Weighted Rewriting: Semiring Semantics for Abstract Reduction Systems

Emma Ahrens
RWTH Aachen University, Aachen, Germany

Jan-Christoph Kassing
RWTH Aachen University, Aachen, Germany

Jürgen Giesl
RWTH Aachen University, Aachen, Germany

Joost-Pieter Katoen
RWTH Aachen University, Aachen, Germany

Abstract

We present novel semiring semantics for abstract reduction systems (ARSs). More precisely, we provide a weighted version of ARSs, where the reduction steps induce weights from a semiring. Inspired by provenance analysis in database theory and logic, we obtain a formalism that can be used for provenance analysis of arbitrary ARSs. Our semantics handle (possibly unbounded) non-determinism and possibly infinite reductions. Moreover, we develop several techniques to prove upper and lower bounds on the weights resulting from our semantics, and show that in this way one obtains a uniform approach to analyze several different properties like termination, derivational complexity, space complexity, safety, as well as combinations of these properties.

2012 ACM Subject Classification Theory of computation → Rewrite systems; Theory of computation → Equational logic and rewriting; Theory of computation → Logic and verification

Keywords and phrases Rewriting, Semirings, Semantics, Termination, Verification

Related Version *See [2]. Full version, including all proofs*: `https://arxiv.org/abs/2505.08496`

Funding funded by the DFG Research Training Group 2236 UnRAVeL

1 Introduction

Rewriting is a prominent formalism in computer science and notions like termination, complexity, and confluence have been studied for decades for abstract reduction systems (ARSs). Moreover, typical problems in computer science like evaluation of a database query or a logical formula can be represented as a reduction system.

In this paper, we tackle the question whether numerous different analyses in the area of rewriting, computation, logic, and deduction are "inherently similar", i.e., whether they can all be seen as special instances of a uniform framework. Analogous research has been done, e.g., in the database and logic community, where there have been numerous approaches to *provenance analysis*, i.e., to not just analyze satisfiability but also explainability of certain results (see, e.g., [14, 15, 20]). In these works, semirings are often used to obtain information beyond just satisfiability of a query. For instance, they may be used to compute the confidence in an answer or to calculate the cost of proving satisfiability.

▶ **Example 1** (Provenance Analysis in Databases). Consider two tables R, P in a database over the universe $U = \{a, b\}$ with $R = U$ and $P = U \times U$ (where each atomic fact, such as Ra or Pab, has a cost in $\mathbb{N} \cup \{\infty\}$), and the formula ψ which represents a database query.

$R =$

	cost
a	2
b	∞

$P =$

		cost			cost
a	a	2	b	a	∞
a	b	7	b	b	10

$\psi = Ra \wedge (Pab \vee Pbb)$

Our aim is to calculate the maximal cost of proving ψ. For the alternative use of information, i.e., for disjunctions $\psi = \varphi \vee \varphi'$, we take the maximal cost of proving that φ or φ' are satisfied. For the joint use of information, i.e., for conjunctions $\psi = \varphi \wedge \varphi'$, we take the sum of costs for proving that φ and φ' hold. We can use the *arctic semiring* $\mathbb{S}_{\mathsf{arc}} = (\mathbb{N}^{\pm\infty}, \max, +, -\infty, 0)$ to formalize this situation (i.e., we consider $\mathbb{N} \cup \{-\infty, \infty\}$ with the operations max and + whose identity elements are $-\infty$ and 0, respectively):

- For an atom α, we set $[\![\alpha]\!] = c$, where c is the cost of the atom α.[1]
- For formulas φ, φ', we set $[\![\varphi \vee \varphi']\!] = \max\{[\![\varphi]\!], [\![\varphi']\!]\}$ and $[\![\varphi \wedge \varphi']\!] = [\![\varphi]\!] + [\![\varphi']\!]$.

This leads to a maximal cost of $[\![\psi]\!] = [\![Ra]\!] + [\![Pab \vee Pbb]\!] = 2 + \max\{7, 10\} = 12$ for proving ψ. We can use a different semiring to calculate a different property. For example, the confidence that a formula ψ holds is computable via the *confidence semiring* $\mathbb{S}_{\mathsf{conf}} = ([0, 1], \max, \cdot, 0, 1)$, where all atomic facts are given a certain confidence score.

In general, to compute the weight $[\![a]\!]$ of an object a, one defines an *interpretation of the facts or atoms* (e.g., the definition of $[\![\alpha]\!]$) which maps objects to elements of the semiring, and an *aggregator function* for each reduction step (e.g., the rules for $[\![\varphi \wedge \varphi']\!]$ and $[\![\varphi \vee \varphi']\!]$ above) which operates on elements of the semiring.

In this work we generalize the idea of evaluating a database query within a given semiring to ARSs. As usual, an ARS is a set A together with a binary relation $\to$ denoting reductions. Note that we have to allow reductions from a single object to multiple ones, as we may have to consider multiple successors (e.g., φ and φ' for the formula $\varphi \wedge \varphi'$) in order to define the weight of $\varphi \wedge \varphi'$, whereas in classical ARSs, objects are reduced to single objects. This leads to the notion of *sequence ARSs*. Similar ideas have been used for probabilistic ARSs [3, 11, 12], where a reduction relates a single object to a multi-distribution over possible results. We will see that probabilistic ARSs can indeed also be expressed using our formalism.

▶ **Example 2** (Provenance Analysis for ARSs). The formulas from Ex. 1 fit into the concept of sequence ARSs: The set A contains all propositional, negation-free formulas over the atomic facts $\mathtt{NF}_{\to} = \{Ru_1, Pu_1u_2 \mid u_1, u_2 \in U\} \subset A$ (the normal forms of the relation $\to$) and the relation $\to$ is defined as $\varphi \wedge \psi \to [\varphi, \psi]$ and $\varphi \vee \psi \to [\varphi, \psi]$, where $[\varphi, \psi]$ denotes the sequence containing φ as first and ψ as second element. Given a semiring, aggregator functions for the reductions steps, and an interpretation of the normal forms, we calculate the weight of a formula as in the previous example. See Sect. 3 for the formal definition.

In order to handle arbitrary ARSs, we have to deal with non-terminating reduction sequences and with (possibly unbounded) non-determinism. For that reason, to ensure that our semantics are well defined, we consider semirings where the natural order (that is induced by addition of the semiring) forms a complete lattice.

In most applications of semiring semantics in logic [15, 20], a higher "truth value" w.r.t. the natural order is more desirable, e.g., for the confidence semiring $\mathbb{S}_{\mathsf{conf}}$ one would like to obtain a value close to the most desirable confidence 1. However, in the application of software verification, it is often the reverse, e.g., for computing the runtime of a reduction or when considering the costs as in Ex. 1 for the arctic semiring $\mathbb{S}_{\mathsf{arc}} = (\mathbb{N}^{\pm\infty}, \max, +, -\infty, 0)$.

[1] We assume that our formulas do not use negation, a typical restriction for semiring semantics for logic.

While every weight $< \infty$ may still be acceptable, the aim is to prove *boundedness*, i.e., that the maximum ∞ (an infinite cost) cannot occur. For example, boundedness can imply termination of the underlying ARS, it can ensure that certain bad states cannot be reached (safety), etc. By considering tuples over different semirings, we can combine multiple analyses into a single combined framework analyzing combinations of properties, where simply performing each analysis separately fails. In Sect. 5, we give sufficient conditions for boundedness and show that the interpretation method, a well-known method to, e.g., prove termination [5, 26, 29] or complexity bounds [9, 10, 22] of ARSs, can be generalized to a sound and (regarding continuous complete lattice semirings) complete technique to prove boundedness. On the other hand, it is also of interest to prove worst-case lower bounds on weights in order to find bugs or potential attacks (e.g., inputs which lead to a long runtime). Moreover, if one determines both worst-case upper and lower bounds, this allows to check whether the bounds are (asymptotically) exact. Thus, we present a technique to find counterexamples with maximal weight in Sect. 6.

So in this paper, we define a uniform framework in the form of *weighted ARSs*, whose special instances correspond to different semiring interpretations. This indeed shows the similarity between numerous analyses in rewriting, computation, logic, and deduction, because they can all be represented within our new framework. In particular, this uniform framework allows to adapt techniques which were developed for one special instance in order to use them for other special instances. While the current paper is a first (theoretical) contribution in this direction, eventually this may also improve the automation of the analysis for certain instances.

Main Results of the Paper:

- We generalize abstract reduction systems to a weighted version (Def. 12) that is powerful enough to express complex notions like termination, complexity, and safety of (probabilistic) rewriting, and even novel combinations of such properties (see Sect. 4).
- We provide several sufficient criteria that ensure boundedness (Thm. 25 and 29).
- Moreover, we give a sound (and complete in case of continuous semirings) technique based on the well-known interpretation method to prove boundedness, i.e., to show that the weight of every object in the ARS is smaller than the maximum of the semiring (Thm. 32).
- Finally, we develop techniques to approximate the weights (Thm. 37) and to detect counterexamples that show unboundedness (Thm. 41).

Related Work: Semirings are actively studied in the database and the logic community. See [20] for the first paper on *semiring provenance* and [19, 21] for further surveys. Moreover, a uniform framework via semirings has been developed in the context of *weighted automata* [16], which has led to a wealth of extensions and practical applications, e.g., in digital image compression and model checking. There is also work on semiring semantics for declarative languages like, e.g., Datalog [23], which presents properties of the semiring that ensure upper bounds on how fast a Datalog program can be evaluated. Semiring semantics for the lambda calculus have been provided in [25]. In [8], a declarative programming framework was presented which unifies the analysis of different weighted model counting problems. Within software verification, semirings have been used in [7] for a definition of *weighted imperative programming*, a Hoare-like semantics, and a corresponding weakest (liberal) precondition semantics, and extended to Kleene algebras with tests in [28]. The weakest precondition semantics of [7] can also be expressed in our formalism, see Sect. A. For ARSs, so far only costs in specific semirings have been considered, e.g., in [3, 4, 24, 27]. Compared to all this related work, we present the first general semiring semantics for abstract reduction systems and demonstrate how to use semirings for analyzing different properties of programs in a unified way.

Structure: We give some preliminaries on abstract reduction systems and semirings in Sect. 2. Then, we introduce the new notion of *weighted ARSs* in Sect. 3 that defines semiring semantics for *sequence ARSs*. We illustrate the expressivity and applicability of this formalism in Sect. 4. In Sect. 5, we show how to prove boundedness. Here, we first give sufficient criteria that guarantee boundedness, and then we introduce the interpretation method for proving boundedness in general. In Sect. 6, we discuss the problem of finding a worst-case lower bound on the weight, or even a counterexample, i.e., we present a technique to find reduction sequences that lead to unbounded weight. In Sect. 7, we conclude and discuss ideas for future work. Finally, in Sect. A we show how to express the semantics of [7] in our setting, and we refer to [2] for all proofs.

2 Preliminaries

In this section, we give a brief introduction to abstract reduction systems and define complete lattice semirings that will ensure well-defined semiring semantics for ARSs in Sect. 3. The following definition of ARSs adheres to the commonly used definition, see, e.g., [5, 29].

▶ **Definition 3** (Abstract Reduction System, Normal Form, Determinism). *An* abstract reduction system (ARS) *is a pair* $(A, \to)$ *with a set* A *and a binary relation* $\to \subseteq A \times A$.

- *An object* $a \in A$ reduces *to* b *in a single step, abbreviated as* $a \to b$, *if* $(a, b) \in \to$.
- *An object* $a \in A$ *is called a* normal form *if there is no* $b \in A$ *such that* $a \to b$. *The set of all normal forms for* $(A, \to)$ *is denoted by* $\mathtt{NF}_\to$.
- *Finally, the ARS* $(A, \to)$ *is* deterministic *if for every object* $a \in A$ *there exists at most one* $b \in A$ *with* $a \to b$. *It is* finitely non-deterministic[2] *if for every object* $a \in A$ *there exist at most finitely many objects* $b \in A$ *with* $a \to b$.

An important property of ARSs is termination, i.e., the absence of infinite behavior.

▶ **Definition 4** (Reduction Sequence, Termination). *Let* $(A, \to)$ *be an ARS. A* reduction sequence *is a finite or infinite sequence* $a_1 \to a_2 \to \cdots$ *with* $a_i \in A$, *and we say that* $(A, \to)$ *is* terminating *if there exists no infinite reduction sequence.*

In our new notion of weighted rewriting, we *weigh* the normal forms in $\mathtt{NF}_\to$ by elements of a semiring, which consists of a set S associated with two operations $\oplus$ and $\odot$.

▶ **Definition 5** (Semiring). *A* semiring $\mathbb{S}$ *is a tuple* $(S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ *consisting of a set* S *(called the carrier) together with two binary functions* $\oplus, \odot : (S \times S) \to S$ *such that* $(S, \oplus, \mathbf{0})$ *is a commutative monoid (i.e.,* $\oplus$ *is commutative and associative with identity element* $\mathbf{0}$*),* $(S, \odot, \mathbf{1})$ *is a monoid, and* $\odot$ *distributes over* $\oplus$. *Furthermore,* $\mathbf{0}$ *is a multiplicative annihilator, i.e.,* $\mathbf{0} \odot s = s \odot \mathbf{0} = \mathbf{0}$ *for all* $s \in S$.

Sometimes we write $\oplus_\mathbb{S}$ or $\mathbf{0}_\mathbb{S}$ to clearly indicate the semiring $\mathbb{S}$. If it is clear from the context, we also use $\mathbb{S}$ to denote the carrier S. In Sect. 1, we already mentioned some examples of semirings, namely the confidence semiring $\mathbb{S}_{\mathsf{conf}}$ and the arctic semiring $\mathbb{S}_{\mathsf{arc}}$. Fig. 1 lists some relevant semirings for this work. Here, the multiplication in the *formal language semiring* $\mathbb{S}_\Sigma$ is pairwise concatenation, i.e., for $P_1, P_2 \subseteq \Sigma^*$, we have $P_1 \cdot P_2 = \{uv \mid u \in P_1, v \in P_2\}$.

[2] When only regarding classical ARSs, this notion is often called "finitely branching" instead. However, since we will regard *sequence* ARSs in Def. 9, in this paper the notion "finitely branching" will refer to the branching of their reduction trees, see Def. 10.

$$
\begin{array}{llll}
\mathbb{S}_{\mathbb{N}^\infty} & = (\mathbb{N}^\infty, +, \cdot, 0, 1) & \mathbb{S}_{\mathbb{R}^\infty} & = (\mathbb{R}^\infty_{\geq 0}, +, \cdot, 0, 1) \\
\mathbb{S}_{\mathsf{trop}} & = (\mathbb{N}^\infty, \min, +, \infty, 0) & \mathbb{S}_{\mathsf{arc}} & = (\mathbb{N}^{\pm\infty}, \max, +, -\infty, 0) \\
\mathbb{S}_{\mathbb{B}} & = (\{\mathsf{false}, \mathsf{true}\}, \vee, \wedge, \mathsf{false}, \mathsf{true}) & \mathbb{S}_{\mathsf{conf}} & = ([0,1], \max, \cdot, 0, 1) \\
\mathbb{S}_{\mathsf{bottle}} & = (\mathbb{R}^{\pm\infty}, \max, \min, -\infty, \infty) & \mathbb{S}_{\Sigma} & = (2^{\Sigma^*}, \cup, \cdot, \varnothing, \{\varepsilon\})
\end{array}
$$

Figure 1 Non-exhaustive list of complete lattice semirings $\mathbb{S} = (S, \oplus, \odot, \mathbf{0}, \mathbf{1})$.

Later, in Sect. 5 and 6, we establish upper and lower bounds on the weight of a given reduction sequence, respectively. Hence, we need an order on the elements in the semiring. Additionally, the order should be defined in a way that guarantees well-definedness of our semantics. We accomplish this by using the natural order[3] induced by the addition $\oplus$.

▶ **Definition 6** (Natural Order). *The* natural order $\preccurlyeq$ *on a semiring* $\mathbb{S}$ *is defined for all* $s, t \in \mathbb{S}$ *as* $s \preccurlyeq t$ *iff there exists an element* $u \in \mathbb{S}$ *with* $s \oplus u = t$. *A semiring is called* naturally ordered *if* $\preccurlyeq$ *is a partial order, i.e., reflexive, transitive, and antisymmetric.*[4]

The lack of "negative elements" in semirings ensures that we can define the natural order, because as soon as there exists an element different from $\mathbf{0}$ with an additive inverse, the relation $\preccurlyeq$ is not antisymmetric anymore. Every semiring in Fig. 1 is naturally ordered. If the additive operation is addition or maximum, then the order corresponds to the usual order on the extended naturals or extended reals. The natural order on $\mathbb{S}_\Sigma$ is the subset relation ($\preccurlyeq_{\mathbb{S}_\Sigma} = \subseteq$). Moreover, since the additive operation in the *tropical semiring* $\mathbb{S}_{\mathsf{trop}}$ is the minimum, the natural order in this semiring is the reverse of the usual order.

Our semantics in Sect. 3 consider demonic non-determinism. Hence, to analyze worst-case behavior, we want to take the least upper bound over all (possibly uncountably many) schedulers to resolve all non-determinism, i.e., we want to take the least upper bound of arbitrary (possibly uncountable) sets. A partially ordered set, where the least upper bound exists for every two elements, is called a *join-semilattice*, see e.g., [1]. Since we also need the existence of least upper bounds for infinite uncountable sets, we require *complete* lattices.[5]

▶ **Definition 7** (Complete Lattice). *A naturally ordered semiring* $\mathbb{S}$ *is a* complete lattice *if the* least upper bound *(or* supremum*)* $\bigsqcup T \in \mathbb{S}$ *exists for every set* $T \subseteq \mathbb{S}$.

All semirings in Fig. 1 are naturally ordered and complete lattices. A complete lattice semiring does not only have a minimum[6] $\mathbf{0} = \bot = \bigsqcup \varnothing \in \mathbb{S}$, but also a maximum $\top = \bigsqcup S \in \mathbb{S}$. Furthermore, the existence of every supremum allows us to define infinite sums and products of sequences, see, e.g., [13].

We define sequences $T = (x_i)_{i \in I} = [x_1, x_2, \ldots] \subseteq \mathbb{S}$, where either $I = \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$ for some $n \in \mathbb{N}$ (then T is a finite sequence of length n) or $I = \mathbb{N}_{\geq 1}$ (then T is an infinite sequence). By $\mathsf{Seq}(X)$, we denote the set of all non-empty sequences over some set X. Furthermore, the subset relation between two sequences $T = (x_i)_{i \in I}, T' = (x'_i)_{i \in I'} \in \mathsf{Seq}(X)$

[3] One can also generalize our results to partially ordered semirings, where the partial order is *compatible* with addition and multiplication, i.e., addition and multiplication are monotonic (see Def. 30). Note that the natural order is the least (w.r.t. $\subseteq$) such partial order.

[4] By definition, $\preccurlyeq$ is reflexive since $s \oplus \mathbf{0} = s$, and transitive since $s \oplus v = t$ and $t \oplus w = u$ imply $s \oplus (v \oplus w) = u$ by associativity of $\oplus$. So the only real requirement is antisymmetry.

[5] Lattices are semilattices, where in addition to suprema also infima are defined. While we do not need infima for our semantics, the existence of infima is guaranteed if one assumes the existence of suprema for all (possibly uncountable) subsets, see [2].

[6] Already in naturally ordered semirings, $\mathbf{0}$ is the minimum: for all $s \in \mathbb{S}$, we have $\mathbf{0} \preccurlyeq s$, since $\mathbf{0} \oplus s = s$.

is defined via prefixes, i.e., we write $T \subseteq T'$ if $I \subseteq I'$ and $x_i = x'_i$ for all $i \in I$. For a finite sequence $T = [s_1, \ldots, s_n] \in \mathsf{Seq}(\mathbb{S})$, we use the common abbreviation $\bigoplus T = \bigoplus_{i=1}^{n} s_i = s_1 \oplus \cdots \oplus s_n$ and $\bigodot T = \bigodot_{i=1}^{n} s_i = s_1 \odot \cdots \odot s_n$.

▶ **Definition 8** (Infinite Sums and Products). *Let $\mathbb{S}$ be a complete lattice semiring and T an infinite sequence*[7] *over $\mathbb{S}$. Then we define the infinite sum and product of T as*

$$\bigoplus T = \bigsqcup \left\{ \bigoplus T_{\mathit{fin}} \mid T_{\mathit{fin}} \text{ is a finite prefix of } T \right\}, \quad \bigodot T = \bigsqcup \left\{ \bigodot T_{\mathit{fin}} \mid T_{\mathit{fin}} \text{ is a finite prefix of } T \right\}.$$

Infinite sums and products are well defined, since the supremum of any set exists in the complete lattice $\mathbb{S}$, hence $\bigoplus T \in \mathbb{S}$ and $\bigodot T \in \mathbb{S}$ for all infinite sequences T over $\mathbb{S}$.

3 Semiring Semantics for Abstract Reduction Systems

In this section, we introduce *weighted abstract reduction systems* by defining semiring semantics for ARSs and show that these semantics are well defined for complete lattice semirings. Our definitions are in line with provenance analysis as introduced in Ex. 1. From now on, whenever we speak of a "semiring" we mean a complete lattice semiring.

As for ordinary ARSs, we represent the relation $\to$ via rules which are selected non-deterministically. However, each rule can have multiple outcomes, as in Ex. 2. Syntactically, for a *sequence abstract reduction system*, we use a relation $\to$ that relates a single object to a sequence of all corresponding outcomes that we later weigh using semiring elements. Note that sequences are necessary for reductions like $\psi \wedge \psi \to [\psi, \psi]$, where both incarnations of ψ may reduce to different objects due to possible non-determinism.

▶ **Definition 9** (Sequence Abstract Reduction System). *A* sequence abstract reduction system *(sARS) is a pair $(A, \to)$ consisting of a set A and a binary relation $\to \subseteq A \times \mathsf{Seq}(A)$.*

- *We write $a \to B = [b_1, b_2, \ldots]$ if $B \in \mathsf{Seq}(A)$ and $(a, B) \in \to$. Normal forms and $\mathtt{NF}_\to$ are defined as for ARSs.*
- *The sARS $(A, \to)$ is* deterministic *if for every $a \in A$ there is at most one B with $a \to B$ and* finitely non-deterministic *if for each $a \in A$, there are only finitely many $B \in \mathsf{Seq}(A)$ with $a \to B$. The sARS is* finitely branching *if for every $a \to B$, the sequence B is finite.*

We have already seen an example of an sARS in Ex. 2. In ordinary ARSs, it suffices to consider reduction sequences. When using sARSs, we obtain ordered reduction trees instead.

▶ **Definition 10** (Reduction Tree). *Let $(A, \to)$ be an sARS. An $(A, \to)$-*reduction tree *($(A, \to)$-RT) $\mathfrak{T} = (V, E)$ is a labeled, ordered tree with nodes V and directed edges $E \subseteq V \times V$, where*

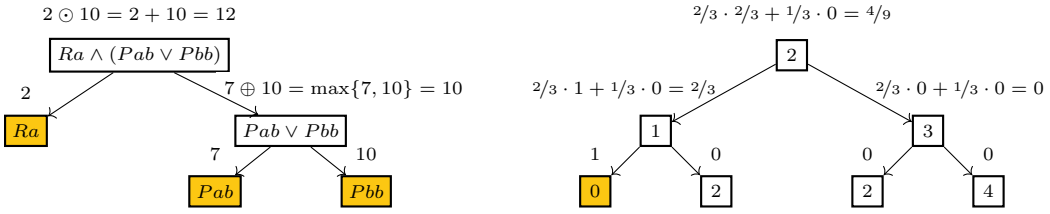
- *every node $v \in V$ is labeled by an object $a_v \in A$ and*
- *every node v together with its sequence of direct successors $vE = [w \in V \mid (v, w) \in E]$ either corresponds to a reduction step $a_v \to [a_w \mid w \in vE]$ or vE is empty.*

We say that $(A, \to)$ is terminating *if all $(A, \to)$-RTs have finite depth.*[8]

---

[7] Note that one typically defines sums for sets and not for sequences in provenance analysis. However, we use the order given by the sequence for our semantics in Sect. 3.

[8] One could instead attempt to define reduction trees in an inductive way. Then every node labeled with a value from A would be a reduction tree and one could lift the reduction $\to$ to a binary relation $\Rightarrow$ which extends reduction trees, i.e., $\mathfrak{T} \Rightarrow \mathfrak{T}'$ holds if there is a leaf v of $\mathfrak{T}$ and a reduction step $a_v \to B$ such that the reduction tree $\mathfrak{T}'$ extends $\mathfrak{T}$ by new leaves w_b with $a_{w_b} = b$ and edges (v, w_b) for all $b \in B$. However, in this way one would only obtain reduction trees of finite depth, whereas we also need reduction trees of infinite depth in order to represent non-terminating reductions, which would require an additional limit step in the construction.



(a) Reduction tree for formula ψ with semantics in $\mathbb{S}_{\mathsf{arc}}$.

(b) Reduction tree for a random walk with semantics in $\mathbb{S}_{\mathbb{R}^\infty}$.

Figure 2 Two example reduction trees, where each node v is labeled with $a_v \in A$ and the small numbers are the corresponding weights $[\![\mathfrak{T}]\!]^v$. Colored nodes are labeled by normal forms.

Fig. 2a depicts a reduction tree for the formula ψ from Ex. 1 and Fig. 2b shows a reduction tree for a biased random walk starting at 2, see Ex. 19. We define the weight of a reduction tree w.r.t. a semiring $\mathbb{S}$ by interpreting the leaf nodes as semiring elements and the inner nodes as combinations of its children. We use so-called *aggregator functions* to combine weights occurring in reductions based on the semiring addition and multiplication.

▶ **Definition 11** (Aggregator). *Let $\mathbb{S}$ be a semiring and $\mathcal{V} = \{v_1, v_2, \ldots\}$ be a set of variables. Then the set of all* aggregators *$\mathcal{F}$ (over $\mathbb{S}$ and $\mathcal{V}$) is the smallest set with*

- *$s \in \mathcal{F}$ for every $s \in \mathbb{S}$ (constants) and $v \in \mathcal{F}$ for every $v \in \mathcal{V}$ (variables),*
- *$\bigoplus F \in \mathcal{F}$ (sums) and $\bigodot F \in \mathcal{F}$ (products) for every $F \in \mathsf{Seq}(\mathcal{F})$.*

If an aggregator is constructed via finite sequences F, then it is a finite aggregator. *Let $\mathcal{V}(f)$ be the set of all variables in $f \in \mathcal{F}$ and let $\mathsf{max}\mathcal{V}(f) = \sup\{i \mid v_i \in \mathcal{V}(f)\} \in \mathbb{N}^\infty$.*

Let $f \in \mathcal{F}$ and $n \in \mathbb{N}^\infty$ with $n \geq \mathsf{max}\mathcal{V}(f)$. Then the aggregator f induces a function *$f : \mathbb{S}^n \to \mathbb{S}$ in the obvious way: For a sequence $T = [s_1, \ldots] \in \mathsf{Seq}(\mathbb{S})$ of length n, we have $s(T) = s$ for constants s, $v_i(T) = s_i$ for variables v_i and $1 \leq i \leq n$, and $(\bigcirc F)(T) = \bigcirc[f_1(T), f_2(T), \ldots]$ for $F = [f_1, f_2, \ldots]$, where $\bigcirc \in \{\bigoplus, \bigodot\}$.*

Weighted rewriting considers an sARS $(A, \to)$ together with a semiring $\mathbb{S}$, and functions $\mathsf{f}_{\mathtt{NF}}$ and $\mathsf{Aggr}_{a \to B}$ in order to map objects from A to elements of $\mathbb{S}$.

▶ **Definition 12** (Weighted Abstract Reduction System). *A tuple $(A, \to, \mathbb{S}, \mathsf{f}_{\mathtt{NF}}, \mathsf{Aggr}_{a \to B})$ is a* weighted abstract reduction system (wARS) *if*

- *$(A, \to)$ is an sARS,*
- *$\mathbb{S}$ is a semiring,*
- *$\mathsf{f}_{\mathtt{NF}} : \mathtt{NF}_\to \to \mathbb{S}$ is the* interpretation of normal forms,
- *$\mathsf{Aggr}_{a \to B} \in \mathcal{F}$ is the* aggregator *for every $a \to B$,*

where $\mathsf{max}\mathcal{V}(\mathsf{Aggr}_{a \to B}) \leq |B|$.

For every $\mathsf{Aggr}_{a \to B}$, we consider the induced function $\mathsf{Aggr}_{a \to B} : \mathbb{S}^n \to \mathbb{S}$ of arity $n = |B|$. Now we introduce our semiring semantics for reduction trees of finite depth by using $\mathsf{f}_{\mathtt{NF}}$ to interpret the leaves of a reduction tree that are labeled by normal forms. To interpret inner nodes, we use aggregator functions that distinguish between the different reductions. So for the example from Ex. 2 and Fig. 2a, we use a function $\mathsf{f}_{\mathtt{NF}}$ where $\mathsf{f}_{\mathtt{NF}}(\alpha)$ is the cost of atom α, and we use $\mathsf{Aggr}_{\varphi \wedge \psi \to [\varphi, \psi]} = v_1 \odot v_2$ for every rule $\varphi \wedge \psi \to [\varphi, \psi]$ and $\mathsf{Aggr}_{\varphi \vee \psi \to [\varphi, \psi]} = v_1 \oplus v_2$ for every rule $\varphi \vee \psi \to [\varphi, \psi]$. Thus, combining the weights of the children via aggregator functions enables us to calculate a weight for the root of any reduction tree with possibly infinite (countable) branching and finite depth.[9]

[9] Alternatively, one could also consider finite-depth reduction trees as first-order ground terms (with

▶ **Definition 13** (Semiring Semantics). *For a wARS* $(A, \to, \mathbb{S}, \mathsf{f}_{\mathtt{NF}}, \mathsf{Aggr}_{a\to B})$ *and an* $(A, \to)$*-RT* $\mathfrak{T} = (V, E)$ *of finite depth, we define the* weight $[\![\mathfrak{T}]\!]^v$ *of* $\mathfrak{T}$ *at node* $v \in V$ *as*

$$
\begin{aligned}
&[\![\mathfrak{T}]\!]^v = \mathsf{f}_{\mathtt{NF}}(a_v) && \text{if } a_v \in \mathtt{NF}_\to \\
&[\![\mathfrak{T}]\!]^v = \mathbf{0} && \text{if } v \text{ is a leaf and } a_v \notin \mathtt{NF}_\to \\
&[\![\mathfrak{T}]\!]^v = \mathsf{Aggr}_{a_v\to B}\,[[\![\mathfrak{T}]\!]^w \mid w \in vE] && \text{if } v \text{ is an inner node and } B = [a_w \mid w \in vE].
\end{aligned}
$$

The weight of the whole RT $\mathfrak{T}$ *is* $[\![\mathfrak{T}]\!] = [\![\mathfrak{T}]\!]^r$*, where* $r \in V$ *is the root node of* $\mathfrak{T}$*.*

Note that the order of the children $B = [a_w \mid w \in vE]$ is crucial when applying $\mathsf{Aggr}_{a_v\to B}$ to $[[\![\mathfrak{T}]\!]^w \mid w \in vE]$. This is needed, e.g., when an aggregator is used to represent different probabilities for the successor objects in a biased random walk, see, e.g., Fig. 2b and Sect. 4.3.

A reduction tree of finite depth represents one possible execution of the sARS up to a certain number of steps, where the non-determinism is resolved by some fixed scheduler. To define the semantics of an object $a \in A$, we consider any finite number of reduction steps and all possible schedulers. Then we define the weight of a as the least upper bound of the weights of all finite-depth reduction trees whose root is labeled with a.

▶ **Definition 14** (Semantics with Demonic Non-Determinism). *For a wARS* $\mathcal{W} = (A, \to, \mathbb{S}, \mathsf{f}_{\mathtt{NF}}, \mathsf{Aggr}_{a\to B})$ *and* $a \in A$*, let* $\Phi(a)$ *be the set of all* $(A, \to)$*-reduction trees of finite depth whose root node is labeled with* a*. Then we define the* weight *of* a *as* $[\![a]\!] = \bigsqcup\{[\![\mathfrak{T}]\!] \mid \mathfrak{T} \in \Phi(a)\}$.[10]

Due to possibly uncountably many schedulers, there might be uncountably many reduction trees each with a different weight, see [2]. Nevertheless, since $\mathbb{S}$ is a complete lattice, the supremum of every set exists and thus, the weight of every object is well defined.

▶ **Corollary 15** (Well-Defined Semantics). *For any wARS* $(A, \to, \mathbb{S}, \mathsf{f}_{\mathtt{NF}}, \mathsf{Aggr}_{a\to B})$*, the weight* $[\![a]\!]$ *is well defined for every object* $a \in A$*.*

The set $\Phi(a)$ takes on different shapes depending on the reduction system. If the reduction system is deterministic, then $\Phi(a)$ consists of all finite-depth prefixes of a single (potentially infinite-depth) tree. If the sARS is non-deterministic, then $\Phi(a)$ may contain uncountably many trees. The maximal size of the sequences in the reduction rules determines the maximal branching degree of the trees. If the sARS is finitely branching, so are the trees in $\Phi(a)$.

The computation of the weight $[\![a]\!]$ is undecidable in general, since computing single steps with $\to$ may already be undecidable. However, even if the reductions $a \to B$, the interpretation of the normal forms $\mathsf{f}_{\mathtt{NF}}$, and the aggregator functions $\mathsf{Aggr}_{a\to B}$ are computable, computing the weight $[\![a]\!]$ can still be undecidable, because it can express notions like termination of deterministic systems as demonstrated in the next section (Sect. 4.1).

4 Expressivity of Semiring Semantics

In this section we give several examples to demonstrate the versatility and expressive power of our new formalism, and show that existing approaches for the analysis of reduction systems actually consider specific semirings.

function symbols of possibly infinite arity). Then the semiring semantics of Def. 13 would correspond to a polynomial interpretation where the polynomials $\mathsf{Aggr}_{a_v\to B}$ are constructed using the operations $\oplus$ and $\odot$ of the semiring.

[10] In principle, $[\![a]\!]$, $[\![\mathfrak{T}]\!]$, and $[\![\mathfrak{T}]\!]^v$ are indexed by $\mathcal{W}$, but we omitted this index for readability.

4.1 Termination and Complexity

We can extend any ARS $(A, \to)$ to a wARS $\mathsf{cplx}(A, \to) = (A, \xrightarrow{s}, \mathbb{S}_{\mathbb{N}\infty}, \mathsf{f}_{\mathtt{NF}}^{\mathsf{cplx}}, \mathsf{Aggr}_{a \xrightarrow{s} B}^{\mathsf{cplx}})$ such that $[\![a]\!]$ is equal to the supremum over the lengths of all reduction sequences starting in $a \in A$. For this, we use the sequence relation $\xrightarrow{s} = \{a \xrightarrow{s} [b] \mid a \to b\}$, the *extended naturals semiring* $\mathbb{S}_{\mathbb{N}\infty}$, the interpretation $\mathsf{f}_{\mathtt{NF}}^{\mathsf{cplx}}(a) = \mathbf{0}_{\mathbb{S}_{\mathbb{N}\infty}} = 0$ for all $a \in \mathtt{NF}_{\xrightarrow{s}}$, and the aggregator $\mathsf{Aggr}_{a \xrightarrow{s} [b]}^{\mathsf{cplx}} = 1 \oplus_{\mathbb{S}_{\mathbb{N}\infty}} v_1 = 1 + v_1$ whenever $a \xrightarrow{s} [b]$. Recall that aggregators use a fixed set of variables $\mathcal{V} = \{v_1, \ldots\}$. The derivational complexity of $(A, \to)$ (i.e., the supremum of the lengths of possible reduction sequences) is obtained by analyzing the weights $[\![a]\!]$ of $\mathsf{cplx}(A, \to)$.

Techniques for automatic complexity and termination analysis have been developed for, e.g., term rewrite systems (TRSs) in the literature [5, 29], and there is an annual *Termination and Complexity Competition* with numerous participating tools [18]. In term rewriting, one considers terms $t \in \mathcal{T}(\Sigma, \mathcal{V})$ over a set of function symbols Σ and a set of variables $\mathcal{V}$. The reduction relation $\to_{\mathcal{R}}$ is defined via a set of rewrite rules $\mathcal{R}$: If the left-hand side of a rewrite rule in $\mathcal{R}$ matches a subterm, we can replace this subterm with the right-hand side of the rewrite rule instantiated by the matching substitution. For details, see, e.g., [5, 29].

▶ **Example 16** (TRS for Addition)**.** Let $\Sigma = \{\mathsf{plus}\}$ and $\mathcal{V} = \{x, y\}$. A TRS $\mathcal{R}$ computing the addition of two natural numbers (given in Peano notation via zero $\mathcal{O}$ and the successor function s) is defined by the two rewrite rules $\mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{plus}(x, y))$ and $\mathsf{plus}(\mathcal{O}, y) \to y$. $\mathcal{R}$ allows for the reduction $\mathsf{plus}(\mathsf{s}(\mathcal{O}), \mathsf{s}(\mathcal{O})) \to_{\mathcal{R}} \mathsf{s}(\mathsf{plus}(\mathcal{O}, \mathsf{s}(\mathcal{O}))) \to_{\mathcal{R}} \mathsf{s}(\mathsf{s}(\mathcal{O}))$. For derivational complexity analysis, we can consider the wARS $(\mathcal{T}(\Sigma, \mathcal{V}), \xrightarrow{s}_{\mathcal{R}}, \mathbb{S}_{\mathbb{N}\infty}, \mathsf{f}_{\mathtt{NF}}^{\mathsf{cplx}}, \mathsf{Aggr}_{a \xrightarrow{s} B}^{\mathsf{cplx}})$.

If the ARS $(A, \to)$ is finitely non-deterministic, then $(A, \to)$ is terminating if and only if $[\![a]\!]_{\mathsf{cplx}(A, \to)} < \infty$ for every $a \in A$. While the "if" direction holds for any ARS, due to possibly infinite non-determinism, the "only if" direction does not hold in general.

▶ **Example 17** (Non-Deterministic ARS)**.** Consider the ARS $(\mathbb{N}_a, \to)$ with $\mathbb{N}_a = \{a\} \cup \mathbb{N}$ and $\to = \{a \to n \mid n \in \mathbb{N}\} \cup \{n+1 \to n \mid n \in \mathbb{N}\}$ from [3]. For $\mathsf{cplx}(\mathbb{N}_a, \to)$, we have $[\![a]\!] = \infty$ as for all $n \in \mathbb{N}$ there is an $(\mathbb{N}_a, \to)$-RT of depth $n+1$ with root a. However, $(\mathbb{N}_a, \to)$ is terminating.

The definition of $\mathsf{cplx}(A, \to)$ can also be adjusted to prove termination and analyze derivational complexity of *sequence* ARSs.

4.2 Size Bounds

In addition to the runtime of a program, its memory footprint is of interest as well. Consider an operating system which should be able to run forever. However, during this infinite execution, certain values that are stored in memory must not become arbitrarily large, i.e., no overflow should occur. To analyze this, we can use the arctic semiring $\mathbb{S}_{\mathsf{arc}}$.

▶ **Example 18** (Memory Consumption of Operating System)**.** Consider a very simplified operating system[11] with two processes P_1 and P_2 that should be performed repeatedly. The operating system can either be idle, run a process, or add a process at the end of the waiting queue. We represent this by the ARS $(\mathsf{OS}, \to)$ with $\mathsf{OS} = \{\mathsf{idle}(p), \mathsf{wait}(p), \mathsf{run}(p) \mid p \in \{P_1, P_2\}^*\}$. So an object from OS represents the current state of the operating system (idle, wait, or run) and the current waiting queue p. The rules of the ARS are $\mathsf{idle}(p) \to \mathsf{wait}(p)$, $\mathsf{idle}(p) \to \mathsf{run}(p)$ (add a new process to the waiting queue or run some process), $\mathsf{wait}(p) \to \mathsf{idle}(pP_1), \mathsf{wait}(p) \to \mathsf{idle}(pP_2)$ (add P_1 or P_2 to the waiting queue), and $\mathsf{run}(P_1 p) \to$

[11] See Sect. A.2 for a more involved operating system algorithm that guarantees mutual exclusion.

$\mathsf{idle}(p), \mathsf{run}(P_2p) \to \mathsf{idle}(p)$ (run the process waiting the longest) for all $p \in \{P_1, P_2\}^*$. We use the wARS $(\mathsf{OS}, \to, \mathbb{S}_{\mathsf{arc}}, \mathsf{f}^{\mathsf{size}}_{\mathtt{NF}}, \mathsf{Aggr}^{\mathsf{size}}_{a\to B})$ with $\mathsf{f}^{\mathsf{size}}_{\mathtt{NF}}(\mathsf{run}(\varepsilon)) = 0$ for $\mathtt{NF}_\to = \{\mathsf{run}(\varepsilon)\}$ and

$$\begin{array}{rcrcrcl}
\mathsf{Aggr}^{\mathsf{size}}_{\mathsf{idle}(p)\to\mathsf{wait}(p)} & = & \mathsf{Aggr}^{\mathsf{size}}_{\mathsf{idle}(p)\to\mathsf{run}(p)} & = & v_1 & & \\
\mathsf{Aggr}^{\mathsf{size}}_{\mathsf{run}(P_1p)\to\mathsf{idle}(p)} & = & \mathsf{Aggr}^{\mathsf{size}}_{\mathsf{run}(P_2p)\to\mathsf{idle}(p)} & = & v_1 & & \\
\mathsf{Aggr}^{\mathsf{size}}_{\mathsf{wait}(p)\to\mathsf{idle}(pP_1)} & = & \mathsf{Aggr}^{\mathsf{size}}_{\mathsf{wait}(p)\to\mathsf{idle}(pP_2)} & = & v_1 \oplus_{\mathbb{S}_{\mathsf{arc}}} (|p|+1) & = & \max\{v_1, |p|+1\}.
\end{array}$$

Note that we may have a different aggregator for every sequence $p \in \{P_1, P_2\}^*$, i.e., $|p|+1$ is a constant. We obtain $[\![\mathsf{idle}(\varepsilon)]\!] = \infty$, proving that a reduction leading to a waiting queue of unbounded size exists.

4.3 Probabilistic Rewriting

In [3, 11, 12], ARSs were extended to the probabilistic setting. The relation $\hookrightarrow$ of a probabilistic ARS has (countable) multi-distributions on the right-hand sides. A *multi-distribution* μ on a set $A \neq \varnothing$ is a countable multiset of pairs $(p : a)$, where $p \in \mathbb{R}$ with $0 < p \leq 1$ is a probability and $a \in A$, with $\sum_{(p:a)\in\mu} p = 1$. $\mathrm{Dist}(A)$ is the set of all multi-distributions on A and $(A, \hookrightarrow)$ with $\hookrightarrow \subseteq A \times \mathrm{Dist}(A)$ is a *probabilistic abstract reduction system* (pARS). Depending on the property of interest, we can, e.g., use the semiring $\mathbb{S}_{\mathbb{R}^\infty}$ to describe the termination probability or even the expected derivational complexity of the pARS.

▶ **Example 19** (Random Walk)**.** Consider the biased random walk on $\mathbb{N}$ given by the probabilistic relation $\hookrightarrow = \{n+1 \hookrightarrow \{2/3 : n, 1/3 : n+2\} \mid n \in \mathbb{N}\}$. We use the sARS $(\mathbb{N}, \to)$ with $\to = \{n+1 \to [n, n+2] \mid n \in \mathbb{N}\}$, the semiring $\mathbb{S}_{\mathbb{R}^\infty}$, the interpretation of the normal form $\mathsf{f}_{\mathtt{NF}}(0) = 1$, and the aggregator $\mathsf{Aggr}_{n+1\to[n,n+2]} = (2/3 \odot_{\mathbb{S}_{\mathbb{R}^\infty}} v_1) \oplus_{\mathbb{S}_{\mathbb{R}^\infty}} (1/3 \odot_{\mathbb{S}_{\mathbb{R}^\infty}} v_2) = 2/3 \cdot v_1 + 1/3 \cdot v_2$ for every $n \in \mathbb{N}$. The weight of the tree $\mathfrak{T}$ from Fig. 2b is $[\![\mathfrak{T}]\!] = 4/9$, since $4/9$ is the probability to reach 0 within two steps. The weight of the infinite extension $\mathfrak{T}_\infty$ of the depicted tree $\mathfrak{T}$ is $[\![\mathfrak{T}_\infty]\!] = 1$, as such a random walk terminates with probability 1. In this way one can use semiring semantics to express *almost-sure termination* (AST) of pARSs [3].

Obviously, we can also consider infinite-support distributions, e.g., consider the probabilistic relation $\hookrightarrow = \{n+1 \hookrightarrow \{\mathrm{Geo}(m) : m \mid m \in \mathbb{N}\} \mid n \in \mathbb{N}\}$, where Geo denotes the *geometric distribution*, i.e., $\mathrm{Geo}(m) = (1/2)^{m+1}$ for all $m \in \mathbb{N}$. Here, we use the sequence ARS $(\mathbb{N}, \to)$ with $\to = \{n+1 \to [0, 1, 2, \ldots] \mid n \in \mathbb{N}\}$, and the aggregator $\mathsf{Aggr}_{n+1\to[0,1,2,\ldots]} = \bigoplus_{m=0}^{\infty} (\mathrm{Geo}(m) \odot_{\mathbb{S}_{\mathbb{R}^\infty}} v_{m+1})$ for every $n \in \mathbb{N}$ ($\mathsf{f}_{\mathtt{NF}}$ and $\mathbb{S}_{\mathbb{R}^\infty}$ remain as above).

Moreover, we can also use different aggregators and interpretations of normal forms to analyze the probability of reaching a certain normal form, or the expected complexity (i.e., the expected number of reduction steps). For the expected derivational complexity of the biased random walk, we again use the semiring $\mathbb{S}_{\mathbb{R}^\infty}$ but switch to the interpretation of the normal form $\mathsf{f}_{\mathtt{NF}}(0) = 0$ and the aggregator $\mathsf{Aggr}_{n+1\to[n,n+2]} = 1 + 2/3 \cdot v_1 + 1/3 \cdot v_2$, i.e., we add 1 in each step and start with 0. Then we obtain $[\![n]\!] \neq \infty$ for every $n \in \mathbb{N}$, i.e., the expected derivational complexity is finite for each possible start of the random walk, which proves *positive* and *strong almost-sure termination* (PAST and SAST) [3, 12].

4.4 Formal Languages

We can use semirings like $\mathbb{S}_\Sigma$ to analyze the behavior of systems. Reconsider the setting from Ex. 18. Instead of analyzing the memory consumption of the waiting queue, we can also analyze the possible orders of running processes.

▶ **Example 20** (Process Order for Operating System)**.** Reconsider the sARS for the operating system from Ex. 18. We can use the wARS $(\mathsf{OS}, \to, \mathbb{S}_\Sigma, \mathsf{f}^{\mathsf{fair}}_{\mathtt{NF}}, \mathsf{Aggr}^{\mathsf{fair}}_{a\to B})$ with $\Sigma = \{P_1, P_2\}$,

$\mathsf{f}^{\mathsf{fair}}_{\mathtt{NF}}(\mathsf{run}(\varepsilon)) = \mathbf{1}_{\mathbb{S}_\Sigma} = \{\varepsilon\}$, and

$$\begin{array}{rcrcl} \mathsf{Aggr}^{\mathsf{fair}}_{\mathsf{idle}(p)\to\mathsf{wait}(p)} & = & \mathsf{Aggr}^{\mathsf{fair}}_{\mathsf{idle}(p)\to\mathsf{run}(p)} & = & v_1 \\ \mathsf{Aggr}^{\mathsf{fair}}_{\mathsf{wait}(p)\to\mathsf{idle}(pP_1)} & = & \mathsf{Aggr}^{\mathsf{fair}}_{\mathsf{wait}(p)\to\mathsf{idle}(pP_2)} & = & v_1 \\ & & \mathsf{Aggr}^{\mathsf{fair}}_{\mathsf{run}(P_1p)\to\mathsf{idle}(p)} & = & \{P_1\} \odot_{\mathbb{S}_\Sigma} v_1 \\ & & \mathsf{Aggr}^{\mathsf{fair}}_{\mathsf{run}(P_2p)\to\mathsf{idle}(p)} & = & \{P_2\} \odot_{\mathbb{S}_\Sigma} v_1 \end{array}$$

Our operating system allows running the processes in any order, since $[\![\mathsf{idle}(\varepsilon)]\!] = \Sigma^*$.

4.5 Combinations of Semirings

Cartesian products (and even matrices) of semirings form a semiring again by performing addition and multiplication pointwise. Moreover, if all the semirings are complete lattices, then so is the resulting Cartesian product semiring.

▶ **Lemma 21** (Cartesian Product Semiring). *Let $(\mathbb{S}_i)_{1\le i\le n}$ be a family of complete lattice semirings. Then $\mathbb{S} = \bigtimes_{i=1}^{n} \mathbb{S}_i$ is a complete lattice semiring with $(x_1,\dots,x_n)\oplus_{\mathbb{S}}(y_1,\dots,y_n) = (x_1 \oplus_{\mathbb{S}_1} y_1,\dots,x_n \oplus_{\mathbb{S}_n} y_n)$, and $(x_1,\dots,x_n) \odot_{\mathbb{S}} (y_1,\dots,y_n) = (x_1 \odot_{\mathbb{S}_1} y_1,\dots,x_n \odot_{\mathbb{S}_n} y_n)$.*

▶ **Example 22** (Analyzing Complexity and Safety Simultaneously). Consider the ARS $(\mathbb{Z},\to)$ with $\to = \{n \to n-2 \mid n \text{ odd}\} \cup \{n \to n+2 \mid n \le -2, n \text{ even}\} \cup \{n \to n-2 \mid n \ge 2, n \text{ even}\}$. Additionally, consider a certain "unsafe" property, e.g., hitting an even number. To analyze whether all infinite sequences are safe, we take the product semiring $\mathbb{S} = \mathbb{S}_{\mathbb{N}^\infty} \times \mathbb{S}_{\mathbb{B}}$ over $\mathbb{S}_{\mathbb{N}^\infty}$ and the *Boolean semiring* $\mathbb{S}_{\mathbb{B}}$. We use the normal form interpretation $\mathsf{f}_{\mathtt{NF}}(0) = (0,\mathsf{true})$ and the aggregator $\mathsf{Aggr}_{n\xrightarrow{s}[m]} = (1,[n \mod 2 = 0]) \oplus_{\mathbb{S}} v_1$ for every $n \to m$. The first component describes the derivational complexity, while the second describes whether we reached an even number at some point during the reduction. In Sect. 5, we will see how to prove boundedness (i.e., $[\![n]\!] \neq (\infty,\mathsf{true})$ for every $n \in \mathbb{Z}$) indicating safety of every infinite reduction.

Taking tuples for verification is not the same as performing two separate analyses. Analyzing safety and complexity on their own for the ARS from Ex. 22 would fail, since the ARS is neither safe nor has finite complexity for every $n \in \mathbb{Z}$. Note the change of quantifiers: Instead of "*all runs are safe, or all runs are finite*", we prove "*all runs are finite or safe*".

4.6 Limitations

The following example illustrates a limit of our approach.

▶ **Example 23** (Starvation Freedom). To analyze *starvation freedom*, i.e., whether every process will eventually be served, one can use the tuple semiring $\mathbb{S}_{\mathbb{N}^\infty} \times \mathbb{S}_{\mathbb{N}^\infty}$ for our operating system from Ex. 18 (a corresponding more complex example for starvation freedom is presented in Sect. A.2). Now the two entries of the tuples count how often a process was already served. Thus, we can use the wARS $(\mathsf{OS},\to,\mathbb{S}_{\mathbb{N}^\infty} \times \mathbb{S}_{\mathbb{N}^\infty}, \mathsf{f}^{\mathsf{starv}}_{\mathtt{NF}}, \mathsf{Aggr}^{\mathsf{starv}}_{a\to B})$ with $\mathsf{f}^{\mathsf{starv}}_{\mathtt{NF}}(\mathsf{run}(\varepsilon)) = \mathbf{0}_{(\mathbb{S}_{\mathbb{N}^\infty}\times\mathbb{S}_{\mathbb{N}^\infty})} = (0,0)$ and the aggregator

$$\begin{array}{rcrcl} \mathsf{Aggr}^{\mathsf{starv}}_{\mathsf{idle}(p)\to\mathsf{wait}(p)} & = & \mathsf{Aggr}^{\mathsf{starv}}_{\mathsf{idle}(p)\to\mathsf{run}(p)} & = & v_1 \\ \mathsf{Aggr}^{\mathsf{starv}}_{\mathsf{wait}(p)\to\mathsf{idle}(pP_1)} & = & \mathsf{Aggr}^{\mathsf{starv}}_{\mathsf{wait}(p)\to\mathsf{idle}(pP_2)} & = & v_1 \\ & & \mathsf{Aggr}^{\mathsf{starv}}_{\mathsf{run}(P_1p)\to\mathsf{idle}(p)} & = & (1,0) \oplus v_1 \\ & & \mathsf{Aggr}^{\mathsf{starv}}_{\mathsf{run}(P_2p)\to\mathsf{idle}(p)} & = & (0,1) \oplus v_1 \end{array}$$

However, starvation freedom cannot be analyzed via our current definition of $[\![a]\!]$ in Def. 14. We have $[\![a]\!] = (\infty,\infty)$ for all $a \in \mathsf{OS} \setminus \{\mathsf{run}(\varepsilon)\}$ (i.e., for all non-normal forms). This means

that for every such start configuration a, there *exists* a ("worst-case") reduction of weight (∞, ∞) where both processes are served infinitely often. However, for starvation freedom, one would have to show that *every* infinite reduction serves both processes infinitely often (i.e., this would need to hold irrespective of how the non-determinism in the reductions is resolved). However, $(\mathsf{OS}, \to)$ is not starvation free, since, e.g., we may only serve P_1 infinitely often.

So a property like starvation freedom cannot be expressed with our current definition of $[\![a]\!]$, because due to the use of the least upper bound in Def. 14, here we only focus on worst-case reductions. An extension of our approach to also analyze (bounds on) best-case reductions in order to prove properties like starvation freedom is an interesting direction for future work.

5 Proving Upper Bounds on Weights

In this section, we present a technique amenable to automation which aims to prove an upper bound on all possible weights $[\![a]\!]$ of objects $a \in A$, i.e., it shows that $[\![a]\!] \neq \top$ for all $a \in A$. For the remaining sections, we fix a wARS $(A, \to, \mathbb{S}, \mathsf{f}_{\mathtt{NF}}, \mathsf{Aggr}_{a \to B})$.

▶ **Definition 24** (Boundedness). *A wARS is* bounded *if* $[\![a]\!] \neq \top$ *for all* $a \in A$.

The examples in Sect. 4 illustrate that boundedness is a crucial property for wARSs, and that depending on the semiring, on the interpretation of the normal forms, and on the aggregators, boundedness may have completely different implications.

We first establish sufficient conditions for boundedness of a wARS in Sect. 5.1. Afterwards, we show in Sect. 5.2 that the well-known interpretation method can be generalized to prove boundedness for wARSs where these conditions are not satisfied.

5.1 Guaranteed Boundedness

One can directly guarantee boundedness by an adequate choice of the semiring, the interpretation of the normal forms, and the aggregators. We say that $\mathsf{f}_{\mathtt{NF}} : \mathtt{NF}_{\to} \to \mathbb{S}$ is *universally bounded* if there exists a universal bound $C \in \mathbb{S} \setminus \{\top\}$ with $\mathsf{f}_{\mathtt{NF}}(a) \preccurlyeq C$ for all $a \in \mathtt{NF}_{\to}$. An aggregator function $\mathsf{Aggr}_{a \to B} : \mathbb{S}^{|B|} \to \mathbb{S}$ is *selective* if for every $[s_1, s_2, \ldots] \in \mathbb{S}^{|B|}$ there exists an $1 \leq i \leq |B|$ such that $\mathsf{Aggr}_{a \to B}[s_1, s_2, \ldots] = s_i$. For example, in the *bottleneck semiring* $\mathbb{S}_{\mathsf{bottle}} = (\mathbb{R}^{\pm\infty}, \max, \min, -\infty, \infty)$, finite aggregator functions without constants are always selective, since max and min are selective functions.

▶ **Theorem 25** (Sufficient Condition for Boundedness (1)). *A wARS is*

- *not bounded if* $\mathsf{f}_{\mathtt{NF}}(a) = \top$ *for some* $a \in \mathtt{NF}_{\to}$.
- *bounded if* $\mathsf{f}_{\mathtt{NF}}$ *is universally bounded and all* $\mathsf{Aggr}_{a \to B}$ *are selective.*

Next, we do not only consider properties of $\mathsf{f}_{\mathtt{NF}}$ and $\mathsf{Aggr}_{a \to B}$, but also properties of the sARS in order to guarantee boundedness.

▶ **Example 26** (Boundedness for Provenance Analysis Example). Reconsider the setting of Ex. 1 and the sARS of Ex. 2. Note that all propositional formulas are finite, hence the sARS is finitely branching and terminating. If none of the atomic facts has infinite cost, then no formula has infinite cost, since finite sums and products in the arctic semiring $\mathbb{S}_{\mathsf{arc}} = (\mathbb{N}^{\pm\infty}, \max, +, -\infty, 0)$ cannot result in ∞ if all of its arguments are smaller than ∞.

The latter property of the semiring is called the *extremal property* (or *convex hull concept*).

▶ **Definition 27** (Extremal Property). *A function $f : \mathbb{S}^n \to \mathbb{S}$ over a semiring $\mathbb{S}$ with $n \in \mathbb{N}$ has the* extremal property *if $f(e_1, \dots, e_n) \neq \top$ for all $e_1, \dots, e_n \in \mathbb{S} \setminus \{\top\}$. A semiring $\mathbb{S} = (S, \oplus, \odot, \mathbf{0}, \mathbf{1})$ has the extremal property if $\oplus$ and $\odot$ have the extremal property.*

If addition and multiplication of a semiring $\mathbb{S}$ satisfy the extremal property, then sums and products of finite sequences $T \subseteq \mathbb{S} \setminus \{\top\}$ do not evaluate to $\top$, i.e., $\bigoplus T \neq \top$ and $\bigodot T \neq \top$. Thus, every finite aggregator function that does not use the constant $\top$ never evaluates to $\top$. However, this does not necessarily hold for infinite sums, products, and aggregators. Consider, e.g., the subset $\mathbb{N} \subset \mathbb{N}^\infty$ of the extended natural numbers, where $\bigoplus_{\mathbb{N}^\infty} \mathbb{N} = \sum \mathbb{N} = \infty = \top_{\mathbb{N}^\infty}$. Selective functions always satisfy the extremal property.

▶ **Example 28** (Extremal Property). Ex. 26 shows that the arctic semiring $\mathbb{S}_{\mathsf{arc}}$ has the extremal property. The extended naturals semiring $\mathbb{S}_{\mathbb{N}^\infty}$ also has the extremal property, since $a+b \neq \infty$ and $a \cdot b \neq \infty$ for all $a, b \in \mathbb{N}$. Actually, the extremal property holds for all semirings in Fig. 1 except for the formal languages semiring $\mathbb{S}_\Sigma$, where, e.g., $\big(\Sigma^* \setminus \{\varepsilon\}\big) \cup \{\varepsilon\} = \Sigma^* = \top_{\mathbb{S}_\Sigma}$. Cartesian products of semirings with the extremal property do not necessarily satisfy the extremal property again: Consider $\mathbb{S}_{\mathsf{arc}} \times \mathbb{S}_{\mathsf{arc}}$, where the addition of two objects that are different from $\top_{\mathbb{S}_{\mathsf{arc}} \times \mathbb{S}_{\mathsf{arc}}}$ yields $(\mathbf{0}, \top) \oplus (\top, \mathbf{0}) = (\top, \top) = \top$.

This yields another sufficient condition for boundedness (see Ex. 26).

▶ **Theorem 29** (Sufficient Condition for Boundedness (2)). *A wARS is bounded if*

- *the sARS $(A, \to)$ is terminating, finitely non-deterministic, and finitely branching,*
- *the semiring $\mathbb{S}$ has the extremal property,*
- *$\mathsf{f}_{\mathtt{NF}}(a) \neq \top$ for all $a \in \mathtt{NF}_\to$, and*
- *all aggregators $\mathsf{Aggr}_{a \to B}$ are finite and do not use $\top$ as a constant.*

While the requirements in Thm. 29 may seem restrictive, the ones on $\mathsf{f}_{\mathtt{NF}}$ and $\mathsf{Aggr}_{a \to B}$ only consider certain edge cases. However, termination of the underlying sARS is a crucial requirement for Thm. 29 that one has to prove beforehand.

Ex. 17 presents an unbounded wARS which satisfies many of the constraints from Thm. 29, but illustrates the importance of *finite* non-determinism. Infinite non-determinism allows the existence of a chain of reduction trees with ascending weights which reaches $\top$ in the limit.

5.2 Proving Boundedness via Interpretations

To handle wARSs that neither satisfy the requirements of Thm. 25 nor of Thm. 29, we now extend the well-known interpretation method (see, e.g., [26]) to prove boundedness of general wARSs. In some cases, e.g., when considering term rewriting as in Sect. 4.1, this often allows proving termination automatically. Before presenting the technique to prove boundedness via interpretations, we introduce the notions of monotonicity and continuity.

▶ **Definition 30** (Monotonicity, Continuity). *A function $f : \mathbb{S} \to \mathbb{S}$ on a semiring $\mathbb{S}$ is* monotonic *if for all $s, t \in \mathbb{S}$, $s \preccurlyeq t$ implies $f(s) \preccurlyeq f(t)$. It is* continuous *if for all $T \subseteq \mathbb{S}$, $\bigsqcup f(T) = \bigsqcup \{f(t) \mid t \in T\} = f(\bigsqcup T)$. A function $f : \mathbb{S}^n \to \mathbb{S}$ with $n \geq 2$ is monotonic (continuous) if it is monotonic (continuous) in every argument.*

The natural order implies monotonicity of $\oplus$ and $\odot$, and thus, every aggregator function is monotonic as well. An analogous result is obtained if additionally $\oplus$ and $\odot$ are continuous.

▶ **Lemma 31** (Monotonicity and Continuity of Aggregator Functions). *For a semiring, the operations $\oplus$, $\odot$, and all aggregator functions are monotonic. Moreover, if $\oplus$ and $\odot$ are continuous, then so are all aggregator functions.*

Now we show how to use interpretations to prove boundedness of a wARS. The idea is to use an "embedding" (or "ranking function") $\mathfrak{e}$ which maps every object from A to a non-maximal element of the semiring $\mathbb{S}$. Due to monotonicity of all aggregator functions, the conditions of Thm. 32 ensure that for all nodes v in any finite-depth reduction tree $\mathfrak{T}$, we have $\mathfrak{e}(a_v) \succcurlyeq [\![\mathfrak{T}]\!]^v$. Hence, $\mathfrak{e}(a)$ is a bound on $[\![\mathfrak{T}]\!]$ for all reduction trees $\mathfrak{T} \in \Phi(a)$.

▶ **Theorem 32** (Sufficient and Necessary Condition for Boundedness). *A wARS is bounded if there exists an* embedding $\mathfrak{e} : A \to \mathbb{S} \setminus \{\top\}$ *such that*

- $\mathfrak{e}(a) \succcurlyeq \mathsf{f}_{\mathtt{NF}}(a)$ *for all* $a \in \mathtt{NF}_{\to}$ *and*
- $\mathfrak{e}(a) \succcurlyeq \mathsf{Aggr}_{a \to B}[\mathfrak{e}(b) \mid b \in B]$ *for all* $a \to B$.

Then $\mathfrak{e}(a) \succcurlyeq [\![a]\!]$ *for all* $a \in A$. *The reverse ("only if") holds if* $\oplus$ *and* $\odot$ *are continuous.*

As shown in Sect. 4.3, for every probabilistic ARS, we can obtain a corresponding wARS to analyze PAST/SAST. Hence, Thm. 32 allows proving boundedness of this wARS which then implies PAST/SAST of the original probabilistic ARS.

▶ **Example 33** (Expected Runtime of Probabilistic ARSs). We use Thm. 32 to prove that the expected derivational complexity of the biased random walk in Ex. 19 is finite. We use the embedding $\mathfrak{e}(n) = 3 \odot n = 3 \cdot n$ for all $n \in \mathbb{N}$. Note that we indeed have $\mathfrak{e}(n) \neq \top = \infty$ for all $n \in A = \mathbb{N}$. Moreover, for 0 (the only normal form) we have $\mathfrak{e}(0) = 0 = \mathsf{f}_{\mathtt{NF}}(0)$. Regarding the reduction steps, we have for any $n+1 \to [n, n+2]$ with $n \in \mathbb{N}$:

$$\begin{array}{rrcl} & \mathfrak{e}(n+1) & \succcurlyeq & \mathsf{Aggr}_{n+1 \to [n,n+2]}[\mathfrak{e}(n), \mathfrak{e}(n+2)] \\ \iff & 3 \odot (n+1) & \succcurlyeq & \mathsf{Aggr}_{n+1 \to [n,n+2]}[3 \odot n, 3 \odot (n+2)] \\ \iff & 3 + 3 \cdot n & \succcurlyeq & 1 \oplus {}^2\!/_3 \odot 3 \odot n \oplus {}^1\!/_3 \odot 3 \odot (n+2) \;=\; 3 + 3 \cdot n \end{array}$$

By Thm. 32, the wARS is bounded, which means that the expected derivational complexity of the biased random walk is finite for every starting position $n \in \mathbb{N}$. The embedding $\mathfrak{e}$ gives us a bound on the expected complexity as well, i.e., by Thm. 32 we infer that the expected number of steps is at most *three* times the start position n.

▶ **Example 34** (Termination of TRSs). The next example shows how our approach can be used for automated termination proofs of term rewrite systems. Reconsider the TRS $\mathcal{R}$ from Ex. 16 with the wARS $(\mathcal{T}(\Sigma, \mathcal{V}), \xrightarrow{s}_{\mathcal{R}}, \mathbb{S}_{\mathbb{N}^\infty}, \mathsf{f}_{\mathtt{NF}}^{\mathsf{cplx}}, \mathsf{Aggr}_{a \xrightarrow{s} B}^{\mathsf{cplx}})$. We define the embedding $\mathfrak{e} : \mathcal{T}(\Sigma, \mathcal{V}) \to \mathbb{N}^\infty$ with $\mathfrak{e}(t) \neq \infty$ for all $t \in \mathcal{T}(\Sigma, \mathcal{V})$ recursively as $\mathfrak{e}(\mathcal{O}) = 0$, $\mathfrak{e}(\mathsf{s}(t)) = \mathfrak{e}(t) \oplus 1$, and $\mathfrak{e}(\mathsf{plus}(t_1, t_2)) = 2 \odot \mathfrak{e}(t_1) \oplus \mathfrak{e}(t_2) \oplus 1$. To prove termination of the rewrite system $\mathcal{R}$ for all terms, we show that the two inequations required by Thm. 32 hold for all instantiated rewrite rules.[12] For all $t \in \mathtt{NF}_{\xrightarrow{s}_{\mathcal{R}}}$ we have $\mathfrak{e}(t) \succcurlyeq \mathsf{f}_{\mathtt{NF}}^{\mathsf{cplx}}(\mathcal{O}) = 0$. For the rule $\mathsf{plus}(\mathsf{s}(x), y) \xrightarrow{s} [\mathsf{s}(\mathsf{plus}(x, y))]$ and all $t_1, t_2 \in \mathcal{T}(\Sigma, \mathcal{V})$ we get

$$\begin{array}{rrcl} & \mathfrak{e}(\mathsf{plus}(\mathsf{s}(t_1), t_2)) & \succcurlyeq & \mathsf{Aggr}_{a \xrightarrow{s} [b]}(\mathfrak{e}(\mathsf{s}(\mathsf{plus}(t_1, t_2)))) \\ \iff & 2 \odot \mathfrak{e}(t_1) \oplus \mathfrak{e}(t_2) \oplus 3 & \succcurlyeq & 1 \oplus \mathfrak{e}(\mathsf{s}(\mathsf{plus}(t_1, t_2))) \\ \iff & 2 \cdot \mathfrak{e}(t_1) + \mathfrak{e}(t_2) + 3 & \succcurlyeq & 2 \odot \mathfrak{e}(t_1) \oplus \mathfrak{e}(t_2) \oplus 3 \;=\; 2 \cdot \mathfrak{e}(t_1) + \mathfrak{e}(t_2) + 3 \end{array}$$

and for the rule $\mathsf{plus}(\mathcal{O}, y) \xrightarrow{s} [y]$ we get $\mathfrak{e}(\mathsf{plus}(\mathcal{O}, t_1)) \succcurlyeq \mathsf{Aggr}_{a \xrightarrow{s} [b]}(\mathfrak{e}(t_1)) \iff \mathfrak{e}(t_1) + 1 \succcurlyeq \mathfrak{e}(t_1) + 1$. Again, by Thm. 32 the wARS is bounded, hence the TRS terminates.

If one fixes the semiring $\mathbb{S}$, the interpretation $\mathsf{f}_{\mathtt{NF}}$, and the aggregators $\mathsf{Aggr}_{a \to B}$, searching for such an embedding $\mathfrak{e}$ can often be automated for arbitrary TRSs $\mathcal{R}$ using SMT solvers.

[12] In order to *lift* the inequations from rules to reduction steps, one has to ensure that the embedding $\mathfrak{e}$ is *strictly monotonic*, see, e.g., [5, 29].

▶ **Example 35** (Complexity and Safety). To prove boundedness of the wARS from Ex. 22, we use the embedding $\mathfrak{e}(n) = (\frac{|n|}{2}, \mathsf{true})$ if $n \in \mathbb{Z}$ is even and $\mathfrak{e}(n) = (\infty, \mathsf{false})$ if $n \in \mathbb{Z}$ is odd. Then we have $\mathfrak{e}(0) = (0, \mathsf{true}) = \mathsf{f}_{\mathtt{NF}}(0)$. For odd n, we obtain $\mathfrak{e}(n) = (\infty, \mathsf{false}) = \mathsf{Aggr}_{n \xrightarrow{s} [n-2]}(\mathfrak{e}(n-2)) = \mathsf{Aggr}_{n \xrightarrow{s} [n-2]}(\infty, \mathsf{false}) = (1 + \infty, [n \mod 2 = 0] \vee \mathsf{false})$. For even $n \geq 2$, we get $\mathfrak{e}(n) = (\frac{n}{2}, \mathsf{true}) = \mathsf{Aggr}_{n \xrightarrow{s} [n-2]}(\mathfrak{e}(n-2)) = \mathsf{Aggr}_{n \xrightarrow{s} [n-2]}(\frac{n}{2} - 1, \mathsf{true}) = (1 + \frac{n}{2} - 1, [n \mod 2 = 0] \vee \mathsf{true})$. For even $n \leq -2$, the reasoning is analogous.

6 Proving Lower Bounds on Weights

Next, we discuss how to analyze lower bounds. In Sect. 6.1, we show how to compute a lower bound on weights $[\![a]\!]$ and in Sect. 6.2, we show how to prove unboundedness (i.e., $[\![a]\!] = \top$). Such lower bounds are useful to find bugs or potential attacks, e.g., inputs leading to very high computational costs in terms of runtime or memory consumption.

6.1 Approximating the Weight

Since the weight $[\![a]\!]$ is defined via the supremum of a set, we can approximate $[\![a]\!]$ from below by considering only nodes up to a certain depth and only certain schedulers. In the following, for every reduction tree $\mathfrak{T}$ and $n \geq 0$, let $\mathfrak{T}|_n$ denote the tree that results from $\mathfrak{T}$ by removing all nodes with depth $> n$.

▶ **Corollary 36** (Lower Bound on Weight). *For any $a \in A$, two subsets $\Psi \subseteq \Theta \subseteq \Phi(a)$, and two integers $n, m \in \mathbb{N}$ with $n \leq m$, we have*

$$\mathbf{0} \preccurlyeq \bigsqcup \{[\![\mathfrak{T}|_{n'}]\!] \mid \mathfrak{T} \in \Psi, n' \leq n\} \preccurlyeq \bigsqcup \{[\![\mathfrak{T}|_{m'}]\!] \mid \mathfrak{T} \in \Theta, m' \leq m\} \preccurlyeq [\![a]\!].$$

Cor. 36 states that we can approximate $[\![a]\!]$ by starting with some reduction tree $\mathfrak{T}$ with root a and considering $[\![\mathfrak{T}|_0]\!]$ as a first approximation of the weight (where $[\![\mathfrak{T}|_0]\!] = \mathbf{0}$ if $a \notin \mathtt{NF}_{\to}$). We can consider nodes of larger depth ($\mathbf{0} \preccurlyeq [\![\mathfrak{T}|_0]\!] \preccurlyeq \bigsqcup\{[\![\mathfrak{T}|_0]\!], [\![\mathfrak{T}|_1]\!]\} \preccurlyeq \bigsqcup\{[\![\mathfrak{T}|_0]\!], [\![\mathfrak{T}|_1]\!], [\![\mathfrak{T}|_2]\!]\} \preccurlyeq \cdots \preccurlyeq [\![a]\!]$) and more schedulers ($\mathbf{0} \preccurlyeq [\![\mathfrak{T}|_n]\!] \preccurlyeq \bigsqcup\{[\![\mathfrak{T}|_n]\!], [\![\mathfrak{T}'|_n]\!]\} \preccurlyeq \cdots \preccurlyeq [\![a]\!]$) to refine this approximation. However, we have to compute $[\![\mathfrak{T}|_n]\!]$ for all reduction trees $\mathfrak{T}$ whose root is labeled with a, leading to an exponential number of trees depending on the considered depth and the maximal number of non-deterministic choices between reduction steps of an object.

Thm. 37 shows that this number of calculations is not as high as it seems, and even feasible for deterministic sARSs. Monotonicity of the aggregator functions (Lemma 31) ensures that we have $\mathbf{0} \preccurlyeq [\![\mathfrak{T}|_0]\!] \preccurlyeq [\![\mathfrak{T}|_1]\!] \preccurlyeq [\![\mathfrak{T}|_2]\!] \preccurlyeq \cdots \preccurlyeq [\![a]\!]$, i.e., to approximate the weight up to depth $n \in \mathbb{N}$, we do not have to compute $[\![\mathfrak{T}|_{n'}]\!]$ for all $n' \leq n$, but just $[\![\mathfrak{T}|_n]\!]$. Moreover, we do not need to consider several reduction trees for deterministic systems, but just the supremum obtained when evaluating the "only possible" reduction tree "as much as possible".

▶ **Theorem 37** (Approximating Deterministic Systems). *Let $(A, \to)$ be a deterministic sARS. Then for every $a \in A$, there exists an $(A, \to)$-reduction tree $\mathfrak{T}$ whose root is labeled with a such that $\mathbf{0} \preccurlyeq [\![\mathfrak{T}|_0]\!] \preccurlyeq [\![\mathfrak{T}|_1]\!] \preccurlyeq [\![\mathfrak{T}|_2]\!] \preccurlyeq \cdots \preccurlyeq [\![a]\!]$ and $\bigsqcup\{[\![\mathfrak{T}|_n]\!] \mid n \in \mathbb{N}\} = [\![a]\!]$.*

6.2 Proving Unboundedness by Increasing Loops

While the interpretation method of Thm. 32 is based on a technique to prove termination of ordinary ARSs, finding loops (i.e., a non-empty reduction sequence $a \to \cdots \to a$) is one of the basic methods to disprove termination. If we find a finite-depth RT $\mathfrak{T}$ whose root is

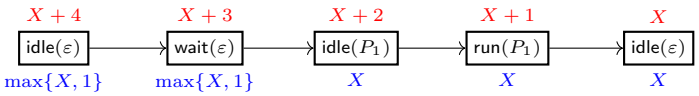


Figure 3 An example of a finite reduction tree containing a loop from $\mathsf{idle}(\varepsilon)$ to itself. The corresponding evaluation of the induced weight polynomial for the runtime is depicted in red above the nodes, and for the space of the waiting list in blue below the nodes.

labeled with a and some other node of $\mathfrak{T}$ is also labeled with a, then this obviously shows non-termination of the underlying sARS. The reason is that we can obtain an RT of infinite depth by simply using the reduction steps from a to a repeatedly. For unboundedness, we additionally require that the weight increases with each loop iteration. However, increasing weights are not sufficient for unboundedness, as shown by the following example.

▶ **Example 38** (Bounded with Increasing Loop)**.** Let $A = \{a, b\}$ with $a \to [a]$ and $a \to [b]$. Moreover, we take the formal languages semiring $\mathbb{S}_\Sigma$ over $\Sigma = \{0, 1\}$, the interpretation $\mathsf{f}_{\mathtt{NF}}(b) = \mathbf{1}_{\mathbb{S}_\Sigma} = \{\varepsilon\}$, and the aggregators $\mathsf{Aggr}_{a\to[a]}(x) = (\{1\} \odot_{\mathbb{S}_\Sigma} x) \oplus_{\mathbb{S}_\Sigma} x = \{1w \mid w \in x\} \cup x$ and $\mathsf{Aggr}_{a\to[b]}(x) = x$. Obviously, the wARS $(A, \to)$ admits a loop from a to a. However, we have $[\![a]\!] = \{1\}^* \neq \Sigma^* = \top$, even though we have a loop with increasing weights.

The problem in Ex. 38 is that $\top$ is not the least upper bound of the weights of the increasing loops. Thus, to infer unboundedness, we require a fixed increase by an element t in each iteration such that $\bigoplus_{i=1}^{\infty} t = \top$. Then, we can use the least upper bound of the series of increasing loops as a lower bound for $[\![a]\!]$, showing unboundedness.

Before we present the corresponding theorem, we define a partial evaluation of finite-depth trees where the label of the leaf v_0 is replaced by a variable X.

▶ **Definition 39** (Induced Weight Polynomial)**.** *Let $\mathfrak{T}$ be a RT of finite depth with a leaf v_0 and let X be a specific variable. Then we define:*

$$
\begin{aligned}
&[\![\mathfrak{T}]\!]^{v_0}_{v_0} = X && \\
&[\![\mathfrak{T}]\!]^{v}_{v_0} = \mathsf{f}_{\mathtt{NF}}(a_v) && \text{if } v \neq v_0 \text{ and } a_v \in \mathtt{NF}_\to, \\
&[\![\mathfrak{T}]\!]^{v}_{v_0} = \mathbf{0} && \text{if } v \neq v_0 \text{ is a leaf and } a_v \notin \mathtt{NF}_\to, \\
&[\![\mathfrak{T}]\!]^{v}_{v_0} = \mathsf{Aggr}_{a_v \to B}\left[[\![\mathfrak{T}]\!]^{w}_{v_0} \mid w \in vE\right] && \text{if } v \text{ is an inner node and } B = [a_w \mid w \in vE]
\end{aligned}
$$

The induced weight polynomial $\mathcal{P}_{v_0}(\mathfrak{T}) \in \mathbb{S}[X]$ *is defined by* $\mathcal{P}_{v_0}(\mathfrak{T}) = [\![\mathfrak{T}]\!]^{r}_{v_0}$*, where $r \in V$ is the root of $\mathfrak{T}$.*

▶ **Example 40** (Induced Weight Polynomial)**.** Reconsider the ARS $(\mathsf{OS}, \to)$ from Ex. 18 in Sect. 4.2. We have the loop $\mathsf{idle}(\varepsilon) \to \mathsf{wait}(\varepsilon) \to \mathsf{idle}(P_1) \to \mathsf{run}(P_1) \to \mathsf{idle}(\varepsilon)$ and the corresponding finite reduction tree can be seen in Fig. 3. Here, the only leaf v_0 is labeled by $\mathsf{idle}(\varepsilon)$. If we consider the runtime by using the wARS $\mathsf{cplx}(\mathsf{OS}, \to)$, then we get the induced weight polynomial $X + 4$. If we consider the size of the waiting list instead, i.e., the wARS $(\mathsf{OS}, \to, \mathbb{S}_{\mathsf{arc}}, \mathsf{f}^{\mathsf{size}}_{\mathtt{NF}}, \mathsf{Aggr}^{\mathsf{size}}_{a\to B})$ from Sect. 4.2, then we get the induced weight polynomial $\max\{X, 1\}$.

▶ **Theorem 41** (Proving Unboundedness via Increasing Loops)**.** *Let $\mathfrak{T}$ be a finite RT where both the root r and a leaf $v_0 \neq r$ are labeled with a. Moreover, let $t \in \mathbb{S}$ with $\bigoplus_{i=1}^{\infty} t = \top$. If $\mathcal{P}_{v_0}(\mathfrak{T})(s) \succcurlyeq s \oplus t$ for all $s \in \mathbb{S}$, then $[\![a]\!] = \top$.*

▶ **Example 42** (Unbounded Runtime). Continuing Ex. 40, we can use Thm. 41 to prove that the runtime of $(\mathsf{OS}, \to)$ (i.e., the wARS $\mathsf{cplx}(\mathsf{OS}, \to)$) is unbounded. Since the induced weight polynomial of the loop $\mathfrak{T}$ from Ex. 40 is $X + 4$, $\sum_{i=1}^{\infty} 4 = \infty$, and $\mathcal{P}_{v_0}(\mathfrak{T})(s) \succcurlyeq s + 4$ for every $s \in \mathbb{S}_{\mathbb{N}^\infty}$, we obtain $[\![\mathsf{idle}(\varepsilon)]\!] = \infty$. However, we cannot use Thm. 41 to prove that the memory consumption of $(\mathsf{OS}, \to)$ (i.e., the wARS $(\mathsf{OS}, \to, \mathbb{S}_{\mathsf{arc}}, \mathsf{f}^{\mathsf{size}}_{\mathsf{NF}}, \mathsf{Aggr}^{\mathsf{size}}_{a \to B})$ to express the size of the waiting list) is unbounded, as there is no $t \in \mathbb{S}_{\mathsf{arc}} \setminus \{\top\}$ with $\bigoplus_{i=1}^{\infty} t = \top$.

There exist several automatic approaches to find loops in, e.g., term rewriting. To lift these techniques to automatic unboundedness proofs for wARSs based on TRSs, one has to formalize the additional property $\exists t \in \mathbb{S} : \bigoplus_{i=1}^{\infty} t = \top \wedge \forall s \in \mathbb{S} : \mathcal{P}_{v_0}(\mathfrak{T})(s) \succcurlyeq s \oplus t$ as an SMT problem over the corresponding semiring theory.

7 Conclusion

We have developed semiring semantics for abstract reduction systems using arbitrary complete lattice semirings. These semantics capture and generalize numerous formalisms that have been studied in the literature. Due to our generalization of these formalisms, we can now use techniques and ideas from, e.g., termination analysis, to prove boundedness or other properties (or combinations of properties) of reduction systems using a completely different semiring (e.g., a tuple semiring). In the future, this may be used to improve the automation of specific analyses and lead to further applications of our uniform framework.

There are many directions for future work, e.g., one can try to improve our techniques on proving and disproving boundedness. In order to develop techniques amenable to automation one could focus on term rewrite systems where the reduction relation is represented by a finite set of rules. For proving termination of TRSs, there exist more powerful techniques than just using interpretations (e.g., the *dependency pair framework* [17]). Thus, we aim to develop a similar framework to analyze boundedness for weighted TRSs in the future. Currently our approach only focuses on (bounds on) worst-case reductions. Therefore, in the future we will also investigate extensions in order to express and analyze properties like starvation freedom where one has to consider all (infinite) reductions. We are also interested in adapting concepts like confluence and unique normal forms to weighted rewriting, e.g., by studying rewrite systems where every reduction tree that starts with the same object has the same weight if it is evaluated "as much as possible".

Acknowledgements: We thank the reviewers for their useful remarks and suggestions.

References

1 Samson Abramsky, Dov M. Gabbay, and Tom S. E. Maibaum. *Handbook of Logic in Computer Science. Volume 3. Semantic Structures*. Clarendon Press, 1994. URL: `https://global.oup.com/academic/product/handbook-of-logic-in-computer-science-9780198537625`.

2 Emma Ahrens, Jan-Christoph Kassing, Jürgen Giesl, and Joost-Pieter Katoen. Weighted rewriting: Semiring semantics for abstract reduction systems. *CoRR*, abs/2505.08496, 2025. `doi:10.48550/arXiv.2505.08496`.

3 Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. On probabilistic term rewriting. *Sci. Comput. Program.*, 185, 2020. `doi:10.1016/j.scico.2019.102338`.

4 Martin Avanzini, Georg Moser, and Michael Schaper. A modular cost analysis for probabilistic programs. *Proc. ACM Program. Lang.*, 4, 2020. `doi:10.1145/3428240`.

5 Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. `doi:10.1017/CBO9781139172752`.

6 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

7 Kevin Batz, Adrian Gallus, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Tobias Winkler. Weighted programming: a programming paradigm for specifying mathematical models. *Proc. ACM Program. Lang.*, 6(OOPSLA1):1–30, 2022. doi:10.1145/3527310.

8 Vaishak Belle and Luc De Raedt. Semiring programming: A semantic framework for generalized sum product problems. *International Journal of Approximate Reasoning*, 126:181–201, 2020. doi:10.1016/j.ijar.2020.08.001.

9 Guillaume Bonfante, Adam Cichon, Jean-Yves Marion, and Hélène Touzet. Algorithms with polynomial interpretation termination proof. *J. Funct. Program.*, 11(1):33–53, 2001. doi:10.1017/S0956796800003877.

10 Guillaume Bonfante, Jean-Yves Marion, and Jean-Yves Moyen. Quasi-interpretations a way to control resources. *Theor. Comput. Sci.*, 412(25):2776–2796, 2011. doi:10.1016/j.tcs.2011.02.007.

11 Olivier Bournez and Claude Kirchner. Probabilistic rewrite strategies. Applications to ELAN. In *Proc. RTA '02*, LNCS 2378, pages 252–266, 2002. doi:10.1007/3-540-45610-4_18.

12 Olivier Bournez and Florent Garnier. Proving positive almost-sure termination. In *Proc. RTA '05*, LNCS 3467, pages 323–337, 2005. doi:10.1007/978-3-540-32033-3_24.

13 Sophie Brinke, Erich Grädel, Lovro Mrkonjić, and Matthias Naaf. Semiring provenance in the infinite. In *The Provenance of Elegance in Computation - Essays Dedicated to Val Tannen*, OASIcs 119, pages 3:1–3:26, 2024. doi:10.4230/OASIcs.Tannen.3.

14 James Cheney, Laura Chiticariu, and Wang Chiew Tan. Provenance in databases: Why, how, and where. *Found. Trends Databases*, 1(4):379–474, 2009. doi:10.1561/1900000006.

15 Katrin M. Dannert and Erich Grädel. Provenance analysis: A perspective for description logics? In *Description Logic, Theory Combination, and All That: Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday*, LNCS 11560, pages 266–285, 2019. doi:10.1007/978-3-030-22102-7_12.

16 Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of Weighted Automata*. Springer, 2009. doi:10.1007/978-3-642-01492-5.

17 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006. doi:10.1007/s10817-006-9057-7.

18 Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The termination and complexity competition. In *Proc. TACAS '19*, LNCS 11429, pages 156–166, 2019. Website of *TermComp*: https://termination-portal.org/wiki/Termination_Competition. doi:10.1007/978-3-030-17502-3_10.

19 Boris Glavic. Data provenance. *Found. Trends Databases*, 9(3-4):209–441, 2021. doi:10.1561/1900000068.

20 Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proc. PODS '07*, pages 31–40, 2007. doi:10.1145/1265530.1265535.

21 Todd J. Green and Val Tannen. The semiring framework for database provenance. In *Proc. PODS '17*, pages 93–99, 2017. doi:10.1145/3034786.3056125.

22 Dieter Hofbauer and Clemens Lautemann. Termination proofs and the length of derivations. In *Proc. RTA '89*, LNCS 355, pages 167–177, 1989. doi:10.1007/3-540-51081-8_107.

23 Mahmoud Abo Khamis, Hung Q. Ngo, Reinhard Pichler, Dan Suciu, and Yisu Remy Wang. Convergence of Datalog over (pre-) semirings. *J. ACM*, 71(2):8:1–8:55, 2024. doi:10.1145/3643027.

24 Cynthia Kop and Deivid Vale. Cost-size semantics for call-by-value higher-order rewriting. In *Proc. FSCD '23*, LIPIcs 260, 2023. doi:10.4230/LIPIcs.FSCD.2023.15.

25 Jim Laird, Giulio Manzonetto, Guy McCusker, and Michele Pagani. Weighted relational models of typed lambda-calculi. In *Proc. LICS '13*, pages 301–310, 2013. doi:10.1109/LICS.2013.36.

26 Dallas S. Lankford. On proving term rewriting systems are Noetherian. Technical Report Memo MTP-3, Department of Mathematics, Louisiana Technical University, 1979. URL: https://www.ens-lyon.fr/LIP/REWRITING/TERMINATION/Lankford_Poly_Term.pdf.

27 Matthias Naaf, Florian Frohn, Marc Brockschmidt, Carsten Fuhs, and Jürgen Giesl. Complexity analysis for term rewriting by integer transition systems. In *Proc. FroCoS '17*, LNCS 10483, pages 132–150, 2017. doi:10.1007/978-3-319-66167-4_8.

28 Igor Sedlár. Kleene algebra with tests for weighted programs. In *In Proc. ISMVL '23*, pages 111–116, 2023. doi:10.1109/ISMVL57333.2023.00031.

29 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

A Comparison to Weakest Preweightings for Weighted Imperative Programs

In this appendix, we briefly compare our semantics to the formalism of *weakest preweightings* (wp) for weighted imperative programs from [7], and we present a more complex mutual exclusion algorithm from [7] which would require *weakest liberal preweightings* (wlp).

A.1 Weakest Preweightings

Assuming familiarity with the concepts and notation introduced in [7], we show how to adapt them to our setting. This should serve as a high-level illustration of the relationship between weighted abstract reduction systems and the weighted imperative programs from [7]. To do so, we show how to transform the imperative program modeling the ski rental problem from [7] into a weighted ARS using the tropical semiring $\mathbb{S}_{\mathsf{trop}}$ such that $[\![\delta_0]\!]$ is equal to the "weakest preweighting" $\mathsf{wp}\ [\![\mathsf{SkiAlg}]\!]\ (\mathbf{1})$ of the *postweighting* $\mathbf{1}$ on the initial *program configuration* δ_0, where SkiAlg is depicted in Algorithm 1.

Algorithm 1

while $n > 0$ **do**
  {
    $\odot\, 1$;
    $n := n - 1$;
  } $\oplus$ {
    $\odot\, y$;
    $n := 0$;
  }

The ski renting problem is a classical optimization problem. Consider a person that does not own a pair of skis but is going on a skiing trip for an initially unknown number of $n \in \mathbb{N}$ days. At the dawn of each day, the person can decide whether to rent a pair of skis for 1€ for that day or buy their own pair of skis for y€ and go skiing without any further costs, instead. What is the optimal strategy for the person to spend a minimal amount of money? In [7] this problem has been modeled via the imperative program SkiAlg in Algorithm 1. In general, weighted algorithms contain the standard control-flow instructions from the guarded command language (GCL) syntax including non-deterministic branching ($\oplus$), with an additional $\odot s$ statement, where $s \in \mathbb{S}$ is an element from a semiring. The statement represents a skip operation (or noop, i.e., skipping the execution step and doing nothing) that is used to give a *weight* to every execution path through the program. The set of all such programs of the *weighted guarded command language* is denoted by wGCL.

We first define the corresponding sARS $(A, \to)$ representing the possible computations of the program. Here, we let A be the set of all *configurations* of the program and $\to$ is defined via the *transition* relation ([7], Definition 3.1). A configuration (C, σ, n, w) consists of the remaining program $C \in \mathsf{wGCL} \cup \{\downarrow\}$, where $\downarrow$ denotes an already terminated program; an instantiation of the program variables $\sigma : \mathcal{V} \to \mathbb{N}$, where Σ is the set of all such instantiations; a number of already performed execution steps $n \in \mathbb{N}$; and finally, a string $w \in \{L, R\}^*$ indicating the performed choices at non-deterministic steps.

▶ **Definition 43** (sARS for Ski Renting Problem)**.** *Following the notations from [7], let* $Q = (\mathsf{wGCL} \cup \{\downarrow\}) \times \Sigma \times \mathbb{N} \times \{L, R\}^*$ *be the set of all* configurations*. Moreover, let*

$\to = \{\delta \to [\delta_1, \delta_2, \ldots] \mid (\delta, w, \delta_i) \in \Delta\}$. *Here,* $\Delta \subseteq Q \times \mathbb{S} \times Q$ *denotes the transition relation according to the small-step operational semantics given in [7], where* w *is the weight of the transition* (δ, w, δ_i) *from configuration* δ *to configuration* δ_i*. To determine the order of the configurations* $\delta_1, \delta_2, \ldots$*, we use an arbitrary total order on the transitions. The sARS representing the algorithm of the ski renting problem is* $(Q, \to)$*.*

To answer the question of the ski renting problem, one can compute the weakest preweighting $\mathsf{wp}\,[\![\mathsf{SkiAlg}]\!]\,(\mathbf{1})$ considering the tropical semiring $\mathbb{S}_{\mathsf{trop}} = (\mathbb{N}^\infty, \min, +, \infty, 0)$ and the *postweighting* $\mathbf{1}$. When applying $\mathsf{wp}\,[\![\mathsf{SkiAlg}]\!]\,(\mathbf{1})$ to the initial configuration where the program variable n has the value n_0, one obtains $n_0 \oplus y = \min\{n_0, y\}$, indicating that it is only beneficial to buy skis if they cost less than the number of skiing days.

A postweighting is a function $f : \Sigma \to \mathbb{S}$ mapping final states to an element in the semiring, i.e., this corresponds to our interpretation of the normal forms $\mathsf{f}_{\mathtt{NF}}$. The wp transformer aggregates the postweighting along the program execution paths by multiplication and by summing over multiple possibilities during non-determinism. In our setting, this can be expressed by choosing aggregators that are weighted sums $\mathsf{Aggr}_{a \to B} = \sum_{1 \le i \le |B|} e_i \odot v_i$ for every $a \to B$ with $e_i \in \mathbb{S}$ for all $1 \le i \le |B|$.

▶ **Definition 44** (Aggregators and Interpretation for Ski Renting Problem)**.** *Given a postweighting* f*, we define the interpretation of normal forms as* $\mathsf{f}_{\mathtt{NF}} = f$ *and the aggregators as* $\mathsf{Aggr}_{a \to B} = \bigoplus[w_i \odot v_i \mid (a, w_i, b_i) \in \Delta]$*.*

Thus, we obtain the following weighted abstract reduction system.

▶ **Definition 45** (wARS for the Ski Renting Problem)**.** *The wARS for the ski renting problem is* $(Q, \to, \mathbb{S}_{\mathsf{trop}}, \mathsf{f}_{\mathtt{NF}}, \mathsf{Aggr}_{a \to B})$*, with* Q*,* $\to$ *as in Def. 43 and* $\mathsf{f}_{\mathtt{NF}}$*,* $\mathsf{Aggr}_{a \to B}$ *as in Def. 44.*

In the end, we get $[\![\delta_0]\!] = \mathsf{wp}\,[\![\mathsf{SkiAlg}]\!]\,(\mathbf{1})(\delta_0) = n_0 \oplus y = \max\{n_0, y\}$, where δ_0 is the initial configuration that sets the program variable n to the natural number $n_0 \in \mathbb{N}$.

Thus, we can express weakest preweightings from [7] in our formalism. One might think that we can even express more than with weakest preweightings, as we can use aggregators that are not just simple weighted sums but arbitrary polynomials. However, we expect that one can transform any wGCL program C into another wGCL program C' that can represent more complex aggregators of C by ordinary weighted sums of C'. This might be an interesting direction for future research.

A.2 Weakest Liberal Preweightings

In order to express "weakest liberal preweightings" (wlp) from [7] in our setting, we would have to extend our definition of $[\![a]\!]$ in Def. 13. This is similar to the problem of analyzing best-case reductions described in Sect. 4.5.

To see why we currently cannot express wlp, consider Algorithm 2 which describes an operating system guaranteeing mutual exclusion. This algorithm from [7] (adapted from [6]) handles N processes that want to access a shared critical section which may only be accessed by y processes simultaneously. The status $\ell[i]$ of a selected process i can be either idle (n), waiting (w), or critical (c). If the process i is idle, it becomes waiting. If it is waiting, one checks whether the shared section may be entered ($y > 0$). Otherwise (if $y = 0$), the process keeps on waiting. If process i is already in the critical section, it releases it and y is updated.

Algorithm 2

```
while true do
    ⨁_{j=1}^{N} {i := j};
    if ℓ[i] = n then
        ℓ[i] := w;
    if ℓ[i] = w then
        if y > 0 then
            ⊙C_i;
            y := y − 1;
            ℓ[i] := c;
        else
            ⊙W_i;
    if ℓ[i] = c then
        ⊙R_i;
        y := y + 1;
        ℓ[i] := n;
```

In [7], the natural language semiring and the alphabet $\Sigma = \{C_i, W_i, R_i \mid 1 \leq i \leq N\}$ is used to describe the different actions, i.e., if process i enters the critical section, waits, or releases the critical section, the corresponding branch is weighted by C_i, W_i, or R_i, respectively. Now, wlp allows to reason about the infinite paths represented by this loop. More precisely, one can analyze the language of ω-words produced by the loop via wlp, and show that there is a word like W_2^ω in the language, which means that process 2 can wait infinitely long. This disproves starvation freedom of Algorithm 2.

We can express this algorithm as an sARS similar to the one in Def. 43. As in Ex. 23, to analyze starvation freedom of such an algorithm, we can use the tuple semiring $\bigtimes_{i=1}^{N} \mathbb{S}_{\mathbb{N}^\infty}$. Then the i-th component in a tuple describes how often the process i has entered the critical section. In the steps with the weight C_i, the process i enters the critical section, so that we have to increase the value in the i-th component of the tuple. The corresponding aggregator for such a step would be $e_i \oplus v_1$, where e_i denotes the tuple $(0, \ldots, 0, 1, 0, \ldots, 0)$ where the i-th component is 1 and all other components are 0. However, as in Ex. 23, starvation freedom cannot be expressed with our current definition of $[\![a]\!]$, but we would have to analyze (bounds on) best-case reductions.