Decision Problems for Additive Regular Functions*

Rajeev Alur and Mukund Raghothaman

University of Pennsylvania
{alur, rmukund}@cis.upenn.edu

Abstract. Additive Cost Register Automata (ACRA) map strings to integers using a finite set of registers that are updated using assignments of the form "$x := y + c$" at every step. The corresponding class of *additive regular functions* has multiple equivalent characterizations, appealing closure properties, and a decidable equivalence problem. In this paper, we solve two decision problems for this model. First, we define the *register complexity* of an additive regular function to be the minimum number of registers that an ACRA needs to compute it. We characterize the register complexity by a necessary and sufficient condition regarding the largest subset of registers whose values can be made far apart from one another. We then use this condition to design a PSPACE algorithm to compute the register complexity of a given ACRA, and establish a matching lower bound. Our results also lead to a machine-independent characterization of the register complexity of additive regular functions. Second, we consider *two-player games over ACRAs*, where the objective of one of the players is to reach a target set while minimizing the cost. We show the corresponding decision problem to be EXPTIME-complete when the costs are non-negative integers, but undecidable when the costs are integers.

1 Introduction

Consider the following scenario: a customer frequents a coffee shop, and each time purchases a cup of coffee costing \$2. At any time, he may fill a survey, for which the store offers to give him a discount of \$1 for each of his purchases that month (including for purchases already made). We model this by the machine M_1 shown in figure 1.1. There are two states q_S and $q_{\neg S}$, indicating whether the customer has filled out the survey during the current month. There are three events to which the machine responds: C indicates the purchase of a cup of coffee, S indicates the completion of a survey, and # indicates the end of a month. The registers x and y track how much money the customer owes the establishment: in the state $q_{\neg S}$, the amount in x assumes that he will not fill out a survey that month, and the amount in y assumes that he will fill out a survey before the end

* The full version of this paper is available on the arXiv (arXiv:1304.7029). This research was partially supported by the NSF Expeditions in Computing award 1138996.

of the month. At any time the customer wishes to settle his account, the machine outputs the amount of money owed, which is always the value in the register x.

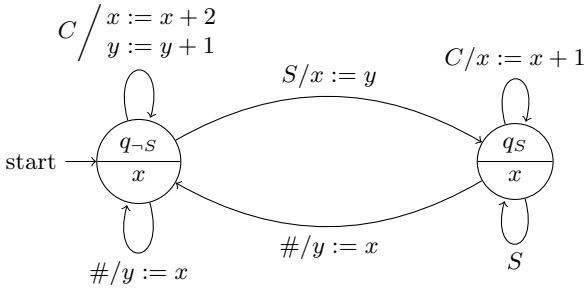


Fig. 1.1: ACRA M_1 models a customer in a coffee shop. It implements a function $f_1 : \{C, S, \#\}^* \to \mathbb{Z}$ mapping the purchase history of the customer to the amount he owes the store.

The automaton M_1 has a finite state space, and a finite set of integer-valued registers. On each transition, each register u is updated by an expression of the form "$u := v + c$", for some register v and constant $c \in \mathbb{Z}$. Which register will eventually contribute to the output is determined by the state after reading the entire input, and so the cost of an event depends not only on the past, but also on the future. Indeed, it can be shown that these machines are *closed under regular lookahead*, i.e. the register updates can be conditioned on regular properties of an as-yet-unseen suffix, for no gain in expressivity. The important limitation is that the register updates are test-free, and cannot examine the register contents.

The motivation behind the model is generalizing the idea of regular languages to quantitative properties of strings. A language $L \subseteq \Sigma^*$ is regular when it is accepted by a DFA. Regular languages are a robust class, permitting multiple equivalent representations such as regular expressions and as formulas in monadic second-order logic. Recently in [2], we proposed the model of regular functions: they are the MSO-definable transductions from strings to expression trees over some pre-defined grammar. The class of functions thus defined depends on the grammar allowed; the simplest is when the underlying domain is the set of integers $\mathbb{Z}$, and expressions involve constants and binary addition, and we call the resulting class *additive regular functions*. Additive regular functions have appealing closure properties, such as closure under linear combination, input reversal, and regular lookahead, and several analysis problems are efficiently decidable – such as containment, shortest paths and equivalence checking. The machine M_1 is an example of an *Additive Cost Register Automaton* (ACRA), and this class defines exactly the additive regular functions

Observe that the machine M_1 has two registers, and it is not immediately clear how (if it is even possible) to reduce this number. This is the first question that this paper settles: Given an ACRA M, how do we determine the minimum number

of registers needed by any ACRA to compute the function it defines, $[\![M]\!]$? We describe a property called register separation, and show that any equivalent ACRA needs at least k registers iff the registers of M are k-separable. It turns out that the registers of M_1 are 2-separable, and hence two registers are necessary. We then go on to show that determining k-separability is PSPACE-complete. Determining the register complexity is the natural analogue of the state minimization problem for DFAs [6].

The techniques used to analyse the register complexity allow us to state a result similar to the pumping lemma for regular languages: The register complexity of f is at least k iff for some m, we have strings $\sigma_0, \ldots, \sigma_m$, $\tau_1, \ldots, \tau_m$, suffixes $w_1, \ldots, w_k$, k distinct coefficient vectors $\mathbf{c}_1, \ldots, \mathbf{c}_k \in \mathbb{Z}^m$, and values $d_1, \ldots, d_k \in \mathbb{Z}$ so that for all vectors $\mathbf{x} \in \mathbb{N}^m$, $f(\sigma_0 \tau_1^{x_1} \sigma_1 \tau_2^{x_2} \ldots \sigma_m w_i) = \sum_j c_{ij} x_j + d_i$. Thus, depending on the suffix w_i, at least one of the cycles $\tau_1, \ldots, \tau_k$ contributes differently to the final cost.

Finally, we consider ACRAs with turn-based alternation. These are games where several objective functions are simultaneously computed, but only one of these objectives will eventually contribute to the output, based on the actions of both the system and its environment. Alternating ACRAs are thus related to multi-objective games and Pareto optimization [12], but are a distinct model because each run evaluates to a single value. We study the reachability problem in ACRA games: Given a budget k, is there a strategy for the system to reach an accepting state with cost at most k? We show that this problem is EXPTIME-complete when the incremental costs assume values from $\mathbb{N}$, and undecidable when the incremental costs are integer-valued.

Related work The traditional model of string-to-number transducers has been (non-deterministic) weighted automata (WA). Additive regular functions are equivalent to unambiguous weighted automata, and are therefore strictly sandwiched between weighted automata and deterministic WAs in expressiveness. Deterministic WAs are ACRAs with one register, and algorithms exist to compute the *state complexity* and for minimization [10]. Mohri [11] presents a comprehensive survey of the field. Recent work on the quantitative analysis of programs [5] also uses weighted automata, but does not deal with minimization or with notions of regularity. Data languages [7] are concerned with strings over a (possibly infinite) data domain $\mathbb{D}$. Recent models [3] have obtained Myhill-Nerode characterizations, and hence minimization algorithms, but the models are intended as acceptors, and not for computing more general functions. Turn-based weighted games [9] are ACRA games with a single register, and in this special setting, it is possible to solve non-negative optimal reachability in polynomial time. Of the techniques used in the paper, difference bound invariants are a standard tool. However when we need them, in section 3, we have to deal with disjunctions of such constraints, and show termination of invariant strengthening – to the best of our knowledge, the relevant problems have not been solved before.

Outline of the paper We define the automaton model in section 2. In section 3, we introduce the notion of separability, and establish its connection to register complexity. In section 4, we show that determining the register complexity is PSPACE-complete. Finally, in section 5, we study ACRA reachability games – in particular, that ACRA ($\mathbb{Z}$) games are undecidable, and that ACRA ($\mathbb{N}$) reachability games are EXPTIME-complete.

2 Additive Regular Functions

We will use additive cost register automata as the working definition of additive regular functions, i.e. a function[1] $f : \Sigma^* \to \mathbb{Z}_\perp$ is regular iff it is implemented by an ACRA. An ACRA is a deterministic finite state machine, supplemented by a finite number of integer-valued registers. Each transition specifies, for each register u, a test-free update of the form "$u := v + c$", for some register v, and constant $c \in \mathbb{Z}$. Accepting states are labelled with output expressions of the form "$v + c$".

Definition 1. *An ACRA is a tuple $M = (Q, \Sigma, V, \delta, \mu, q_0, F, \nu)$, where Q is a finite non-empty set of states, Σ is a finite input alphabet, V is a finite set of registers, $\delta : Q \times \Sigma \to Q$ is the state transition function, $\mu : Q \times \Sigma \times V \to V \times \mathbb{Z}$ is the register update function, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accepting states, and $\nu : F \to V \times \mathbb{Z}$ is the output function.*

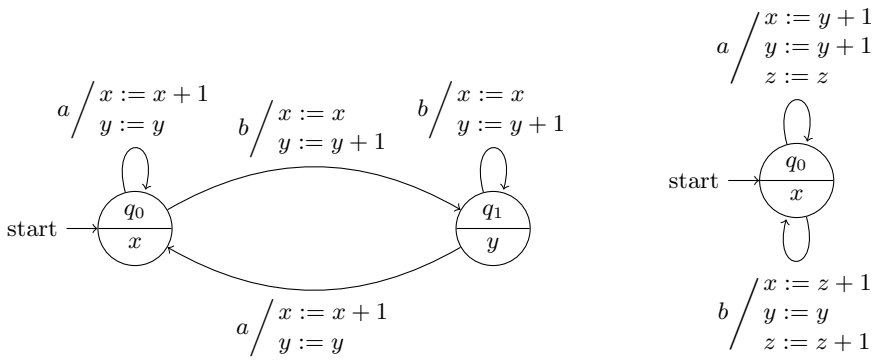
The configuration of the machine is a pair $\gamma = (q, val)$, where q is the current state, and $val : V \to \mathbb{Z}$ maps each register to its value. Define $(q, val) \to^a (q', val')$ iff $\delta(q, a) = q'$ and for each $u \in V$, if $\mu(q, a, u) = (v, c)$, then $val'(u) = val(v) + c$.

The machine M implements a function $[\![M]\!] : \Sigma^* \to \mathbb{Z}_\perp$ defined as follows. For each $\sigma \in \Sigma^*$, let $(q_0, val_0) \to^\sigma (q_f, val_f)$, where $val_0(v) = 0$ for all v. If $q_f \in F$ and $\nu(q_f) = (v, c)$, then $[\![M]\!](\sigma) = val_f(v) + c$. Otherwise $[\![M]\!](\sigma) = \perp$.

We will write $val(u, \sigma)$ for the value of a register u after the machine has processed the string σ starting from the initial configuration. In the rest of this section, we summarize some known results about ACRAs [2]:

Equivalent characterizations Additive regular functions are equivalent to unambiguous weighted automata [11] over the tropical semiring. These are nondeterministic machines with a single counter. Each transition increments the counter by an integer c, and accepting states have output increments, also integers. The unambiguous restriction requires that there be a single accepting path for each string in the domain, thus the "min" operation of the tropical semiring is unused. Recently, streaming tree transducers [1] have been proposed as the regular model for string-to-tree transducers – ACRAs are equivalent in expressiveness to MSO-definable string-to-term transducers with binary addition as the base grammar.

[1] By convention, we represent a partial function $f : A \to B$ as a total function $f : A \to B_\perp$, where $B_\perp = B \cup \{\perp\}$, and $\perp \notin B$ is the "undefined" value.



(a) M_2. (b) M_3.

Fig. 2.1: ACRAs M_2 and M_3 operate over the input alphabet $\Sigma = \{a, b\}$. Both implement the function defined as $f_2(\epsilon) = 0$, and for all σ, $f_2(\sigma a) = |\sigma a|_a$, and $f_2(\sigma b) = |\sigma b|_b$. Here $|\sigma|_a$ is the number of occurrences of the symbol a in the string σ.

Closure properties What makes additive[2] regular functions interesting to study is their robustness to various manipulations:

1. For all $c \in \mathbb{Z}$, if f_1 and f_2 are regular functions, then so are $f_1 + f_2$ and cf_1.
2. If f is a regular function, then f_{rev} defined as $f_{rev}(\sigma) = f(\sigma^{rev})$ is also regular.
3. If f_1 and f_2 are regular functions, and L is a regular language, then the function f defined as $f(\sigma) = \text{if } \sigma \in L, \text{ then } f_1(\sigma), \text{ else } f_2(\sigma)$ is also regular.
4. ACRAs are closed under regular lookahead, i.e. even if the machine were allowed to make decisions based on a regular property of the suffix rather than simply the next input symbol, there would be no increase in expressiveness.

Analysis problems Given ACRAs M_1 and M_2, equivalence-checking and the min-cost problem ($\min_{\sigma \in \Sigma^*} [\![M]\!](\sigma)$) can be solved in polynomial time. It follows then that containment (for all σ, $[\![M_1]\!](\sigma) \leq [\![M_2]\!](\sigma)$) also has a polynomial time algorithm.

3 Characterizing the Register Complexity

The register complexity of an additive regular function f is the minimum number of registers an ACRA needs to compute it. For example the register complexity of both $[\![M_1]\!]$ in figure 1.1 and $[\![M_2]\!]$ in figure 2.1a is 2. Computing the register complexity is the first problem we solve, and is the subject of this section and the next.

[2] We will often drop the adjective "additive", and refer simply to regular functions.

Definition 2. *Let $f : \Sigma^* \to \mathbb{Z}_\perp$ be a regular function. The register complexity of f is the smallest number k such that there is an ACRA M implementing f with only k registers.*

Informally, the registers of M are separable in some state q if their values can be pushed far apart. For example, consider the registers x and y of M_1 in the state q_0. For any constant c, there is a string $\sigma = C^c$ leading to q_0 so that $|val(x, \sigma) - val(y, \sigma)| \geq c$. In formalizing this idea, we need to distinguish registers that are live in a given state, i.e. those that can potentially contribute to the output. For example, M_1 could be augmented with a third register z tracking the length of the string processed. However, the value of z would be irrelevant to the computation of f_1. Informally, a register v is live[3] in a state q if for some suffix $\sigma \in \Sigma^*$, on processing σ starting from q, the initial value of v is what influences the final output.

Definition 3. *Let $M = (Q, \Sigma, V, \delta, \mu, q_0, \nu)$ be an ACRA. The registers of M are* k-separable *if there is some state q, and a subset $U \subseteq V$ so that*

1. *$|U| = k$, all registers $v \in U$ are live in q, and*
2. *for all $c \in \mathbb{Z}$, there is a string σ, such that $\delta(q_0, \sigma) = q$ and for all distinct $u, v \in U$, $|val(u, \sigma) - val(v, \sigma)| \geq c$.*

The registers of a machine M are not k-separable if at every state q, and subset U of k live registers, there is a constant c such that for all strings σ to q, $|val(u, \sigma) - val(v, \sigma)| < c$, for some distinct $u, v \in U$. Note that the specific registers which are close may depend on σ. For example, in the machine M_3 from figure 2.1b, if a string σ ends with an a, then x and y will have the same value, while if the last symbol was a b, then x and z are guaranteed to be equal.

Theorem 1. *Let $f : \Sigma^* \to \mathbb{Z}_\perp$ be a function defined by an ACRA M. Then the register complexity of f is at least k iff the registers of M are k-separable.*

We now sketch the proofs for each direction.

k-separability implies a lower bound on the register complexity

Consider the machine M_1 from figure 1.1. Here $k = 2$, and the registers x and y are separated in the state $q_{\neg S}$. Let $\sigma_1 = \epsilon$, i.e. the empty string, and $\sigma_2 = S$ – these are suffixes which, when starting from $q_{\neg S}$, "extract" the values currently in x and y respectively.

Now suppose an equivalent counter-example machine M' is proposed with only one register v. At each state q' of M', observe the "effect" of processing suffixes σ_1 and σ_2. Each of these can be summarized by an expression of the form $v + c_{q'i}$ for $i \in \{1, 2\}$, the current value of register v, and $c_{q'i} \in \mathbb{Z}$. Thus, the outputs differ by no more than $|(v + c_{q'1}) - (v + c_{q'2})| \leq |c_{q'1}| + |c_{q'2}|$. Fix $n = \max_{q'}(|c_{q'1}| + |c_{q'2}|)$, and observe that for all σ, $|[\![M']\!](\sigma\sigma_1) - [\![M']\!](\sigma\sigma_2)| \leq n$. However, for $\sigma = C^{n+1}$, we know that $|f_1(\sigma\sigma_1) - f_1(\sigma\sigma_2)| > n$, so M' cannot be equivalent to M_1. This argument can be generalized to obtain:

[3] Live registers are formally defined in the full version of this paper.

Lemma 1. *Let M be an ACRA whose registers are k-separable. Then the register complexity of the implemented function f is at least k.*

Non-separability permits register elimination

Consider an ACRA M whose registers are not k-separable. We can then state an invariant at each state q: there is a constant c_q such that for every subset $U \subseteq V$ of live registers with $|U| = k$, and for every string σ with $\delta(q_0, \sigma) = q$, there must exist distinct $u, v \in U$ with $|val(u, \sigma) - val(v, \sigma)| < c$. For example, with 3 registers x, y, z, this invariant would be $\exists c, |x - y| < c \vee |y - z| < c \vee |z - x| < c$. We will construct a machine M', where each state $q' = (q, C, \mathbf{v})$ has 3 components: the first component q is the state of the original machine, and C identifies some term (not necessarily unique) in the disjunction which is currently satisfied. Now for example, if we know that $|x - y| < c$, then it suffices to explicitly maintain the value of only one register, and the (bounded) difference can be stored in the state – this is the third component $\mathbf{v}$.

Since we need to track these register differences during the execution, the invariants must be inductive: if D_q and $D_{q'}$ are the invariants at states q and q' respectively, and $q \to^a q'$ is a transition in the machine, then it must be the case that $D_q \implies \text{WP}(D_{q'}, q, a)$. Here WP refers to the standard notion of the weakest precondition from program analysis: $\text{WP}(D_{q'}, q, a)$ is exactly that set of variable valuations val so that $(q, val) \to^a (q', val')$ for some $D_{q'}$-satisfying valuation val'.

The standard technique to make a collection of invariants inductive is strengthening: if $D_q \not\implies \text{WP}(D_{q'}, q, a)$, then D_q is replaced with $D_q \wedge \text{WP}(D_{q'}, q, a)$, and this process is repeated at every pair of states until a fixpoint is reached. This procedure is seeded with the invariants asserting non-separability. However, before the result of this back-propagation can be used in our arguments, we must prove that the method terminates – this is the main technical problem solved in this section.

We now sketch a proof of this termination claim for a simpler class of invariants. Consider the class of difference-bound constraints – assertions of the form $C = \bigwedge_{u,v \in V} a_{uv} < u - v < b_{uv}$, where for each u, v, $a_{uv}, b_{uv} \in \mathbb{Z}$ or $a_{uv}, b_{uv} \in \{-\infty, \infty\}$. When in closed form[4], C induces an equivalence relation $\equiv_C$ over the registers: $u \equiv_C v$ iff $a_{uv}, b_{uv} \in \mathbb{Z}$. Let C and C' be some pair of constraints such that $C \not\implies C'$. Then the assertion $C \wedge C'$ is strictly stronger than C. Either $C \wedge C'$ relates a strictly larger set of variables – $\equiv_C \subsetneq \equiv_{C \wedge C'}$ – or (if $\equiv_C = \equiv_{C \wedge C'}$) for some pair of registers u, v, the bounds $a'_{uv} < u - v < b'_{uv}$ imposed by $C \wedge C'$ are a strict subset of the bounds $a_{uv} < u - v < b_{uv}$ imposed by C. Observe that the first type of strengthening can happen at most $|V|^2$ times, while the second type of strengthening can happen only after a_{uv}, b_{uv} are established for a pair of registers u, v, and can then happen at most $b_{uv} - a_{uv}$ times. Thus the process of repeated invariant strengthening must terminate. This argument can be generalized to disjunctions of difference-bound constraints, and we conclude:

[4] For all $u, v \in V$, $a_{uv} = -b_{vu}$, and for all $u, v, w \in V$, $a_{uv} + a_{vw} \leq a_{uw}$.

Lemma 2. *Consider an ACRA M whose registers are not k-separable. Then, we can effectively construct an equivalent machine M' with only $k-1$ registers.*

4 Computing the Register Complexity

4.1 Computing the register complexity is in PSPACE

We reduce the problem of determining the register complexity of $[\![M]\!]$ to one of determining reachability in a directed "register separation" graph with $O\left(|Q|\,2^{|V|^2}\right)$ nodes. The presence of an edge in this graph can be determined in polynomial space, and thus we have a PSPACE algorithm to determine the register complexity. Otherwise, if polynomial time algorithms are used for graph reachability and 1-counter 0-reachability, the procedure runs in time $O\left(c^3\,|Q|^4\,2^{4|V|^2}\right)$, where c is the largest constant in the machine.

We first generalize the idea of register separation to that of separation relations: an arbitrary relation $\| \subseteq V \times V$ separates a state q if for every $c \in \mathbb{Z}$, there is a string σ so that $\delta(q_0, \sigma) = q$, and whenever $u \parallel v$, $|val(u,\sigma) - val(v,\sigma)| \geq c$. Thus, the registers of M are k-separable iff for some state q and some subset U of live registers at q, $|U| = k$ and $\{(u,v) \mid u, v \in U, u \neq v\}$ separates q.

Consider a string $\tau \in \Sigma^*$, so for some q, $\delta(q,\tau) = q$. Assume also that:

1. For every register u in the domain or range of $\|$, $\mu(q,\tau,u) = (u, c_u)$, for some $c_u \in \mathbb{Z}$, and
2. for some pair of registers x, y, $\mu(q,\tau,x) = (x,c)$ and $\mu(q,\tau,y) = (y,c')$ for distinct c, c'.

Thus, every pair of registers that is already separated is preserved during the cycle, and some new pair of registers is incremented differently. We call such strings τ "separation cycles" at q. They allow us to make conclusions of the form: If $\|$ separates q, then $\| \cup \{(x,y)\}$ also separates q.

Now consider a string $\sigma \in \Sigma^*$, such that for some q, q', $\delta(q,\sigma) = q'$. Pick arbitrary relations $\|$, $\|'$, and assume that whenever $u' \parallel' v'$, and $\mu(q,\sigma,u') = (u, c_u)$, $\mu(q,\sigma,v') = (v, c_v)$, we have $u \parallel v$. We can then conclude that if $\|$ separates q, then $\|'$ separates q' We call such strings σ "renaming edges" from $(q,\|)$ to $(q',\|')$.

We then show that if $\|$ separates q and $\|$ is non-empty, then there is a separation cycle-renaming edge sequence to $(q,\|)$ from some strictly smaller separation $(q',\|')$. Thus, separation at each node can be demonstrated by a sequence of separation cycles with renaming edges in between, and thus we reduce the problem to that of determining reachability in an exponentially large register separation graph. Finally, we show that each type of edge can be determined in PSPACE.

Theorem 2. *Given an ACRA M and a number k, there is a* PSPACE *procedure to determine whether its register complexity is at least k.*

4.2 Pumping lemma for ACRAs

The following theorem is the interpretation of a path through the register separation graph. Given a regular function f of register complexity at least k, it guarantees the existence of m cycles $\tau_1, \ldots, \tau_m$, serially connected by strings σ_0, $\ldots$, σ_m, so that based on one of k suffixes $w_1, \ldots, w_k$, the cost paid on one of the cycles must differ. These cycles are actually the separation cycles discussed earlier, and intermediate strings σ_i correspond to the renaming edges. Consider for example, the function f_2 from figure 2.1, and let $\sigma_0 = \epsilon$, $\tau_1 = aab$, and $\sigma_1 = \epsilon$. We can increase the difference between the registers x and y to arbitrary amounts by pumping the cycle τ_1. Now if the suffixes are $w_1 = a$, and $w_2 = b$, then the choice of suffix determines the "cost" paid on each iteration of the cycle.

Theorem 3. *A regular function $f : \Sigma^* \to \mathbb{Z}_\perp$ has register complexity at least k iff there exist strings $\sigma_0, \ldots, \sigma_m, \tau_1, \ldots, \tau_m$, and suffixes $w_1, \ldots, w_k$, and k distinct coefficient vectors $\mathbf{c}_1, \ldots, \mathbf{c}_k \in \mathbb{Z}^m$, and values $d_1, \ldots, d_k \in \mathbb{Z}$ so that for all $x_1, \ldots, x_m \in \mathbb{N}$,*

$$f\left(\sigma_0 \tau_1^{x_1} \sigma_1 \tau_2^{x_2} \ldots \sigma_m w_i\right) = \sum_j c_{ij} x_j + d_i.$$

4.3 Computing the register complexity is PSPACE-hard

We reduce the DFA intersection non-emptiness checking problem [8] to the problem of computing the register complexity. Let $A = (Q, \Sigma, \delta, q_0, \{q_f\})$ be a DFA. Consider a single-state ACRA M with input alphabet Σ. For each state $q \in Q$, M maintains a register v_q. On reading a symbol $a \in \Sigma$, M updates $v_q := v_{\delta(q,a)}$, for each q. Observe that this is simulating the DFA in reverse: if we start with a special tagged value in v_{q_f}, then after processing σ, that tag is in the register v_{q_0} iff σ^{rev} is accepted by A. Also observe that doing this in parallel for all the DFAs no longer requires an exponential product construction, but only as many registers as a linear function of the input size. We use this idea to construct in polynomial time an ACRA M whose registers are $(k+2)$-separable iff there is a string $\sigma \in \Sigma^*$ which is simultaneously accepted by all the DFAs. Therefore:

Theorem 4. *Given an ACRA M and a number k, deciding whether the register complexity of $[\![M]\!]$ is at least k is* PSPACE*-hard.*

5 Games over ACRAs

We now study games played over ACRAs. We extend the model of ACRAs to allow alternation – in each state, a particular input symbol may be associated with multiple transitions. The system picks the input symbol to process, while the environment picks the specific transition associated with this input symbol. Accepting states are associated with output functions, and the system may choose to end the game in any accepting state. Given a budget k, we wish to decide

whether the system has a winning strategy with worst-case cost no more than k. We show that ACRA games are undecidable when the incremental costs are integer-valued, and EXPTIME-complete when the incremental costs are from $\mathbb{D} = \mathbb{N}$.

Definition 4. *An ACRA $(\mathbb{D})$ reachability game G is a tuple $(Q, \Sigma, V, \delta, \mu, q_0, F, \nu)$, where Q, Σ, and V are finite sets of states, input symbols and registers respectively, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $\mu : \delta \times V \to V \times \mathbb{D}$ is the register update function, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accepting states, and $\nu : F \to V \times \mathbb{D}$ is the output function.*

The game configuration is a tuple $\gamma = (q, val)$, where $q \in Q$ is the current state, and $val : V \to \mathbb{D}$ is the current register valuation. A run π is a (possibly infinite) sequence of game configurations $(q_1, val_1) \to^{a_1} (q_2, val_2) \to^{a_2} \cdots$ with the property that

1. *the transition $q_i \to^{a_i} q_{i+1} \in \delta$ for each i, and*
2. *$val_{i+1}(u) = val_i(v) + c$, where $\mu(q_i \to^{a_i} q_{i+1}, u) = (v, c)$, for each register u and for each transition i.*

A strategy is a function $\theta : Q^* \times Q \to \Sigma$ that maps a finite history $q_1 q_2 \ldots q_n$ to the next symbol $\theta(q_1 q_2 \ldots q_n)$.

Definition 5. *A run π is consistent with a strategy θ if for each i, $\theta(q_1 q_2 \ldots q_i) = a_i$. θ is winning from a configuration (q, val) with a budget of $k \in \mathbb{D}$ if for every consistent run π starting from $(q_1, val_1) = (q, val)$, for some i, $q_i \in F$ and $\nu(q_i, val_i) \leq k$.*

For greater readability, we write tuples $(q, a, q') \in \delta$ as $q \to^a q'$. If $q \in F$, and val is a register valuation, we write $\nu(q, val)$ for the result $val(v) + c$, where $\nu(q) = (v, c)$. When we omit the starting configuration for winning strategies it is understood to mean the initial configuration (q_0, val_0) of the ACRA.

5.1 ACRA $(\mathbb{N})$ reachability games can be solved in EXPTIME

Consider the simpler class of (unweighted) graph reachability games. These are played over a structure $G^f = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite state space, and Σ is the input alphabet. $\delta \subseteq Q \times \Sigma \times Q$ is the state transition relation, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of accepting states. If the input symbol $a \in \Sigma$ is played in a state q, then the play may adversarially proceed to any state q' so that $(q, a, q') \in \delta$. The system can force a win if every run compatible with some strategy $\theta^f : Q^* \times Q \to \Sigma$ eventually reaches a state $q_f \in F$. Such games can be solved by a recursive back-propagation algorithm – corresponding to model checking the formula $\mu X \cdot \left(F \vee \bigvee_{a \in \Sigma} [a] X\right)$ – in time $O(|Q|\,|\Sigma|)$. In such games, whenever there is a winning strategy, there is a memoryless winning strategy θ_{small} which guarantees that no state is visited twice.

From every ACRA $(\mathbb{N})$ reachability game $G = (Q, \Sigma, V, \delta, \mu, q_0, F, \nu)$, we can project out an unweighted graph reachability game $G^f = (Q, \Sigma, \delta, q_0, F)$. Also, G^f has a winning strategy iff for some $k \in \mathbb{N}$, G has a k-winning strategy.

Consider the cost of θ_{small} (computed for G^f) when used with G. Since no run ever visits the same state twice, θ_{small} is $c_0 |Q|$-winning, where c_0 is the largest constant appearing in G. We have thus established an upper-bound on the optimal reachability strategy, if it exists.

Given an upper-bound $k \in \mathbb{N}$, we would like to determine whether a winning strategy θ exists within this budget. Because the register increments are non-negative, once a register v achieves a value larger than k, it cannot contribute to the final output on any suffix σ permitted by the winning strategy. We thus convert G into an unweighted graph reachability G_k^f, where the value of each register is explicitly tracked in the state, as long as it is in the set $\{0, 1, \ldots, k\}$. This game can be solved for the optimal reachability strategy, and so we have:

Theorem 5. *The optimal strategy θ for an ACRA $(\mathbb{N})$ reachability game G can be computed in time $O\left(|Q|\,|\Sigma|\,2^{|V|\log c_0|Q|}\right)$, where c_0 is the largest constant appearing in the description of G.*

Note that the optimal strategy in ACRA $(\mathbb{N})$ games need not be memoryless: the strategy may visit a state again with a different register valuation. However, the strategy θ constructed in the proof of the above theorem is memoryless given the pair (q, val) of the current state and register valuation.

5.2 Hardness of solving ACRA $(\mathbb{D})$ reachability games

We reduce the halting problem for two-counter machines to the problem of solving an ACRA $(\mathbb{Z})$ reachability game. Informally, we construct a game G_M given a two-counter machine M so that the player has a 0-winning strategy through G_M iff M halts. This strategy encodes the execution of M, and the adversary verifies that the run is valid. A similar idea is used to show that deciding ACRA $(\mathbb{N})$ reachability games is EXPTIME-hard. The reduction in that case proceeds from the halting problem for linearly bounded alternating Turing machines [4]. Given such a machine M, we construct in polynomial time a game gadget G_M where the only strategy is to encode the runs of the Turing machine.

Theorem 6. *Determining whether there is a winning strategy with budget k in an ACRA $(\mathbb{N})$ reachability game is* EXPTIME*-hard.*

Theorem 7. *Determining whether there is a winning strategy with budget k in an ACRA $(\mathbb{Z})$ reachability game is undecidable.*

6 Conclusion

In this paper, we studied two decision problems for additive regular functions: determining the register complexity, and alternating reachability in ACRAs. The register complexity of an additive regular function f is the smallest number k so there is some ACRA implementing f with only k registers. We developed an abstract characterization of the register complexity as separability and showed

that computing it is PSPACE-complete. We then studied the reachability problem in alternating ACRAs, and showed that it is undecidable for ACRA $(\mathbb{Z})$ and EXPTIME-complete for ACRA $(\mathbb{N})$ games. Future work includes proving similar characterizations and providing algorithms for register minimization in more general models such as streaming string transducers. String concatenation does not form a commutative monoid, and the present paper is restricted to unary operators (increment by constant), and so the technique does not immediately carry over. Another interesting question is to find a machine-independent characterization of regular functions $f : \Sigma^* \to \mathbb{Z}_\perp$. A third direction of work would be extending these ideas to trees and studying their connection to alternating ACRAs.

References

1. Rajeev Alur and Loris D'Antoni. Streaming tree transducers. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 42–53. Springer, 2012.
2. Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. *To appear in the 28th Annual Symposium on Logic in Computer Science*, Full version available at http://www.cis.upenn.edu/~alur/rca12.pdf, 2013.
3. Mikolaj Bojanczyk, Bartek Klin, and Slawomir Lasota. Automata with group actions. In *26th Annual Symposium on Logic in Computer Science*, pages 355–364, 2011.
4. Ashok Chandra, Dexter Kozen, and Larry Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, January 1981.
5. Krishnendu Chatterjee, Laurent Doyen, and Thomas Henzinger. Quantitative languages. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic*, volume 5213 of *Lecture Notes in Computer Science*, pages 385–400. Springer, 2008.
6. John Hopcroft, Rajeev Motwani, and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation.* Prentice Hall, 3$^{\text{rd}}$ edition, 2006.
7. Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.
8. Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science, 1977.*, pages 254–266, 31 Oct – 2 Nov 1977.
9. Nicolas Markey. Weighted automata: Model checking and games. *Lecture notes*, Available at http://www.lsv.ens-cachan.fr/~markey/Teaching/MPRI/2008-2009/MPRI-2.8b-4.pdf, 2008.
10. Mehryar Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234:177–201, 2000.
11. Mehryar Mohri. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science, pages 213–254. Springer, 2009.
12. Christos Papadimitriou and Mihalis Yannakakis. Multiobjective query optimization. In *Proceedings of the 20th Symposium on Principles of Database Systems*, PODS '01, pages 52–59. ACM, 2001.