4.1 Path Expressions

Semistructured Data Model: Edge-Labeled Graph

... In a sequence of edge labels

Path Expression: 11.12. ... In a sequence of edge labels

The path expression ends.

A path expression may be viewed as a simple query whose
result is a set of nodes in the edge-labeled graph where

{"with the string": "Compaq", "Roux", "Smith"

Consider the data in Figure 4.1 (Page 56)

{\{\"book\", \"author\"\} results in the set of nodes \{ni, n2\}

{\{\"book\", author\} results in the nodes associated
The path expressions can be expressed in terms of some properties.

paths that form the edge labels to describe such properties.
The general syntax for regular expressions on paths is

\( (s|s)(\text{section}|\text{paragraph})? (s|s) \)

For example, to specify more complex label patterns, we use \( \text{grep} \) patterns:

\[ e ::= \begin{array}{l}
\text{ } \quad \text{ } \quad \text{ } \\
1 \quad \text{ } - \quad \text{ } \quad \text{ } \\
(\text{e}) \quad (\text{e}) \quad (\text{e}) \quad (\text{e}) \\
\text{+} \quad \text{*} \quad \text{?} \quad \text{?} \\
\end{array} \]

sections, sections, sections, sections, paragraphs, paragraphs
matches any one of six patterns:
The string "heading"

or paragraph edge followed by an edge whose label contains
with a section label followed by either title or little edge
eaches any path that starts with a block label and ends

biblio.*.section."title" | paragraph."heading."heading.*""

Labels are enclosed within quotes:
and those for path expressions, the regular expressions on
To avoid ambiguity between regular expressions for labels
Query Language features will be necessary for this.

which requires joining ability.
They cannot produce semi-structured data
nodes of the Data Graph.
Path expressions produce a set of
4.2. A Core Language
{"author": "Roux", "author": "Combesclus", "author": "Smith"}

and expression out of the nodes: computes the set of book authors and forms a

from biblio.book.author X

select author X

% Query o1 based on OQL (Object Query Language)

4.2.1 The Basic Syntax
be the node represented by x. Here x.author is a path expression whose root is taken to be the root of the node. The "in" predicate tests for set membership.

{ ... row: { author: "Smith", date: 1999, title: "Database Systems" }, ... }

computes the answer:

WHERE "Smith" IN x.author
FROM bibliography x
SELECT row: x
% Query 92
The following query collects all authors of publications whose title consists of the word "database".

Assume a matches predicate exists which matches strings to regular expressions.
variables \( x, y, z \) are replaced by \( x_1, y_1, z_1 \) in expression \( E \). Where the expression \( E(x_1, y_1, z_1) \) denotes the expression \( E \) in which the

\[ (x_1, y_1, z_1) \] where

Step 3: Construct the SSD-expression

Step 2: Filter the bindings that satisfy \( \mathcal{C} \); let the resulting data graph.

Each binding maps the variables to objects in the

Step 1: Find the set of all bindings of the variables

Select \( E \) from \( B \) where \( \mathcal{C} \)

Semantics of query
The result will be constructed as follows:

```
some queries create more than one new node:
```
See Figure 4.1 (d) for a graphical representation of the query output. Compare with 4.1 (p), answer to query q1.

```sql
{"row": {"author": "Smith"},
{"row": {"author": "Roux", "author": "Compaust"},
{"row": {"author": "Roux", "author": "Compaust"},
{"row": {"author": "Smith"}}
```

The output of this query is:

```
from biblio.book X
from X.author Y
select row (select author: Y)%
```

Another means of creating many new nodes is by

nesting subqueries in the select clause.
shown in graphicaL form in Figure 4.1 (e)

```sql
ROW: {author: "CompuUser", title: "Database Systems"},
{author: "Roux", title: "Database Systems"},
```

Output of q4:

```sql
WHERE "Roux" IN X.author
FROM book, book X
(SELECT row: (select author, title) T)
```

Another nested-select query.
Rows have at least one common value. A join will take place if the two sets of B values in the table overlap. Multiple B values were allowed in r1 and r2.

```
where B=B'
from r1, r2
select a, c:
% query q JOIN
project(a, c)(r1 JOIN r2)
{ { { { c: 3, b: 2, row: 4, } } b: 2, row: 3, } } b: 2, row: 4,}
{ { { c: 1, b: 1, row: 2, } } b: 2, row: 3, }
{ { c: 1, b: 2, row: 1, } } b: 2, row: 4,}
```

Join examples:
matches("Database", X.title)
M in Z.author and
where NOT (Y=Z) and
Y.author M, X.refers-to Z
from bibliography X, X.refers-to Y,
select row: M

Get authors who are referred to at least twice in some paper with "Database" in the title.

Another join example:
{ answer: "Bouy", answer: "Spinal", answer: "Smith"

So, the answer to above query will be

default label in lore is answer

from pluto.book.author x
select x

commission of labels:

core language plus syntactic shortcuts.

lore: lightweight object repository

4.3 More on lor (core language; query language for lore)
(from x.author: y
    select author: y
    x.author can be understood as the nested query
    from biblio.book x
    select x.author
    % Query I

Use of Path Expressions in Select Clause:
In general, an expression of the form:

\[ X.p.l, \]

where \( p \) is an arbitrarily complex path expression, is understood as the nested query

\[
\begin{align*}
\text{select } & l:Y \\
\text{from } & \quad X.p.l \ Y
\end{align*}
\]