HAL
open science

Long-Run Cost Analysis byApproximation of Linear Operators over Dioids

David Cachera, Thomas Jensen, Arnaud Jobin, Pascal Sotin

To cite this version:

David Cachera, Thomas Jensen, Arnaud Jobin, Pascal Sotin. Long-Run Cost Analysis byApproximation of Linear Operators over Dioids. [Research Report] RR-6338, 2007, pp.35. inria-00182338v3

HAL Id: inria-00182338

https://inria.hal.science/inria-00182338v3

Submitted on 25 Jan 2008 (v3), last revised 25 Jan 2008 (v4)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Long-Run Cost Analysis by Approximation of Linear Operators over Dioids

David Cachera — Thomas Jensen — Arnaud Jobin — Pascal Sotin

N° 6338 — version 2

initial version October 2007 — revised version January 2008

Thème SYM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--6338--FR+ENG

Long-Run Cost Analysis by Approximation of Linear Operators over Dioids

David Cachera*, Thomas Jensen†, Arnaud Jobin‡, Pascal Sotin§

Thème SYM — Systèmes symboliques
Projet Lande

Rapport de recherche n° 6338 — version 2 — initial version October 2007 — revised version January 2008 — 32 pages

Abstract: We present a static analysis technique for modeling and approximating the long-run resource usage of programs. The approach is based on a quantitative semantic framework where programs are represented as linear operators over dioids. We show how to extract the long-run cost of a program from the matrix representation of its semantics. An essential contribution is to provide abstraction techniques which make it feasible to compute safe over-approximations of this cost. A theorem is proved stating that such abstractions yield correct approximations of the program's long-run cost. The theoretical developments are illustrated on a concrete example taken from the analysis of the cache behaviour of a simple bytecode language.

Key-words: Programming Language Semantics, Static Analysis, Resource Analysis, Long-Run Cost, Dioids

* ENS Cachan/CNRS
† CNRS
‡ INRIA
§ CNRS/DGA

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Analyse de coût moyen par approximation d'opérateurs linéaires sur des dioïdes

Résumé : Nous présentons une technique d'analyse statique pour modéliser et calculer de façon approchée l'utilisation asymptotique moyenne de ressources par un programme. L'approche se fonde sur un modèle de sémantique quantitative où les programmes sont représentés par des opérateurs linéaires sur un dioïde. Nous montrons comment extraire le coût moyen asymptotique d'un programme de la représentation matricielle de sa sémantique. Une contribution essentielle consiste à fournir des techniques d'abstraction permettant de calculer effectivement une surapproximation de ce coût. Nous prouvons que de telles abstractions fournissent des approximations correctes. Nous illustrons ces notions sur un exemple d'analyse de défauts de cache pour un langage simple de bytecode.

Mots-clés : Sémantique des langages de programmation, analyse statique, analyse de consommation de ressources, coût moyen asymptotique, dioïdes

Contents

1 Introduction **3**

2 Linear operator semantics **4**
2.1 Cost dioid 5
2.2 Semantics as linear operators over dioids 7
2.3 Running example: quantitative semantics 8

3 Abstraction **10**
3.1 Galois connections and pseudo-inverses 10
3.2 Abstraction over cost dioids 11
3.3 Induced abstract semantics 11
3.4 Running example: abstraction 12

4 Long-run cost **14**
4.1 Semantics of the long run cost 15
4.2 Ensuring correctness 15
4.3 Traces meet cycles 16
4.4 Running example: long-run cost 16

5 Related work **17**

6 Conclusion **18**

A Some rules of a bytecode quantitative semantics **21**

B Cost dioids **22**

C Trace semantics and long run cost **25**

D Correctness of the long-run cost **30**

1 Introduction

This article is concerned with the semantics-based program analysis of quantitative properties pertaining to the use of resources (time, memory, ...). Analysis of such non-functional properties relies on an operational model of program execution where the cost of each computational step is made explicit. We take as starting point a standard small-step operational semantics expressed as a transition relation $\sigma \to^q \sigma'$ between states $\sigma, \sigma' \in \Sigma$ extended with *costs* $q \in Q$ associated to each transition. The set Q of costs is supposed to have two operations for composing costs: a "product" operator that combines the costs along an execution path, and a "sum" operator that combines costs coming from different paths. These operators will give Q a structure of dioid. The sum operator induces a partial order on costs that will serve as a basis for approximating costs. From such a rule-based semantics, there is a straightforward way to obtain a transition matrix, in which a matrix entry represents the cost of passing from one state of the program to another. This expresses the semantics of a program as a linear operator on $Q(\Sigma)$, the moduloid of vectors of elements of Q indexed over Σ.

In this paper, we are interested in analysing programs with cyclic behaviour (such as reactive systems) in which the asymptotic average cost along cycles, rather than the global cost of the entire execution, is of interest. We define the notion of *long-run cost* for a program which 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. 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, in order to return an approximation of the program costs that is feasible to compute.

In line with the semantic machinery used to model programs, abstractions are also defined as linear operators from the moduloid over the concrete state space into the moduloid over the abstract one. Given such an abstraction over the semantic domains, we then have to abstract the transition matrix of the program itself into a matrix of reduced size. We give a sufficient condition for an abstraction of the semantics to be correct, *i.e.* to give an over-approximation of the real cost, and show how an abstract semantics that is correct by construction can be derived from the concrete one. The long-run cost of a program is thus safely approximated by an abstract long-run cost, with respect to the order relation induced by the summation operator of the dioid.

The framework proposed here covers a number of different costs related to resource usage (time and memory) of programs. To demonstrate the generality of the framework, our running example considers the less common (compared to time and space) analysis of cache behaviour and the number of cache misses in programs. We illustrate the notions of quantitative semantics, abstraction and long-run cost on a program written in a simple, intermediate bytecode language (inspired by Java Card) onto which we impose a particular cache model.

The paper is structured as follows. Section 2 defines the quantitative semantics as a linear operator over a moduloid. We give the general form of this semantics, and precisely define the notion of cost dioid 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. 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 lists related work and Section 6 concludes and discusses 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 quantitative operational semantics of P is given as a transition relation, defined by inference rules of the following form: $\sigma \rightarrow^q \sigma'$ where σ, σ' are states of Σ, and q is the cost attached to the transition from σ to σ' (q is function of σ and σ'). The set Q of costs and its structure will be made precise in the

next subsection. We associate to P the transition system $T = \langle \rightarrow^{\cdot}, I \rangle$, where I is the set of initial states of P. The trace semantics of P is defined as the trace semantics of T.

$$[\![P]\!]_{tr} = [\![T]\!]_{tr} = \{\sigma_0 \rightarrow^{q_0} \ldots \sigma_{n-1} \rightarrow^{q_{n-1}} \sigma_n \mid \sigma_1 \in I, \sigma_i \rightarrow^{q_i} \sigma_{i+1}\}$$

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 $\rightarrow^{\cdot} \subseteq \Sigma \times \Sigma \rightarrow Q$, written $\sigma \rightarrow^{q} \sigma'$. 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 [3]) 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, as presented in the next subsection.

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

There may be several ways to reach a state σ' from a state σ, due to the presence of loops and 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' is 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, commutative and has a neutral element* e*. Quantity* e *represents a transition that costs nothing.*
2. *Operator* $\oplus$ *is associative, commutative and has* $\perp$ *as neutral element. Quantity* $\perp$ *represents the impossibility of a transition.*
3. $\otimes$ *is distributive over* $\oplus$*, and* $\perp$ *is absorbing element for* $\otimes$ ($\forall x. x \otimes \perp = \perp \otimes x = \perp$)*.*
4. *The preorder defined by* $\oplus$ ($a \leq b \Leftrightarrow \exists c : a \oplus c = b$) *is an order relation (*i.e. *it satisfies* $a \leq b \wedge b \leq a \Rightarrow a = b$*).*

By 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 [18, Proposition 6.1.7].

Lemma 1 $\oplus$ *and* $\otimes$ *preserve the order* $\leq$*, i.e., for all* $a, b, c \in Q$ *with* $a \leq b$*,*

$$a \otimes c \leq b \otimes c \qquad \textit{and} \qquad a \oplus c \leq b \oplus c$$

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, *i.e.* we work with idempotent dioids.

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 are, respectively, the orders $\leq$ and $\geq$ over real numbers, extended to $\overline{\mathbb{R}}$ in the usual way. Note that in an idempotent dioid $a \leq b \Leftrightarrow a \oplus b = b$. 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 impose the assumption that our dioids are complete [7].

Definition 3 *An idempotent dioid is* complete *if it is closed with respect to infinite sums*[1]*, and the distributivity law holds also for an infinite number of summands.*

A complete dioid is naturally equipped with a top element, that we shall write $\top$, which is the sum of all elements in the dioid. Remark that a complete dioid is always a complete lattice, thus equiped with a meet operator $\wedge$ [5].

The notion of long-run cost 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 a nth root function.

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

A sequence containing n transitions, each costing, on the average, $\sqrt[n]{q}$, will thus cost q. Some examples of nth root can be found in Figure 1.

To be able to easily deal with the nth root, we make the assumption that the nth power is $\oplus$-lower-semicontinuous ($\oplus$-lsc for short).

Definition 5 *The* n*th 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 $X \subseteq Q$ (in that case we say that the nth power is a $\oplus$*-morphism*).

The following definition summarizes the required conditions for our structure.

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

In such dioids, the following properties are naturally verified.

Proposition 1 *In a cost dioid* Q*, we have:*

(i) The n*th root is* $\oplus$*-lsc:* $\forall X \subseteq Q, \forall n > 0,\ \sqrt[n]{\bigoplus_{x \in X} x} = \bigoplus_{x \in X} \sqrt[n]{x}$*,*

[1] If we see the operator $\oplus$ as a least upper bound, this notion is analogous to that of a *complete semi-lattice* in classical lattice theory.

(ii) Forall $a, b \in Q$ and $n, m > 0$, $\sqrt[n]{a} \oplus \sqrt[m]{b} \geq \sqrt[n+m]{a \otimes b}$.

Property (i) immediately follows from the fact that the nth power is $\oplus$-lsc whereas an intermediate lemma is needed to prove property (ii) *(proofs are given in appendix B).*

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 $a, b \in Q, a \oplus b =$ either a or b.*

Definition 8 *A dioid $(Q, \oplus, \otimes)$ is* double-idempotent *if both operators $\oplus$ and $\otimes$ are idempotent.*

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

Definition 9 *A dioid $(Q, \oplus, \otimes)$ satisfies the* cancellation condition *if $\otimes$ is cancellative, which means that for all $a, b, c \in Q$, the relation $a \otimes b = a \otimes c$ and $a \neq \perp$ imply $b = c$.*

Proposition 2 *The following dioids are cost dioids.*

(1) Complete and selective commutative dioids equipped 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 $X^n = q$ always has at least one solution.

For dioids of kind (1) and (2) we only have to prove that the nth power is $\oplus$-lsc. For dioids of type (3) we also have to prove that if equation $X^n = q$ has a solution, then this solution is unique *(proofs are given in appendix B).*

For instance, $(\overline{\mathbb{R}}, \mathtt{max}, +)$ is a cost dioid that may be used for the definition of the Worst Case Execution Time: when two states can be joined by several sequences of transitions which cost different times, the worst time is taken. To compute the cost of a sequence of transitions, we sum the costs of each transition. Figure 1 lists some examples of cost dioids.

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* $M \in \mathcal{M}_{\Sigma \times \Sigma}(Q)$ with

$$M_{\sigma,\sigma'} = \begin{cases} q \text{ if } \sigma \rightarrow^q \sigma' \\ \perp \text{ otherwise} \end{cases}$$

Here, $\mathcal{M}_{\Sigma \times \Sigma}(Q)$ 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

	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
	$\mathcal{P}(S)$	$\cap$	$\cup$	q
	$\mathcal{P}(S)$	$\cup$	$\cap$	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	$+$	$\frac{q}{n}$

Figure 1: Some examples of cost dioids

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 embed the costs for paths of length n). 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: $M \leq M' \Leftrightarrow \forall i, j. M_{i,j} \leq M'_{i,j}$. A transition matrix may also be seen as a linear operator on the moduloid $Q(\Sigma)$, as defined below.

Definition 10 *Let $(E, \oplus, \otimes)$ is a commutative dioid. A moduloid over E is a set V equipped 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 ranges from $E \times V$ to V, and verifies*
   - *(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 $\perp \odot x = 0$,*
   - *(e) $\forall \lambda \in E, \lambda \odot 0 = 0$.*

If E is an idempotent dioid, then any moduloid V over E is also an idempotent dioid, equipped with a canonical order defined from the $\oplus$ operation.

As for vector spaces, if n is a given integer, E^n, 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 generalize 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: $(Mu)_i = \bigoplus_{j=1}^{+\infty} \delta_M(i, j) \otimes \delta_u(j)$. Everywhere in this paper, we will keep the matrix notation for the sake of simplicity, even for an infinite set of indices.

2.3 Running example: quantitative semantics

We illustrate the notions of cost and quantitative semantics on a simple bytecode language, inspired by the Java Card language. Figure 2 shows part of the

factorial program written in this language (together with its source code). The quantity we are interested in is the number of cache misses related to read accesses (read miss behaviour). In order to describe the read miss behaviour of programs, we extend the semantics of a simple bytecode language [20, 16, 6] with a cache model and with quantities expressing the number of read misses.

A state in our semantics contains a heap, a call stack of frames, and within each frame an instruction pointer for the current method, an array of local variables and an operand stack. In addition to these standard elements, a state also contains a set of *logical addresses*, representing which values are present in the cache at this point of the execution. This set is managed similarly to the cache. For example, the maximum size of this set will correspond to the size of the physical cache, and the replacement policy will model the one provided by the cache (*e.g.* LRU, FIFO). The cache description is hidden in a function $C' = update(C, [access])$ where C and C' denote the cache before and after a transition, respectively, and where $[access]$ is a list of memory accesses.

Memory is accessed with two operators (a read or write access) which take two parameters, specifying what volume of data is to be accessed, and where these data are stored. For example, $read_\tau(\texttt{heap}.3.x)$ means that data of type τ is read at the address `heap`.*3.x*, *i.e.* field x of the third object in the heap. In the same way, `stack`.*frameId.n* points to the n-nth element in the operand stack of a given frame, and `local`.*frameId.local* points to a local variable in a certain frame. We give an example of a semantic rule: the `load` instruction, which loads a typed local variable, indexed by i, on the top of the operand stack. The first two hypotheses of the rule correspond to the standard semantics. The third and fourth rules define how the cache evolves when executing a `load`. The fifth hypothesis computes the cost. *Some other rule examples can be found in appendix A.*

Source	Bytecode
x=1;	`1: push 1`
	`2: store x`
for (i=2;...	`3: push 2`
	`4: store i`
...i<=n;...	`5: load i`
	`6: load n`
	`7: if ≤goto 14`
*x=x*i;*	`8: load x`
	`9: load i`
	`10: numop mul`
	`11: store x`
...i++)	`12: inc i`
	`13: goto 5`
return x;	`14: load x`
	`15: return`

Figure 2: Factorial program

$$\frac{\begin{array}{c} InstrAt(m, ip) = \texttt{load}\ \tau\ i \wedge L[i] = d \\ S' = d :: S \wedge size(S) = t \\ access = [read_\tau(\texttt{local}.f.i); write_\tau(\texttt{stack}.f.t+1)] \\ C' = update(C, access) \\ q = nbRmiss(C, access) \end{array}}{\langle H, \ll f, m, ip, L, S \gg :: fr, C\rangle \rightarrow^q \langle H, \ll f, m, ip+1, L, S' \gg :: fr, C'\rangle}$$

The number of read misses depends on the current state of the cache and the way it is accessed. This is defined precisely by the function $nbRmiss(C, access)$ that computes the number of read misses generated by the list of memory ac-

cesses *access* if the cache at the beginning of the instruction is C. Here is the pseudocode of function *nbRmiss*.

$$\mathit{nbRmiss}\ c\ [\,] = 0$$
$$\mathit{nbRmiss}\ c\ [a/r] = \mathit{nbRmiss}\ (\mathit{update}\ c\ [a])\ r + \begin{cases} 1 & \text{if } a = \mathit{read}\ m \text{ and } m \notin c \\ 0 & \text{otherwise} \end{cases}$$

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 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 [11] in terms of linear operators over moduloids.

3.1 Galois connections and pseudo-inverses

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

Definition 11 *Let $(C, \leq_C)$ and $(D, \leq_D)$ be two partially ordered sets. Two mappings $\alpha : C \mapsto D$ and $\gamma : D \mapsto C$ satisfying $\forall c \in C, \forall d \in D, c \leq_C \gamma(d) \iff \alpha(c) \leq_D d$, are called a Galois connection (C, α, γ, D).*

As above, α is an abstraction function, while γ is called a concretization function. An equivalent definition is:

Definition 12 *Let $(C, \leq_C)$ and $(D, \leq_D)$ be two partially ordered sets, and $\alpha : C \mapsto D$ and $\gamma : D \mapsto C$ two monotonic mappings. Then (C, α, γ, D) forms a Galois connection if and only if $\alpha \circ \gamma \leq Id_D$ and $Id_C \leq \gamma \circ \alpha$.*

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 that of the existence of a concretization function, given an abstraction α. In [14], Di Pierro and Wiklicky describe the framework of Probabilistic Abstract Interpretation, where the abstraction function is a linear operator over the semiring of probabilities. They obtain a concretization function through the Moore-Penrose pseudo-inverse. As we will not be able to define an exact inverse in the general case, nor to 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 *Let E and F be two complete dioids, f a monotone mapping from E to F. We call subsolution of equation $f(x) = b$ an element y such that $f(y) \leq b$. 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 from F into E which is upper[2] semi-continuous such that $f \circ f^\dagger \leq Id_F$ and $Id_E \leq f^\dagger \circ f$.*

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

3.2 Abstraction over cost dioids

We now show how the notions of abstraction and concretization 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 $\alpha^\uparrow \in \mathcal{M}_{\Sigma^\sharp \times \Sigma}(Q)$ by setting

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

In what follows, $\alpha^\uparrow$ will be denoted by α when no confusion can arise and $\leq$ will stand for the order defined on $\mathcal{M}_{\Sigma^\sharp \times \Sigma^\sharp}(Q)$ in Section 2.2.

Recall that the set of vectors over a complete idempotent dioid is itself a complete idempotent dioid. As the abstraction function is linear, it trivially fulfills requirements 2 of Proposition 3 and we immediately get the following result.

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

$$\alpha^\uparrow \circ \alpha^\dagger \leq Id_{\Sigma^\sharp} \qquad \textit{and} \qquad Id_\Sigma \leq \alpha^\dagger \circ \alpha^\uparrow.$$

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, both from the point of view of its traces and from the costs it will 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 13 (Correct abstraction) *Let $T = \langle M, I \rangle$ a transition system over the concrete domain, with $M \in \mathcal{M}_{\Sigma \times \Sigma}(Q)$ and $I \subseteq \Sigma$. Let $T^\sharp = \langle M^\sharp, I^\sharp \rangle$ be a transition system over the abstract domain, with $M^\sharp \in \mathcal{M}_{\Sigma^\sharp \times \Sigma^\sharp}(Q)$ and $I^\sharp \subseteq \Sigma^\sharp$. Let α be an abstraction from Σ to $\Sigma^\sharp$. The triple $(T, T^\sharp, \alpha)$ is a correct abstraction from Σ to $\Sigma^\sharp$ if*

$$\alpha^\uparrow \circ M \leq M^\sharp \circ \alpha^\uparrow \quad \textit{and} \quad \{\alpha(\sigma) \mid \sigma \in I\} \subseteq I^\sharp$$

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

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 $M \in \mathcal{M}_{\Sigma\times\Sigma}(Q)$ 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*

$$M^\sharp = \alpha^\uparrow \circ M \circ \alpha^\dagger \quad \textit{and} \quad I^\sharp = \{\alpha(\sigma) \mid \sigma \in I\}$$

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 $M^\sharp \leq M'$ and $I^\sharp \subseteq I'$.

Proof. The proof immediately follows from the facts that $Id \leq \alpha^\dagger \circ \alpha$ and $\alpha \circ \alpha^\dagger \leq Id$. □

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

Proposition 5 *Let consider the transition system $T = \langle q, I\rangle$ with $I \subseteq \Sigma$ its set of initial states and $q : \Sigma \times \Sigma \to Q$ its quantitative transition system in the cost semiring 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:*

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

then forall $t = \sigma_0 \to^{q_0} \ldots \sigma_n \in [\![T]\!]_{tr}, |t| = n$, there exists $t^\sharp = \sigma_0^\sharp \to^{q_0'} \ldots \sigma_n^\sharp \in [\![T^\sharp]\!]_{tr}, |t| = n$ such that $q_i \leq q_i' \quad \forall i \in [0, n-1]$ 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 $q^\sharp$.

3.4 Running example: abstraction

In 2.3, we introduced a quantitative semantics describing the number of cache misses in read access. M is the matrix describing this quantitative semantics for the factorial program (see Figure 2 for the code). The exact computation of the semantics would be too costly, even if we work with bounded numerical domains. In this subsection, we are using the abstractions techniques in order to compute an abstract semantics $M^\sharp$ from the matrix M.

We abstract a concrete state by the instruction pointer and the k last data accessed. Within this abstract domain, the loss of information lies in three points:

- Values (*i.e.* locals, stack and heap) are forgotten. This prevents us from determining the value of branching condition.
- The cache size is reduced to k elements. When k grows, precision increase, and so do the cost of the analysis.
- The method call stack is forgotten. We turn the analysis into an intra-procedural one, not for efficiency but for clearer notation, as our factorial function involves only one non-recursive function.

We write the abstract state as $(ip, [v_1, \ldots, v_k])$ where ip is the instruction pointer and $[v_1, \ldots, v_k]$ is a list of logical addresses of the last data accessed, v_k being the most recent. $\mathtt{s}.0$ refers to the bottom element of the local stack, $\mathtt{l}.\mathrm{f}$ refers to the local variable called x in the source code. We model the maximum number of read misses using the dioid $Q = (\overline{\mathbb{R}}, \mathtt{max}, +)$.

We construct the abstract matrix associated to our abstract system. Its size is bounded in terms of the cardinality of $\mathcal{I}$, the set of all instruction pointers appearing in program P, and the number of of up-to-k-combinations of the different logical data used in this function (which form a finite set $\mathcal{L}$). A value $q^\sharp$ of this matrix, standing at row $a^\sharp$ and column $b^\sharp$ ($a^\sharp$ and $b^\sharp$ are two abstract states), is computed in this way: let A and B be the set of concrete states abstracted by $a^\sharp$ and $b^\sharp$. Then $q^\sharp = \bigoplus\{q \mid a \rightarrow^q b, a \in A, b \in B\}$. For example

- $\sigma = (8, [\mathtt{l}.x]) \rightarrow^0 (9, [\mathtt{l}.x, \mathtt{s}.0]) = \sigma'$,
  Whatever the concrete state and its precise values, if it is abstracted by σ, then it can turn into a state abstracted by σ' for a cost of 0 read miss.
- $\sigma = (8, [\mathtt{s}.0]) \rightarrow^1 (9, [\mathtt{l}.x, \mathtt{s}.0]) = \sigma'$,
  In the same way, all states abtracted by σ can generate up to a read miss on their next instruction, turning into states abstracted by σ'.

We recall that in the factorial program, instruction 8 is `load` x. The load rule of the semantics describes these accesses to the cache.

$$[\mathit{read}_\tau(\mathtt{local}.f.i); \mathit{write}_\tau(\mathtt{stack}.f.t[+1])]$$

with i the local variable, t the current stack height and f the current frame.

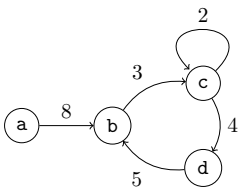
A $\perp$-transition denotes an incompatibility between the two abstract states, either in its control flow or its the cache evolution. Most of the matrix will be filled by $\perp$. This kind of matrix is called sparse matrix, and permits the use of particularly small representations together with efficient algorithms. We give here a submatrix of the abstract matrix. $M^\sharp \in \mathcal{M}_{(\mathcal{I} \times \{\emptyset \cup \mathcal{L} \cup \mathcal{L}^2\})^2}(Q)$.

$$M^\sharp = \left(\begin{array}{c|ccccc}
 & \ldots & 9, [\mathtt{l}.x, \mathtt{s}.0] & \ldots & 9, [\mathtt{l}.i, \mathtt{l}.x, \mathtt{s}.0] & \ldots \\
\hline
\vdots & & & & & \\
8, [] & & 1 & & \perp & \\
8, [\mathtt{l}.x] & & 0 & & \perp & \\
8, [\mathtt{s}.0] & & 1 & & \perp & \\
8, [\mathtt{s}.0, \mathtt{l}.x] & & 0 & & \perp & \\
8, [\mathtt{l}.x, \mathtt{s}.0] & & 0 & & \perp & \\
8, [\mathtt{l}.i, \mathtt{l}.x] & & \perp & & 0 & \\
8, [\mathtt{l}.i] & & \perp & & 1 & \\
\vdots & & & & &
\end{array}\right)$$

4 Long-run cost

So far, we have seen that all single-transition costs can be summarized in a transition matrix. We now use this matrix and the mathematical results of dioid algebra to define a notion of long run cost for a whole program. In [21] we proposed a notion of *global cost* of a program, representing its cost from initial to final states. It correctly deals with programs which 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 rather unsatisfactory as it does not tell anything about the concrete cost. For this case and for the case of programs which are not meant to terminate (as reactive systems), we propose the notion of *long-run cost*, that represents a maximal average cost over cycles of transitions. This terminology is taken from [1, 9], in the context of probabilistic processes modelled by Markov decision processes. Behaviour patterns of interest (described by labelled graphs) are associated to real numbers representing the success or the 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 (w.r.t. 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 *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 `cc` average cost $= 2/1 = 2$
Long-run cost $= \mathtt{max}(4, 2) = 4$

By the properties of the dioids we consider, 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 14 *Let P be a program having $T = \langle M, I\rangle$ for transition system. Let R be M restricted to the set of states, Σ, reachable from I. The long-run cost of program P is defined as the long-run cost of T*

$$\rho(P) = \rho(T) = \bigoplus_{k=1}^{|\Sigma|} \sqrt[k]{tr\, R^k}$$

where $tr\, R = \bigoplus_1^{|\Sigma|} R_{i,i}$

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 $(Time, \mathtt{max}, +)$,

where *Time* is isomorphic to $\overline{\mathbb{R}}$, $\rho(P)$ 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 the passing that its definition coincides with the definition of the maximum of eigenvalues of the matrix, in the case of an irreducible matrix in an idempotent semiring [17].

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 6 *Let Γ be the set of cycles appearing in the transition system T. Then*

$$\rho(T) = \bigoplus_{c \in \Gamma} \tilde{q}(c)$$

The idea of the proof is to show that the cycles of length less than $|\Sigma|$ are enough to know average costs, and that a partition of these cycles is related with the different iterates of the matrix appearing in Definition 14. The proof becomes straightforward in the case of an infinite set of states.

As we aim at giving 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 w.r.t. the cycles, this definition considers arbitrarily long traces. Unlike $\rho(P)$, $lrc(P)$ is not suitable for computation, even if the set of states is finite. We will see in Subsection 4.3 that those 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 to know if 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*

$$\rho(T) \leq_Q \rho(T^\sharp)$$

The proof of theorem relies on the fact that the correctness is preserved when the concrete and abstract matrices are iterated simultaneously. *Proof can be found in appendix D.*

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

Proposition 7 *If $(T, T^\sharp, \alpha)$ is a correct abstraction, then*

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

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 w.r.t. accessible cycles coincides with the notion of long-run cost defined as the limit of the maximum average cost of traces which 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 which do not belong to a cycle are such that:*

$$q(\delta) \neq +\infty$$

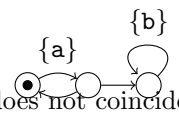
Hypothesis 1 excludes certain pathological matrices with atomic operations that have infinite costs. If a cycle contains a $+\infty$ transition, the ρ value indicates it.

Theorem 3 *The $T = \langle M, I\rangle$ 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 $+$, then with Hypothesis 1, we have*

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

This theorem establishes a link between the semantics and a computable definition of the long-run cost. The key points of the proof are to ensure that this limit exist, and to show that a small part of a trace can be neglected for very long traces. This is proved by bounding $\bigoplus \tilde{q}(t)$. *(Proofs of Proposition 6 and Theorem 3 can be found in appendix C.)*

A dioid which does not fulfill Theorem 3 is $(\mathcal{P}(S), \cap, \cup)$. This dioid may be used *e.g.* to analyse if certain portions of the code are executed for sure; however, in this setting, $\rho(T)$ does not coincide with the asymptotic behaviour of T. The diagram on the right illustrates this problem: While $\rho(T) = \emptyset$, all traces long enough include $\{\mathtt{a}\}$, and so does the intersection of their cost (the black dot is the initial state).



4.4 Running example: long-run cost

To illustrate the use of the long-run cost (ρ), we will consider a cache which can contain 4 integers; its replacement policy is still LRU. Such a small size could seem weird to the reader and unrealistic for a cache size, but the term cache can be interpreted here as some kind of registers. The semantics of the factorial program is abstracted as described in Section 3.4, with $k = 4$.

Using Definition 14, we compute $\rho(M^\sharp) = 2/9$, meaning that in an execution long enough, we observe on average 2 cache misses each 9 instructions. A

prototype is under developement for the standard Java bytecode, that already handles this kind of example. It implements the analysis technique presented in this paper with the help of an existing Scilab library for max-plus algebra, that offers an efficient ways to compute the long-run cost (through sparse matrices and Howard's algorithm for eigenvalues [10]).

Note that if we now consider a FIFO replacement policy for the same 4 integer registers, we obtain a different long-run cost. The FIFO policy implementation is cheaper in electronic components than the LRU one, but the analysis of the factorial function says that $\rho = 4/9$, *i.e.*, that we now have on average 4 cache misses for 9 instruction executed. Such a slowdown is coherent with the observations that small cache memory requires more advanced cache policies.

5 Related work

The present work is inspired by the quantitative abstract interpretation framework developed by Di Pierro and Wiklicky [14]. We have followed their approach in modeling programs as linear operators over a vector space, with the notable technical difference that their operators act over a semiring of probabilities whereas ours work with idempotent dioids. Working with idempotent dioids means that we have been able to exploit known results from Discrete Event Systems theory which makes intensive use of such structures. Another difference with respect to [14] lies in the kind of program being analyzed: we have been considering an intermediate bytecode language rather than declarative languages (probablistic concurrent constraint programming and the lambda calculus [13]).

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

Several other works make use of idempotent semiring for describing quantitative aspects of computations, namely under the form of constraint semirings [8]. Recently, these have been used in the field of Quality of Services [12], in particular with systems modelled by graph rewriting mechanisms [19]. In all these approaches, the $\oplus$ and $\otimes$ operators of the constraint semiring are used for combining constraints. In [4], Aziz makes use of semirings in a mobile process calculus derived from the π-calculus, in order 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 $(\{\mathit{low}, \mathit{medium}, \mathit{high}\}, \min, \max)$ one. This is similar to our use of dioids, but none of these approaches makes use of a notion of long-run cost to express an average quantitative behaviour of a system.

Our running example of estimating cache usage is meant for illustrative purposes and is based on a rather abstract view of cache analysis, compared

e.g. to the detailed modeling and cache abstraction of Alt, Ferdinand, Martin, and Wilhelm [2] who have proposed a cache behaviour prediction by abstract interpretation. In our proposition, the whole cache model is hidden behind the function *update*, which still has to be defined. Three points of their work could be almost directly used in our framework: the various models of cache (*e.g.* direct-mapped, A-way associative) to implement our update function, their abstract domain, in order to design our quantitative abstractions, and their observations about caches and writing, in order to develop an accurate model.

6 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 dioids. In such cases, we have shown that the semantics is a linear operator over the moduloid associated to this dioid. We have used a well-known characterization of the asymptotic behaviour of a discrete event system to define the notion of long-run cost of such a semantics, and proposed a novel way of analyzing the long-run behaviour of the program. We have characterized 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. The notions of dioids, quantitative semantics, abstraction and long-run cost have been illustrated all along the paper through a cache miss analysis on a program written in a simple bytecode language.

Future work The examples in the paper have been computed both by hand (for the abstraction part) and by a prototype analyzer for the computation of long-run costs themselves. Future work includes improvement of the prototype and developement of a framework for validating experimental results.

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, as opposed to the qualitative case where static analyses must err on the safe side. Work in this direction has been reported by Di Pierro and Wiklicky [15], where the difference between the concrete and abstract operator is estimated through a metric. However, such a notion still has to be defined for the dioids considered here.

References

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

[2] M. Alt, C. Ferdinand, F. Martin, and R. Wilhelm. Cache Behavior Prediction by Abstract Interpretation. In *Static Analysis Symposium (SAS'96)*,

volume 1145 of *Lecture Notes in Computer Science*, pages 51–66, September 1996.

[3] D. Aspinall, L. Beringer, M. Hofmann, H.-W. Loidl, and A. Momigliano. A Program Logic for Resources. *Theoretical Computer Science*. to appear.

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

[5] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity*. Wiley, 1992.

[6] P. Bertelsen. Dynamic Semantics of Java Bytecode. *Future Gener. Comput. Syst.*, 16(7):841–850, 2000.

[7] S. Bistarelli and F. Gadducci. Enhancing Constraints Manipulation in Semiring-Based Formalisms. In *European Conference on Artificial Intelligence*, pages 63–67, 2006.

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

[9] T. Brazdil, J. Esparza, and A. Kucera. 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*, pages 521–530, Washington, DC, USA, 2005. IEEE Computer Society.

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

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

[12] R. De Nicola, G. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A Basic Calculus for Modelling Service Level Agreements. In *International Conference on Coordination Models and Languages*, volume 3454 of *Lecture Notes in Computer Science*, pages 33 – 48, Namur (Belgium), April 2005. Springer Verlag.

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

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

[15] A. Di Pierro and H. Wiklicky. Measuring the Precision of Abstract Interpretations. In *LOPSTR 2000, Tenth International Workshop on Logic-based Program Synthesis and Transformation*, number 2042 in Lecture Notes in Computer Science, pages 147–164, 2001.

[16] S. N. Freund and J. C. Mitchell. A Formal Framework for the Java Bytecode Language and Verifier. *ACM SIGPLAN Notices*, 34(10):147–166, 1999.

[17] S. Gaubert. *Introduction aux systèmes dynamiques à événements discrets.* INRIA Rocquencourt, 1999.

[18] M. Gondran and M. Minoux. *Graphes, dioïdes et semi-anneaux.* Tec & Doc, Paris, 2001.

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

[20] I. Siveroni. Operational Semantics of the Java Card Virtual Machine. *J. Logic and Automated Reasoning*, 2004.

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

A Some rules of a bytecode quantitative semantics

We have developed a semantics for a small object-oriented bytecode language, inspired from the Carmel formalization of the Java Card instructions. We haven't dealt with features like exceptions and subroutines but we have treated enough instructions to provide a representative set of rules. Three of these rules are given in below.

Unconditional jump.

The `goto` a instruction causes an unconditional jump, within the current method. Execution proceeds at an offset a from the address of this `goto` instruction.

$$\frac{InstrAt(m, ip) = \texttt{goto}\ a}{< H, \ll f, m, ip, L, S \gg :: fr, C > \rightarrow^{0} < H, \ll f, m, ip + a, L, S \gg :: fr, C >}$$

Setfield for the current object.

The `setfield this` τ $i_{\texttt{s}}$ instruction sets the field, typed τ, pointed at by $i_{\texttt{s}}$, of the current object (`this`) to the top operand stack value. In Java, a reference to the current object is stored in the local variables at index 0.

$$\frac{\begin{array}{c} InstrAt(m, ip) = \texttt{setfield this}\ \tau\ i_{\texttt{s}} \\ S = v_\tau :: Sr \wedge L[0] = ref_{\texttt{a}} \\ S' = Sr \wedge H' = H \uplus \{ref_{\texttt{a}}(i_{\texttt{s}}) \mapsto v_\tau\} \wedge C' = update(C,access) \\ q = nbRmiss(C, access) \end{array}}{< H, \ll f, m, ip, L, S \gg :: fr, C > \rightarrow^{q} < H', \ll f, m, ip + 1, L, S' \gg :: fr, C' >}$$

$$\text{Where} \begin{cases} access = [read_\tau(\texttt{stack}.f.t); read_{\texttt{a}}(\texttt{local}.f.0); write_\tau(\texttt{heap}.ref_{\texttt{a}}.i_{\texttt{s}})] \\ size(S) = t \end{cases}$$

Typed return.

The `return` τ instruction ends the current method call by returning the control and a value of type τ to the invoker. The current stack frame is popped after its top operand has been placed on the operand stack of the invoker.

$$\frac{\begin{array}{c} InstrAt(m2, ip2) = \texttt{return}\ \tau \wedge S2 = v_\tau :: Sr2 \\ S1' = v_\tau :: S1 \wedge C' = update(C,access) \\ q = nbRmiss(C, access) \end{array}}{\begin{array}{c} < H, \ll f2,m2,ip2,L2,S2 \gg :: \ll f1,m1,ip1,L1,S1 \gg :: fr, C > \\ \rightarrow^{q} < H, \ll f1,m1,ip1 + 1,L1,S1' \gg :: fr, C' > \end{array}}$$

$$\text{Where} \begin{cases} access = [read_\tau(\texttt{stack}.f2.t2); write_\tau(\texttt{stack}.f1.t1[+1])] \\ size(S2) = t2 \wedge size(S1) = t1 \end{cases}$$

B Cost dioids

We first prove **Proposition 1** which states that in a cost dioid Q, we have:

(i) nth root is a $\oplus$-lsc: $\forall X \subseteq Q, \forall n > 0,\ \sqrt[n]{\bigoplus_{x \in X} x} = \bigoplus_{x \in X} \sqrt[n]{x}$,

(ii) $\forall a, b \in Q, \forall n, m > 0,\ \sqrt[n]{a} \oplus \sqrt[m]{b} \geq \sqrt[n+m]{a \otimes b}$.

Property (i) comes immediately with the definition of a cost dioid. We will need two lemmas to prove property (ii). The first one is a Cauchy's inequality.

Lemma 2 *In a cost dioid Q, we have:* $\forall n \in \mathbb{N},\ \forall x_1, \cdots, x_n \in Q,\ x_1 \otimes \cdots \otimes x_n \leq x_1^n \oplus \cdots \oplus x_n^n$.

Proof. $x_1 \otimes \cdots \otimes x_n$ 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 remaining of the expansion. We conclude with the definition of the order relation. □

Lemma 3 *In a cost dioid Q, we have:* $\forall a, b \in Q,\ a \leq b \Leftrightarrow a^n \leq b^n$.

Proof. ($\Rightarrow$) This is just the expression of the fact that in a dioid, the nth power is monotone, which is a direct consequence of Lemma 1.
($\Leftarrow$) In a cost dioid, the nth root is a $\oplus$-morphism. Thus it is monotone [5]. □

We now prove property (i) of Proposition 1.

Proof. By raising inequality

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

to the power $mn(m+n)$ and by applying Lemma 3, we get

$$a^{mn} \otimes b^{mn} \leq a^{m(m+n)} \oplus b^{n(m+n)}$$

This latter inequality is a Cauchy's inequality. Indeed, $a^{mn} = a^m \otimes \cdots \otimes a^m$ (n times) and $b^{mn} = b^n \otimes \cdots \otimes b^n$ (m times). By Lemma 2, we thus have

$$a^{mn} \otimes b^{mn} = a^m \otimes \cdots \otimes a^m \otimes b^n \otimes \cdots \otimes b^n \leq (a^m)^{m+n} \oplus (b^n)^{m+n}.$$

□

□

We now prove **Proposition 2** which states that the following dioids are cost dioids.

(1) Complete and selective commutative dioids equipped 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 $X^n = q$ always has at least one solution.

Proof. In a selective dioid, for all subset X of Q, there exists y in X such that $\bigoplus_{x\in X} x = y$. The nth power is thus $\oplus$-lsc, which proves that dioids of class (1) are cost dioids.

In a double-idempotent dioid, the nth power is the identity function. Thus, the nth power is $\oplus$-lsc.

For dioids of classes (1) and (3), we first prove that if the dioid is equipped with an nth root operation, then the nth power is $\oplus$-lsc if and only if it is a $\oplus$ morphism. We naturally just have to prove the *if* part.

A FINIR (tableau)

Now, for dioids of class (1), we can immediately conclude, since the nth power is a $\oplus$ morphism and we already assume the existence of a unique nth root for each element.

For dioids of class (3), we first have to prove that if the equation $X^n = q$ has a solution, then this solution is unique. First of all, note that if $q = \perp$ the equation above becomes $x^n = \perp$. The unique solution of this equation is $\lambda = \perp$. Indeed, $\lambda = \perp$ is solution and if $\lambda \neq \perp$ is solution then $\lambda^n = \perp = \perp \otimes \lambda$ which proves by cancellation that $\lambda^{n-1} = \perp$ and by iterating this process, $\lambda = \perp$. Let us now assume that $q \neq \perp$. 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 $\lambda_1 \neq \perp$, $\lambda_2 \neq \perp$. Lemma 2 yields

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

and then

$$\lambda_1^{n-1} \otimes \lambda_2 \leq \lambda_1^n$$

We thus have $\lambda_2 \leq \lambda_1$ by successive cancellations by λ_1. The same reasoning applies for proving $\lambda_1 \leq \lambda_2$. Thus, $\lambda_1 = \lambda_2$ and the equation $X^n = q$ has a unique solution.

We now prove that, for dioids of class (3), we have

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

We first note that if $a = \perp$ then $(a \oplus b)^n = b^n$, thus the equality trivially holds if $a = \perp$ or $b = \perp$. Let us now assume that $a \neq \perp$ and $b \neq \perp$ and proceed by induction on n. The wanted property trivially holds for $n = 0$ and $n = 1$. Let us assume it for $n \geq 1$, and prove it at rank $n + 1$. We have

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

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

We then just have to prove equality of terms $(*)$ and $(**)$. If it holds, we will indeed have $(a \oplus b)^{n+1} \otimes (a \oplus b) = (a^{n+1} \oplus b^{n+1}) \otimes (a \oplus b)$. As $a \oplus b \neq \perp$, the cancellation hypothesis allows for simplifying by $(a \oplus b)$, which establishes the property at rank $n+1$. Let us thus show this equality of $(*)$ and $(**)$. We have $a^2 b^n \oplus a^n b^2 = ab(ab^{n-1} \oplus a^{n-1}b)$. Moreover, the induction hypothesis gives $a^n \oplus b^n = (a \oplus b)^n = ab^{n-1} \oplus a^{n-1}b \oplus a^n \oplus b^n \oplus (\oplus_{k=2}^{n-2} a^k b^{n-k}$, which in turn

yields $ab^{n-1} \oplus a^{n-1}b \leq a^n \oplus b^n$. Since $\otimes$ preserves the order, we get the wanted result. This concludes the proof that dioids of class (3) are cost dioids. $\square$

$\square$

C Trace semantics and long run cost

This appendix contains the proof that the two alternatives formulations of long run cost given in Section 4 (Proposition 6 and Theorem 3) are equivalent to the original definition (Definition 14):

> Let P be a program having $T = \langle M, I\rangle$ for transition system. Let R be M restricted to the set of states, Σ, reachable from I. The long-run cost of program P is defined as the long-run cost of T
> $$\rho(P) = \rho(T) = \bigoplus_{k=1}^{|\Sigma|} \sqrt[k]{tr\, R^k}$$
> where $tr\, R = \bigoplus_1^{|\Sigma|} R_{i,i}$

We first consider Proposition 6 which states

> Let Γ be the set of cycles appearing in the transition system 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 $\pi, \tilde{q}(\pi)$ stands for $\sqrt[|\pi|]{q(\pi)}$. To prove the proposition, we first establish Lemma 4 and Lemma 5.

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

Lemma 4
$$\forall c \in \Gamma, \exists \Gamma_c \subseteq \Gamma_{\leq|\Sigma|}, \quad \tilde{q}(c) \leq \bigoplus_{c_e\in\Gamma_c} \tilde{q}(c_e)$$

Proof. By strong induction on the length of c,

- If $|c| \leq |\Sigma|$ (this holds in particular when Σ is infinite), then $\Gamma_c = \{c\}$ and the inequality trivially holds.
- If $|c| > |\Sigma|$, then there exists a state σ' such that
$$c = \sigma \overset{\pi_1}{\Rightarrow} \sigma' \overset{\pi_2}{\Rightarrow} \sigma' \overset{\pi_3}{\Rightarrow} \sigma \quad \text{with} \begin{cases} \pi_1\pi_3 \in \Gamma & \wedge \quad |\pi_1\pi_3| < |c| \\ \pi_2 \in \Gamma & \wedge \quad |\pi_2| < |c| \end{cases}$$
With these notations, we have
$$\begin{aligned} \tilde{q}(c) = \tilde{q}(\pi_1\pi_2\pi_3) &= \sqrt[|c|]{q(\pi_1\pi_2\pi_3)} \\ &= \sqrt[|c|]{q(\pi_1\pi_3) \otimes q(\pi_2)} && (1) \\ &\leq \sqrt[|\pi_1\pi_3|]{q(\pi_1\pi_3)} \oplus \sqrt[|\pi_2|]{q(\pi_2)} = \tilde{q}(\pi_1\pi_3) \oplus \tilde{q}(\pi_2) && (2) \end{aligned}$$
Inequality (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 ii of 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 5

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

Proof. We add some elements to the right-side of the inequality of Lemma 4, complementing the sum up to $\Gamma_{\leq|\Sigma|}$ (remind that $a \leq b \Rightarrow a \leq b \oplus c$). Hence

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

Summing for all $c \in \Gamma$ (remind the idempotency of $\oplus$), we obtain that the maximal average transition cost for all cycles is lower or equal than the maximal average transition cost of a bounded subset of cycles, those no longer than $|\Sigma|$.

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

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

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

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

□

We now prove Proposition 6 itself.

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

$$\Gamma_{\leq|\Sigma|} = \bigcup_{\substack{n \leq |\Sigma| \\ \sigma \in \Sigma}} \mathcal{C}_n^\sigma$$

We have

$$\bigoplus_{c\in\Gamma_{\leq|\Sigma|}} \tilde{q}(c) = \bigoplus_{c\in\Gamma_{\leq|\Sigma|}} \sqrt[|c|]{q(c)} = \bigoplus_{n\leq|\Sigma|} \bigoplus_{\sigma\in\Sigma} \bigoplus_{c\in\mathcal{C}_n^\sigma} \sqrt[n]{q(c)}$$

$$= \bigoplus_{n\leq|\Sigma|} \sqrt[n]{\bigoplus_{\sigma\in\Sigma} \bigoplus_{c\in\mathcal{C}_n^\sigma} q(c)}$$

This step is allowed by Property i of Proposition 1.

If σ is reachable, then all states in path starting from σ are reachable. Let R be matrix M where indices are restricted to Σ, the reachable states of T. We finally have:

$$R^n_{\sigma,\sigma} = \bigoplus_{c \in \mathcal{C}^\sigma_n} q(c)$$

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

□

□

We now prove Theorem 3, which states

> The $T = \langle M, I\rangle$ 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 $+$, 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 with "usual" arithmetic, in order to be 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 $\leq$ may be interpreted either by `max` and $\leq$ or by `min` and $\geq$.

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 $n \geq N$, we can decompose t

$$t = p_0.c_0.p_2 = 1.c_1 \ldots c_{k-1}.p_k \text{ where } \begin{cases} \text{each } c_i \text{ is a cycle} \\ r = \sum_i |p_i| < N \\ \text{each } p_i \text{ is acyclic}\end{cases}$$

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

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

Proposition 6 says that the mean cost per transition in the cycles of T is upper-bounded by the long run cost of T, $\rho(T)$. For a given transition system whose graph contains cycles, let q_- and q_+ be the respective lower and upper bounds of all one-step reachable transition costs of T. By definition of a possible transition, $q_- \neq \bot$ and by hypothesis, $q_+ \neq \top$. We have

$$q(t) = (q(p) + q(s)) + q(c) \leq r.q_+ + (n - r).\rho(T) \tag{5}$$

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

$$q(t) = (q(p) + q(s)) + q(c) \leq N.\,|q_+| + (n - A).\rho(T) \tag{6}$$

A being either N or $-N$, depending on the sign of $\rho(T)$. We sum (6) for all t of size n to get (7).

$$\bigoplus_{t \in [\![T]\!]^n_{tr}} q(t) \leq N.\,|q_+| + (n - A).\rho(T) \tag{7}$$

This gives the upper bound.

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

$$q(t_{\mathtt{max}}) \leq \bigoplus_{t \in [\![T]\!]^n_{tr}} q(t) \tag{8}$$

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

$$\begin{aligned} q(t_{\mathtt{max}}) &= (q(p) + q(s)) + q(c) \\ q(t_{\mathtt{max}}) &\geq N.\,|q_-| + (n - A).\rho(T) \end{aligned} \tag{9}$$

This inequality is guaranteed by the fact that $r.q_-$ is a lower bound for $q(p.s)$, and that $(n - r).\rho(T)$ is the exact cost of path c. Inequalities (8) and (9) lead to

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

We combine inequalities (10) and (7) to get

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

which turns into

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

by dividing by n. Finally, as $\frac{a}{n} \oplus \frac{b}{n} = \frac{a \oplus b}{n}$, this yields

$$F(n) \leq \bigoplus_{t \in [\![T]\!]^n_{tr}} \frac{q(t)}{n} \leq G(n) \tag{11}$$

We now study the limit behavior of (11) when n tends to infinity.

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

It implies

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

written with "usual" arithmetic, *i.e.* using the semiring arithmetical notations

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

□

D Correctness of the long-run cost

We prove Theorem 2 which states that in the case of a correct abstraction $(\langle M, I\rangle, \langle M^\sharp, I^\sharp\rangle, \alpha)$, (Definition 13), we compute a correct long-run cost (Definition 14), *i.e.*

$$\alpha \circ M \leq M^\sharp \circ \alpha \Rightarrow \rho(M) \leq_Q \rho(M^\sharp)$$

We first prove Lemma 6 and 7.

Lemma 6 *If $\alpha^\uparrow \circ M \leq M^\sharp \circ \alpha^\uparrow$ then $\alpha^\uparrow \circ M^n \leq (M^\sharp)^n \circ \alpha^\uparrow$.*

Proof. Lemma 6 states that:

$$\forall n \geq 1 \qquad \alpha \circ M \leq M^\sharp \circ \alpha \Rightarrow \alpha \circ M^n \leq_Q (M^\sharp)^n \circ \alpha \tag{12}$$

We prove (12) by induction on n. The case where $n = 1$ is trivial. We then assume that $\alpha \circ M \leq M^\sharp \circ \alpha$ and $\alpha \circ M^n \leq (M^\sharp)^n \circ \alpha$. Lemma 1 gives that

$$(\alpha \circ M^n) \circ M \leq ((M^\sharp)^n \circ \alpha) \circ M$$

We then have

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

□

□

Lemma 7 *For all correct abstraction $(\langle M, I\rangle, \langle M^\sharp, I^\sharp\rangle, \alpha)$ and α lifted as stated in Section 3,*

$$\forall \sigma \in \Sigma, \sigma^\sharp \in \Sigma^\sharp, \qquad \alpha(\sigma) = \sigma^\sharp \Rightarrow M_{c,c} \leq_Q (M^\sharp)_{d,d}$$

Proof. We have $\alpha^\uparrow \circ M \leq M^\sharp \circ \alpha^\uparrow$, and so in particular $(\alpha^\uparrow \circ M)_{\sigma^\sharp,\sigma} \leq_Q (M^\sharp \circ \alpha^\uparrow)_{\sigma^\sharp,\sigma}$ which rewrites into

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

Decomposing both summations, this yields

$$\begin{aligned}
&\bigoplus_{\sigma_i \in \alpha^{-1}(\sigma^\sharp)} (\alpha_{\sigma^\sharp,\sigma_i} \otimes M_{\sigma_i,\sigma}) \oplus \bigoplus_{\sigma_i \notin \alpha^{-1}(\sigma^\sharp)} (\alpha_{\sigma^\sharp,\sigma_i} \otimes M_{\sigma_i,\sigma})\\
&\qquad \leq_Q ((M^\sharp_{\sigma^\sharp,\sigma^\sharp}) \otimes \alpha_{\sigma^\sharp,\sigma}) \oplus \bigoplus_{\sigma_i^\sharp \neq \sigma^\sharp} ((M^\sharp_{\sigma^\sharp,\sigma_i^\sharp}) \otimes \alpha_{\sigma_i^\sharp,\sigma})
\end{aligned}$$

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

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

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

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

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

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

□

□

We now prove Theorem 2 itself.

Proof. By combining Lemma 6 and 7, we have for any $k \geq 1$,

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

$$tr\, M^k \leq_Q tr(M^\sharp)^k \tag{13}$$

$$\sqrt[k]{tr\, M^k} \leq_Q \sqrt[k]{tr(M^\sharp)^k} \tag{14}$$

Inequality 13 simply uses the definition of a trace. Inequality 14 holds because the nth root operation is order preserving. Summing this last inequality for all k from 1 to $|\Sigma|$ yields

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

and hence we have that $\rho(M) \leq_Q \rho(M^\sharp)$ □

□

We prove Proposition 5, which states:

> Let consider the transition system $T = \langle q, I \rangle$ with $I \subseteq \Sigma$ its set of initial states and $q : \Sigma \times \Sigma \to Q$ its quantitative transition system in the cost semiring 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:
>
> - $I^\sharp = \{\alpha(\sigma) \mid \sigma \in I\}$
> - $\alpha^{-1} : \Sigma^\sharp \to \mathcal{P}(\Sigma)$ with $\alpha^{-1}(\sigma^\sharp) = \{\sigma \mid \alpha(\sigma) = \sigma^\sharp\}$
> - $q^*(\Sigma_1, \Sigma_2) = \bigoplus_{(\sigma_1,\sigma_2) \in \Sigma_1 \times \Sigma_2} q(\sigma_1, \sigma_2)$
> - $q^\sharp(\sigma^\sharp_1, \sigma^\sharp_2) = q^*(\alpha^{-1}(\sigma^\sharp_1), \alpha^{-1}(\sigma^\sharp_2))$
>
> then forall $t = \sigma_0 \to^{q_0} \ldots \sigma_n \in [\![T]\!]_{tr}, |t| = n$, there exists $t^\sharp = \sigma^\sharp_0 \to^{q'_0} \ldots \sigma^\sharp_n \in [\![T^\sharp]\!]_{tr}, |t| = n$ such that $q_i \leq q'_i \quad \forall i \in [0, n-1]$ and $\sigma^\sharp_i = \alpha(\sigma_i) \quad \forall i \in [0, n]$. In addition, $M^\sharp = \alpha^\uparrow \circ M \circ \alpha^\dagger$ is the transition matrix for $q^\sharp$.

Proof. Let t be $\sigma_0 \rightarrow^{q_0} \ldots \sigma_n$. $t \in [\![\langle q, I\rangle]\!]_{tr}$. We choose $t^\sharp = \sigma_0^\sharp \rightarrow^{q'_0} \ldots \sigma_n^\sharp$ with $\sigma_i^\sharp = \alpha(\sigma_i), \forall i \in [0, n]$ and $q'_i = q^\sharp(\sigma_i^\sharp, \sigma_{i+1}^\sharp), \forall i \in [0, n-1]$. We trivially have $|t| = |t^\sharp|$. We have $q'_i = q^\sharp(\alpha(\sigma_i), \alpha(\sigma_{i+1})) = q^*(\alpha^{-1} \circ \alpha(\sigma_i), \alpha^{-1} \circ \alpha\sigma_{i+1})$. By definition of α^{-1}, we have $x \subseteq \alpha^{-1} \circ \alpha(x)$, so:

$$\begin{aligned} q'_i = q(\sigma_i, \sigma_{i+1}) \oplus & \bigoplus_{(\sigma_1,\sigma_2) \in \begin{array}{c}(\alpha^{-1} \circ \alpha(\sigma_i) \times \alpha^{-1} \circ \alpha(\sigma_{i+1})) \\ -\{(\sigma_i, \sigma_{i+1})\}\end{array}} q(\sigma_1, \sigma_2) \\ \geq_Q q(\sigma_i, \sigma_{i+1}) = q_i & \end{aligned}$$

We eventually 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 $i \in [0, n-1]$, $q^\sharp(\sigma_i^\sharp, \sigma_{i+1}^\sharp) = q'_i \geq_Q q_i >_Q \perp_Q$. □

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

□

We prove Proposition 7:

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

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

Proof. For all n, we have: $\forall t = \sigma_0 \rightarrow^{q_0} \ldots \sigma_n \in [\![T]\!]_{tr}, |t| = n, \exists t^\sharp = \sigma_0^\sharp \rightarrow^{q'_0} \ldots \sigma_n^\sharp \in [\![T^\sharp]\!]_{tr}$ such as described in Proposition 5. Since for all $i \in [0, n-1]$, $q_i \leq_Q q'_i$, we have $q(t) \leq_Q q^\sharp(t^\sharp)$. In the dioid, $a \leq b \wedge c \leq d \Rightarrow a \otimes c \leq b \otimes d$, so $\bigotimes_{i=0}^{n-1} q_i \leq \bigotimes_{i=0}^{n-1} q'_i$. The function $\sqrt[n]{\ }$ being a $\oplus$-morphism we also have $\sqrt[n]{q(t)} \leq_Q \sqrt[n]{q^\sharp(t^\sharp)}$. The dioid Q being complete, we can sum this inequality for all trace t of length n. Hence, it exists a set $\Pi_n^\sharp \subseteq [\![T^\sharp]\!]_{tr}$ such that:

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

The last step is due to $a \leq b \Rightarrow a \leq b \oplus c$ in the dioid. This inequality being true for all n, we take its superior limit and prove the proposition. □

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399