VODCA 2004 `Preliminary Version`

A Logic for Graphs with QoS

Gianluigi Ferrari [1,2] and Alberto Lluch-Lafuente [1,3]

Dipartimento di Informatica
Università di Pisa

Abstract

We introduce a simple graph logic that supports specification of *Quality of Service* (QoS) properties of applications. The idea is that we are not only interested in representing whether two sites are connected, but we want to express the QoS level of the connection. The evaluation of a formula in the graph logic is a value of a suitable algebraic structure, a c-semiring, representing the QoS level of the formula and not just a boolean value expressing whether or not the formula holds. We present some examples and briefly discuss the expressiveness and complexity of our logic.

Key words: Graph Logics, QoS, Semirings, Distributed Systems.

---

1 Introduction

Graphs and graphs transformation systems [19] are a suitable formalism for systems involving issues such as concurrency, distribution and mobility. The graphical nature of such systems also appears in other modeling formalism including algebras for communicating processes [24], where the implicitly communication structure can be seen as a graph.

The properties of such systems mainly regard aspects such as behavior in time and structural properties, that can be expressed by logics which are used as a basis for a formal verification method, like model checking [12], which have been shown to be surprisingly effective in practice [11].

When reasoning about wide area network applications another issue becomes crucial, namely *Quality of Service* (QoS). QoS refers to non-functional aspects, like network bandwidth, time response or security degree. A clear example of this are *Service Overlay Networks* [18] (SON). The notion of SON

---

[1] This work has been supported by the European Research Training Network SEGRAVIS and the European FET project PROFUNDIS.

[2] Email: `giangi@di.unipi.it`

[3] Email: `lafuente@di.unipi.it`

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

has gained major attention during recent years as a flexible mechanism to manage the complexity in the creation and deployment of network services with certain Quality of Services assurances. A SON abstracts from the Internet complexity, such as packet routing across different autonomous systems, and considers network services as connected by virtual or application-level links: the overlay network. The application level links correspond to end-to-end Internet connections and are associated with QoS parameters. Moreover, service functionalities are described together with their *Service Level Agreement* (SLA) contracts. The SON abstraction has the main benefit of modeling a logical end-to-end service infrastructure with certain QoS guarantees.

The development of applications over SON requires evolutionary software design methodologies. Standard approaches describe SON architectures through labeled directed graphs where labels of edges represent QoS assurances and labels of nodes are SLA constraints. Much research work adopts graph theoretic models to describe the delivery of services over a SON. See [22] for instance. However, according to our knowledge, little has been done to develop effective verification techniques to support formal verification of applications over SON.

A challenging problem is to provide formal machineries that supports verification of both behavioral and QoS properties of SON in an integrated way. This paper provides a first step towards this goal. Here, we elaborate a logical framework that allows to express and reason about QoS properties of graphs.

Related Work.

Our approach builds on previous work on an extension of temporal logics to non-boolean domains [23], and research on a spatial logic for graphs (GL) [8,15].

Graph logics allow to express properties of graphs. They have been exhaustively investigated by Courcelle (see [14] for instance), whose work is mainly based on the *monadic second order logic* on graphs (MS) and its fragments. Spatial logics are used to reason about the structure of models, such as heaps [26], trees [9], processes calculi [7,6] and graphs [8,15]. The common concept in such approaches is that if a notion of model composition exists (like parallel composition in process calculi) one can reason about decompositions in the corresponding logic. The usual way is via a composition operator $|$, where $\phi|\psi$ is satisfied by models that can be decomposed in two sub-models, one satisfying ϕ and another one satisfying ψ. In graphs, composition is closely related with the second-order quantification over set of edges used in MS. Indeed, the expressive power of the fixpoint-free fragment of GL have been shown to be included in MS [15]. The full logic, on the other hand, is able to express properties unlikely to be represented in MS [15]. It is an open question whether GL subsumes MS.

Temporal logics support reasoning about the behavior of systems in time. For instance, one can ask whether a certain event never happens or whether an event happens infinitely often. Temporal logics are usually interpreted over transition systems in a boolean way, i.e. the evaluation of a formula is either *true* or *false*. However, some approaches exist which interpret temporal logics over more general domains like boolean algebras [10] or probabilities [16]. There is also a vast number of works regarding the analysis and verification of systems where the focus is on probabilities and time. We cite among others the work on CSL [1], a logic that combines these two aspects.

Contrary to such works we do not concentrate on specific issues like time or probabilities but rather consider an abstract representation of QoS by means of a suitable algebraic structure. More precisely, our own contribution to this field of *quantitative* temporal logics is described in [23], were we propose temporal logics like the μ-calculus to be defined over constraint-semirings, a formalism for QoS.

Constraint semirings (c-semirings) are algebraic structures consisting of a domain and two operations called *additive* and *multiplicative* satisfying some properties. The basic idea is that the former is used to select among values and the latter to combine values. Different kind of semirings have been proposed to model costs, for instance closed semirings in the algebraic path problem [27] or (max,+) algebras for different applications [21]. C-semirings are a specific kind of semirings which are sufficient to capture most of the significant QoS attributes used in practice. They were originally proposed as a suitable structure to describe and program constraints problems [4] including the analysis of security protocols [3]. The basic idea of the approach is that the additive operation of the semiring is used to project constraints while the multiplicative operation is used to combine constraints.

Contribution.

In this paper we introduce and analyze the properties of a simple graph logic defined over c-semirings. In particular, we extend the spatial logic of [15]. The choice of this logic is mainly motivated for the recent interest in spatial logics. Most of the ideas we present, however, could be applied to other graph logics as well. We introduce functions that associate c-semiring values to vertices and edges of a graph. Logical constants and connectives are substituted by c-semiring constants and functions. As a result the evaluation of a formula in the extended logic is a value of the c-semiring, representing the QoS assurance of the formula and not just a boolean value expressing whether or not the formula holds. Instead of expressing the existence of a certain path in the graph, for example, we can express the QoS of the optimal path in the graph.

The main technical contribution of this paper is the definition of the logic. We show how to express properties through a running example: the *arrow distributed directory protocol* [17] which ensures exclusive access to a mobile service in a distributed system. Complexity and expressivity properties of our logic are discussed in an informal way.

Structure.

Section 2 introduces our motivations with an example. Technical background describing graphs, c-semirings and their properties is contained in Section 3. Section 4 presents syntax and semantics of our logic together with an example. A next section outlines potential applications of our approach. Finally we conclude the paper sketching future research avenues.

2 The Arrow Distributed Directory Protocol

The *arrow distributed directory protocol* [17] is a solution to ensure exclusive access to mobile objects in a distributed system. The distributed system is given as an undirected graph G, where vertices and edges respectively represent nodes and (reliable) communication links. Costs are associated with links in the usual way, and a mechanism for optimal routing is assumed.

The protocol works with a minimal spanning tree T of G. Each node has an arrow which, roughly speaking, indicates the *direction* in which the object lies. If a node owns the object the arrow points to itself, we say that both the arrow and the node are *terminal*. The directed graph induced by the arrows is called $\mathcal{L}$. Roughly speaking the protocol works by propagating requests and updating arrows such that at any moment the arrows either lead to a terminal owning the object or waiting for it.

More precisely, the protocol works as follows: Initially $\mathcal{L}$ is set such that every path leads to the node owning the object. When a node u wants to acquire the object, it sends a request message $\mathit{find}(u)$ to $a(u)$, the target of the arrow starting at u, and sets $a(u)$ to u, i.e. it becomes a terminal node. When a node u whose arrow does not point to itself receives a $\mathit{find}(w)$ message from a node v, it forwards the message to node $a(u)$ and sets $a(u)$ to v. On the other hand, if $a(u) = u$ (the object is not necessarily at u but will be received if not) the arrows are updated as in the previous case but this time the request not forwarded but enqueued. If a node owns the object and its queue of requests is not empty, it sends the object to the (unique) node u of its queue sending a $\mathit{move}(u)$ message to v. This message goes *optimally* through G. A formal definition of the protocol can be found in [17].

Hence, the protocol works over a SON, which offers different services: sending of requests through links of the minimal spanning tree and delivering of the object. Each of the services has an associated QoS. For example, each propagation link might have a delay, while the delivering of the object may have a price.

Figure 1 illustrates two states of protocol. Nodes v_1,..,v_5 and the object o are represented by vertices. Arrows are denoted by a-labeled edges. An h-labeled edge from the object to a node represents that the node has the object. Other edges of the protocol, like those representing the delivering of the object, those representing the queues of requests and those forming the minimal spanning tree are not depicted for the sake of simplicity. The state

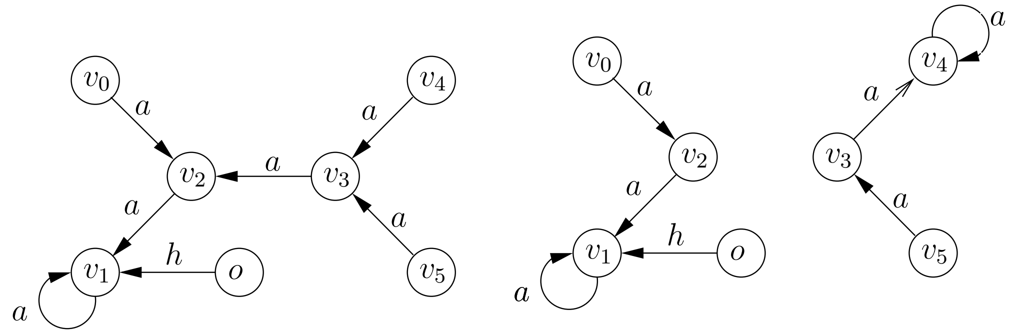

Fig. 1. Two states of the directory: initial (left) and after v_3 processes a request from v_4 (right).

on the left is the initial one: node v_1 has the object and all paths induced by the arrows lead to it. The state on the right of the figure is the result of two steps: 1) node v_4 sends a request for the object through its arrow and 2) v_3 processes it by updating the arrows properly, i.e. the arrow points now to v_4 instead of v_2.

The protocol has been shown to have some properties. The first one [17, Theorem 5] is stated as follows:

Property 1 *In every state of the directory, the maximal path from any node induced by the non-terminal arrows is unique and always leads either to the owner of the object or to a node that has requested it (a* waiter*).*

This property involves spatial and temporal aspects. It requires a certain kind of paths to exist in every state of the protocol, but it says nothing about QoS properties. Note that the algorithm works over a SON, where each service might be offered with a certain QoS. Hence we might be interested in reasoning about QoS, for instance, in specifying the cost such paths.

3 Preliminaries

3.1 C-Semirings.

Our logic is defined over the domain of a c-semiring. We give first the definition and postpone the intuition behind it for next lines showing some examples. Formally a c-semiring is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that:

- A is a set;
- $\mathbf{0}$ and $\mathbf{1}$ are elements of A;
- $+ : 2^A \to A$ is defined over (possibly infinite) sets of elements of A as follows[4]: $\sum\{a\} = a$, $\sum\emptyset = \mathbf{0}$, $\sum A = \mathbf{1}$ and $\sum(\bigcup A_i) = \sum\{\sum A_i\}$, for $A_i \subseteq A$, $i \geq 0$;

---

[4] When $+$ is applied to a set with two elements we use $+$ as binary operator in infix notation, while in all other cases we use symbol $\sum$ in prefix notation.

- $\times : A \times A \to A$ is a binary associative, commutative operation that distributes over $\mathbf{+}$, has $\mathbf{1}$ as unit element and $\mathbf{0}$ as absorbing element.

The fact that $\mathbf{+}$ is defined over sets of elements, automatically assumes it to be associative, commutative and idempotent. Moreover, one can show that it has $\mathbf{0}$ as unit element and $\mathbf{1}$ as absorbing element [4]. In the rest of the paper we assume that $\times$ is defined over infinite sequences too and use symbol $\prod$ in postfix notation. Most of the c-semirings used in practice satisfy this.

To enhance readability, operation $\mathbf{+}$ is called *additive operation*, while $\times$ is called *multiplicative operation*. Note that we use a boldfaced $\mathbf{+}$ and symbol $\times$ to avoid confusion with the additive and multiplicative operations over reals ($+$ and $\cdot$).

Examples and application to QoS.

C-semirings are the formal structure of many QoS attributes. For example:

- The boolean c-semiring $\langle\{\mathit{true}, \mathit{false}\}, \vee, \wedge, \mathit{false}, \mathit{true}\rangle$ can be used to model network and service availability.
- The optimization c-semiring $\langle\mathbb{R}^+, \mathit{min}, +, +\infty, 0\rangle$ and the tropical c-semiring $\langle\mathbb{N}^+, \mathit{min}, +, +\infty, 0\rangle$ apply to a wide range of cases, like prices or propagation delay.
- Bandwidth can be formalized using the max/min c-semiring which is given by $\langle\mathbb{R}^+, \mathit{max}, \mathit{min}, 0, +\infty\rangle$.
- Performances can be represented by means of the probabilistic c-semiring $\langle[0,1], \mathit{max}, \cdot, 0, 1\rangle$ or the fuzzy one $\langle[0,1], \mathit{max}, \mathit{min}, 0, 1\rangle$.
- Set-based $\langle 2^N, \cup, \cap, \emptyset, N\rangle$ (set-based, where N is a set): Capabilities and access rights.
- Security degrees are modeled via the c-semiring $\langle[0,1,..,n], \mathit{max}, \mathit{min}, 0, n\rangle$, where n is the maximal security level (unknown) and 0 is the minimal one (public) [3].

C-Semirings and Lattices.

The additive operation of the c-semiring induces a partial order as follows: $a \leq_S b$ iff $a \mathbf{+} b = b$. For example, in the optimization c-semiring, $\leq_S$ corresponds to the arithmetic relation $\geq$. One can show that $\leq_S$ is indeed a partial order, that $\mathbf{+}$, $\times$ are monotone over $\leq_S$, $\mathbf{0}$ and $\mathbf{1}$ are respectively the minimum and maximum elements of $\leq_S$, and $\langle A, \leq_S\rangle$ is a complete lattice [4].

In some instances of a c-semiring, the multiplicative operation is idempotent. This implies, among other things, that $\mathbf{+}$ distributes over $\times$ and $\langle A, \leq_S\rangle$ is a distributive lattice [4]. The logical and fuzzy c-semirings are an example of this, since logical disjunction is idempotent, i.e. $p \wedge p$ is p. To the contrary, the multiplicative operations of the optimization and probabilistic c-semirings, i.e. addition and multiplication of reals are not idempotent. In most cases, like in all the examples presented, $\langle A, \leq_S\rangle$ is distributive.

Negation in C-Semirings.

C-Semirings have no complement or negation operator in general. Consider, for instance, the classical De Morgan negation, i.e. a bijective operator $\text{-} : A \to A$, such that $\text{-}a \in A$ and $\text{--}(a) = a$ for all $a \in A$ (involution), and $\text{-}\bigsqcup\{A'\} = \bigsqcap\{\text{-}a \mid a \in A'\}$ for all $A' \subseteq A$ (De Morgan) and $a \leq b \Leftrightarrow \text{-}b \leq \text{-}a$ (antimonotonicity), where $\bigsqcup$ and $\bigsqcap$ are the *lowest upper bound* and *greatest lower bound* operators of the lattice $\langle A, \leq_S \rangle$. It is not possible to define a De Morgan negation for the tropical c-semiring. Suppose the contrary and let n denote -1. By antimonotonicity $-(n+1) < 1$ but this is only possible if $n = +\infty$, which will imply $-1 = +\infty$. Since $-0 = \infty$, $-$ is not bijective. Other c-semirings (fuzzy, boolean, max/min), however, can be equipped with a negation, resulting in most cases in boolean algebras.

Composition of C-Semirings.

C-semiring based methods have a unique advantage when problems with multiple QoS criteria must be tackled. In fact, it turns out that Cartesian products, exponentials and power constructions of c-semirings are c-semirings. Thus the same concepts and algorithms can be applied again and again. The Hoare Power Domain of a c-semiring is of special interest since it let us formalize multi-criteria optimization problems. The Cartesian product of c-semirings is not enough to solve such problems. The problem lies in the fact that in the Cartesian product, values of the original c-semirings are combined independently. The solution is to use the Hoare power domain of the Cartesian product of the various optimization c-semirings as explained in [5]. Intuitively, the Hoare Power Domain works with sets of non-dominated values.

3.2 Graphs.

Graphs in [8] are described by terms in a suitable algebra [13], but we prefer to use a representation inspired by the relational structures used by Courcelle [14]. Let $\mathcal{X}$ and $\mathcal{E}$ respectively, be infinite sets of nodes and edges. Let further C be a c-semiring and $\mathcal{K}$ an infinite set of names of functions $E \to A$ associating each edge with a value of the domain of the c-semiring. In addition let $\mathcal{F}$ be the universe containing all the c-semiring functions $A^i \to A$, $i \geq 0$. Set $\mathcal{F}$ also contains constant names as zero-adic functions.

A graph is a tuple $\langle X \cup E, \mathit{edge}, C, K, I \rangle$, where $X \subseteq \mathcal{X}$, $E \subseteq \mathcal{E}$, $K \subseteq \mathcal{K}$ and $\mathit{edge} : E \to X \times X$ is a function that associates an edge with its source and target vertices. Function $I : (K \cup \mathcal{F}) \to (E \to A) \cup \bigcup_{i \geq 0}(A^i \to A)$ is an interpretation, i.e. a function respectively mapping cost and c-semiring function names with actual cost and c-semiring functions. The sets X and E are of course assumed to be disjoint.

In the rest of the paper we consider finite graphs only, i.e. graphs where X and E, and we will denote the set of all graphs with set node X, c-semiring C, function set K and interpretation I with $\mathcal{G}(X, C, K, I)$ or just $\mathcal{G}$ if X, C,

K and I are clear from the context.

A decomposition of a graph G is an ordered pair of graphs that have the same nodes, c-semiring, cost functions and interpretation but have disjoint set of edges which together form the original set of edges. More precisely, a decomposition of $G = \langle X \cup E, \mathit{edge}, C, K, I\rangle$ is a pair of graphs $G_1 = \langle X \cup E_1, \mathit{edge}_1, C, K, I\rangle$, $G = \langle X \cup E, \mathit{edge}_2, C, K, I\rangle$ such that $E_1 \uplus E_2 = E$ and $\mathit{edge}_1 \uplus \mathit{edge}_2 = \mathit{edge}$.

The set of all decompositions of a graph G will be denoted by $\Theta(G)$. It is not hard to see that the size of $\Theta(G)$ is $2^{|E|}$.

3.3 Example.

Each state of the arrow distributed protocol is represented by a graph $G = \langle X \cup E, \mathit{edge}, C, K, I\rangle$, where X has an element for each node and one for the object. The set E includes edges for arrows, object sending service, object ownship and queues.

The c-semiring C might include different QoS aspects. The first one is the identity of nodes which can be modeled with the set based c-semiring $C_0 = \langle 2^X, \cup, \cap, \emptyset, X\rangle$. Then we need to model the kind of edges. Let $N = \{a, h, s, q\}$ be the set of all edge types, where a, h, s and q stand for arrow, ownship, object sending and queue. We define then a set-based c-semiring $C_1 = \langle 2^N, \cup, \cap, \emptyset, N\rangle$. We might have that propagation links have a certain delay which we represent with an optimization c-semiring C_2. Sending of the object might involve a price to pay, represented by a tropical c-semiring C_3. Hence, C is defined as $P^H(C_0 \times C_1 \times C_2 \times C_3)$, i.e. the Hoare Power Domain of the Cartesian product of C_0, C_1, C_2 and C_3.

Cost functions of K are intended to associate such QoS attributes to edges and nodes. Note that if a edge or arc does not have one of the basic attributes (type, delay or prize) the function must associate the top element to such attribute. For example, we need a function *cost* to associate arrows, and sending edges with their QoS. Arrows have type and delay, but no price or node name. Hence, $\mathit{cost} : E \to A$ associates to every arrow edge a value $\{(X, a, d, 0)\}$, for some real d. To simplify things we assume to have one cost functions for characterizing each type of arc. More precisely we consider to have functions a, q, s and o. For example $a(\mathrm{e})$ returns $\mathbf{1}$ if edge e has type a and $\mathbf{0}$ otherwise.

4 Graph logic over c-semirings

Once fixed our notion of QoS we present our logic. We first describe syntax and semantics mentioning some differences with the spatial logic for graphs of [15]. Then we show some how to express some QoS properties of our example.

4.1 Syntax.

Let V_X be a set of node variables and V_R be a set of recursion variables. Formulae of the graph logic are generated by ϕ in the following grammar:

$$\begin{array}{lll} \phi ::= & \mathbf{nil} \;\mid\; k(\xi) \;\mid\; k(\xi,\xi) \;\mid\; \phi|\phi \;\mid\; \phi\|\phi & \text{spatial operators} \\ & a \;\mid\; \phi{+}\phi \;\mid\; \phi \times \phi \;\mid\; f(\phi,\ldots,\phi) & \text{c-semiring operators} \\ & \sum \mathbf{x}.\phi \;\mid\; \prod \mathbf{x}.\phi & \text{quantification} \\ & \mathbf{r}(\overline{\xi}) \;\mid\; (\mu\mathbf{r}(\overline{\mathbf{x}}).\phi)\overline{\xi} \;\mid\; (\nu\mathbf{r}(\overline{\mathbf{x}}).\phi)\overline{\xi} & \text{fixpoints} \end{array}$$

where $a \in \mathcal{A}$, $x \in \mathcal{X}$, $\mathbf{x} \in V_X$, $\xi \in \mathcal{X} \cup V_X$, $k \in \mathcal{K}$, $f \in \mathcal{F}$, $\mathbf{r} \in V_R$. In the last three terms of ϕ, $\overline{\xi}$ and $\overline{\mathbf{x}}$ are vectors and we respectively require $|\mathbf{r}| = |\overline{\xi}|$, $|\mathbf{r}| = |\overline{\xi}| = |\overline{\mathbf{x}}|$ and $\mathbf{r}(\overline{\xi})$ to occur as operand of a monotone function or under an even number of antimonotonic functions.

Before giving a formal definition of the semantics of our logic, which is done in the next section, we give an intuition of the different syntactic ingredients. With **nil** we characterize graphs with no edges, $k(\xi)$ and $k(\xi,\xi)$ are used to express the cost of nodes and edges. With $\phi_1|\phi_2$ we range over all decompositions (G_1, G_2) of the graph multiplying the evaluation of ϕ_1 in G_1 and the evaluation of ϕ_2 in G_1. To all such values, the additive operation is applied. The spatial operator $\|$ is dual with respect to + and ×. Node quantification evaluates ϕ for each node $\mathbf{x}$ and then quantifies using + or ×. Finally, c-semiring operations have a straightforward interpretation and fixpoints have the usual meaning.

We highlight some differences with the spatial logic of [15]. Instead of boolean constants and connectives, we have c-semiring constants, operators and functions. Edge existence is enriched with a function k which assigns a c-semiring value to the edge. Cost functions can also be applied to nodes and quantifier symbols are substituted by the corresponding quantifiers $\sum$ and $\prod$, and non-derivable operators like duals are introduced explicitly since we cannot assume the existence of a negation operator which provides the usual derivations. Our logic does not include name equality, since it can be done via c-semiring functions. It suffices to consider node names as a QoS attribute an include a c-semiring function for expressing and comparing names. We neglect an explicit representation of labels, preferring to model edge labels as a QoS attributes (as we do with edge types in the example). As a consequence we do not include label quantification.

4.2 Semantics.

We interpret a formula as a mapping from the set of graphs $\mathcal{G}$ into the domain of the c-semiring A. Let $\sigma : V_X \to X$ denote the name and label environment that maps node variables with nodes [5], and let ρ be the usual propositional

---

[5] V_X, X are required to be disjoint.

environment mapping recursion variables into mappings $\mathcal{G} \to A$. With abuse of notation we let environments be applied to actual names and mappings resulting in the identity, and we abbreviate their application using postfix notation. The interpretation of formulae is as follows:

$$\begin{aligned}
[\![\mathbf{nil}]\!]_{\sigma;\rho}(G) &= E = \emptyset \\
[\![k(\xi)]\!]_{\sigma;\rho}(G) &= (\xi\sigma \in X) \times I(k)(x\sigma) \\
[\![k(\xi_1,\xi_2)]\!]_{\sigma;\rho}(G) &= (E = \{e\}) \times (\mathit{edge}(e) = (\xi_1\sigma, \xi_2\sigma)) \times I(k)(e) \\
[\![\phi_1|\phi_2]\!]_{\sigma;\rho}(G) &= \textstyle\sum_{(G_1,G_2)\in\Theta(G)}\{[\![\phi_1]\!]_{\sigma;\rho}(G_1) \times [\![\phi_2]\!]_{\sigma;\rho}(G_2)\} \\
[\![\phi_1\|\phi_2]\!]_{\sigma;\rho}(G) &= \textstyle\prod_{(G_1,G_2)\in\Theta(G)}\{[\![\phi_1]\!]_{\sigma;\rho}(G_1) + [\![\phi_2]\!]_{\sigma;\rho}(G_2)\} \\
[\![\phi_1+\phi_2]\!]_{\sigma;\rho}(G) &= [\![\phi_1]\!]_{\sigma;\rho}(G) + [\![\phi_2]\!]_{\sigma;\rho}(G) \\
[\![\phi_1 \times \phi_2]\!]_{\sigma;\rho}(G) &= [\![\phi_1]\!]_{\sigma;\rho}(G) \times [\![\phi_2]\!]_{\sigma;\rho}(G) \\
[\![f(\phi_1,\ldots,\phi_n)]\!]_{\sigma;\rho}(G) &= I(f)([\![\phi_1]\!]_{\sigma;\rho}(G), \ldots, [\![\phi_n]\!]_{\sigma;\rho}(G)) \\
[\![\kappa\mathbf{x}.\phi]\!]_{\sigma;\rho}(G) &= \kappa_{\mathbf{x}\in X}[\![\phi]\!]_{\sigma;\rho}(G) \\
[\![\mathbf{r}(\overline{\xi})]\!]_{\sigma;\rho}(G) &= \mathbf{r}\rho(\overline{\xi}\sigma) \\
[\![(\mu\mathbf{r}(\overline{\mathbf{x}}).\phi)\overline{\xi}]\!]_{\sigma;\rho}(G) &= \mathit{lfp}(\lambda\mathbf{s}.\lambda\overline{\mathbf{y}}.[\![\phi]\!]_{\sigma[\overline{\mathbf{y}}/\overline{\mathbf{x}}],\rho[\mathbf{s}/\mathbf{r}]})(\overline{\xi}\sigma)(G) \\
[\![(\nu\mathbf{r}(\overline{\mathbf{x}}).\phi)\overline{\xi}]\!]_{\sigma;\rho}(G) &= \mathit{gfp}(\lambda\mathbf{s}.\lambda\overline{\mathbf{y}}.[\![\phi]\!]_{\sigma[\overline{\mathbf{y}}/\overline{\mathbf{x}}],\rho[\mathbf{s}/\mathbf{r}]})(\overline{\xi}\sigma)(G),
\end{aligned}$$

where $\kappa \in \{\sum, \prod\}$. All terms in the right hand side are interpreted over C. For example $\xi\sigma \in X$ returns $\mathbf{1}$ if $\xi\sigma$ is an element of X and $\mathbf{0}$ otherwise.

One can show that the sets R_i of pointwise-ordered total functions of type $X^i \to (\mathcal{G} \to A)$ are complete lattices or, more generally[6], c-semirings. It can be also shown that with the mentioned syntactic restrictions functions $\lambda\overline{\mathbf{y}}.[\![\phi]\!]_{\sigma[\overline{\mathbf{y}}/\overline{\mathbf{x}}],\rho[\mathbf{s}/\mathbf{r}]}$ are monotone on $\leq_S$. Thus, the fixpoints are well defined [28]. If, in addition, functions $\lambda\mathbf{s}.\lambda\overline{\mathbf{y}}.[\![\phi]\!]_{\sigma[\overline{\mathbf{y}}/\overline{\mathbf{x}}],\rho[\mathbf{s}/\mathbf{r}]}$ are continuous with respect to $\sqcap, \sqcup$, fixpoint iteration can be applied.

Note that some of the formulae just evaluate to $\mathbf{0}$ or $\mathbf{1}$. In a way, they allow us to include boolean reasoning. For instance, **nil** is just $\mathbf{1}$ if the graph has no edges and $\mathbf{0}$ otherwise. On the other hand, formulae like $k(\xi)$ or $k(\xi_1, \xi_2)$ return the cost associated to the corresponding nodes and edges if they exist in the graph. More precisely $k(\xi_1, \xi_2)$ is $\mathbf{0}$ if the set of edges is not an edge between $\xi_1\sigma$ and $\xi_2\sigma$. Otherwise it evaluates to the cost associated to the only edge of the graph by function k.

Composition $\phi_1 \mid \phi_2$ is interpreted as follows. One looks at every possible decomposition of the graph into two graphs G_1 and G_2. The evaluation of ϕ_1 in G_1 and ϕ_2 in G_2 are combined using ×. The set of values we get for all the decomposition is then combined using the additive operation.

---

[6] Recall c-semirings form complete lattices [4].

As a trivial example, assume we want to find optimal QoS among the edges of a graph that go from a node X to a node y. Suppose that the QoS of an edge is given by a function k. The formula we need is $k(x,y)|\mathbf{1}$. Note that $[\![k(x,y)]\!](G_1) \times [\![\mathbf{1}]\!](G_2)$ is $\mathbf{0}$ if G_1 is empty or has more than two edges. Thus it is easy to see that $k(x,y)|\mathbf{1}$ is the addition of the QoS of all edges from x to y.

To illustrate the use of the other form of composition suppose we want to measure the QoS of a graph as the combination of the QoS of all edges of the graph. For this purpose we write the formula $(\mathbf{void}+(\mathbf{void}|\mathbf{void})+k(x,y))||\mathbf{0}$. Note that the left part of the decomposition evaluates to $\mathbf{1}$ if applied to a graph that is either empty ($\mathbf{void}$) or has more than one edge ($(\mathbf{void}|\mathbf{void})$). If the graph is composed by one edge the term evaluates to the QoS of that edge. Thus, $(\mathbf{void}+(\mathbf{void}|\mathbf{void})+k(x,y))||\mathbf{0}$ is the product of the QoS of all the edges of a graph.

The intuition behind the rest of the formulae should be clear: c-semiring operations and functions have a straightforward interpretation and quantification applies the additive or the multiplicative operation to the evaluation of a formula for every substitution of $\mathbf{x}$ by a node of the graph. Least and greatest point fixpoint use multiple recursion variables and have the usual meanining.

In addition to the fact that we interpret formulae as values of a c-semiring, the main semantic difference with the spatial logic of [8] is that quantification here is done over the set X of nodes of the graph and not over the infinite set of node names $\mathcal{X}$. This does not compromise decidability in GL, since one can indeed look at a finite set of nodes [15]. This would not apply to our logic since we allow formulae like $\prod \mathbf{x}.k(x)$ which might require to examine all the nodes.

Another difference is that in the original logic, decomposition is quantified over all decompositions, and not just those preserving the set of nodes and labels, as we do. It is easy to see that in the original logic this makes no difference, since one cannot distinguish between unconnected nodes and nodes that are not in the graph. To the contrary, our logic allows this since we find it interesting to make such distinctions.

Example.

Let us now briefly illustrate how to represent Property 1 of our motivating example, namely that the maximal unique path induced by non-terminal arrows leads either to the owner of the object or to a node that has requested it. The property is purely boolean and one can use the spatial logic of [8] to express it. Though the use of recursion is not necessary we will make use of it in order to use as many ingredients of the logic as possible. We show first how to state that there exists a path of arrows between two nodes:

$$\mathbf{path}(\mathbf{x},\mathbf{y}) \equiv \mu \mathbf{r}(\mathbf{x},\mathbf{y}).(\mathbf{x}=\mathbf{y}) + \sum \mathbf{z}.a(\mathbf{x},\mathbf{z})|\mathbf{r}(\mathbf{z},\mathbf{y}).$$

The intuition behind this formula is the following: there is a path from $\mathbf{x}$ to $\mathbf{y}$ in a graph if $\mathbf{x}$ is exactly $\mathbf{y}$ ($\mathbf{x} = \mathbf{y}$) or the graph can be decomposed in two subgraphs such that in one subgraph we have an arc from $\mathbf{x}$ to a node $\mathbf{z}$ and in the other subgraph there is a path from $\mathbf{z}$ to $\mathbf{y}$ ($\sum \mathbf{z}.a(\mathbf{x},\mathbf{z})|\mathbf{r}(\mathbf{z},\mathbf{y})$).

In our example we are interested in paths that end in a *terminal* node, i.e. a node with a self-arrow. But before we define an abbreviation for a formula stating that a node has no outgoing arrow edges: $\mathbf{out}_0(\mathbf{y}) \equiv (\sum \mathbf{x}.a(\mathbf{y},\mathbf{x})|\mathbf{1}) = \mathbf{0}$. Observe that $\sum \mathbf{x}.a(\mathbf{y},\mathbf{x})|\mathbf{1}$ is $\mathbf{1}$ in graphs with outgoing arrows from node $\mathbf{y}$. Thus we require the value of that expression to be $\mathbf{0}$.

Now we can characterize terminal nodes by $\mathbf{t}(\mathbf{x}) \equiv a(\mathbf{x},\mathbf{x})|\mathbf{out}_0(x)$, i.e. nodes that have only one outgoing self-arrow.

The kind of paths we want to want to express must be *unique*, i.e. the number of outgoing transitions from every node of the path must be one (except for the last node). Hence, we have:

$$\mathbf{ump}(\mathbf{x},\mathbf{y}) \equiv \mu\mathbf{r}(\mathbf{x},\mathbf{y}).(\mathbf{x} = \mathbf{y}) \times \mathbf{t}(\mathbf{y}) + \sum \mathbf{z}.a(\mathbf{x},\mathbf{z})|(\mathbf{out}_0(x) \times \mathbf{r}(\mathbf{z},\mathbf{y})),$$

where **ump** stands for unique maximal path. Finally, we can state that for each node such a path exists and ends in a node owning the object or waiting for it as :

$$\mathbf{property1} \equiv \prod \mathbf{x}. \sum \mathbf{y}.(\mathbf{ump}(\mathbf{x},\mathbf{y}) \times (\mathbf{owner}(\mathbf{y}) + \mathbf{waiter}(\mathbf{y}))|\mathbf{1})$$

In words, for every node $\mathbf{x}$ there is a node $\mathbf{y}$ such that we have unique terminal arrow path from $\mathbf{x}$ to $\mathbf{y}$, which is the node owning the object ($\mathbf{owner}(\mathbf{y})$) or a terminal node that is not the owner ($\mathbf{waiter}(\mathbf{y})$).

Abbreviation $\mathbf{owner}(\mathbf{y})$ is $\mathbf{t}(\mathbf{y},\mathbf{y}) \times o(\mathbf{y},o)$ and $\mathbf{waiter}(\mathbf{y})$ is $\mathbf{t}(\mathbf{y},\mathbf{y}) \times (\mathbf{owner}(\mathbf{y}) = \mathbf{0})$.

We now extend Property 1 to include QoS aspects. Instead of the existence of a path, we can now reason about the QoS attributes of a path:

$$\mathbf{costp}(\mathbf{x},\mathbf{y}) \equiv \mu\mathbf{r}(\mathbf{x},\mathbf{y}).(\mathbf{x} = \mathbf{y}) + \sum \mathbf{z}.(a(\mathbf{x},\mathbf{z}) \times \mathit{cost}(\mathbf{x},\mathbf{z}))|\mathbf{r}(\mathbf{z},\mathbf{y})$$

Note that the only thing that changes with respect to **path** is the introduction of the cost function $\mathit{cost}(\mathbf{x},\mathbf{z})$. The rest is the same. Indeed the formula combines both boolean aspects to characterize paths and QoS aspects. The formula should be intuitively interpreted as follows: The optimal path from $\mathbf{x}$ to $\mathbf{y}$ is $\mathbf{0}$ if $\mathbf{x}$ and $\mathbf{y}$ are the same node or the optimal between cost of and edge from $\mathbf{x}$ to a node $\mathbf{z}$ combined with the cost of the optimal path from $\mathbf{z}$ to $\mathbf{y}$. Note that the expression of optimal paths by decomposition is possible since we work with a c-semiring, i.e. since the multiplicative operation distributes over the additive operation.

Now, we can combine this formula with the one defined to characterize the correct paths. Thus obtaining a formula to represent the QoS of unique maximal arrow paths:

$$\mathbf{costump}(\mathbf{x},\mathbf{y}) \equiv \mathbf{ump}(\mathbf{x},\mathbf{y}) \times \mathbf{costp}(\mathbf{x},\mathbf{y})$$

Recall that Property 1 requires a path to lead either to the object owner or to a waiter. Hence, we can take into account the QoS attributes associated to acquiring the object. We can now write the following formula:

$$\mathbf{costump}(x, y) \times (\mathit{cost}(y, x) \times \mathit{sending}(y, x)|\mathbf{1})$$

4.3 Computing the semantics.

We informally discuss some issues around the problem of evaluating our formulae. First of all when considering complexity issues we abstract from the complexity of c-semiring operations as usual [20] since this depends on the actual c-semiring instance. Take for example, the power set of a c-semiring which domain is infinite. The result is a c-semiring where elements of the domain are possibly infinite sets. Storing and manipulating such elements might be unfeasible.

We consider first the fixpoint-free fragment of the logic. Evaluating **nil**, $k(\xi)$ and $k(\xi, \xi)$ can be done in constant time. One of the most complex computations regards composition. Indeed, additive and multiplicative composition require to enumerate all decompositions of a graph, from which there are exponentially many in the number of edges of the graph. Node and label quantification requires to enumerate finite sets X and A respectively.

Thus, it is easy to see that this fragment of the logic is decidable. Our logic subsumes a fragment of the spatial logic for graphs [8] for which the model checking problem has been shown to be PSPACE-Complete [15]. Hence, our logic is in PSPACE, and we conjecture that model checking the fixpoint-free fragment of our logic is PSPACE-complete if we abstract from the complexity of the c-semiring.

The main drawback of our approach regards the full logic, since fixpoint iteration might be infeasible. Take, for example this alternative formula for computing the QoS of the optimal arrow path between two nodes:

$$\mathbf{costp'}(\mathbf{x}, \mathbf{y}) \equiv (\mu\mathbf{r}(\mathbf{x}, \mathbf{y})\mathbf{x} = \mathbf{y}) + \sum \mathbf{z}.(a(\mathbf{x}, \mathbf{z}) \times \mathit{cost}(\mathbf{x}, \mathbf{z})|\mathbf{1}) \times \mathbf{costp'}(\mathbf{z}, \mathbf{y})$$

It is easy to see that, if we are given a graph with cycles, two nodes x, y such that y is not reachable from x and an optimization c-semiring then evaluation $\mathbf{costp'}(x, y)$ might require infinitely many iterations to return $+\infty$, the bottom of the c-semiring. Observe, however that this is not a problem in the original formula **costp** because there the recursion variable appears as operand of composition. Hence, fixpoint iteration necessarily ends after a finite number of steps; the initial graph is finite and each iteration *removes* an arc.

Restricting the use of recursion, such that recursion appears inside composition where the other side evaluates to $\mathbf{0}$ in the empty graph only will guarantee fixpoint iteration to terminate. It is an open question whether this syntactic restriction implies a significant loss of expressive power. We still did not find a property that can be expressed by a fixpoint formula where recursion does not appear as operand of composition, that cannot be expressed by a formula that satisfies the restriction. Intuitively, it might not be interesting to consider an edge more than once in a recursive formula.

5 Further Applications.

Constrained-based routing [29] is an interesting field of application of our logic. The problem is in general to find optimal routes satisfying some constraints, where both the optimization and the constraint criteria concern QoS properties. For example, one is interested in optimizing bandwidth and price while requiring the total delay to be less or equal than a certain value. For a survey on such problems we refer to [29].

Various of these problems prioritize some QoS attributes over others. For example, the shortest-widest path aims at finding the shortest path among those which have optimal bandwidth. In such cases the implicit QoS structure might not form a c-semiring. This is indeed the case of the shortest-widest path problem. A solution to this problem is to ignore the priorities in a first step and optimize all the QoS attributes using the Hoare Power Domain. The evaluation of a formula will be a set of non-dominated tuples, from which one can select a subset according to the priorities.

The expression of constrained-based unicast route optimization has a naive solution. Let **path**(**x**,**y**) be a formula that characterizes graphs which are paths from node **x** to node **y**, let **cost** be a formula that expresses the multiplication of the costs of all arcs in a graph, and let c be the global QoS constraint on the path, i.e. the maximal QoS level allowed for the path. Note that inequalities can be considered as part of the c-semiring functions in $\mathcal{F}$. The desired formula is then:

$$(\mathbf{path}(\mathbf{x},\mathbf{y}) \times \mathbf{cost} \times \mathbf{cost} \geq c)|\mathbf{1}$$

This formula suggests a general pattern to express graph constrained optimization problems:

- Define a formula p that characterizes the valid graphs.
- Define a formula q that assigns the QoS level to a graph (not necessarily the product of all edge costs).
- Define a formula r that constraints the QoS of a graph (not necessarily a global constraint on the QoS level of a graph).
- The desired formula is then $(p \times q \times r)|\mathbf{1}$.

While this pattern allows to express such problems in a fairly intuitive way, the actual computation of the resulting formulae might be very inefficient, mainly due to redundancy in the formula and the nesting of compositions. Observe that the intuition behind the pattern is that one enumerates all subgraphs of the graph and selects the optimal one satisfying the desired properties and constraints. In a way this is a sort of brute-force specification. In some cases redundancy can be avoided. Formula **costp**, for example, combines both path characterization and cost expression and profits from the distributive property of c-semirings to use recursion in an efficient way.

6 Conclusions.

We have extended a spatial logic for graphs with QoS attributes which formal structure is a c-semiring. The evaluation of a formula of our logic is not just a boolean value but a value of the domain of the c-semiring, representing the QoS level associated to a graph. We have shown how our logic can be used to express common QoS properties of graphs, like optimal constrained routes or minimal spanning trees, concentrating in an example of a distributed protocol, using a fairly intuitive design pattern. Unfortunately, the resulting formulae may lead to inefficient computations. In current work we are trying to establish formulae for common QoS problems that avoids such problems.

We are currently studying the complexity and expressivity of our logic. The main problem concerns the computation of fixpoints. Fixpoint iteration is not guaranteed to terminate in a finite number of steps in general. Syntactic restrictions can ensure termination but it is an open question to establish if the loss of expressivity due to the restriction is significant.

We also plan to precisely analyze potential applications of our logic in the domain of SON systems and to extend our concepts to a logic including QoS, temporal and spatial aspects. This is an interesting avenue that would allow us to reason about QoS in graph transition systems (roughly speaking, transition system which states are graphs). Approaches to express and verify boolean properties of graph transitions systems already exist, e.g., [25,2] and can serve as inspiration for our purposes.

References

[1] Baier, C., B. Havekort, H. Hermanns and J.-P. Katoen, *Model checking continuous-time Markov chains by transient analyisis*, in: *Computer Aided Verification*, LNCS **1855**, 2000, pp. 358–372.

[2] Baldan, P., A. Corradini, B. König and B. König, *Verifying a behavioural logic for graph transformation systems*, in: *CoMeta'03*, ENTCS, 2004.

[3] Bella, G. and S. Bistarelli, *Soft constraint programming to analysing security protocols*, Theory and Practice of Logic Programming To appear.

[4] Bistarelli, S., U. Montanari and F. Rossi, *Semiring-based constraint satisfaction and optimization*, Journal of the ACM **44** (1997), pp. 201–236.

[5] Bistarelli, S., U. Montanari and F. Rossi, *Soft constraint logic programming and generalized shortest path problems*, Journal of Heuristics **8** (2002), pp. 25–41.

[6] Caires, L. and L. Cardelli, *A spatial logic for concurrency (part II)*, in: *Proceedings of the 13th International Conference on Concurrency Theory* (2002), pp. 209–225.

[7] Caires, L. and L. Cardelli, *A spatial logic for concurrency (part I)*, Inf. Comput. **186** (2003), pp. 194–235.

[8] Cardelli, L., P. Gardner and G. Ghelli, *A spatial logic for querying graphs*, in: P. W. et al, editor, *ICALP'2002*, Lecture Notes in Computer Science **2380**.

[9] Cardelli, L., P. Gardner and G. Ghelli, *Manipulating trees with hidden labels*, in: *Foundations of Software Science and Computation Structures (FOSSACS)*, Lecture Notes in Computer Science (2003), pp. 216–232.

[10] Chechik, M., S. Easterbrook and A. Gurfinkel, *Multi-valued symbolic model-checking*, ACM Transactions on Software Engineering and Methodology (2003), to appear.

[11] Clarke, E. and J. Wing, *Formal methods: State of the art and future directions.*, ACM Computing Surveys **28** (1996), pp. 626–643.

[12] Clarke, E. M., O. Grumberg and D. A. Peled, "Model Checking," MIT Press, 1999.

[13] Corradini, A., U. Montanari and F. Rossi, *An abstract machine for concurrent modular systems: Charm*, Theoretical Computer Science (1994), pp. 165–200.

[14] Courcelle, B., **1 : Foundations**, World Scientific, 1997 pp. 313–400.

[15] Dawar, A., P. Gardner and G. Ghelli, *Expressiveness and complexity of graph logic*, Technical Report 2004/3, Imperial College, Department of Computing (2004).

[16] de Alfaro, L., *Quantitative verification and control via the mu-calculus*, in: R. M. Amadio and D. Lugiez, editors, *Proceedings of 14th International Conference on Concurrency Theory*, Lecture Notes in Computer Science **2761** (2003), pp. 102–126.

[17] Demmer, M. J. and M. Herlihy, *The arrow distributed directory protocol*, in: *International Symposium on Distributed Computing (DISC 98)*, 1998, pp. 119 –133.

[18] Duan, Z., Z. Zhang and Y. T. Hou, *Service overlay networks: SLAs, QoS, and bandwidth provisioning*, IEEE/ACM Transactions on Networking (TON) **11** (2003), pp. 870–883.

[19] Ehrig, H., G. Engels, H.-J. Kreowski and G. Rozenberg, editors, "Handbook of Graph Grammars and Computing by Graph Transformation," World Scientific, 1999.

[20] Fink, E., *A survey of sequential and systolic algorithms for the algebraic path problem*, Technical Report CS-92-37, Department of Computer Science, University of Waterloo (1992).

[21] Gaubert, S., *Methods and applications of (max,+) linear algebra*, Technical Report 3088, Institut National de Recherche en Informatiqu et en Automatique (1997).

[22] Gu, X., K. Nahrstedt, R. Chang and C. Ward, *QoS-assured service composition in managed service overlay networks*, in: *IEEE International Conference on Distributed Computing Systems (ICDCS2003)* (2003).

[23] Lluch-Lafuente, A. and U. Montanari, *Quantitative mu-calculus and CTL based on constraint semirings*, in: A. Cerone and A. di Pierro, editors, *2nd Workshop on Quantitative Aspects of Programming Languages*, Electronic Notes in Theoretical Computer Science (2004).

[24] Milner, R., "Communication and Concurrency," Prentice Hall, 1989.

[25] Rensink, A., *Towards model checking graph grammars*, in: *3rd Workshop on Automated Verification of Critical Systems*, Tech. Report DSSE-TR-2003, 2003, pp. 150–160.

[26] Reynolds, J., *Separation logic: A logic for shared mutable data structures*, in: *LICS 2002*, 2002, pp. 55–74.

[27] Rote, A., *A systolic array algorithm for the algebraic path problem (shortest path; matrix inversion)*, Computing (1985), pp. 191–219.

[28] Tarski, A., *A lattice-theoretical fixpoint theorem and its applications*, Pacific Journal of Mathematics (1955).

[29] Younis, O. and S. Fahmy, *Constraint-based routing in the internet: Basic principles and recent research*, IEEE Communications Surveys and Tutorials **5** (2003).