Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)

Querying Data Graphs with Arithmetical Regular Expressions

Maciej Graboń and **Jakub Michaliszyn** and **Jan Otop** and **Piotr Wieczorek**
Institute of Computer Science, University of Wrocław

Abstract

We propose a query language LARE for graphs whose edges are labelled by a finite alphabet and nodes store unbounded data values. LARE is based on a variant of regular expressions with memory. Queries of LARE can compare quantities of memorised graph nodes and their neighbourhoods. These features allow us to express a number of natural properties, while keeping the data complexity (with a query fixed) in non-deterministic logarithmic space. We establish an algorithm that works efficiently not only with LARE, but also with a wider language defined using effective relational conditions, another formalism we propose.

1 Introduction

Recently, there has been a growing interest in graphs as means of representing data (see surveys [Barceló, 2013; Angles and Gutierrez, 2008]), where besides querying the stored data, one can reason about the links among the data.

In these applications, the graph databases tend to be too big to fit in the modern computers' memory. Therefore, a typical criterion of feasibility of a query language is having data complexity in NL, the class of problems solvable in non-deterministic logarithmic space [Calvanese *et al.*, 2006; Artale *et al.*, 2007; Barceló *et al.*, 2012]. Checking whether there is a path between two given nodes is already NL-complete, so NL is the best complexity we can obtain for any reasonable language.

Our contribution. We propose a query language, called LARE, which subsumes several previous formalisms, allows to express new interesting properties and at the same time keeps data complexity of query answering in NL.

LARE allows for writing queries against both nodes and paths given as input. A query can existentially quantify nodes and paths and check relationship between many paths using relational conditions defined by arithmetic regular expressions (ARE), which are regular expressions with registers that allow for various arithmetic comparisons between registers as well as for nesting. The main innovation of the LARE queries lays in the ability to express various arithmetic and aggregative properties of nodes and paths, hence we assume natural numbers as the data values. Nevertheless, our approach can be adjusted to different data domains.

LARE is powerful enough to find, for example, *nodes s, t such that there is a path p from s to t such that p visits a node with the maximal value in the graph and the total sum of all the elements of p is in a given interval*. LARE allows for using nested queries and their negation. This facilitates formulation of properties such as *there is a one-way path from s to t, i.e. a path in which for any two consecutive nodes v, v', there is no path from v' to v*. Properties like these occur naturally in reasoning about multi-agent systems. Agents are often represented by a Kripke structure, which is basically a labelled graph with distinguished initial states. The considered Kripke structures are typically very large due to the state explosion problem. The query answering algorithm we propose works in (non-deterministic) logarithmic space in size of the generated Kripke structure, but can be easily adjusted to reason in polynomial space in size of a succinct representation of a multiagent system, i.e., without generating the exponentially large Kripke structure. Further examples are in Section 5.

We associate with each graph a separate finite automaton, which recognizes the paths of this graph satisfying the query. Based on this idea, we introduce *effective relational conditions*, a succinct formalism for representing a family of such automata, containing one automata for each graph. The query answering problem for effective relational conditions is NL-complete. We provide a translation from LARE queries into effective relational conditions, which proves that the data complexity of the query answering problem for LARE queries is in NL. The combined complexity is PSPACE-complete.

Query languages for graphs and LARE. Many of the query languages for graphs are extensions of *Regular Path Queries (RPQ)* [Cruz *et al.*, 1987]. RPQs can be written in the form $x \to^{\pi} y \wedge \pi \in L(e)$ where e is a (standard) regular expression. Such queries return pairs of nodes (v, v') connected by a path π such that the labelling of π form a word in $L(e)$. *Conjunctive Regular Path Queries (CRPQs)* (see Q_{CRPQ} on Fig. 1) are closure of RPQs under conjunction and existential quantification [Consens and Mendelzon, 1990; Mendelzon and Wood, 1995], *CRPQs with inverse* [Calvanese *et al.*, 2000] allow traversing graph edges back.

Barcelo et al., [2012] considered *extended CRPQs (ECRPQs)* that have the ability to output and compare not only tuples of nodes, but also tuples of paths (see r_{ECRPQ} on Fig. 1).

Tuples of paths can be compared by *regular relations* [Elgot and Mezei, 1965; Frougny and Sakarovitch, 1993]. Examples of such relations are path equality, length comparisons, prefix (i.e., a path is a prefix of another path), fixed edit distance etc. Regular relations on n-tuples of paths can be defined by the standard regular expressions over alphabet of n-tuples of edge symbols. Such regular expressions are the basic building block of LARE, in particular LARE queries without register assignment and constraints can be seen as unions of ECRPQs. Query answering for ECRPQs is computationally feasible: its combined complexity is PSPACE-complete and data complexity is NL.

The formalisms mentioned above assume that graph edges are labelled by a finite alphabet. In practice, graph nodes often store *data values* from infinite alphabet and there is a strong need for query formalisms that can combine graph topology and data values tests. In such graphs, paths are interleaved sequences of data values and edge labels. This is closely related to *data words* [Neven *et al.*, 2004; Demri *et al.*, 2007; Segoufin, 2006; Bojańczyk *et al.*, 2011]. Data complexity of query answering for most of the formalisms for data words is NP-hard [Libkin *et al.*, 2016].

Regular Queries with Memory (RQMs) [Libkin and Vrgoč, 2012; Libkin *et al.*, 2016] are again of the form $x \to^{\pi} y \wedge \pi \in L(e)$, however e is now *Regular Expression with Memory (REM)*. Such queries return pairs of nodes connected by a path in $L(e)$. REMs resemble standard regular expressions but they can store in a register the data value at the current position and test its equality with other values stored already in registers (see r_{RQM} on Fig. 1). Data complexity of RQMs is NL. Arithmetical regular expressions of LARE, called ARE, have been inspired by REM—essentially ARE over single path and with constraints having only (dis)equality tests on data values are equivalent to REMs. However, in contrast to REM, ARE work over tuples of paths, which can be compared by regular relations (as in ECRPQs), registers of ARE store nodes of the graph rather than data values and ARE incorporate arithmetical functions and arbitrary comparisons in register constraints. REMs together with their weaker versions (REWB and REWE) have been further studied in [Libkin *et al.*, 2015].

Walk Logic [Hellings *et al.*, 2013] is a powerful extension of FO with path quantification, and tests of equality of data values on paths. Query answering for WL is decidable but its data complexity is not elementary [Barceló *et al.*, 2015].

Finally, Register Logic [Barceló *et al.*, 2015], is essentially, the language of REMs closed under Boolean combinations, node, path and register assignment quantification. Interestingly, it allows for comparing data values in different paths. Its query answering is costly but for the positive fragment RL^{+} data complexity drops to NL. A particularly interesting fragment is *nested* RL^{+} where *nested REMs (NREMs)* can be used instead of REMs. Nested REMs extend REMs with a branching operator that can filter those nodes in a graph that are the starting point of a path that can be parsed by a nested query which can be NREM again. Data complexity for NRL^{+} stays in NL. Constraints of ARE allow for nested queries which capture this kind of branching.

Our language LARE can express all the ECRPQ and NRL^{+}

"x and y have a common descendant":
$Q_{\mathrm{CRPQ}}(x,y) = x \to^{\pi} z \wedge y \to^{\pi'} z \wedge \pi, \pi' \in L(\mathrm{desc}^{*})$
"x and y are connected to some node by paths having the same labelling from the alphabet Σ":
$$Q_{\mathrm{ECRPQ}}(x,y) = x \to^{\pi} z \wedge y \to^{\pi'} z \wedge \begin{pmatrix}\Sigma\\ \Sigma\end{pmatrix}^{*}(\pi,\pi')$$
"x and y are connected by a path in which a data value is repeated":
$Q_{\mathrm{RQM}} = x \to^{\pi} y \wedge \pi \in L(\Sigma^{*} \cdot \downarrow r \cdot \Sigma^{+} \cdot [r^{=}] \cdot \Sigma^{*})$.

Figure 1: Basic features of formalisms that LARE builds over. $\downarrow r$ stores the current data value in a register r and $[r^{=}]$ tests whether the data value stored in r equals the current one.

properties and more, as LARE can test various arithmetical conditions rather than just the equality and refer to nodes' neighbourhood. We provide some concrete examples in the paper, e.g. in Example 1 we show a query α_{fad} that looks for paths that always choose the next node with the most occurring value in a set of candidates. This query is not expressible in NRL^{+} or ECRPQ.

Figueira and Libkin [2015] proposed a language to express properties of edge labels of paths, such as *the number of a-edges following b-edges is the same as the number of b-edges following a-edges*. Data complexity of its query answering problem is in NL, and the proof relies on a tailored version of Parikh automata. We note that it is possible to combine the result with ours, i.e., to have a query language able to express both kind of properties with NL complexity.

2 Graphs and queries

Graphs. We fix a finite set of edge labels Σ. A $\Sigma, \mathbb{N}$-*labelled graph*, or simply a *graph*, is a tuple $G = \langle V, E, \lambda \rangle$ where V is a finite set of nodes, $E \subseteq V \times \Sigma \times V$ is a set of edges labelled by Σ, and $\lambda : V \to \mathbb{N}$ is a labelling of nodes.

The *size* of a graph G is defined as $|G| = |V| + |E| + \sum_{v \in V} \lambda(v)$, i.e., the labels of nodes are represented in the unary notation, which means that their binary representation is logarithmic in the size of the graph. This allows us to compute arithmetical relations on these labels in logarithmic space. The question, which arithmetical relations can be computed in logarithmic space w.r.t. input number given in binary, is related to long-standing open problems, e.g., whether linear programming admits strongly-polynomial algorithm.

A *word* is an element of the language defined by $V(\Sigma V)^{*}$.

A node v' is an e-*successor* of v if $E(v, e, v')$. A *path* is a word $p = v_0 e_1 v_1 \dots v_k$ such that v_i is an e_i-successor of v_{i-1} for every $i \in \{1, \dots, k\}$.

An n-ary *relational condition* is a graph indexed family of relations $\mathrm{R} = \{\mathrm{R}^{G} : G \text{ is a graph}\}$ such that each R^{G} is an n-ary relation on the paths from G. This may be seen as an n-ary relation on paths, which depends on a graph.

Syntax of queries. *Queries* Q are defined by the BNF expression $Q ::= Q \vee Q \mid Q \wedge Q \mid x_i \to^{x_k} x_j \mid R(x_{i_1}, \dots, x_{i_n})$, where $n, i, j, k, i_1, \dots, i_n \in \mathbb{N} \setminus \{0\}$, R ranges over n-ary relational conditions and $x_1, x_2, \dots$ are variables.

Variables are intended to range over paths; nodes are considered as special cases of paths. Free variables of a query can be distinguished by listing them after the query name, e.g.,

$Q(x_1,\dots,x_k)$ denotes a query whose variables $x_1,\dots,x_k$ are free; the remaining variables are existentially quantified.

Semantics of queries. The satisfaction relation $\vDash^G$, which takes a vector of paths $\vec{p} = (p_1, p_2, \dots)$. and a query Q, is recursively defined as follows. We assume that for each i, the path p_i is the value of the query variable x_i.

- $\vec{p} \vDash^G Q_1 \vee Q_2$ if $\vec{p} \vDash^G Q_1$ or $\vec{p} \vDash^G Q_2$,
- $\vec{p} \vDash^G Q_1 \wedge Q_2$ if $\vec{p} \vDash^G Q_1$ and $\vec{p} \vDash^G Q_2$,
- $\vec{p} \vDash^G x_i \to^{x_k} x_j$ if p_i, p_j are single-node paths and p_k is a path from p_i to p_j,
- $\vec{p} \vDash^G R(x_{i_1},\dots,x_{i_n})$ if $R^G(p_{i_1},\dots,p_{i_n})$.

A query $Q(x_1,\dots,x_l)$ *holds* for paths $p_1,\dots,p_l$ of a graph G, denoted as $Q^G(p_1,\dots,p_l)$, if there are paths $p_{l+1}, p_{l+2}, \dots$ such that $(p_1, p_2, \dots) \vDash^G Q$. For example, $Q(x_1) = x_1 \to^{x_2} x_3 \wedge x_4 \to^{x_5} x_1 \wedge R(x_2, x_5)$ states that there are two paths, one starting and one ending in x_1 satisfying R.

The *query-answering problem* is formalised as follows: given a graph G, a query $Q(x_1,\dots,x_l)$ and (input) paths $p_1,\dots,p_l$, does $Q^G(p_1,\dots,p_l)$ hold? We are interested in the *data complexity* of the problem, where the size of a query and its input paths is treated as constant, and *combined complexity*, where there is no such restriction.

3 Arithmetical regular expressions

Different relational conditions lead to different expressive power and complexity of queries. As we are interested in queries regarding large systems, representing, for example, Web topology, social networks or Kripke structures, we restrict our attention to relational conditions defining queries whose data complexity is in NL. In this section, we propose n-ary arithmetical regular expressions (ARE), which define relational conditions satisfying this complexity requirement. ARE are regular expressions with arithmetical functions and memory. The memory is formalized as an infinite set of registers $\mathbb{R} = \{r_i \mid i \in \mathbb{N}\}$, storing nodes.

It is convenient to reason about paths of the same length. To cope with paths of different lengths, we use $\square$ as a special blank (padding) symbol for nodes and edges and assume $E(v, \square, \square)$ for any $v \in V_\square$ and $\lambda(\square) = \square$. One may think that G has an additional dummy node $\square$, which is a $\square$-successor of all nodes. By $X_\square$ we denote the set X extended with $\square$.

We assume a finite set $\mathcal{F}$ of functions $f : (\mathbb{N}_\square \cup \{\infty\})^* \to \mathbb{N} \cup \{\infty\}$ computable by a non-deterministic Turing machine whose size of working tape and output tape while computing $f(x_1,\dots,x_k)$ is $O(\log k + \max(|x_1|,\dots,|x_k|))$, assuming binary representation and $|\infty| = |\square| = 1$. These conditions are satisfied by aggregative functions such as summation, maximum, minimum (∞ is included for empty set), counting and conditional functions like *if* x_1 *is odd then* x_2 *else* x_3.

Syntax of ARE. We define *register constraints* C and n-ary *arithmetical regular expressions* α as follows

$$\begin{aligned} C ::= & C \vee C \mid \neg C \mid \exists r.C \mid f(P,\dots,P) \leq f(P,\dots,P) \\ & \mid r = r \mid E(r, e, r) \mid [\![Q]\!](r,\dots,r) \\ \alpha ::= & \epsilon \mid \langle\!\langle C \rangle\!\rangle \mid [r \leftarrow j] \mid \vec{e} \mid \alpha + \alpha \mid \alpha.\alpha \mid \alpha^+ \end{aligned}$$

where r ranges over $\mathbb{R}$, f ranges over $\mathcal{F}$, P ranges over expressions of the form $\lambda(r)$ or $f[r : C]$, $e \in \Sigma$, $\vec{e} \in \Sigma_\square^n$, $j \in \{1,\dots,n,\square\}$ and Q is a (nested) query.

By ARE$_n$ we denote the class of all n-ary arithmetical regular expressions. We allow standard logical abbreviations, such as $\wedge, \forall, \Rightarrow$, defined as usual. We put $E(r, \_, r') = \bigvee_{a \in \Sigma} E(r, a, r')$ to express *there is a (Σ-labelled) edge between* r *and* r'. Notice that our constraints are standalone entities rather than evaluations applied to subexpressions.

To simplify presentation we require that the register constraints and arithmetical functions only express properties of nodes stored in registers. Hence, all the nodes of the paths have to be stored in registers using the $[r \leftarrow i]$ syntax prior to their access. However, it is easy to circumvent this by assuming n distinguished registers storing the values of the current nodes.

Register constraints allow for Boolean operators (including negation), comparing arithmetical formulas, quantification, checking equality and connectedness of nodes stored in registers and checking nested queries. Note that we allow for negation in front of nested queries, and that the parameters of nested queries are only (nodes stored in) registers. The arithmetical formulas are the way of expressing properties of values of the nodes stored in registers. The construction $f[r : C]$, that can be read as *the function* f *applied to the values of all* r *satisfying a given condition*. For example, $\max[r_1 : E(r_2, e, r_1)]$ stands for the maximum value of e-successors of the node stored in r_2.

A *valuation* $\sigma : \mathbb{R} \to V_\square$ is a function that assigns nodes to the registers. For a valuation σ, we define $\sigma[r \leftarrow v_i]$ as the valuation such that $\sigma[r \leftarrow v_i](r) = v_i$ and $\sigma[r \leftarrow v_i]$ coincides with σ on all inputs except r.

Let $f\langle X \rangle$ denote the result of function $f \in \mathcal{F}$ whose arguments are elements of X given in a non-decreasing order.

Semantics of register constraints. A graph $G = (V, E, \lambda)$ and a valuation σ *satisfy* a constraint C, denoted by $G, \sigma \vDash C$, if one of the following holds:

- $C \equiv C_1 \vee C_2$ and $G, \sigma \vDash C_1$ or $G, \sigma \vDash C_2$,
- $C \equiv \neg C'$ and $G, \sigma \nvDash C'$,
- $C \equiv \exists r.C'$ and there is $v \in V$ s.t. $G, \sigma[r \leftarrow v] \vDash C'$,
- $C \equiv f(P_1^1 \dots P_k^1) \leq g(P_1^2 \dots P_l^2)$ and $f(v_1^1 \dots v_k^1) \leq g(v_1^2 \dots v_l^2)$, where $v_i^j = \lambda(\sigma(r))$ if $P_i^j \equiv \lambda(r)$ and $v_i^j = f\langle\{\lambda(v) \mid G, \sigma[r \leftarrow v] \vDash C\}\rangle$ if $P_i^j \equiv f[r : C]$.
- $C \equiv r = r'$ and $\sigma(r) = \sigma(r')$,
- $C \equiv E(r, e, r')$ and $E(\sigma(r), e, \sigma(r'))$,
- $C \equiv [\![Q]\!](r_{i_1},\dots,r_{i_k})$ and $Q^G(\sigma(r_{i_1}),\dots,\sigma(r_{i_k}))$.

The *convolution* of sequences $s_1,\dots,s_n$, denoted by $s_1 \otimes \dots \otimes s_n$, is the sequence s of the length k of the longest sequence among $s_1,\dots,s_n$, such that for every $i \in \{1,\dots,k\}$, the ith element of s is the vector $(a_1,\dots,a_n)$, where a_j is the ith element of s_j if it exists and $\square$ otherwise. In plain English, convolution joins n sequences into one sequence of the length of the longest sequence and fills the missing places with $\square$.

An *n-path of* G is the convolution of n paths of G. We define a concatenation . of two n-paths $p_1 = v_0 e_1 \dots v_k$ and $p_2 =$

$v'_0 e'_1 \ldots v'_k$ such that $v_k = v'_0$ as $p_1.p_2 = v_0 e_1 \ldots v_k e'_1 \ldots v'_k$ (i.e., the common node is not repeated). A *splitting* of an n-path p is a sequence of n-paths $p_1, \ldots, p_l$ such that $p = p_1.p_2.\ldots.p_l$.

Language of ARE. We define the relation $\vdash$ with the following meaning: $(\alpha, p, \sigma) \vdash^G \sigma'$ if evaluating an expression α over a n-path p of a graph G with a valuation σ results in valuation σ', i.e., if one of the following holds.

- $\alpha \equiv \epsilon$, p is a single element and $\sigma = \sigma'$,
- $\alpha \equiv \langle\!\langle C \rangle\!\rangle$, p is a single element, $G, \sigma \vDash C$ and $\sigma = \sigma'$,
- $\alpha \equiv [r \leftarrow j]$, p is a single element $(a_1, \ldots, a_n)$ and $\sigma' = \sigma[r \leftarrow a_j]$ if $j \neq \square$ and $\square$ otherwise,
- $\alpha \equiv \vec{e}$, $p = v_1 \vec{e} v_2$ where v_1, v_2 are single elements and $\sigma = \sigma'$,
- $\alpha \equiv \alpha_1.\alpha_2$ and there is a splitting $p = p_1.p_2$ and a valuation σ'' s.t. $(\alpha_1, p_1, \sigma) \vdash^G \sigma''$ and $(\alpha_2, p_2, \sigma'') \vdash^G \sigma'$,
- $\alpha \equiv \alpha_1 + \alpha_2$ and $(\alpha_1, p, \sigma) \vdash^G \sigma'$ or $(\alpha_2, p, \sigma) \vdash^G \sigma'$,
- $\alpha \equiv \alpha_1^+$ and there are a splitting $p = p_1.\ldots.p_k$ and valuations $\sigma = \sigma_0, \sigma_1, \ldots, \sigma_k = \sigma'$ such that for each $i \in \{1, \ldots, k\}$ we have $(\alpha_1, p_i, \sigma_{i-1}) \vdash^G \sigma_i$,

Then, the *language* of an ARE α is defined as $L^G(\alpha) = \{p \mid \exists \sigma, \sigma'.(\alpha, p, \sigma) \vdash^G \sigma'\}$. We define $\alpha^G(p_1, \ldots, p_n)$ iff $\vec{p}_1 \otimes \cdots \otimes \vec{p}_n \in L^G(\alpha)$; therefore, each ARE can be treated as a relational condition.

Example 1. Consider the following ARE.
$\alpha_{grd} = ([r_1 \leftarrow 1]\Sigma[r_2 \leftarrow 1]\langle\!\langle C_{grd} \rangle\!\rangle)^*$, where
$C_{grd} = \forall r_3.(E(r_1, \_, r_3) \Rightarrow \lambda(r_3) \leq \lambda(r_2))$.
$\alpha_{fad} = ([r_1 \leftarrow 1]\Sigma[r_2 \leftarrow 1]\langle\!\langle \forall r_3.\ P(r_3) \leq P(r_2) \rangle\!\rangle)^*$, where
$P(z) = \mathrm{count}[r_4 : E(r_1, \_, r_4) \wedge \lambda(r_4) = \lambda(z)]$.
$\alpha_{com} = [r_1 \leftarrow 1] \ldots [r_n \leftarrow n]\alpha_{gss}\langle\!\langle r_1 = r_2 \ldots = r_n \rangle\!\rangle$, where
$\alpha_{gss} = ([r_1 \leftarrow 1] + \ldots + [r_n \leftarrow n] + \Sigma^n_\square)^*$

The language of α_{grd} consists of all greedy paths, i.e., paths in which, at every position, the following node on the path has maximal value among all successors of the current node. To see that, observe that while traversing path α_{grd} keeps the next-to-latest (on the path) node in register r_1 and the latest node in r_2. After each edge move, denoted by Σ, a *greedy* condition C_{grd} is checked, whether it holds that r_2 is a maximal neighbour of r_1. The language of α^G_{fad} consists of paths in which, at every position, the following node on the path has the value that has the most number of occurrences among the successors of the current node. To express that, α_{fad} uses arithmetic formula $P(z)$ counting number of occurrences of value $\lambda(z)$ in the neighbourhood of r_1. count is a function (from $\mathcal{F}$) that returns the number of elements given on input. The last example demonstrates processing multiple paths. We have $\alpha^G_{com}(x_1, \ldots, x_n)$ iff paths $x_1, \ldots, x_n$ have a common node. The idea is to guess (α_{gss}) a common node for each path separately, and verify the equality at the end.

The *query language* LARE is the language containing queries whose all relational conditions are ARE. We conclude this section with a Theorem whose proof will follow from more general theorems provided later on.

Theorem 1. *The query answering problem for* LARE *queries with bounded nesting depth is in* PSPACE *and its data complexity is* NL.

4 Effective relational conditions

We introduce the notion of *effective relational conditions*, which facilitates the proof of Theorem 1, and show that ARE are effective relational conditions (Theorem 4). The name effective is justified by the fact that answering queries with effective relational conditions can be done within the desired complexity bounds (Theorem 3). We assume that the reader is familiar with finite automata and Turing machines (see [Hopcroft and Ullman, 1979]).

An Automata Giving Turing Machine (AGTM) is a nondeterministic Turing Machine which works in logarithmic space and only accepts inputs of the form $i?w$, where $i \in \{0,1\}^*$, $? \in \{?_\Gamma, ?_S, ?_I, ?_F, ?_\delta\}$, and $w \in \{0, 1, ;\}^*$ is such that $|w| = O(\log(|i|))$. An AGTM $\mathcal{M}$ *gives* (*on-the-fly*) a set of (nondeterministic) automata $\{\mathcal{A}_i\}_{i \in \{0,1\}^*}$ such that each $\mathcal{A}_i$ is of the form $(\Gamma_i, S_i, I_i, \delta_i, F_i)$, where

- $\Gamma_i \subset \{0,1\}^*$ consists of *labels* e s.t. $\mathcal{M}$ accepts on $i?_\Gamma e$,
- $S_i \subset \{0,1\}^*$ consists of *states* q s.t. $\mathcal{M}$ accepts on $i?_S q$.
- I_i consists of *initial states* $q \in S_i$ s.t. $\mathcal{M}$ accepts on $i?_I q$.
- F_i consists of *final states* $q \in S_i$ s.t. $\mathcal{M}$ accepts on $i?_F q$.
- δ_i consists of *transitions* $(q, e, q') \in S_i \times \Gamma_i \times S_i$ s.t. $\mathcal{M}$ accepts on $i?_\delta q; e; q'$.

An AGTM M *represents* a n-ary relational condition R if M gives a set of automata $\{\mathcal{A}_i\}_{i \in \{0,1\}^*}$ s.t. for all graphs G and words $w_1, \ldots, w_n$, the automaton $\mathcal{A}_G$ accepts $w_1 \otimes \ldots \otimes w_n$ iff $(w_1, \ldots, w_n) \in R^G$ (we assume an encoding of graphs as binary sequences). A relational condition R is *effective* if there is an AGTM M that represents R. A query is effective if its all relational conditions are given as AGTMs.

Lemma 2. *Let* $\mathrm{R}_\to$ *be a relational condition such that* $\mathrm{R}^G_\to(x, x_1, x')$ *holds iff* $x \to^{x_1} x'$. *Then* $\mathrm{R}_\to$ *is effective.*

Proof. We define an AGTM $\mathcal{M}$ that gives automata $\{\mathcal{A}_i\}_{i \in \{0,1\}^*}$ such that for a given graph G, the automaton $\mathcal{A}_G$ recognizes words over the alphabet $V^3_\square \cup \Sigma^3_\square$, which encode convolutions $w_1 \otimes w_2 \otimes w_3$ such that w_1, w_3 are single-node paths and w_2 is a path in G from the node w_1 to w_3. $\mathcal{A}_G$ reads letters corresponding to nodes v and labels e and stores last read letters in its state. Next, while it reads a letter corresponding to a node v', it checks whether G has an edge from v to v' labeled with e. Moreover, the automaton $\mathcal{A}_G$ stores in its state the node v_f, which is supposed to be the last node and accepts only if the last read letter is v_f.

The machine $\mathcal{M}$ answers questions $i?_X x$ regarding components (the alphabet, the set of states, etc.) of $\mathcal{A}_G$, where G is the graph encoded by i. All components are sets of tuples of nodes and labels of G extended with the padding symbol $\square$. All these sets of tuples are defined using the edge relation in G, equality among components of the tuples and equality to $\square$. In any reasonable representation of G; e.g., a list of nodes followed by an adjacency list, $|G|$ is proportional to i, length of description of labels and nodes of G is logarithmic in i, and the machine $\mathcal{M}$ can compute in $\log i$ space (hence in $\log |G|$ space) whether a given word encodes a label (resp., a node or an edge) of G. Since the length of elements of tuples, i.e., labels and nodes of G is logarithmic in i, the equality among components of tuples (and equality to $\square$) can be checked in

$\log\log i$ space. Therefore, the machine $\mathcal{M}$ on inputs $i?_X x$ requires $\log(i)$ space, i.e., $\mathcal{M}$ is an AGTM. $\square$

Theorem 3. *Answering effective queries with m conditions is in* $\mathrm{NSPACE}(m + c\log(n))$*, where $c\log(n)$ is the space bound on AGTMs representing the conditions.*

Proof sketch. Consider a query $Q(x_1, \ldots, x_k)$ whose relational conditions are effective. For any paths $p_1, \ldots, p_k$, we define in the straightforward way an effective relational condition $R_{p_1,\ldots,p_k}$ such that $R^G_{p_1,\ldots,p_k}(\pi_1, \ldots, \pi_k)$ iff $p_1 = \pi_1$, $\ldots$, $p_k = \pi_k$. This and Lemma 2 imply that we can assume that our query has no free variables and is built of conjunction, disjunction and effective relational conditions. One can observe that effective relational conditions are closed under shuffling, joining and adding spurious arguments, and therefore we can assume that all of them have the same arguments. By employing an argument similar to the standard powerset construction for finite automata, we can reduce in polynomial time a query to a single effective relational condition $R_{Q,p_1,\ldots,p_k}$.

Given a graph G, answering $Q^G(p_1, \ldots, p_k)$ amounts to checking emptiness of $R^G_{Q,p_1,\ldots,p_k}$, which amounts to checking whether a corresponding automaton A_G is empty. This can be done by employing the standard graph reachability algorithm, which checks reachability of an accepting state from initial states by guessing the consecutive states. Since states are logarithmic in i, the problem is in NL.

The AGTM for the relational condition R_Q works in space proportional to the number of atomic formulas of the query (m) plus the maximum of the space usage of AGTMs representing relational conditions in Q ($c\log(n)$). Therefore, the combined complexity is as required. $\square$

Let 0-ARE be the empty set and $(d+1)$-ARE be the subset of ARE containing expressions whose all nested queries are built of relational conditions from d-ARE.

Theorem 4. *Every* ARE *relational condition R is effective and if R is d-*ARE*, then an AGTM working in space $|\mathrm{R}|\cdot\log(n)$ representing R is computable in polynomial time.*

Proof sketch. We prove Theorem 4 by induction w.r.t. d. The basis of induction is trivial as the set 0-ARE is empty. Assume that for any $\mathrm{R} \in d$-ARE one can compute in polynomial time in $|\mathrm{R}|$ an AGTM computing R that works in space $|\mathrm{R}|\log(n)$.
Evaluating register constraints. We first show that for a given register constraint C whose nested queries are built of d-ARE, one can compute in polynomial time in $|C|$ a non-deterministic Turing machine $\mathcal{M}_C$ that works in space $|C|\log|G|$ and decides, for a graph G and a registers valuation σ, whether $G, \sigma \models C$. The construction of $\mathcal{M}_C$ is a top-down recursion on C and can be done in polynomial time. The space used by $\mathcal{M}_C$ is bounded by $O(|C|\log|G|)$, which, using tape compression [Hopcroft and Ullman, 1979], can be replaced by $|C|\log|G|$. Below we discuss the most interesting recursive cases.

To compute $f[r : C]$, observe that using additional space $\log|G|$, we can implement a virtual tape storing $\lambda(v_1) \le \ldots \le \lambda(v_l)$, where $v_1, \ldots, v_l$ are all nodes of G satisfying C, i.e., we can implement a subroutine that for a given i returns $\lambda(v_i)$. Such a subroutine iterates over all nodes v of G. For each v, it computes i^v_1 (resp., i^v_2) defined as the number of nodes u, which satisfy C and $\lambda(u) < \lambda(v)$ (resp., $\lambda(u) \le \lambda(v)$), and checks whether $i^v_1 < i \le i^v_2$. In such a case we know that all elements between $i^v_1 + 1$ and i^v_2 have the same value $\lambda(v)$. Next, with such a virtual input tape we execute a non-deterministic Turing machine $\mathcal{M}_f$ computing $f \in \mathcal{F}$. All such Turing machines work in space $O(\log|G|)$.

Consider a nested query $[\![Q]\!](r_{i_1}, \ldots, r_{i_k})$. All the relational conditions of Q are d-ARE, therefore by Theorem 3 and the inductive assumption, we can compute (in polynomial time) a non-deterministic machine $\mathcal{M}_Q$ that computes $[\![Q]\!](r_{i_1}, \ldots, r_{i_k})$ in space $|Q|\log|G|$. We also compute a non-deterministic machine $\mathcal{M}_{\neg Q}$ that in space $c_{IS}\cdot|Q|\log|G|$ computes $\neg[\![Q]\!](r_{i_1}, \ldots, r_{i_k})$, where c_{IS} is the constant from Immerman-Szelepcsényi theorem [Immerman, 1988] showing NL = coNL. We execute $\mathcal{M}_Q$ and $\mathcal{M}_{\neg Q}$ in parallel.
Evaluating $(d+1)$-ARE. Now we prove the inductive thesis. Let R be a $(d+1)$-ARE relational condition defined by $(d+1)$-ARE α. For a graph G, we define three automata $\mathcal{A}^G_1, \mathcal{A}^G_2$ and $\mathcal{A}^G_3$ over an alphabet Γ consisting of tuples of edge labels $\Sigma^n_\square$, nodes $V^n_\square$, symbols $\{a_C : C \text{ from } R\}$ referring to register constraints and $\{b_{r,j} : [r \leftarrow j] \text{ from } R\}$ referring to register assignments. We substitute in α subexpressions $\langle\!\langle C \rangle\!\rangle$ (resp., $[r \leftarrow j]$) by letter a_C (resp., $b_{r,j}$). The resulting α^{reg} is a regular expression and $\mathcal{A}^G_1$ recognizes the language of α^{reg}, i.e., $\mathcal{A}^G_1$ accepts words over Γ that satisfy R while neglecting the graph structure G and the register constraints checks in R. For the register constraints checks, this automaton verifies only that the letters a_C (resp., $b_{r,j}$) occur at positions that correspond to register constraints checks $\langle\!\langle C \rangle\!\rangle$ (resp., assignments $[r \leftarrow j]$) in R. The automaton $\mathcal{A}^G_2$ checks consistency of register constraints checks marked by a_C provided that register assignments are executed done according to markings $b_{r,j}$, employing the machine $\mathcal{M}_C$ defined above. The automaton $\mathcal{A}^G_3$ recognizes words that, with letters a_C and $b_{r,j}$ deleted, are n-paths of G. Finally, we construct $\mathcal{A}^G_{\mathrm{R}}$, which is the product of automata $\mathcal{A}^G_1, \mathcal{A}^G_2$ and $\mathcal{A}_3$ projected on the alphabet $\Sigma^n \cup V^n$. Observe that $\mathcal{A}^G_{\mathrm{R}}$ recognizes the languages of all n-paths satisfying R. It can be shown that the construction of automata $\mathcal{A}^G_1, \mathcal{A}^G_2$ and $\mathcal{A}^G_3$ can be performed in a way that ensures that the sets containing all such automata can be given on-the-fly. $\square$

Finally, we show how Theorems 3 and 4 imply Theorem 1. Let $d > 0$ and consider LARE queries of nesting depth bounded by d, i.e., queries whose relational conditions are $(d+1)$-ARE. For every such query Q, Theorem 4 states that for every relational condition R from Q, one can construct in polynomial time an AGTM working in $|\mathrm{R}|\log(n)$ space, which represents R. Next, observe that the sum of the number and the sizes of relational conditions from Q is bounded by Q. Therefore, by Theorem 3, space required to answer Q is bounded by $|Q|\log(n)$. Consequently, the query answering problem is in PSPACE and its data complexity is in NL.
Remark. We have shown that for a given query Q and a graph G we can obtain a single relational condition R

and then an automaton $\mathcal{A}_{\mathrm{R}}^{G}$ accepting all n-paths w over $V^n(\Sigma^n V^n)^*$ such that w is a convolution of paths that satisfy Q. Note that we can modify $\mathcal{A}_{\mathrm{R}}^{G}$ to be able to process paths that remember also the values of the nodes, i.e., the paths over $(V \times \mathbb{N})^n(\Sigma^n(V \times \mathbb{N})^n)^*$. Therefore, we can state our result in the following alternative version:

Corollary 5. For each $\Sigma, \mathbb{N}$-labelled graph G and a LARE query Q with k variables, there is a $\Sigma^k, \mathbb{N}^k$-labelled graph G' of size polynomial in $|G|$ and sets of nodes I, F such that a word w is a path from I to F in G' iff w is a convolution of paths that satisfy Q.

5 Examples

We show how ARE power the LARE queries, and combine nicely with language features like nested queries, condition negation, or aggregative functions. For convenience, we define a macro $[\overset{n}{\leftarrow}] = [r_1 \leftarrow 1] \dots [r_n \leftarrow n]$ that stores nodes from first n paths into registers $1, \dots, n$. We allow to use different variables than x_i and assume variables x, y, x_i, y_i to be nodes, this can be enforced by appending the atom $x \to^{\pi} x$ to query body; we treat it as implicitly written.

Basic techniques. Consider the queries

$Q_{\neq}(x, \pi_1, \pi_2) = \alpha_{\in}(x, \pi_1) \wedge \alpha_{\notin}(x, \pi_2)$, where
$\alpha_{\in} = [r_1 \leftarrow 1](\Sigma_{\square}^2)^*[r_2 \leftarrow 2]\langle\!\langle r_1 = r_2 \rangle\!\rangle(\Sigma_{\square}^2)^*$
$\alpha_{\notin} = [\overset{2}{\leftarrow}](\langle\!\langle r_2 \neq r_1 \rangle\!\rangle \Sigma_{\square}^2 [r_2 \leftarrow 2])^* \langle\!\langle r_2 \neq r_1 \rangle\!\rangle$.
$Q_2(\pi) = ([r_2 \leftarrow 1]\Sigma[r_1 \leftarrow 1]\langle\!\langle C_2 \rangle\!\rangle)^*(\pi)$,
$C_2 = \mathrm{count}[z : r_2 \to^{\pi} z] > \mathrm{count}[z : r_1 \to^{\pi} z]$.
$Q_3() = \langle\!\langle \forall z \forall z'. \neg [\![x \to^{\pi_1} y \wedge y \to^{\pi_2} x]\!](z, z') \rangle\!\rangle()$.
$Q_4(x, y) = x \to^{\pi} y \wedge \alpha(x, y)$,
$\alpha = [\overset{2}{\leftarrow}]\langle\!\langle \forall z. \neg [\![Q_{4.1}]\!](r_1, r_2, z) \rangle\!\rangle$,
$Q_{4.1}(x, y, z) = x \to^{\pi_1} y \wedge x \to^{\pi_2} y \wedge Q_{\neq}(x, \pi_1, \pi_2)$.

The query $Q_{\neq}$ checks whether a node x appears in path π_1 but not π_2. The query Q_2 checks whether a given path has unique nodes with respect to strongly connected components. The query Q_3 that check whether a given graph is a *directed acyclic graph*. The query has no parameters, so it is considered over the 0-path consisting of one node which is an empty tuple, $()$. We use negation on query saying π has return edge. No return edges means that G is acyclic.

The query $Q_4(x, y)$ asks whether there is only one path between x and y, up to nodes rearrangement. $Q_{4.1}(x, y, z)$ says that one can find two paths between x and y that differ on whether they contain node z. Expression α recognizes pairs of nodes x_1, y_1 for whom it is impossible to show two paths, that would differ on some node. So, in other words, all paths between x and y take usage of same set of nodes. $Q_4(x, y)$ additionally checks that there is at least one path from x to y.

Aggregative properties. We show that LARE is capable of recognizing some basic aggregative properties of paths. We demonstrate this on the ARE $sum_{\leq c}$, stating that the sum of weights on a path do not exceed the constant c.

$sum_{\leq c} = \alpha_{\text{init}}.\alpha_{\text{store}}.(\Sigma.\alpha_{\text{store}})^*.\langle\!\langle \sum(r_2, \dots, r_{c+1}) \leq c \rangle\!\rangle$,
$\alpha_{\text{init}} = [r_2 \leftarrow \square] \dots [r_{c+2} \leftarrow \square]$,
$\alpha_{\text{store}} = [r_1 \leftarrow 1](\langle\!\langle \lambda(r_1) = 0 \rangle\!\rangle + \alpha_1 + \dots + \alpha_c)$,
$\alpha_k = \langle\!\langle \lambda(r_1) > 0 \wedge r_{k+1} = r_{c+2} \rangle\!\rangle [r_{k+1} \leftarrow 1]$.

The register r_1 is used to store the current node, registers $r_2, \dots, r_{c+1}$ contain the previous nodes whose data value was positive, and r_{c+2} contains $\square$ (for comparing). At every node, the regular expression evaluates α_{store}. If the value of the node is 0, then nothing happens. If the value is positive, then the node is stored in one of registers among $r_2, \dots, r_{c+1}$ containing $\square$. If there is no such register, it means that there are already c nodes with positive value on the paths, meaning that sum of weights on the path exceeds c, in which case the path will not be matched. At the end, it is checked whether the sum of nodes in the registers do not exceed c.

The concept of checking lower and upper bound on the sum of path weight can be extended in natural way to n-paths.

Kripke structures and multiagent systems. LARE may be seen as a handful tool to verify properties of Kripke structures, which are essentially graphs with distinguished initial states. Finite Kripke structures are often obtained from programs with unbounded numbers by means of abstraction [Clarke *et al.*, 2000], and therefore the obtained labels may be treated as data values. We are especially interested in the multiagents setting, where edge labels correspond to agents' actions. LARE is expressive enough to express properties such as there is a path from an initial state q_0 to a final state q_f such that no action of one of the agents at any single state will result in reaching an undesirable state q_B. For epistemic logics [Ågotnes *et al.*, 2015], node labels may be seen as a representation of a local state of an agent. The technique presented in this paper can be straightforwardly generalised to graphs with more than one node-labelling function in order to deal with more agents.

Our query language then is powerful enough to formulate indistinguishability in the epistemic interval-temporal logic EHS [Lomuscio and Michaliszyn, 2013], where two paths are indistinguishable for an agent i if they are of the same length and the corresponding states of both paths are indistinguishable/have the same label for i. In particular, Q_{K1} (below) expresses that there is a path indistinguishable but different from π. Q_{K2} expresses that there is a path from x to y indistinguishable from π_1 that visits z while π_1 does not. In other words, an agent, who can observe only their local state (node labelling), does not know whether a given path avoids z.

$Q_{K1}(\pi) = \alpha_i(\pi, \pi_1) \wedge \alpha_{\neq}(\pi, \pi_1)$
$\alpha_i = ([\overset{2}{\leftarrow}]\langle\!\langle \lambda(r_1) = \lambda(r_2) \rangle\!\rangle \Sigma^2)^*[\overset{2}{\leftarrow}]\langle\!\langle \lambda(r_1) = \lambda(r_2) \rangle\!\rangle$
$\alpha_{\neq} = (\Sigma^2)^*[\overset{2}{\leftarrow}]\langle\!\langle r_1 \neq r_2 \rangle\!\rangle(\Sigma^2)^*$
$Q_{K2}(x, y, z, \pi_1) = x \to^{\pi_2} y \wedge \alpha_i(\pi_1, \pi_2) \wedge Q_{\neq}(z, \pi_2, \pi_1)$

6 Conclusion and future work

We introduced the query language LARE for data graphs, suited for expressing arithmetical properties of nodes, paths between nodes and their neighbourhoods. We showed that the query answering problem for LARE can be solved in logarithmic space in size of the graph by employing a new formalism called effective relational conditions.

LARE expresses some aggregative properties of paths, as exemplified in Section 5. In our future work we plan to incorporate more properties of this type. In particular, are interested in formalisms that can compare sums of values on different paths while keeping the complexity low.

Acknowledgement We would like to thank Leonid Libkin who has introduced the subject to us. This paper has been supported by Polish National Science Center grant UMO-2014/15/D/ST6/00719.

References

[Ågotnes *et al.*, 2015] T. Ågotnes, V. Goranko, W. Jamroga, and M. Wooldridge. Knowledge and ability. In *Handbook of Logics for Knowledge and Belief*. College Publications, 2015.

[Angles and Gutierrez, 2008] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1, 2008.

[Artale *et al.*, 2007] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. DL-lite in the light of first-order logic. In *Proc. of the national conference on artificial intelligence*, volume 22, page 361. AAAI Press; MIT Press, 2007.

[Barceló *et al.*, 2012] Pablo Barceló, Leonid Libkin, Anthony W. Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems*, 37:31:1–31:46, 2012.

[Barceló *et al.*, 2015] Pablo Barceló, Gaëlle Fontaine, and Anthony Widjaja Lin. Expressive path queries on graph with data. *Logical Methods in Computer Science*, 11(4), 2015.

[Barceló, 2013] Pablo Barceló. Querying graph databases. In *Proc. of the 32nd Symposium on Principles of Database Systems (PODS13)*, pages 175–188, 2013.

[Bojańczyk *et al.*, 2011] Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27, 2011.

[Calvanese *et al.*, 2000] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. of the 7th Int. Conf. Principles of Knowledge Representation and Reasoning (KR00)*, pages 176–185, 2000.

[Calvanese *et al.*, 2006] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. In *Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR06)*, volume 6, pages 260–270, 2006.

[Clarke *et al.*, 2000] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. of the 12th Int. Conf. on Computer Aided Verification (CAV00)*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.

[Consens and Mendelzon, 1990] Mariano P. Consens and Alberto O. Mendelzon. GraphLog: a visual formalism for real life recursion. In *Proc. of the 9th Sympo. n Principles of Database Systems (PODS90)*, pages 404–416, 1990.

[Cruz *et al.*, 1987] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. In *Proc. of the ACM Special Interest Group on Management of Data (SIGMOD87)*, pages 323–330, 1987.

[Demri *et al.*, 2007] Stéphane Demri, Ranko Lazic, and David Nowak. On the freeze quantifier in constraint LTL: decidability and complexity. *Inf. Comput.*, 205(1):2–24, 2007.

[Elgot and Mezei, 1965] C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Dev.*, 9(1):47–68, January 1965.

[Figueira and Libkin, 2015] Diego Figueira and Leonid Libkin. Path logics for querying graphs: Combining expressiveness and efficiency. In *Proc. of the 30th Annual Symposium on Logic in Computer Science (LICS15)*, pages 329–340, 2015.

[Frougny and Sakarovitch, 1993] Christiane Frougny and Jacques Sakarovitch. Synchronized rational relations of finite and infinite words. *Theor. Comput. Sci.*, 108(1):45–82, 1993.

[Hellings *et al.*, 2013] Jelle Hellings, Bart Kuijpers, Jan Van den Bussche, and Xiaowang Zhang. Walk logic as a framework for path query languages on graph databases. In *Proc. of the 16th Int. Conf. on Database Theory (ICDT13)*, pages 117–128, 2013.

[Hopcroft and Ullman, 1979] J. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Adison-Wesley Publishing Company, 1979.

[Immerman, 1988] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[Libkin and Vrgoč, 2012] Leonid Libkin and Domagoj Vrgoč. Regular path queries on graphs with data. In *Proc. of the 15th Int. Conf. on Database Theory (ICDT12)*, pages 74–85. ACM, 2012.

[Libkin *et al.*, 2015] Leonid Libkin, Tony Tan, and Domagoj Vrgoč. Regular expressions for data words. *Journal of Computer and System Sciences*, 81(7):1278 – 1297, 2015.

[Libkin *et al.*, 2016] Leonid Libkin, Wim Martens, and Domagoj Vrgoč. Querying graphs with data. *J. ACM*, 63(2):14:1–14:53, March 2016.

[Lomuscio and Michaliszyn, 2013] A. Lomuscio and J. Michaliszyn. An epistemic Halpern-Shoham logic. In *Proc. of the 23rd International Joint Conference on Artificial Intelligence (IJCAI13)*, pages 1010–1016. AAAI Press, 2013.

[Mendelzon and Wood, 1995] Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computation*, 24(6):1235–1258, 1995.

[Neven *et al.*, 2004] Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

[Segoufin, 2006] Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Proc. of the 20th Int. Workshop on Computer Science Logic (CSL06)*, pages 41–57, 2006.