Math. Struct. in Comp. Science (2010), *vol.* 20, *pp.* 589–624. © Cambridge University Press 2010
doi:10.1017/S0960129510000113 First published online 3 June 2010

Long-run cost analysis by approximation of linear operators over dioids

DAVID CACHERA[†], THOMAS JENSEN[‡], ARNAUD JOBIN[§] and PASCAL SOTIN[¶]

[†]*IRISA/ENS Cachan (Bretagne), Campus de Beaulieu, F-35042 Rennes cedex, France*
Email: david.cachera@irisa.fr
[‡]*IRISA/CNRS, Campus de Beaulieu, F-35042 Rennes cedex, France*
Email: thomas.jensen@irisa.fr
[§]*Université Rennes 1, Campus de Beaulieu, F-35042 Rennes cedex, France*
Email: arnaud.jobin@irisa.fr
[¶]*CNRS/DGA, F-35042 Rennes cedex, France*
Email: pascal.sotin@irisa.fr

Received 13 July 2009; revised 19 March 2010

In this paper we present a semantics-based framework for analysing the quantitative behaviour of programs with respect to resource usage. We start from an operational semantics in which costs are modelled using a dioid structure. The dioid structure of costs allows the definition of the quantitative semantics as a linear operator. We then develop a theory of approximation of such a semantics, which is akin to what is offered by the theory of abstract interpretation for analysing qualitative properties, in order to compute effectively global cost information from the program. We focus on the notion of long-run cost, which models the asymptotic average cost of a program. The abstraction of the semantics has to take two distinct notions of order into account: the order on costs and the order on states. We prove that our abstraction technique provides a correct approximation of the concrete long-run cost of a program.

1. Introduction

This paper is concerned with mathematical structures for analysing quantitative properties of a program's use of resources (time, memory, ...). Such properties can be characterised formally using an operational model of program execution where such non-functional properties are made explicit. In particular, our concern is to develop a theory of approximation of such a quantitative operational semantics that is akin to what is offered by the theory of abstract interpretation for defining semantics-based program analysis of qualitative program properties. More precisely, we will study a standard small-step operational semantics expressed as a transition relation $\sigma \rightarrow^{q} \sigma'$ between states $\sigma, \sigma' \in \Sigma$

[†] The work of this author was partly supported by INRIA.

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

extended with *a cost* $q \in Q$ associated with each transition. The set Q of costs has two operations for composing costs: a 'product' operator $\otimes$ that combines the costs along an execution path and a 'sum' operator $\oplus$ that combines costs coming from different paths. These operators will give Q the structure of a dioid, that is, a semiring enriched with additional properties that makes it particularly well suited for approximating program behaviour.

Using a dioid structure has two advantages. First, the sum operator induces a partial order on costs that will serve as a basis for approximating costs in the abstraction process. Secondly, due to the distributivity property of semirings, there is a straightforward way to transform the labelled operational semantics into a transition matrix whose entries represent the cost of passing from one state of the program to another. This recasts the semantics of a program as a linear operator on the moduloid of vectors of elements of Q indexed over Σ. Such a semantics benefits from a rich set of algebraic properties of linear operators, and allows the extraction of global quantitative information about the behaviour of the program.

We focus on analysing programs with cyclic behaviour (such as reactive systems) in which the property of interest is the asymptotic average cost along cycles, rather than the global cost of the entire execution. We define the notion of *long-run cost* for a program that provides an over-approximation of the average cost per transition of long traces. This notion corresponds to the maximum average of costs accumulated along a cycle of the program semantics and is computed from the traces of the successive iterates of the cost matrix. We show that, for a restricted but common class of dioids where the multiplication corresponds to the arithmetical plus, this notion of long-run cost can indeed represent the asymptotic behaviour of the program, since it corresponds to the limit of the maximum average cost of arbitrarily long traces.

The quantitative operational semantics operates on state spaces that may be large or even infinite, so the computation of quantitative semantic models, like their qualitative counterparts, is usually not tractable. Hence, it is necessary to develop techniques for abstracting this semantics so that we can obtain an approximation of the program costs that is feasible to compute. The abstract interpretation approach commonly used for static analysis of programs relies on the notion of a Galois connection between a pair of concrete and abstract semantic domains. The concrete semantics is often represented as a set (of reachable states, traces, and so on), and elements of the abstract domain represent properties, that is, subsets of concrete states. These two domains are equipped with a *lattice* structure, where the (partial) order between elements represents a quantity of information carried by the property. The Galois connection itself is a pair of mappings (α, γ) that are mutually pseudo-inverse: the abstraction function α from the concrete to the abstract domain and the concretisation function γ are monotone functions verifying $c \leqslant \gamma(d) \iff \alpha(c) \leqslant d$. Using such a Galois connection allows us to define optimal abstract transfer functions from the concretes ones that are correct in the sense that they over-approximate the concrete behaviour of the program.

In our model, where the concrete semantics takes not only states but also costs into account, we have to design an abstraction method that encompasses both aspects. The order used for over-approximation is no longer the lattice order of states, but the order over vectors of costs induced by the addition operator of the cost dioid. We then have to find a

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

pair $(\alpha^{\uparrow}, \gamma^{\uparrow})$ of mappings between moduloids that are pseudo-inverse with respect to that order. When no lattice structure on the set of states is assumed, there is a straightforward method to obtain this pair. This method is adequate for simple abstractions that result in partitionning the concrete state space, but yields matrices of huge dimension, and does not allow the design of more complex abstractions, in particular, by reusing the existing abstract domains provided by the abstract interpretation framework. In order to benefit from this collection of existing abstractions, we had to make a deeper correspondence between lattices and moduloids that is compatible between the two different notions of order that are involved in each structure.

Given such an abstraction–concretisation pair over the semantic domains, we abstract the transition matrix of the program itself into a matrix of reduced size. As with the usual abstract interpretation, a sufficient condition for an abstraction of the semantics to be correct, that is, to give an over-approximation of the real cost with respect to the order relation induced by the summation operator of the dioid, is expressed by composing abstraction with the concrete or abstract semantics, respectively. This condition can be written as

$$\alpha^{\uparrow} \circ M \leqslant M^{\sharp} \circ \alpha^{\uparrow}$$

where $\circ$ is the composition operator between linear mappings, and thus matrix multiplication, and M and $M^{\sharp}$ are matrices representing the concrete and abstract semantics, respectively. The order here is the dioid order extended pointwise to matrices.

An important feature of our framework is that an abstract semantics that is correct by construction can be derived from the concrete one. The main property of interest is that, in addition to the usual safe over-approximation (from a set-order point of view) of the concrete semantics by the abstract one, the costs are also over-approximated with respect to the dioid order. In particular, the long-run cost of a program is safely approximated by an abstract long-run cost. Note that both concrete and abstract long-run costs belong to the same cost dioid.

The paper is organised as follows. Section 2 defines the quantitative semantics as a linear operator over a moduloid. We give the general form of this semantics together with a precise definition of the notion of a cost dioid that we use throughout the paper. Section 3 defines the notion of abstraction together with its correctness, and shows how we can derive an abstract semantics that is correct by construction. It also explains how classical Galois connections can be lifted in our linear operator model. Section 4 defines the notion of long-run cost, relating it to the asymptotic behaviour of the trace semantics, and shows how a correct abstraction yields an over-approximation of the concrete long-run cost of a program. Section 5 gives an application example to a simple imperative language with explicit management of energy levels. Section 6 discusses related work and Section 7 gives conclusions and describes some future research directions.

2. Linear operator semantics

We give a general framework for expressing quantitative operational semantics. Transitions of these semantics will be equipped with *quantities* (or *costs*) depending on the accessed states. Let P be a program; its semantic domain is the countable set of states Σ. The

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

quantitative operational semantics of P is given as a transition relation, which is defined by transitions of the following form: $\sigma \to^q \sigma'$ where σ, σ' are states of Σ and q is the cost attached to the transition from σ to σ' (q is a function of σ and σ'). The set Q of costs and its structure will be made precise in Section 2.1. We associate with P the transition system $T = \langle \to_\cdot, I \rangle$, where I is the set of initial states of P. The trace semantics of P is defined as the trace semantics of T:

$$[\![P]\!]_{tr} = [\![T]\!]_{tr} = \{\sigma_0 \to^{q_0} \dots \sigma_{n-1} \to^{q_{n-1}} \sigma_n \mid \sigma_0 \in I, \sigma_i \to^{q_i} \sigma_{i+1}\}$$

2.1. *Cost dioid*

The small-step quantitative operational semantics induces a labelled transition system over Σ with labels in Q and a transition relation $\to_\cdot \subseteq \Sigma \times \Sigma \to Q$, written $\sigma \to^q \sigma'$. Such a transition states that a direct (one-step) transition from σ to σ' costs q. These unitary transitions can be combined into big-step transitions, using two operators: $\otimes$ for accumulating costs and $\oplus$ to get a maximum of different costs. These operators will form a dioid on Q, as explained below. Costs can be defined in more general ways (for instance, one could use a more general algebra of costs as in Aspinall *et al.* (2007)) but the present definition covers a number of different costs and has interesting computational properties since it can be used within a linear operator semantic framework – see Section 2.2.

The operator $\otimes$ on Q defines the global cost of a sequence of transitions, $\sigma \to^{q_1} \cdots \to^{q_n} \sigma'$ simply as $q = q_1 \otimes \cdots \otimes q_n$. This is written $\sigma \overset{\pi}{\Rightarrow}^{q} \sigma'$ where π is a sequence of states that has σ and σ' as the first and last states, respectively.

There may be several paths between a state σ and a state σ' due to the presence of loops and potential non-determinism in the semantics. Let the set of possible paths be $\Pi_{\sigma,\sigma'} = \{\pi \mid \sigma \overset{\pi}{\Rightarrow}^{q_\pi} \sigma'\}$. The global cost between σ and σ' will be defined using the operator $\oplus$ on Q to be $q = \bigoplus_{\pi \in \Pi_{\sigma,\sigma'}} q_\pi$. Formally, the two operators have to fulfill the conditions of a (commutative) dioid.

Definition 1. A *commutative dioid* is a structure $(Q, \oplus, \otimes)$ such that:

1 Operator $\otimes$ is associative and commutative and has a neutral element e. Quantity e represents a transition that costs nothing.
2 Operator $\oplus$ is associative and commutative and has $\bot$ as neutral element. Quantity $\bot$ represents the impossibility of a transition.
3 $\otimes$ is distributive over $\oplus$, and $\bot$ is an absorbing element for $\otimes$ ($\forall x. x \otimes \bot = \bot \otimes x = \bot$).
4 The preorder defined by $\oplus$ ($a \leqslant b \Leftrightarrow \exists c : a \oplus c = b$) is an order relation (that is, it satisfies $a \leqslant b$ and $b \leqslant a \Rightarrow a = b$).

By its nature, a dioid cannot be a ring, since there is an inherent contradiction between the fact that $\oplus$ induces an order relation and the fact that every element has an inverse for $\oplus$. The following lemma is a classical result of dioid theory.

Lemma 1 (Gondran and Minoux 2008; Bistarelli *et al.* 1997, Theorem 2.4). $\oplus$ and $\otimes$ preserve the order $\leqslant$, that is, for all $a, b, c \in Q$ with $a \leqslant b$, $a \otimes c \leqslant b \otimes c$ and $a \oplus c \leqslant b \oplus c$.

If several paths go from some state σ to a state σ' at the same cost q, we will require that the global cost is also q, that is, we work with idempotent dioids.

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

Definition 2. A dioid $(Q, \oplus, \otimes)$ is *idempotent* if $q \oplus q = q$ for all q in Q.

For instance, $(\overline{\mathbb{R}}, \mathtt{max}, +)$ and $(\overline{\mathbb{R}}, \mathtt{min}, +)$ are idempotent dioids, where $\overline{\mathbb{R}}$ stands for $\mathbb{R} \cup \{-\infty, +\infty\}$. The induced orders in these cases are, respectively, the orders $\leqslant$ and $\geqslant$ over real numbers, extended to $\overline{\mathbb{R}}$ in the usual way. Note that in an idempotent dioid $a \leqslant b \Leftrightarrow a \oplus b = b$. This equivalence shows that there is a tight link between the notion of idempotent addition and the sup-semilattice structure: on the one hand, an idempotent addition induces an ordered structure (as we have already seen in Definition 1) in which each pair of elements (a, b) has the upper bound $a \oplus b$. On the other hand, considering a sup-semilattice and defining the result of the addition of two elements as their upper bound, we can in turn define an idempotent addition (Baccelli *et al.* 1992).

Idempotent dioids are also called tropical semirings in the literature. The fact that sets of states may be infinite, together with the use of residuation theory in Section 3, means that our structure must contain the addition of any set of costs[†].

Definition 3. An idempotent dioid is *complete* if it is closed with respect to infinite sums and the distributivity law also holds for an infinite number of summands: for any set $X \subseteq Q$, the (possibly infinite) sum

$$\bigoplus_{x \in X} x$$

exists in the dioid, and for all $a \in Q$,

$$a \otimes \left(\bigoplus_{x \in X} x \right) = \bigoplus_{x \in X} (a \otimes x).$$

A complete dioid is naturally equipped with a top element, which we shall write $\top$, that is the sum of all its elements. Recall that a complete dioid is always a complete lattice, and is thus equiped with a meet operator $\wedge$ (Baccelli *et al.* 1992). The notion of long-run cost that we will define in Section 4 relies on the computation of an average cost along the transitions of a cycle. This requires the existence of an nth root function.

Definition 4. A dioid $(Q, \oplus, \otimes)$ is equipped with an nth *root function* if for all q in Q, equation $X^n = q$ has a *unique* solution in Q, which we denote by $\sqrt[n]{q}$.

A sequence containing n transitions, each costing $\sqrt[n]{q}$ on average, will thus cost q. Some examples of nth roots can be found in Table 1. To enable us to deal easily with the nth root, we assume that the nth power is $\oplus$-lower-semicontinuous ($\oplus$-lsc for short).

Definition 5. In a complete dioid Q, the nth power is said to be $\oplus$-*lsc* if for all $X \subseteq Q$, $(\bigoplus_{x \in X} x)^n = \bigoplus_{x \in X} x^n$.

This assumption and its consequences will be very useful for the theorems relating long-run cost and trace semantics in Section 4. Note that this equality remains true for finite X (in that case the nth power is said to be a $\oplus$-*morphism*).

The following definition summarises the required conditions for our structure.

[†] In this way we define a complete sup-semilattice over Q.

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

	carrier set	$\oplus$	$\otimes$	$\sqrt[n]{q}$
Double-idempotent	$\mathbb{Q} \cup \{+\infty, -\infty\}$	min	max	q
	$\mathbb{R} \cup \{+\infty, -\infty\}$	max	min	q
	$\mathscr{P}(S)$	$\cap$	$\cup$	q
	$\mathscr{P}(S)$	$\cup$	$\cap$	q
Cancellative	$\mathbb{R}_+^m \cup \{+\infty\}$	min	$+$	$\frac{q}{n}$
Selective	$\mathbb{R}_+ \cup \{+\infty\}$	max	$\times$	$q^{\frac{1}{n}}$
	$\mathbb{Q} \cup \{+\infty, -\infty\}$	max	$+$	$\frac{q}{n}$
	$\mathbb{R} \cup \{+\infty, -\infty\}$	min	$+$	$\frac{q}{n}$

Table 1. *Some examples of cost dioids*

Definition 6 (cost dioid). A *cost dioid* is a *complete* and *idempotent* commutative dioid equipped with an nth root operation, where the nth power is $\oplus$-lsc.

The following properties are naturally true for such dioids.

Proposition 1. In a cost dioid Q:

(1) The nth root is $\oplus$-lsc, that is, $\forall X \subseteq Q, \forall n > 0$,

$$\sqrt[n]{\bigoplus_{x \in X} x} = \bigoplus_{x \in X} \sqrt[n]{x}.$$

(2) For all $a, b \in Q$ and $n, m > 0$,

$$\sqrt[n]{a} \oplus \sqrt[m]{b} \geqslant \sqrt[n+m]{a \otimes b}.$$

Property (1) follows immediately from the fact that the nth power is $\oplus$-lsc.

We will need the following two lemmas to prove property (2). The first is a Cauchy inequality (Dudnikov and Samborskii 1987; Dudnikov and Samborskii 1992).

Lemma 2. In a cost dioid Q, we have $\forall n \in > 0,\ \forall x_1, \ldots, x_n \in Q$,

$$x_1 \otimes \cdots \otimes x_n \leqslant x_1^n \oplus \cdots \oplus x_n^n.$$

Proof. The product $x_1 \otimes \cdots \otimes x_n$ appears in the expansion of $(x_1 \oplus \cdots \oplus x_n)^n$. Since we work in a cost dioid, the nth power is a $\oplus$-morphism. So we can write $x_1^n \oplus \cdots \oplus x_n^n = (x_1 \oplus \cdots \oplus x_n)^n = x_1 \otimes \cdots \otimes x_n \oplus d$ where d stands for the rest of the expansion. We can then conclude by using the definition of the order relation. □

Lemma 3. In a cost dioid Q, we have $\forall a, b \in Q, \forall n > 0$,

$$a \leqslant b \Leftrightarrow a^n \leqslant b^n.$$

Proof.

(⇒) This direction is just an expression of the fact that in a dioid, the nth power is monotone, which is a direct consequence of Lemma 1.

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

(⇐) In a cost dioid, the nth root is a $\oplus$-morphism, so it is monotone (Baccelli *et al.* 1992). □

We can now prove property (2) of Proposition 1.

Proof of Proposition 1, part 2. We proceed by equivalence, applying Lemma 3 to $\sqrt[n+m]{a \otimes b}$ and $\sqrt[n]{a} \oplus \sqrt[m]{b}$, elements of Q, and to $mn(m+n)$, which is non-negative.

$$\begin{aligned} & & \sqrt[n+m]{a \otimes b} &\leqslant \sqrt[n]{a} \oplus \sqrt[m]{b} \\ &\iff & a^{mn} \otimes b^{mn} &\leqslant a^{m(m+n)} \oplus b^{n(m+n)} \\ &\iff \underbrace{a^m \otimes \cdots \otimes a^m}_{\text{n times}} \otimes \underbrace{b^n \otimes \cdots \otimes b^n}_{\text{m times}} &&\leqslant (a^m)^{m+n} \oplus (b^n)^{m+n} \end{aligned}$$

The former inequality is always true in a cost dioid. Indeed, it is just an instance of Lemma 2 with the following $m+n$ terms: $x_1 = \cdots = x_n = a^m$ and $x_{n+1} = \cdots = x_{n+m} = b^n$. As property (2) is equivalent to this inequality, this completes the proof. □

Although the definition of cost dioids may seem rather restrictive, we now show that many classes of dioids found in the literature are indeed cost dioids. We first recall some standard definitions.

Definition 7. A dioid $(Q, \oplus, \otimes)$ is:

— *selective* if for all a, b in Q, we have $a \oplus b =$ either a or b;
— *double-idempotent* if both $\oplus$ and $\otimes$ are idempotent;
— *cancellative* if for all a, b, c in Q, we have $a \otimes b = a \otimes c$ and $a \neq \bot$ implies $b = c$.

Note that in a double-idempotent dioid, $x^n = x$. Thus, a double-idempotent dioid is naturally equipped with an nth root, which is the identity function.

Proposition 2. The following dioids are cost dioids.

(1) Complete and selective commutative dioids with an nth root operation.
(2) Complete and double-idempotent commutative dioids.
(3) Complete idempotent commutative dioids satisfying the cancellation condition, and for which for all q in Q, equation $X^n = q$ has at least one solution.

For dioids of classes (1) and (3), we will need the following lemma.

Lemma 4. In a complete and idempotent commutative dioid equipped with an nth root, the nth power is $\oplus$-lsc if and only if it is a $\oplus$-morphism.

Proof. Naturally we just need to prove the *if* part. Assume that the nth power is a $\oplus$-morphism. This immediately implies that the nth root is a $\oplus$-morphism. Thus, the nth root is monotone (Baccelli *et al.* 1992). Let X be a non-empty subset of Q. If $x \in X$, we

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

have:

$$x \leqslant \bigoplus_{x\in X} x$$
$$x^n \leqslant \left(\bigoplus_{x\in X} x\right)^n \qquad \text{(monotony of the } n\text{th power)}$$
$$\left(\bigoplus_{x\in X} x^n\right) \leqslant \left(\bigoplus_{x\in X} x\right)^n. \qquad \text{(idempotency)}$$

We now prove the reverse inequality. If $x \in X$, we have:

$$x^n \leqslant \bigoplus_{x\in X} x^n$$
$$x \leqslant \sqrt[n]{\bigoplus_{x\in X} x^n} \qquad \text{(monotony of the } n\text{th root)}$$
$$\left(\bigoplus_{x\in X} x\right) \leqslant \sqrt[n]{\bigoplus_{x\in X} x^n} \qquad \text{(idempotency)}$$
$$\left(\bigoplus_{x\in X} x\right)^n \leqslant \bigoplus_{x\in X} x^n. \qquad \text{(monotony of the } n\text{th power)}$$

Thus, $(\bigoplus_{x\in X} x)^n = \bigoplus_{x\in X} x^n$, which means that the nth power is $\oplus$-lsc. □

For dioids of class (3), we also prove that the nth power is a $\oplus$-morphism (Dudnikov and Samborskii 1987; Dudnikov and Samborskii 1992).

Lemma 5. In dioids of class (3),

$$\forall n \in \mathbb{N},\ \forall a, b \in Q,\ (a \oplus b)^n = a^n \oplus b^n.$$

Proof. First note that if $a = \bot$, then $(a \oplus b)^n = b^n$, so the equality trivially holds if $a = \bot$ or $b = \bot$. Let us now assume that $a \neq \bot$ and $b \neq \bot$ and proceed by induction on n. The result holds trivially for $n = 0$ and $n = 1$, so we assume it for $n \geqslant 1$, and prove it at rank $n + 1$. We have

$$\begin{aligned}(a \oplus b)^{n+1} \otimes (a \oplus b) &= ((a \oplus b)^n \otimes (a \oplus b)) \otimes (a \oplus b)\\ &= ((a^n \oplus b^n) \otimes (a \oplus b)) \otimes (a \oplus b) \qquad \text{(induction hypothesis)}\\ &= (a^{n+1} \oplus ab^n \oplus a^n b \oplus b^{n+1}) \otimes (a \oplus b)\\ &= a^{n+2} \oplus ab^{n+1} \oplus a^{n+1}b \oplus b^{n+2}\\ &\quad \oplus a^2 b^n \oplus a^n b^2 \qquad (*)\end{aligned}$$

$$(a^{n+1} \oplus b^{n+1}) \otimes (a \oplus b) = a^{n+2} \oplus ab^{n+1} \oplus a^{n+1}b \oplus b^{n+2}. \qquad (**)$$

Now we just have to prove that the terms $(*)$ and $(**)$ are equal, which would mean we would indeed have $(a \oplus b)^{n+1} \otimes (a \oplus b) = (a^{n+1} \oplus b^{n+1}) \otimes (a \oplus b)$. As $a \oplus b \neq \bot$, the cancellation hypothesis allows us to simplify by cancelling $(a \oplus b)$, which establishes

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

the property at rank $n+1$. We will now show that (*) and (**) are equal. We have $a^2b^n \oplus a^n b^2 = ab(ab^{n-1} \oplus a^{n-1}b)$. Moreover, the induction hypothesis gives

$$a^n \oplus b^n = (a \oplus b)^n = ab^{n-1} \oplus a^{n-1}b \oplus \left(a^n \oplus b^n \oplus \left(\oplus_{k=2}^{n-2} a^k b^{n-k}\right)\right),$$

which in turn yields

$$ab^{n-1} \oplus a^{n-1}b \leqslant a^n \oplus b^n.$$

Since $\otimes$ preserves the order, we get the required result. □

We now prove that dioids of class (3) are equipped with an nth root (Dudnikov and Samborskii 1987; Dudnikov and Samborskii 1992).

Lemma 6. In dioids of class (3), if the equation $X^n = q$ has a solution, then this solution is unique.

Proof. First note that if $q = \bot$, the above equation becomes $x^n = \bot$. The unique solution of this equation is $\lambda = \bot$. This is because $\lambda = \bot$ is clearly a solution, and if $\lambda \neq \bot$ were also a solution, then $\lambda^n = \bot = \bot \otimes \lambda$, which would prove by cancellation that $\lambda^{n-1} = \bot$, and, by iterating this process, that $\lambda = \bot$, which contradicts the assumption.

So we now assume that $q \neq \bot$. If λ_1 and λ_2 are two solutions of the equation $X^n = q$, we have $\lambda_1^n = \lambda_2^n = q$ and $\lambda_1 \neq \bot$, $\lambda_2 \neq \bot$. As the nth power is a $\oplus$-morphism (Lemma 5), the result of Lemma 2 holds true:

$$\lambda_1^{n-1} \otimes \lambda_2 = \lambda_1 \otimes \cdots \otimes \lambda_1 \otimes \lambda_2 \leqslant \lambda_1^n \oplus \lambda_2^n = q \oplus q = q = \lambda_1^n.$$

So

$$\lambda_1^{n-1} \otimes \lambda_2 \leqslant \lambda_1^n.$$

Thus $\lambda_2 \leqslant \lambda_1$ by successive cancellations by λ_1. The same reasoning can be applied to prove $\lambda_1 \leqslant \lambda_2$. So $\lambda_1 = \lambda_2$ and the equation $X^n = q$ has a unique solution. □

Finally, we can put all these results together to prove Proposition 2.

Proof of Proposition 2. In a double-idempotent dioid, the nth power is the identity function. Thus, the nth power is $\oplus$-lsc, which proves that dioids of class (2) are cost dioids.

For selective dioids, $(a \oplus b)^n = a^n$ or b^n, so the nth power is a $\oplus$-morphism. Thus, according to Lemma 4, dioids of class (1) are cost dioids.

As dioids of class (3) are equipped with an nth root, Lemmas 4 and 5 allow us to conclude that the nth power is $\oplus$-lsc. Dioids of class (3) are thus cost dioids. □

For instance, $(\overline{\mathbb{R}}, \mathtt{max}, +)$ is a cost dioid that may be used to define the Worst Case Execution Time: when two states can be joined by several sequences of transitions that cost different times, the worst time is taken. To compute the cost of a sequence of transitions, we sum the costs of each transition. Another example of a cost dioid is $(\mathcal{P}(S), \cap, \cup)$ (S being finite of infinite), where the cost to reach a state gives information about which elements of S must have been seen before. It can be used to guarantee the execution of a security check or record which parts of a code have been executed. Table 1 gives a non-exhaustive list of cost dioids.

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

2.2. *Semantics as linear operators over dioids*

The upshot of using the adequate cost dioid is that the cost computation can be defined in terms of matrix operations in this dioid. The set of one-step transitions can be equivalently represented by a *transition matrix* $M \in \mathscr{M}_{\Sigma\times\Sigma}(Q)$ with

$$M_{\sigma,\sigma'} = \begin{cases} q & \text{if } \sigma \to^q \sigma' \\ \bot & \text{otherwise.} \end{cases}$$

Here, $\mathscr{M}_{\Sigma\times\Sigma}(Q)$ stands for the set of matrices with rows and columns indexed over Σ and values in Q. This set of matrices is naturally equipped with two operators $\oplus$ and $\otimes$ in the classical way: operator $\oplus$ is extended pointwise, and operator $\otimes$ corresponds to the matrix product (note that the iterate M^n embeds the costs for paths of length n). Recall that the dioid is complete, ensuring convergence of the sum for each coefficient of the product matrix. The resulting structure is also an idempotent and complete dioid. The order induced by $\oplus$ corresponds to the pointwise extension of the order over Q: $M \leqslant M' \Leftrightarrow \forall i, j. M_{i,j} \leqslant M'_{i,j}$. A transition matrix may also be viewed as a linear operator on the moduloid $Q(\Sigma)$, as defined below.

Definition 8. Let $(E, \oplus, \otimes)$ be a commutative dioid. A moduloid over E is a set V with an internal operation $\oplus$ and an external operation $\odot$ such that:

(1) $(V, \oplus)$ is a commutative monoid, with 0 as neutral element.
(2) The $\odot$ operator maps $E \times V$ on V and satisfies the following axioms:
   (a) $\forall \lambda \in E, \forall (x, y) \in V^2, \lambda \odot (x \oplus y) = (\lambda \odot x) \oplus (\lambda \odot y)$.
   (b) $\forall (\lambda, \mu) \in E^2, \forall x \in V, (\lambda \oplus \mu) \odot x = (\lambda \odot x) \oplus (\mu \odot x)$.
   (c) $\forall (\lambda, \mu) \in E^2, \forall x \in V, \lambda \odot (\mu \odot x) = (\lambda \otimes \mu) \odot x$.
   (d) $\forall x \in V, e \odot x = x$ and $\bot \odot x = 0$.
   (e) $\forall \lambda \in E, \lambda \odot 0 = 0$.

If E is an idempotent dioid, then for any moduloid V over E the addition operator $\oplus$ defined pointwise is also idempotent, and thus defines a canonical order. As for vector spaces, if n is a given integer, E^n, the set of vectors with n components in E, is a moduloid. More generally, a vector $u \in E(\Sigma)$, with Σ finite, $|\Sigma| = n$, can be seen as a function $\delta_u : [1, n] \to E$. Since Q is complete, we can generalise to the infinite (countable) case: δ_u becomes a mapping from $\mathbb{N}$ to E, and the same technique applies for matrices. The matrix–vector product is defined by

$$(Mu)_i = \bigoplus_{j=1}^{+\infty} \delta_M(i, j) \otimes \delta_u(j).$$

In this paper, we will continue to use the matrix notation for the sake of simplicity, even when there is an infinite set of indices.

3. Abstraction

The transition matrix representing a program is in general of infinite dimension, so neither transitive closure nor traces can be computed in finite time. To overcome this problem, we define an abstract matrix that can be used to approximate the computations of the original

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

matrix. For example, if we compute the minimum memory needed to run a program, a correct approximation of this quantity must be greater than the effective minimum. In this section we give a sufficient condition for this approximation to be correct with respect to the ordering induced by the dioid. To prove the correctness of an abstraction, we re-state the classical abstract interpretation theory (Cousot and Cousot 1977) in terms of linear operators over moduloids.

3.1. *Galois connections and pseudo-inverses*

We first briefly recall the definition of Galois connections used in classical abstract interpretation theory.

Definition 9. Let $(C, \leqslant_C)$ and $(D, \leqslant_D)$ be two partially ordered sets. Two mappings $\alpha : C \mapsto D$ (called the abstraction function) and $\gamma : D \mapsto C$ (called the concretisation function) form a Galois connection (C, α, γ, D) if and only if:

— $\forall c \in C, \forall d \in D, c \leqslant_C \gamma(d) \iff \alpha(c) \leqslant_D d$, or, equivalently,
— α and γ are monotonic and $\alpha \circ \gamma \leqslant Id_D$ and $Id_C \leqslant \gamma \circ \alpha$.

In our setting, the partial orders will be the orders induced by the $\oplus$ operators over vectors in a moduloid. The question that naturally arises is whether, given an abstraction α, a concretisation function exists. Di Pierro and Wiklicky (2000) describes the framework of Probabilistic Abstract Interpretation over the semiring of probabilities. In their framework, the abstraction function A is a bounded linear map between Hilbert spaces. They obtain a concretisation function using the Moore–Penrose pseudo-inverse of A, that is, the (unique) mapping G such that:

— $AGA = A$
— $GAG = G$
— $(AG)^* = AG$
— $(GA)^* = GA$,

where the star operator denotes the conjugate (see Di Pierro *et al.* (2005b) for details). As we will not be able to define an exact inverse in the general case, nor apply the Moore–Penrose pseudo-inverse since we do not work in a field, we will use the theory of *residuation* to get a kind of inverse for α. We thus consider the following proposition.

Proposition 3 (Gondran and Minoux 2008). Let E and F be two sets equipped with a complete partial order, f a monotone mapping from E to F. We call an element y such that $f(y) \leqslant b$ a subsolution of the equation $f(x) = b$. The following properties are equivalent:

(1) For all $b \in E$, there exists a greatest subsolution to the equation $f(x) = b$.
(2) $f(\bot_E) = \bot_F$, and f is $\oplus$-lsc.
(3) There exists a monotone mapping $f^\dagger : F \to E$ that is upper[†] semi-continuous such that $f \circ f^\dagger \leqslant Id_F$ and $Id_E \leqslant f^\dagger \circ f$.

Consequently, $f^\dagger$ is unique. When f satisfies these properties, it is said to be residuated, and $f^\dagger$ is called its residual.

[†] Upper semi-continuity is the analogue of lower semi-continuity for the $\wedge$ operator.

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

In our framework, the complete orders are the moduloid orders defined pointwise from the cost dioid order. This application of residuation to the particular setting of dioids is the key to defining an analogue of Galois connections for our framework, as described in the rest of Section 3.

3.2. *Abstraction over cost dioids*

We now show how the notions of abstraction and concretisation can be recast in our setting. In the following, Σ will denote a set of *concrete* states and $\Sigma^\sharp$ a set of *abstract* states. An abstraction function maps concrete states in Σ to their abstraction in $\Sigma^\sharp$. Given an abstraction function α, we can lift it to a linear abstraction operator $\alpha^\uparrow \in \mathscr{M}_{\Sigma^\sharp \times \Sigma}(Q)$ by setting (recall that e denotes the neutral element for $\otimes$)

$$\alpha^\uparrow_{\sigma^\sharp,\sigma} = \begin{cases} e & \text{if } \alpha(\sigma) = \sigma^\sharp \\ \bot & \text{otherwise.} \end{cases}$$

In the following, we will write $\leqslant$ for the order defined on $\mathscr{M}_{\Sigma\times\Sigma}(Q)$ or $\mathscr{M}_{\Sigma^\sharp\times\Sigma^\sharp}(Q)$ in Section 2.2. Recall that this order is the pointwise extension of the order over Q, and that we do not assume an order on either concrete or abstract states. The pointwise orders defined on moduloids constructed over a complete dioid are also complete. We thus get the following theorem.

Theorem 1. Let Σ and $\Sigma^\sharp$ be the domains of concrete and abstract states, α be a mapping from Σ to $\Sigma^\sharp$, and $\alpha^\uparrow \in \mathscr{M}_{\Sigma^\sharp\times\Sigma}(Q)$ be the linear mapping obtained by lifting α. There exists a unique monotonic $\gamma^\uparrow$ such that

$$\alpha^\uparrow \circ \gamma^\uparrow \leqslant Id_{\Sigma^\sharp} \quad \text{and} \quad Id_\Sigma \leqslant \gamma^\uparrow \circ \alpha^\uparrow$$

where Id_Σ and $Id_{\Sigma^\sharp}$ denote the identity matrices in $\mathscr{M}_{\Sigma\times\Sigma}(Q)$ and $\mathscr{M}_{\Sigma^\sharp\times\Sigma^\sharp}(Q)$, respectively.

Proof. As the abstraction function is linear, it trivially fulfills requirements (2) of Proposition 3, and we get the result by taking $\gamma^\uparrow = (\alpha^\uparrow)^\dagger$. □

In our settings, the simplicity of $\alpha^\uparrow$ gives rise to a very simple expression of $\gamma^\uparrow$. Indeed, $\gamma^\uparrow$ is just the transpose matrix of $\alpha^\uparrow$.

3.3. *Induced abstract semantics*

Let T be a transition system in the concrete domain Σ over the cost dioid $(Q, \oplus, \otimes)$. We now want to define an abstract transition system over the abstract domain $\Sigma^\sharp$ that is 'compatible' with T with regard to both its traces and the costs it will be lead to compute. The following definition of a correct abstraction will ensure that the long-run cost of a program, as defined in the next section, will be correctly over-approximated during the abstraction process.

Definition 10 (correct abstraction). Let $T = \langle M, I\rangle$ be a transition system over the concrete domain with $M \in \mathscr{M}_{\Sigma\times\Sigma}(Q)$ and $I \subseteq \Sigma$. Let $T^\sharp = \langle M^\sharp, I^\sharp\rangle$ be a transition system over the abstract domain with $M^\sharp \in \mathscr{M}_{\Sigma^\sharp\times\Sigma^\sharp}(Q)$ and $I^\sharp \subseteq \Sigma^\sharp$. Let α be an abstraction from Σ

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

to $\Sigma^\sharp$. The triple $(T, T^\sharp, \alpha)$ is a correct abstraction from Σ to $\Sigma^\sharp$ if $\alpha^\uparrow \circ M \leqslant M^\sharp \circ \alpha^\uparrow$ and $\{\alpha(\sigma) \mid \sigma \in I\} \subseteq I^\sharp$.

The classical framework of abstract interpretation gives a way to define a best correct abstraction for a given concrete semantic operator. In the same way, given an abstraction α and a concrete semantics linear operator, we can define an abstract semantics operator that is correct by construction, as expressed by the following proposition.

Proposition 4. Let α be an abstraction from Σ to $\Sigma^\sharp$ and $T = \langle M, I\rangle$ be a transition system with $M \in \mathscr{M}_{\Sigma\times\Sigma}(Q)$ a linear operator over the concrete moduloid and I the subset of initial states. We set $T^\sharp = \langle M^\sharp, I^\sharp\rangle$ with

$$M^\sharp = \alpha^\uparrow \circ M \circ \gamma^\uparrow \quad \text{and} \quad I^\sharp = \{\alpha(\sigma) \mid \sigma \in I\}.$$

Then $(T, T^\sharp, \alpha)$ is a correct abstraction from Σ to $\Sigma^\sharp$. Moreover, given T and α, $T^\sharp$ provides the best possible abstraction in the sense that if $(T, \langle M', I'\rangle, \alpha)$ is another correct abstraction, then

$$M^\sharp \leqslant M' \quad \text{and} \quad I^\sharp \subseteq I'.$$

Proof. The statement follows from the fact that $Id \leqslant \gamma^\uparrow \circ \alpha^\uparrow$ and $\alpha^\uparrow \circ \gamma^\uparrow \leqslant Id$, since $\gamma^\uparrow$ is defined as $(\alpha^\uparrow)^\dagger$. □

This best choice of $M^\sharp$ turns the inequality of Definition 10 into an equality. This kind of property is usually represented by a diagram like the one below, which expresses the fact that the abstraction and transfer functions commute.

$$\begin{array}{ccc} Q(\Sigma) & \xrightarrow{\alpha^\uparrow} & Q(\Sigma^\sharp) \\ \downarrow{\scriptstyle M} & & \downarrow{\scriptstyle M^\sharp} \\ Q(\Sigma) & \xrightarrow{\alpha^\uparrow} & Q(\Sigma^\sharp) \end{array}$$

The above definitions and properties deal with the matrix view of the semantics, but what can we say about traces? The following proposition states that for each program trace, there exists an 'abstract' trace of the same length for which the costs are given by the induced abstract matrix. This property will be useful for proving the correctness of abstractions in Section 4.

Proposition 5. Consider the transition system $T = \langle q, I\rangle$ where $I \subseteq \Sigma$ is its set of initial states and $q : \Sigma \times \Sigma \to Q$ is its quantitative transition system in the cost dioid Q. Let α be an abstraction function from Σ to $\Sigma^\sharp$. Let $T^\sharp = \langle q^\sharp, I^\sharp\rangle$ an abstract transition system defined by:

— $I^\sharp = \{\alpha(\sigma) \mid \sigma \in I\}$
— $\alpha^{-1} : \Sigma^\sharp \to \mathscr{P}(\Sigma)$ with $\alpha^{-1}(\sigma^\sharp) = \{\sigma \mid \alpha(\sigma) = \sigma^\sharp\}$
— $q^\bullet(\Sigma_1, \Sigma_2) = \bigoplus_{(\sigma_1,\sigma_2)\in\Sigma_1\times\Sigma_2} q(\sigma_1, \sigma_2)$
— $q^\sharp(\sigma_1^\sharp, \sigma_2^\sharp) = q^\bullet(\alpha^{-1}(\sigma_1^\sharp), \alpha^{-1}(\sigma_2^\sharp))$.

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

Then for all $t = \sigma_0 \to^{q_0} \dots \sigma_n \in [\![T]\!]_{tr}, |t| = n$, there exists $t^\sharp = \sigma_0^\sharp \to^{q_0'} \dots \sigma_n^\sharp \in [\![T^\sharp]\!]_{tr}, |t| = n$ such that $q_i \leqslant q_i' \quad \forall i \in [0, n-1]$ and $\sigma_i^\sharp = \alpha(\sigma_i) \quad \forall i \in [0, n]$. In addition, $M^\sharp = \alpha^\uparrow \circ M \circ \alpha^\dagger$ is the transition matrix for $q^\sharp$.

Proof. Let t be $\sigma_0 \to^{q_0} \dots \sigma_n$. $t \in [\![\langle q, I\rangle]\!]_{tr}$. We choose $t^\sharp = \sigma_0^\sharp \to^{q_0'} \dots \sigma_n^\sharp$ with $\sigma_i^\sharp = \alpha(\sigma_i), \forall i \in [0, n]$ and $q_i' = q^\sharp(\sigma_i^\sharp, \sigma_{i+1}^\sharp), \forall i \in [0, n-1]$. Trivially, we have $|t| = |t^\sharp|$. We also have $q_i' = q^\sharp(\alpha(\sigma_i), \alpha(\sigma_{i+1})) = q^\bullet(\alpha^{-1} \circ \alpha(\sigma_i), \alpha^{-1} \circ \alpha(\sigma_{i+1}))$. By the definition of α^{-1}, we have $x \subseteq \alpha^{-1} \circ \alpha(x)$, so

$$\begin{aligned} q_i' &= q(\sigma_i, \sigma_{i+1}) \oplus \bigoplus_{\substack{(\sigma_1,\sigma_2) \in (\alpha^{-1}\circ\alpha(\sigma_i)) \times \alpha^{-1}\circ\alpha(\sigma_{i+1})) \\ \setminus\{(\sigma_i,\sigma_{i+1})\}}} q(\sigma_1, \sigma_2) \\ &\geqslant_Q q(\sigma_i, \sigma_{i+1}) \\ &= q_i. \end{aligned}$$

Eventually, we prove that $t^\sharp \in [\![\langle q^\sharp, I^\sharp\rangle]\!]_{tr}$. $\sigma_0^\sharp \in I^\sharp$ because $\sigma_0 \in I$. For all $i \in [0, n-1]$, we have $q^\sharp(\sigma_i^\sharp, \sigma_{i+1}^\sharp) = q_i' \geqslant_Q q_i >_Q \bot_Q$.

In addition, a development of the matrix multiplication $M^\sharp = \alpha^\uparrow \circ M \circ \alpha^\dagger$ for a location $(\sigma_1^\sharp, \sigma_2^\sharp)$ leads us to its equality with $q^\sharp(\sigma_1^\sharp, \sigma_2^\sharp)$. □

3.4. *Link with the classical abstract interpretation*

In Section 3.2 we showed how to lift any abstraction function $\alpha : \Sigma \to \Sigma^\sharp$ into a linear mapping $\alpha^\uparrow \in \mathscr{M}_{\Sigma^\sharp \times \Sigma}(Q)$, where domains Σ and $\Sigma^\sharp$ are not assumed to have a particular structure. In order to benefit from the already existing abstractions provided by the classical abstract interpretation theory, we will now show how to translate them into our model. As abstract interpretation relies on lattices and Galois connections, in this section we will compare these structures and investigate how they are transposed to moduloids and linear operators.

Up to this point the way we have lifted an abstraction has represented a state σ of Σ by a vector of the form $(\bot, \dots, \bot, e, \bot, \dots, \bot)^T$ where e appears in the σ-place (recall Σ is countable). The set of concrete states Σ is thus represented by the moduloid $\Sigma^\uparrow = (\{\bot, e\}^{|\Sigma|}, \oplus, \otimes)$. If we now assume that Σ is a lattice, this lifting has two drawbacks: it obviously creates matrices of unnecessarily huge dimension, and it forgets about the ordered structure of Σ. The latter is unfortunate since $\Sigma^\uparrow$ naturally has an ordered structure given by the $\oplus$-law. Thus, we are naturally led to ask how we can translate Σ and $\Sigma^\sharp$ into moduloids while preserving their respective lattice orders. This order-preservation property will be referred to as the lift-order property in the remainder of this section.

3.4.1. *Lifting a Galois connection into a linear mapping.* Abstract interpretation often considers Galois connections $B \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ where B is a powerset[†] representing the concrete semantic domain and A is a complete lattice representing the abstract domain. In order to

[†] Powersets are naturally equipped with a particular complete lattice structure called a boolean lattice (Davey and Priestley 1990).

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

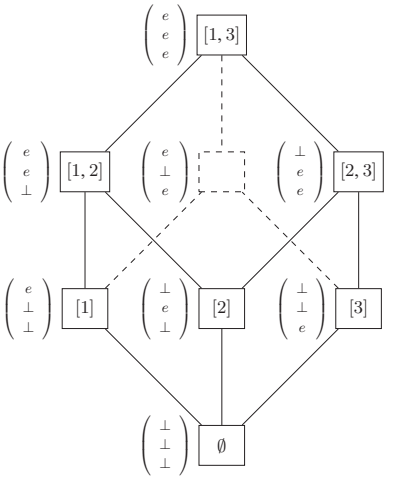


Fig. 1. An example of a set lattice (interval lattice on the set $\{1, 2, 3\}$) and its associated powerset $(\mathscr{P}(\{1, 2, 3\}), \cup)$

lift α into a linear mapping, we will focus on how to lift-order these particular structures. Naturally, the easy case is that of boolean lattices.

Lift-ordering within boolean lattices. A boolean lattice B is generated by its set of atoms $\mathscr{A}(B)$, which correspond to the singletons in the case of a powerset. Indeed, for each $b \in B$, $b = \vee\{a \in \mathscr{A}(B) \mid a \leqslant b\}$ (Davey and Priestley 1990). We thus choose to code the atoms a as vectors $a^{\uparrow}$ in $\{\bot, e\}^{|\mathscr{A}(B)|}$ in the same way as earlier, and the coding of the other elements will then follow from the use of $\oplus$:

$$b^{\uparrow} = \oplus\{a^{\uparrow} \mid a \leqslant b\}.$$

We use $B^{\uparrow}$ to denote the complete moduloid constructed in this way from B, where the $\oplus$ operator of $B^{\uparrow}$ matches the $\cup$ operator of B by construction.

Now that we have expressed boolean lattices as moduloids, we can easily lift-order the abstraction function of a Galois connection $B_1 \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} B_2$, where B_1 and B_2 are boolean lattices. Lift-ordering these lattices, we obtain two moduloids $(B_1^{\uparrow}, \oplus_1, \otimes_1)$ and $(B_2^{\uparrow}, \oplus_2, \otimes_2)$. Since the $\cup_i$ and $\oplus_i$ operators coincide, and as α is a union morphism, its linear translation $\alpha^{\uparrow}$ is defined by its values on the base vectors of $B_1^{\uparrow}$, that is, the vector codings of atoms of B_1.

$$\begin{array}{ccccccc} \alpha(\{b_1\} & \cup_1 & \{b_2\}) & = & \alpha(\{b_1\}) & \cup_2 & \alpha(\{b_2\}) \\ & \updownarrow & & & & \updownarrow & \\ \alpha^{\uparrow}(b_1^{\uparrow} & \oplus_1 & b_2^{\uparrow}) & = & \alpha^{\uparrow}(b_1^{\uparrow}) & \oplus_2 & \alpha^{\uparrow}(b_2^{\uparrow}) \end{array}$$

The general case. In most cases, A is not a powerset but a more general complete lattice, for which vectorial translation is not so straightforward. The representation theorem of finite distributive lattices (Davey and Priestley 1990) asserts that any lattice A verifying

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

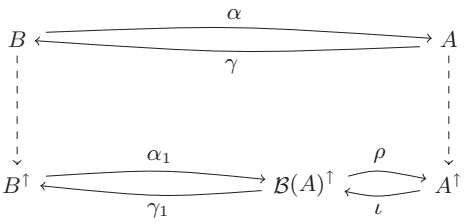


Fig. 2. A Galois connection and its lift

these properties is isomorphic to a lattice of sets. Thus, A can be seen as a sublattice of a given powerset, which we will denote by $\mathscr{B}(A)$. The previous coding applies to $\mathscr{B}(A)$ and *a fortiori* to A. However, the set of vectors $A^{\uparrow}$ constructed in this way no longer has the structure of a complete moduloid, unlike $\mathscr{B}(A)^{\uparrow}$.

Once we have lifted the lattices, we might want to express abstractions as linear operators. We thus have to define $\alpha^{\uparrow}$ on the basis of our moduloid. The problem now is that there is no match between the $\oplus$ law and $\cup$, the join law of the lattice. For instance, $[1] \cup [3] = [1,3]$ and $[1]^{\uparrow} \oplus [3]^{\uparrow} = (e, \bot, e)^T$ and $[1,3]^{\uparrow} = (e,e,e)^T$. This makes it impossible to express $\alpha^{\uparrow}$ as a linear mapping, since, for instance, $\alpha^{\uparrow}(\{1\}^{\uparrow} \oplus \{3\}^{\uparrow}) = (e,e,e)^T \neq \alpha^{\uparrow}(\{1\}^{\uparrow}) \oplus \alpha^{\uparrow}(\{3\}^{\uparrow}) = (e, \bot, e)^T$.

We thus have to weaken our requirement: in the following, we choose to lift-order Galois connections into non-linear, but still residuable, mappings.

3.4.2. *Lifting a Galois connection into a residuable mapping.* Since $\mathscr{B}(A)^{\uparrow}$ is a complete boolean lattice, we will decompose $\alpha^{\uparrow}$ into a linear part from $B^{\uparrow}$ to $\mathscr{B}(A)^{\uparrow}$, and a projection from $\mathscr{B}(A)^{\uparrow}$ into its sublattice $A^{\uparrow}$ that we are interested in, representing the vector encoding of A (see Figure 2).

The linear part of $\alpha^{\uparrow}$, denoted by α_1, is defined as in the case of a connection between two boolean lattices: α_1 is defined on the set of atoms of B by $\alpha_1(b^{\uparrow}) = \alpha(b)^{\uparrow}$ where b is an atom of B, and then extended to $B^{\uparrow}$ by linearity.

This linear mapping is then composed with a projection ρ in order to yield a vector in $A^{\uparrow}$ corresponding to an element of the (non-boolean) lattice A. As we want to maintain the lift-order property for all $x \in \mathscr{B}(A)$, we define $\rho(x)$ as the smallest element $z \in A^{\uparrow}$ such that $z \geqslant x^{\dagger}$. Note that ρ defined in this way is an upper closure operator in $\mathscr{B}(A)$.

As α_1 is a linear mapping between two complete moduloids, by Proposition 3 there exists an usc pseudo inverse γ_1 for α_1, that is, $\alpha_1 \circ \gamma_1 \leqslant Id_{\mathscr{B}(A)^{\uparrow}}$ and $\gamma_1 \circ \alpha_1 \geqslant Id_{B^{\uparrow}}$. Passing from $A^{\uparrow}$ to $\mathscr{B}(A)^{\uparrow}$ is simply done by a canonical injection ι.

We finally prove the following property, which allows us to define a pseudo-invertible lift of our initial Galois connection.

Proposition 6. Mappings $\rho \circ \alpha_1$ and $\gamma_1 \circ \iota$ as defined above are pseudo-inverse, and thus form a Galois connection between moduloids (seen as lattices) $B^{\uparrow}$ and $A^{\uparrow}$.

Proof. We first note that $\rho \circ \alpha_1$ and $\gamma_1 \circ \iota$ are monotonic by composition of monotonic mappings. We then show that $(\gamma_1 \circ \iota) \circ (\rho \circ \alpha_1) \geqslant Id_{B^{\uparrow}}$: for all $a \in B^{\uparrow}$, we have $\rho(\alpha_1(a)) \geqslant \alpha_1(a)$

† The complete nature of A and the equivalence between the order on A and on its lift-version $A^{\uparrow}$ ensure the existence of this element.

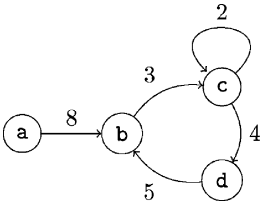
https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

because ρ is extensive. As γ_1 is monotonic and the pseudo-inverse of α_1, we have $\gamma_1 \circ \rho(\alpha_1(a)) \geqslant \gamma_1(\alpha_1(a)) \geqslant a$. Finally, we show that $(\rho \circ \alpha_1) \circ (\gamma_1 \circ \imath) \leqslant Id_{\mathscr{B}(A)^\uparrow}$: as $< \alpha_1, \gamma_1 >$ is a Galois connection, $\alpha_1 \circ \gamma_1 \circ \imath(x) = \alpha_1 \circ \gamma_1(x) \leqslant x$ for all $x \in A^\uparrow$. Applying the monotonic function ρ to each member of this inequality, we then get $\rho(\alpha_1 \circ \gamma_1(x)) \leqslant \rho(x)$. As $x \in A^\uparrow$, we have $\rho(x) = x$, which allows us to conclude the proof. □

4. Long-run cost

All single-transition costs can be summarised in a transition matrix, as we showed in Section 2. We now use this matrix and the mathematical results of dioid algebra to define a notion of the long-run cost for a whole program. In Sotin *et al.* (2006) we proposed a notion of the *global cost* of a program, representing its cost from initial to final states. It correctly deals with programs that are meant to terminate, but in some cases this global cost turns out to be $\top$, in particular when it is evaluated on a coarse abstraction of the initial system. Getting $\top$ as a result for the global cost is unsatisfactory as it does not tell us anything about the concrete cost. For this case, and for the case of programs that are not meant to terminate (such as reactive systems), we propose the notion of the *long-run cost*, which represents a maximal average cost over cycles of transitions. This terminology is taken from De Alfaro (1998) and Brazdil *et al.* (2005), where it was used in the context of probabilistic processes modelled by Markov decision processes. Real numbers are associated with the behaviour patterns of interest (described by labelled graphs) to represent the success or duration of the pattern, and extensions of branching-time temporal logics are proposed in order to measure their long-run average outcome.

The average cost of a finite path is defined as the arithmetical mean (with respect to the $\otimes$ operator) of the costs labelling its transitions. In other words, it is the nth root of the global cost of the path, where n is its length. We write $\tilde{q}(\pi) = \sqrt[|\pi|]{q(\pi)}$ for the average cost of path π, where $q(\pi)$ is the global cost of π, and $|\pi|$ its length. The 'maximum' average cost of all cycles in the graph will be the quantity we are interested in: this quantity will be called the *long-run cost*. The following example illustrates these notions on a simple graph:



Average cost of path abc $= (8 + 3)/2 = 5.5$.
Cycle bcdb average cost $= (3 + 4 + 5)/3 = 4$.
Cycle bccdb average cost $= 14/4 = 3.5$.
Cycle cc average cost $= 2/1 = 2$.
Long-run cost $= 4$.

From the properties of the dioids we consider, the matrix M^k sums up the transition costs of all paths of length k. The diagonal of this matrix thus contains the costs of all cycles of length k. If we add up all the elements on this diagonal, we get the trace of the matrix. This observation gives rise to the following definition.

Definition 11. Let $T = \langle M, I \rangle$ be a transition system over Σ. Let R be M restricted to the set of states Σ_I reachable from I. The long-run cost of T is defined as

$$\rho(T) = \bigoplus_{k=1}^{|\Sigma_I|} \sqrt[k]{tr\, R^k} \quad \text{where} \quad tr\, R = \bigoplus_{i=1}^{|\Sigma_I|} R_{i,i}.$$

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

Note that this definition is valid even for an infinite number of states since we work with complete dioids. As an example, if we work in the dioid $(\mathit{Time}, \max, +)$, where *Time* is isomorphic to $\overline{\mathbb{R}}$, $\rho(T)$ is the maximal average of time spent per instruction, where the average is computed on any cycle by dividing the total time spent in the cycle by the number of instructions in this cycle. In the case of a finite set of states, the long-run cost is computable, and we note in passing that its definition coincides with the definition of the maximum of the eigenvalues of the matrix in the case of an irreducible matrix in an idempotent semiring (Cochet-Terrasson *et al.* 1998).

4.1. *Semantics of the long run cost*

The following proposition establishes in a more formal manner the link between this definition of long-run cost and the cycles of the semantics.

Proposition 7. Let Γ be the set of cycles in T. Then $\rho(T) = \bigoplus_{c\in\Gamma} \tilde{q}(c)$.

To improve the readability of the proofs, we introduce the following notation: for any path or cycle π, we use $\tilde{q}(\pi)$ to stand for $\sqrt[|\pi|]{q(\pi)}$. To prove the proposition, we first establish Lemmas 7 and 8.

By definition, a cycle is a path that starts and finishes in the same state, and contains at least one transition. As C is defined with respect to the trace semantics, it only contains reachable states that are all included in Σ, the countable set of reachable states of the semantics. Let $\Gamma_{\leqslant|\Sigma|}$ be the set of cycles whose length is less than or equal to $|\Sigma|$.

Lemma 7.

$$\forall c \in \Gamma, \exists \Gamma_c \subseteq \Gamma_{\leqslant|\Sigma|}, \quad \tilde{q}(c) \leqslant \bigoplus_{c_e\in\Gamma_c} \tilde{q}(c_e)$$

Proof. We use strong induction on the length of c:

— If $|c| \leqslant |\Sigma|$ (this holds in particular when Σ is infinite), then $\Gamma_c = \{c\}$ and the inequality holds trivially.

— If $|c| > |\Sigma|$, there exists a state σ' such that

$$c = \sigma \overset{\pi_1}{\Rightarrow} \sigma' \overset{\pi_2}{\Rightarrow} \sigma' \overset{\pi_3}{\Rightarrow} \sigma \quad \text{with} \begin{cases} \pi_1\pi_3 \in \Gamma \ \wedge \ |\pi_1\pi_3| < |c| \\ \pi_2 \in \Gamma \quad \wedge \ |\pi_2| < |c| \,. \end{cases}$$

With this notation, we have

$$\begin{aligned} \tilde{q}(c) &= \tilde{q}(\pi_1\pi_2\pi_3) \\ &= \sqrt[|c|]{q(\pi_1\pi_2\pi_3)} \\ &= \sqrt[|c|]{q(\pi_1\pi_3) \otimes q(\pi_2)} && (1) \\ &\leqslant \sqrt[|\pi_1\pi_3|]{q(\pi_1\pi_3)} \oplus \sqrt[|\pi_2|]{q(\pi_2)} && (2) \\ &= \tilde{q}(\pi_1\pi_3) \oplus \tilde{q}(\pi_2) \end{aligned}$$

Equality (1) is justified by the commutativity of $\otimes$ and the definition of the cost of a path with respect to its decomposition. Inequality (2) follows from Property (2) of

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

Proposition 1. The property we want to prove then holds by induction, with

$$\Gamma_c = \Gamma_{\pi_1\pi_3} \cup \Gamma_{\pi_2}.$$

□

Lemma 8.

$$\bigoplus_{c\in\Gamma} \tilde{q}(c) = \bigoplus_{c_e\in\Gamma_{\leqslant|\Sigma|}} \tilde{q}(c_e).$$

Proof. We add some elements to the right-hand side of the inequality of Lemma 7, complementing the sum up to $\Gamma_{\leqslant|\Sigma|}$ (recall that $a \leqslant b \Rightarrow a \leqslant b \oplus c$). Hence

$$\forall c \in \Gamma, \tilde{q}(c) \leqslant \bigoplus_{c_e\in\Gamma_c} \tilde{q}(c_e) \leqslant \bigoplus_{c_e\in\Gamma_{\leqslant|\Sigma|}} \tilde{q}(c_e).$$

Summing for all $c \in \Gamma$ (recall the idempotency of $\oplus$), we get that the maximal average transition cost for all cycles is less than or equal to the maximal average transition cost of the bounded subset of cycles that are no longer than $|\Sigma|$:

$$\bigoplus_{c\in\Gamma} \tilde{q}(c) \leqslant \bigoplus_{c_e\in\Gamma_{\leqslant|\Sigma|}} \tilde{q}(c_e). \tag{3}$$

As $\Gamma_{\leqslant|\Sigma|} \subseteq \Gamma$, the opposite inequality is also true, and we trivially have

$$\bigoplus_{c_e\in\Gamma_{\leqslant|\Sigma|}} \tilde{q}(c_e) \leqslant \bigoplus_{c\in\Gamma} \tilde{q}(c) \tag{4}$$

Combining Inequalities (3) and (4) proves the lemma. □

We can now prove Proposition 7 itself.

Proof of Proposition 7. $\Gamma_{\leqslant|\Sigma|}$ can be partitioned into sets of cycles having same length (n) and the same state as the first vertex of a cycle (σ). Note that $n \geqslant 1$, and that σ is reachable. We write $\mathscr{C}_n^\sigma$ for such a set. Then

$$\Gamma_{\leqslant|\Sigma|} = \bigcup_{\substack{n\leqslant|\Sigma|\\ \sigma\in\Sigma}} \mathscr{C}_n^\sigma.$$

According to this equality and Lemma 8, we have

$$\begin{aligned}\bigoplus_{c\in\Gamma_{\leqslant|\Sigma|}} \tilde{q}(c) &= \bigoplus_{c\in\Gamma_{\leqslant|\Sigma|}} \sqrt[|c|]{q(c)}\\ &= \bigoplus_{n\leqslant|\Sigma|} \bigoplus_{\sigma\in\Sigma} \bigoplus_{c\in\mathscr{C}_n^\sigma} \sqrt[n]{q(c)}\\ &= \bigoplus_{n\leqslant|\Sigma|} \sqrt[n]{\bigoplus_{\sigma\in\Sigma} \bigoplus_{c\in\mathscr{C}_n^\sigma} q(c)}.\end{aligned}$$

This step is allowed by Property (1) of Proposition 1.

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

If σ is reachable, then all states in the path starting from σ are reachable. Let R be the matrix M with indices restricted to Σ, the reachable states of T. Finally, we have

$$\begin{aligned} R^n_{\sigma,\sigma} &= \bigoplus_{c\in\mathscr{C}^\sigma_n} q(c) \\ \bigoplus_{c\in\Gamma} \sqrt[|c|]{q(c)} &= \bigoplus_{n\leqslant|\Sigma|} \sqrt[n]{\bigoplus_{\sigma\in\Sigma} R^n_{\sigma,\sigma}} \\ &= \bigoplus_{n\leqslant|\Sigma|} \sqrt[n]{tr\, R^n} \\ &= \rho(T). \end{aligned}$$

□

As we aim to give a characterisation of the asymptotic behaviour of a program, an alternative definition for long-run cost could have been

$$lrc(T) = \limsup_{n\to\infty} \bigoplus_{\substack{t\in[\![T]\!]_{tr}\\ |t|=n}} \tilde{q}(t).$$

Instead of defining the long-run cost with respect to the cycles, this definition considers arbitrarily long traces. Unlike $\rho(T)$ however, $lrc(T)$ is not suitable for computation, even if the set of states is finite. We will see in Subsection 4.3 that these two notions coincide in a restricted class of cost dioids and when the set of states is finite.

4.2. *Ensuring correctness*

The question that naturally arises is whether the notion of long-run cost is preserved by abstraction. The following theorem states that a correct abstraction gives an over-approximation of the concrete long-run cost.

Theorem 2. If $(T, T^\sharp, \alpha)$ is a correct abstraction, then $\rho(T) \leqslant_Q \rho(T^\sharp)$.

To prove this, we first prove the following two lemmas.

Lemma 9. If $\alpha^\uparrow \circ M \leqslant M^\sharp \circ \alpha^\uparrow$, then $\alpha^\uparrow \circ M^n \leqslant (M^\sharp)^n \circ \alpha^\uparrow$.

Proof. Another way of stating this lemma is that $\forall n \geqslant 1$,

$$\alpha \circ M \leqslant M^\sharp \circ \alpha \Rightarrow \alpha \circ M^n \leqslant_Q (M^\sharp)^n \circ \alpha \tag{5}$$

We prove (5) by induction on n. The case where $n = 1$ is trivial. We then assume that $\alpha \circ M \leqslant M^\sharp \circ \alpha$ and $\alpha \circ M^n \leqslant (M^\sharp)^n \circ \alpha$. Lemma 1 gives

$$(\alpha \circ M^n) \circ M \leqslant ((M^\sharp)^n \circ \alpha) \circ M.$$

We then have

$$\begin{aligned} \alpha \circ M^{n+1} = \alpha \circ (M^n \circ M) &\leqslant (M^\sharp)^n \circ (\alpha \circ M) && \text{(associativity of } \circ) \\ &\leqslant (M^\sharp)^n \circ (M^\sharp \circ \alpha) && \text{(hypothesis)} \\ &\leqslant (M^\sharp)^{n+1} \circ \alpha && \text{(associativity of } \circ) \end{aligned}$$

□

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

Lemma 10. For all correct abstractions $(\langle M, I\rangle, \langle M^\sharp, I^\sharp\rangle, \alpha)$, and α lifted as stated in Section 3, $\forall \sigma \in \Sigma, \sigma^\sharp \in \Sigma^\sharp$,

$$\alpha(\sigma) = \sigma^\sharp \Rightarrow M_{c,c} \leqslant_Q (M^\sharp)_{d,d}.$$

Proof. We have $\alpha^\uparrow \circ M \leqslant M^\sharp \circ \alpha^\uparrow$, so, in particular, $(\alpha^\uparrow \circ M)_{\sigma^\sharp,\sigma} \leqslant_Q (M^\sharp \circ \alpha^\uparrow)_{\sigma^\sharp,\sigma}$, which can be rewritten as

$$\bigoplus_{\sigma_i \in \Sigma} (\alpha_{\sigma^\sharp,\sigma_i} \otimes M_{\sigma_i,\sigma}) \leqslant_Q \bigoplus_{\sigma_i^\sharp \in \Sigma^\sharp} ((M^\sharp_{\sigma^\sharp,\sigma_i^\sharp}) \otimes \alpha_{\sigma_i^\sharp,\sigma}).$$

Decomposing both summations, this yields

$$\bigoplus_{\sigma_i \in \alpha^{-1}(\sigma^\sharp)} (\alpha_{\sigma^\sharp,\sigma_i} \otimes M_{\sigma_i,\sigma}) \oplus \bigoplus_{\sigma_i \notin \alpha^{-1}(\sigma^\sharp)} (\alpha_{\sigma^\sharp,\sigma_i} \otimes M_{\sigma_i,\sigma}) \leqslant_Q ((M^\sharp_{\sigma^\sharp,\sigma^\sharp}) \otimes \alpha_{\sigma^\sharp,\sigma}) \oplus \bigoplus_{\sigma_i^\sharp \neq \sigma^\sharp} ((M^\sharp_{\sigma^\sharp,\sigma_i^\sharp}) \otimes \alpha_{\sigma_i^\sharp,\sigma}).$$

Given the properties of α, which is lifted from a function (see Section 3), this simplifies into $\bigoplus_{\sigma_i \in \alpha^{-1}(\sigma^\sharp)} M_{\sigma_i,\sigma} \leqslant_Q M^\sharp_{\sigma^\sharp,\sigma^\sharp}$. As σ belongs to $\alpha^{-1}(\sigma^\sharp)$, we also have the inequality

$$M_{\sigma,\sigma} \leqslant_Q \bigoplus_{\sigma_i \in \alpha^{-1}(\sigma^\sharp)} M_{\sigma_i,\sigma} \leqslant_Q M^\sharp_{\sigma^\sharp,\sigma^\sharp}.$$

From this result and idempotency, we deduce that for any $\sigma^\sharp$,

$$\bigoplus_{\sigma \in \alpha^{-1}(\sigma^\sharp)} M_{\sigma,\sigma} \leqslant_Q M^\sharp_{\sigma^\sharp,\sigma^\sharp}.$$

Then, summing over $\sigma^\sharp$, we finally get

$$\bigoplus_{\sigma \in \Sigma} M_{\sigma,\sigma} \leqslant_Q \bigoplus_{\sigma^\sharp \in D} M^\sharp_{\sigma^\sharp,\sigma^\sharp}.$$

□

We now prove Theorem 2 itself.

Proof of Theorem 2. Combining Lemmas 9 and 10, we have for any $k \geqslant 1$,

$$\bigoplus_{c \in C} M^k_{c,c} \leqslant_Q \bigoplus_{d \in D} (M^\sharp)^k_{d,d}$$

$$tr\, M^k \leqslant_Q tr(M^\sharp)^k \tag{6}$$

$$\sqrt[k]{tr\, M^k} \leqslant_Q \sqrt[k]{tr(M^\sharp)^k}. \tag{7}$$

Inequality 6 simply uses the definition of a trace. Inequality 7 holds because the nth root operation is order preserving. Summing this last inequality for all k from 1 to $|\Sigma|$ yields

$$\bigoplus_{k=1}^{|\Sigma|} \sqrt[k]{tr\, M^k} \leqslant_Q \bigoplus_{k=1}^{|\Sigma|} \sqrt[k]{tr(M^\sharp)^k},$$

so we have $\rho(M) \leqslant_Q \rho(M^\sharp)$. □

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

Recall that Proposition 5 states that for any concrete trace, there exists an abstract trace of the same length for which the cost is over the concrete trace. It follows that the alternative definition of long-run cost given in Section 4.1 is also preserved by abstraction.

Proposition 8. If $(T, T^\sharp, \alpha)$ is a correct abstraction, then

$$\limsup_{n\to\infty} \bigoplus_{\substack{t\in[\![T]\!]_{tr}\\ |t|=n}} \tilde{q}(t) \leqslant \limsup_{n\to\infty} \bigoplus_{\substack{t^\sharp\in[\![T^\sharp]\!]_{tr}\\ |t^\sharp|=n}} \tilde{q}^\sharp(t^\sharp).$$

Proof. For all n, we have

$$\forall t = \sigma_0 \to^{q_0} \dots \sigma_n \in [\![T]\!]_{tr}, |t| = n, \exists t^\sharp = \sigma_0^\sharp \to^{q_0'} \dots \sigma_n^\sharp \in [\![T^\sharp]\!]_{tr}$$

as described in Proposition 5. Since for all $i \in [0, n-1]$, $q_i \leqslant_Q q_i'$, we have $q(t) \leqslant_Q q^\sharp(t^\sharp)$. In the dioid, $a \leqslant b \wedge c \leqslant d \Rightarrow a \otimes c \leqslant b \otimes d$, so $\bigotimes_{i=0}^{n-1} q_i \leqslant \bigotimes_{i=0}^{n-1} q_i'$. Since the function $\sqrt[n]{\cdot}$ is a $\oplus$-morphism, we also have $\sqrt[n]{q(t)} \leqslant_Q \sqrt[n]{q^\sharp(t^\sharp)}$. Since the dioid Q is complete, we can sum this inequality for all traces t of length n. Hence, there exists a set $\Pi_n^\sharp \subseteq [\![T^\sharp]\!]_{tr}$ such that

$$\begin{aligned} \bigoplus_{\substack{t\in[\![T]\!]_{tr}\\ |t|=n}} \sqrt[n]{q(t)} &\leqslant_Q \bigoplus_{\substack{t^\sharp\in\Pi_n^\sharp\\ |t^\sharp|=n}} \sqrt[n]{q^\sharp(t^\sharp)} \\ &\leqslant_Q \bigoplus_{\substack{t^\sharp\in[\![T^\sharp]\!]_{tr}\\ |t^\sharp|=n}} \sqrt[n]{q^\sharp(t^\sharp)}\,. \end{aligned}$$

The last step follows from the fact that $a \leqslant b \Rightarrow a \leqslant b \oplus c$ in the dioid. Since this inequality is true for all n, we can take its superior limit to complete the proof. □

4.3. *Traces meet cycles*

We now show that if Σ is finite, and for dioids where the carrier set is $\overline{\mathbb{R}}$ and operator $\otimes$ is the arithmetical $+$ (so that the nth root operator corresponds to division by n), the notion of long-run cost defined with respect to accessible cycles coincides with the notion of long-run cost defined as the limit of the maximum average cost of traces whose length tends to infinity. To establish this result, we have to show that the cost of a prefix of a trace becomes negligible when this trace becomes arbitrarily long. We thus impose the following hypothesis.

Hypothesis 1. All transitions δ that are not in a cycle satisfy the condition $q(\delta) \neq +\infty$.

Hypothesis 1 excludes certain pathological matrices with atomic operations that have infinite costs. If a cycle contains a $+\infty$ transition, the ρ value indicates it.

Theorem 3. Let $T = \langle M, I\rangle$ be a transition system with $M \in \Sigma \times \Sigma \to Q$. If Σ is finite and Q is a cost dioid where the carrier set is $\overline{\mathbb{R}}$ and operation $\otimes$ is the arithmetical $+$,

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

then with Hypothesis 1, we have

$$\rho(T) = \lim_{n\to\infty} \bigoplus_{\substack{t\in[\![T]\!]_{tr} \\ |t|=n}} \tilde{q}(t).$$

Proof. Let $[\![T]\!]^n_{tr}$ be the subset of all traces of length n in the trace semantics of T. We prove the theorem by bounding

$$\bigoplus_{t\in[\![T]\!]^n_{tr}} \tilde{q}(t).$$

The proof is written using the 'usual' arithmetic notation to make it more readable and intuitive. The $\otimes$ operator is set to $+$, which implies that n times the same value q is $n.q$ and that the nth root operator corresponds to division by n. The $\oplus$ operator and its associated order $\leqslant$ may be interpreted either by `max` and $\leqslant$ or by `min` and $\geqslant$.

We first handle the case where the $\top$ value appears in the (reachable) cycles of the trace semantics. In that case, there exists a length n_0 such that, for any n greater than n_0, it is possible to construct a trace of length n including this transition of maximum cost. The global and average costs of that trace are also equal to $\top$, which is equal to $\rho(T)$ as well. The equality is thus trivially verified.

We now consider the case where $\top$ does not appear at all in the reachable transitions of the semantics, and we bound

$$\bigoplus_{t\in[\![T]\!]^n_{tr}} \frac{q(t)}{|t|}.$$

Given a trace t of size $n \geqslant N$, we can decompose t as follows:

$$t = p_1.c_1 \dots c_{k-1}.p_k \text{ where } \begin{cases} \text{each } c_i \text{ is a cycle} \\ r = \sum_i |p_i| < N \\ \text{each } p_i \text{ is acyclic.} \end{cases}$$

For notational simplicity, we just treat the case were $k = 1$ since the general case is treated in the same way. Trace t may thus be written as

$$t = p.c.s \text{ where } \begin{cases} c \text{ is a cycle} \\ r = |p| + |s| \leqslant N \\ p \text{ and } s \text{ are acyclic.} \end{cases}$$

Proposition 7 says that the mean cost per transition in the cycles of T is upper-bounded by $\rho(T)$, the long-run cost of T. For a given transition system whose graph contains cycles, let q_- and q_+ be the lower and upper bounds, respectively, of all the one-step reachable transition costs of T. By the definition of a possible transition, $q_- \neq \bot$, and by hypothesis, $q_+ \neq \top$. We have

$$q(t) = (q(p) + q(s)) + q(c) \leqslant r.q_+ + (n - r).\rho(T). \tag{8}$$

This inequality is guaranteed by the fact that q_+ and $\rho(T)$ are upper bounds of any single transition cost and the average transition cost in a cycle, respectively. By definition, r and

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

$n - r$ are positive integers. We thus have

$$q(t) = (q(p) + q(s)) + q(c) \leqslant N.|q_+| + (n - A).\rho(T) \tag{9}$$

since A is either N or $-N$, depending on the sign of $\rho(T)$. We sum (9) for all t of size n to get

$$\bigoplus_{t \in [\![T]\!]^n_{tr}} q(t) \leqslant N.|q_+| + (n - A).\rho(T). \tag{10}$$

This gives the upper bound.

Since we have restricted ourselves to the set of reachable states, note that for all n, there exists a trace in $[\![T]\!]^n_{tr}$ such that, if this trace contains cycles, then they are critical, that is, their average cost equals $\rho(T)$. We will just consider one of those traces. Let $t_{\max} = p.c.s$ be such that: p is cycle-free; the cycle c is only composed of critical cycles; and suffix s is cycle-free. We have

$$q(t_{\max}) \leqslant \bigoplus_{t \in [\![T]\!]^n_{tr}} q(t). \tag{11}$$

This inequation is trivially true since $t_{\max} \in [\![T]\!]^n_{tr}$ and $\oplus$ is idempotent. We then make explicit the cost of the trace and find an under-approximation similarly:

$$\begin{aligned} q(t_{\max}) &= (q(p) + q(s)) + q(c) \\ q(t_{\max}) &\geqslant N.|q_-| + (n - A).\rho(T). \end{aligned} \tag{12}$$

This inequality is guaranteed by the fact that $r.q_-$ is a lower bound for $q(p.s)$ and $(n - r).\rho(T)$ is the exact cost of path c. Inequalities (11) and (12) lead to

$$N.|q_-| + (n - A).\rho(T) \leqslant \bigoplus_{t \in [\![T]\!]^n_{tr}} q(t). \tag{13}$$

Combining inequalities (13) and (10), we get

$$N.|q_-| + (n - A).\rho(T) \leqslant \bigoplus_{t \in [\![T]\!]^n_{tr}} q(t) \leqslant N.|q_+| + (n - A).\rho(T),$$

which, divided by n becomes

$$F(n) = \frac{N.|q_-|}{n} + \frac{n - A}{n}.\rho(T) \leqslant \frac{\bigoplus_{t \in [\![T]\!]^n_{tr}} q(t)}{n} \leqslant \frac{N.|q_+|}{n} + \frac{n - A}{n}.\rho(T) = G(n).$$

Finally, as

$$\frac{a}{n} \oplus \frac{b}{n} = \frac{a \oplus b}{n},$$

this yields

$$F(n) \leqslant \bigoplus_{t \in [\![T]\!]^n_{tr}} \frac{q(t)}{n} \leqslant G(n). \tag{14}$$

We now study the limit behaviour of (14) when n tends to infinity:

$$\lim_{n \to \infty} F(n) = \lim_{n \to \infty} G(n) = \rho(T).$$

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

This implies, using the 'usual' arithmetic notation,

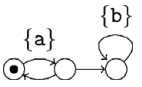
$$\lim_{n\to\infty} \bigoplus_{t\in[\![T]\!]^n_{tr}} \frac{q(t)}{n} = \rho(T)$$

or, using the semiring arithmetical notation,

$$\lim_{n\to\infty} \bigoplus_{t\in[\![T]\!]^n_{tr}} \sqrt[n]{q(t)} = \rho(T).$$

□

An example of a dioid that does not fulfill Theorem 3 is $(\mathscr{P}(S), \cap, \cup)$, which may be used, for example, to determine whether certain portions of the code are certain to be executed. Here, $\rho(T)$ does not coincide with the asymptotic behaviour of T. The diagram below illustrates this problem: while $\rho(T) = \varnothing$, all traces long enough include $\{\mathtt{a}\}$, and so does the intersection of their cost (the black dot is the initial state).



5. Case study

In this section we present a simple case study to illustrate our method. We propose a simple imperative language augmented with *array operations*, for which the cost depends on the size of the array. The costs have two components: a *time* component expressing the number of cycles that is necessary to achieve an operation and an *energy* component expressing the power consumption of that operation. A program may execute within different *energy modes*, which determine different cost behaviours (for example, different co-processors might be used to carry out the array operations) and may dynamically switch between modes. Energy costs are naturally expressed in the $(\mathbb{Q}, \max, +)$ dioid, provided we give a conversion on timed transitions – see Section 5.3.3 for details.

5.1. *Syntax*

Our language is a simple imperative language with explicit program labels (which is inspired by the Simple language defined in Miné (2004)) augmented with array manipulations. It is called ArraySimple, and its syntax is given in Figure 5.1.

We will not give the details of the classical language constructs such as `if` or `while`, which have a standard behaviour. Arrays have a length, which can be assigned through the `setlength` instruction and retrieved with the `length` operator. In addition to standard assignments to array cells, arrays can be manipulated by *global* operators through the `apply` instruction. For instance, one can compute the maximum element of an array, or perform more complex operations such as a permutation or sort. These operations will be viewed as primitive, and only their quantitative, non-functional behaviour will

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

$$
\begin{array}{lcll}
expr & ::= & X & X \in \mathcal{V} \\
 & | & -\ expr & \\
 & | & expr \diamond expr & \diamond \in \{+, -, \times, /\} \\
 & | & \texttt{length}\ A & A \in \mathcal{A} \\
 & | & A[expr] & A \in \mathcal{A} \\
test & ::= & expr \bowtie expr & \bowtie \in \{=, \neq, <, \leq\} \\
 & | & \texttt{not}\ test & \\
 & | & test\ \texttt{and}\ test & \\
 & | & test\ \texttt{or}\ test & \\
inst & ::= & X \leftarrow expr & X \in \mathcal{V} \\
 & | & \texttt{if}\ test\ \{block\}\ \texttt{else}\ \{block\} & \\
 & | & \texttt{while}\ pc_1\ test\ \{block\} & pc_1 \in \mathcal{L} \\
 & | & A[expr] \leftarrow expr & A \in \mathcal{A} \\
 & | & \texttt{apply}\ op\ A & A \in \mathcal{A}, op \in \mathcal{O} \\
 & | & \texttt{setlength}\ expr\ A & A \in \mathcal{A} \\
 & | & \texttt{setmode}\ M & M \in \mathbb{M} \\
block & ::= & pc_1\ inst;\ pc_2\ \ldots inst\ pc_n & pc_1 \ldots pc_n \in \mathcal{L}
\end{array}
$$

$\mathcal{L}$ is a finite set of program labels
$\mathcal{V}$ is a finite set of scalar variables
$\mathcal{A}$ is a finite set of array variables
$\mathcal{O}$ is a finite set of array operations
$\mathbb{M}$ is a finite set of energy modes.

Fig. 3. Syntax of the ArraySimple language.

be described. Finally, the `setmode` instruction changes the current energy mode to the specified one.

5.2. *Quantitative operational semantics*

We define an operational semantics that takes the cost of array operations into account. This semantics abstracts away the values in arrays, retaining only their size. We start from a non-deterministic collecting semantics for expressions and tests in the style of Miné (2004). Environments are mappings from scalar variables to a set of values $\mathbb{I}$ (which is $\mathbb{Z}$, $\mathbb{Q}$ or $\mathbb{R}$), and from the set of array variables to values in $\mathbb{N}$, representing their size: $\mathscr{E} = \mathscr{V} \to \mathbb{I} \cup \mathscr{A} \to \mathbb{N}$. We use $[\![e]\!]$ to denote the semantics of the expression e, that is, a mapping from environments to sets of numerical values: $[\![expr]\!] : \mathscr{E} \to \mathscr{P}(\mathbb{I}) \cup \mathbb{N}$. The collecting semantics for instructions is denoted by $\{\!|.|\!\}$ and is a mapping from $\mathscr{P}(\mathscr{E})$ to itself. For instance, the semantics of an assignment is defined by $\{\!|X \leftarrow e|\!\}R = \{\rho[X \mapsto v] \mid \rho \in R, v \in [\![e]\!]\rho\}$. See Miné (2004) for more details.

We will focus here on the transition system with the quantitative aspects of that semantics. States are composed of a program counter, an environment and an energy mode: formally, we have $\Sigma = \mathscr{L} \times \mathscr{E} \times \mathbb{M}$. The transitions of the operational semantics

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

are instrumented with costs that are elements of $\mathbb{N} \times \mathbb{Q}$, standing for time (in cycles) and energy per cycle, that is, power.

We now present the rules defining the transitions that have an impact on energy consumption. Instructions are equipped with their initial and final program labels. The `setmode` instruction simply changes the current energy mode, at a cost $c_m(m, m')$ that is a pair of constants depending on the current and new energy modes:

$$\frac{pc \ \texttt{setmode}\ m' \ pc'}{(pc, \rho, m) \rightarrow^{c_m(m,m')} (pc', \rho, m')}.$$

The cost of an assignment instruction is of greater interest, since it is extracted from a table mapping energy modes to a pair (time, energy). This table has three columns: mode, time (in cycles) and power (in mWh per cycle). The table is accessed via a function *lookup* that finds the most accurate existing mode. Being in mode **m**, the function looks up the highest mode less than or equal to **m**. If no such mode exists, the lowest mode in the table is picked. In the example we develop here, we use five modes, from **A** to **E**, **A** being the slowest but with the lowest instant power and **E** being the fastest but requiring more instant power. Note that a mode **E** can be power saving with respect to mode **A** when considering the entire operation (time $\times$ instant power). As an example, the table

$$T_a = \begin{array}{|c|c|c|}\hline \mathbf{A} & 7 & 1 \\ \hline \mathbf{B} & 4 & 1.5 \\ \hline \mathbf{E} & 2 & 4 \\ \hline \end{array}$$

is used to compute the costs of an assignment: in mode **B**, an assignment operation has a cost of 4 cycles and with an energy consumption of 1.5 mWh per cycle[†]. Given table T_a, the semantic rule for assignment is

$$\frac{pc \ \ X \leftarrow expr \ pc' \quad \rho' \in \{\!|X \leftarrow expr|\!\}_!\{\rho\} \quad c_e = lookup(T_a, m)}{(pc, \rho, m) \rightarrow^{c_e} (pc', \rho', m)}.$$

We now present the transition rule for a global array operation – the costs are also inferred from a table, and may depend on the size of the array, which is recorded in the environment:

$$\frac{pc \ \texttt{apply}\ op \ A \ pc' \quad size = \rho(A) \quad c_a = lookup([\![op]\!]_c(size), m)}{(pc, \rho, m) \rightarrow^{c_a} (pc', \rho, m)}$$

The cost semantics of an array operation $[\![.]\!]_c$ binds a size variable n to a table where the energy consumption is a function of n. Note that the number of cycles must remain constant in this model (see the discussion at the end of the paper for this point).

[†] These quantities are given here for illustrative purpose and do not reflect any real computation.

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

For instance, the following tables give cost computation for three array operations:

$$[\![\texttt{read}]\!]_c = \lambda n.\ \begin{array}{|c|c|c|}\hline \mathbf{B} & 5 & n \\ \hline \mathbf{D} & 3 & 2+3n \\ \hline \end{array}$$

Read n inputs and store them in an array.

$$[\![\texttt{min}]\!]_c = \lambda n.\ \begin{array}{|c|c|c|}\hline \mathbf{A} & 2 & \log(n) \\ \hline \mathbf{B} & 1 & 2\log(n) \\ \hline \mathbf{D} & 0 & 2n \\ \hline \end{array}$$

Put the minimum value of the array in cell 0.

$$[\![\texttt{shift } k]\!]_c = \lambda n.\ \begin{array}{|c|c|c|}\hline \mathbf{A} & 4 & 1 \\ \hline \mathbf{C} & 1 & \min(2,k) \\ \hline \end{array}$$

Right circular shift of k cells[†].

As array lengths will be determined by a static analysis using numerical abstract domains, note that if the size of the array is unbounded the analysis will give $\top$ as the result. The same will happen if the size of the array is set to the value of an expression that cannot be accurately determined by the numerical abstract domain we rely on. In that case, the analysis yields $\top$ as a conservative approximation.

In order to simplify our presentation, assignments to array cells and array length modification are given the same cost as a scalar assignment. Since array values are abstracted by the array length, array cell assignment does not modify the environment:

$$\frac{pc\ \ A[expr_1] \leftarrow expr_2\ \ pc'}{(pc,\rho,m) \rightarrow^{c_e} (pc',\rho,m)}$$

$$\frac{pc\ \ \texttt{setlength}\ expr\ A\ \ pc' \quad \rho' \in \{\!|\texttt{setlength}\ expr\ A|\!\}\{\rho\}}{(pc,\rho,m) \rightarrow^{c_e} (pc',\rho',m)}.$$

The initial collecting semantics is extended in order to take the `setlength` instruction into account:

$$\{\!|\texttt{setlength}\ expr\ A|\!\}R \stackrel{def}{=} \{\rho[A \mapsto n] \mid \rho \in R, n \in \mathbb{N} \cap [\![expr]\!]\rho\}$$

$$\begin{aligned}[\![\texttt{length}\ A]\!]\rho &\stackrel{def}{=} \{\rho(A)\} \\ [\![A[expr]]\!]\rho &\stackrel{def}{=} \mathscr{P}(\mathbb{I}).\end{aligned}$$

5.3. *Abstraction*

The abstraction we carry out on our language is based on an existing analysis using polyhedra as numerical abstract domains. In this section, we first explain how we create an interface between our language and an existing static analysis tool in order to compute the abstraction of states, then show how costs are abstracted, and, finally, how the semantics

[†] Note that k is statically determined. We do not consider operations whose consumption behaviour might depend on an expression passed as a parameter.

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

is adapted in order to compute a long-run cost that takes the timing information into account.

5.3.1. *Interfacing with* `Concurinterproc`. The `Concurinterproc` analyser (Argoud *et al.*) provides several numerical static analyses for a simple imperative language. In order to fit in with the tool's input syntax (the Simple language) and to abstract away array cell values, we first perform the following syntactic transformation[†]:

ArraySimple language	↦	Simple language
`setmode` m		`__mode__` $= m$
`apply` *op* A		`skip`
$A[expr] \leftarrow expr$		`skip`
`setlength` *expr* A		$A \leftarrow expr$
`length` A		A
$A[expr]$		`?`

We also add the following enumerated type in the program header:

```
typedef Mode = enum { A, B, C, D, E }
var __mode__;
```

The `Concurinterproc` analyser then provides for each couple (mode, program counter) a convex polyhedron that represents a relational over-approximation of the program variable values, where each dimension of the polyhedron represents a variable.

5.3.2. *Cost over-approximation.* Recall that for some transitions, the energy cost is a function of the environment ρ. See, for example, the `apply` instruction, where the cost is a function c_a from `length` A to $\mathbb{Q}$ for some array A.

The static analysis described above computes an abstract environment (a polyhedron) $\rho^\sharp$ for each couple in $\mathscr{L} \times \mathbb{M}$. This abstract environment is convex and allows us to bound `length` A by an interval by projecting the polyhedron on the dimension corresponding to the variable A. In order to compute the quantitative abstract semantics, we need a sound cost function $q^\bullet$ from `inter`($\mathbb{N}$) to $\mathbb{Q}$, where `inter`($\mathbb{N}$) is the set of intervals over $\mathbb{N}$.

The best cost approximation we can provide is defined by

$$q^\bullet_{best}(J) = \bigoplus \{q(n) \mid n \in J\},$$

which corresponds to the maximum cost associated with an element of interval J. Any cost function $q^\bullet \geqslant q^\bullet_{best}$ is a sound approximation of $q^\bullet_{best}$.

We will restrict ourselves to monotone cost functions in arrays operations. Thus

$$q^\bullet_{best} = \lambda[a, b].q(b)$$

[†] Note that the array operations are translated into `skip` instructions, but they still have their own cost in the abstract view of the operational semantics.

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

where $q(b)$ is the limit of $q(n)$ when n tends to $+\infty$ if q is a bounded function, and $+\infty$ otherwise[†].

5.3.3. $\mathbb{N}$-*timed conversion.* Recall that we are aiming to compute a long-run cost for an ArraySimple program. Up to this point, our costs have had two components: a natural number expressing the number of cycles needed by an instruction and an energy consumption per cycle. For each transition requiring n cycles, we thus create a path of length n, such that the long-run cost effectively computes an average cost per cycle.

Let $(\Sigma, I, \rightarrow)$ be a transition system where $I \subseteq \Sigma$ is the set of initial states and $\rightarrow \subseteq \Sigma \times \mathbb{N} \times \mathbb{Q} \times \Sigma$ is the transition relation where $\sigma_1 \xrightarrow{t,w} \sigma_2$ means that we can go from state σ_1 to state σ_2 in t time units (cycles) consuming each w resources. We stress the fact that the consumption here is per time unit, but we could also have used a model where the consumption is assigned to the whole transition: as $\mathbb{Q}$ is equipped with an nth root operator, we divide the global cost w for the transition into n parts, where n is the number of cycles.

We define the algebraic type Σ',

$$\Sigma' = \mathit{Just}\ \Sigma \mid \mathit{Soon}\ \Sigma \times \mathbb{N} \times \mathbb{Q},$$

together with a new transition system $(I', \Rightarrow)$, by unfolding $\rightarrow$ transitions requiring more than one cycle defined by

$$\begin{aligned} I' &\subseteq \Sigma' \\ \Rightarrow &\subseteq \Sigma' \times \mathbb{Q} \times \Sigma' \end{aligned}$$

with $I' = \{\mathit{Just}\ \sigma \mid \sigma \in I\}$ and $\Rightarrow$ satisfying the following rules:

$$\text{fire one:}\ \frac{\sigma \xrightarrow{1,w} \sigma'}{\mathit{Just}\ \sigma \overset{w}{\Rightarrow} \mathit{Just}\ \sigma'}$$

$$\text{fire several:}\ \frac{\sigma \xrightarrow{t,w} \sigma' \quad t > 1}{\mathit{Just}\ \sigma \overset{w}{\Rightarrow} \mathit{Soon}\ \sigma', t-1, w}$$

$$\text{fire zero:}\ \frac{\sigma \xrightarrow{0,w} \sigma' \quad \mathit{Just}\ \sigma' \overset{w'}{\Rightarrow} \sigma''}{\mathit{Just}\ \sigma \overset{w \otimes w'}{\Rightarrow} \sigma''}$$

$$\text{countdown:}\ \frac{t > 1}{\mathit{Soon}\ \sigma', t, w \overset{w}{\Rightarrow} \mathit{Soon}\ \sigma', t-1, w}$$

$$\text{end countdown:}\ \frac{}{\mathit{Soon}\ \sigma', 1, w \overset{w}{\Rightarrow} \mathit{Just}\ \sigma'}.$$

We allow zero-timed transitions; to avoid the risk of an infinite stack of *fire zero* rules, we require that all loops contain at least a transition with a time cost greater than or equal to one cycle.

[†] In the unfortunate case where the analysis is unable to provide a bound for array lengths, this cost is poor information.

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

5.4. *Experimental results*

Abstraction and long-run cost computations are performed in three phases:

— First, the `Concurinterproc` tool is used to compute polyhedral approximations in order to give bounds on array sizes. Abstract states corresponding to couples in $\mathscr{L} \times \mathbb{M}$ are enumerated, by just retaining those abstract states of interest, that is, that are reachable from initial states.
— Then a matrix, indexed over abstract states, is constructed whose values are determined by the cost tables and the cost approximation inferred by the polyhedral analysis. This step has been implemented in Objective Caml.
— The long-run cost is finally computed with the help of a Scilab library for max-plus algebra, which offers an efficient way to compute the long-run cost through sparse matrices and Howard's algorithm for eigenvalues (Cochet-Terrasson *et al.* 1998) and handles up to 20k states.

As an example, consider the following program, which iteratively solves the linear system `mat * x = b`, where `mat` is a non-singular $n \times n$ matrix, `b` is the right-hand side for the linear system and `x` is the solution. It uses the Gauss–Seidel Method with relaxation to compute an approximation of the exact solution. The ArraySimple version of the algorithm is depicted in Figure 4, where some operations on scalars have been simplified, since they do not influence the cost computation.

We define five energy modes, from **A** to **E**, which determine the energy consumption of array operations. For instance, the following table gives values for the `copy` operation:

A	5	0.2
C	2	$\lambda n.n$
D	1	$\lambda n.3. * n$
E	0	$\lambda n.5. * n$

The aim of the experiment is to compute the long-run cost (average energy consumption per cycle) with different mode switches in the code, choosing an energy depending on the common size `n` of the arrays. To this end, the `__H1__` string in the initial code is successively replaced by `assume n <= 1000; if n <` xx `then setmode` m_H`; else setmode` m_L, where xx is a constant threshold in $\{100, 200, \ldots, 900\}$, and m_H and m_L are high and low energy modes, respectively.

The result of long-run cost computations is given in Table 2. As expected, the higher the energy level, the higher the long-run cost. One might want, for instance, to fix a maximal admissible energy consumption, and tune the cost switch in the code in order to achieve the best time performance. Table 3 gives the best thresholds for an energy consumption of 250mWh and 50mWh, respectively.

6. Related work

This article follows on from Sotin *et al.* (2006) and Cachera *et al.* (2008), where we presented a preliminary version of this work. This new version broadens and provides

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

```
var x,n,residue,b,i,iter,ITERMAX,mat,omega,tmp,eps,m,p :int,
dumb:int;
begin

assume length n <= 1000;
setsize x n;
setsize residue n;
setsize b n;
setsize tmp n;
apply copy_to residue,x;

__H1__

m = eps - 1;
iter = 0;

while iter < ITERMAX and m <= eps do
      i = 0;
      apply copy_to b, tmp;
      while i < n do
            __H1__
            apply prod residue, mat; // :-> p //
            dumb = dumb; // tmp[i] = tmp[i] - p;
      i = i+1; done;
      apply mult_scal tmp, omega;
      apply sum residue, tmp;

      apply sub x, residue;
      apply max x; // :-> m //
      apply copy_to residue, x;
iter = iter+1; done;

skip;
setmode A;

apply copy_to b, tmp;
i = 0;
while i < n do
      apply prod x, mat; // :-> p //
      dumb = dumb; // residue[i] = p;

i = i+1; done;

end
```

Fig. 4. Iterative solving of a linear system in the ArraySimple language

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

Mode		Threshold								
High	Low	100	200	300	400	500	600	700	800	900
A	B	3.6	3.6	3.6	3.6	3.6	3.6	3.6	3.6	3.6
A	C	13.7	23.0	32.3	41.6	50.9	60.2	69.5	78.8	88.1
A	D	39.9	77.8	115.7	153.6	191.6	229.5	267.4	305.4	343.3
A	E	81.2	162.2	243.1	324.1	405.1	486.0	567.0	647.9	728.9
B	C	13.7	23.0	32.3	41.6	50.9	60.2	69.5	78.8	88.1
B	D	39.9	77.8	115.7	153.6	191.6	229.5	267.4	305.4	343.3
B	E	81.2	162.2	243.1	324.1	405.1	486.0	567.0	647.9	728.9
C	D	97.5	97.5	115.7	153.6	191.6	229.5	267.4	305.4	343.3
C	E	97.5	162.2	243.1	324.1	405.1	486.0	567.0	647.9	728.9
D	E	381.6	381.6	381.6	381.6	405.1	486.0	567.0	647.9	728.9

Table 2. *Long-run cost for different values of threshold for high and low energy modes*

		(a) maximum lrc = 250 Low			
	250	A	B	C	D
High	E	300	300	300	none
	D	600	600	600	
	C	900	900		
	B	900			
		(b) maximum lrc = 250 Low			
	50	A	B	C	D
High	E	none	none	none	none
	D	100	100	none	
	C	400	400		
	B	900			

Table 3. *Best choices for threshold values*

details of the theoretical developments. In particular, the link with classical abstract interpretation theory has been made clearer, since the earlier versions of this work only considered partition-based abstractions, which are fairly limiting. Moreover, a completely new case study is proposed, switching from a cache miss analysis to a power consumption analysis.

The present work was inspired by the quantitative abstract interpretation framework developed by Di Pierro and Wiklicky (Di Pierro and Wiklicky 2000). We have followed their approach in modelling programs as linear operators over a vector space, with the notable technical difference that their operators act over a semiring of probabilities

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

while ours work with idempotent dioids. Working with idempotent dioids means that we have been able to exploit known results from Discrete Event Systems theory (Baccelli *et al.* 1992), which makes intensive use of such structures. Another difference compared with the approach of Di Pierro and Wiklicky lies in the kind of program being analysed: we have considered an intermediate bytecode language rather than declarative languages (probabilistic concurrent constraint programming and the lambda calculus (Di Pierro *et al.* 2005a)).

In Di Pierro and Wiklicky's work, the relation to abstract interpretation is justified by the use of the pseudo-inverse of a linear operator, which is similar to a Galois connection mechanism, to enforce the soundness of abstractions. Our approach can be seen as intermediate between their approach and the classical abstract interpretation: on the one hand, we use residuation theory to get a pseudo-inverse for linear abstraction functions, but on the other hand, we benefit from the partially ordered structure of dioids to give guarantees of soundness under the assumption $\alpha \circ M \leqslant_D M^\sharp \circ \alpha$, which is a classical requirement in abstract interpretation.

The work of several other authors makes use of an idempotent semiring to describe quantitative aspects of computations, specifically, in the form of constraint semirings (Bistarelli *et al.* 1997; Bistarelli 2004), particularly under the name of *soft constraints*. These have been used in the field of Quality of Service (De Nicola *et al.* 2005; Santini 2008), in particular, with systems modelled by graph rewriting mechanisms (Hirsch and Tuosto 2005). In all these approaches, the $\oplus$ and $\otimes$ operators of the constraint semiring are used to combine constraints. Amongst this work, two similar approaches deserve particular mention since they deal with abstraction mechanisms. Aziz (2006) makes use of semirings in a mobile process calculus derived from the π-calculus to model the cost of communicating actions. He also defines a static analysis framework by abstracting 'concrete' semirings into abstract semirings of reduced cardinality and defining abstract semiring operators accordingly. For instance, the $(\mathbb{R}_+ \cup \{+\infty\}, \min, +)$ semiring can be abstracted by a $(\{low, medium, high\}, \min, \max)$ one. Bistarelli *et al.* (2002) defines an abstract interpretation based framework for abstracting soft constraint satisfaction problems (SCSPs). As in Aziz's approach, they get an abstract SCSP by just changing the associated semiring, leaving unchanged the remainder of the structure. Concrete and abstract semirings are related by means of a Galois insertion, which provides correctness results. A major difference between these approaches and ours is that they abstract the semiring and leave the system itself unchanged, while we abstract the structures of states and retain the same dioid. Moreover, even though they also deal with dioids, none of these approaches makes use of a notion of long-run cost to express an average quantitative behaviour of a system.

7. Conclusion

We have shown how to abstract the long-run cost of programs whose operational semantics is defined as transition systems labelled by costs taken from a particular kind of dioid. In such cases, we have shown that the semantics is a linear operator over the moduloid associated with this dioid. We have used a well-known characterisation of

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

the asymptotic behaviour of a discrete event system to define the notion of the long-run cost of such a semantics, and proposed a novel way of analysing the long-run behaviour of the program. We have characterised this long-run cost as being a maximal average cost per transition on very long traces of the semantics. Computing the exact long-run cost of a program is in general too expensive, so we have extended the linear operator framework with a notion of abstraction of the semantics, which is also expressed as a linear operator. A correctness relation between concrete and abstract semantic matrices ensures that the cost computed from the abstract semantics is an over-approximation of the concrete one. We illustrated the notions of dioids, quantitative semantics, abstraction and long-run cost by analysing a program written in a simple imperative language manipulating arrays and explicitly taking energy consumption into account. The cost of an instruction is composed of a constant number of cycles and of an energy consumption per cycle, depending on the operand size. A less restrictive model with non-constant time costs can be designed, where the dioid of costs is the dioid of functions from positive reals (time) to positive reals (energy). This kind of model is currently under investigation.

An interesting avenue for further work would be to relax the correctness criterion so that the abstract estimate is 'close' to (but not necessarily greater than) the exact quantity. For certain quantitative measures, a notion of 'closeness' might be of interest, in contrast with the qualitative case, where static analyses must err on the safe side.

References

De Alfaro, L. (1998) How to Specify and Verify the Long-Run Average Behavior of Probabilistic Systems. In: *Proceedings 13th Symposium on Logic in Computer Science (LICS'98)*, IEEE Computer Society Press 174–183.

Argoud, M., Jeannet, B. and Lalire, G. The `concurinterproc analyzer`. Available at `http://pop-art.inrialpes.fr/interproc/concurinterprocweb.cgi`.

Aspinall, D., Beringer, L., Hofmann, M., Loidl, H.-W. and Momigliano, A. (2007) A program logic for resources. *Theor. Comput. Sci.* **389** (3) 411–445.

Aziz, B. (2006) A Semiring-based Quantitative Analysis of Mobile Systems. *Electronic Notes in Theoretical Computer Science* **157** (1) 3–21.

Baccelli, F., Cohen, G., Olsder, G. and Quadrat, J.-P. (1992) *Synchronization and Linearity*, Wiley.

Bistarelli, S. (2004) Semirings for Soft Constraint Solving and Programming. *Springer-Verlag Lecture Notes in Computer Science* **2962**.

Bistarelli, S., Codognet, P. and Rossi, F. (2002) Abstracting soft constraints: framework, properties, examples. *Artif. Intell.* **139** (2) 175–211.

Bistarelli, S., Montanari, U. and Rossi, F. (1997) Semiring-Based Constraint Satisfaction and Optimization. *Journal of the ACM* **44** (2) 201–236.

Brazdil, T., Esparza, J. and Kucera, A. (2005) Analysis and Prediction of the Long-Run Behavior of Probabilistic Sequential Programs with Recursion (Extended Abstract). In: *FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society 521–530.

Cachera, D., Jensen, T., Jobin, A. and Sotin, P. (2008) Long-run cost analysis by approximation of linear operators over dioids. In: Meseguer, J. and Roşu, G. (eds.) Proceedings of the 12th International Conference on Algebraic Methodology and Software Technology. *Springer-Verlag Lecture Notes in Computer Science* **5140** 122–138.

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press

Cochet-Terrasson, J., Cohen, G., Gaubert, S., Mc Gettrick, M. and Quadrat, J.-P. (1998) Numerical Computation of Spectral Elements in Max-Plus Algebra. In: *Proceedings of the IFAC Conference on System Structure and Control.*

Cousot, P. and Cousot, R. (1977) Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In: *Proceedings 4th Symposium on Principles of Programming Languages (POPL'77)* 238–252.

Davey, B. A. and Priestley, H. A. (1990) *Introduction to Lattices and Order*, Cambridge University Press.

De Nicola, R., Ferrari, G., Montanari, U., Pugliese, R. and Tuosto, E. (2005) A Basic Calculus for Modelling Service Level Agreements. In: Proceedings International Conference on Coordination Models and Languages *Springer-Verlag Lecture Notes in Computer Science* **3454** 33–48.

Di Pierro, A., Hankin, C. and Wiklicky, H. (2005a) Probabilistic λ-calculus and Quantitative Program Analysis. *J. Logic and Computation* **15** (2) 159–179.

Di Pierro, A., Hankin, C. and Wiklicky, H. (2005b) Measuring the confinement of probabilistic systems. *Theor. Comput. Sci.* **340** (1) 3–56.

Di Pierro, A. and Wiklicky, H. (2000) Concurrent Constraint Programming: Towards Probabilistic Abstract Interpretation. In: *Principles and Practice of Declarative Programming* 127–138.

Dudnikov, P. and Samborskii, S. (1992) Endomorphisms of finitely generated free semimodules. In: Maslov, V. P. and Samborskii, S. N. (eds.) Idempotent analysis. *Advances in Soviet Mathematics* **13**, American Mathematical Society 65–85.

Dudnikov, P. and Samborskii, S. (1987) Endomorphisms of semimodules over semirings with an idempotent operation – preprint of the Mathematical Institute of the Ukrainian Academy of Sciences (in Russian). (English translation in *Math. USSR Izvestiya* (1992) **38** (1) 91–105).

Gondran, M. and Minoux, M. (2008) *Graphs, Dioids and semirings, New Models and Algorithms*, Springer-Verlag.

Hirsch, D. and Tuosto, E. (2005) SHReQ: Coordinating Application Level QoS. In: *SEFM '05: Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods*, IEEE Computer Society 425–434.

Miné, A. (2004) *Weakly Relational Numerical Abstract Domains*, Ph.D. thesis, École Polytechnique, Palaiseau, France.

Santini, S. (2008) Managing quality of service with soft constraints. In: *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*, AAAI Press 1869–1870.

Sotin, P., Cachera, D. and Jensen, T. (2006) Quantitative Static Analysis over Semirings: Analysing Cache Behaviour for Java Card. In: Di Pierro, A. and Wiklicky, H. (eds.) *QAPL06, Quantitative Aspects of Programming Languages.*

https://doi.org/10.1017/S0960129510000113 Published online by Cambridge University Press