p_5:p_6]·a ⊥
⊥ ⊥ ⊥ ⊥ ⊥ ϵ ⊥
⊥ ⊥ ⊥ ⊥ ⊥ ⊥ ⊥

(c) as $[w(p_2{:}p_6] \cdot a]_{\cong_K} = [w(p_4{:}p_6] \cdot a]_{\cong_K}$ and $[w(p_1{:}p_6] \cdot a]_{\cong_K} = [w(p_3{:}p_6] \cdot a]_{\cong_K}$, we cross out the third and fourth rows and columns

$$\begin{array}{ccccccc}
\tau_w(p_1{:}p_2] & \tau_w(p_1{:}p_3] & \tau_w(p_1{:}p_6] & \tau_w(p_1{:}p_6]{\cdot}a & \bot & \bot & \bot \\
\bot & \tau_w(p_2{:}p_3] & \tau_w(p_2{:}p_6] & \tau_w(p_2{:}p_6]{\cdot}a & \bot & \bot & \bot \\
\bot & \bot & \tau_w(p_5{:}p_6] & \tau_w(p_5{:}p_6]{\cdot}a & \bot & \bot & \bot \\
\bot & \bot & \bot & \epsilon & \bot & \bot & \bot \\
\bot & \bot & \bot & \bot & \bot & \bot & \bot \\
\bot & \bot & \bot & \bot & \bot & \bot & \bot \\
\bot & \bot & \bot & \bot & \bot & \bot & \bot
\end{array}$$

(d) after processing position x

Figure 4.6: Detailed update of the matrix

We now detail how to update this matrix (namely M) by processing the next position, i.e. $x+1$, which is done in two steps: first we "add" a new column, and then we shrink the matrix as some equivalence classes can merge when we process this letter.

We first put values in the $n+1$-th column: $M(i, n+1) = M(i,n)\cdot w[x+1]$, for any $i < n$; and $M(n, n+1) = [\varepsilon]_{\cong_K}$. For the latter case, we rely on the fact that as $K > 1$, ε is a K-type on its own, hence position x is alone in its $\sim (x)$ equivalence class, so it is the last occurring $\sim (x)$ equivalence class, and $x+1$ is obviously related to x by ε. For the other cases, we just built $[w(p_i{:}x+1]]_{\cong_K}$ from $[w(p_i{:}x]]_{\cong_K}$.

We then "shrink" our matrix, the idea is as follows: if two distinct $\sim (x)$ equivalence classes (i-th and i'-th) merged into a single $\sim (x+1)$ equivalence class, we want to remove the track of the newest (i.e. the i'-th if $i' > i$), we do so by "crossing out" in the matrix the i'-th row and column; formally the entry (x, y) is now entry (x', y') with $x' = x$ (resp. $y' = y$) if $x < i'$ (resp $y < i'$), $x' = x+1$ (resp. $y' = y+1$) if $x \geq i'$ (resp. $y \geq i'$). By definition two $\sim (x)$ equivalence classes (i and i') have merge into one $\sim (x+1)$ equivalence class if and only if $M(i, n+1) = M(i', n+1)$. We argue that, thanks to additivity of K-types, all the information held by the matrix is contained by the $M(i, i+1)$, for $i < |\Theta_K|$, as $M(i,j) = M(i,i+1) + \ldots + M(j-1, j)$. The following lemma states that look-ahead guessed using transitions in our matrix state-transition system are correct.

Lemma 4.17. *The set of infinitely fired transitions when running an infinite string in this transition system gives us the number of $\sim$-equivalence classes in this string. Furthermore, for each $\sim$-equivalence class, it gives us the K-type of the look-ahead at positions in this equivalence class.*

Proof. There is a transition that merges in this set of infinitely often firing transitions, indeed, each transition that does not merge increases by one the number of tracked representative of equivalence classes; as the number of tracked equivalence classes is bounded, this cannot happen. Let us denote m the smallest index of a column that is discarded in this set of transitions: $m-1$ is the number of equivalence classes.

Assume there are m $\sim$-equivalence classes in the infinite string. Let us call e_m the first position of the last appearing $\sim$-equivalence class (the m-th). There is some position (called t) where any two $\sim$-equivalent position before position e_m have merged. Hence at position t, the first m column in the matrix correspond to the first element of each of the m $\sim$-equivalence classes (which are seen as $\sim (t)$-equivalence classes). After this threshold no two of the m first column can merge. If the m-th column is crossed out (that is merged into another i-th column, $i < m$) infinitely often such a threshold does not exist which means that there are not as many as m $\sim$-equivalence classes. If the m-th column is not crossed out infinitely often (that is no infinitely occurring transition crosses out this column) this means that there are at least m $\sim$-equivalence classes.

Not only does the set of infinitely occurring transitions give us the number of $\sim$-equivalence classes, but it also gives us the K-type of the regular look-ahead from each position in any equivalence class.

Take one infinitely occurring merging transition which has a minimal index of crossed out column (among the set of infinitely occurring transitions). Say it merges column m into column $m' < m$, then we know that the K-type of the regular look-ahead from any position in the m'-th equivalence class is $(M(m,m'))^\omega$.

Let us define inductively a factorization of the suffix from $e_{m'}$: let $x_0 = e_{m'}$ the first element in the m'-th appearing $\sim$-equivalence class. Let us denote t_n the position where x_n has merged in the m'-th column (that is $e_{m'} \sim x_n(t_n)$). x_{n+1} is the first position that appears after t_n and which will merge through the m-th column in the m'-th, it is guaranteed to exist as there are these two columns are merged infinitely often. The matrix told us that $[w[e_{m'}{:}x_{n+1}]]_{\cong_K} = M(m',m)$, as x_{n+1} appeared after $e_{m'}$ and x_n had merged, we deduce $[w(x_n{:}x_{n+1}]]_{\cong_K} = M(m',m)$. The infinite sequence of (x_n) gives us a factorization of $w(e_{m'}{:}\infty]$, which proves its K-type to be $M(m',m)^\omega$.

We can similarly deduce that for any $m'' < m$, ($e_{m''}$ denoting the first position in the m''-th occurring $\sim$-equivalence class) $[w(e_{m''}{:}\infty]]_{\cong_K} M(m'',m')(M(m',m))^\omega$. □

From the set of infinitely occurring transitions, we therefore deduce the K-type of the regular look-ahead from the equivalence classes, but we only know the equivalence class once it has merged, so we will need to somehow delay the output of our transducer: at each step we "pre-output" some tree according to a guess of the regular look-ahead, and effectively output it when the corresponding position has merged with the equivalence class. We will show how to achieve that by delaying output.

Delaying output

We present the details of outputting for a given set S of infinitely occurring transitions. This introduces finitely many new registers, among which a distinguished output register for this set S. The other cases of infinitely occurring transitions can be handled simultaneously. As S is a set of infinitely occurring transitions, it forms a strongly connected component, so it contains a merging transition and therefore there is an integer m such that m is the lowest index of column that get erased by merging. Let us denote $t \in S$ such a transition that merges column m into some column $m' < m$), this gives us the K-type (namely e^ω) of the regular look-ahead from any position in the m' $\sim$-equivalence class.

We introduce a set of $(|\Theta_K|-m)$ registers $(r_{S,m}, \dots, r_{S,|\Theta_K|})$ that will track output candidates, and the output register r_S for the set S. When processing position x in a string w, that fired a transition $t' \in S$, denote n the number of columns of the state just after t'. This transition will have the following behavior on the $r_{S,i}$:

- In $r_{S,n}$ we put the tree that should be output just after processing x, would the regular look-ahead be e^ω: we introduce a new output candidate, and we will show that for this candidate, we will be able to say whether the guess for the regular look-ahead was correct or not.

- In $r_{S,i}$ for $m \le i < n$ we put the older content of $r_{S,j}$ where j corresponds to the index of the column that was mapped to the i-th by the transition (or the greatest index of the column that was merged if some columns were merged into the i-th).

Hence each register $r_{S,i}$ (for any i smaller than the number of columns in current state) contains the tree that would be output by the SSTT$_{\mathrm{rla}}$ if the regular look-ahead was e^ω at position y where y is the last position in the i-th $\sim(x)$-equivalence class.

We finally detail how we can distinguish among our guesses which ones were correct, and should therefore be placed in the output register $r_S = F(S)$. We obtain this information when processing transition t, we know that the content of $r_{S,m}$ contains the value of the tree output by the SSTT$_{\mathrm{rla}}$ (would the regular look-ahead be e^ω) at the last position in the m-th $\sim(x-1)$ equivalence class. As the m-th $\sim(x-1)$ equivalence class has merged into the $m' \sim(x)$-equivalence class with this transition t, if the m'-th equivalence class of merging up to current position equivalence class is already the m'-th $\sim$-equivalence class (which happens eventually) then the guess was correct, though delayed.

This construction yields from an SSTT$_{\mathrm{rla}}$ an SSTT that will output an infinite subsequence of the sequence of output produced by the SSTT$_{\mathrm{rla}}$. We will sketch how we can furthermore, by adding finitely more registers, syntactically ensure the convergence of the output register of the SSTT. The idea is to memorize, how was updated the content of the register from its previous value: In addition of each $r_{S,i}$, we take n registers (n being the number of holes in trees in r_S, which can be determined), each which containing what was appended to the corresponding hole in the older content of r_S. If a transition outside S is fired, these new registers won't be considered until t is fired. When t was fired, as long as we only fire transitions in S, we update not the $r_{S,i}$, but the corresponding n registers. Finally, instead of updating r_S as the content of $r_{S,m}$, we update it by appending the n trees in the n registers corresponding to $r_{S,m}$. Which yields a syntactic insurance of convergence of the output.

4.4 Decision Problems

The computational model of SSTT yields as primary result, the computability in linear time of the image of an input word, as it is processes in a single pass the input [4].

We will show in this section, that likewise finite state automata, this computational model also yields some other decision procedure, which are much more efficient than in the logical framework, as the logical definition consists of MSO formulas for which the decision procedure is non-elementary.

4.4.1 Functional Equivalence Problem

The automata model allows to check efficiently equivalence: it is quasi-linear [HU79] for DFA, and exponential [Saf88] for Büchi automata. The equivalence checking for DFA relies on the existence of a canonical minimal automaton, whereas for the Büchi automaton it goes through the construction of a Büchi automaton accepting

4. We need to make the reasonable consideration that any register term from a finite fixed set be evaluted in constant time

$$\begin{array}{lllllllllll}
w: & a & b & b & a & c & b & a & b & \ldots \\
T_{\neg 1}(w) & & b & b & a & c & b & a & b & \ldots \\
T_{\ll}(w) & b & b & a & c & b & a & b & \ldots &
\end{array}$$

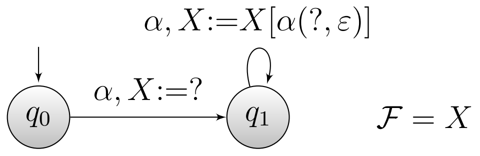


Figure 4.7: Transducers $T_{\neg 1}$ and $T_{\ll}$ implementing the same transformation.

the symetric difference of the two languages: emptiness of the automaton is then checked.

The corresponding problem for MSOT is checking functional equivalence: given two transformations, do they produce the same output on any input. There is as of now no notion of canonical form for SSTTs, minimizing an SSTT is challenging, as one can usually decrease the number of registers at the cost of introducing more states and conversely. However, one can hope for characterizing the minimal number of registers, in a similar manner as in [AR13], where they derive a bound in the case of finite string to finite string over a unary alphabet transformations.

Hence, the proof presented here is conceptually much closer to the proof of equivalence checking for Büchi automata, by describing a counter machine (where only incrementing the counters is allowed), that recognizes the language over which two SSTTs differ.

A difficulty arises from the fact that two non equivalent logical definitions can define the same function. In Figure 4.7, although both transducers $T_{\neg 1}$ and $T_{\ll}$ implement equivalent transformations, they differ in their logical characterization. Transducer $T_{\neg 1}$ does not modify the labels of the nodes but excludes the first one from the output, while $T_{\ll}$ shifts all the labels to the left. The SSTT for $T_{\ll}$ will output at each step one string which will always "lag" one symbol from the image up to the current position. This lag is not necessarily bounded, and it is challenging to ensure that, despite such delay, the successive outputs converge towards the image by the transducer.

Nevertheless equivalence of string-to-string transformations is decidable [Cou94]. We will use the computational model of SSTTs to extend this proof to the infinite string-to-tree case. That is given two SSTTs M_1 and M_2, one can decide whether for any word w $M_1(w) = M_2(w)$. The case of finite input leads to a finite output, and it has been shown that for these finite cases this functional equivalence problem can be solved in NEXPTIME [AD12], essentially by guessing non deterministically the index (in the in-order traversal) where two output trees differ. This problem is reduced to checking in a non-deterministic two-counter system (where counters can only be incremented) the existence of a reachable configuration where the two counters hold the same value. This counter system inputs a word[5] and tracks a conflicting position in each image: each counter stores the number of symbols produced so far (i.e. contained in some register at that point during the processing by the streaming transducer) that appear before the tracked position in each streaming transducer. The equality between the counters at the end ensure the same position was tracked.

5. then labels over transitions are erased so that the conflicting word is also non-deterministically guessed by the counter system

As in the infinite case, the output trees can have infinite branches the positions in the trees cannot be indexed as easily as their index of occurrence in the in-order traversal. We will rely on the fact that the number of infinite branches is bounded (Corollary 4.11) to characterize a conflicting position. We will non-deterministically guess not only a conflicting position in two outputs but also the places where to chop the infinite branches so that we can characterize this conflicting position in the in-order traversal of those two identically trimmed trees.

Definition 4.18. Given a tree t, and an integer p, such that $p \leq |\mathrm{dom}(t)|$, let us denote $\#_p(t)$ the p-th occurring node in t in its preorder traversal, we denote $\sigma\chi_p(t)$ the subtree of t such that exactly every successor of $\#_p(t)$ has been removed.

We denote $\sigma\chi_{p_1,\dots,p_k}(t)$ for $\sigma\chi_{p_k}(\sigma\chi_{p_{k-1}}(\dots\sigma\chi_{p_1}(t)))$.

Lemma 4.19. *Given two trees t_1 and t_2, with at most k infinite branches, $t_1 \neq t_2$ if and only if there exists $p_1, \dots, p_k, p_c \in \mathbb{N}$ such that $t'_1 = \sigma\chi_{p_1,\dots,p_k}(t_1)$ and $t'_2 = \sigma\chi_{p_1,\dots,p_k}(t_2)$ differ at position p_c, that is*

- *either their shape differs around p_c, $\#_{p_c}(t'_1) \cdot a \in \mathrm{dom}(t'_1)$ and $\#_{p_c}(t'_2) \cdot a \notin \mathrm{dom}(t'_2)$ (for some $a \in \{1, 2\}$) or vice-versa,*
- *or the label of $\#_{p_c}$ differ $t'_1(\#_{p_c}(t'_1)) \neq t'_2(\#_{p_c}(t'_2))$.*

Exploiting Lemma 4.19, the proof [AD12] of functional equivalence problem for finite string-to-tree case yield the proof for the following result.

Theorem 4.20. *Given two SSTTs M_1 and M_2, the problem of checking whether $[\![M_1]\!] \neq [\![M_2]\!]$ is decidable.*

Proof. Two SSTTs are not equivalent if either for some input has an image through only one of the two SSTTs or there exists an input w such that $M_1(w) \neq M_2(w)$. We find if it exists such a counterexample thanks to lemma 4.19.

We sketch a reduction of this problem very similarly as in [AD12], except that we also guess the positions $p_1, \dots, p_k$ where to trim the tree. Therefore we can reduce this equivalence problem to checking in a nondeterministic $2(k+1)$-counters $(c_1, \dots, c_k, c_c, c'_1, \dots, c'_k, c'_c)$ system the reachability of a configuration in which each pair (c_i, c'_i) of counters has the same value.

If the two transducers are inequivalent, the counters (c_i, c'_i) will eventually contain the integers p_i that are exhibited by a counterexample.

The set of states will be a product of the set of states of the two SSTTs and also track the number of holes in each register and what is the relative position of the content of each register w.r.t. to these positions $p_1, \dots, p_k, p_c$. There are many possible cases some of them are depicted in Figure 4.8, for starters the content of the register might not appear in the output, or its root can be a successor of a position p_i, or a position p_i can be on the path to a hole (as p_3 in Figure 4.8), or “between” two holes (as p_2), or be the successor of some hole (as p_4), or appear “before” (as p_1, not “above”) or “after” (as p_5, p_c) the content of the register.

We non-deterministically make all possible choices, when a term of the form $a(e_1, e_2)$ (this a can correspond to a position p_i or not, notice that in the case of position p_c one must also ensure the divergence at this position in the two output

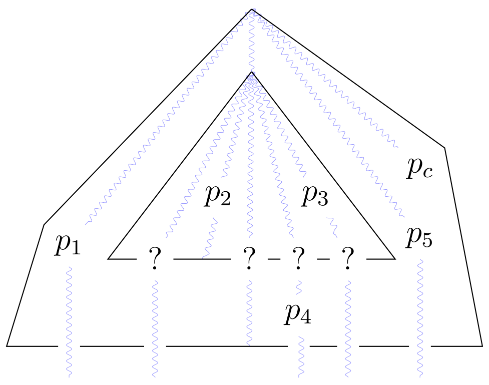


Figure 4.8: Tracking the positions of p_i w.r.t. registers.

trees, for example it was a $a(e_1, e_2)$ in M_1 and $b(e_1, e_2)$ in M_2) each corresponding positions in the two trees need not to appear on the same transition. Then we check that the combinations of registers is sound (a register appearing above some position cannot be inserted in a hole of a register appearing below that position; a register whose content does not appear in the image cannot be inserted into a register whose content is). Also for each term of the form $a(e_1, e_2)$ in the update we also need to increment the value of the corresponding counters (e.g. none if it appears below a p_i or all those corresponding to positions appearing below of after it).

If one can reach a configuration in which one output register for each SSTT, is such that each hole is below a position (meaning counter won't get incremented anymore) , and the values of each counter pairwise agrees with those in the others, then one has found an input word on which the two SSTTs produce different output. This is guaranteed to exist if the two SSTTs produce a different output on a same word. □

4.4.2 Typechecking

We can easily remark, that MSOTs do not preserve regularity: if the input language is a regular language, then its image by some transformation T is not necessarily regular, nor even context-free, e.g. we can easily express the transformation that produces $a^n b^n c^n$ where n denotes the length of the input.

However, from the logical definition of those transformations typechecking which consists in deciding whether the image of a given regular language is in a given regular language can easily be decided. We will show here that we can make use of the SSTTs and the automata describing these languages to decide this problem rather than performing some MSO model-checking (which would yield a non-elementary lower bound). We will reduce this typechecking problem to emptiness of some Büchi automaton, whose states will hold some regular properties over the content of registers.

Theorem 4.21. *Typechecking of SSTTs is decidable by an automatic construction*

Proof. Given an SSTT T, a Muller automaton M and a Rabin automaton R, one can decide whether $T(L(M)) \subseteq L(R)$.

With a finite case decomposition, we can consider, that any word $w \in L(M)$ visits the same set of edges in T infinitely often. We build a Büchi automaton as a product of T and M, and we also embbed in each state some information of the content of each register, as what is the state reached by its content in R (and what are the states reached by nodes in the content of the register) for each distribution of states of R for each of its holes. Finally, we make final only states for which the content of the output register is a final state of R. It finally remains to check that the infinite branches satisfy the Rabin condition. We need to check that exactly states that should appear infinitely often on the infinite branches are present in the registers that get appended in the holes of the output registers. □

4.5 Conclusion

In this chapter we detailed a rather direct reduction from MSO transducers to streaming string transducers, we only went through the intermediate model of streaming string transducer with regular look-around. Streaming string transducers were known to capture MSO transducers expressiveness in some restricted cases [AČ10, AFT12] (from finite strings to string and from infinite strings to trees), but the proofs went through many intermediate models. This reduction therefore gives a more direct proof for those known results. Furthermore the decidability of functional equivalence for MSO string to tree transformations is established through this computational model.

A key feature of our reduction is that it only exploits logical properties of MSO, such as the finiteness of k-types and the composition theorem, to yield a computational model. Thanks to this feature, it is immediate that this proof can be adapted with some efforts to establish a computational model (an appropriate star-free restriction of SSTTs) for first-order definable transformations. More ambitiously perhaps, our reduction can provide a framework for showing reductions from more expressive logic based transducers (symbolic transducers) to corresponding computational models, given an analog for the finiteness of k-types and the composition theorem.

The syntactic restriction on the update of registers in streaming string transducers is crucial to ensure a linear output, otherwise they could implement transformations that could not be expressed by MSO transducers. This restriction however poses a challenge to extend this model to capture MSO string to graph transformations as the number of edges between the nodes can be quadratic w.r.t. the number of nodes and hence the size of the input. However streaming string transducer can be relaxed to produce a quadratic output: notice that if we suppress the restriction on the update of registers we can even have exponential output. This restriction is crucial to have no more expressive power than MSO transformations. However relaxing it on edges but not on nodes might lead to a streaming string transducer model that captures exactly MSO-definable string to graph transformations.

However this restriction to tree output paves the way for extending the theory of regular cost functions [ADD$^+$13] to associate costs to infinite strings: associating as image of the input word not the tree, but an evaluation of that tree.

Bibliography

[AČ10] Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In *FSTTCS*, volume 8, pages 1–12, 2010.

[AČ11] Rajeev Alur and Pavol Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *POPL*, pages 599–610, 2011.

[AD12] Rajeev Alur and Loris D'Antoni. Streaming tree transducers. In *ICALP (2)*, volume 7392 of *Lecture Notes in Computer Science*, pages 42–53. Springer, 2012.

[ADD$^+$13] Rajeev Alur, Loris D'Antoni, Jyotirmy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular Functions and Cost Register Automata. In *LICS*, pages 13–22. IEEE Computer Society, 2013.

[ADGT13] Rajeev Alur, Antoine Durand-Gasselin, and Ashutosh Trivedi. From Monadic Second-Order Definable String Transformations to Transducers. In *LICS*, pages 458–467. IEEE Computer Society, 2013.

[AFT12] Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *LICS*, pages 65–74. IEEE Computer Society, 2012.

[AR13] Rajeev Alur and Mukund Raghothaman. Decision problems for additive regular functions. In *ICALP (2)*, volume 7966 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2013.

[BC96] Alexandre Boudet and Hubert Comon. Diophantine Equations, Presburger Arithmetic and Finite Automata. In *CAAP*, volume 1059 of *Lecture Notes In Computer Science*, pages 30–43. Springer, 1996.

[Ber77] Leonard Berman. Precise Bounds for Presburger Arithmetic and the Reals with Addition: Preliminary Report. In *Foundations of Computer Science*, pages 95–99. IEEE Computer Society, 1977.

[BFLP03] Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci. Fast: Fast acceleration of symbolikc transition systems. In *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*, pages 118–121. Springer, 2003.

[BG00] Achim Blumensath and Erich Grädel. Automatic structures. In *LICS*, pages 51–62. IEEE Computer Society, 2000.

[BHV04] Ahmed Bouajjani, Peter Habermehl, and Tomás Vojnar. Abstract regular model checking. In *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2004.

[BJNT00] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000.

[Büc60] Julius Richard Büchi. Weak second-order Arithmetic and Finite Automata. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[CDG$^+$07] Hubert Comon, Max Dauchet, Rémif Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[Coo72] D. C. Cooper. Theorem Proving in Arithmetic without Multiplication. *Machine Intelligence*, pages 91–100, 1972.

[Cou94] Bruno Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126(1):53–75, 1994.

[DDHR06] Martin De Wulf, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Antichains: A new algorithm for checking universality of finite automata. In *Proceedings of CAV 2006: Computer-Aided Verification*, Lecture Notes in Computer Science 4144, pages 17–30. Springer-Verlag, 2006.

[DGH10] Antoine Durand-Gasselin and Peter Habermehl. On the Use of Non-deterministic Automata for Presburger Arithmetic. In *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 373–387. Springer, 2010.

[DGH12] Antoine Durand-Gasselin and Peter Habermehl. Ehrenfeucht-fraïssé goes elementarily automatic for structures of bounded degree. In Christoph Dürr and Thomas Wilke, editors, *STACS*, volume 14 of *LIPIcs*, pages 242–253. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[EH01] Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2:216–254, 2001.

[Eis08] Jochen Eisinger. Upper bounds on the automata size for integer and mixed real and integer linear arithmetic (extended abstract). In *CSL*, volume 5213 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 2008.

[FR74] Michael J. Fischer and Michael O. Rabin. Super-exponential Complexity of Presburger Aithmetic. In *Symposium on Applied Mathematics*, volume VII, pages 27–41. SIAM-AMS Proceedings, 1974.

[FR79] Jeanne Ferrante and Charles Weill Rackoff. *The computational complexity of logical theories*, volume 718 of *Lecture Notes in Mathematics.* Springer-Verlag, 1979.

[FV59] Solomon Feferman and Robert Lawson Vaught. The first order properties of products of algebraic systems. *Fundamenta Mathematicae*, 47:57–103, 1959.

[Gai82] Haim Gaifman. On local and non-local properties. In *Proc. of the Herbrand Symposium*, volume 107 of *Studies in Logic and the Foundations of Mathematics*, pages 105–135. Elsevier, 1982.

[Gri68] Timothy V. Griffiths. The unsolvability of the equivalence problem for λ-free nondeterministic generalized machines. *Journal of the ACM*, 15(3):409–413, 1968.

[Hin53] Jaakko Hintikka. *Distributive normal forms in the calculus of predicates.* Number 6 in Acta philosophica Fennica. Edidit Societas Philosophica, 1953.

[Hoa69] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications ACM*, 12(10):576–580, 1969.

[Hod82] Bernard R. Hodgson. On direct products of automaton decidable theories. *Theoretical Computer Science*, 19:331–335, 1982.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Kar09] Alexander Kartzow. FO Model Checking on Nested Pushdown Trees. In *MFCS*, volume 5734 of *Lecture Notes in Computer Science*, pages 451–463. Springer, 2009.

[KL04] H. Jerome Keisler and Wafik Boulos Lotfallah. Shrinking games and local formulas. *Annals of Pure and Applied Logic*, 128(1-3):215–225, 2004.

[KL11] Dietrich Kuske and Markus Lohrey. Automatic structures of bounded degree revisited. *Journal of Symbolic Logic*, 76(4):1352–1380, 2011.

[Kla08] Felix Klaedtke. Bounds on the Automata Size for Presburger Arithmetic. *ACM Transactions on Computational Logic*, 9(2):1–34, 2008.

[Kla10] Felix Klaedtke. Ehrenfeucht-Fraïssé goes automatic for real addition. *Information and Computation*, 208(11):1283–1295, 2010.

[KN94] Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In Daniel Leivant, editor, *LCC*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer, 1994.

[Kus09] Dietrich Kuske. Theories of Automatic Structures and Their Complexity. In *CAI*, volume 5725 of *Lecture Notes in Computer Science*, pages 81–98. Springer, 2009.

[Lad77] Richard E. Ladner. Application of model theoretic games to discrete linear orders and finite automata. *Information and Control*, 33(4):281–303, 1977.

[Ler08] Jérôme Leroux. Structural Presburger Digit Vector Automata. *Theoretical Computer Science*, 409(3):549–556, 2008.

[LLRT06] Michel Latteux, Aurélien Lemay, Yves Roos, and Alain Terlutte. Identification of biRFSA Languages. *Theoretical Computer Science*, 356(1-2):212–223, 2006.

[Opp78] Derek C. Oppen. A $2^{2^{2^{pn}}}$ Upper Bound on the Complexity of Presburger Arithmetic. *Journal of Computer and System Sciences*, 16(3):323–332, 1978.

[Pre30] Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du Premier Congrès des Mathematiciens des Pays Slaves*, pages 92–101. Ksiaznica Atlas, 1930.

[Saf88] Shmuel Safra. On the complexity of ω-automata. In *Foundations of Computer Science*, pages 319–327. IEEE Computer Society, 1988.

[Sei92] Helmut Seidl. Single-valuedness of tree transducers is decidable in polynomial time. *Theorical Computer Science*, 106(1):135–181, 1992.

[She75] Saharon Shelah. The monadic theory of order. *Annals of Mathematics*, 102(3):379–419, 1975.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word Problems Requiring Exponential Time: Preliminary Report. In *STOC*, pages 1–9. ACM, 1973.

[WB95] Pierre Wolper and Bernard Boigelot. An Automata-theoretic Approach to Presburger Arithmetic Constraints. In *Proceedings of Static Analysis Symposium*, volume 983 of *Lecture Notes In Computer Science*, pages 21–32. Springer-Verlag, September 1995.

[Web90] Andreas Weber. On the valuedness of finite transducers. *Acta Informatica*, 27(8):749–780, 1990.

[z1] Mona homepage. `http://www.brics.dk/mona/`.

[z2] Lash homepage. `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.

[z3] PresTAF homepage. `http://tapas.labri.fr/`.