A Spatial Logic for Querying Graphs

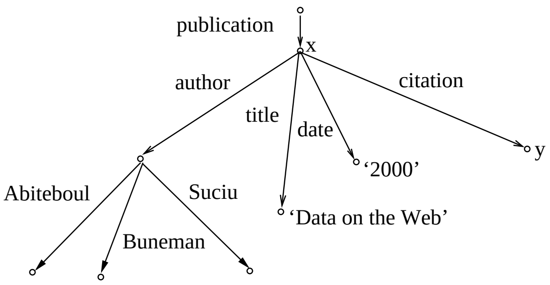
Luca Cardelli, Philippa Gardner and Giorgio Ghelli[1]

Abstract. We study a spatial logic for reasoning about labelled directed graphs, and the application of this logic to provide a query language for analysing and manipulating such graphs. We give a graph description using constructs from process algebra. We introduce a spatial logic in order to reason locally about disjoint subgraphs. We extend our logic to provide a query language which preserves the multiset semantics of our graph model. Our approach contrasts with the more traditional set-based semantics found in query languages such as TQL, Strudel and GraphLog.

1 Introduction

Semi-structured data plays an important role in the exchange of information between globally distributed applications: examples include BibTex files and XML documents. Whilst the research community mostly agree on defining semi-structured data using labelled directed graphs or trees with 'graphical' links, the study of how to query, modify and manipulate such data is still very active.

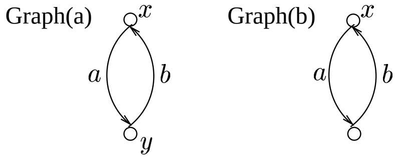
Motivating Examples A standard example used by the semi-structured data community [ABS00] is a bibtex file with an article entry of the form:



The global name (object identifier) x denotes the citation name of the publication, which is used to refer to the particular bibtex entry. The citation entry might be a simple text entry, or might point to another entry in the bibtex file. Another example with a more graphical emphasis is the correspondence between counties and towns, where counties contain towns and towns are in counties. A more complicated example is given by links between web pages, where names correspond to URLs. Such links display all manner of graphical linking. These simple examples illustrate that the typical data models for semi-structured data are either labelled directed graphs, or labelled trees with 'graphical links'. In this paper, we focus on labelled directed graphs.

[1] Cardelli's address: Microsoft Research, Cambridge. Gardner's address: Imperial College of Science, Technology and Medicine, London. Supported by an EPSRC Advanced Fellowship. Ghelli's address: University of Pisa, Pisa.

Graph Model We use a well-known graph description based on constructs from process algebra [CMR94]. The models consist of labelled edges and two kinds of nodes: the *global* nodes identified with unique names x, y, z and the *local* nodes whose identifiers are not known. In our bibtex example, the citation x corresponds to a global node labelled x, whereas the author field has no explicit citation. Similarly, the Internet's Domain Name Service globally registers IP addresses, but not all IP addresses are global. Our notation for describing graph (a) is $a(x, y) \mid b(y, x)$, where $a(x, y)$ denotes an edge and $\_|\_$ is the usual composition operator for processes used in this case to describe multisets of edges.



Graph (b) is given by $(\mathsf{local}\ y)(a(x, y) \mid b(y, x))$. The local operator is analogous to restriction in the π-calculus. It means that the previously identified node cannot now have any more edges attached to it.

Spatial Logic for Graphs Spatial logics were introduced by Caires, Cardelli and Gordon for reasoning about trees and processes [CG00,Cai99], and also by O'Hearn and Reynolds for reasoning about pointers [IO01,Rey00] using the bunched logic of O'Hearn and Pym [OP99]. Such logics provide local reasoning about disjoint substructures. We introduce a spatial logic for analysing graphs. It combines standard first-order logic with additional structural connectives. The structural formula $\phi \mid \psi$ specifies that a graph can be split into two parts: one part satisfying ϕ, the other ψ. Composition allows us to count edges. For example,

$$\exists \mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{u}.\ \ a(\mathbf{x}, \mathbf{y}) \mid b(\mathbf{y}, \mathbf{z}) \mid a(\mathbf{z}, \mathbf{u}) \mid \mathsf{true} \qquad (\dagger)$$

specifies that there are at least three different edges in the graph, with a following b following a. In contrast, conjunction allows us to describe paths with

$$\exists \mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{u}.\ \ (a(\mathbf{x}, \mathbf{y}) \mid \mathsf{true}) \wedge (b(\mathbf{y}, \mathbf{z}) \mid \mathsf{true}) \wedge (a(\mathbf{z}, \mathbf{u}) \mid \mathsf{true})$$

describing the existence of a *path* a followed by b followed by a. The path formula is satisfied by graph (a), but the composition formula (†) is not.

Our graph logic (without recursion) sits naturally between first-order logic FOL and monadic second-order logic MSOL: in FOL we can only quantify over single edges; in our logic, the formula $\phi \mid \mathsf{true}$ existentially quantifies a property ϕ over all subgraphs; in MSOL we can arbitrarily nest quantifications over sets of edges. Our logic can be viewed as a sublogic of MSOL. However, we can reason locally about disjoint subgraphs. FOL and MSOL require complex disjointness conditions to reason about such subgraphs: for example, the composition formula (†) requires such conditions to specify that the three edges are disjoint. Dawar, Gardner and Ghelli are studying expressivity results for the graph logic. Our current results are reported in [CGG01].

Query Language We define a query language based on pattern matching and recursion. Our approach integrates well with our graph description, and contrasts with the standard set-based approach found in Cardelli and Ghelli's TQL, a query language based on the ambient logic [CG01a], and the graphical query languages StruQL [FFK$^+$97] and GraphLog [CM90] based on first-order logic.

To illustrate the standard approach, consider a simple query $\mathsf{input\_graph} \models^{?} \mathbf{a}(\mathbf{x},\mathbf{y}) \,|\, \mathsf{true}$. This query asks for a substitution σ such that the satisfaction relation $\mathsf{input\_graph} \models^{\sigma} \mathbf{a}(\mathbf{x},\mathbf{y}) \,|\, \mathsf{true}$ holds in our logic. For example, if the input graph is $a(x,y) \,|\, b(y,x)$, then there are two solutions:

$$(\mathbf{a} \mapsto a,\, \mathbf{x} \mapsto x,\, \mathbf{y} \mapsto y) \qquad \text{or} \qquad (\mathbf{a} \mapsto b,\, \mathbf{x} \mapsto y,\, \mathbf{y} \mapsto x)$$

The *from/select* expressions take such solutions and build new graphs. For example, the expression

$$\mathbf{from}\ \ \mathsf{input\_graph}\ \models^{?} \mathbf{a}(\mathbf{x},\mathbf{y}) \,|\, \mathsf{true}\ \mathbf{select}\ \mathbf{a}(\mathbf{y},\mathbf{x}) \qquad (*)$$

takes *every* substitution σ which satisfies the query, and creates a new graph consisting of the composition of the edges $\mathbf{a}\sigma(\mathbf{y}\sigma,\mathbf{x}\sigma)$. In our example, the resulting new graph is $a(y,x) \,|\, b(x,y)$. Given the input graph $a(x,y) \,|\, a(x,y)$ instead, there is *one* substitution $\sigma : \mathbf{x} \mapsto x, \mathbf{y} \mapsto y$ which satisfies the query. The resulting graph is just $a(y,x)$. This collapse of information can be an advantage. It does mean however that we cannot accurately take a copy of a graph.

Instead we define a query language based on *queries* and *transducers*. Queries build new graphs from old. Transducers relate input graphs with output graphs. A basic transducer $\phi \Rrightarrow Q$ relates any input graph satisfying ϕ with the query Q which might depend on witnesses from ϕ. For example, the transducer

$$\exists \mathbf{a},\mathbf{x},\mathbf{y}.\ (\mathbf{a}(\mathbf{x},\mathbf{y}) \,|\, \mathsf{true} \Rrightarrow \mathbf{a}(\mathbf{y},\mathbf{x}))$$

relates an input graph with edge $\mathbf{a}\sigma(\mathbf{x}\sigma,\mathbf{y}\sigma)$ with the output graph $\mathbf{a}\sigma(\mathbf{y}\sigma,\mathbf{x}\sigma)$. Given the input graph $a(x,y) \,|\, b(y,x)$, there are two possible output graphs, either $a(y,x)$ or $b(x,y)$. This example does the pattern-matching part of the from/select expression $(*)$. It does not combine the inverted edges. Instead this role is played by recursion. Consider the transducer

$$\mathbf{R} \overset{\text{def}}{=} (\mathsf{nil} \Rrightarrow \mathsf{nil}) \vee (\exists \mathbf{a},\mathbf{x},\mathbf{y}.\ (\mathbf{a}(\mathbf{x},\mathbf{y}) \Rrightarrow \mathbf{a}(\mathbf{y},\mathbf{x})) \,|\, \mathbf{R})$$

Either the input graph is empty and relates to the empty output graph. Or the input graph can be split into an edge and the rest of the graph. The output graph consists of the inverted edge composed with the output associated with the remaining graph. Given input graph $a(x,y) \,|\, a(x,y)$ for example, the output graph is the exact inverted copy $a(y,x) \,|\, a(y,x)$.

We study two query languages: a basic language which can express our motivating examples, and a general language which has a simple formalism but is too expressive to implement. We were surprised to observe that the from/select expressions can be embedded in our general language.

2 Labelled Directed Graphs

We use a simple graph algebra [CMR94] to describe labelled directed graphs. Assume an infinite set $\mathcal{X}$ of names ranged over by $u, \ldots, z$, and an infinite set of edge labels $\mathcal{A}$ ranged over by a, b, c. We also use the notation $\tilde{z}$ to denote a sequence of names, and $|\tilde{z}\,|$ to denote the length of the sequence.

Definition 1
The set $\mathcal{G}(\mathcal{X}, \mathcal{A})$ of *graph terms* generated by $\mathcal{X}$ and $\mathcal{A}$ is given by the grammar

$$\begin{array}{lll} G ::= & \mathsf{nil} & \text{empty} \\ & a(x,y) & \text{edge} \\ & G \,|\, G & \text{composition} \\ & (\mathsf{local}\ x)\, G & \text{hiding} \end{array}$$

We sometimes write $\mathcal{G}$ instead of $\mathcal{G}(\mathcal{X}, \mathcal{A})$. The definitions of *free* and *bound* names are standard: the hiding operator $(\mathsf{local}\ x)\, G$ binds x in G; x is free in process $a(x,y)$. We write $\mathsf{fn}(G)$ to denote the set of free names in G. We use the capture-avoiding substitution, denoted by $G\{y/x\}$.

Our graph model is based on a multiset semantics, with the graph term $a(x,y) \,|\, a(x,y)$ denoting a graph with two edges. We give a natural structural congruence on graph terms (definition 2) which corresponds to the usual notion of graph isomorphism [CMR94]. Our choice contrasts with the approach taken in the query language StruQL, which has a set-based semantics with $a(x,y) \,|\, a(x,y)$ corresponding to $a(x,y)$. It also contrasts with the language UnQL [BDHS96], which is based on graph bisimulation rather than graph isomorphism.

Definition 2
The *structural congruence* between graph terms, written $\equiv$, is the smallest congruence closed with respect to $\_\,|\,\_$ and $(\mathsf{local}\ x)\_$, and satisfying the axioms:

$$\begin{array}{cc} G \,|\, \mathsf{nil} \equiv G & (\mathsf{local}\ x)(\mathsf{local}\ y)G \equiv (\mathsf{local}\ y)(\mathsf{local}\ x)G \\ (G_1 \,|\, G_2) \,|\, G_3 \equiv G_1 \,|\, (G_2 \,|\, G_3) & (\mathsf{local}\ x)(G_1 \,|\, G_2) \equiv (\mathsf{local}\ x)G_1 \,|\, G_2, \quad x \notin \mathsf{fn}(G_2) \\ G_1 \,|\, G_2 \equiv G_2 \,|\, G_1 & (\mathsf{local}\ x)\mathsf{nil} \equiv \mathsf{nil} \\ & (\mathsf{local}\ x)G \equiv (\mathsf{local}\ y)G\{y/x\}, \quad y \notin \mathsf{fn}(G) \end{array}$$

2.1 Comparison with Courcelle

We give a set-theoretic description of graphs in the spirit of Courcelle [Cou97], which is equivalent to our graph description. We have made some different choices to Courcelle, which we will discuss after the definition. We assume disjoint infinite sets of vertices $\mathcal{V}$, edge identifiers $\mathcal{E}$, edge labels $\mathcal{A}$, and names $\mathcal{X}$.

Definition 3
The *graph structure* $G_S = \langle V \cup E \cup A, \{\mathsf{edge} \subseteq E \times A \times V \times V\}, \mathsf{src} : X \to V \rangle$ is defined by

1. $V \subseteq \mathcal{V}$, $E \subseteq \mathcal{E}$, $A \subseteq \mathcal{A}$, $X \subseteq \mathcal{X}$ are finite sets;
2. each edge identifier has a unique label, domain node and codomain node: $\forall e, a_i, v_i, w_i . \mathsf{edge}(e, a_1, v_1, w_1) \wedge \mathsf{edge}(e, a_2, v_2, w_2) \Rightarrow a_1 = a_2 \wedge v_1 = v_2 \wedge w_1 = w_2$;
3. the edge identifiers, edge labels and vertices are related using edge:

   $\forall d\ \exists d_1, d_2, d_3 .$
   $\mathsf{edge}(d, d_1, d_2, d_3) \vee \mathsf{edge}(d_1, d, d_2, d_3) \vee \mathsf{edge}(d_1, d_2, d, d_3) \vee \mathsf{edge}(d_1, d_2, d_3, d)$
4. src is an injective function.

This definition differs from Courcelle's approach in several ways. Courcelle permits nodes to be unattached to edges. He considers both finite and infinite graphs, whereas we use the finite case since it is enough for this paper. He also does not treat A as part of the domain. Instead, he defines a family of relations $\mathsf{edge}_a \subseteq E \times V \times V$. This last point is significant when comparing our different logics for reasoning about graphs. Courcelle considers two systems, one where src is injective and one where it is not. The graphs presented here correspond to the injective case; the non-injective case corresponds to adding *name fusions* $x = y$ to our graphical description, as introduced by Gardner and Wischik [GW00].

In [Cou97], Courcelle studies a graph grammar which is similar to ours. Courcelle's motivation is to explore the expressive power of MSOL. In contrast, our motivation is to use our graphs to model semi-structured data, and to introduce a spatial logic for locally reasoning about such data.

3 The Graph Logic

We will only consider the simple case of graphs without hiding. It is possible to incorporate a quantifier for reasoning about hidden nodes [CC01,CG01b], and we believe that our query language will extend. For the rest of this paper, G ranges over the terms generated by the simple grammar: $G ::= \mathsf{nil} \ \big|\ a(x, y) \ \big|\ G \,|\, G$. The set $\mathcal{G}(\mathcal{X}, \mathcal{A})$ denotes the set of all such terms.

3.1 Logical Formulae

Formulae are constructed from a name set $\mathcal{X}$ and label set $\mathcal{A}$. They also depend on the disjoint sets of name variables $V_{\mathcal{X}}$, label variables $V_{\mathcal{A}}$ and parametrised recursion variables $\mathcal{V}_{\mathcal{R}}$. A recursion variable $\mathbf{R}$ comes with a fixed *arity* $|\mathbf{R}|$.

Definition 4 (Logical formulae)
The set of *pre-formulae* $\mathcal{F}_{\mathsf{pre}}(\mathcal{X}, \mathcal{A})$ is given by the grammars

name expressions	$\xi ::=$	x	name, $x \in \mathcal{X}$
		$\mathbf{x}$	name variable
label expressions	$\alpha ::=$	a	label, $a \in \mathcal{A}$
		$\mathbf{a}$	label variable

formulae	$\phi, \psi ::=$	nil	empty
		$\alpha(\xi_1, \xi_2)$	edge
		$\phi \mid \psi$	composition
		true	true
		$\phi \wedge \psi$	conjunction
		$\neg \phi$	classical negation
quantifiers		$\exists \mathbf{x}. \phi$	exist. quant. over names
		$\exists \mathbf{a}. \phi$	exist. quant. over labels
recursion		$\mathbf{R}(\tilde{\xi})$	$\lvert\mathbf{R}\rvert = \lvert\tilde{\xi}\rvert$
		$(\mu \mathbf{R}(\tilde{\mathbf{x}}).\ \phi)\tilde{\xi}$	least fix-pt; $\lvert\tilde{\xi}\rvert = \lvert\tilde{\mathbf{x}}\rvert = \lvert\mathbf{R}\rvert$, $\mathbf{R}(\tilde{\xi})$ occurs positively,
equality tests		$\xi_1 = \xi_2, \alpha_1 = \alpha_2$	equalities

The sets of free variables are standard. The set of *formulae* $\mathcal{F}(\mathcal{X}, \mathcal{A})$ are those pre-formulae with no free recursion variables. The order of binding precedence is $\_ = \_$, $\neg \_$, $\_ \mid \_$, $\_ \wedge \_$, with negation binding strongest. We write $x \neq y$ for $\neg(x = y)$. The scope of $\exists \mathbf{x}. \_$ and $\mu \mathbf{R}(\tilde{\mathbf{x}}). \_$ is always the maximum possible.

The nil formula specifies the empty graph. The edge formula $\alpha(\xi_1, \xi_2)$ specifies that a graph is just one edge. The composition formula $\phi \mid \psi$ specifies that a graph can be split into two parts with one part satisfying ϕ and the other ψ. The other formulae should be familiar. It is also logically natural to add other connectives such as a spatial negation and implication [OP99,CG00].

3.2 Satisfaction Relation

The satisfaction relation determines which graphs satisfy which formulae. It is defined by an interpretation function which maps pre-formulae to sets of graphs.

Definition 5 (Satisfaction)
We assume name set $\mathcal{X}$ and edge set $\mathcal{A}$. Let $\sigma : \mathcal{V}_{\mathcal{X}} \to \mathcal{X}$ denote a substitution from name and label variables to names and labels respectively, and let ρ send recursion variables of arity n to elements of the set of functions $(\mathcal{X}^n \to \mathcal{P}(\mathcal{G}))$. The *satisfaction interpretation* $\llbracket \_ \rrbracket_{\sigma;\rho} : \mathcal{F}_{\mathsf{pre}} \to \mathcal{P}(\mathcal{G})$ is defined inductively by:

$$
\begin{aligned}
\llbracket \mathsf{nil} \rrbracket_{\sigma;\rho} &= \{G : G \equiv \mathsf{nil}\} \\
\llbracket \alpha(\xi_1, \xi_2) \rrbracket_{\sigma;\rho} &= \{G : G \equiv \alpha\sigma(\xi_1\sigma, \xi_2\sigma)\} \\
\llbracket \phi \mid \psi \rrbracket_{\sigma;\rho} &= \{G : G \equiv G_1 \mid G_2 \wedge G_1 \in \llbracket \phi \rrbracket_{\sigma;\rho} \wedge G_2 \in \llbracket \psi \rrbracket_{\sigma;\rho}\} \\
\llbracket \mathsf{true} \rrbracket_{\sigma;\rho} &= \mathcal{G} \\
\llbracket \phi \wedge \psi \rrbracket_{\sigma;\rho} &= \llbracket \phi \rrbracket_{\sigma;\rho} \cap \llbracket \psi \rrbracket_{\sigma;\rho} \\
\llbracket \neg \phi \rrbracket_{\sigma;\rho} &= \mathcal{G} / \llbracket \phi \rrbracket_{\sigma;\rho} \\
\llbracket \exists \mathbf{x}.\ \phi \rrbracket_{\sigma;\rho} &= \bigcup_{x \in \mathcal{X}} \llbracket \phi \rrbracket_{\sigma, \mathbf{x} \mapsto x;\rho}
\end{aligned}
$$

$$
\begin{aligned}
[\![\exists \mathbf{a}.\ \phi]\!]_{\sigma;\rho} &= \bigcup_{a \in \mathcal{A}} [\![\phi]\!]_{\sigma,\mathbf{a}\mapsto a;\rho} \\
[\![\mathbf{R}(\tilde{\xi})]\!]_{\sigma;\rho} &= \mathbf{R}\rho(\tilde{\xi}\sigma), \quad |\tilde{\xi}| = n, \quad \mathbf{R}\rho : \mathcal{X}^n \to \mathcal{P}(\mathcal{G}) \\
[\![(\mu\mathbf{R}(\tilde{\mathbf{x}}).\phi)\tilde{\xi}]\!]_{\sigma;\rho} &= (\bigsqcap\{S \in (\mathcal{X}^{|\tilde{\mathbf{x}}|} \to \mathcal{P}(\mathcal{G})) : (\lambda\tilde{y}.\ [\![\phi]\!]_{\sigma,\tilde{\mathbf{x}}\mapsto\tilde{y};\rho,R\mapsto S}) \sqsubseteq S\})(\tilde{\xi}\sigma) \\
&\ \text{where } S \sqsubseteq S' \text{ iff } \forall \tilde{y} \in \mathcal{X}^{|\tilde{\mathbf{x}}|}.\, S(\tilde{y}) \subseteq S'(\tilde{y}) \\
[\![\xi_1 = \xi_2]\!]_{\sigma;\rho} &= \mathcal{G},\ \text{if } \xi_1\sigma = \xi_2\sigma;\ \emptyset \text{ otherwise} \\
[\![\alpha_1 = \alpha_2]\!]_{\sigma;\rho} &= \mathcal{G},\ \ \text{if } \alpha_1\sigma = \alpha_2\sigma;\ \emptyset \text{ otherwise}
\end{aligned}
$$

Definition 5 is shown to be well-defined by structural induction on formulae. For the recursive case, observe that the set of all pointwise-ordered total functions of type $\mathcal{X}^{|\tilde{\mathbf{x}}|} \to \mathcal{P}(\mathcal{G})$ is a complete lattice. Define the *satisfaction relation* $G \vDash^\sigma \phi$ for formula ϕ if and only if $G \in [\![\phi]\!]_{\sigma;\_}$, where $\_$ denotes an arbitrary ρ.

Proposition 6 (Satisfaction Properties)
The satisfaction relation satisfies the following standard properties:

$$
\begin{aligned}
G \vDash^\sigma \mathsf{nil} &\Leftrightarrow G \equiv \mathsf{nil} \\
G \vDash^\sigma \alpha(\xi_1, \xi_2) &\Leftrightarrow G \equiv \alpha\sigma(\xi_1\sigma, \xi_2\sigma) \\
G \vDash^\sigma \phi \,|\, \psi &\Leftrightarrow \exists G_1, G_2 \in \mathcal{G}.\ (G \equiv G_1 \,|\, G_2\ \wedge\ G_1 \vDash^\sigma \phi \wedge G_2 \vDash^\sigma \psi) \\
G \vDash^\sigma \mathsf{true} &\Leftrightarrow G \in \mathcal{G} \\
G \vDash^\sigma \phi \wedge \psi &\Leftrightarrow G \vDash^\sigma \phi\ \wedge\ G \vDash^\sigma \psi \\
G \vDash^\sigma \neg\phi &\Leftrightarrow \neg(G \vDash^\sigma \phi) \\
G \vDash^\sigma \exists\mathbf{x}.\phi &\Leftrightarrow \exists x \in \mathcal{X}.\ G \vDash^\sigma \phi\{x/\mathbf{x}\} \\
G \vDash^\sigma \exists\mathbf{a}.\phi &\Leftrightarrow \exists a \in \mathcal{A}.\ G \vDash^\sigma \phi\{a/\mathbf{a}\} \\
G \vDash^\sigma (\mu\mathbf{R}(\tilde{\mathbf{x}}).\ \phi)(\tilde{\xi}) &\Leftrightarrow G \vDash^\sigma \phi\{\tilde{\xi}/\tilde{\mathbf{x}}\}[(\mu\mathbf{R}(\tilde{\mathbf{x}}).\phi)/\mathbf{R}] \\
G \vDash^\sigma \xi_1 = \xi_2 &\Leftrightarrow \xi_1\sigma = \xi_2\sigma \\
G \vDash^\sigma \alpha_1 = \alpha_2 &\Leftrightarrow \alpha_1\sigma = \alpha_2\sigma
\end{aligned}
$$

The recursion case requires a substitution and monotonicity lemma showing that the function $\lambda\tilde{y}.\ [\![\phi]\!]_{\sigma,\tilde{\mathbf{x}}\mapsto\tilde{y};\rho,R\mapsto S}$ is monotone in S. Then we apply the fix-point theorem.

Definition 7 (Derived Formulae)
We give some derived formulae which are used throughout the paper:

$$
\begin{aligned}
\mathsf{false} &\stackrel{\text{def}}{=} \neg\mathsf{true} & \phi \,||\, \psi &\stackrel{\text{def}}{=} \neg(\neg\phi \,|\, \neg\psi) \\
\phi \vee \psi &\stackrel{\text{def}}{=} \neg(\neg\phi \wedge \neg\psi) & \mathsf{subgraph}_\exists(\phi) &\stackrel{\text{def}}{=} \phi \,|\, \mathsf{true} \\
\phi \Rightarrow \psi &\stackrel{\text{def}}{=} \neg\phi \vee \psi & \mathsf{subgraph}_\forall(\phi) &\stackrel{\text{def}}{=} \phi \,||\, \mathsf{false} \\
\forall\mathbf{x}.\ \phi &\stackrel{\text{def}}{=} \neg\exists\mathbf{x}.\neg\phi
\end{aligned}
$$

The connective $\_||\_$ is the de Morgan dual of $\_|\_$. The binding precedence is $\_ \wedge \_$, $\_\vee\_$, $\_ \Rightarrow \_$, with conjunction binding strongest. The scope of $\forall\mathbf{x}.\_$ is the maximum possible.

Example We revisit the two examples discussed in the introduction:

$$\begin{array}{c} \exists \mathbf{x},\mathbf{y},\mathbf{z},\mathbf{u}.\quad a(\mathbf{x},\mathbf{y}) \mid b(\mathbf{y},\mathbf{z}) \mid a(\mathbf{z},\mathbf{u}) \mid \mathsf{true} \\ \exists \mathbf{x},\mathbf{y},\mathbf{z},\mathbf{u}.\quad (a(\mathbf{x},\mathbf{y}) \mid \mathsf{true}) \wedge (b(\mathbf{y},\mathbf{z}) \mid \mathsf{true}) \wedge (a(\mathbf{z},\mathbf{u}) \mid \mathsf{true}) \end{array}$$

Recall that the first formula specifies that a graph has at least three different edges; the second that a graph has a path of three edges.

Example We specify the property that there exists a path from x to y in our logic without recursion. This is interesting since it is not expressible in first-order logic without recursion. First we give some preliminary derived formulae:

$$\begin{array}{rl} \text{no edge into } x & \mathsf{in}_0(x) \stackrel{\text{def}}{=} \neg\exists \mathbf{y},\mathbf{a}.\ \mathbf{a}(\mathbf{y},x) \mid \mathsf{true} \\ n+1 \text{ edges into } x & \mathsf{in}_{n+1}(x) \stackrel{\text{def}}{=} \exists \mathbf{y},\mathbf{a}.\ \mathbf{a}(\mathbf{y},x) \mid \mathsf{in}_n(x) \\ \text{a minimal graph satisfying } \phi & \mathsf{min}(\phi) \ \stackrel{\text{def}}{=}\ \phi \wedge \neg(\phi \mid \neg\mathsf{nil}) \\ x \text{ is a node in the graph} & \mathsf{in\_graph}(x) \ \stackrel{\text{def}}{=}\ \exists \mathbf{y},\mathbf{a}.\,(\mathbf{a}(x,\mathbf{y}) \vee \mathbf{a}(\mathbf{y},x)) \mid \mathsf{true} \end{array}$$

The formulae $\mathsf{out}_n(x)$ are defined similarly to $\mathsf{in}_n(x)$. We now give a formula which specifies that a graph is just a straight path from x to y and does not contain a cycle (when $x = y$ the formula is satisfied by the empty graph):

$$\begin{array}{l} \mathsf{straight\_path}(\mathbf{x},\mathbf{y}) \stackrel{\text{def}}{=} \mathsf{min}[\mathbf{x} = \mathbf{y} \vee (\mathsf{in}_0(\mathbf{x}) \wedge \mathsf{out}_1(\mathbf{x}) \wedge \mathsf{in}_1(\mathbf{y}) \wedge \mathsf{out}_0(\mathbf{y}) \wedge \\ \qquad\qquad\qquad \forall \mathbf{z}.\ \mathbf{z} \neq \mathbf{x} \wedge \mathbf{z} \neq \mathbf{y} \wedge \mathsf{in\_graph}(\mathbf{z}) \Rightarrow \mathsf{out}_1(\mathbf{z}) \wedge \mathsf{in}_1(\mathbf{z}))] \end{array}$$

This formula specifies that the graph contains one start node $\mathbf{x}$, one end node $\mathbf{y}$ and all the other nodes must have one incoming and one outgoing edge (hence no cycles). Minimality ensures that there are no disconnected cycles. The property that there exists a path from x to y is now specified by the formula $\mathsf{exists\_path}(\mathbf{x},\mathbf{y}) \stackrel{\text{def}}{=} \mathsf{subgraph}_\exists(\mathsf{straight\_path}(\mathbf{x},\mathbf{y}))$.

Example We give an equivalent formula to $\mathsf{exists\_path}(x,y)$ using recursion—we use the notation $\mathbf{R}(\tilde{\mathbf{x}}) \stackrel{\text{def}}{=} \phi$, as an abbreviation for $\mathbf{R}(\tilde{\xi}) \stackrel{\text{def}}{=} (\mu\mathbf{R}(\tilde{\mathbf{x}}).\,\phi)(\tilde{\xi})$:

$$\mathbf{exists\_path}(\mathbf{x},\mathbf{y}) \stackrel{\text{def}}{=} \mathbf{x} = \mathbf{y} \vee (\exists \mathbf{z},\mathbf{a}.\ \mathbf{a}(\mathbf{x},\mathbf{z}) \mid \mathbf{exists\_path}(\mathbf{z},\mathbf{y})).$$

This combination of composition and recursion can be regarded as an induction on the graph structure. Consider the graph $a(x,z) \mid b(z,z) \mid c(z,y)$. There are just two ways to check that this graph satisfies the formula: either by checking that edge a is followed by c; or that a is followed by b is followed by c.

Example A classic property associated with compiler optimisation is 'a node z *dominates* node y iff every path from some declared initial node x to y passes through z'. First we specify the property that a graph *is* a path from x to y:

$$\mathbf{path}(\mathbf{x},\mathbf{y}) \stackrel{\text{def}}{=} (\mathbf{x} = \mathbf{y} \wedge \mathsf{nil}) \vee (\exists \mathbf{z},\mathbf{a}.\ \mathbf{a}(\mathbf{x},\mathbf{z}) \mid \mathbf{path}(\mathbf{z},\mathbf{y}))$$

The addition of nil ensures that *all* the edges are checked. For example, in graph $a(x,z) \mid b(z,z) \mid c(z,y)$ the only way that $\mathbf{path}(\mathbf{x},\mathbf{y})$ is satisfied is by checking that a follows b follows c. It is now simple to specify the property we seek:

$$\mathsf{dominates}(x,y,z) \ \stackrel{\text{def}}{=}\ \mathsf{subgraph}_\forall(\mathbf{path}(x,y) \Rightarrow \mathsf{in\_graph}(z)).$$

4 A Query Language

Our basic language consists of *queries* and *transducers*. Queries build new graphs from old. Transducers associate input graphs with output graphs. These concepts are related. The basic transducer $\phi \Rrightarrow Q$ relates input graphs satisfying ϕ with output graphs given by Q. The query (**apply** τ **to** Q) applies the transducer τ to the input graphs given by Q, to yield the corresponding set of output graphs.

Definition 8 (Query Language)
The sets of *pre-queries* and *pre-transducers*, denoted $\mathcal{Q}_{\mathsf{pre}}(\mathcal{X}, \mathcal{A})$ and $\mathcal{T}_{\mathsf{pre}}(\mathcal{X}, \mathcal{A})$ respectively, are given by the grammars from definition 4 and the grammars:

$Q ::=$		queries	$\tau ::=$		transducers
	$\mathbf{G}$	graph variable		$\phi \Rrightarrow Q$	basic transducer
	nil	empty graph		$\lambda\mathbf{G}.Q$	abstraction
	$\alpha(\xi_1, \xi_2)$	edge graph		$\tau \mid \tau$	transducer composition
	$Q \mid Q$	composition		$\tau \vee \tau$	disjunction
	apply τ **to** Q	application		$\exists\mathbf{x}.\tau$	exist. quant. of names
				$\exists\mathbf{a}.\tau$	exist. quant. of labels
				$\mathbf{R_T}$	recursion
				$\mu\mathbf{R_T}.\,\tau$	least fix-pt, $\mathbf{R_T}$ positive

The sets of *queries* and *transducers*, denoted by $\mathcal{Q}(\mathcal{X}, \mathcal{A})$ and $\mathcal{T}(\mathcal{X}, \mathcal{A})$, contain those pre-queries and pre-transducers with no free recursion variables. We use $\mathbf{R_T} \overset{\text{def}}{=} \tau$ to denote $\mathbf{R_T} \overset{\text{def}}{=} \mu\mathbf{R_T}.\,\tau$. We overload notation: $\_|\_$ denotes the composition of formulae, queries and transducers. The connective $\_ \Rrightarrow \_$ has the weakest binding strength; the other connectives are as before. A glaring omission is the absence of a renaming technique for node identifiers, such as Skolemization. Our approach is enough for this paper. Other transducer connectives are feasible. Our choice was determined by our aim to have a simple language in which to express our motivating examples. We describe a more general approach in section 4.1.

Definition 9 (Query interpretation)
Assume name set $\mathcal{X}$ and label set $\mathcal{A}$. Let σ denote a substitution from name and label variables to names and labels respectively, let δ denote a substitution from graph variables to elements of $\mathcal{G}$, and let function ρ map transducer recursion variables to the set $\mathcal{P}(\mathcal{G} \times \mathcal{G})$. The *query interpretation* $[\![\_]\!]_{\sigma;\tau;\rho} : \mathcal{Q}_{\mathsf{pre}} \to \mathcal{P}(\mathcal{G})$ and the *transducer interpretation* $[\![\_]\!]_{\sigma;\rho;\tau} : \mathcal{T}_{\mathsf{pre}} \to \mathcal{P}(\mathcal{G} \times \mathcal{G})$, are defined by a simultaneous induction on the structure of pre-queries and pre-transducers:

$$\begin{aligned}
[\![\mathbf{G}]\!]_{\sigma;\delta;\rho} &= \{G : G \equiv \mathbf{G}\delta\} \\
[\![\mathsf{nil}]\!]_{\sigma;\delta;\rho} &= \{G : G \equiv \mathsf{nil}\} \\
[\![\alpha(\xi_1, \xi_2)]\!]_{\sigma;\delta;\rho} &= \{G : G \equiv \alpha\sigma(\xi_1\sigma, \xi_2\sigma)\} \\
[\![Q_1 \mid Q_2]\!]_{\sigma;\delta;\rho} &= \{G : G \equiv G_1 \mid G_2 \wedge G_1 \in [\![Q_1]\!]_{\sigma;\delta;\rho} \wedge G_2 \in [\![Q_2]\!]_{\sigma;\delta;\rho}\} \\
[\![\mathbf{apply}\ \tau\ \mathbf{to}\ Q]\!]_{\sigma;\delta;\rho} &= \{G' : \exists G.\ (G, G') \in [\![\tau]\!]_{\sigma;\delta;\rho} \wedge G \in [\![Q]\!]_{\sigma;\delta;\rho}\}
\end{aligned}$$

$$\begin{aligned}
\llbracket \phi \Rrightarrow Q \rrbracket_{\sigma;\delta;\rho} &= \{(G, G') : G \in \llbracket \phi \rrbracket_{\sigma;\_} \wedge G' \in \llbracket Q \rrbracket_{\sigma;\delta;\rho}\} \\
\llbracket \lambda \mathbf{G}.\, Q \rrbracket_{\sigma;\delta;\rho} &= \{(G, G') : G' \in \llbracket Q \rrbracket_{\sigma;\delta,\mathbf{G}\mapsto G;\rho}\} \\
\llbracket \tau_1 \mid \tau_2 \rrbracket_{\sigma;\delta;\rho} &= \\
\{(G, G') : G \equiv G_1 \mid G_2 \wedge G' &\equiv G'_1 \mid G'_2 \wedge (G_1, G'_1) \in \llbracket \tau_1 \rrbracket_{\sigma;\delta;\rho} \wedge (G_2, G'_2) \in \llbracket \tau_2 \rrbracket_{\sigma;\delta;\rho}\} \\
\llbracket \tau_1 \vee \tau_2 \rrbracket_{\sigma;\delta;\rho} &= \llbracket \tau_1 \rrbracket_{\sigma;\delta;\rho} \cup \llbracket \tau_2 \rrbracket_{\sigma;\delta;\rho} \\
\llbracket \exists \mathbf{x}.\tau \rrbracket_{\sigma;\delta;\rho} &= \textstyle\bigcup_{x\in\mathcal{X}} \llbracket \tau \rrbracket_{\sigma,\mathbf{x}\mapsto x;\delta;\rho} \\
\llbracket \exists \mathbf{a}.\tau \rrbracket_{\sigma;\delta;\rho} &= \textstyle\bigcup_{a\in\mathcal{A}} \llbracket \tau \rrbracket_{\sigma,\mathbf{a}\mapsto a;\delta;\rho} \\
\llbracket \mathbf{R_T} \rrbracket_{\sigma;\delta;\rho} &= \mathbf{R_T}\rho \\
\llbracket \mu \mathbf{R_T}.\, \phi \rrbracket_{\sigma;\delta;\rho} &= \textstyle\bigcap \{S \in \mathcal{P}(\mathcal{G} \times \mathcal{G}) : \llbracket \phi \rrbracket_{\sigma;\delta;\rho,\mathbf{R_T}\mapsto S} \subseteq S\}
\end{aligned}$$

Example: inverting edges Consider the transducer

$$\exists \mathbf{a}, \mathbf{x}, \mathbf{y}.\ \mathbf{a}(\mathbf{x}, \mathbf{y}) \mid \mathsf{true}\ \Rrightarrow \mathbf{a}(\mathbf{y}, \mathbf{x})$$

It returns one inverted edge of any non-empty input graph. The transducer is *non-deterministic*: given input graph $a(x, y) \mid b(y, x)$, the set of possible output graphs is $\{a(y, x), b(x, y)\}$. Now consider the query

$$\textbf{apply}\ (\exists \mathbf{a}, \mathbf{x}, \mathbf{y}.\ \mathbf{a}(\mathbf{x}, \mathbf{y}) \mid \mathsf{true}\ \Rrightarrow \mathbf{a}(\mathbf{y}, \mathbf{x}))\ \textbf{to}\ \mathsf{input\_graph}$$

When the input graph is $a(x, y) \mid b(y, x)$ the resulting output is either $a(y, x)$ or $b(x, y)$; when the input graph is $a(x, y) \mid a(x, y)$ the result can only be $a(y, x)$.

Example: case analysis The connective $\_ \vee \_$ can be used for case analysis:

$$(\mathsf{nil} \Rrightarrow \mathsf{nil}) \vee (\exists \mathbf{a}, \mathbf{x}, \mathbf{y}.\ \mathbf{a}(\mathbf{x}, \mathbf{y}) \mid \mathsf{true}\ \Rrightarrow \mathbf{a}(\mathbf{y}, \mathbf{x}))$$

Either the input graph is empty and we return the empty output graph. Or the input graph is non-empty and we return an inverted edge.

Example: exact inverted copy We can execute a query against every edge. For example, the transducer relating an input graph with its inverted copy is

$$\mathbf{R_T} \overset{\text{def}}{=} (\mathsf{nil} \Rrightarrow \mathsf{nil}) \vee (\exists \mathbf{a}, \mathbf{x}, \mathbf{y}.\ \mathbf{a}(\mathbf{x}, \mathbf{y}) \Rrightarrow \mathbf{a}(\mathbf{y}, \mathbf{x})) \mid \mathbf{R_T}$$

Either the input graph is empty and we return the empty graph. Or the graph can be split into an edge and the rest of the graph. We return the inverted edge and execute the transducer on the smaller graph. Given the input graph $a(x, y) \mid a(x, y)$, we return the exact inverted copy.

We can adapt this example to execute a query against every edge provided it satisfies a certain logical formula. For example, consider the transducer

$$\begin{aligned}
\mathbf{R_T} \overset{\text{def}}{=}\ & (\mathsf{nil} \Rrightarrow \mathsf{nil}) \vee \\
& \quad (\exists \mathbf{a}, \mathbf{x}, \mathbf{y}.\ ((\mathbf{a}(\mathbf{x}, \mathbf{y}) \wedge \mathbf{x} \neq \mathbf{y} \Rrightarrow \mathbf{a}(\mathbf{y}, \mathbf{x})) \vee (\mathbf{a}(\mathbf{x}, \mathbf{y}) \wedge \mathbf{x} = \mathbf{y} \Rrightarrow \mathsf{nil})) \mid \mathbf{R_T})
\end{aligned}$$

Either the input graph is empty and we return the empty graph. Or the input graph is non-empty and we pick an edge. If the domain and codomain of the edge are different then return the inverted edge; if they are the same then return the empty graph. Apply the transducer to the remaining smaller graph.

Example: transitive closure A standard example is the *transitive closure* of a graph. It illustrates the power of mixing abstraction with recursion. For this example only, we assume the edge labelled set $A = \{a\}$. The following transducer, when applied to graph G, returns the minimum graph TC which contains G and satisfies the property: if $a(x,y)$ and $a(y,z)$ are in TC then so is $a(x,z)$:

$$\begin{aligned}\mathbf{R_T} \overset{\text{def}}{=} \lambda\mathbf{G}.\ &(\neg\exists \mathbf{x},\mathbf{y},\mathbf{z}.\ (a(\mathbf{x},\mathbf{y}) \mid \mathsf{true} \wedge a(\mathbf{y},\mathbf{z}) \mid \mathsf{true} \wedge \neg(a(\mathbf{x},\mathbf{y}) \mid \mathsf{true})) \Rrightarrow \mathbf{G}) \vee \\ &\exists \mathbf{x}\,\mathbf{y},\mathbf{z}.\, a(\mathbf{x},\mathbf{y}) \mid \mathsf{true} \wedge a(\mathbf{y},\mathbf{z}) \mid \mathsf{true} \wedge \neg(a(\mathbf{x},\mathbf{z}) \mid \mathsf{true}) \Rrightarrow \textbf{apply } \mathbf{R_T} \textbf{ to } (\mathbf{G} \mid a(\mathbf{x},\mathbf{z}))\end{aligned}$$

4.1 Generalised Transducers

We generalise the definition of transducers (definition 8). Our approach is simple, but too expressive to implement. The semantic interpretation (definition 11) gives us the flexibility to adapt our choice of basic language if we wish.

Definition 10 (Generalised Transducers)
Assume name set $\mathcal{X}$ and label set $\mathcal{A}$. The set of *generalised pre-transducers*, denoted $\mathcal{GT}_{\mathsf{pre}}(\mathcal{X},\mathcal{A})$, is given by the grammar:

$$\begin{array}{lll} \tau ::= \mathsf{id} & \text{identity} & \mathsf{nil} \quad \text{empty input graph} \\ \qquad\ \tau_1 ; \tau_2 & \text{composition} & \ldots \quad \text{analogous cases from definition 4} \\ \qquad\ \mathbf{G} & \text{graph variable} & \exists\mathbf{G}.\tau\ \text{existential quantification over graphs} \end{array}$$

Generalised transducers relate input and output graphs. A logical formula ϕ regarded as a generalised transducer relates input graphs satisfying ϕ to arbitrary output graphs. The identity transducer relates structurally congruent graphs. The transducer composition $\tau_1;\tau_2$ is relational composition. Identity and composition allows us to specify properties of the output graphs. For example, the transducer $\mathsf{true}; (\phi \wedge \mathsf{id})$ relates arbitrary input graphs with output graphs satisfying ϕ. Queries correspond to such generalised transducers.

Definition 11 (Interpretation of Generalised Transducers)
Assume name set $\mathcal{X}$ and label set $\mathcal{A}$. The query interpretation $[\![\_]\!]_{\sigma;\delta;\rho} : \mathcal{GT} \to \mathcal{P}(\mathcal{G} \times \mathcal{G})$, where σ denotes a substitution from name and label variables to names and labels respectively, δ maps graph variables to graphs, and function ρ maps recursion variables of arity n to functions $\mathcal{X}^n \to \mathcal{R}(\mathcal{G} \times \mathcal{G})$, is defined by induction on the structure of the extended formulae:

$$\begin{aligned} [\![\mathsf{id}]\!]_{\sigma;\delta;\rho} &= \{(G,G') : G \equiv G'\} \\ [\![\tau_1;\tau_2]\!]_{\sigma;\delta;\rho} &= \{(G,G') : \exists G_1.\ (G,G_1) \in [\![\tau_1]\!]_{\sigma;\delta;\rho} \wedge (G_1,G') \in [\![\tau_2]\!]_{\sigma;\delta;\rho}\} \\ [\![\mathbf{G}]\!]_{\sigma;\delta;\rho} &= \{G : \mathbf{G}\delta = G\} \times \mathcal{G} \\ [\![\mathsf{nil}]\!]_{\sigma;\delta;\rho} &= \{G : G \equiv \mathsf{nil}\} \times \mathcal{G} \\ [\![\alpha(\xi_1,\xi_2)]\!]_{\sigma;\delta;\rho} &= \{G : G \equiv \alpha\sigma(\xi_1\sigma,\xi_2\sigma)\} \times \mathcal{G} \\ [\![\tau_1 \mid \tau_2]\!]_{\sigma;\delta;\rho} &= \end{aligned}$$

$$\{(G,G') : G \equiv G_1 \mid G_2 \wedge G' \equiv G'_1 \mid G'_2 \wedge (G_1,G'_1) \in [\![\tau_1]\!]_{\sigma;\delta;\rho} \wedge (G_2,G'_2) \in [\![\tau_2]\!]_{\sigma;\delta;\rho}\}$$

$$
\begin{aligned}
[\![\mathsf{true}]\!]_{\sigma;\delta;\rho} &= \mathcal{G} \times \mathcal{G} \\
[\![\tau_1 \wedge \tau_2]\!]_{\sigma;\delta;\rho} &= [\![\tau_1]\!]_{\sigma;\delta;\rho} \cap [\![\tau_2]\!]_{\sigma;\delta;\rho} \\
[\![\neg\tau]\!]_{\sigma;\delta;\rho} &= (\mathcal{G} \times \mathcal{G}) \setminus [\![\tau]\!]_{\sigma;\delta;\rho} \\
[\![\exists \mathbf{x}.\tau]\!]_{\sigma;\delta;\rho} &= \textstyle\bigcup_{x \in \mathcal{X}} [\![\tau]\!]_{\sigma,\mathbf{x}\mapsto x;\delta;\rho} \\
[\![\exists \mathbf{a}.\tau]\!]_{\sigma;\delta;\rho} &= \textstyle\bigcup_{a \in \mathcal{A}} [\![\tau]\!]_{\sigma,\mathbf{a}\mapsto a;\delta;\rho} \\
[\![\exists \mathbf{G}.\tau]\!]_{\sigma;\delta;\rho} &= \textstyle\bigcup_{G \in \mathcal{G}} [\![\tau]\!]_{\sigma;\delta,\mathbf{G}\mapsto G;\rho} \\
[\![\mathbf{R}(\tilde{\xi})]\!]_{\sigma;\delta;\rho} &= \mathbf{R}\rho(\tilde{\xi}\sigma) \\
[\![(\mu\mathbf{R}(\tilde{\mathbf{x}}).\,\tau)(\tilde{\xi})]\!]_{\sigma;\delta;\rho} &= (\textstyle\bigsqcap\{S \in \mathcal{X}^{|\tilde{x}|} \to \mathcal{P}(\mathcal{G} \times \mathcal{G}) : \lambda\tilde{y}.\,[\![\tau]\!]_{\sigma,\tilde{\mathbf{x}}\mapsto\tilde{y};\delta;\rho,\mathbf{R}\mapsto S} \sqsubseteq S\})(\tilde{\xi}\sigma) \\
&\ \text{where } S \sqsubseteq S' \text{ iff } \forall \tilde{y} \in \mathcal{X}^{|\tilde{x}|}.\, S(\tilde{y}) \subseteq S'(\tilde{y}) \\
[\![\xi_1 = \xi_2]\!]_{\sigma;\rho} &= \mathcal{G} \times \mathcal{G} \ \text{ if } \ \xi_1\sigma = \xi_2\sigma; \quad \emptyset \ \text{ otherwise} \\
[\![\alpha_1 = \alpha_2]\!]_{\sigma;\rho} &= \mathcal{G} \times \mathcal{G} \ \text{ if } \ \alpha_1\sigma = \alpha_2\sigma; \quad \emptyset \ \text{ otherwise}
\end{aligned}
$$

Proposition 12
There exists embeddings $(\_)^\circ : \mathcal{Q}_{\mathsf{pre}} \to \mathcal{GT}_{\mathsf{pre}}$, $(\_)^\circ : \mathcal{F} \to \mathcal{GT}_{\mathsf{pre}}$ and $(\_)^\circ : \mathcal{T}_{\mathsf{pre}} \to \mathcal{GT}_{\mathsf{pre}}$ such that

1. for all queries Q, $[\![Q^\circ]\!]_{\sigma;\delta;\rho} = \mathcal{G} \times [\![Q]\!]_{\sigma;\delta;\rho}$;
2. for all logical formulae ϕ, $[\![\phi^\circ]\!]_{\sigma;\delta;\rho} = [\![\phi]\!]_{\sigma;\_} \times \mathcal{G}$;
3. for all basic transducers τ, $[\![\tau^\circ]\!]_{\sigma;\delta;\rho} = [\![\tau]\!]_{\sigma;\delta;\rho}$.

Proof. The embeddings are give in [CGG01]. The query (**apply** τ **to** Q) is interpreted by the sequential composition. The basic transducer $\phi \Rrightarrow Q$ is interpreted by conjunction. The abstraction $\lambda\mathbf{G}.\,Q$ by the existential quantification on $\mathbf{G}$.

Example Consider the derived transducers:

$$
\begin{aligned}
\mathsf{subgraph} &\overset{\text{def}}{=} \mathsf{id} \mid (\mathsf{nil} \Rrightarrow \mathsf{true}) & \mathsf{strict\_subgraph} &\overset{\text{def}}{=} \mathsf{id} \mid (\mathsf{nil} \Rrightarrow \neg\mathsf{nil}) \\
\tau_1 ;; \tau_2 &\overset{\text{def}}{=} \neg(\tau_1 ; \neg\tau_2) & \mathsf{min\_out}\,(\tau) &\overset{\text{def}}{=} \tau \wedge \neg(\tau ; \mathsf{strict\_subgraph}) \\
\mathsf{finite\_lub}\,(\tau) &\overset{\text{def}}{=} \mathsf{min\_out}(\tau ;; \mathsf{subgraph})
\end{aligned}
$$

The transducer subgraph relates G_1 to G_2 if and only if $G_1 \subseteq G_2$: that is, $G_1 \mid H \equiv G_2$ for some H. The strict_subgraph is the strict version. The connective ;; is the de Morgan dual of ;. Unravelling the definition, it states that

$$
(G, G') \in [\![\tau_1 ;; \tau_2]\!]_{\sigma;\delta;\rho} \Leftrightarrow (\forall G_1.\,(G, G_1) \in [\![\tau_1]\!]_{\sigma;\delta;\rho} \Rightarrow (G_1, G') \in [\![\tau_2]\!]_{\sigma;\delta;\rho})
$$

This operator allows us to work with *all* output graphs associated with a given input. For example, the transducer $\tau;;\mathsf{subgraph}$ relates a graph G with all the finite upper bounds of $[\![\tau]\!](G)$ (where $[\![\tau]\!](G)$ is the set of all graphs G' such that $(G, G') \in [\![\tau]\!]$). These finite upper bounds do not necessarily exist, in which case $[\![\tau;;\mathsf{subgraph}]\!](G)$ is the empty set. We may adapt our finite semantics to the infinite case, by using the infinite version of the set-theoretic presentation given in section 2.1. The $\mathsf{min\_out}(\tau)$ transducer relates a graph G with the minimal graphs in $[\![\tau]\!](G)$. The transducer finite_lub (τ) relates a graph G with the minimal finite upper bound of $[\![\tau]\!](G)$, when it exists. The infinite semantics would give rise to a least upper bound. In the introduction, we discuss a standard set-theoretic language based on from/select expressions. These expressions are embeddable in our general language using this finite-lub construction [CGG01].

We must give an in-depth comparison between our query language and other query languages based on graphs [FFK$^+$97,CM90,BDHS96]. Our language is closely related to XDuce [HP01], a processing language for XML documents based on pattern-matching and a simple typing scheme analogous to the structural component of our spatial logic. Our ambitious aim is to achieve a level of understanding of query languages for semi-structured data which rivals that of languages associated with the relational model.

References

[ABS00] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web.* Morgan Kaufmann, 2000.

[BDHS96] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *SIGMOD*, LNCS 2044, pages 505–515, 1996.

[Cai99] L. Caires. *A Model for Declarative Programming and Specification with Concurrency and Mobility.* PhD thesis, University of Lisbon, 1999.

[CC01] L. Caires and L. Cardelli. A spatial logic for concurrency (part 1). In *TACS*, LNCS 2215. Springer, 2001. Journal paper to be in Information and Comp.

[CG00] L. Cardelli and A. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *POPL*. ACM, 2000.

[CG01a] L. Cardelli and G. Ghelli. A query language based on the ambient logic. In *ESOP/ETAPS*, LNCS 2028. Springer, 2001.

[CG01b] L. Cardelli and A. Gordon. Logical properties of name restriction. In *TLCA*, LNCS 2044. Springer, 2001.

[CGG01] L. Cardelli, P. Gardner, and G. Ghelli. A spatial logic for querying graphs. Fuller version found at http://www.doc.ic.ac.uk/˜pg, 2001.

[CM90] M. Consens and A. Mendelzon. Graphlog: a visual formalism for real life recursion. In *Principles of Database Systems*, pages 404–416. ACM, 1990.

[CMR94] A. Corradini, U. Montanari, and F. Rossi. An abstract machine for concurrent modular systems: Charm. *TCS*, 122:165–200, 1994.

[Cou97] Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. *Graph grammars and computing by graph transformations*, 1:313–400, 1997.

[FFK$^+$97] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. Strudel: A web-site management system. In *SIGMOD Management of Data*, 1997.

[GW00] P. Gardner and L. Wischik. Explicit fusions. *MFCS*, LNCS 1893, 2000. Journal version submitted to Theoretical Computer Science.

[HP01] H. Hosoya and B. Pierce. Regular expression pattern matching for xml. In *POPL*. ACM, 2001.

[IO01] S. Ishtiaq and P. O'Hearn. Bi as an assertion language for mutable data structures. In *POPL*, 664. ACM, 2001.

[OP99] P. O'Hearn and D. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

[Rey00] J.C. Reynolds. Intuitionistic reasoning about shared mutable data structure. *Millenial Perspectives in Computer Science*, Palgrove, 2000.