Synchronous Subsequentiality and Approximations to Undecidable Problems

Christian Wurm
cwurm@phil.hhu.de
Universität Düsseldorf

We introduce the class of synchronous subsequential relations, a subclass of the synchronous relations which embodies some properties of subsequential relations. If we take relations of this class as forming the possible transitions of an infinite automaton, then most decision problems (apart from membership) still remain undecidable (as they are for synchronous and subsequential rational relations), but on the positive side, they can be approximated in a meaningful way we make precise in this paper. This might make the class useful for some applications, and might serve to establish an intermediate position in the trade-off between issues of expressivity and (un)decidability.

1 Introduction

Automata play an important role for a huge number of tasks, ranging from formal language theory to program semantics and model checking for logical languages. The most widespread applications have been found for finite automata; their properties and advantages are well-known to an extent which makes comments unnecessary. There can be several reasons to move from finite automata to infinite automata: in formal language theory, it is expressive power, same holds for automata as program semantics (a program might have at least in theory infinitely many possible states). There might also be another reason: for model checking, if we can compute a finite automaton directly from some formal (logical) language specifying the properties of the model (see for example [8]), 1. the resulting automaton might be too big to be effectively stored, or even if we can avoid this, it might happen that 2. though the resulting minimal automaton is manageable in size, its construction involves intermediate steps with automata too large to handle. For overview and motivation in logic, see [8]; in program semantics, this problem is referred to as *state explosion problem*, see [7].

So there are a number of reasons to use infinite automata. Their obvious disadvantage is that we cannot simply write them down, as they are infinite objects. What we rather do is the following: we specify a recursive procedure by which an automaton constructs its own state set *on the fly*, meaning it "constructs" a certain state only when reading a certain input. This leads to the theory of infinite automata (see [12] for an overview, [5],[6] for some important results), and it is this line of research on which we will investigate here. To specify a class of infinite automata, the main work lies in specifying a class of possible primitive transitions relations, which are associated with the input letters, and, maybe of less importance, classes possible sets of initial and final states.

For classes of infinite automata, there is usually the following situation of trade-off: Choice 1: we take a class of relations which is very restricted, such as typically relations corresponding to transitions of pushdown automata (henceforth: PDA) or slight extensions thereof (see for example [4]). In this case, problems such as whether one state can be reached from another one, and whether the language an automaton recognizes is non-empty are decidable. The problem is that many computations cannot be simulated by these transitions, and recognizing power remains quite limited. Choice 2 is that we take a

J. Esparza, E. Tronci (Eds.): Games, Automata,
Logics and Formal Verification 2015 (GandALF'15)
EPTCS 193, 2015, pp. 58–72, doi:10.4204/EPTCS.193.5

rather expressive class of relations such as the rational (or regular) relations, computed by (synchronous) finite-state transducers. These will provide sufficient expressive power for most purposes. But on the downside, most problems – reachability of states, emptiness of recognized language etc. – are undecidable. In between these two choices, there is usually thought to be little to gain (see also [12], [9]). In particular, for classes of relations which do not have PDA-style restrictions, undecidability strikes very quickly, so there is little use in investigating expressive subclasses of the rational relations.

Yet this is exactly what we will do here: we will investigate the class of synchronous subsequential relations, a rather large subclass of the regular relations, which embodies some properties of subsequential rational relations. It comes as no surprise that also for this class, the reachability problem is undecidable. Yet this class allows for an interesting way of approximating the reachability problem. This in turn allows to approximate other decision problems, which might make this class interesting for some applications. The notion of approximation we use is of some interest in itself: its intuitive meaning is that we get arbitrarily close to a solution of our binary problem, but we might never reach it. In a binary problem, this is of course less satisfying than in a numerical one. However, this can be put to use in various ways, which correspond to various interpretations we can give to reachability problems.

2 The General Setup: Self-constructing Automata

2.1 The Algebraic Setting: Self-constructing Automata

One can present automata in many different ways, the most standard one being probably the following: an automaton as **state-transition system** (STS) is a tuple $\mathfrak{A} = (\Sigma, Q, \delta, F, I)$, where Σ is a finite input alphabet, Q a set of states, $\delta \subseteq Q \times \Sigma \times Q$ a transition relation, $F \subseteq Q$ a set of accepting states, $I \subseteq Q$ the set of initial states. As our main interest will be in *infinite* automata, we cannot simply write down Q; same for δ and maybe I, F. The solution to this is simple: we define $Q \subseteq \Omega^*$ for some finite alphabet Ω inductively by 1. $I \subseteq Q$, and 2. if $q \in Q$, $a \in \Sigma$, then $\delta(q,a) \subseteq Q$,[1] and 3. nothing else is in Q. Next, we simply define δ as some computable relation, I, F as recursive sets, and we have a finite specification of an infinite state machine.

As in order to define an infinite STS, the main work lies in specifying the transition relation, we now introduce a more genuinely relational perspective, which amounts to a sort of non-standard definition of automata, which we dub **self-constructing automata**, for short SCA. This is more a notional innovation than a substantial one, and we just adopt it for convenience.

Self constructing automata can be roughly conceived of as mappings from the free monoid $\langle \Sigma^*, \cdot, \varepsilon \rangle$ to the monoid $\langle \wp(\Omega^* \times \Omega^*), \circ, id_{\Omega^*} \rangle$, where Ω^* is the free monoid over Ω; $\circ$ denotes relation composition; id_{Ω^*} denotes the identity relation on Ω^*. We call the former the *outer* algebra, the latter the *inner* one. We will as much as possible stick to the convention of using Σ if something is supposed to belong to the outer algebra, and Ω otherwise, though both designate finite alphabets throughout. We define a semi-SCA as a tuple $\langle \Sigma, \phi \rangle$, where ϕ is a map $\phi : \Sigma \to \wp(\Omega^* \times \Omega^*)$, thus mapping letters in Σ onto relations over Ω. It is extended to strings in the usual fashion, where $\phi(aw) = \phi(a) \circ \phi(w)$; so ϕ is a homomorphism from the outer algebra into the inner algebra. A word $w \in \Sigma^*$ from the outer alphabet then induces a relation $\phi(w) \subseteq \Omega^* \times \Omega^*$. We call the relations of the form $\phi(a) : a \in \Sigma$ the **primitive transition relations**. To get a full automaton, we still need an *accepting relation* of initial and accepting states. One usually specifies a single initial and a set of accepting states, yielding an accepting relation $\{x_0\} \times F$. As for us, acceptance will only play a minor role, we will take a slightly more general convention and assume

[1] Here and throughout this paper, we treat relations as functions into a powerset whenever it is convenient.

that SCA specify an **accepting relation** $F_R \subseteq \Omega^* \times \Omega^*$, which might be a Cartesian product, but need not be. We denote the accepting relation by F_R, where the subscript is just a reminder that we have a relation rather than a set of accepting states. Thus a full SCA is a tuple $\langle \Sigma, \phi, F_R \rangle$. We can now define the language recognized by an SCA. Let $\mathfrak{A} = \langle \Sigma, \phi, F_R \rangle$ be an SCA; then $L(\mathfrak{A}) := \{w \in \Sigma^* : \phi(w) \cap F_R \neq \emptyset\}$.

For convenience, we always put $\phi(\varepsilon) = id_{\Omega^*}$ (though this is not necessary in all cases). One might at this point wonder how we deal with ε-transitions. There is an easy solution to that, by changing ϕ. Before we state it, we give the following definitions: Given a set of relations $\{R_i : i \in I\}$, I an arbitrary index set, by $\{R_i : i \in I\}^{\otimes}$ we define the smallest set such that:

1. $\{R_i : i \in I\} \subseteq \{R_i : i \in I\}^{\otimes}$;
2. if $R, R' \in \{R_i : i \in I\}^{\otimes}$, then $R \circ R' \in \{R_i : i \in I\}^{\otimes}$.
3. $id_{\Omega^*} \in \{R_i : i \in I\}^{\otimes}$.

In words, $\{R_i : i \in I\}^{\otimes}$ contains $\{R_i : i \in I\}$, the identity, and is closed under composition. $[-]^{\otimes}$ is obviously related to the Kleene-star closure, but for composition of relations rather than for concatenation of strings. Be also careful to keep in mind that we do not take the union over this closure, so $\{R_i : i \in I\}^{\otimes}$ is *not* a relation but a set of relations. We will sometimes refer to a set of the form $\{R_i : i \in I\}^{\otimes}$ as a **relation monoid**, as it is easy to see that this set is a monoid with operation $\circ$. We will also need the union of this set and so define $\{R_i : i \in I\}^{\oplus} := \bigcup(\{R_i : i \in I\}^{\otimes})$. This is not exactly the smallest reflexive, transitive relation containing every $R_i : i \in I$, because it contains the full identity on Ω^*; apart from this however it equals the reflexive transitive closure. We urge the reader to be careful to not confuse $\{R_i : i \in I\}^{\otimes}$, a set of relations, with $\{R_i : i \in I\}^{\oplus}$, a relation. Now assume we want a relation R_ε to correspond to ε, whereas for each letter $a \in \Sigma$, we want a corresponding relation R_a. We can simulate this in the algebraic setting by putting $\phi(a) := (R_\varepsilon)^{\oplus} \circ R_a \circ (R_\varepsilon)^{\oplus}$.

Given some classes of relations $\mathscr{R}, \mathscr{R}_1, \mathscr{R}_2$, by $\mathscr{S}_{\mathscr{R}}$ we denote the class of all semi-SCA $\langle \Sigma, \phi \rangle$, where for all $\sigma \in \Sigma$, $\phi(\sigma) \in \mathscr{R}$. By $\mathscr{S}_{\mathscr{R}_1,\mathscr{R}_2}$ we denote the class of all SCA $\langle \Sigma, \phi, F_R \rangle$ where $\phi(\sigma) \in \mathscr{R}_1$ and $F_R \in \mathscr{R}_2$. A **class of SCA** is a class of the form $\mathscr{S}_{\mathscr{R}_1,\mathscr{R}_2}$ for some classes of relations $\mathscr{R}_1, \mathscr{R}_2$.

2.2 Decision Problems in the Relational Setting

The most important decision problems in this paper can be states as follows:
Reachability problem: Given a class of automata $\mathscr{S}$, is there an algorithm, which for any $\langle \Sigma, \phi, F_R \rangle \in \mathscr{S}$, $(x,y) \in \Omega^* \times \Omega^*$, determines in a finite number of steps whether there is $w \in \Sigma^*$ such that $(x,y) \in \phi(w)$?
Emptiness problem: Given a class of automata $\mathscr{S}$, is there an algorithm, which for any given automaton $\mathfrak{A} \in \mathscr{S}$ determines in a finite number of steps whether $L(\mathfrak{A}) = \emptyset$?
Inclusion problem: Given a class of automata $\mathscr{S}$, is there an algorithm, which for any given automata $\mathfrak{A}, \mathfrak{A}' \in \mathscr{S}$ determines in a finite number of steps whether $L(\mathfrak{A}) \subseteq L(\mathfrak{A}')$?

If there is such an algorithm, we say the problem is *decidable* for $\mathscr{S}$, if there is no such algorithm, we say it is *undecidable*. Note that in most cases (if $\mathscr{S}$ contains the class of finite automata), the inclusion problem subsumes the emptiness and universality problem, as they amount to decide whether $L(\mathfrak{A}) \subseteq \emptyset$, $\Sigma^* \subseteq L(\mathfrak{A})$. Obviously, there is a close connection between emptiness, reachability and recursiveness of relation monoids; it is made precise by the following lemma:

Lemma 1 *Let* FIN *be the class of finite relations. Given a class of relations $\mathscr{R}$, the following are equivalent:*

1. *The emptiness problem for $\mathscr{S}_{\mathscr{R},\mathsf{FIN}}$ is decidable.*

2. *The reachability problem for $\mathscr{S}_{\mathscr{R},\mathsf{FIN}}$ is decidable.*

3. *Every relation of the form $\{R_i : i \in I\}^{\oplus}$, where $R_i \in \mathscr{R} : i \in I$, $|I| < \omega$, is recursive.*

Proof. $1 \Rightarrow 2$: Take a semi-automaton $\langle \Sigma, \phi \rangle$, and ask whether there is $w \in \Sigma^*$ such that $(x,y) \in \phi(w)$. This is the case if and only if $L(\langle \Sigma, \phi, \{(x,y)\}\rangle) \neq \emptyset$, which by assumption is decidable.

$2 \Rightarrow 3$: Assume the reachability problem for $\mathscr{S}_{\mathscr{R},\mathsf{FIN}}$ is decidable, and for $R_i \in \mathscr{R} : i \in I$, $|I| < \omega$, ask whether $(x,y) \in (\{R_i : i \in I\})^{\oplus}$. We just put $\Sigma := I$, $\phi(i) = R_i$. Then we know whether there is $w \in I^*$ such that $(x,y) \in \phi(w)$, which holds if and only if $(x,y) \in (\{R_i : i \in I\})^{\oplus}$.

$3 \Rightarrow 1$: Ask whether for $\langle \Sigma, \phi, F_R \rangle \in \mathscr{S}_{\mathscr{R},\mathsf{FIN}}$, $L(\langle \Sigma, \phi, F_R \rangle) = \emptyset$. We take $(\{\phi(a) : a \in \Sigma\})^{\oplus}$, which is recursive by assumption. Now we have $L(\langle \Sigma, \phi, F_R \rangle) = \emptyset$ if and only if for all $(x,y) \in F_R$, $(x,y) \notin (\{\phi(a) : a \in \Sigma\})^{\oplus}$. As F_R is finite, this can be effectively checked for all members. ⊣

Of course it is a rather significant restriction to only consider finite accepting relations. But keep in mind that for many (most?) automata, we can boil it down to this case without loss of recognizing power, provided we allow ε-transitions.[2]

3 Synchronous Subsequential Relations

3.1 Synchronicity and Subsequentiality

The intuitive notion of synchronicity is that we use finite-state transducers which do not have ε-transitions. A (finite-state) transducer is a tuple $\langle Q, \Sigma, F, q_0, \delta \rangle$, where $F \subseteq Q$, $q_0 \in Q$, Q, Σ are finite, and $\delta \subseteq Q \times \Sigma \cup \{\varepsilon\} \times \Sigma \cup \{\varepsilon\} \times Q$. Transducers are based on the operation $\cdot$, where $(a,b) \cdot (c,d) = (ac, bd)$. We extend δ to $\hat{\delta}$ by $\hat{\delta}(q,a,b) = \delta(q,a,b)$, and $\hat{\delta}(q,aw,bv) = \{\hat{\delta}(q',w,v) : q' \in \delta(q,a,b)\}$; $L(\mathfrak{A}) = \{(w,v) : \delta(q_0,w,v) \cap F \neq \emptyset\}$. If R is recognized by a finite state transducer having transitions of the form (ε, a) and/or (a, ε), then it is **rational**. The main advantage of synchronicity is that without ε appearing in components, the operation $\cdot$ still gives rise to a free monoid: each term has a unique maximal decomposition. This property gets lost with ε: $(a,\varepsilon) \cdot (\varepsilon, b) = (\varepsilon, b) \cdot (a, \varepsilon)$. So this means that synchronous transducers can be reduced to simple string automata for most properties, whereas transducers in general cannot.

However, under this strict definition disallowing ε-transitions, the corresponding relations are rather useless for infinite state transition systems, as from each state, only a subset of the finite set of equally long states is reachable. So one uses a more liberal definition: we allow ε-transitions to occur, but only if in the component in which they occur, there are no more other letters to come. By $|w|$ we denote the length of a string, by w^k its k iterations. Put $\Omega_\perp := \Omega \cup \{\perp\}$, for $\perp \notin \Omega$. The **convolution** $\odot(w,v)$ of a pair of strings $(w,v) \in \Omega^* \times \Omega^*$ is defined by $\odot(w,v) := (w\perp^{max\{0,|v|-|w|\}}, v\perp^{max\{0,|w|-|v|\}})$. So in simple words: we take the shorter word of the two and add $\perp$-symbols to make it as long as the other. The convolution of a relation $R \subseteq (\Omega^*)^2$ is defined as $\odot R := \{\odot(w,v) : (w,v) \in R\}$. A relation $R \subseteq \Omega^* \times \Omega^*$ is **regular**, if there is an (ε-free) finite state transducer over $(\Omega_\perp)^2$ recognizing $\odot R$; we denote this class by REG. This is the notion of synchronicity we will use here. Regular relations as defined here form a proper subclass of the relations defined by finite state transducers in general, but the restriction comes with a huge gain: REG is closed under Boolean operations, whereas the rational relations are not closed under intersection and complement (see [1]). This makes the regular relations the more natural choice for application for example in logic (see [11]).

[2] We need not talk of FSA, but this holds also for PDA,TM and many intermediate classes.

It is easy to see that $\mathscr{S}_{REG}$ has an undecidable reachability problem, as Turing machine transitions are regular. A highly non-trivial result shows that $\mathscr{S}_{REG,\mathsf{FIN}}$ recognizes exactly the context-sensitive languages (proved in [10]).

The intuitive notion of **subsequentiality** is the following: relations on strings are subsequential, if computations depend on prefixes that have already been read, but *not* on the part of the input which has not been read yet (in the sequel, we refer to this as "future"). It is easy to define this concept for rational relations: let $\mathfrak{T} = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a transducer (possibly using ε-transitions) which is *total*, that is, for every $a \in \Sigma, q \in Q$, there is some $(q, a, a', q') \in \delta$. Then $\mathfrak{T}$ is **subsequential**, if $Q = F$. In this sense, there is no way to discard any computation we have made so far. A relation R is *subsequential* if it is computed by some subsequential transducer. We denote this class by *SUB*. The reachability problem for $\mathscr{S}_{SUB}$ is again *undecidable*. Though this result is not literally stated at any point we know, it easily falls off from results in [2] (who prove that iterated subsequential transduction generates languages which are not recursive). Moreover, as below we show a stronger result (theorem 9), we omit the formal statement and proof of this one. What we now do is to pair the concept of subsequentiality with the concept of synchronicity.

3.2 Synchronous Subsequential Relations

It is not reasonable to simply define a synchronous subsequential relation as being computed by a synchronous transducer with only accepting states: because in this case still computations depend on "the future": simply because whether we can map (or read) some ε-pendant $\perp$ depends on the symbols *which are still to come* (as $\perp$ can only occur final); so in some cases, we would have to discard computations which have been executed, which contradicts the essence of subsequentiality. So we have to use another definition of synchronous subsequential relations, taking a detour over (one-sided) infinite words. Let Ω be an alphabet; we denote the set of infinite words over Ω by Ω^ω; formally, we can think of them as functions $\mathbb{N} \to \Omega$; so they have the form $a_1 a_2 ... a_n ...$: we have a first letter, every letter has an immediate successor, and every letter is preceded by finitely many letters. For clarity, we will designate infinite words with $\overline{x}$,$\overline{y}$ etc. Furthermore, we need one special letter $\Box$, which we assume to be in all our alphabets Ω over which we form infinite words (please read this as a dummy for an arbitrary letter, which however has to be explicitly designated). We can get back from infinite words to finite words via the following map $\eta : \Omega^\omega \to \Omega^\omega \cup \Omega^*$:

$$\eta(a_1 a_2 ... a_n ...) = \begin{cases} a_1 \eta(a_2 ... a_n ..), \text{ if } a_2 ... a_n ... \notin \{\Box\}^\omega, \text{ and} \\ \varepsilon \text{ otherwise.} \end{cases}$$

So for $\overline{x} \in \Omega^\omega$, $\eta(\overline{x}) \in \Omega^*$ iff all but finitely many letters are $\Box$, and $\eta(\overline{x}) = \overline{x}$ otherwise. We now complete our definition of synchronous subsequential relations as follows: let $\mathfrak{T} := \langle Q, \delta, \Sigma, q_0 \rangle$ be a total (wrt. input q, a) ε-free transducer without accepting states. We put $L^\omega(\mathfrak{T}) := \{(a_1 a_2 ..., b_1 b_2 ...) \in \Omega^\omega : \delta(q_0, a_1, b_1, q) \in \delta$, and for all $i \in \mathbb{N}$, there is some $q \in \hat{\delta}(q_0, a_1 ... a_i, b_1 ... b_i)$ and q' such that $(q, a_{i+1}, b_{i+1}, q') \in \delta\}$. We say $\mathfrak{T}$ is **synchronous subsequential**, if for every $\overline{w} \in \Omega^\omega$, if $\eta(\overline{w}) \in \Omega^*$, $(\overline{w}, \overline{v}) \in L^\omega(\mathfrak{T})$, then $\eta(\overline{v}) \in \Omega^*$. We thus want for any input with a finite η-image the output to have a finite η-image as well. Technically, this can be easily implemented by making sure that after some number of input of $\Box$-symbols, we end up in states which only give $\Box$ as output given a $\Box$-input. Call these states **final** (not to be confused with accepting). The motivation for this definition is the following: we can use infinite words with finite η-image as if they were finite words (in the sense of effective computation).

Definition 2 *We say that a relation $R \subseteq \Omega^\omega \times \Omega^\omega$ is in* SyS*, the class of synchronous subsequen-*

tial relations, if there is a synchronous subsequential transducer $\mathfrak{T}$ *such that* $R = \{(\overline{w},\overline{v}) : (\overline{w},\overline{v}) \in L_\omega(\mathfrak{T}) \;\&\; \eta(\overline{w}) \in \Omega^*\}$.

The trick is the following: as we do not have accepting states, there is no additional complication when considering infinite words as opposed to finite words, as complications arise from mode of acceptance. As we do not have any ε-transition, if we restrict ourselves to have a finite input (modulo η), that is sufficient to ensure we have a finite output (mod η). Note that $\Box$ plays a double role in this definition: if it is followed by some other symbol $a \neq \Box$, then it is a symbol just like any other. If it is followed by $\Box$-symbols only, it basically plays the role of $\perp$ in the regular relations, that is, it is a dummy-symbol mapped to ε. Note that this "trick" is necessary to pair synchronicity with subsequentiality, as we do need a dummy symbol which gets deleted if final, and at the same time we need to ensure that transitions are total. An easy way to understand the relation of finite/infinite words is by analogy with the real numbers: a number as 3.05 is in some sense a "shorthand" for the number 3.050000...; we can cut away final 0s, but only if there is no other number to come.

For purposes of practical computation, this oscillation between finite and infinite words does not pose any problem: we can just take a finite input word w and take all the outputs we reach with some input $w\Box^n$ with which go to a final state. The resulting set of outputs (mod η) is exactly the output for the infinite input word $w\Box^\omega$. So if we have a specific finite input word, we can compute the output by means of a finite transducer with some accepting states (corresponding to the final states). However, this does *not* hold in the more general case where we have an infinite set of input words. This is because when reading an input $w\Box^n$, which is a prefix of an infinite word $\overline{w}$, we can *never be sure* whether $\eta(\overline{w}) = w$, so we can never actually stop the computation. This happens for example in relation composition, and one might consider this problematic. However, in this case we cannot write down the set of outputs anyway; and so we have to compute a finite representation of the infinite output set (or relation). The most important lemma to ensure applicability of SyS for infinite automata is the following:

Lemma 3 *If* $R, R' \in \mathsf{SyS}$, *then* $R \circ R' \in \mathsf{SyS}$.

Proof. We can easily show this by the standard transducer construction: given synchronous rational transducers $\mathfrak{A}_1 = (Q_1, \Sigma_1, q_0^1, \delta_1), \mathfrak{A}_2 = (Q_2, \Sigma_2, q_0^2, \delta_2)$, we simply construct $\mathfrak{A}_3 = (Q_1 \times Q_2, (q_0^1, q_0^2), \Sigma_1 \cup \Sigma_2, \delta_3)$, where δ_3 is defined as follows: $((q,r),a,b,(q'r')) \in \delta_3$, iff $(q,a,c,q') \in \delta_1$ and $(r,c,b,r') \in \delta_2$. It can be easily checked (by induction on word length[3]) that this construction works. $\dashv$

Lemma 4 $\mathsf{SyS} \subsetneq REG$.

Proof. 1. $\subseteq$. Assume $R \in \mathsf{SyS}$. Take the synchronous subsequential transducer $\mathfrak{T}$ recognizing R. Make an additional, disjoint copy of the final states that is 1. accepting, 2. absorbing (no leaving arcs) and 3. where you change $\Box$ to $\perp$. This is synchronous and does the job.

2. $\neq$. Every finite relation is synchronous. A finite relation which is not synchronous subsequential is $\{(aac,aa),(aad,ab)\}$. This is because SyS-transducers have to be total, and if we compute a prefix (aa,aa), we have an input letter d, there must be some output word corresponding to input aad. $\dashv$

By $pref(w)$ we denote all prefixes of w. Relations in *SUB* generally have the *prefix property*: if $(w,v) \in R$, then for every $w' \in pref(w)$, there is $v' \in pref(v)$ such that $(w',v') \in R$ (proof is straightforward). SyS does not have this property, as we see below:

Lemma 5 $\mathsf{SyS} \not\subseteq SUB$

[3] One might correctly observe that an induction is insufficient for the case of infinite words. However, we can reduce this case to the finite case as sketched above. We skip details as the construction is standard.

Proof. Take simply a relation R which computes the identity on all words in $\{\Box, a\}^*$, which end with a. Obviously, $R \in \mathsf{SyS}$; to see that $R \notin SUB$, just note that R does not have the prefix property. ⊣

The underlying reason is given by the sketch above: up to some extent, the computations of a synchronous subsequential transducer depend "on the future", namely as regards the question whether $\Box$ is mapped to ε or not. $SUB \not\subseteq \mathsf{SyS}$ is obvious, as $SUB \not\subseteq REG$. Note moreover that in the case of a one-letter alphabet, all SyS-relations over this alphabet are trivial.

3.3 SyS and Program Semantics: an Example

To exemplify the meaning of SyS-relations in terms of computational power, let us give the states of infinite automata the following interpretation: A state, being a finite string, represents a current state of a program. This program state in turn is a vector containing values of all declared objects (this comprises all variables, memory, registers, program counters etc.). We can thus think of *regions* of the string as values of variables. For example, we might have $\overbrace{a_1a_2a_3}^{\text{value of } x_1}\ \overbrace{a_4}^{\text{boundary}}\ \overbrace{a_5...a_8}^{\text{value of } x_2}$ .... Now assume the program computes the following function for all i such that x_{i+1} is a declared object:

$$f(x_{i+1}) := \begin{cases} f_1(x_{i+1}), \text{ if } x_1,...,x_i \text{ satisfy condition } C_1 \\ f_2(x_{i+1}) \text{ if } x_1,...,x_i \text{ satisfy condition } C_2 \\ \dots \end{cases}$$

where $f_1, f_2, ..$ are more basic computations and $C_1, C_2, ...$ are exhaustive. This kind of computations is exactly what relations in SyS do. An intermediate sequence of $\Box$ symbols can be thought of as saying: "variables are not instantiated", just giving them some default value. We can thus perform computation steps where the computation on variable x_{i+1} depends on the values of (and computations performed on) earlier variables, but not on those which have higher index. This is due to the restriction to subsequentiality. The restriction regarding synchronicity concerns a restricted ability to insert new variables into the existing order (it is clear that the order of variables is of crucial importance in this model): in one computation step, we can insert only a globally bounded number of new variables into the existing hierarchy, but not a arbitrary number (say, after any existing variable, insert a new one into the hierarchy). Moreover, it poses some bounds to value changes of variables.

We can thus say: $\mathscr{S}_{REG}$ corresponds to computations where values of all variables are computed in dependence of one another, *PDA* correspond to computations where only *one* variable can change value in the course of a computation step. $\mathscr{S}_{\mathsf{SyS}}$ then corresponds to the intermediate situation where there is a linear hierarchy (i.e. sequence) of variables, where the computations of variables higher up in the hierarchy (i.e. having lower index) have impact on computations on lower variables, but not vice-versa. So there are some restrictions; still it is easy to see that a huge number of computations can be performed in this way!

4 Some Formal Properties of SyS

We now scrutinize some more properties of SyS. We denote by $\diamond$ the product of two infinite strings, which is defined as follows: $(a_1a_2....) \diamond (b_1b_2...) = (a_1,b_1)(a_2,b_2)....$ So if $\overline{w} \in (\Omega_1)^\omega$, $\overline{v} \in (\Omega_2)^\omega$, then $\overline{w} \diamond \overline{v} \in (\Omega_1 \times \Omega_2)^\omega$. We lift this operations to sets in the canonical fashion and extend it canonically to relations, such that $R \diamond R' = \{((\overline{x} \diamond \overline{x'}), (\overline{y} \diamond \overline{y'})) : (\overline{x}, \overline{y}) \in R, (\overline{x'}, \overline{y'}) \in R'\}$. It is easy to see that we could also define this operation for finite words, but it would require some additional definitions to "synchronize"

them. There is one thing we have to take care of: we have used a symbol $\Box$ with a special meaning in the definition of SyS. We define a map $f_\Box$, which is a string homomorphism defined by $f_\Box(\Box,\Box) = \Box$, and $f_\Box(x) = x$ otherwise. Again, we lift this map to sets and relations in the canonical fashion. A simple, yet important lemma is the following:

Lemma 6 *If $R, R' \in \mathsf{SyS}$, then $f_\Box(R \diamond R') \in \mathsf{SyS}$.*

Proof. Take $\mathfrak{T}_1 = (Q, \Sigma_1, q_0, \delta_1), \mathfrak{T}_2 = (R, \Sigma_2, r_0, \delta_2)$. We put $\mathfrak{T}_3 = (Q \times R, f_\Box(\Sigma_1 \times \Sigma_2), (q_0, r_0), \delta_3)$, where δ_3 is defined by: $((q,r), f_\Box(a,b), f_\Box(a',b'), (q',r')) \in \delta_3$ iff $(q,a,a',q') \in \delta_1$,$(r,b,b',r') \in \delta_2$. This is a standard construction for FSA, so we leave its verification to the reader.[4] $\dashv$

Lemma 7 *Let $\mathscr{R}_1, \mathscr{R}_2$ be classes of relations. If $\mathscr{R}_1, \mathscr{R}_2$ are closed under $\diamond$, then the class of languages $L(\mathscr{S}_{\mathscr{R}_1,\mathscr{R}_2})$ is closed under intersection.*

Proof. We can show this by construction: take $\mathfrak{A}_1 := \langle \Sigma_1, \phi_1, F_{R1} \rangle$, $\mathfrak{A}_2 := \langle \Sigma_2, \phi_2, F_{R2} \rangle$. Without loss of generality, we assume that $\Sigma_1 = \Sigma_2$.[5] We construct a new automaton $\mathfrak{A}_3 := \langle \Sigma_1, \langle \phi_1, \phi_2 \rangle, (F_{R1} \diamond F_{R2}) \rangle$, where $\langle \phi_1, \phi_2 \rangle : \Sigma_1 \to (\Omega_1 \times \Omega_2)^\omega \times (\Omega_1 \times \Omega_2)^\omega$ is defined by $\langle \phi_1, \phi_2 \rangle(w) = \phi_1(w) \diamond \phi_2(w)$. It is easy to see that $L(\mathfrak{A}_3) = L(\mathfrak{A}_1) \cap L(\mathfrak{A}_2)$. $\dashv$

It is also easy to see that in particular the class FIN of finite relations is closed under $\diamond$. Moreover, regarding SyS, the map $f_\Box$ does not affect recognizing power, so it remains a technical detail we will ignore in the sequel. Next we prove the following:

Lemma 8 *Let L be a CFL. Then there is a $\mathfrak{A} \in \mathscr{S}_{\mathsf{SyS},\mathsf{FIN}}$ such that $L = L(\mathfrak{A})$.*

Proof. We show how to encode a PDA in SyS. The argument is conceptually simple yet tedious in full formality, so we give a rather informal explanation.

We encode stacks (and control states) as strings, and assume we read them from bottom to top. Moves which consist in pushing something onto the stack are unproblematic: we just have to make sure that after reading (x,x) we have a transition $(\Box, y)$, and this is sufficient if we make sure for all reachable states that if we encounter $\Box$ at some point, then only $\Box$ is to follow in that component.

The problem comes with pop-moves: reading $\overline{x}$ from left to right, we cannot tell whether a certain symbol is final in $\eta(\overline{x})$. We therefore proceed as follows: a set of popping PDA-transitions $\{(xa, x) : x \in \Omega^*, a \in \Omega\}$ is encoded by a set of transitions $\{(xab_1b_2..., x\Box c_1c_2...) : x \in \Omega^*$ and for all $i \in \mathbb{N}$, if $b_i \neq \Box$, then $c_i = \blacksquare$ and $c_i = \Box$ otherwise$\}$ (where $\blacksquare$ does not figure in the original stack alphabet). This relation is clearly in SyS. Note that for all states not containing $\blacksquare$, this preserves the property that if we encounter $\Box$ at some point, then only $\Box$ is to follow in that component.

Thereby, we use $\blacksquare$ as an "absorbing symbol", that is, a symbol which can never be eliminated from a string, and which does not figure in any accepting state of the automaton $\mathfrak{A} \in \mathscr{S}_{\mathsf{SyS},\mathsf{FIN}}$. We use it to exclude all transitions which do not correspond to "well-formed" stack moves from reaching an accepting state. This creates some additional non-determinism wrt. the PDA, but makes sure all $\blacksquare$-free states have been reached by legitimate PDA-moves. $\dashv$

These lemmas, taken together, already have a strong immediate consequence:

Theorem 9 *Given $R_i \in \mathsf{SyS} : i \in I$, $|I| < \omega$, it is undecidable whether $(x,y) \in \{R_i : i \in I\}^\oplus$. Equivalently, the reachability problem for $\mathscr{S}_{\mathsf{SyS}}$ is undecidable.*

[4] A very similar construction is performed when constructing the automaton recognizing the intersection of two regular languages.

[5] We can always enlarge alphabets with a new letter σ, putting $\phi(\sigma) = \emptyset$. Of course we have to assume $\emptyset$ is in any class of relations.

Proof. Assume conversely the problem is decidable. Then equivalently, the problem whether $L(\mathfrak{A}) = \emptyset$ is decidable for all $\mathfrak{A} \in \mathscr{S}_{\mathsf{SyS,FIN}}$. Now we take two context-free languages L_1, L_2, accepted by two PDA by empty stack. We encode the two PDA into $\mathfrak{A}_1, \mathfrak{A}_2 \in \mathscr{S}_{\mathsf{SyS,FIN}}$. By lemma 8, we have $\mathfrak{A}_3 \in \mathscr{S}_{\mathsf{SyS,FIN}}$, such that $L(\mathfrak{A}_3) = L(\mathfrak{A}_1) \cap L(\mathfrak{A}_2)$. By assumption, we can decide whether $L(\mathfrak{A}_3) = \emptyset$, which holds if and only if $L_1 \cap L_2 = \emptyset$. But this is well-known to be an undecidable problem for two context-free languages - contradiction. ⊣

Corollary 10 *The emptiness, universality and inclusion problems for $\mathscr{S}_{\mathsf{SyS,FIN}}$ are undecidable.*

So we do have a negative result for reachability in the first place. The proof also tells us how powerful SyS is despite its restrictions, as we can recognize any intersection of context-free languages, and so if we allow ε-transitions, we can already recognize every recursively enumerable language (it is also well-known that by simulating two pushdowns we can easily simulate a Turing machine). However, we can show that there is a way to approximate its decision problem in a way to be made precise.

5 Decompositions and Approximations

We define the maps $\tau_n : n \in \mathbb{N}$ by $\tau_n(a_1a_2..., b_1b_2...) = (a_n, b_n)$. For R a relation, we put $\tau_n(R) = \{\tau_n(\overline{w}, \overline{v}) : (\overline{w}, \overline{v}) \in R\}$ (again, for simplicity we give the definition only for infinite words). Note that τ_n is not necessarily a homomorphism for relation composition: Put $R_1 = \{(ab..., ba...)\}, R_2 = \{(bb..., aa...)\}$; then $\tau_1(R_1) \circ \tau_1(R_2) = \{(a,a)\}$, whereas $\tau_1(R_1 \circ R_2) = \emptyset$. However, we have one inclusion: for all $n \in \mathbb{N}$, $R, R' \in \{R_i : i \in I\}^{\otimes}$, $\tau_n(R \circ R') \subseteq \tau_n(R) \circ \tau_n(R')$, as is easy to see.

These notions are related to the notion of direct and subdirect decompositions, fundamental to universal algebra (see [3]). We can say a set $\{R_i : i \in I\}$ is **directly decomposable**, if for all $n \in \mathbb{N}$, τ_n is a $\circ$-homomorphism for $\{R_i : i \in I\}^{\otimes}$, and in addition, $\prod_{n<\omega}\{\tau_n((R_i : i \in I)^{\otimes}\} = \{R_i : i \in I\}^{\otimes}$ (here, $\prod$ refers to the operation $\cdot$ – the pointwise concatenation of pairs – which is extended to sets). We can say $\{R_i : i \in I\}^{\otimes}$ is **subdirectly decomposable**, if for all $n \in \mathbb{N}$, τ_n is a $\circ$-homomorphism. One can readily check that this is sufficient for the usual conditions of (sub)direct products to hold. We also define the related family of maps $\sigma_n : n \in \mathbb{N}$ by $\sigma_n(\overline{x}, \overline{y}) = \prod_{i=1}^{n} \tau_i(\overline{x}, \overline{y})$. The main motivation for considering SyS is the following result:

Theorem 11 *Assume $\langle \Sigma, \phi \rangle \in \mathscr{S}_{\mathsf{SyS}}$. Then for all $n \in \mathbb{N}, a, b \in \Omega, x, y \in \Omega^*$, the sets $\{w \in \Sigma^* : (a,b) \in \tau_n(\phi(w))\}$ and $\{w \in \Sigma^* : (x,y) \in \sigma_n(\phi(w))\}$ are regular languages.*

We prove this only for σ, as the two proofs are almost identical. If we have a function $f : M \to N$, then for $X \subseteq M$, we write $f[X] := \{f(x) : x \in X\}$.

Proof. Take an arbitrary fixed n. For $\overline{x}, \overline{y} \in \Omega^{\omega}$, we write $\overline{x} \sim_n \overline{y}$, if $\overline{x} = z\overline{x'}, \overline{y} = z\overline{y'}$ with $|z| = n$. This is obviously an equivalence relation. The notion of synchronous subsequentiality now allows for the following argument: If $\overline{x} \sim_n \overline{x'}$, then for any $w \in \Sigma^*$, we have $(\overline{x}, \overline{y}) \in \phi(w)$, if and only if we have $(\overline{x'}, \overline{y'}) \in \phi(w)$ for some $\overline{y'} : \overline{y'} \sim_n \overline{y}$. Moreover, if $\overline{x} \sim_n \overline{x'}, \overline{y} \sim_n \overline{y'}$, then $\sigma_n(\overline{x}, \overline{y}) = \sigma_n(\overline{x'}, \overline{y'})$.

By this, we can obtain a congruence: Define $\phi_{\sim_n}$ by $\phi_{\sim_n}(w) = [\phi(w)]_{\sim_n}$, that is, the set of $\sim_n$-equivalence classes of $\phi(w)$. It is then clear that we have $\sigma_n \circ \phi(w) = \sigma_n \circ \phi_{\sim_n}(w)$, if we choose an arbitrary representative for each equivalence class. Moreover, the monoid $\phi_{\sim_n}[\Sigma^*]$ is *finite*, simply because Ω^{ω} modulo $\sim_n$ has only finitely many equivalence classes. So $\phi_{\sim_n}$ is a map from Σ^* into a finite set of relations, and hence for every element, the pre-image consisting of the words mapped to relations containing this element, is a regular set (this is a fundamental result of the algebraic theory of finite automata, see [1]). Hence $\{w \in \Sigma^* : (x,y) \in \sigma_n(\phi(w))\}$ is regular. ⊣

To see how σ, τ are connected, consider that we have

$$\{w \in \Sigma^* : (a_1...a_n, b_1...b_n) \in \sigma_n(\phi(w))\} = \bigcap_{i \leq n} \{w \in \Sigma^* : (a_i, b_i) \in \tau_i(\phi(w))\}$$

As regular languages are closed under intersection, we thus obtain a regular language for any finite sequence of indices. Moreover, the emptiness, universality and inclusion problem for regular languages are decidable, and our computation effective. We now take the following convention: given $x,y \in \Omega^*$, we put $f_\omega(x,y) = \{(x\overline{x}, y\overline{y}) : \overline{x},\overline{y} \in \Omega^\omega, \eta(\overline{x}), \eta(\overline{y}) \in \Omega^*\}$. Then the main result on approximation reads:

Corollary 12 *For any $\langle \Sigma, \phi \rangle \in \mathscr{S}_{\mathsf{SyS}}$, $(x,y) \in \Omega^* \times \Omega^*$, the set $\{w : \phi(w) \cap f_\omega(x,y) \neq \emptyset\}$ is a regular set which can be effectively computed.*

This means that we can approach the problem of emptiness/reachability by calculating the language for longer and longer prefixes of the infinite pair $(\overline{x},\overline{y})$, which yields smaller and smaller regular languages. SyS can be characterized in another very natural fashion:

Lemma 13 *Let $\{R_i : i \in I\}$ be a set of relations, such that for all $i \in I$, $R_i \in \mathsf{SyS}$. Then $\sigma_n(\{R_i : i \in I\}^\oplus)) = (\sigma_n[\{R_i : i \in I\}])^\oplus$.*

Proof. Follows directly from the conditions of synchronicity, subsequentiality and totality. ⊣

This is in some sense the essence of synchronous subsequentiality. The converse implication of lemma 13 is however wrong: just take any singleton set $\{R\}$, where $R \circ R = \emptyset$. This satisfies the above equation, and it is easy to find $R \notin \mathsf{SyS}$ for this.

6 The Limits of Approximation

One might think that this allows for *effective* approximation of the reachability problem in the sense that if $(\overline{x},\overline{y}) \notin \{\phi(a) : a \in \Sigma\}^\oplus$, then there is some finite (x,y) such that $(\overline{x},\overline{y}) \in f_\omega(x,y)$ and $\{\phi(a) : a \in \Sigma\}^\oplus \cap f_\omega(x,y) = \emptyset$. In this case, the approximation would always succeed in the case that actually $(\overline{x},\overline{y}) \notin \{\phi(a) : a \in \Sigma\}^\oplus$, the only problem being that we never know whether it is still worth continuing the approximation. So the situation would be similar to a recursive enumeration (as opposed to a terminating decision procedure). Unfortunately, this is **wrong**, and we want to underline this fact to not give a misleading impression of the power of the methods developed here:

Theorem 14 *There exists $\langle \Sigma, \phi \rangle \in \mathscr{S}_{\mathsf{SyS}}, (\overline{x},\overline{y})$, such that 1. $(\overline{x},\overline{y}) \notin \{\phi(a) : a \in \Sigma\}^\oplus$, and yet 2. for every finite prefix $(x,y) = \sigma_n(\overline{x},\overline{y})$, $\{\phi(a) : a \in \Sigma\}^\oplus \cap f_\omega(x,y) \neq \emptyset$.*

Proof. Assume that the above claim is wrong, which means: if $(\overline{x},\overline{y}) \notin \{\phi(a) : a \in \Sigma\}^\oplus$, then there is a pair of finite strings $(x,y) = \sigma_n(\overline{x},\overline{y})$ and $f_\omega(x,y) \cap \{\phi(a) : a \in \Sigma\}^\oplus = \emptyset$. We show how to construct a decision procedure for checking the reachability problem out of this fact, contradicting theorem 9.

We have a recursive enumeration of $\{\phi(a) : a \in \Sigma\}^\oplus$. So we can by turns enumerate an element of $\{\phi(a) : a \in \Sigma\}^\oplus$, and compute $\{w : \phi(w) \cap f_\omega(x,y) \neq \emptyset\}$ for a growing prefix (x,y) of $(\overline{x},\overline{y})$. By this method, either at some point we will find $(\overline{x},\overline{y})$ in our enumeration, or we will, for some finite $(x,y) \in (\Omega^*)^2$, find that $\{w : \phi(w) \cap f_\omega(x,y) \neq \emptyset\} = \emptyset$, meaning that $\{\phi(a) : a \in \Sigma\}^\oplus \cap f_\omega(x,y) = \emptyset$. This entails that $\{\phi(a) : a \in \Sigma\}^\oplus$ is recursive – contradiction. ⊣

This is a negative result, but keep in mind we have never promised a decision procedure, just a method for approximations! Now we can also state more precisely what it means for us to approximate a binary problem: if $(\overline{x},\overline{y}) \notin \{\phi(a) : a \in \Sigma\}^\oplus$, then the set of candidate strings will diminish for $\sigma_n(\overline{x},\overline{y})$ as n increases; so we get closer to the empty set, though it is not said we reach it after a finite number of steps (theorem 9). Note that we can still be more precise: the reachability problem is undecidable only if for all $n \in \mathbb{N}$, $|\{w : \phi(w) \cap \sigma_n(\overline{x},\overline{y})\}| = \infty$ (otherwise we can easily check by brute force).

Now there is one substantial concern which has to be addressed: could it be that approximations of the above kind are trivial, in the sense they can always be performed in one way or another? Assume we want to check whether $(\overline{x},\overline{y}) \notin \{\phi(a) : a \in \Sigma\}^{\oplus}$. Then we just take an enumeration of Σ^*, and check for each w whether $(\overline{x},\overline{y}) \in \phi(w)$. This way, we get a growing set of strings we can exclude, and *in the limit*, we find a solution to our problem. There is however a substantial difference between this kind of approximation and the method we are proposing: if $(\overline{x},\overline{y}) \notin \{\phi(a) : a \in \Sigma\}^{\oplus}$, the latter enumeration method *can never succeed*, because after any finite number of steps, we will necessarily remain with an infinite set of strings to check. So because this method comes with no hope of success, it is completely useless. On the other side, in our case, we know that approximation *can* succeed: because each step of approximation excludes a possibly infinite language of candidate strings. This does of course not mean that it necessarily succeeds: then it would be a decision procedure. This is, in other words, what we mean by an approximation to the binary problem of reachability, and this is the meaning of our results. We try to make this more precise in the following section.

7 The Prospects of Approximation

7.1 Three Potential Applications of Theorem 11

Apart from this negative result, there are many upsides to our notion of approximation. In particular, there are three different ways to "interpret" theorem 11, which are based on different interpretations of states, namely as atomic entities, as encoding objects in a space with a real-valued distance, and as vectors of values of program variables. One should keep in mind that sketching methods for SyS, they are *a fortiori* applicable to transition relations which can be simulated by SyS such as those of pushdown automata and many other methods in use.

The first interpretation of the notion of approximation is based on states as discrete objects. Given a semi-automaton $\langle \Sigma, \phi \rangle$, a relation R, we put $\chi_\phi(R) := \{w \in \Sigma^* : \phi(w) \cap R \neq \emptyset\}$. Given an SCA $\mathfrak{A} = \langle \Sigma, \phi, F_R \rangle \in \mathscr{S}_{\mathsf{SyS},\mathsf{FIN}}$, we can use the methods described here to *approximate* $L(\mathfrak{A})$ (note that the restriction that $F_R \in \mathsf{FIN}$ is not strictly necessary, as $\sigma_n[F_R]$ is finite anyway, but it simplifies things). We can obviously compute $f_\omega \circ \sigma_n[F_R]$. We then know that for any $\langle \Sigma, \phi, F_R \rangle \in \mathscr{S}_{\mathsf{SyS},\mathsf{FIN}}$, $\chi_\phi(f_\omega \circ \sigma_n[F_R])$ is a regular language which is effectively computable (a finite union of regular languages). Consequently, the sequence $(\chi_\phi(f_\omega \circ \sigma_n[F_R]))_{n\in\mathbb{N}}$ is a decreasing sequence of regular languages, each containing $L(\mathfrak{A})$; that is, $\chi_\phi(f_\omega \circ \sigma_n[F_R]) \subseteq \chi_\phi(f_\omega \circ \sigma_{n+1}[F_R]) \subseteq L(\mathfrak{A})$ for all $n \in \mathbb{N}$.

Definition 15 *Given an SCA $\mathfrak{A} = \langle \Sigma, \phi, F_R \rangle$, we define its n-**approximation** $\mathfrak{A}_n = \langle \Sigma, \phi, f_\omega \circ \sigma_n[F_R] \rangle$.*

This kind of approximation is of course of little interest if we consider the membership problem, which is decidable for $\mathscr{S}_{\mathsf{SyS},\mathsf{FIN}}$ anyways. However, it might be very interesting as soon as we consider problems which are undecidable for $\mathscr{S}_{\mathsf{SyS},\mathsf{FIN}}$. In the prior examples, we have focussed on reachability; emptiness is related in the obvious way stated in lemma 1. But we can also consider problems like inclusion and universality. As basically all important decision problems are decidable for the regular languages, we can obviously compute these problems for arbitrary approximations $\mathfrak{A}_n$. So we get the following in a straightforward fashion:

Lemma 16 *Given $\mathfrak{A}, \mathfrak{A}' \in \mathscr{S}_{\mathsf{SyS},\mathsf{FIN}}$, we can decide whether $L(\mathfrak{A}_n) = \emptyset$, $L(\mathfrak{A}_n) = \Sigma^*$, $L(\mathfrak{A}_n) \subseteq L(\mathfrak{A}'_n)$ for arbitrary $n \in \mathbb{N}$.*

This is clear because all are regular languages we can effectively compute, and we approximate all these decision problems in this sense. What we have to show is that these approximations are arbitrarily

precise: let $(L_n)_{n\in\mathbb{N}}$ be a decreasing sequence of languages, that is $L_n \supseteq L_{n+1}$. We denote by $lim_{n\to\infty}(L_n)$ the unique language L such that 1. for all $n \in \mathbb{N}$, $L_n \supseteq L$, and 2. if for all $n \in \mathbb{N}$, $L_n \supseteq L'$, then $L' \subseteq L$. This language always exists, and can be simply defined as $\{w : \forall n \in \mathbb{N} : w \in L_n\} = \bigcap_{n\in\mathbb{N}} L_n$. The following lemma establishes the desired correlation:

Lemma 17 $lim_{n\to\infty} L(\mathfrak{A}_n) = L(\mathfrak{A})$.

Proof. $\supseteq$ is obvious, as for all $n \in \mathbb{N}$, $L(\mathfrak{A}_n) \supseteq L(\mathfrak{A})$, and $lim_{n\to\infty} L(\mathfrak{A}_n)$ is the largest language satisfying this condition.

$\subseteq$: assume $(\overline{x},\overline{y}) \in f_\omega \circ \sigma_n[F_R]$ for all $n \in \mathbb{N}$. After some finite number of positions (say k), all letters in $(\overline{x},\overline{y})$ are $\square$, and for $(\overline{x}',\overline{y}') \in F_R$ the same holds (say after k' positions). Consequently, if the two agree on the first $max\{k,k'\}$ positions, they will agree on all. Hence, we will have $(\overline{x},\overline{y}) \in F_R$, and if for all $n \in \mathbb{N}$, $\phi(w) \cap f_\omega \circ \sigma_n[F_R] \neq \emptyset$, then $\phi(w) \cap F_R \neq \emptyset$; consequently, if $w \in L(\mathfrak{A}_n)$ for all $n \in \mathbb{N}$, then $w \in L(\mathfrak{A})$. ⊣

Note however that this limit-construction does not need to preserve *any* computational properties: we can easily construct a decreasing sequence or regular languages $(L_n)_{n\in\mathbb{N}}$ such that $lim_{n\to\infty} L_n$ is not recursively enumerable: let $L \subseteq \Sigma^*$ be a language which is not recursively enumerable, put $L_1 = \Sigma^*$, and $L_{n+1} = L_n - \{w\}$ for some $w \in L_n$. Then for all $n \in \mathbb{N}$, L_n is a co-finite language and therefore regular, whereas $lim_{n\to\infty}(L_n) = L$.

We now sketch how we might use this approximation to introduce a limit value which has some similarity to a probability, though the values involved are clearly not probabilities in a technical sense. We let $\mathscr{I}$ denote the inclusion problem, such that $\mathscr{I}(\mathfrak{A},\mathfrak{A}') = 1$ if $L(\mathfrak{A}) \subseteq L(\mathfrak{A}')$, and $\mathscr{I}(\mathfrak{A},\mathfrak{A}') = 0$ otherwise. As we have said, this allows us to encode the problem of emptiness and universality. We can now simply define a limit for stepwise approximation as follows:

$$Lim(\mathscr{I}(\mathfrak{A},\mathfrak{A}') = 1) = lim_{n\to\infty} \frac{|\{i : i \leq n \,\&\, \mathscr{I}(\mathfrak{A}_i,\mathfrak{A}'_i) = 1\}|}{n} \text{ (provided this limit exists)}$$

That is, the chances that $\mathscr{I}(\mathfrak{A},\mathfrak{A}')$ has a positive answer would be defined as the limit of the cardinality of numbers $\leq n$ where it has a positive answer for $\mathfrak{A}_n, \mathfrak{A}'_n$, divided by n.

Note that it is not said that our limit gives the correct answer to the problem: take the case of $\mathfrak{A}_\emptyset$, which is an automaton $\langle \Sigma, \phi, \emptyset\rangle$ recognizing the empty language, and $\mathfrak{A} \in \mathscr{S}_{\mathsf{SyS,FIN}}$, which is such that 1. $L(\mathfrak{A}) = \emptyset$, and 2. for all $n \in \mathbb{N}$, $L(\mathfrak{A}_n) \neq \emptyset$ (this automaton must exist, otherwise emptiness would be decidable). Now if we consider $\mathscr{I}(\mathfrak{A},\mathfrak{A}_\emptyset)$, we find that for all $n \in \mathbb{N}$, $\mathscr{I}(\mathfrak{A}_n,(\mathfrak{A}_\emptyset)_n) = 0$, hence $Lim(\mathscr{I}(\mathfrak{A},\mathfrak{A}_\emptyset)) = 0$, whereas $\mathscr{I}(\mathfrak{A},\mathfrak{A}_\emptyset) = 1$. So the limit does not necessarily coincide with the correct answer to the problem!

However, in the case of universality, things work out better: let $\mathfrak{A}_{\Sigma^*}$ be a finite automaton such that $L(\mathfrak{A}_{\Sigma^*}) = \Sigma^*$, and hence for all $n \in \mathbb{N}$, $L((\mathfrak{A}_{\Sigma^*})_n) = \Sigma^*$. Let $\mathfrak{A}$ be just any automaton in $\mathscr{S}_{\mathsf{SyS,FIN}}$. It is easy to see that in this case, we have $Lim(\mathscr{I}(\mathfrak{A}_{\Sigma^*},\mathfrak{A})) = \mathscr{I}(\mathfrak{A}_{\Sigma^*},\mathfrak{A})$. This is due to the fact that in our approach, we always approximate from the top. We can slightly generalize this concept:

Definition 18 *An instance of the inclusion problem $\mathscr{I}(\mathfrak{A},\mathfrak{A}')$ is **correctly approximated**, if $Lim(\mathscr{I}(\mathfrak{A},\mathfrak{A}'))$ exists and $Lim(\mathscr{I}(\mathfrak{A},\mathfrak{A}')) = \mathscr{I}(\mathfrak{A},\mathfrak{A}')$.*

Lemma 19 *Let $\mathfrak{F}$ be a finite SCA.*[6] *Then the problem $\mathscr{I}(\mathfrak{F},\mathfrak{A})$ is correctly approximated, whereas $\mathscr{I}(\mathfrak{A},\mathfrak{F})$ in general is not.*

Proof. The second part already follows from the counterexample above. For the first part, consider that for some $k \in \mathbb{N}$ and for all $n > k$ we have $L(\mathfrak{F}_n) = L(\mathfrak{F})$. So if $\mathscr{I}(\mathfrak{F},\mathfrak{A}) = 1$, then for all $n > k$, $\mathscr{I}(\mathfrak{F}_n,\mathfrak{A}_n) = 1$, and so $Lim(\mathscr{I}(\mathfrak{F},\mathfrak{A})) = 1$.

[6] By this, we mean an SCA such that $\phi[\Sigma]$ is finite (mod η). This is obviously equivalent to a finite state automaton.

Conversely, assume $Lim(\mathscr{I}(\mathfrak{F},\mathfrak{A}))=1$. Then if $w\in L(\mathfrak{F})$, then $w\in L(\mathfrak{A}_n)$ for all $n\in\mathbb{N}$. As $L(\mathfrak{A})=lim_{n\to\infty}L(\mathfrak{A}_n)$, it follows that $w\in L(\mathfrak{A})$, so $\mathscr{I}(\mathfrak{F},\mathfrak{A})=1$. ⊣

Note that by this result, we have *a fortiori* the negative result that $\mathscr{I}(\mathfrak{A},\mathfrak{A}')$ cannot be correctly approximated in the general case where $\mathfrak{A},\mathfrak{A}'\in\mathscr{S}_{\mathsf{SyS,FIN}}$. Moreover, in this case it is not clear whether the limit even exists. From the fact that the universality problem can be correctly approximated, it follows that even if the limit exists, in general it is not computable. We can, however, compute $Lim_n(\mathscr{I}(\mathfrak{A},\mathfrak{A}')=1)=\frac{|\{i:i\leq n\&\mathscr{I}(\mathfrak{A}_i,\mathfrak{A}'_i)=1\}|}{n}$ for any n. This way, we might be able to reasonably estimate the chances that a certain instance of the inclusion problem has a certain answer, at least if the problem is correctly approximated.

To make clear what about these results is peculiar to SyS, we should add the following: we can provide limit constructions for approximations of reachability for any recursive class of relations, for example by taking the limit of reachability with words of length $\leq n$. The difference of this approach to the ones we sketched here is that the former are always confined to finite stringsets; we never make the step to *infinite* languages approximating the target language. This is a fundamental shortcoming, as we can never talk about infinitary properties. On the other side, the two methods might be combined: as we have seen, our approximation of languages is always "from the top", proceeding to smaller languages, whereas a concept as "reachability with words of length $\leq n$" always comes from the bottom.

The following two interpretations of theorem 11 are more genuine to SyS, and there is no way to get similar results without the fundamental properties of synchronous subsequential relations. We first provide an interpretation of states which allows for a numeric approximation. It is based on the concept of state-strings being allocated in a space, where for each two strings we have a unique real-valued distance. A distance function on Ω^* is any function $f:\Omega^*\times\Omega^*\to\mathbb{R}_0^+$, such that $f(w,w)=0$ for all $w\in\Omega^*$, and $f(w,v)=f(v,w)$. A distance function measures *how close* a string w is to another string v.

This is a very general notion; we will consider more restrictive distance functions for the set $\{\eta(\overline{w}):\overline{w}\in\Omega^\omega\}$. By $gcp(w,v)$ we denote the *greatest common prefix* of w,v, that is, $gcp(aw,av)=a(gcp(w,v))$, and $gcp(aw,bv)=\varepsilon$ if $a\neq b$. A **normal distance function** on Ω^* is defined as follows:

1. there is a map $val:\Omega^+\to\mathbb{R}$, such that if $\square^n\in pref(w)$, $\square^n\notin pref(v)$, then $val(w)>val(v)$.
2. For strings w,w',v, if $|gcp(w,v)|>|gcp(w',v)|$, then $dist(w,v)<dist(w',v)$ – that is, the longer the common prefix, the smaller the distance.
3. If $v=wx$, then $dist(w,v)=val(x)$.

Note that 3. implies that $val(\varepsilon)=0$ and $dist(\varepsilon,w)=val(w)$. Therefore condition 1. only applies to Ω^+. An easy example of such a measure is if we let words represent real numbers in $[0,1]$ (in $|\Omega|$-ary representation). We simply specify a linear order $<$ on Ω. Then val is just the map from the representation to the number it represents, and $dist(x,y)$ is defined by $|val(x)-val(y)|$. Note that our definition is independent on the order $<$ in Ω! Given a normal distance measure $dist$, we call a **convex region** of Ω^* a set of words M, such that $M=\{w:dist(v,w)\leq x$ for some $v\in\Omega^*$, $x\in\mathbb{R}\}$; we also say M has *center* v. The following properties are easy to see:

1. For every normal distance function $dist$, $n\in\mathbb{N}$, $\overline{w}\in\Omega^\omega$, $\eta[f_\omega\circ\sigma_n(\overline{w})]$ forms a convex region in the space defined by $dist$. We call such a space a **normal subspace** of Ω^*.
2. Given two normal subspaces $X,Y\subseteq\Omega^*$ and $\langle\Sigma,\phi\rangle\in\mathscr{S}_{\mathsf{SyS}}$, $\chi_\phi(X\times Y)$ is a regular language. This follows from theorem 11, which is basically a statement on normal subspaces.
3. For any two $x,y\in\Omega^*$, $\langle\Sigma,\phi\rangle\in\mathscr{S}_{\mathsf{SyS}}$, we can effectively compute the language $\chi_\phi(X\times Y)$ for arbitrarily small normal subspaces X,Y with center x and y, respectively; moreover, this language is regular.

So there is an interpretation where our notion of approximation is very useful: if we consider a string as representing a unique numeric value, we *cannot* determine whether the system can reach value y from value x; but we can determine, for any $\varepsilon > 0$, whether we can reach the interval $[y-\varepsilon, y+\varepsilon]$ from the interval $[x-\varepsilon, x+\varepsilon]$. But of course, approximation works in scenarios which are much more general than the interpretation of strings as real numbers. What is particularly interesting about this approach is that it provides a numeric approximation in a purely symbolic setting.

There is a third interpretation which is based on the encoding of program semantics as strings. As we have said, in order to be able to model a program in terms of relations in SyS, we need a linear hierarchy of variables which can be thought of in terms of importance: because influence goes only in one direction, the more to the left a variable is encoded in the string, the more important it is. Now the approximation means: we reach the desired configuration at least as regards the *n-most important parameters*, for $n \to \infty$. Moreover, we can effectively compute the set of sequences of computation steps which yield this result.

7.2 Complexity Issues

There is one most fundamental problem regarding the complexity of our approximation techniques, which is the following: given a finite set $\mathbf{R} \subseteq \mathsf{SyS}$ of relations, what is the complexity for computing $\sigma_n[\mathbf{R}^\oplus]$? On the positive side, we know from lemma 13 that $\sigma_n[\mathbf{R}]^\oplus = (\sigma_n[\mathbf{R}])^\oplus$; so the problem is surely computable, as $\sigma_n[\mathbf{R}]$ is a finite set of finite relations. $\sigma_n[\mathbf{R}]$ is rather easily computed in some way or other; the difficult thing is to compute the composition-closure. As $\sigma_n(R)$ is always a finite relation, this will be computable in a finite number of steps. The problem is to find the smallest n such that $\bigcup_{m\leq n}\{R_i : i \in I\}^m = \{R_i : i \in I\}^\oplus$. The bad thing is that this n depends on the size of M, where M is the underlying set from which tuple components in the relation are taken. For example, consider the relation $S_k = \{(n, n+1) : n \in \mathbb{N}, n < k\}$ for some $k \in \mathbb{N}$. This is a finite relation; still to get $\{S_k\}^\oplus$, we need no less than k iterated compositions. In our case, the underlying set is Ω^n for some alphabet Ω, and it is easy to see that this grows exponentially by factor $|\Omega|$ in terms of n. So assume for all $R \in \mathbf{R}$, we have $R \subseteq (\Omega \times \Omega)^*$; then in the worst case, in order to compute $\sigma_n[\mathbf{R}]^\oplus$ we need at least $|\Omega|^n$ computation steps. Note that this is only the most fundamental of all problems, which is necessary in order to compute for example $\chi_\phi(f_\omega \circ \sigma_n[F_R])$ etc. This is where the notions of direct and subdirect product become interesting, as for relations relations being (sub)directly decomposable, the approximation is much easier. This is however beyond our current scope.

The fact that computing approximations is exponential in terms of n is discouraging, so our preliminary results may not be fully satisfying. Still, in practical application many problems are not as hard as in theory (which always takes the worst case). In the theory of infinite automata, there is a necessary trade-off between expressiveness on the one hand and decidability issues on the other. Our notion of SyS relations and results on approximation, other than exploring the space of possibilities, might serve to establish some reasonable position in between the two.

8 Conclusion

In this paper, we have introduced the class SyS of synchronous subsequential relations and investigated its properties for the theory of infinite automata. SyS covers many possible models of computation, such as simple or embedded stacks (as in [13]) and many more. The underlying intuition is that there is a linear hierarchy of program variables such that computations performed on higher variables affect

computations on lower variables, but not vice versa. Though automata with primitive transitions relations in SyS have an undecidable reachability (and emptiness, universality and inclusion) problem, there is a way to approximate the problem which does not seem to work for more expressive classes such as the regular or subsequential relations, as it presupposes both a 1-1 correspondence of input- and output-letters, and independence of computations of later computation steps. The method in itself seems to be of some interest and is not necessarily bound to SyS (though it requires some fundamental properties of SyS), so one might further investigate on it. We have given a sketch of how one might compute the chances that an instance of a decision problem has a certain answer, and how the problem can be approximated in the sense of numeric distances and the hierarchy of program variables. Finally, the methods presented here are based on (algebraic) decompositions of relation monoids. There seems to be rather little work in this vein, so we hope the application of (relation-)algebraic methods to the theory of infinite automata, to which we have laid some fundamentals in this paper, might open the road to further interesting results.

References

[1] Jean Berstel (1979): *Transductions and Context-free Languages*. Teubner, Stuttgart, doi:10.1007/978-3-663-09367-1.

[2] Henning Bordihn, Henning Fernau, Markus Holzer, Vincenzo Manca & Carlos Martín-Vide (2006): *Iterated sequential transducers as language generating devices*. *Theor. Comput. Sci.* 369(1-3), pp. 67–81, doi:10.1016/j.tcs.2006.07.059.

[3] Stanley Burris & H. P. Sankappanavar (1981): *A Course in Universal Algebra*. *Graduate Texts in Mathematics* 78, Springer, doi:10.1007/978-1-4613-8130-3_7.

[4] Didier Caucal (1992): *On the Regular Structure of Prefix Rewriting*. *Theor. Comput. Sci.* 106(1), pp. 61–86, doi:10.1016/0304-3975(92)90278-N.

[5] Didier Caucal (2003): *On infinite transition graphs having a decidable monadic theory*. *Theor. Comput. Sci.* 290(1), pp. 79–115, doi:10.1016/S0304-3975(01)00089-5.

[6] Didier Caucal (2003): *On the transition graphs of turing machines*. *Theor. Comput. Sci.* 296(2), pp. 195–223, doi:10.1016/S0304-3975(02)00655-2.

[7] Edmund M. Clarke, Orna Grumberg & Doron Peled (2004): *Model checking*, 5. print. edition. MIT Press, Cambridge, Mass. [u.a.].

[8] Bruno Courcelle & Irène Durand (2012): *Automata for the verification of monadic second-order graph properties*. *J. Applied Logic* 10(4), pp. 368–409, doi:10.1016/j.jal.2011.07.001.

[9] Alan Pierce (2011): *Decision Problems on Iterated Length-Preserving Transducers*. Unpublished manuscript.

[10] Chloe Rispal (2002): *The synchronized graphs trace the context-sensitive languages*. *Electr. Notes Theor. Comput. Sci.* 68(6), pp. 55–70, doi:10.1016/S1571-0661(04)80533-4.

[11] Sasha Rubin (2008): *Automata Presenting Structures: A Survey of the Finite String Case*. *Bulletin of Symbolic Logic* 14(2), pp. 169–209, doi:10.2178/bsl/1208442827.

[12] Wolfgang Thomas (2001): *A Short Introduction to Infinite Automata*. In Werner Kuich, Grzegorz Rozenberg & Arto Salomaa, editors: *Developments in Language Theory*, *Lecture Notes in Computer Science* 2295, Springer, pp. 130–144, doi:10.1007/3-540-46011-X_10.

[13] David J. Weir (1992): *A Geometric Hierarchy Beyond Context-Free Languages*. *Theor. Comput. Sci.* 104(2), pp. 235–261, doi:10.1016/0304-3975(92)90124-X.