# A Pattern Based Browsing Model

Jan Hidders and Cora Hoskens
Eindhoven University of Technology (TUE)
P.O. Box 513, 5600 MB Eindhoven
the Netherlands
e-mail: {hidders,cora}@win.tue.nl

Jan Paredaens
University of Antwerp (UIA)
Universiteitsplein 1, B-2610 Antwerp
Belgium
e-mail: pareda@uia.ua.ac.be

## Abstract

*This paper introduces a browsing model that does not describe some new innovative browsing technique but provides a general model to describe intuitive ideas about browsing. In this model it is assumed that the database scheme, as well as the instance of the database are represented by graphs. The most important browsing step in this model is the pattern step. It is based upon finding subgraphs in the instance matching a pattern and supplied with a browsing condition that links it to previous steps. This allows the user to visually specify a browsing step based upon the results of previous steps. Other browsing steps and operators in the model allow the user to randomly select some subgraphs found by a step, replace an old browsing step with a new one or undo some of the last browsing steps. After presenting the model we compare its expressive power with that of the relational algebra.*

**keywords:** browsing, graphs, pattern-matching

## 1 Introduction

Browsing provides a means to investigate the contents of a database in a special way. It adds to querying the possibility to reuse former results. It is like moving around in the database by specifying intermediate results and using these to get more specific ones. In a sequence of steps the user tries to get closer to the information he wants to get.

What characterizes browsing, is that it is an interactive and iterative process of specifying queries and investigating the results of those queries in order to be able to state new ones. This is particularly useful when a user does not know exactly what he is looking for, or how to access the information he is looking for.

An example of a browsing facility is available in Smalltalk [8]. Smalltalk provides the possibility to wander around the class structure in order to find the particular class that the user is looking for. The classes are ordered in a tree structure and by means of browsing one can go from one class to one of its subclasses. Without this possibility it would be very hard to find the classes of interest.

Another way of browsing is available in hypertext documents [7]. In a hypertext, links are provided to other (parts of) documents which in some way or another are related to the current document. By choosing the right links, the user tries to find the document he is interested in. World Wide Web is a very nice example of a (world wide) hypertext [2]. Both examples show the characteristic of browsing: the use of intermediate results in order to get the required result.

This paper does not describe some new innovative browsing technique. It provides a general model to describe intuitive ideas about browsing [3], [4].

In this particular browsing model it is assumed that the database scheme, as well as the instances of the database are represented by graphs, such as in the GOOD model [1], [5], [6]. In this model, the nodes of the graph represent objects while the edges represent the properties of and the relationships between the objects.

A pattern is also a graph, and can be used to select subsets of an instance by the notion of pattern matching. This means that every subgraph of an instance that matches the pattern is selected. Such a subgraph is called an *embedding*.

With the pattern browsing technique, every possible action that the user can take is called a *browsing statement*. Browsing statements can be divided into *browsing steps* and *browsing operators*. There are two different steps in the model and two different operators. Actually, the steps are the elementary browsing statements. They provide the user with all necessary actions to enable him to browse. The operators, on the other hand, are meant to ease the task of the user. We first discuss the two possible browsing steps.

First of all, embeddings can be selected by specifying a pattern. As such it is a pattern matching step, or *pattern step* for short. The nodes in the pattern are labeled, and they can also be given a condition. Such a condition is called a *node condition*. By means of these node conditions, the user can restrict the subgraphs that are selected. The node conditions put an extra restriction on embeddings. Thus,

an *embedding* is a subgraph that matches the pattern and fulfills the node conditions.

However, in the context of browsing, a more general condition can apply to a pattern step, which is therefore called the *browsing condition*. This condition links the pattern to embeddings found in previous steps, and is therefore essential for the process of browsing. In such a way, it is possible to combine the results of several steps by combining conditions, that refer to different steps, into one condition.

Secondly, it is possible to select a set of embeddings amongst the ones that resulted from a previous step. Such a step is called the *selection step*. That way it is possible to select only the embeddings that seem interesting to the user. It narrows down future searches.

Besides these two steps, there are two operators. Each of these operators changes, in fact, a browsing program, which is a sequence of browsing steps. The first one, the *change operator* replaces one step in the program by a new one. This may affect the result of the changed step and also of later steps that refer to that step. The second operator, on the other hand, the *rollback operator*, rolls back one or more of the last browsing steps.

The operators are introduced because they add to the intuitive ideas about browsing. Browsing means investigating the contents of the database in an interactive manner. The possibility to backtrack (as provided by the change and rollback operator), and retrace steps (as with the change operator) can be part of that process. Including the operators makes it possible to have more user friendly browsings sessions.

It is possible to show the results of subsequent statements in a tree, as will be shown later in this paper. The embeddings that are found in the pattern and selection steps make up the nodes of this tree, while each edge is labeled with the step label. Each of these steps adds one layer to the browsing tree. The layer that contains the result of step $l$ is called layer $l$. The browsing tree is a visual aid for the representation of the result. It not only shows the results of each step, but also shows (by means of the edges) the dependencies between these results.

Every node in a pattern has a unique node label. Moreover, we presume that these node labels of a pattern are linearly ordered and that it is therefore also possible to speak about e.g. the third node of a pattern. Consequently, embeddings can be represented by unlabeled tuples.

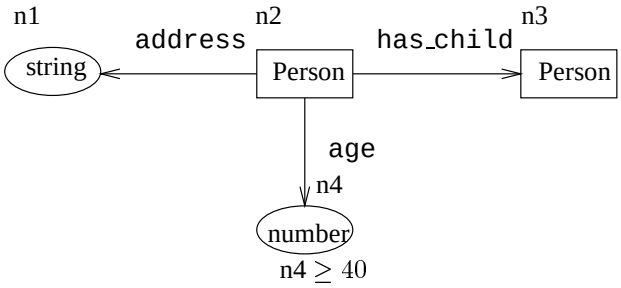
## 2 An example

First an informal example is given that illustrates the notion of browsing and the intended meaning of the different statements. In this example steps as well as operators are used.

**Example 1** *The example uses a small instance, given in Figure 1. In this instance the objects have an identification label, which makes it possible to distinguish between them.*

*The instance represents four persons with their name, address and age (if those are known) together with their parent–child relationships.*

```
-------------------------------------
Statement 1 (pattern step):
 label: l1
 pattern:
```



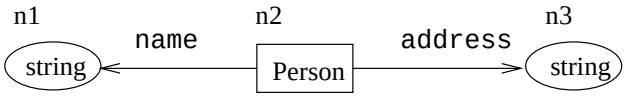

```
 condition: T
-------------------------------------
```

*This query asks for all persons that are at least 40 years of age, together with their address and one of their children.*

*The result of this statement is a set of embeddings that match the pattern. Every embedding can therefore be represented by a (address, person, person, age)–tuple. The result can be represented by the following table:*

| *n1* | *n2* | *n3* | *n4* |
|---|---|---|---|
| *Antwerp* | *p1* | *p2* | *40* |
| *Antwerp* | *p1* | *p3* | *40* |

```
-------------------------------------
Statement 2 (pattern step):
 label: l2
 pattern:
```




```
 condition: (n2,n3)=anc[l1](n3,n1)
-------------------------------------
```

*Node n2 of the instance (Person) has to match node n3 of a tuple of step l1, which is the 'child' object of that pattern. In the same way, node n3 of the instance (Address) has to match node n1 of the same tuple, which is the 'address of*

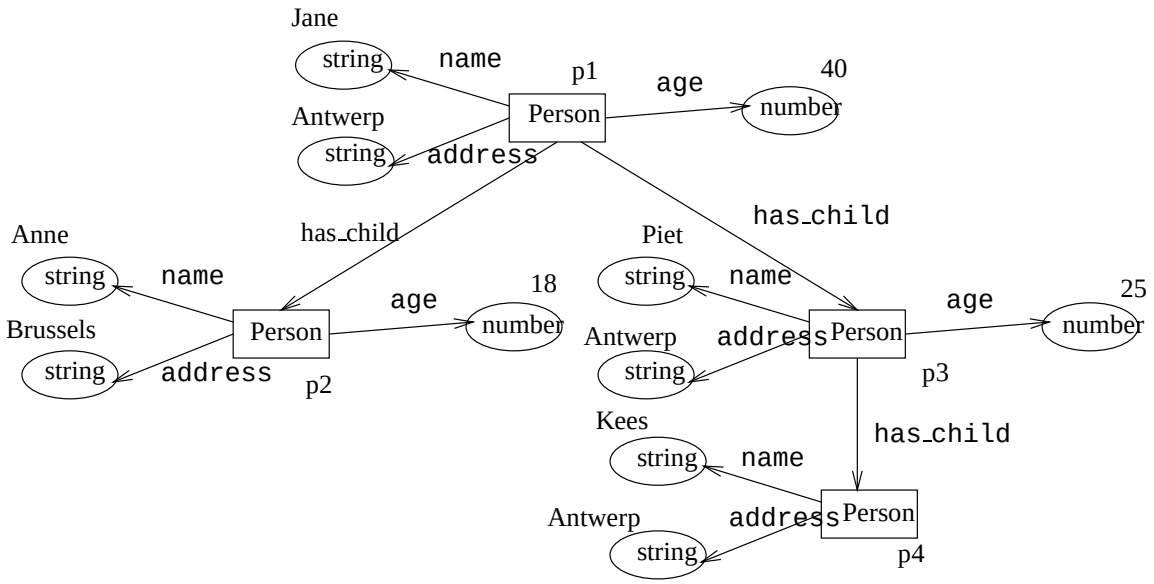
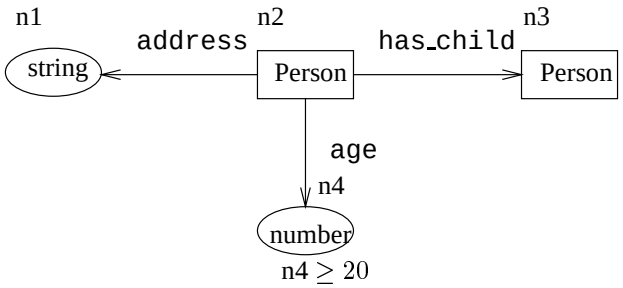



Figure 1: A simple instance

*the parent' of that pattern. This means that this query asks for all children who live at the same address of one of their parents, for those parents that were selected in step l1, together with their name and address. The following table will be the result:*

| *n1* | *n2* | *n3* |
|---|---|---|
| *Piet* | *p3* | *Antwerp* |

```
----------------------------------------
Statement 3 (change operator):
 change:
  ( label: l1
    pattern:
```




```
    condition: T )
----------------------------------------
```

*This operator will change the original step l1. This adapted version of step l1 asks for all persons that are at least 20 years of age, together with their address and one of their children. Step l1 and l2 are executed again, which will now result in the following tables:*

*Step l1:*

| *n1* | *n2* | *n3* | *n4* |
|---|---|---|---|
| *Antwerp* | *p1* | *p2* | *40* |
| *Antwerp* | *p1* | *p3* | *40* |
| *Antwerp* | *p3* | *p4* | *25* |

*Step l2:*

| *n1* | *n2* | *n3* |
|---|---|---|
| *Piet* | *p3* | *Antwerp* |
| *Kees* | *p4* | *Antwerp* |

```
----------------------------------------
Statement 4 (selection step):
 label: l3
 from: l2
 select:
```

| *n1* | *n2* | *n3* |
|---|---|---|
| *Kees* | *p4* | *Antwerp* |

```
----------------------------------------
```

*This query can be formulated as follows: select from the embeddings that where found in step l2, exactly the ones given in the table.*

*The result of this statement is that the given embeddings are selected. This narrows down future searches as will be visualized when the results are presented in a browsing tree further on in this paper. The result is given by the following table:*

| *n1* | *n2* | *n3* |
|---|---|---|
| *Kees* | *p4* | *Antwerp* |

*Now imagine that the user finds that he made the wrong selection in step l3. What he can do now, is either change*

*it with a change operator, or throw away step l3 altogether by means of a rollback operator. Imagine that he wants to do the latter. The fifth statement would then look like:*

```
--------------------------------------
Statement 5 (rollback operator):
 rollback_to: l3
--------------------------------------
```

*This query can be translated as: throw away the steps including and after l3. The result of this step is that the result of step l3 is removed, and can no longer be used in subsequent statements.*

This concludes the example. Notice that steps have a `label:` but operators do not. Therefore only steps can be changed or rolled-back to.

# 3 Browsing programs

Here we introduce the notion of a browsing program. This is a sequence of browsing statements. The syntax of the steps and operations are given by the following definitions.

**Definition 1** *A* pattern step *has the following syntax,*

(`label:` $l$, `pattern:` $p$, `condition:` $c$)

*consisting of a label $l$, a pattern $p$ and a pattern condition $c \in C$, where $C$ is defined by:*

$$\begin{aligned} C \quad ::= \quad & (n_1, \ldots, n_k) = \texttt{anc}[l'](m_1, \ldots, m_k) \mid \\ & (n_1, \ldots, n_k) = \texttt{any}[l'](m_1, \ldots, m_k) \mid \\ & \texttt{NOT}(C) \mid (C \ \texttt{AND}\ C) \mid (C \ \texttt{OR}\ C) \mid \texttt{T}. \end{aligned}$$

*where $n_1, \ldots, n_k$ and $m_1, \ldots, m_k$ are node labels, and $l'$ is a step label.*

*All nodes of the pattern are labeled with a unique identifier. In the pattern, each node may have a node condition. The syntax of the node condition of a node $n_i$ is as follows:*

$$\begin{aligned} NC ::= \quad & n_i = e \mid n_i \leq e \mid n_i < e \mid n_i \geq e \mid n_i > e \mid \\ & \texttt{NOT}(NC) \mid (NC \ \texttt{AND}\ NC) \mid (NC \ \texttt{OR}\ NC). \end{aligned}$$

*where expression $e$ is either a node label, a value, or an numerical expression (in which the binary operators +, −, *, / can be used) containing values and node labels. The node labels in the expression must all be in pattern $p$.*

**Definition 2** *A* selection step *has the following syntax,*

(`label:` $l$, `from:` $l'$, `select:` $S$)

*consisting of a label $l$, a label $l'$ indicating the step of which the embeddings are to be selected and a selection set $S$ containing the selected embeddings.*

**Definition 3** *A* change operation *has the following syntax,*

(`change:` $s$)

*consisting of a step $s$ with label $l$, that should replace the preceding step in the program with label $l$.*

**Definition 4** *A* rollback operation *has the following syntax,*

(`rollback_to:` $l$)

*consisting of a step label $l$ indicating from which step on the results are to be removed.*

Not every list of browsing steps and operations is well formed. For instance, there can not be two steps in the program with the same label. Also, statements can only refer to preceding steps that were not undone by a rollback operation. Furthermore, it should hold that if the browsing condition of a pattern step contains an expression of the form $(n_1, \ldots, n_k) = \texttt{anc}[l'](m_1, \ldots, m_k)$ (or the same with `any`) then $n_1, \ldots, n_k$ are node labels in the pattern of the pattern step, and $m_1, \ldots, m_k$ are node labels embedded by the result of step $l'$. The exact rules for well-formedness are, however, beyond the scope of this paper.

The intermediate results of a browsing program are stored in a tree; the browsing tree. Every step extends the browsing tree with a new layer. For each node in the bottom layer a (possibly empty) set of embeddings is added to the tree. The edges from the bottom node to the embeddings will be labeled with the label of the step. The layer that contains the result of step $l$ is called layer $l$.

**Definition 5** *A* browsing tree *is a labeled tree where each edge is labeled with a step label and each node is labeled with an embedding.*

We will now present the semantics of the different browsing steps and operations by looking at the intermediate results of the program in Example 1.

First, every program starts with the *empty browsing tree*, i.e., the browsing tree consisting of only the root labeled with the empty embedding. The first statement in the example program is a pattern step. A pattern step adds to every node in the bottom layer (here only the root node) an edge and a node for every embedding of the pattern that fulfills the node conditions and browsing condition (here `T` is always true). The edge is labeled with the step label and the node is labeled with the embedding. The result of the first statement is shown in Figure 2.

The second statement in the example is a pattern step with a more complex browsing condition. The condition `(n2,n3)=anc[l1](n3,n1)` is fulfilled for a bottom layer node by an embedding if it maps the nodes n2 and n3 to, respectively, the same instance nodes, as n3 and n1 are mapped to, by the *ancestral embedding* of step l1. The ancestral embedding of a step $l$ is the embedding in the result of step $l$ that is encountered on the path from the bottom layer node to the root of the browsing tree. For example,

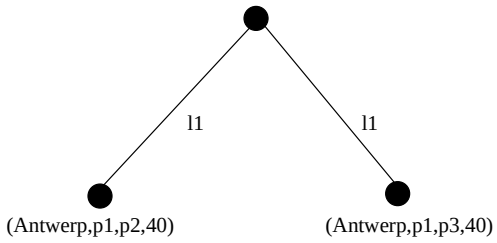



Figure 2: Result of the first statement

in Figure 2 the ancestral embedding of step l1 for the left bottom layer node is (Antwerp,p1,p2,40), and for the right bottom layer node it is (Antwerp,p1,p3,40). Notice that if a browsing condition is evaluated for a certain bottom layer node, the ancestral embedding of a step is unique if it exists. The result of the second statement is shown as the solid part of Figure 3.

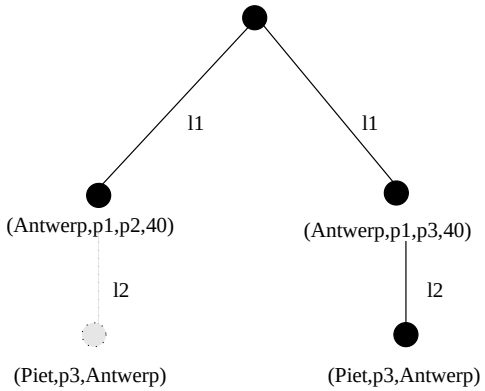



Figure 3: Result of the second statement

The condition `(n2,n3)=any[l1](n3,n1)` is fulfilled by an embedding $e$ if there is *any* embedding in the result of step l1 that maps the nodes n2 and n3 to, respectively, the same instance nodes that n3 and n1 are mapped to by the embedding $e$. If this condition would have been the browsing condition of the second statement then both bottom layer nodes would have been extended with the same set of embeddings. The result would have been as shown in Figure 3 including the dotted edge and node.

The third statement of the example is a change operator. A change operator replaces an old pattern or selection step with a new one and recalculates the resulting browsing tree. The result of the third statement is shown in Figure 4.

The fourth statement of the example is a selection step. It adds to every node in the bottom layer its own embedding if the ancestral embedding of step l2 is in the selection set. The result is shown in Figure 5 ignoring the cutting edge.

The fifth and last statement of the example is a rollback operation. It removes the result of step l3 and the layers below, from the browsing tree. This is demonstrated by the

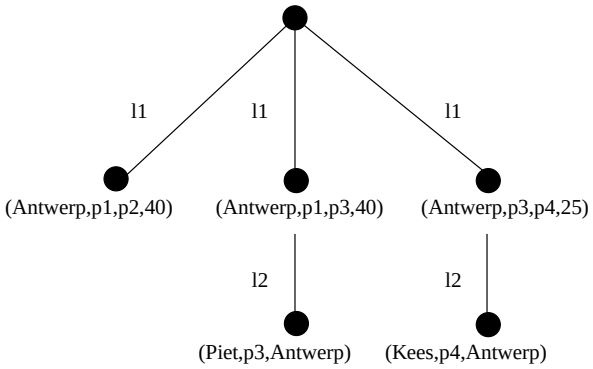



Figure 4: Result of the third statement

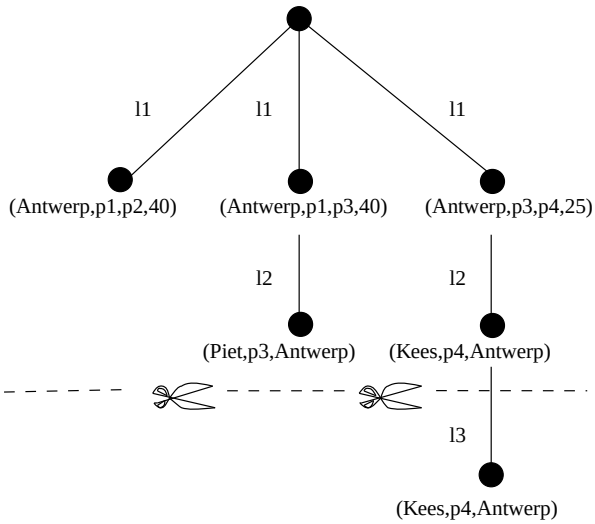



Figure 5: Result of the fourth and fifth statement

cutting edge in Figure 5.

# 4 The expressive power of the browsing model

In this section we discuss the theoretical expressive power of the browsing model. With expressive power we mean the ability to calculate certain transformations. The browsing model is compared with the relational algebra in several ways. The relevance of this comparison is to find out whether the provided browsing steps are expressive enough to let a user obtain all the results that he also might have found using, for example, SQL.

We will consider only the model without operations since these can be regarded as syntactic sugar. Also, we do not consider arithmetic in the node conditions because this would make the model incomparable with the relational algebra.

There is an obvious link between the relational model and our browsing model since the result of every step can be regarded as a relational table. An obvious question is therefore whether any relational algebra expression that combines the intermediate results, can be simulated with a browsing program. The answer to this question is given in the following theorem.

**Theorem 1** *Given a browsing tree $bt$ with layers $l_1, \ldots, l_n$ and a relational algebra expression $ae$ over the tables $l_1, \ldots, l_n$ containing the results of the layers with the same name, there is a browsing program that extends the browsing tree $bt$ such that the bottom layer contains exactly all the embeddings that are tuples in the result of $ae$.*

**Proof:** *(Sketch)* Every single algebra operator can be simulated by a pattern step. The complete algebra expression can be simulated by concatenating these pattern steps. □

The expressive power of the browsing model as stated in Theorem 1 is slightly "crude". It says something about the manipulation of complete layers whereas in the browsing model every step calculates an extension for *every separate bottom layer node*. The theorem also holds, for example, for the browsing model without the `anc`-conditions. It is possible to give a more subtle definition of expressive power by letting the algebra expression $ae$ also be defined over the tables $l'_1, \ldots, l'_n$ containing the ancestral embedding of every step. Then, $ae$ could calculate a separate extension for every bottom layer node of the browsing tree. (A bottom layer node is always uniquely identified by the ancestral embeddings on the path to the root.) A browsing program would then simulate $ae$ if it adds to every bottom layer node a subtree that contains exactly the result of $ae$ in its bottom layer. The question whether the browsing model is as expressive as the relational algebra in this way, is still open.

Up to now we have regarded the expressive power of the browsing model as a manipulation language for browsing trees. Although the browsing tree is interesting to show the intermediate results, the user might be more interested in the final result (the set of embeddings in the bottom layer) he can obtain when starting with a certain instance. Therefore, it may be more interesting to see which sets of embeddings can be found given a certain instance and starting with the empty browsing tree. Although the instance is defined as a graph it can be straightforwardly translated to a relational database consisting of unary and binary tables representing, respectively, the nodes and the edges.

**Theorem 2** *Given an instance $I$ with node labels $N_1, \ldots, N_k$ and edge labels $E_1, \ldots, E_l$ and a relational algebra expression $ae$ defined over the unary tabels $N_1, \ldots, N_k$ and binary tables $E_1, \ldots, E_l$, then there is a browsing program that, starting with an empty browsing tree, results in a browsing tree with the result of $ae$ in its bottom layer.*

**Proof:** *(Sketch)* Every unary and binary table can be encoded in a layer with a simple pattern step. Then we can simulate $ae$ using Theorem 1. □

Under the same interpretation we might also ask whether every transformation expressed by the browsing model can be expressed in the relational algebra.

**Theorem 3** *Given an instance $I$ with node labels $N_1, \ldots, N_k$ and edge labels $E_1, \ldots, E_l$ and browsing program $bp$, there is a relational algebra expression $ae$ defined over the unary tables $N_1, \ldots, N_k$ and binary tables $E_1, \ldots, E_l$ that results in a table containing exactly those embeddings found in the bottom layer of the browsing tree that is the result of $bp$ starting with an empty browsing tree.*

**Proof:** *(Sketch)* First, it is proven that any pattern step and selection step can be simulated in the relational algebra. Since these steps operate upon the instance graph and the browsing tree we need to encode these into relational tables. The encoding of the instance graph is straightforward. The browsing tree is mapped to a table for every layer, containing all the embeddings in that layer. Furthermore, the entire browsing tree is mapped to a large table that contains for every bottom layer node a tuple consisting of the concatenation of all its ancestral embeddings. An example of this mapping is given in Figure 6 where the encoding of the browsing tree in Figure 4 is shown. This encoding is not lossless but it is enough to simulate the browsing model. It can now be shown that given a browsing step there are algebra expressions over the encoding of the instance and the previous browsing tree, that result in the tables constituting the encoding of the resulting browsing tree. Finally, the entire browsing program can be simulated by concatenating the simulations of individual browsing steps. □

Theorem 2 only shows that the browsing model can express the relational algebra operating on *unary and binary* relations. Relational tables of arbitrary arity can, however, be encoded as graphs as shown in Figure 7. Here the top node labeled with $R$, the name of the relation, represents the relation itself. This node has *element*-edges to zero or more nodes representing the tuples in the relation. These tuple nodes are labeled with $T$ and have edges for every field $a_1, a_2, \ldots, a_p$ of the tuples in $R$. These edges are labeled with the name of the field and arrive in a node representing the value of that field.

| l1 | | | |
|---|---|---|---|
| n1 | n2 | n3 | n4 |
| Antwerp | p1 | p2 | 40 |
| Antwerp | p1 | p3 | 40 |
| Antwerp | p3 | p4 | 25 |

| l2 | | |
|---|---|---|
| n1 | n2 | n3 |
| Piet | p3 | Antwerp |
| Kees | p4 | Antwerp |

| $BT$ | | | | | | |
|---|---|---|---|---|---|---|
| l1.n1 | l1.n2 | l1.n3 | l2.n4 | l2.n1 | l2.n2 | l2.n3 |
| Antwerp | p1 | p3 | 40 | Piet | p3 | Antwerp |
| Antwerp | p3 | p4 | 25 | Kees | p4 | Antwerp |

Figure 6: The browsing tree of step l2 encoded in tables

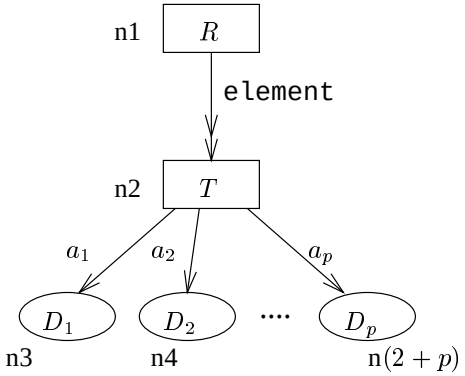



Figure 7: Pattern for representing simulated tables in layers

**Theorem 4** *Given a relational database containing the relations $R_1, \ldots, R_n$ and a relational algebra expression $ae$ over these relations, there is a browsing program $bp$ that, when starting with the empty browsing tree and an instance containing the encoding of $R_1, \ldots, R_k$ in a graph, results in a browsing tree with the result of $ae$ in its bottom layer.*

**Proof:** *(Sketch)* For every relation we can use a pattern step with the pattern of Figure 7 to encode the table into a layer. Then we can use Theorem 1 to simulate the relational algebra expression. □

# 5 Conclusions

In this paper we have introduced a browsing model that provides a general model to describe intuitive ideas about browsing. The model assumes that the database scheme, as well as the instance of the database are represented by graphs. This makes it possible to specify browsing steps with the help of pattern matching. Together with browsing conditions that can link a pattern to patterns of previous steps, this constitutes a simple and effective way of describing browsing steps. Apart from specifying patterns with conditions, the model also offers operations that allow the user to select an arbitrary subset of the result, recalculate the previous steps after replacing an old step with a new step and undoing the last steps. Finally, the model was shown to be expressive enough to simulate the relational algebra in several ways.

# References


[1] M. Andries, M. Gemis, J. Paredaens, I. Thyssens, and J. van den Bussche. Concepts for Graph-oriented Object Manipulation. volume 580 of *Lecture Notes in Computer Science*, Berlin, March 1992. Springer–Verlag. Proceedings of the third EDBT conference held in Vienna, Austria.

[2] T.J. Berners-Lee, R. Cailliau, J-F Groff, and B. Pollermann. World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, 2(1):52–58, 1992.

[3] A. D'Atri and L. Tarantino. From Browsing to Querying. *Data Engineering*, 12(2):46–53, June 1989.

[4] A. D'Atri and L. Tarantino. A Browsing Theory and its Application to Database Naviagation. In J. Paredaens and L. Tenenbaum, editors, *Advances in database systems, implementations and applications*, number 347 in CISM Courses and Lectures, pages 161–179. Springer–Verlag, Wien, New–York, 1994.

[5] M. Gemis and J. Paredaens. An Object-Oriented Pattern Matching Language. volume 742 of *Lecture Notes in Computer Science*, pages 339–355, Berlin, 1993. Springer. Proceedings of the international symposium on Object Technologies for Advanced Software; held in Kanazawa, Japan.

[6] M. Gyssens, J. Paredaens, J. Van den Bussche, and D. Van Gucht. A Graph-Oriented Object Database Model. *IEEE Transactions on Knowledge and Data Engineering*, 6(4):572–586, August 1994.

[7] J. Nielsen. *Hypertext & Hypermedia*. Academic Press, Inc., Dan Diego, CA, 1990.

[8] L. J. Pinson and R. S. Wiener. *An Introduction to Object-Oriented Programming and Smalltalk*. Addison-Wesley, Amsterdam, 1988.