

Formalizing GQL

presentation by Leonid Libkin

University of Edinburgh and RelationalAI

LDBC TUC 2023

Standards are great but not for academics

Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax

Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax

```
IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

Function
Specify a pattern to match a single path in a property graph.

Format
<path pattern expression> ::=
  <path term>
  | <path multiset alternation>
  | <path pattern union>
<path multiset alternation> ::=
  <path term> <multiset alternation operator> <path term>
  [ { <multiset alternation operator> <path term> }... ]
<path pattern union> ::=
  <path term> <vertical bar> <path term> [ { <vertical bar> <path term> }... ]
<path term> ::=
  <path factor>
  | <path concatenation>
<path concatenation> ::=
  <path term> <path factor>
<path factor> ::=
  <path primary>
  | <quantified path primary>
  | <questioned path primary>
<quantified path primary> ::=
  <path primary> <graph pattern quantifier>
<questioned path primary> ::=
  <path primary> <question mark>
NOTE 131 — Unlike most regular expression languages, <question mark> is not equivalent to the quantifier {0,1}; the
quantifier {0,1} exposes variables as group, whereas <question mark> does not change the singleton variables that it exposes
to group. However, <question mark> does expose any singleton variables as conditional singletons.
<path primary> ::=
  <element pattern>
  | <parenthesized path pattern expression>
  | <simplified path pattern expression>
<element pattern> ::=
  <node pattern>
  | <edge pattern>
<node pattern> ::=
  <left paren> <element pattern filler> <right paren>
<element pattern filler> ::=
  [ <element variable declaration> ]
  [ <is label expression> ]
  [ <element pattern predicate> ]
  + WS1924-022 +
<element variable declaration> ::=
```

Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax

IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

Function

Specify a pattern to match a single path in a property graph.

Format

```
<path pattern expression> ::=
  <path term>
  | <path multiset alternation>
  | <path pattern union>
  | <path multiset alternation> ::=
    <path term> <multiset alternation operator> <path term>
    [ { <multiset alternation operator> <path term> }... ]
  | <path pattern union> ::=
    <path term> <vertical bar> <path term> [ { <vertical bar> <path term> }... ]
  | <path term> ::=
    <path factor>
    | <path concatenation>
  | <path concatenation> ::=
    <path term> <path factor>
  | <path factor> ::=
    <path primary>
    | <quantified path primary>
    | <questioned path primary>
  | <quantified path primary> ::=
    <path primary> <graph pattern quantifier>
  | <questioned path primary> ::=
    <path primary> <question mark>
```

NOTE 1) — Unlike most regular expression languages, <question mark> is not equivalent to the quantifier {1}. The quantifier {1} exposes variables as group, whereas <question mark> does not change the singleton variables that it exposes to group. However, <question mark> does expose any singleton variables as conditional singletons.

```
<path primary> ::=
  <element pattern>
  | <parenthesized path pattern expression>
  | <simplified path pattern expression>
  | <node pattern>
  | <edge pattern>
  | <node pattern> ::=
    <left paren> <element pattern filler> <right paren>
  | <element pattern filler> ::=
    [ <element variable declaration> ]
    [ <is label expression> ]
    [ <element pattern predicate> ]
    | <is or colon> ::=
      <is or colon> ::=
      is
      | <colon>
```

221

IWD 39075:202y(E)
16.10 <path pattern expression>

```
[ <TMP> ] <element variable>
<is label expression> ::=
<is or colon> <label expression>
is
| <colon>
<element pattern predicate> ::=
  <element pattern where clause>
  | <element property specification>
  | <element pattern where clause> ::=
    WHERE <search condition>
  | <element property specification> ::=
    <left brace> <property key value pair list> <right brace>
  | <property key value pair list> ::=
    <property key value pair> [ { <comma> <property key value pair> }... ]
  | <property key value pair> ::=
    <property name> <colon> <value expression>
  | <edge pattern> ::=
    <full edge pattern>
    | <abbreviated edge pattern>
  | <full edge pattern> ::=
    <full edge pointing left>
    | <full edge pointing right>
    | <full edge left or undirected>
    | <full edge left or undirected> ::=
      <full edge undirected or right>
      | <full edge left or right>
      | <full edge any direction>
      | <full edge pointing left> ::=
        <left arrow bracket> <element pattern filler> <right bracket minus>
      | <full edge undirected> ::=
        <tilde left bracket> <element pattern filler> <right bracket tilde>
      | <full edge pointing right> ::=
        <minus left bracket> <element pattern filler> <bracket right arrow>
      | <full edge left or undirected> ::=
        <left arrow tilde bracket> <element pattern filler> <right bracket tilde>
      | <full edge undirected or right> ::=
        <tilde left bracket> <element pattern filler> <bracket tilde right arrow>
      | <full edge left or right> ::=
        <left arrow bracket> <element pattern filler> <bracket right arrow>
      | <full edge any direction> ::=
        <minus left bracket> <element pattern filler> <right bracket minus>
```

**** Editor's Note (number 73) ****

In the BNF for <full edge any direction>, the delimiter tokens <-[]-> have been suggested as a synonym for [] as part of Feature GAO7, "Undirected edge pattern". The synonym for the abbreviated edge pattern - <minus sign> would then be <->, the synonym for <simplified defaulting any direction> would use the delimiter tokens <- /-> and the synonym for

222

Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax

```
IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

Function
Specify a pattern to match a single path in a property graph.

Format
<path pattern expression> ::=
  <path term>
  | <path multiset alternation>
  | <path pattern union>

<path multiset alternation> ::=
  <path term> <multiset alternation operator> <path term>
  | { <multiset alternation operator> <path term> }... ]

<path pattern union> ::=
  <path term> <vertical bar> <path term> [ { <vertical bar> <path term> }... ]

<path term> ::=
  <path factor>
  | <path concatenation>

<path concatenation> ::=
  <path term> <path factor>

<path factor> ::=
  <path primary>
  | <quantified path primary>
  | <questioned path primary>

<quantified path primary> ::=
  <path primary> <graph pattern quantifier>

<questioned path primary> ::=
  <path primary> <question mark>

NOTE 131 — Unlike most regular expression languages, <question mark> is not equivalent to the quantifier {0,1}; the
quantifier {0,1} exposes variables as group, whereas <question mark> does not change the singleton variables that it exposes
to group. However, <question mark> does expose any singleton variables as conditional singletons.

<path primary> ::=
  <element pattern>
  | <parenthesized path pattern expression>
  | <simplified path pattern expression>

<element pattern> ::=
  <node pattern>
  | <edge pattern>

<node pattern> ::=
  <left paren> <element pattern filler> <right paren>

<element pattern filler> ::=
  [ <element variable declaration> ]
  | <is label expression> ]
  | <element pattern predicate> ]

* WC3:924-022 *

<element variable declaration> ::=
```

```
IWD 39075:202y(E)
16.10 <path pattern expression>

[ TEND ] <element variable>
<is label expression> ::=
  <is or colon> <label expression>
  | <colon>
  | <colon>

<element pattern predicate> ::=
  <element pattern where clause>
  | <element property specification>

<element pattern where clause> ::=
  WHERE <search condition>

<element property specification> ::=
  <left brace> <property key value pair list> <right brace>

<property key value pair list> ::=
  <property key value pair> [ { <comma> <property key value pair> }... ]

<property key value pair> ::=
  <property name> <colon> <value expression>

<edge pattern> ::=
  <full edge pattern>
  | <abbreviated edge pattern>

<full edge pattern> ::=
  <full edge pointing left>
  | <full edge undirected>
  | <full edge pointing right>
  | <full edge left or undirected>
  | <full edge undirected or right>
  | <full edge left or right>
  | <full edge any direction>

<full edge pointing left> ::=
  <left arrow bracket> <element pattern filler> <right bracket minus>

<full edge undirected> ::=
  <tilde left bracket> <element pattern filler> <right bracket tilde>

<full edge pointing right> ::=
  <minus left bracket> <element pattern filler> <bracket right arrow>

<full edge left or undirected> ::=
  <left arrow tilde bracket> <element pattern filler> <right bracket tilde>

<full edge undirected or right> ::=
  <tilde left bracket> <element pattern filler> <bracket tilde right arrow>

<full edge left or right> ::=
  <left arrow bracket> <element pattern filler> <bracket right arrow>

<full edge any direction> ::=
  <minus left bracket> <element pattern filler> <right bracket minus>

** Editor's Note (number 73) **
In the BNF for <full edge any direction>, the delimiter tokens <-| -> have been suggested as a synonym for [ ] as part
of Feature GAWT, "Undirected edge pattern". The synonym for the <abbreviated edge pattern> (<colon sign>) would then be
<->, the synonym for <simplified defaulting any direction> would use the delimiter tokens <- / -> and the synonym for
```

```
IWD 39075:202y(E)
16.10 <path pattern expression>

<simplified override any direction> would use the tokens <- and > surrounding a label as originally proposed in WC3:MANU
066. These synonyms might be considered to make the table of edge patterns more harmonious and internally consistent. See
Language Opportunity (CQR-212).

<abbreviated edge pattern> ::=
  <left arrow>
  | <tilde>
  | <right arrow>
  | <left arrow tilde>
  | <tilde right arrow>
  | <left minus right>
  | <minus sign>

* WC3:924-038 deleted one Editor's Note *

<parenthesized path pattern expression> ::=
  <left paren>
  [ <subpath variable declaration> ]
  <path mode prefix> ]
  <path pattern expression>
  <parenthesized path pattern where clause> ]
  <right paren>

<subpath variable declaration> ::=
  <subpath variable> <equals operator>

<parenthesized path pattern where clause> ::=
  WHERE <search condition>

Syntax Rules
1) Let RIGHTMINUS be the following collection of <tokens>: <right bracket minus>, <left arrow>, <slash
minus>, and <minus sign>.
NOTE 132 — These are the tokens ], <-, /, and -, which expose a minus sign on the right.
2) Let LEFTMINUS be the following collection of <tokens>: <minus left bracket>, <right arrow>, <minus
slash>, and <minus sign>.
NOTE 133 — These are the tokens [ -, /, and -, which expose a minus sign on the left. <minus sign> itself is in both
RIGHTMINUS and LEFTMINUS.
3) A <path pattern expression> shall not juxtapose a <tokens> from RIGHTMINUS followed by a <tokens>
from LEFTMINUS without a <separator> between them.
NOTE 134 — Otherwise, the concatenation of the two tokens would include the sequence of two <minus signs>,
which is a <simple comment introducer>.
4) A <path pattern expression> that contains at the same depth of graph pattern matching a variable
quantifier, a <questioned path primary>, a <path multiset alternation>, or a <path pattern union>
is a <possibly variable length path pattern>.
5) A <path pattern expression> that is not a <possibly variable length path pattern> is a <fixed length path
pattern>.
6) The <minimum path length> of certain BNF non-terminals defined in this Subclause is defined
recursively as follows:
a) The <minimum path length> of a <node pattern> is 0 (zero).
b) The <minimum path length> of an <edge pattern> is 1 (one).
```

Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax

```
IWD 39075:202y(E)
16.10 <path pattern expression>

Function
Specify a pattern to match a single path in a property graph.

Format
<path pattern expression> ::=
  <path term>
  | <path multiset alternation>
  | <path pattern union>

<path multiset alternation> ::=
  <path term> <multiset alternation operator> <path term>
  | { <multiset alternation operator> <path term> }... ]

<path pattern union> ::=
  <path term> <vertical bar> <path term> [ { <vertical bar> <path term> }... ]

<path term> ::=
  <path factor>
  | <path concatenation>

<path concatenation> ::=
  <path term> <path factor>

<path factor> ::=
  <path primary>
  | <quantified path primary>
  | <questioned path primary>

<quantified path primary> ::=
  <path primary> <graph pattern quantifier>

<questioned path primary> ::=
  <path primary> <question mark>

NOTE 131 — Unlike most regular expression languages, <question mark> is not equivalent to the quantifier {1}. The
quantifier {1} exposes variables as group, whereas <question mark> does not change the singleton variables that it exposes
to group. However, <question mark> does expose any singleton variables as conditional singlens.

<path primary> ::=
  <element pattern>
  | <parenthesized path pattern expression>
  | <simplified path pattern expression>

<element pattern> ::=
  <node pattern>
  | <edge pattern>

<node pattern> ::=
  <left paren> <element pattern filler> <right paren>

<element pattern filler> ::=
  [ <element variable declaration> ]
  | <is label expression> ]
  | <element pattern predicate> ]

* WC3:924-022 *
```

```
IWD 39075:202y(E)
16.10 <path pattern expression>

[ TEND ] <element variable>

<is label expression> ::=
  <is or colon> <label expression>
  | <colon>

<element pattern predicate> ::=
  <element pattern where clause>
  | <element property specification>

<element pattern where clause> ::=
  WHERE <search condition>

<element property specification> ::=
  <left brace> <property key value pair list> <right brace>

<property key value pair list> ::=
  <property key value pair> [ { <comma> <property key value pair> }... ]

<property key value pair> ::=
  <property name> <colon> <value expression>

<edge pattern> ::=
  <full edge pattern>
  | <abbreviated edge pattern>

<full edge pattern> ::=
  <full edge pointing left>
  | <full edge undirected>
  | <full edge pointing right>
  | <full edge left or undirected>
  | <full edge undirected or right>
  | <full edge left or right>
  | <full edge any direction>

<full edge pointing left> ::=
  <left arrow bracket> <element pattern filler> <right bracket minus>

<full edge undirected> ::=
  <tilde left bracket> <element pattern filler> <right bracket tilde>

<full edge pointing right> ::=
  <minus left bracket> <element pattern filler> <bracket right arrow>

<full edge left or undirected> ::=
  <left arrow tilde bracket> <element pattern filler> <right bracket tilde>

<full edge undirected or right> ::=
  <tilde left bracket> <element pattern filler> <bracket tilde right arrow>

<full edge left or right> ::=
  <left arrow bracket> <element pattern filler> <bracket right arrow>

<full edge any direction> ::=
  <minus left bracket> <element pattern filler> <right bracket minus>

** Editor's Note (number 73) **
In the BNF for <full edge any direction>, the delimiter tokens <[ ]> have been suggested as a synonym for { } as part of
Feature GAWT, "Undirected edge pattern". The synonym for the <abbreviated edge pattern> <colon sign> would then be
<->, the synonym for <simplified defaulting any direction> would use the delimiter tokens <- / -> and the synonym for
```

```
IWD 39075:202y(E)
16.10 <path pattern expression>

<simplified override any direction> would use the tokens <- and > surrounding a label as originally proposed in WC3:MANU
066. These synonyms might be considered to make the table of edge patterns more harmonious and internally consistent. See
Language Opportunity (CQ-212).

<abbreviated edge pattern> ::=
  <left arrow>
  | <tilde>
  | <right arrow>
  | <left arrow tilde>
  | <tilde right arrow>
  | <left minus right>
  | <minus sign>

* WC3:924-018 deleted one Editor's Note *

<parenthesized path pattern expression> ::=
  <left paren>
  [ <subpath variable declaration> ]
  <path mode prefix> ]
  <path pattern expression>
  <parenthesized path pattern where clause> ]
  <right paren>

<subpath variable declaration> ::=
  <subpath variable> <equals operator>

<parenthesized path pattern where clause> ::=
  WHERE <search condition>

Syntax Rules
1) Let RIGHTMINUS be the following collection of <tokens>: <right bracket minus>, <left arrow>, <slash
minus>, and <minus sign>.
NOTE 132 — These are the tokens ], <-, /, and -, which expose a minus sign on the right.
2) Let LEFTMINUS be the following collection of <tokens>: <minus left bracket>, <right arrow>, <minus
slash>, and <minus sign>.
NOTE 133 — These are the tokens [, <-, /, and -, which expose a minus sign on the left. <minus sign> itself is in both
RIGHTMINUS and LEFTMINUS.
3) A <path pattern expression> shall not juxtapose a <tokens> from RIGHTMINUS followed by a <tokens>
from LEFTMINUS without a <separator> between them.
NOTE 134 — Otherwise, the concatenation of the two tokens would include the sequence of two <minus signs>,
which is a <simple comment introducer>.
4) A <path pattern expression> that contains at the same depth of graph pattern matching a variable
quantifier, a <questioned path primary>, a <path multiset alternation>, or a <path pattern union>
is a <possibly variable length path pattern>.
5) A <path pattern expression> that is not a <possibly variable length path pattern> is a <fixed length path
pattern>.
6) The <minimum path length> of certain BNF non-terminals defined in this Subclause is defined
recursively as follows:
a) The <minimum path length> of a <node pattern> is 0 (zero).
b) The <minimum path length> of an <edge pattern> is 1 (one).
```

```
IWD 39075:202y(E)
16.10 <path pattern expression>

c) The <minimum path length> of a <path concatenation> is the sum of the minimum path lengths
of its operands.
d) The <minimum path length> of a <path pattern union> or <path multiset alternation> is the
minimum of the minimum path length of its operands.
e) The <minimum path length> of a <quantified path primary> is the product of the minimum path
length of the simply contained <path primary> and the value of the <lower bound>.
f) The <minimum path length> of a <questioned path primary> is 0 (zero).
g) The <minimum path length> of a <parenthesized path pattern expression> is the minimum path
length of the simply contained <path pattern expression>.
h) If BNT1 and BNT2 are two BNF non-terminals such that BNT1 := BNT2 and the minimum path
length of BNT2 is defined, then the minimum path length of BNT1 is also defined and is the
same as the minimum path length of BNT2.
7) The <path primary> immediately contained in a <quantified path primary> or <questioned path
primary> shall have minimum path length that is greater than 0 (zero).
8) The <path primary> simply contained in a <quantified path primary> shall not contain a <quantified
path primary> at the same depth of graph pattern matching.

** Editor's Note (number 74) **
It may be possible to permit nested quantifiers. WC3:WB1-014 contained a discussion of a way to support aggregates
at different depths of aggregation if there are nested quantifiers. See Language Opportunity (CQ-016).

9) Let PMA be a <path multiset alternation>.
a) A <path term> simply contained in PMA is a <multiset alternation operand of PMA>.
b) Let NOPMA be the number of multiset alternation operands of PMA. Let OPMA1, ..., OPMANOPMA
be an enumeration of the operands of PMA.
c) Any <subpath variables> declared by <subpath variable declarations> simply contained in
the multiset alternation operands of PMA shall be mutually distinct.
d) Let SOPMA1, ..., SOPMANOPMA be implementation-dependent [UV008] <identifier> that are
mutually distinct and distinct from every <element variable>, <subpath variable> and <path
variable> contained in GP.
e) For every i, 1 (one) ≤ i ≤ NOPMA.
Case:
i) If OPMAi is a <parenthesized path pattern expression> that simply contains a <subpath
variable declaration>, then let OPMAi be OPMAi.
ii) Otherwise, let OPMAi be the <parenthesized path pattern expression>
(SOPMAi = OPMAi)
f) PMA is equivalent to:
OPMA1 | ... | OPMANOPMA
10) A <path term> PPUOP simply contained in a <path pattern union> PSD is a <path pattern union
operand of PSD>.
```

Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax

IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

Function

Specify a pattern to match a single path in a property graph.

Format

```
<path pattern expression> ::=
  <path term>
  | <path multiset alternation>
  | <path pattern union>

<path multiset alternation> ::=
  <path term> <multiset alternation operator> <path term>
  [ { <multiset alternation operator> <path term> }... ]

<path pattern union> ::=
  <path term> <vertical bar> <path term> [ { <vertical bar> <path term> }... ]

<path term> ::=
  <path factor>
  | <path concatenation>

<path concatenation> ::=
  <path term> <path factor>

<path factor> ::=
  <path primary>
  | <quantified path primary>
  | <questioned path primary>

<quantified path primary> ::=
  <path primary> <graph pattern quantifier>

<questioned path primary> ::=
  <path primary> <question mark>

NOTE 131 — Unlike most regular expression languages, <question mark> is not equivalent to the quantifier {1}. The quantifier {0,1} exposes variables as group, whereas <question mark> does not change the singleton variables that it exposes to group. However, <question mark> does expose any singleton variables as conditional singleness.

<path primary> ::=
  <element pattern>
  | <parenthesized path pattern expression>
  | <simplified path pattern expression>

<element pattern> ::=
  <node pattern>
  | <edge pattern>

<node pattern> ::=
  <left paren> <element pattern filler> <right paren>

<element pattern filler> ::=
  [ <element variable declaration> ]
  [ <is label expression> ]
  [ <element pattern predicate> ]
  * WG3:924-022 *
```

<element variable declaration> ::=

221

IWD 39075:202y(E)
16.10 <path pattern expression>

```
[ <TDSP> ] <element variable>

<is label expression> ::=
  <is or colon> <label expression>

<is or colon> ::=
  is
  | <colon>

<element pattern predicate> ::=
  <element pattern where clause>
  | <element property specification>

<element pattern where clause> ::=
  WHERE <search condition>

<element property specification> ::=
  <left brace> <property key value pair list> <right brace>

<property key value pair list> ::=
  <property key value pair> [ { <comma> <property key value pair> }... ]

<property key value pair> ::=
  <property name> <colon> <value expression>

<edge pattern> ::=
  <full edge pattern>
  | <abbreviated edge pattern>

<full edge pattern> ::=
  <full edge pointing left>
  | <full edge pointing right>
  | <full edge undirected>
  | <full edge left or undirected>
  | <full edge right or undirected>
  | <full edge left or right>
  | <full edge any direction>

<full edge pointing left> ::=
  <left arrow bracket> <element pattern filler> <right bracket minus>

<full edge undirected> ::=
  <tilde left bracket> <element pattern filler> <right bracket tilde>

<full edge pointing right> ::=
  <minus left bracket> <element pattern filler> <bracket right arrow>

<full edge left or undirected> ::=
  <left arrow tilde bracket> <element pattern filler> <right bracket tilde>

<full edge undirected or right> ::=
  <tilde left bracket> <element pattern filler> <bracket tilde right arrow>

<full edge left or right> ::=
  <left arrow bracket> <element pattern filler> <bracket right arrow>

<full edge any direction> ::=
  <minus left bracket> <element pattern filler> <right bracket minus>
```

**** Editor's Note (number 73) ****

In the BNF for <full edge any direction>, the delimiter tokens <[]> have been suggested as a synonym for {} as part of Feature GAW7, "Undirected edge pattern". The synonym for the <abbreviated edge pattern> <colon> sign would then be <=>, the synonym for <simplified defaulting any direction> would use the delimiter tokens <-/ -> and the synonym for

222

IWD 39075:202y(E)
16.10 <path pattern expression>

```
<simplified override any direction> would use the tokens <=> and > surrounding a label as originally proposed in WG3:MANU 066. These synonyms might be considered to make the table of edge patterns more harmonious and internally consistent. See Language Opportunity (CQ-212).
```

```
<abbreviated edge pattern> ::=
  <left arrow>
  | <tilde>
  | <right arrow>
  | <left arrow tilde>
  | <tilde right arrow>
  | <left minus right>
  | <minus sign>

* WG3:924-018 deleted one Editor's Note *
```

```
<parenthesized path pattern expression> ::=
  <left paren>
  [ <subpath variable declaration> ]
  <path mode prefix>
  <path pattern expression>
  <parenthesized path pattern where clause>
  <right paren>

<subpath variable declaration> ::=
  <subpath variable> <equals operator>

<parenthesized path pattern where clause> ::=
  WHERE <search condition>
```

Syntax Rules

- 1) Let *RIGHTMINUS* be the following collection of <tokens>: <right bracket minus>, <left arrow>, <slash minus>, and <minus sign>.
NOTE 132 — These are the tokens], </, /, and -, which expose a minus sign on the right.
- 2) Let *LEFTMINUS* be the following collection of <tokens>: <minus left bracket>, <right arrow>, <minus slash>, and <minus sign>.
NOTE 133 — These are the tokens [, </, /, and -, which expose a minus sign on the left. <minus sign> itself is in both *RIGHTMINUS* and *LEFTMINUS*.
- 3) A <path pattern expression> shall not juxtapose a <tokens> from *RIGHTMINUS* followed by a <tokens> from *LEFTMINUS* without a <separator> between them.
NOTE 134 — Otherwise, the concatenation of the two tokens would include the sequence of two <minus signs>, which is a <simple comment introducer>.
- 4) A <path pattern expression> that contains at the same depth of graph pattern matching a variable quantifier, a <questioned path primary>, a <path multiset alternation>, or a <path pattern union> is a <possibly variable length path pattern>.
- 5) A <path pattern expression> that is not a <possibly variable length path pattern> is a <fixed length path pattern>.
- 6) The <minimum path length> of certain BNF non-terminals defined in this Subclause is defined recursively as follows:
 - a) The <minimum path length> of a <node pattern> is 0 (zero).
 - b) The <minimum path length> of an <edge pattern> is 1 (one).

223

IWD 39075:202y(E)
16.10 <path pattern expression>

- a) The <minimum path length> of a <path concatenation> is the sum of the <minimum path lengths> of its operands.
- d) The <minimum path length> of a <path pattern union> or <path multiset alternation> is the <minimum of the minimum path length> of its operands.
- e) The <minimum path length> of a <quantified path primary> is the product of the <minimum path length> of the simply contained <path primary> and the value of the <lower bound>.
- f) The <minimum path length> of a <questioned path primary> is 0 (zero).
- g) The <minimum path length> of a <parenthesized path pattern expression> is the <minimum path length> of the simply contained <path pattern expression>.
- h) If *BNT1* and *BNT2* are two BNF non-terminals such that *BNT1* ::= *BNT2* and the <minimum path length> of *BNT2* is defined, then the <minimum path length> of *BNT1* is also defined and is the same as the <minimum path length> of *BNT2*.

7) The <path primary> immediately contained in a <quantified path primary> or <questioned path primary> shall have <minimum path length> that is greater than 0 (zero).

8) The <path primary> simply contained in a <quantified path primary> shall not contain a <quantified path primary> at the same depth of graph pattern matching.

**** Editor's Note (number 74) ****

It may be possible to permit nested quantifiers. WG3:SM-052 contained a discussion of a way to support aggregation at different depths of aggregation if there are nested quantifiers. See Language Opportunity (CQ-016).

- 9) Let *PMA* be a <path multiset alternation>.
 - a) A <path term> simply contained in *PMA* is a <multiset alternation operand of PMA>.
 - b) Let *NOPMA* be the number of multiset alternation operands of *PMA*. Let *OPMA₁*, ..., *OPMA_{NOPMA}* be an enumeration of the operands of *PMA*.
 - c) Any <subpath variables> declared by <subpath variable declarations> simply contained in the multiset alternation operands of *PMA* shall be mutually distinct.
 - d) Let *SOPMA₁*, ..., *SOPMA_{NOPMA}* be implementation-dependent [(UV008)] <identifiers> that are mutually distinct and distinct from every <element variable>, <subpath variable> and <path variable> contained in *GP*.
 - e) For every *i*, 1 (one) ≤ *i* ≤ *NOPMA*.
Case:
 - i) If *OPMA_i* is a <parenthesized path pattern expression> that simply contains a <subpath variable declaration>, then let *OPMA_i* be *OPMA_i*.
 - ii) Otherwise, let *OPMA_i* be the <parenthesized path pattern expression>
(*SOPMA_i* = *OPMA_i*)
 - f) *PMA* is equivalent to:
OPMA₁ | ... | *OPMA_{NOPMA}*
- 10) A <path term> *PPUOP* simply contained in a <path pattern union> *PSD* is a <path pattern union operand of PSD>.

224

IWD 39075:202y(E)
16.10 <path pattern expression>

**** Editor's Note (number 75) ****

Path pattern union is not defined using left recursion. WG3:SM-052 believed that it should be possible to support left recursion but declined to do so for expediency. It is a Language Opportunity to support left recursion. See Language Opportunity (CQ-025).

PPUOP shall not contain a reference to an element variable that is not declared in *PPUOP* or outside of *PSD*.

- 11) An <element pattern> *EP* that contains an <element pattern where clause> *EPWC* is transformed as follows:
 - a) Let *EPF* be the <element pattern filler> simply contained in *EP*.
 - b) Let *PREFIX* be the <delimiter tokens> contained in *EP* before *EPF* and let *SUFFIX* be the <delimiter tokens> contained in *EP* after *EPF*.
 - c) Let *EV* be the <element variable> simply contained in *EPF*. Let *LLE* be the <is label expression> contained in *EPF*, if any; otherwise, let *LLE* be the zero-length string.
 - d) *EP* is replaced by
(*PREFIX EV LLE SUFFIX EPWC*)
- 12) An <element pattern> that does not contain an <element variable declaration>, an <is label expression>, or an <element pattern predicate> is said to be <empty>.
- 13) Each <path pattern expression> is transformed in the following steps:
 - a) If the <path primary> immediately contained in a <quantified path primary> or <questioned path primary> is an <edge pattern> *EP*, then *EP* is replaced by
(*EP*)
NOTE 135 — For example,
...
becomes:
(-> {>,
which in later transformations becomes:
(1) -> (1) {>.
 - b) If two successive <element patterns> are contained in a <path concatenation> at the same depth of graph pattern matching and are <edge patterns>, then an implicit empty <node pattern> is inserted between them.
 - c) If an edge pattern *EP* contained in a <path term> *PST* at the same depth of graph pattern matching is not preceded by a <node pattern> contained in *PST* at the same depth of graph pattern matching, then an implicit empty <node pattern> *VP* is inserted in *PST* immediately prior to *EP*.
 - d) If an edge pattern *EP* contained in a <path term> *PST* at the same depth of graph pattern matching is not followed by a <node pattern> contained in *PST* at the same depth of graph pattern matching, then an implicit empty <node pattern> *VP* is inserted in *PST* immediately after *EP*.
NOTE 136 — As a result of the preceding transformations, a fixed length path pattern has an odd number of <element patterns>, beginning and ending with <node patterns>, and alternating between <node patterns> and <edge patterns>.

225

Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax

IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

Function

Specify a pattern to match a single path in a property graph.

Format

```
<path pattern expression> ::=
<path term>
| <path multiset alternation>
| <path pattern union>

<path multiset alternation> ::=
<path term> «multiset alternation operator» <path term>
| { «multiset alternation operator» <path term> }... ]

<path pattern union> ::=
<path term> «vertical bar» <path term> [ { «vertical bar» <path term> }... ]

<path term> ::=
<path factor>
| <path concatenation>

<path concatenation> ::=
<path term> «path factor»

<path factor> ::=
<path primary>
| «quantified path primary»
| «questioned path primary»

<quantified path primary> ::=
<path primary> «graph pattern quantifier»

<questioned path primary> ::=
<path primary> «question mark»

NOTE 131 — Unlike most regular expression languages, «question mark» is not equivalent to the quantifier {0,1}; the quantifier {0,1} exposes variables as group, whereas «question mark» does not change the singleton variables that it exposes to group. However, «question mark» does expose any singleton variables as conditional singletons.
```

221

IWD 39075:202y(E)
16.10 <path pattern expression>

b) If *BNF1* and *BNF2* are two BNF non-terminals such that *BNF1* ::= *BNF2* and the minimum node count of *BNF2* is defined, then the minimum node count of *BNF1* is also defined and is the same as the minimum node count of *BNF2*.

15) The «path pattern expression» simply contained in a «path pattern» shall have a minimum node count that is greater than 0 (zero).

NOTE 137 — The minimum node count is computed after the syntactic transform that adds implicit node patterns. Thus a single «edge pattern» is a permitted «path pattern» because it implies two «node patterns».

«WG3:W24-022»

16) An «element variable» *EV* contained in an «element variable declaration» *GPVD* is said to be declared by *GPVD*, and by the «element pattern» *EP* that simply contains *GPVD*. The «element variable» is the name of an element variable, which is also declared by *GPVD* and *EP*. If *GPVD* simply contains *TEMP*, then *EV* is a temporary element variable. *EV* is a primary variable.

NOTE 138 — Element bindings to temporary element variables are removed prior to set-theoretic dot-product matches. See 10) of subclause 16.8, «graph pattern» and 10) of subclause 21.2, «machinery for graph pattern matching».

17) Prior to the application of syntactic transformations, conforming GQL language shall not contain an «element variable declaration» that immediately contains *TEMP*.

18) An element variable that is declared by a «node pattern» is a node variable. An element variable that is declared by an «edge pattern» is an edge variable.

«WG3:W24-022»

19) The scope of an «element variable» *EV* that is declared by an «element pattern» *EP* is defined as follows. If *EV* is a temporary element variable, then the scope of *EV* is the innermost «path term» containing *EP*; otherwise, the scope of *EV* is the innermost «graph pattern binding table» containing *EP*.

20) A «subpath variable» *SV* contained in a «subpath variable declaration» *SVVD* is said to be declared by *SVVD*, and by the «parenthesized path pattern expression» *PPPE* that simply contains *SVVD*. *SV* is the name of a subpath variable, which is also declared by *SVVD* and *PPPE*.

21) If *EP* is an «element pattern» that contains an «element pattern where clause» *EPWC*, then *EP* shall simply contain an «element variable declaration» *GPVD*.

22) If *EV* is an element variable or subpath variable, and *BNF1* is an instance of a BNF non-terminal, then the terminology «*BNF1* exposes *EV*» is defined as follows. The full terminology is one of the following: «*BNF1* exposes *EV* as an unconditional singleton variable», «*BNF1* exposes *EV* as a conditional singleton variable», «*BNF1* exposes *EV* as an effectively bounded group variable» or «*BNF1* exposes *EV* as an effectively unbounded group variable». The terms «unconditional singleton variable», «conditional singleton variable», «effectively bounded group variable», and «effectively unbounded group variable» are called the *degree of exposure*.

a) An «element pattern» *EP* that declares an element variable *EV* exposes *EV* as an unconditional singleton.

b) A «parenthesized path pattern expression» *PPPE* that simply contains a «subpath variable declaration» that declares *EV* exposes *EV* as an unconditional singleton variable. *PPPE* shall not contain another «parenthesized path pattern expression» that declares *EV*.

c) If a «path concatenation» *PPC* declares *EV* then let *PT* be the «path term» and let *PF* be the «path factor» simply contained in *PPC*.

Case:

i) If *EV* is exposed as an unconditional singleton by both *PT* and *PF*, then *EV* is exposed as an unconditional singleton by *PPC*. *EV* shall not be a subpath variable.

227

IWD 39075:202y(E)
16.10 <path pattern expression>

```
[ TEMP ] «element variable»

«is label expression» ::=
«is or colon» «label expression»

«is or colon» ::=
is
| «colon»

«element pattern predicate» ::=
«element pattern where clause»
| «element property specification»

«element pattern where clause» ::=
WHERE «search condition»

«element property specification» ::=
«left brace» «property key value pair list» «right brace»

«property key value pair list» ::=
«property key value pair» [ { «comma» «property key value pair» }... ]

«property key value pair» ::=
«property name» «colon» «value expression»

«edge pattern» ::=
«full edge pattern»
| «abbreviated edge pattern»

«full edge pattern» ::=
«full edge pointing left»
| «full edge pointing right»
| «full edge undirected»
| «full edge left or undirected»
| «full edge right or undirected»
| «full edge left or right»
| «full edge any direction»

«full edge pointing left» ::=
«left arrow bracket» «element pattern filler» «right bracket minus»

«full edge undirected» ::=
«tilde left bracket» «element pattern filler» «right bracket tilde»

«full edge pointing right» ::=
«minus left bracket» «element pattern filler» «bracket right arrow»

«full edge left or undirected» ::=
«left arrow tilde bracket» «element pattern filler» «right bracket tilde»

«full edge left or undirected» «element pattern filler» «bracket tilde right arrow»

«full edge left or right» ::=
«left arrow bracket» «element pattern filler» «bracket right arrow»

«full edge any direction» ::=
«minus left bracket» «element pattern filler» «right bracket minus»

** Editor's Note (number 73) **

In the BNF for «full edge any direction», the delimiter tokens <-|>- have been suggested as a synonym for [ ] as part of Feature GAWT, "Undirected edge pattern". The synonym for the «abbreviated edge pattern» («minus sign») would then be <->, the synonym for «simplified defaulting any direction» would use the delimiter tokens <- / -> and the synonym for
```

222

IWD 39075:202y(E)
16.10 <path pattern expression>

```
«simplified override any direction» would use the tokens <- and > surrounding a label as originally proposed in WG3:MANU 066. These synonyms might be considered to make the table of edge patterns more harmonious and internally consistent. See Language Opportunity (CQ-212).
```

```
«abbreviated edge pattern» ::=
«left arrow»
| «tilde»
| «right arrow»
| «left arrow tilde»
| «tilde right arrow»
| «left minus right»
| «minus sign»

«parenthesized path pattern expression» ::=
«left paren»
[ «subpath variable declaration» ]
[ «path mode prefix» ]
«path pattern expression»
«parenthesized path pattern where clause» ]
«right paren»

«subpath variable declaration» ::=
«subpath variable» «equals operator»

«parenthesized path pattern where clause» ::=
WHERE «search condition»
```

Syntax Rules

1) Let *RIGHTMINUS* be the following collection of «tokens»: «right bracket minus», «left arrow», «slash minus», and «minus sign».

NOTE 132 — These are the tokens |, <, /, and -, which expose a minus sign on the right.

2) Let *LEFTMINUS* be the following collection of «tokens»: «minus left bracket», «right arrow», «minus slash», and «minus sign».

NOTE 133 — These are the tokens |, >, /, and -, which expose a minus sign on the left. «minus sign» itself is in both *RIGHTMINUS* and *LEFTMINUS*.

3) A «path pattern expression» shall not juxtapose a «tokens» from *RIGHTMINUS* followed by a «tokens» from *LEFTMINUS* without a «separator» between them.

NOTE 134 — Otherwise, the concatenation of the two tokens would include the sequence of two «minus signs», which is a simple comment introducer.

4) A «path pattern expression» that contains at the same depth of graph pattern matching a variable quantifier, a «questioned path primary», a «path multiset alternation», or a «path pattern union» is a possibly variable length path pattern.

5) A «path pattern expression» that is not a possibly variable length path pattern is a fixed length path pattern.

6) The minimum path length of certain BNF non-terminals is defined in this Subclause as follows:

a) The minimum path length of a «node pattern» is 0 (zero).

b) The minimum path length of an «edge pattern» is 1 (one).

223

IWD 39075:202y(E)
16.10 <path pattern expression>

c) The minimum path length of a «path concatenation» is the sum of the minimum path lengths of its operands.

d) The minimum path length of a «path pattern union» or «path multiset alternation» is the minimum of the minimum path length of its operands.

e) The minimum path length of a «quantified path primary» is the product of the minimum path length of the simply contained «path primary» and the value of the «lower bound».

f) The minimum path length of a «questioned path primary» is 0 (zero).

g) The minimum path length of a «parenthesized path pattern expression» is the minimum path length of the simply contained «path alternation expression».

h) If *BNF1* and *BNF2* are two BNF non-terminals such that *BNF1* ::= *BNF2* and the minimum path length of *BNF2* is defined, then the minimum path length of *BNF1* is also defined and is the same as the minimum path length of *BNF2*.

7) The «path primary» immediately contained in a «quantified path primary» or «questioned path primary» shall have minimum path length that is greater than 0 (zero).

8) The «path primary» simply contained in a «quantified path primary» shall not contain a «quantified path primary» at the same depth of graph pattern matching.

** Editor's Note (number 74) **

It may be possible to permit nested quantifiers. WG3:W1-014 contained a discussion of a way to support aggregation at different depths of aggregation if there are nested quantifiers. See Language Opportunity (CQ-016).

9) Let *PMA* be a «path multiset alternation».

a) A «path term» simply contained in *PMA* is a multiset alternation operand of *PMA*.

b) Let *NOPMA* be the number of multiset alternation operands of *PMA*. Let *OPMA₁*, ..., *OPMA_{NOPMA}* be an enumeration of the operands of *PMA*.

c) Any «subpath variables» declared by «subpath variable declarations» simply contained in the multiset alternation operands of *PMA* shall be mutually distinct.

d) Let *SOPMA₁*, ..., *SOPMA_{NOPMA}* be implementation-dependent [(UV008) «identifier»s that are mutually distinct and distinct from every «element variable», «subpath variable» and «path variable» contained in *GP*.

e) For every *i*, 1 (one) ≤ *i* ≤ *NOPMA*.

Case:

i) If *OPMA_i* is a «parenthesized path pattern expression» that simply contains a «subpath variable declaration», then let *OPMA_i* be *OPMA_i*.

ii) Otherwise, let *OPMAX_i* be the «parenthesized path pattern expression»

(*SOPMA₁* | ... | *OPMA_{NOPMA}*)

f) *PMA* is equivalent to:

OPMA₁ | ... | *OPMA_{NOPMA}*

10) A «path term» *PPUOP* simply contained in a «path pattern union» *PSD* is a path pattern union operand of *PSD*.

224

IWD 39075:202y(E)
16.10 <path pattern expression>

** Editor's Note (number 75) **

Path pattern union is not defined using left recursion. WG3:SM-052 believed that it should be possible to support left recursion but declined to do so for expediency. It is a Language Opportunity to support left recursion. See Language Opportunity (CQ-025).

PPUOP shall not contain a reference to an element variable that is not declared in *PPUOP* or outside of *PSD*.

11) An «element pattern» *EP* that contains an «element pattern where clause» *EPWC* is transformed as follows:

a) Let *EPF* be the «element pattern filler» simply contained in *EP*.

b) Let *PREFIX* be the «delimiter tokens» contained in *EP* before *EPF* and let *SUFFIX* be the «delimiter tokens» contained in *EP* after *EPF*.

c) Let *EV* be the «element variable» simply contained in *EPF*. Let *LLE* be the «is label expression» contained in *EPF*, if any; otherwise, let *LLE* be the zero-length string.

d) *EP* is replaced by

(*PREFIX EV LLE SUFFIX EPWC*)

12) An «element pattern» that does not contain an «element variable declaration», an «is label expression», or an «element pattern predicate» is said to be empty.

13) Each «path pattern expression» is transformed in the following steps:

a) If the «path primary» immediately contained in a «quantified path primary» or «questioned path primary» is an «edge pattern» *EP*, then *EP* is replaced by

(*EP*)

NOTE 135 — For example,

...
becomes:
(->) (>)
which in later transformations becomes:
(| -> |) (>)

b) If two successive «element patterns» contained in a «path concatenation» at the same depth of graph pattern matching are edge patterns, then an implicit empty «node pattern» is inserted between them.

c) If an edge pattern *EP* contained in a «path term» *PST* at the same depth of graph pattern matching is not preceded by a «node pattern» contained in *PST* at the same depth of graph pattern matching, then an implicit empty «node pattern» *VP* is inserted in *PST* immediately prior to *EP*.

d) If an edge pattern *EP* contained in a «path term» *PST* at the same depth of graph pattern matching is not followed by a «node pattern» contained in *PST* at the same depth of graph pattern matching, then an implicit empty «node pattern» *VP* is inserted in *PST* immediately after *EP*.

NOTE 136 — As a result of the preceding transformations, a fixed length path pattern has an odd number of «element patterns», beginning and ending with «node patterns», and alternating between «node patterns» and «edge patterns».

225

Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax

IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

Function

Specify a pattern to match a single path in a property graph.

Format

```
<path pattern expression> ::=
<path term>
| <path multiset alternation>
| <path pattern union>

<path multiset alternation> ::=
<path term> ~multiset alternation operator~ <path term>
| { ~multiset alternation operator~ <path term> } ... }

<path pattern union> ::=
<path term> ~vertical bar~ <path term> [ { ~vertical bar~ <path term> } ... ]

<path term> ::=
<path factor>
| <path concatenation>

<path concatenation> ::=
<path term> ~path factor~

<path factor> ::=
<path primary>
| <quantified path primary>
| <questioned path primary>

<quantified path primary> ::=
<path primary> ~graph pattern quantifier~

<questioned path primary> ::=
<path primary> ~question mark~

NOTE 131 — Unlike most regular expression languages, <question mark> is not equivalent to the quantifier {0,1}; the quantifier {0,1} exposes variables as group, whereas <question mark> does not change the singleton variables that it exposes to group. However, <question mark> does expose any singleton variables as conditional singletons.
```

221

IWD 39075:202y(E)
16.10 <path pattern expression>

```
[ TEMP ] <element variable>

<iis label expression> ::=
<iis or colon> ~label expression~
| <colon>

<element pattern predicate> ::=
<element pattern where clause>
| <element property specification>

<element pattern where clause> ::=
WHERE ~search condition~

<element property specification> ::=
<left brace> ~property key value pair list~ <right brace>

<property key value pair list> ::=
<property key value pair> [ { ~comma~ <property key value pair> } ... ]

<property key value pair> ::=
<property name> ~colon~ <value expression>

<edge pattern> ::=
<full edge pattern>
| <abbreviated edge pattern>

<full edge pattern> ::=
<full edge pointing left>
| <full edge pointing right>
| <full edge left or undirected>
| <full edge right or undirected>
| <full edge left or right>
| <full edge any direction>

<full edge pointing left> ::=
<left arrow bracket> ~element pattern filler~ <right bracket minus>

<full edge undirected> ::=
<tilde left bracket> ~element pattern filler~ <right bracket tilde>

<full edge pointing right> ::=
<tilde left bracket> ~element pattern filler~ <right bracket>

<full edge left or undirected> ::=
<left arrow tilde bracket> ~element pattern filler~ <right bracket tilde>

<full edge left or right> ::=
<left arrow bracket> ~element pattern filler~ <right arrow>

<full edge any direction> ::=
<minus left bracket> ~element pattern filler~ <right bracket minus>
```

222

IWD 39075:202y(E)
16.10 <path pattern expression>

Syntax Rules

- Let *RIGHMINUS* be the following collection of <tokens>: <right bracket minus>, <left arrow>, <slash minus>, and <minus sign>.
- Let *LEFTMINUS* be the following collection of <tokens>: <minus left bracket>, <right arrow>, <minus slash>, and <minus sign>.
- A <path pattern expression> that contains at the same depth of graph pattern matching a variable quantifier, a <questioned path primary>, a <path multiset alternation>, or a <path pattern union> is a <possibly variable length path pattern>.
- A <path pattern expression> that is not a <possibly variable length path pattern> is a <fixed length path pattern>.
- The <minimum path length of certain BNF non-terminals defined in this Subclause is defined recursively as follows:
 - The <minimum path length of a <node pattern> is 0 (zero).
 - The <minimum path length of an <edge pattern> is 1 (one).

223

IWD 39075:202y(E)
16.10 <path pattern expression>

- The <minimum path length of a <path concatenation> is the sum of the minimum path lengths of its operands.
- The <minimum path length of a <path pattern union> or <path multiset alternation> is the minimum of the minimum path length of its operands.
- The <minimum path length of a <quantified path primary> is the product of the minimum path length of the simply contained <path primary> and the value of the <lower bound>.
- The <minimum path length of a <questioned path primary> is 0 (zero).
- The <minimum path length of a <parenthesized path pattern expression> is the minimum path length of the simply contained <path pattern expression>.
- If *BNT1* and *BNT2* are two BNF non-terminals such that *BNT1* ::= *BNT2* and the minimum path length of *BNT2* is defined, then the minimum path length of *BNT1* is also defined and is the same as the minimum path length of *BNT2*.
- The <path primary> immediately contained in a <quantified path primary> or <questioned path primary> shall have minimum path length that is greater than 0 (zero).
- The <path primary> simply contained in a <quantified path primary> shall not contain a <quantified path primary> at the same depth of graph pattern matching.

224

IWD 39075:202y(E)
16.10 <path pattern expression>

**** Editor's Note (number 75) ****

Path pattern union is not defined using left recursion. WG3.SXM-052 believed that it should be possible to support left recursion but declined to do so for expediency. It is a Language Opportunity to support left recursion. See Language Opportunity [LO8-025](#).

PPUOP shall not contain a reference to an element variable that is not declared in *PPUOP* or outside of *PSD*.

- An <element pattern> *EP* that contains an <element pattern where clause> *EPWC* is transformed as follows:
 - Let *EPF* be the <element pattern filler> simply contained in *EP*.
 - Let *PREFIX* be the <delimiter tokens> contained in *EP* before *EPF* and let *SUFFIX* be the <delimiter tokens> contained in *EP* after *EPF*.
 - Let *EV* be the <element variable> simply contained in *EPF*. Let *LLE* be the <is label expression> contained in *EPF*, if any; otherwise, let *LLE* be the zero-length string.
 - EP* is replaced by
(*PREFIX* *EV* *LLE* *SUFFIX* *EPWC*)
- An <element pattern> that does not contain an <element variable declaration>, an <is label expression>, or an <element pattern predicate> is said to be *empty*.
- Each <path pattern expression> is transformed in the following steps:
 - If the <path primary> immediately contained in a <quantified path primary> or <questioned path primary> is an <edge pattern> *EP*, then *EP* is replaced by
(*EP*)
NOTE 135 — For example,
...
becomes:
(- -) (-)
which in later transformations becomes:
(() - - ()) (-)
 - If two successive <element patterns> contained in a <path concatenation> at the same depth of graph pattern matching are <edge patterns>, then an implicit empty <node pattern> is inserted between them.
 - If an edge pattern *EP* contained in a <path term> *PST* at the same depth of graph pattern matching is not preceded by a <node pattern> contained in *PST* at the same depth of graph pattern matching, then an implicit empty <node pattern> *VP* is inserted in *PST* immediately prior to *EP*.
 - If an edge pattern *EP* contained in a <path term> *PST* at the same depth of graph pattern matching is not followed by a <node pattern> contained in *PST* at the same depth of graph pattern matching, then an implicit empty <node pattern> *VP* is inserted in *PST* immediately after *EP*.
NOTE 136 — As a result of the preceding transformations, a fixed length path pattern has an odd number of <element patterns>, beginning and ending with <node patterns>, and alternating between <node patterns> and <edge patterns>.

225

IWD 39075:202y(E)
16.10 <path pattern expression>

- If *BNF1* and *BNF2* are two BNF non-terminals such that *BNF1* ::= *BNF2* and the minimum node count of *BNF2* is defined, then the minimum node count of *BNF1* is also defined and is the same as the minimum node count of *BNF2*.
- The <path pattern expression> simply contained in a <path pattern> shall have a minimum node count that is greater than 0 (zero).
NOTE 137 — The minimum node count is computed after the syntactic transform that adds implicit node patterns. Thus a single <edge pattern> is a permitted <path pattern> because it implies two <node patterns>.
- An <element variable> *EV* contained in an <element variable declaration> *GPVD* is said to be declared by *GPVD*, and by the <element pattern> *EP* that simply contains *GPVD*. The <element variable> is the name of an element variable, which is also declared by *GPVD* and *EP*. If *GPVD* simply contains *TEMP*, then *EV* is a temporary element variable. *EV* is a *primary* variable.
NOTE 138 — Element bindings to temporary element variables are removed prior to set-theoretic deduplication of matches. See GR 10) of Subclause 16.8, “graph patterns” and GR 14) of Subclause 21.2, “Machinery for graph pattern matching”.
- Prior to the application of syntactic transformations, conforming GQL-language shall not contain an <element variable declaration> that immediately contains *TEMP*.
- An element variable that is declared by a <node pattern> is a *node variable*. An element variable that is declared by an <edge pattern> is an *edge variable*.
- The scope of an <element variable> *EV* that is declared by an <element pattern> *EP* is defined as follows. If *EV* is a temporary element variable, then the scope of *EV* is the innermost <path term> containing *EP*; otherwise, the scope of *EV* is the innermost <graph pattern binding table> containing *EP*.
- A <subpath variable> *SV* contained in a <subpath variable declaration> *SYD* is said to be declared by *SYD*, and by the <parenthesized path pattern expression> *PPPE* that simply contains *SYD*. *SV* is the name of a subpath variable, which is also declared by *SYD* and *PPPE*.
- If *EP* is an <element pattern> that contains an <element pattern where clause> *EPWC*, then *EP* shall simply contain an <element variable declaration> *GPVD*.
- If *EV* is an element variable or subpath variable, and *BNT* is an instance of a BNF non-terminal, then the terminology “*BNT* exposes *EV*” is defined as follows. The full terminology is one of the following: “*BNT* exposes *EV* as an unconditional singleton variable”, “*BNT* exposes *EV* as a conditional singleton variable”, “*BNT* exposes *EV* as an effectively bounded group variable” or “*BNT* exposes *EV* as an effectively unbounded group variable”. The terms “unconditional singleton variable”, “conditional singleton variable”, “effectively bounded group variable”, and “effectively unbounded group variable” are called the *degree of exposure*.
 - An <element pattern> *EP* that declares an element variable *EV* exposes *EV* as an unconditional singleton.
 - A <parenthesized path pattern expression> *PPPE* that simply contains a <subpath variable declaration> that declares *EV* exposes *EV* as an unconditional singleton variable. *PPPE* shall not contain another <parenthesized path pattern expression> that declares *EV*.
 - If a <path concatenation> *PPC* declares *EV* then let *PT* be the <path term> and let *PF* be the <path factor> simply contained in *PPC*.
Case:
 - If *EV* is exposed as an unconditional singleton by both *PT* and *PF*, then *EV* is exposed as an unconditional singleton by *PPC*. *EV* shall not be a subpath variable.

227

IWD 39075:202y(E)
16.10 <path pattern expression>

- If *BNF1* and *BNF2* are two BNF non-terminals such that *BNF1* ::= *BNF2* and the minimum node count of *BNF2* is defined, then the minimum node count of *BNF1* is also defined and is the same as the minimum node count of *BNF2*.
- The <path pattern expression> simply contained in a <path pattern> shall have a minimum node count that is greater than 0 (zero).
NOTE 137 — The minimum node count is computed after the syntactic transform that adds implicit node patterns. Thus a single <edge pattern> is a permitted <path pattern> because it implies two <node patterns>.
- An <element variable> *EV* contained in an <element variable declaration> *GPVD* is said to be declared by *GPVD*, and by the <element pattern> *EP* that simply contains *GPVD*. The <element variable> is the name of an element variable, which is also declared by *GPVD* and *EP*. If *GPVD* simply contains *TEMP*, then *EV* is a temporary element variable. *EV* is a *primary* variable.
NOTE 138 — Element bindings to temporary element variables are removed prior to set-theoretic deduplication of matches. See GR 10) of Subclause 16.8, “graph patterns” and GR 14) of Subclause 21.2, “Machinery for graph pattern matching”.
- Prior to the application of syntactic transformations, conforming GQL-language shall not contain an <element variable declaration> that immediately contains *TEMP*.
- An element variable that is declared by a <node pattern> is a *node variable*. An element variable that is declared by an <edge pattern> is an *edge variable*.
- The scope of an <element variable> *EV* that is declared by an <element pattern> *EP* is defined as follows. If *EV* is a temporary element variable, then the scope of *EV* is the innermost <path term> containing *EP*; otherwise, the scope of *EV* is the innermost <graph pattern binding table> containing *EP*.
- A <subpath variable> *SV* contained in a <subpath variable declaration> *SYD* is said to be declared by *SYD*, and by the <parenthesized path pattern expression> *PPPE* that simply contains *SYD*. *SV* is the name of a subpath variable, which is also declared by *SYD* and *PPPE*.
- If *EP* is an <element pattern> that contains an <element pattern where clause> *EPWC*, then *EP* shall simply contain an <element variable declaration> *GPVD*.
- If *EV* is an element variable or subpath variable, and *BNT* is an instance of a BNF non-terminal, then the terminology “*BNT* exposes *EV*” is defined as follows. The full terminology is one of the following: “*BNT* exposes *EV* as an unconditional singleton variable”, “*BNT* exposes *EV* as a conditional singleton variable”, “*BNT* exposes *EV* as an effectively bounded group variable” or “*BNT* exposes *EV* as an effectively unbounded group variable”. The terms “unconditional singleton variable”, “conditional singleton variable”, “effectively bounded group variable”, and “effectively unbounded group variable” are called the *degree of exposure*.
 - An <element pattern> *EP* that declares an element variable *EV* exposes *EV* as an unconditional singleton.
 - A <parenthesized path pattern expression> *PPPE* that simply contains a <subpath variable declaration> that declares *EV* exposes *EV* as an unconditional singleton variable. *PPPE* shall not contain another <parenthesized path pattern expression> that declares *EV*.
 - If a <path concatenation> *PPC* declares *EV* then let *PT* be the <path term> and let *PF* be the <path factor> simply contained in *PPC*.
Case:
 - If *EV* is exposed as an unconditional singleton by both *PT* and *PF*, then *EV* is exposed as an unconditional singleton by *PPC*. *EV* shall not be a subpath variable.

227

Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax

16.10 <path pattern expression>
Function
Specify a pattern to match a single path in a property graph.
Format
<pre><path pattern expression> ::= <path term> <path multiset alternation> <path pattern union> <path multiset alternation> ::= <path term> «multiset alternation operator» <path term> { «multiset alternation operator» <path term> }... } <path pattern union> ::= <path term> «vertical bar» <path term> { «vertical bar» <path term> }... <path term> ::= <path factor> <path concatenation> <path concatenation> ::= <path factor> «path concatenation operator» <path factor> <path factor> ::= <path primary> «quantified path primary» «questioned path primary» <quantified path primary> ::= <path primary> «graph pattern quantifier» <questioned path primary> ::= <path primary> «question mark» NOTE 131 — Unlike most regular expression languages, «question mark» is not equivalent to the quantifier {0,1}; the quantifier {0,1} exposes variables as group, whereas «question mark» does not change the singleton variables that it exposes to group. However, «question mark» does expose any singleton variables as conditional singletons.</pre>

221

16.10 <path pattern expression>
<pre>[TDMP] «element variable» «is label expression» ::= «is or colon» «label expression» «colon» «element pattern predicate» ::= «element pattern where clause» «element property specification» «element pattern where clause» ::= WHERE «search condition» «element property specification» ::= «left brace» «property key value pair list» «right brace» «property key value pair list» ::= «property key value pair» { «comma» «property key value pair» }... } «property key value pair» ::= «property name» «colon» «value expression» «edge pattern» ::= «full edge pattern» «abbreviated edge pattern» «full edge pattern» ::= «full edge pointing left» «full edge pointing right» «full edge undirected» «full edge pointing left or right» «full edge left or undirected» «full edge undirected or right» «full edge left or right» «full edge any direction» «full edge pointing left» ::= «left arrow bracket» «element pattern filler» «right bracket minus» «full edge undirected» ::= «tilde left bracket» «element pattern filler» «right bracket tilde» «full edge pointing right» ::= «tilde left bracket» «element pattern filler» «right bracket right arrow» «full edge left or undirected» ::= «left arrow tilde bracket» «element pattern filler» «right bracket tilde» «full edge left or right» ::= «left arrow tilde bracket» «element pattern filler» «right bracket tilde right arrow» «full edge any direction» ::= «left arrow bracket» «element pattern filler» «right bracket minus» «tilde left bracket» «element pattern filler» «right bracket tilde» «right arrow bracket» «element pattern filler» «right bracket right arrow» «full edge any direction» ::= «left arrow bracket» «element pattern filler» «right bracket minus» «tilde left bracket» «element pattern filler» «right bracket tilde» «right arrow bracket» «element pattern filler» «right bracket right arrow» ** Editor's Note (number 73) ** In the BNF for «full edge any direction», the delimiter tokens «- » have been suggested as a synonym for « » as part of Feature GAWT, "Undirected edge pattern". The synonym for the «abbreviated edge pattern» «-minus sign» would then be «-», the synonym for «simplified defaulting any direction» would use the delimiter tokens «- » and the synonym for</pre>

222

16.10 <path pattern expression>
<p>h) If <i>BNF1</i> and <i>BNF2</i> are two BNF non-terminals such that <i>BNF1</i> ::= <i>BNF2</i> and the minimum node count of <i>BNF2</i> is defined, then the minimum node count of <i>BNF1</i> is also defined and is the same as the minimum node count of <i>BNF2</i>.</p> <p>15) The «path pattern expression» simply contained in a «path pattern» shall have a minimum node count that is greater than 0 (zero).</p> <p>NOTE 137 — The minimum node count is computed after the syntactic transform that adds implicit node patterns. Thus a single «edge pattern» is a permitted «path pattern» because it implies two «node patterns».</p> <p>«WG3:W24-022»</p> <p>16) An «element variable» <i>EV</i> contained in an «element variable declaration» <i>GPVD</i> is said to be declared by <i>GPVD</i>, and by the «element pattern» <i>EP</i> that simply contains <i>GPVD</i>. The «element variable» is the name of an element variable, which is also declared by <i>GPVD</i> and <i>EP</i>. If <i>GPVD</i> simply contains <i>TEMP</i>, then <i>EV</i> is a temporary element variable. <i>EV</i> is a primary variable.</p> <p>NOTE 138 — Element bindings to temporary element variables are removed prior to set-theoretic deduplication of matches. See GR 10) of Subclause 10.6, «graph patterns» and GR 14) of Subclause 21.2, "Mechanism for graph pattern matching".</p> <p>17) Prior to the application of syntactic transformations, conforming GQL-language shall not contain an «element variable declaration» that immediately contains <i>TEMP</i>.</p> <p>18) An element variable that is declared by a «node pattern» is a node variable. An element variable that is declared by an «edge pattern» is an edge variable.</p> <p>«WG3:W24-022»</p> <p>19) The scope of an «element variable» <i>EV</i> that is declared by an «element pattern» <i>EP</i> is defined as follows. If <i>EV</i> is a temporary element variable, then the scope of <i>EV</i> is the innermost «path term» containing <i>EP</i>; otherwise, the scope of <i>EV</i> is the innermost «graph pattern binding table» containing <i>EP</i>.</p> <p>20) A «subpath variable» <i>SV</i> contained in a «subpath variable declaration» <i>SYD</i> is said to be declared by <i>SYD</i>, and by the «parenthesized path pattern expression» <i>PPPE</i> that simply contains <i>SYD</i>. <i>SV</i> is the name of a subpath variable, which is also declared by <i>SYD</i> and <i>PPPE</i>.</p> <p>21) If <i>EP</i> is an «element pattern» that contains an «element pattern where clause» <i>EPWC</i>, then <i>EP</i> shall simply contain an «element variable declaration» <i>GPVD</i>.</p> <p>22) If <i>EV</i> is an element variable or subpath variable, and <i>BNF1</i> is an instance of a BNF non-terminal, then the terminology "BNF exposes <i>EV</i>" is defined as follows. The full terminology is one of the following: "BNF exposes <i>EV</i> as an unconditional singleton variable", "BNF exposes <i>EV</i> as a conditional singleton variable", "BNF exposes <i>EV</i> as an effectively bounded group variable" or "BNF exposes <i>EV</i> as an effectively unbounded group variable". The terms "unconditional singleton variable", "conditional singleton variable", "effectively bounded group variable", and "effectively unbounded group variable" are called the <i>degree of exposure</i>.</p> <p>a) An «element pattern» <i>EP</i> that declares an element variable <i>EV</i> exposes <i>EV</i> as an unconditional singleton.</p> <p>b) A «parenthesized path pattern expression» <i>PPPE</i> that simply contains a «subpath variable declaration» that declares <i>EV</i> exposes <i>EV</i> as an unconditional singleton variable. <i>PPPE</i> shall not contain another «parenthesized path pattern expression» that declares <i>EV</i>.</p> <p>c) If a «path concatenation» <i>PPC</i> declares <i>EV</i> then <i>PT</i> be the «path term» and let <i>PP</i> be the «path factor» simply contained in <i>PPC</i>.</p> <p>Case:</p> <p>i) If <i>EV</i> is exposed as an unconditional singleton by both <i>PT</i> and <i>PP</i>, then <i>EV</i> is exposed as an unconditional singleton by <i>PPC</i>. <i>EV</i> shall not be a subpath variable.</p>

227

16.10 <path pattern expression>
<pre>«simplified override any direction» would use the tokens «-» and «- » surrounding a label as originally proposed in WG3:MANU 066. These synonyms might be considered to make the table of edge patterns more harmonious and internally consistent. See Language Opportunity (CQ-212).</pre>
<pre>«abbreviated edge pattern» ::= «left arrow» «tilde» «right arrow» «tilde right arrow» «tilde left arrow» «tilde minus right» «minus sign» «parenthesized path pattern expression» ::= [«subpath variable declaration»] «path mode prefix» «path pattern expression» [«parenthesized path pattern where clause»] «right paren» «subpath variable declaration» ::= «subpath variable» «equals operator» «parenthesized path pattern where clause» ::= WHERE «search condition» ** Editor's Note (number 74) ** It may be possible to permit nested quantifiers. WG3:W61-014 contained a discussion of a way to support aggregation at different depths of aggregation if there are nested quantifiers. See Language Opportunity (CQ-016).</pre>

16.10 <path pattern expression>
<p>Syntax Rules</p> <p>1) Let <i>RIGHTMINUS</i> be the following collection of «tokens»: «right bracket minus», «left arrow», «slash minus», and «minus sign».</p> <p>NOTE 132 — These are the tokens , <, /, and -, which expose a minus sign on the right.</p> <p>2) Let <i>LEFTMINUS</i> be the following collection of «tokens»: «minus left bracket», «right arrow», «minus slash», and «minus sign».</p> <p>NOTE 133 — These are the tokens , >, /, and -, which expose a minus sign on the left. «minus sign» itself is in both <i>RIGHTMINUS</i> and <i>LEFTMINUS</i>.</p> <p>3) A «path pattern expression» shall not juxtapose a «tokens» from <i>RIGHTMINUS</i> followed by a «tokens» from <i>LEFTMINUS</i> without a «separator» between them.</p> <p>NOTE 134 — Otherwise, the concatenation of the two tokens would include the sequence of two «minus signs», which is a simple comment introducer.</p> <p>4) A «path pattern expression» that contains at the same depth of graph pattern matching a variable quantifier, a «questioned path primary», a «path multiset alternation», or a «path pattern union» is a possibly variable length path pattern.</p> <p>5) A «path pattern expression» that is not a possibly variable length path pattern is a <i>fixed length path pattern</i>.</p> <p>6) The minimum path length of certain BNF non-terminals defined in this Subclause is defined recursively as follows:</p> <p>a) The minimum path length of a «node pattern» is 0 (zero).</p> <p>b) The minimum path length of an «edge pattern» is 1 (one).</p>

223

16.10 <path pattern expression>
<p>NOTE 139 — This case expresses an implicit join on <i>EV</i> within <i>PPC</i>. Implicit joins between conditional singleton variables, group variables, or subpath variables are forbidden.</p> <p>ii) Otherwise, <i>EV</i> shall only be exposed by one of <i>PT</i> or <i>PP</i>. In this case <i>EV</i> is exposed by <i>PPC</i> in the same degree that it is exposed by <i>PT</i> or <i>PP</i>.</p> <p>d) If a «path pattern union» or «path multiset alternation» <i>PA</i> declares <i>EV</i>, then</p> <p>Case:</p> <p>i) If every operand of <i>PA</i> exposes <i>EV</i> as an unconditional singleton variable, then <i>PA</i> exposes <i>EV</i> as an unconditional singleton variable.</p> <p>ii) If at least one operand of <i>PA</i> exposes <i>EV</i> as an effectively unbounded group variable, then <i>PA</i> exposes <i>EV</i> as an effectively unbounded group variable.</p> <p>iii) If at least one operand of <i>PA</i> exposes <i>EV</i> as an effectively bounded group variable, then <i>PA</i> exposes <i>EV</i> as an effectively bounded group variable.</p> <p>iv) Otherwise, <i>PA</i> exposes <i>EV</i> as a conditional singleton variable.</p> <p>e) If a «quantified path primary» <i>QPP</i> declares <i>EV</i>, then let <i>PP</i> be the «path primary» simply contained in <i>QPP</i>.</p> <p>Case:</p> <p>i) If <i>QPP</i> contains a «graph pattern quantifier» that is a «fixed quantifier» or a «general quantifier» that contains an «upper bound» and <i>PP</i> does not expose <i>EV</i> as an effectively unbounded group variable, then <i>QPP</i> exposes <i>EV</i> as an effectively bounded group variable.</p> <p>ii) If <i>QPP</i> is contained at the same depth of graph pattern matching in a restrictive «parenthesized path pattern expression», then <i>QPP</i> exposes <i>EV</i> as an effectively bounded group variable.</p> <p>NOTE 140 — The preceding definition is applied after the syntactic transformation to insure that every «path mode prefix» is at the head of a «parenthesized path pattern expression».</p> <p>iii) Otherwise, <i>QPP</i> exposes <i>EV</i> as an effectively unbounded group variable.</p> <p>f) If a «questioned path primary» <i>QPP</i> declares <i>EV</i>, then let <i>PP</i> be the «path primary» simply contained in <i>QPP</i>.</p> <p>Case:</p> <p>i) If <i>PP</i> exposes <i>EV</i> as a group variable, then <i>QPP</i> exposes <i>EV</i> as a group variable with the same degree of exposure.</p> <p>ii) Otherwise, <i>QPP</i> exposes <i>EV</i> as a conditional singleton variable.</p> <p>g) A «parenthesized path pattern expression» exposes the same variables as the simply contained «path pattern expression», in the same degree of exposure.</p> <p>NOTE 141 — A restrictive «path mode» declared by a «parenthesized path pattern expression» makes variables effectively bounded, but it does so even for proper subexpressions within the scope of the «path mode» and has already been handled by the rules for «quantified path primary».</p> <p>h) If a «path pattern» <i>PP</i> declares <i>EV</i>, then let <i>PPE</i> be the simply contained «path pattern expression».</p> <p>Case:</p> <p>i) If <i>PPE</i> exposes <i>EV</i> as an unconditional singleton, a conditional singleton, or an effectively bounded group variable, then <i>PP</i> exposes <i>EV</i> with the same degree of exposure.</p>

227

16.10 <path pattern expression>
<p>c) The minimum path length of a «path concatenation» is the sum of the minimum path lengths of its operands.</p> <p>d) The minimum path length of a «path pattern union» or «path multiset alternation» is the minimum of the minimum path length of its operands.</p> <p>e) The minimum path length of a «quantified path primary» is the product of the minimum path length of the simply contained «path primary» and the value of the «lower bound».</p> <p>f) The minimum path length of a «questioned path primary» is 0 (zero).</p> <p>g) The minimum path length of a «parenthesized path pattern expression» is the minimum path length of the simply contained «path pattern expression».</p> <p>h) If <i>BNF1</i> and <i>BNF2</i> are two BNF non-terminals such that <i>BNF1</i> ::= <i>BNF2</i> and the minimum path length of <i>BNF2</i> is defined, then the minimum path length of <i>BNF1</i> is also defined and is the same as the minimum path length of <i>BNF2</i>.</p> <p>7) The «path primary» immediately contained in a «quantified path primary» or «questioned path primary» shall have minimum path length that is greater than 0 (zero).</p> <p>8) The «path primary» simply contained in a «quantified path primary» shall not contain a «quantified path primary» at the same depth of graph pattern matching.</p>

224

16.10 <path pattern expression>
<p>9) Let <i>PMA</i> be a «path multiset alternation».</p> <p>a) A «path term» simply contained in <i>PMA</i> is a <i>multiset alternation operand</i> of <i>PMA</i>.</p> <p>b) Let <i>NOPMA</i> be the number of multiset alternation operands of <i>PMA</i>. Let <i>OPMA</i>₁, ..., <i>OPMA</i>_{<i>NOPMA</i>} be an enumeration of the operands of <i>PMA</i>.</p> <p>c) Any «subpath variables» declared by «subpath variable declarations» simply contained in the multiset alternation operands of <i>PMA</i> shall be mutually distinct.</p> <p>d) Let <i>SOPMA</i>₁, ..., <i>SOPMA</i>_{<i>NOPMA</i>} be implementation-dependent [[UV08]] «identifiers» that are mutually distinct and distinct from every «element variable», «subpath variable» and «path variable» contained in <i>GP</i>.</p> <p>e) For every <i>i</i>, 1 (one) ≤ <i>i</i> ≤ <i>NOPMA</i>.</p> <p>Case:</p> <p>i) If <i>OPMA</i>_{<i>i</i>} is a «parenthesized path pattern expression» that simply contains a «subpath variable declaration», then let <i>OPMA</i>_{<i>i</i>} be <i>OPMA</i>_{<i>i</i>}.</p> <p>ii) Otherwise, let <i>OPMA</i>_{<i>i</i>} be the «parenthesized path pattern expression» (<i>SOPMA</i>_{<i>i</i>} = <i>OPMA</i>_{<i>i</i>}).</p> <p>f) <i>PMA</i> is equivalent to:</p> <p><i>OPMA</i>₁ ... <i>OPMA</i>_{<i>NOPMA</i>}</p> <p>10) A «path term» <i>PPUOP</i> simply contained in a «path pattern union» <i>PSD</i> is a <i>path pattern union operand</i> of <i>PSD</i>.</p>

224

16.10 <path pattern expression>
<p>** Editor's Note (number 75) **</p> <p>Path pattern union is not defined using left recursion. WG3:SM-052 believed that it should be possible to support left recursion but declined to do so for expediency. It is a Language Opportunity to support left recursion. See Language Opportunity (CQ-025).</p> <p><i>PPUOP</i> shall not contain a reference to an element variable that is not declared in <i>PPUOP</i> or outside of <i>PSD</i>.</p> <p>11) An «element pattern» <i>EP</i> that contains an «element pattern where clause» <i>EPWC</i> is transformed as follows:</p> <p>a) Let <i>EPF</i> be the «element pattern filler» simply contained in <i>EP</i>.</p> <p>b) Let <i>PREFIX</i> be the «delimiter tokens» contained in <i>EP</i> before <i>EPF</i> and let <i>SUFFIX</i> be the «delimiter tokens» contained in <i>EP</i> after <i>EPF</i>.</p> <p>c) Let <i>EV</i> be the «element variable» simply contained in <i>EPF</i>. Let <i>LLE</i> be the «is label expression» contained in <i>EPF</i>, if any; otherwise, let <i>LLE</i> be the zero-length string.</p> <p>d) <i>EP</i> is replaced by</p> <p>(<i>PREFIX</i> <i>EV</i> <i>LLE</i> <i>SUFFIX</i> <i>EPWC</i>)</p> <p>12) An «element pattern» that does not contain an «element variable declaration», an «is label expression», or an «element pattern predicate» is said to be <i>empty</i>.</p> <p>13) Each «path pattern expression» is transformed in the following steps:</p> <p>a) If the «path primary» immediately contained in a «quantified path primary» or «questioned path primary» is an «edge pattern» <i>EP</i>, then <i>EP</i> is replaced by</p> <p>(<i>EP</i>)</p> <p>NOTE 135 — For example,</p> <p>... becomes: (- -) (>) (<) which in later transformations becomes: (- -) (>) (<)</p> <p>b) If two successive «element patterns» contained in a «path concatenation» at the same depth of graph pattern matching are «edge patterns», then an implicit empty «node pattern» is inserted between them.</p> <p>c) If an edge pattern <i>EP</i> contained in a «path term» <i>PST</i> at the same depth of graph pattern matching is not preceded by a «node pattern» contained in <i>PST</i> at the same depth of graph pattern matching, then an implicit empty «node pattern» <i>VP</i> is inserted in <i>PST</i> immediately prior to <i>EP</i>.</p> <p>d) If an edge pattern <i>EP</i> contained in a «path term» <i>PST</i> at the same depth of graph pattern matching is not followed by a «node pattern» contained in <i>PST</i> at the same depth of graph pattern matching, then an implicit empty «node pattern» <i>VP</i> is inserted in <i>PST</i> immediately after <i>EP</i>.</p> <p>NOTE 136 — As a result of the preceding transformations, a fixed length path pattern has an odd number of «element patterns», beginning and ending with «node patterns», and alternating between «node patterns» and «edge patterns».</p>

225

16.10 <path pattern expression>
<p>NOTE 139 — This case expresses an implicit join on <i>EV</i> within <i>PPC</i>. Implicit joins between conditional singleton variables, group variables, or subpath variables are forbidden.</p> <p>ii) Otherwise, <i>EV</i> shall only be exposed by one of <i>PT</i> or <i>PP</i>. In this case <i>EV</i> is exposed by <i>PPC</i> in the same degree that it is exposed by <i>PT</i> or <i>PP</i>.</p> <p>d) If a «path pattern union» or «path multiset alternation» <i>PA</i> declares <i>EV</i>, then</p> <p>Case:</p> <p>i) If every operand of <i>PA</i> exposes <i>EV</i> as an unconditional singleton variable, then <i>PA</i> exposes <i>EV</i> as an unconditional singleton variable.</p> <p>ii) If at least one operand of <i>PA</i> exposes <i>EV</i> as an effectively unbounded group variable, then <i>PA</i> exposes <i>EV</i> as an effectively unbounded group variable.</p> <p>iii) If at least one operand of <i>PA</i> exposes <i>EV</i> as an effectively bounded group variable, then <i>PA</i> exposes <i>EV</i> as an effectively bounded group variable.</p> <p>iv) Otherwise, <i>PA</i> exposes <i>EV</i> as a conditional singleton variable.</p> <p>e) If a «quantified path primary» <i>QPP</i> declares <i>EV</i>, then let <i>PP</i> be the «path primary» simply contained in <i>QPP</i>.</p> <p>Case:</p> <p>i) If <i>QPP</i> contains a «graph pattern quantifier» that is a «fixed quantifier» or a «general quantifier» that contains an «upper bound» and <i>PP</i> does not expose <i>EV</i> as an effectively unbounded group variable, then <i>QPP</i> exposes <i>EV</i> as an effectively bounded group variable.</p> <p>ii) If <i>QPP</i> is contained at the same depth of graph pattern matching in a restrictive «parenthesized path pattern expression», then <i>QPP</i> exposes <i>EV</i> as an effectively bounded group variable.</p> <p>NOTE 140 — The preceding definition is applied after the syntactic transformation to insure that every «path mode prefix» is at the head of a «parenthesized path pattern expression».</p> <p>iii) Otherwise, <i>QPP</i> exposes <i>EV</i> as an effectively unbounded group variable.</p> <p>f) If a «questioned path primary» <i>QPP</i> declares <i>EV</i>, then let <i>PP</i> be the «path primary» simply contained in <i>QPP</i>.</p> <p>Case:</p> <p>i) If <i>PP</i> exposes <i>EV</i> as a group variable, then <i>QPP</i> exposes <i>EV</i> as a group variable with the same degree of exposure.</p> <p>ii) Otherwise, <i>QPP</i> exposes <i>EV</i> as a conditional singleton variable.</p> <p>g) A «parenthesized path pattern expression» exposes the same variables as the simply contained «path pattern expression», in the same degree of exposure.</p> <p>NOTE 141 — A restrictive «path mode» declared by a «parenthesized path pattern expression» makes variables effectively bounded, but it does so even for proper subexpressions within the scope of the «path mode» and has already been handled by the rules for «quantified path primary».</p> <p>h) If a «path pattern» <i>PP</i> declares <i>EV</i>, then let <i>PPE</i> be the simply contained «path pattern expression».</p> <p>Case:</p> <p>i) If <i>PPE</i> exposes <i>EV</i> as an unconditional singleton, a conditional singleton, or an effectively bounded group variable, then <i>PP</i> exposes <i>EV</i> with the same degree of exposure.</p>

225

Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax

IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

Function

Specify a pattern to match a single path in a property graph.

Format

```
<path pattern expression> ::=
<path term>
| <path multiset alternation>
| <path pattern union>

<path multiset alternation> ::=
<path term>
| { ( <multiset alternation operator> <path term>
... ) }

<path pattern union> ::=
<path term> | <vertical bar> <path term> | ... |

<path term> ::=
<path factor>
| <path concatenation>
| <path term> <path factor>

<path factor> ::=
<path primary>
| <quantified path primary>
| <questioned path primary>

<quantified path primary> ::=
<path primary> <graph pattern quantifier>

<questioned path primary> ::=
<path primary> <question mark>
```

NOTE 131 — Unlike most regular expression languages, <question mark> is not equivalent to the quantifier {0,1} the quantifier {0,1} exposes variables as group, whereas <question mark> does not change the singleton variables that it exposes to group. However, <question mark> does expose any singleton variables as conditional singletons.

```
<path primary> ::=
<element pattern>
| <parenthesized path pattern expression>
| <simplified path pattern expression>

<element pattern> ::=
<node pattern>
| <edge pattern>

<node pattern> ::=
<left paren> <element pattern filler> <right paren>

<element pattern filler> ::=
| <element variable declaration> |
| <is label expression> |
| <element pattern predicate> |
| ... |

+ WG3:W24-022 +
```

```
<element variable declaration> ::=
<element variable declaration> ::=
<is label expression> ::=
| <element pattern predicate> ::=
| ... |

+ WG3:W24-022 +
```

221

IWD 39075:202y(E)
16.10 <path pattern expression>

```
[ <TEMP> <element variable>
<is label expression> ::=
<is or colon> <label expression>
| <colon>

<element pattern predicate> ::=
| <element pattern where clause>
| <element property specification>

<element pattern where clause> ::=
WHERE <search condition>

<element property specification> ::=
<left brace> <property key value pair list> <right brace>

<property key value pair list> ::=
<property key value pair> | { <comma> <property key value pair> } ... |

<property key value pair> ::=
<property name> <colon> <value expression>

<edge pattern> ::=
<full edge pattern>
| <abbreviated edge pattern>

<full edge patterns> ::=
| <full edge pointing left>
| <full edge undirected>
| <full edge pointing right>
| <full edge left or undirected>
| <full edge undirected or right>
| <full edge left or right>
| <full edge any direction>

<full edge pointing left> ::=
<left arrow bracket> <element pattern filler> <right bracket minus>

<full edge undirected> ::=
<tilde left bracket> <element pattern filler> <right bracket tilde>

<full edge pointing right> ::=
<minus left bracket> <element pattern filler> <bracket right arrow>

<full edge left or undirected> ::=
<left arrow tilde bracket> <element pattern filler> <right bracket tilde>

<full edge undirected or right> ::=
<tilde left bracket> <element pattern filler> <bracket tilde right arrow>

<full edge left or right> ::=
<left arrow bracket> <element pattern filler> <bracket right arrow>

<full edge any direction> ::=
<minus left bracket> <element pattern filler> <right bracket minus>
```

```
** Editor's Note (number 73) **
In the BNF for <full edge any direction>, the delimiter tokens <-|> have been suggested as a synonym for |] as part of Feature GAW7, "Undirected edge pattern". The synonym for the <abbreviated edge pattern> (<colon> sign) would then be <->, the synonym for <simplified defaulting any direction> would use the delimiter tokens <- / -> and the synonym for
```

222

IWD 39075:202y(E)
16.10 <path pattern expression>

```
<simplified override any direction> would use the tokens <- and -> surrounding a label as originally proposed in WG3:MANU 066. These synonyms might be considered to make the table of edge patterns more harmonious and internally consistent. See Language Opportunity (CQ-212)

<abbreviated edge pattern> ::=
| <left arrow>
| <tilde>
| <right arrow>
| <tilde right arrow>
| <tilde left arrow>
| <tilde minus right>
| <minus sign>

+ WG3:W24-022 deleted one Editor's Note +

<parenthesized path pattern expression> ::=
<left paren>
| <subpath variable declaration> |
| <path mode prefix> |
| <path pattern expression>
| <parenthesized path pattern where clause> |
| <right paren>

<subpath variable declaration> ::=
<subpath variable> <equals operator>

<parenthesized path pattern where clause> ::=
WHERE <search condition>
```

Syntax Rules

- Let *RIGHTMINUS* be the following collection of <tokens>- <right bracket minus>, <left arrow>, <slash minus>, and <minus sign>.
NOTE 132 — These are the tokens |, <-, /, and -, which expose a minus sign on the right.
- Let *LEFTMINUS* be the following collection of <tokens>- <minus left bracket>, <right arrow>, <minus slash>, and <minus sign>.
NOTE 133 — These are the tokens |, <-, /, and -, which expose a minus sign on the left. <minus sign> itself is in both *RIGHTMINUS* and *LEFTMINUS*.
- A <path pattern expression> shall not juxtapose a <tokens> from *RIGHTMINUS* followed by a <tokens> from *LEFTMINUS* without a <separator> between them.
NOTE 134 — Otherwise, the concatenation of the two tokens would include the sequence of two <minus signs>, which is a simple common introducer.
- A <path pattern expression> that contains at the same depth of graph pattern matching a variable quantifier, a <questioned path primary>, a <path multiset alternation>, or a <path pattern union> is a possibly variable length path pattern.
- A <path pattern expression> that is not a possibly variable length path pattern is a *fixed length path pattern*.
- The minimum path length of certain BNF non-terminals defined in this Subclause is defined recursively as follows:
 - The minimum path length of a <node pattern> is 0 (zero).
 - The minimum path length of an <edge pattern> is 1 (one).

223

IWD 39075:202y(E)
16.10 <path pattern expression>

- The minimum path length of a <path concatenation> is the sum of the minimum path lengths of its operands.
- The minimum path length of a <path pattern union> or <path multiset alternation> is the minimum of the minimum path length of its operands.
- The minimum path length of a <quantified path primary> is the product of the minimum path length of the simply contained <path primary> and the value of the <lower bound>.
- The minimum path length of a <questioned path primary> is 0 (zero).
- The minimum path length of a <parenthesized path pattern expression> is the minimum path length of the simply contained <path pattern expression>.
- If *BNT1* and *BNT2* are two BNF non-terminals such that *BNT1* ::= *BNT2* and the minimum path length of *BNT2* is defined, then the minimum path length of *BNT1* is also defined and is the same as the minimum path length of *BNT2*.
- The <path primary> immediately contained in a <quantified path primary> or <questioned path primary> shall have minimum path length that is greater than 0 (zero).
- The <path primary> simply contained in a <quantified path primary> shall not contain a <quantified path primary> at the same depth of graph pattern matching.

```
** Editor's Note (number 74) **
It may be possible to permit nested quantifiers. WG3:W1-014 contained a discussion of a way to support aggregates at different depths of aggregation if there are nested quantifiers. See Language Opportunity (CQ-016)
```

- Let *PMA* be a <path multiset alternation>.
 - A <path term> simply contained in *PMA* is a *multiset alternation operand of PMA*.
 - Let *NOPMA* be the number of multiset alternation operands of *PMA*. Let *OPMA*₁, ..., *OPMA*_{NOPMA} be an enumeration of the operands of *PMA*.
 - Any <subpath variables> declared by <subpath variable declarations> simply contained in the multiset alternation operands of *PMA* shall be mutually distinct.
 - Let *SOPMA*₁, ..., *SOPMA*_{NOPMA} be implementation-dependent (UV008) <identifiers> that are mutually distinct and distinct from every <element variable>, <subpath variable> and <path variable> contained in *P*.
 - For every *i*, 1 (one) ≤ *i* ≤ *NOPMA*.
 - If *OPMA*_i is a <parenthesized path pattern expression> that simply contains a <subpath variable declaration>, then let *OPMA*_i be *OPMA*_i.
 - Otherwise, let *OPMA*_i be the <parenthesized path pattern expression> (*OPMA*₁ - *OPMA*_{NOPMA})
 - PMA* is equivalent to:
*OPMA*₁ | ... | *OPMA*_{NOPMA}
- A <path pattern expression> simply contained in a <path pattern union> is a *path pattern union operand of PSD*.

224

- Otherwise, *PP* exposes *EV* as an effectively bounded group variable.
NOTE 142 — That is, even if *PPE* exposes *EV* as an effectively unbounded group variable, *PP* shall expose *EV* as effectively bounded, because in this case *PP* is required to be selective <path pattern>.
- If *BNT1* and *BNT2* are two BNF non-terminals such that *BNT1* ::= *BNT2* and *BNT2* exposes *EV*, then *BNT1* exposes *EV* to the same degree of exposure as *BNT2*.

```
** Editor's Note (number 76) **
WG3:W04-009R1 defined "effectively bounded group variable" but did not use the definition. The definition will be used when we define predicates on aggregates, at which time we will want a Syntax Rules stating that if a group variable EV is referenced in a WHERE clause, then it shall be effectively bounded and the reference shall be contained in an aggregated argument of an aggregate function. See Possible Problem (CQ-050)
```

- If *BNT1* is a BNF non-terminal that exposes a graph pattern variable *GPV* with a degree of exposure *DEGREE*, then *BNT1* is also said to expose the name of *GPV* with degree of exposure *DEGREE*.
- A <parenthesized path pattern where clause> *PPW* simply contained in a <parenthesized path pattern expression> *PPP* shall not reference a path variable.

```
** Editor's Note (number 77) **
WG3:W04-009R1 recognized that a graph query may have a sequence of MATCH clauses, with the bindings of one MATCH clause M1 usable in all subsequent MATCH clauses in the same invocation of <graph table>, and that it should be permissible to reference such variables in any <parenthesized path pattern where clause> simply contained in a subsequent MATCH clause M2. The relevance of this LD to GQL needs to be investigated. See Language Opportunity (CQ-051)
```

General Rules

None.
NOTE 143 — The evaluation of a <path pattern expression> is performed by the General Rules of Subclause 21.3, "Evaluation of a <path pattern expression>".

Conformance Rules

- Without Feature G030, "Path Multiset Alternation", conforming GQL language shall not contain a <path multiset alternation>.
- Without Feature G031, "Path Multiset Alternation: variable length path operands", in conforming GQL language, an operand of a <path multiset alternation> shall be a fixed length path pattern.
- Without Feature G032, "Path Pattern Union", conforming GQL language shall not contain a <path pattern union>.
- Without Feature G033, "Path Pattern Union: variable length path operands", in conforming GQL language, an operand of a <path pattern union> shall be a fixed length path pattern.
- Without Feature G035, "Quantified Paths", conforming GQL language shall not contain a <quantified path primary> that does not immediately contain a <path primary> that is an <edge pattern>.
- Without Feature G036, "Quantified Edges", conforming GQL language shall not contain a <quantified path primary> that immediately contains a <path primary> that is an <edge pattern>.
- Without Feature G037, "Questioned Paths", conforming GQL language shall not contain a <questioned path primary>.

IWD 39075:202y(E)
16.10 <path pattern expression>

```
** Editor's Note (number 75) **
Path pattern union is not defined using left recursion. WG3:SM-052 believed that it should be possible to support left recursion but declined to do so for expediency. It is a Language Opportunity to support left recursion. See Language Opportunity (CQ-025)
```

PPUOP shall not contain a reference to an element variable that is not declared in *PPUOP* or outside of *PSD*.

- An <element pattern> *EP* that contains an <element pattern where clause> *EPWC* is transformed as follows:
 - Let *EPF* be the <element pattern filler> simply contained in *EP*.
 - Let *PREFIX* be the <delimiter tokens> contained in *EP* before *EPF* and let *SUFFIX* be the <delimiter tokens> contained in *EP* after *EPF*.
 - Let *EV* be the <element variable> simply contained in *EPF*. Let *LLE* be the <is label expression> contained in *EPF*, if any; otherwise, let *LLE* be the zero-length string.
 - EP* is replaced by
(*PREFIX* *EV* *LLE* *SUFFIX* *EPWC*)
- An <element pattern> that does not contain an <element variable declaration>, an <is label expression>, or an <element pattern predicate> is said to be *empty*.
- Each <path pattern expression> is transformed in the following steps:
 - If the <path primary> immediately contained in a <quantified path primary> or <questioned path primary> is an <edge pattern> *EP*, then *EP* is replaced by
(*EP*)
NOTE 135 — For example,
...
...
becomes:
(->) (x.)
which in later transformations becomes:
(1) -> (1) (x.)
 - If two successive <element pattern> are contained in a <path concatenation> at the same depth of graph pattern matching are <edge pattern>, then an implicit empty <node pattern> is inserted between them.
 - If an edge pattern *EP* contained in a <path term> *PST* at the same depth of graph pattern matching is not preceded by a <node pattern> contained in *PST* at the same depth of graph pattern matching, then an implicit empty <node pattern> *VP* is inserted in *PST* immediately prior to *EP*.
 - If an edge pattern *EP* contained in a <path term> *PST* at the same depth of graph pattern matching is not followed by a <node pattern> contained in *PST* at the same depth of graph pattern matching, then an implicit empty <node pattern> *VP* is inserted in *PST* immediately after *EP*.

NOTE 136 — As a result of the preceding transformations, a fixed length path pattern has an odd number of <element pattern>, beginning and ending with <node pattern>, and alternating between <node pattern> and <edge pattern>.

225

227

227

227

227

227

Standards are great but not for academics

You want to write a paper about pattern matching and start with the syntax

IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

Function

Specify a pattern to match a single path in a property graph.

Format

```
<path pattern expression> ::=
<path term>
| <path multiset alternation>
| <path pattern union>

<path multiset alternation> ::=
<path term>
| { <multiset alternation operator> <path term>
  { <multiset alternation operator> <path term> } ... }

<path pattern union> ::=
<path term> | <vertical bar> <path term> | ... |

<path term> ::=
<path factor>
| <path concatenation>

<path concatenation> ::=
<path term> <path factor>

<path factor> ::=
<path primary>
| <quantified path primary>
| <questioned path primary>

<quantified path primary> ::=
<path primary> <graph pattern quantifier>

<questioned path primary> ::=
<path primary> <question mark>

NOTE 131 — Unlike most regular expression languages, <question mark> is not equivalent to the quantifier {0,1} the
quantifier {0,1} exposes variables as group, whereas <question mark> does not change the singleton variables that it exposes
to group. However, <question mark> does expose any singleton variables as conditional singletons.

<path primary> ::=
<element pattern>
| <parenthesized path pattern expression>
| <simplified path pattern expression>

<element pattern> ::=
<node pattern>
| <edge pattern>

<node pattern> ::=
<left paren> <element pattern filler> <right paren>

<element pattern filler> ::=
[ <element variable declaration> ]
[ <is label expression> ]
[ <element pattern predicate> ]

+ WG1:924-022 +
<element variable declaration> ::=
```

221

IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

Function

Specify a pattern to match a single path in a property graph.

Format

```
[ <TMP> ] <element variable>

<is label expression> ::=
<is or colon> <label expression>
| <colon>

<element pattern predicate> ::=
<element pattern where clause>
| <element property specification>

<element pattern where clause> ::=
WHERE <search condition>

<element property specification> ::=
<left brace> <property key value pair list> <right brace>

<property key value pair list> ::=
<property key value pair> | { <comma> <property key value pair> } ... }

<property key value pair> ::=
<property name> <colon> <value expression>

<edge pattern> ::=
<full edge pattern>
| <abbreviated edge pattern>

<full edge patterns> ::=
<full edge pointing left>
| <full edge pointing right>
| <full edge undirected>
| <full edge pointing left and right>
| <full edge left or undirected>
| <full edge right or undirected>
| <full edge left or right>
| <full edge any direction>

<full edge pointing left> ::=
<left arrow bracket> <element pattern filler> <right bracket minus>

<full edge undirected> ::=
<tilde left bracket> <element pattern filler> <right bracket tilde>

<full edge pointing right> ::=
<minus left bracket> <element pattern filler> <right bracket right arrow>

<full edge left or undirected> ::=
<left arrow tilde left bracket> <element pattern filler> <right bracket tilde>

<full edge undirected or right> ::=
<tilde left bracket> <element pattern filler> <right bracket tilde right arrow>

<full edge left or right> ::=
<left arrow bracket> <element pattern filler> <right bracket right arrow>

<full edge any direction> ::=
<minus left bracket> <element pattern filler> <right bracket minus>
```

222

IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

- b) If *BNF1* and *BNF2* are two BNF non-terminals such that *BNF1* ::= *BNF2* and the minimum node count of *BNF2* is defined, then the minimum node count of *BNF1* is also defined and is the same as the minimum node count of *BNF2*.
- 15) The <path pattern expression> simply contained in a <path pattern> shall have a minimum node count that is greater than 0 (zero).
- NOTE 137 — The minimum node count is computed after the syntactic transform that adds implicit node patterns. Thus a single <edge pattern> is a permitted <path pattern> because it implies two <node patterns>.
- + WG1:924-022 +
- 16) An <element variable> *EV* contained in an <element variable declaration> *GPVD* is said to be declared by *GPVD*, and by the <element pattern> *EP* that simply contains *GPVD*. The <element variable> is the name of an element variable, which is also declared by *GPVD* and *EP*. If *GPVD* simply contains *TEMP*, then *EV* is a temporary element variable. *EV* is a primary variable.
- NOTE 138 — Element bindings to temporary element variables are removed prior to the theoretical deduplication of matches. See GR 10) of Subclause 16.8, “Graph Patterns” and GR 11) of Subclause 21.2, “Mechanism for graph pattern matching”.
- 17) Prior to the application of syntactic transformations, conforming GQL language shall not contain an <element variable declaration> that immediately contains *TEMP*.
- 18) An element variable that is declared by a <node pattern> is a node variable. An element variable that is declared by an <edge pattern> is an edge variable.
- + WG1:924-022 +
- 19) The scope of an <element variable> *EV* that is declared by an <element pattern> *EP* is defined as follows. If *EV* is a temporary element variable, then the scope of *EV* is the innermost <path term> containing *EP*; otherwise, the scope of *EV* is the innermost <graph pattern binding table> containing *EP*.
- 20) A <subpath variable> *SV* contained in a <subpath variable declaration> *SYD* is said to be declared by *SYD*, and by the <parenthesized path pattern expression> *PPPE* that simply contains *SYD*. *SV* is the name of a subpath variable, which is also declared by *SYD* and *PPPE*.
- 21) *EP* is an <element pattern> that contains an <element pattern where clause> *EPWC*, then *EP* shall simply contain an <element variable declaration> *GPVD*.
- 22) If *EV* is an element variable or subpath variable, and *BNT* is an instance of a BNF non-terminal, then the terminology “*BNT* exposes *EV*” is defined as follows. The full terminology is one of the following: “*BNT* exposes *EV* as an unconditional singleton variable”, “*BNT* exposes *EV* as a conditional singleton variable”, “*BNT* exposes *EV* as an effectively bounded group variable”, or “*BNT* exposes *EV* as an effectively unbounded group variable”. The terms “unconditional singleton variable”, “conditional singleton variable”, “effectively bounded group variable”, and “effectively unbounded group variable” are called the *degree of exposure*.
- a) An <element pattern> *EP* that declares an element variable *EV* exposes *EV* as an unconditional singleton.
- A <parenthesized path pattern expression> *PPPE* that simply contains a <subpath variable declaration> that declares *EV* exposes *EV* as an unconditional singleton variable. *PPPE* shall not contain another <parenthesized path pattern expression> that declares *EV*.
- c) If a <path concatenation> *PPC* declares *EV* then let *PT* be the <path term> and let *PF* be the <path factor> simply contained in *PPC*.
- Case:
- i) If *EV* is exposed as an unconditional singleton by both *PT* and *PF*, then *EV* is exposed as an unconditional singleton by *PPC*. *EV* shall not be a subpath variable.

227

IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

Function

Specify a pattern to match a single path in a property graph.

Format

```
<abbreviated edge pattern> ::=
<left arrow>
| <tilde>
| <right arrow>
| <tilde right arrow>
| <tilde left arrow>
| <tilde minus right>
| <minus sign>

+ WG1:924-024 deleted one Editor's Note +

<parenthesized path pattern expression> ::=
<left paren>
[ <subpath variable declaration> ]
[ <path mode prefix> ]
<path pattern expression>
<right paren>

<subpath variable declaration> ::=
<subpath variable> <equals operator>

<parenthesized path pattern where clause> ::=
WHERE <search condition>
```

223

IWD 39075:202y(E)
16.10 <path pattern expression>

- NOTE 139 — This case expresses an implicit join on *EV* within *PPC*. Implicit joins between conditional singleton variables, group variables, or subpath variables are forbidden.
- ii) Otherwise, *EV* shall only be exposed by one of *PT* or *PF*. In this case *EV* is exposed by *PPC* in the same degree that it is exposed by *PT* or *PF*.
- d) If a <path pattern union> or <path multiset alternation> *PA* declares *EV*, then
- Case:
- i) If every operand of *PA* exposes *EV* as an unconditional singleton variable, then *PA* exposes *EV* as an unconditional singleton variable.
- ii) If at least one operand of *PA* exposes *EV* as an effectively unbounded group variable, then *PA* exposes *EV* as an effectively unbounded group variable.
- iii) If at least one operand of *PA* exposes *EV* as an effectively bounded group variable, then *PA* exposes *EV* as an effectively bounded group variable.
- iv) Otherwise, *PA* exposes *EV* as a conditional singleton variable.
- e) If a <quantified path primary> *QPP* declares *EV*, then let *PP* be the <path primary> simply contained in *QPP*.
- Case:
- i) If *QPP* contains a <graph pattern quantifier> that is a <fixed quantifier> or a <general quantifier> that contains an <upper bound> and *PP* does not expose *EV* as an effectively unbounded group variable, then *QPP* exposes *EV* as an effectively bounded group variable.
- ii) If *QPP* is contained at the same depth of graph pattern matching in a restrictive <parenthesized path pattern expression>, then *QPP* exposes *EV* as an effectively bounded group variable.
- NOTE 140 — The preceding definition is applied after the syntactic transformation to insure that every <path mode prefix> is at the head of a <parenthesized path pattern expression>.
- iii) Otherwise, *QPP* exposes *EV* as an effectively unbounded group variable.
- f) If a <questioned path primary> *QPP* declares *EV*, then let *PP* be the <path primary> simply contained in *QPP*.
- Case:
- i) If *PP* exposes *EV* as a group variable, then *QPP* exposes *EV* as a group variable with the same group of exposure.
- ii) Otherwise, *QPP* exposes *EV* as a conditional singleton variable.
- g) A <parenthesized path pattern expression> *PPPE* that simply contains a <subpath variable declaration> that declares *EV* exposes *EV* as an unconditional singleton variable. *PPPE* shall not contain another <parenthesized path pattern expression> that declares *EV*.
- h) If a <path concatenation> *PPC* declares *EV*, then let *PT* be the <path term> and let *PF* be the <path factor> simply contained in *PPC*.
- Case:
- i) If *PPC* exposes *EV* as an unconditional singleton, a conditional singleton, or an effectively bounded group variable, then *PP* exposes *EV* with the same degree of exposure.

227

IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

Function

Specify a pattern to match a single path in a property graph.

Format

```
c) The minimum path length of a <path concatenation> is the sum of the minimum path lengths of its operands.

d) The minimum path length of a <path pattern union> or <path multiset alternation> is the minimum of the minimum path length of its operands.



e) The minimum path length of a <quantified path primary> is the product of the minimum path length of the simply contained <path primary> and the value of the <lower bound>.



f) The minimum path length of a <questioned path primary> is 0 (zero).



g) The minimum path length of a <parenthesized path pattern expression> is the minimum path length of the simply contained <path pattern expression>.



h) If BNT1 and BNT2 are two BNF non-terminals such that BNT1 ::= BNT2 and the minimum path length of BNT2 is defined, then the minimum path length of BNT1 is also defined and is the same as the minimum path length of BNT2.



7) The <path primary> immediately contained in a <quantified path primary> or <questioned path primary> shall have minimum path length that is greater than 0 (zero).



8) The <path primary> simply contained in a <quantified path primary> shall not contain a <quantified path primary> at the same depth of graph pattern matching.



** Editor's Note (number 74) **



It may be possible to permit nested quantifiers. WG1:924-024 contained a discussion of a way to support aggregation at different depths of aggregation if there are nested quantifiers. See Language Opportunity QOQ-024.


```

224

IWD 39075:202y(E)
16.10 <path pattern expression>

- ii) Otherwise, *PP* exposes *EV* as an effectively bounded group variable.
- NOTE 142 — That is, even if *PPPE* exposes *EV* as an effectively unbounded group variable, *PP* shall expose *EV* as an effectively bounded, because in this case *PP* is required to be selective <path pattern>.
- i) If *BNT1* and *BNT2* are two BNF non-terminals such that *BNT1* ::= *BNT2* and *BNT2* exposes *EV*, then *BNT1* exposes *EV* to the same degree of exposure as *BNT2*.
- ** Editor's Note (number 76) **
- WG1:904-0091 defined “effectively bounded group variable” but did not use the definition. The definition will be used when we define predicates on aggregates, at which time we will want a Syntax Rules stating that if a group variable *GV* is referenced in a *WHERE* clause, then it shall be effectively bounded and the reference shall be contained in an aggregated argument of an aggregate function. See Possible Problem QOQ-050.
- 23) If *BNT* is a BNF non-terminal that exposes a graph pattern variable *GPV* with a degree of exposure *DEGREE*, then *BNT* is also said to expose the name of *GPV* with degree of exposure *DEGREE*.
- 24) A <parenthesized path pattern where clause> *PPPPC* simply contained in a <parenthesized path pattern expression> *PPPE* shall not reference a path variable.
- ** Editor's Note (number 77) **
- WG1:904-0091 recognized that a graph query may have a sequence of *MATCH* clauses, with the bindings of one *MATCH* clause *MC1* visible to all subsequent *MATCH* clauses in the same invocation of <graph table>, and that it should be permissible to reference such variables in any <parenthesized path pattern where clause> simply contained in a subsequent *MATCH* clause *MC2*. The relevance of this ID to GQL needs to be investigated. See Language Opportunity QOQ-051.

224

IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

Function

Specify a pattern to match a single path in a property graph.

Format

```
** Editor's Note (number 75) **
```

Path pattern union is not defined using left recursion. WG1:904-052 believed that it should be possible to support left recursion but decided to do so for expediency. It is a Language Opportunity to support left recursion. See Language Opportunity QOQ-025.

PPUOP shall not contain a reference to an element variable that is not declared in *PPUOP* or outside of *PSD*.

11) An <element pattern> *EP* that contains an <element pattern where clause> *EPWC* is transformed as follows:

a) Let *EPF* be the <element pattern filler> simply contained in *EP*.

b) Let *PREFIX* be the <delimiter tokens> contained in *EP* before *EPF* and let *SUFFIX* be the <delimiter tokens> contained in *EP* after *EPF*.

c) Let *EV* be the <element variable> simply contained in *EPF*. Let *L* be the <is label expression> contained in *EPF*, if any; otherwise, let *L* be the zero-length string.

d) *EP* is replaced by

```
( PREFIX EV L SUFFIX EPWC )
```

12) An <element pattern> that does not contain an <element variable declaration>, an <is label expression>, or an <element pattern predicate> is said to be empty.

13) Each <path pattern expression> is transformed in the following steps:

a) If the <path primary> immediately contained in a <quantified path primary> or <questioned path primary> is an <edge pattern> *EP*, then *EP* is replaced by

```
( EP )
```

NOTE 135 — For example,

```
...
...
becomes:
( - ) ( - )
which in later transformation becomes:
( - ) - ( - ) ( - )
```

b) If two successive <element pattern> are contained in a <path concatenation> at the same depth of graph pattern matching are <edge pattern>, then an implicit empty <node pattern> is inserted between them.

c) If an edge pattern *EP* contained in a <path term> *PST* at the same depth of graph pattern matching is not preceded by a <node pattern> contained in *PST* at the same depth of graph pattern matching, then an implicit empty <node pattern> *VP* is inserted in *PST* immediately prior to *EP*.

d) If an edge pattern *EP* contained in a <path term> *PST* at the same depth of graph pattern matching is not followed by a <node pattern> contained in *PST* at the same depth of graph pattern matching, then an implicit empty <node pattern> *VP* is inserted in *PST* immediately after *EP*.

NOTE 136 — As a result of the preceding transformations, a fixed length path pattern has an odd number of <element pattern>, beginning and ending with <node pattern>, and alternating between <node pattern> and <edge pattern>.

225

IWD 39075:202y(E)
16.10 <path pattern expression>

16.10 <path pattern expression>

- 8) Without Feature G038, “Parenthesized path pattern expression”, conforming GQL language shall not contain a <parenthesized path pattern expression>.
- 9) Without Feature G041, “Non-local element pattern predicates”, in conforming GQL language, the <element pattern where clause> of an <element pattern> *EP* shall only reference the <element variables> declared in *EP*.
- 10) Without Feature G043, “Complete Full Edge Patterns”, conforming GQL language shall not contain a <full edge pattern> that is not a <full edge any direction>, a <full edge pointing left>, or a <full edge pointing right>.
- 11) Without Feature G044, “Basic Abbreviated Edge Patterns”, conforming GQL language shall not contain an <abbreviated edge pattern> that is a <minus sign>, <left arrow>, or <right arrow>.
- 12) Without Feature G045, “Complete Abbreviated Edge Patterns”, conforming GQL language shall not contain an <abbreviated edge pattern> that is not a <minus sign>, <left arrow>, or <right arrow>.
- 13) Without Feature G046, “Related topological consistency: Adjacent vertex patterns”, in conforming GQL language, between any two <node patterns> contained in a <path pattern expression> there shall be at least one <edge pattern>, <left paren>, or <right paren>.
- 14) Without Feature G047, “Related topological consistency: Concise edge patterns”, in conforming GQL language, any <edge pattern> shall be immediately preceded and followed by a <node pattern>.
- 15) Without Feature G048, “Parenthesized Path Pattern: Subpath variable declaration”, conforming GQL language shall not contain a <parenthesized path pattern expression> that simply contains a <subpath variable declaration>.
- 16) Without Feature G049, “Parenthesized Path Pattern: Path mode prefix”, conforming GQL language shall not contain a <parenthesized path pattern expression> that immediately contains a <path mode prefix>.
- 17) Without Feature G050, “Parenthesized Path Pattern: Where clause”, conforming GQL language shall not contain a <parenthesized path pattern where clause>.
- 18) Without Feature G051, “Parenthesized Path Pattern: Non-local predicates”, in conforming GQL language, a <parenthesized path pattern where clause> simply contained in a <parenthesized path pattern expression> *PPPE* shall not reference an <element variable> that is not declared in *PPPE*.

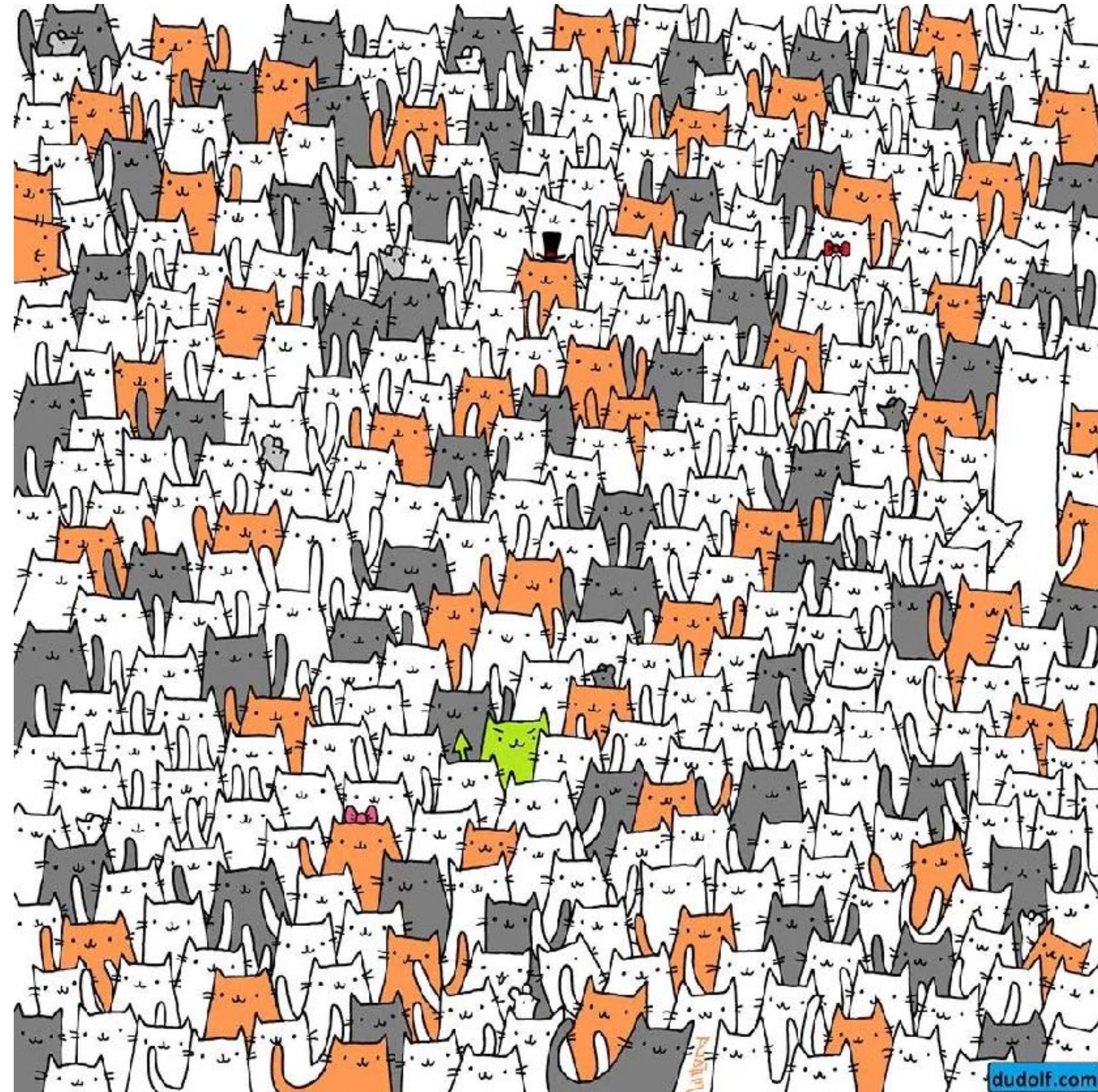
225

Your page limit is over and you're 20% into Definition 1

229

230

And then you want to work with it but it's like *“find the rabbit”*



Our Goal

Our Goal

GQL to the (academic) masses

Our Goal

GQL to the (academic) masses

- Distill

Our Goal

GQL to the (academic) masses

- Distill
- Formalize, provide the semantics

Our Goal

GQL to the (academic) masses

- Distill
- Formalize, provide the semantics
- Plus initial results

Our Goal

GQL to the (academic) masses

- Distill
- Formalize, provide the semantics
- Plus initial results
- Explain what is similar to / different from DB research concepts such as RPQs, CRPQs etc

Our Goal

GQL to the (academic) masses

- Distill
- Formalize, provide the semantics
- Plus initial results
- Explain what is similar to / different from DB research concepts such as RPQs, CRPQs etc
- Outline research challenges that GQL brings

Our Goal

GQL to the (academic) masses

- Distill
- Formalize, provide the semantics
- Plus initial results
- Explain what is similar to / different from DB research concepts such as RPQs, CRPQs etc
- Outline research challenges that GQL brings

Word of caution

GQL is a moving target
We do our best.....

Our Goal

GQL to the (academic) masses

- Distill
- Formalize, provide the semantics
- Plus initial results
- Explain what is similar to / different from DB research concepts such as RPQs, CRPQs etc
- Outline research challenges that GQL brings

Word of caution

GQL is a moving target
We do our best.....



Our Goal

GQL to the (academic) masses

- Distill
- Formalize, provide the semantics
- Plus initial results
- Explain what is similar to / different from DB research concepts such as RPQs, CRPQs etc
- Outline research challenges that GQL brings

Word of caution

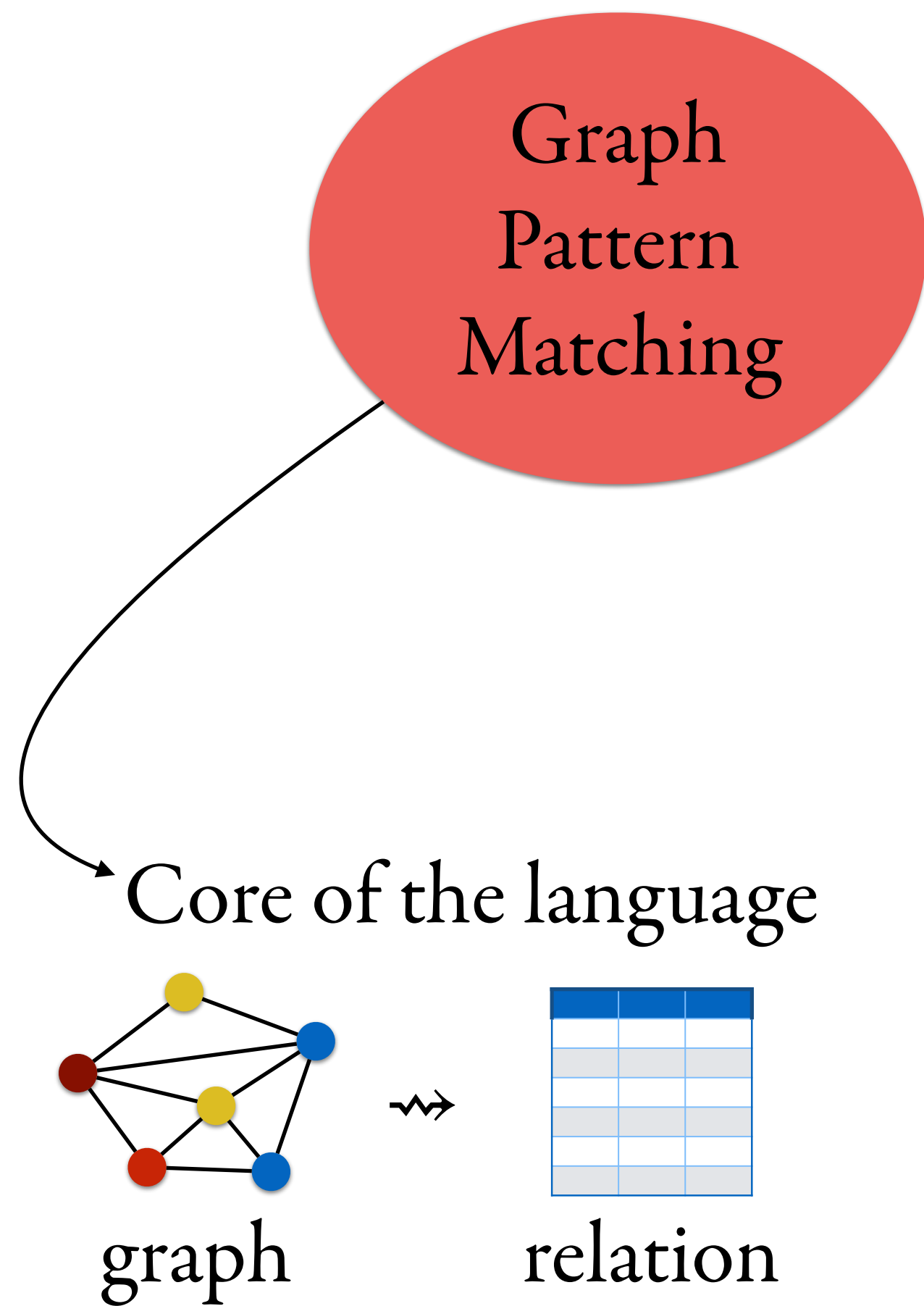
GQL is a moving target
We do our best.....



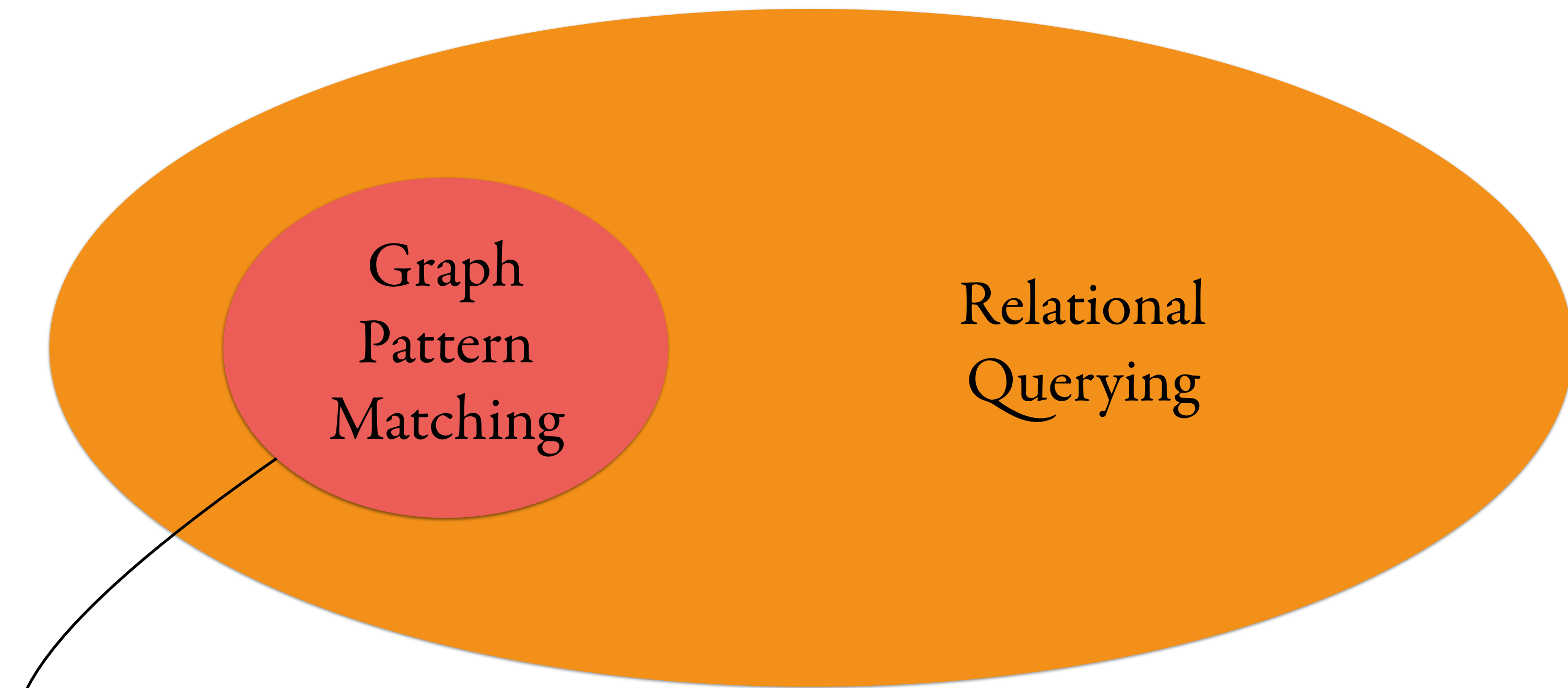
Papers/talks

- Last year: [SIGMOD'22](#) on pattern matching (WG3+FSWG)
- Then [PODS'23](#) paper: formalization of pattern matching
 - also subject of KR 2023 keynote
- [EDBT/ICDT 2023](#) keynote: core GQL
 - talk + paper

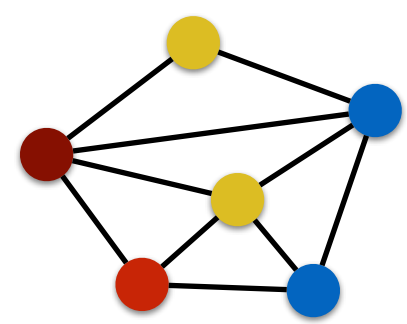
GQL in a Nutshell



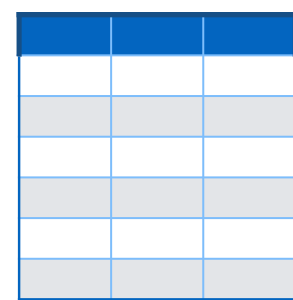
GQL in a Nutshell



Core of the language

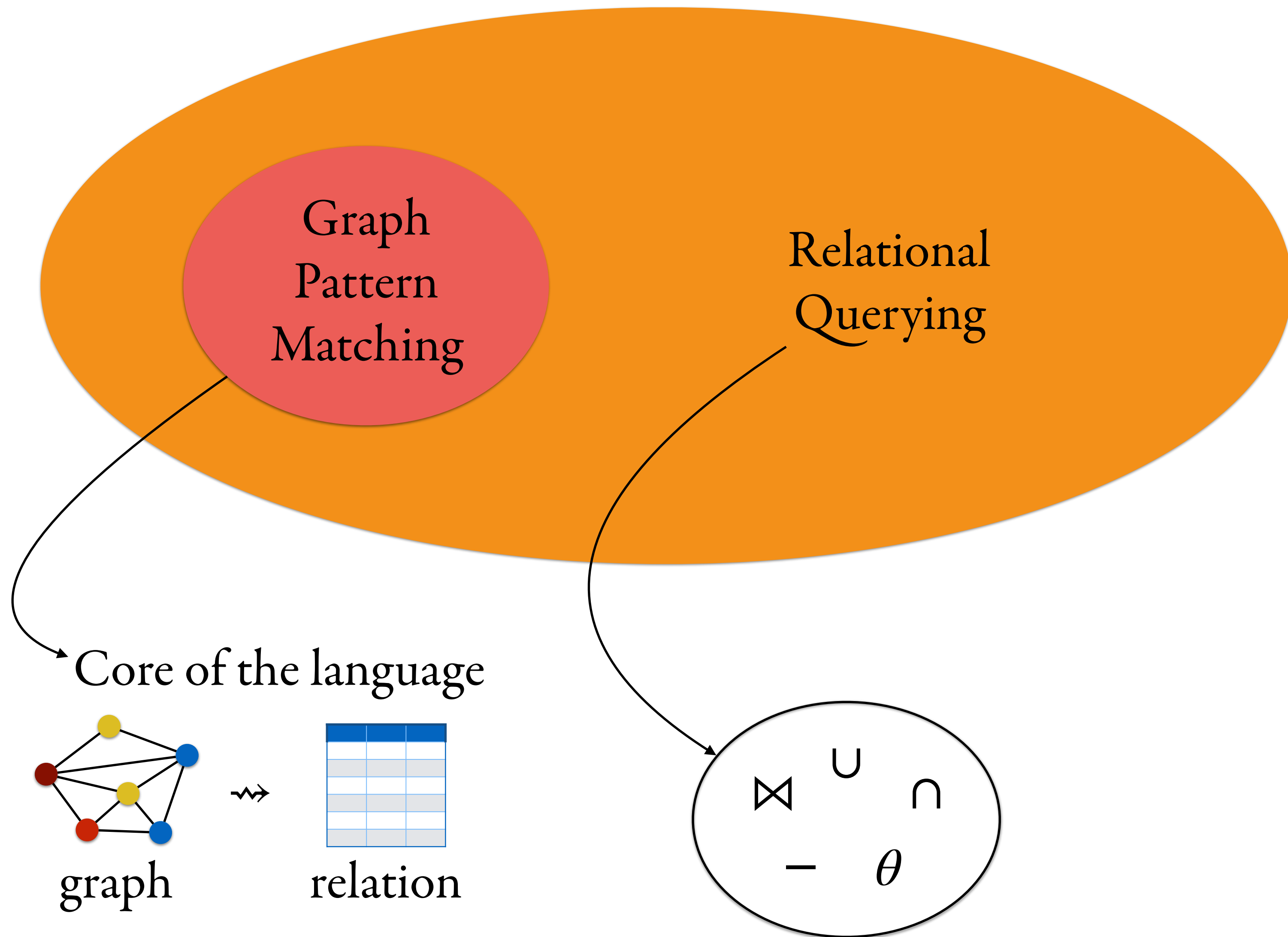


graph

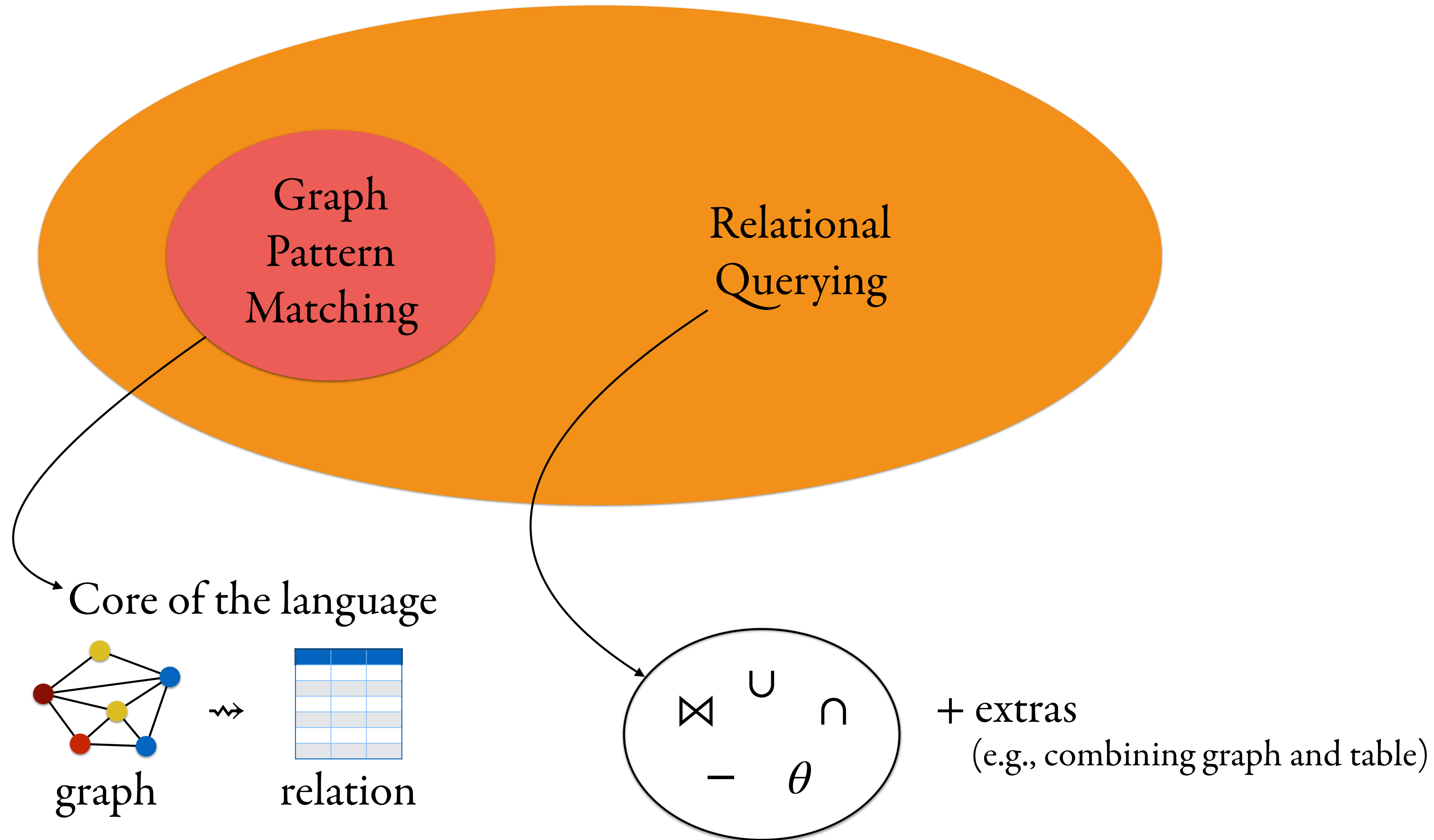


relation

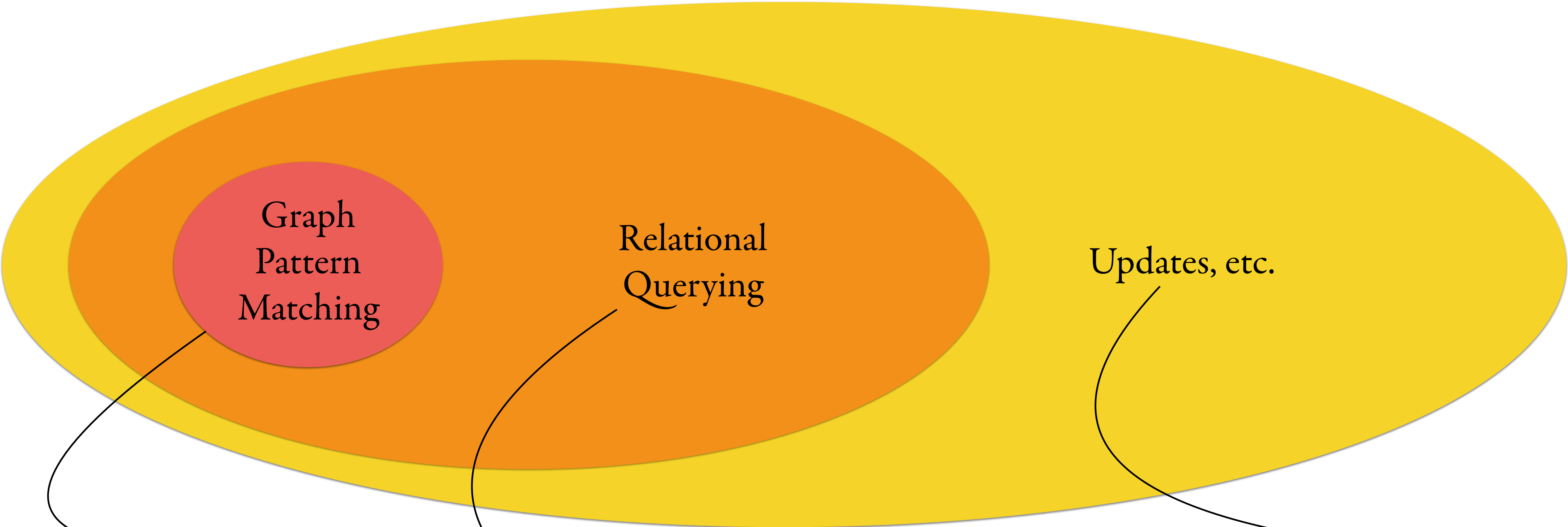
GQL in a Nutshell



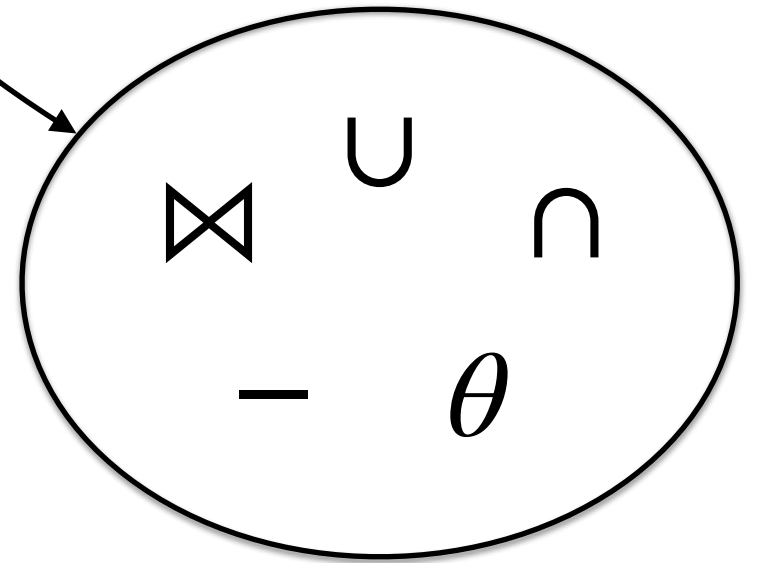
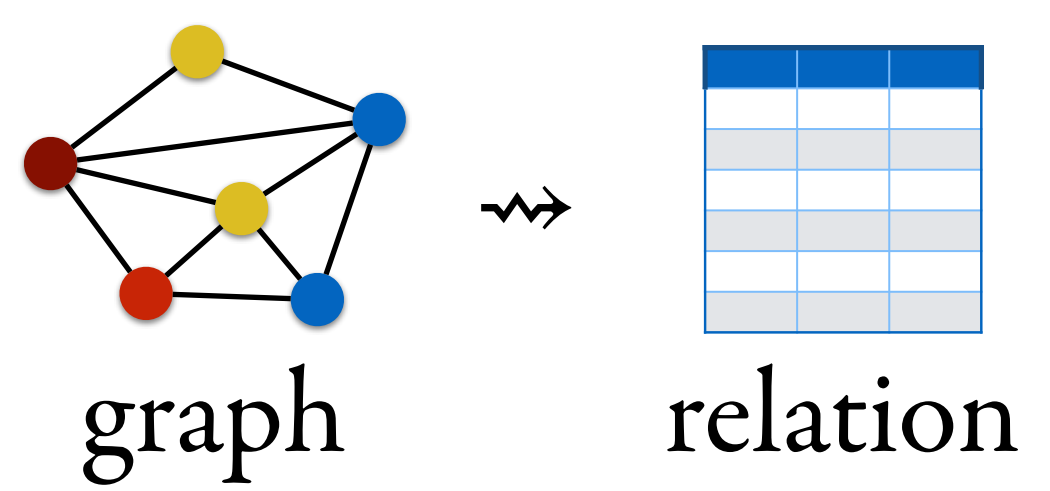
GQL in a Nutshell



GQL in a Nutshell



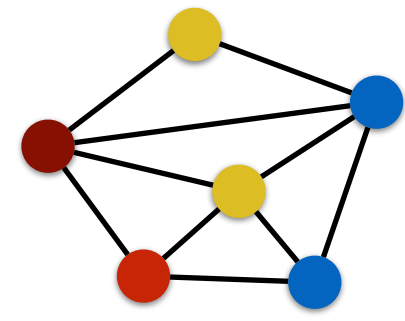
Core of the language



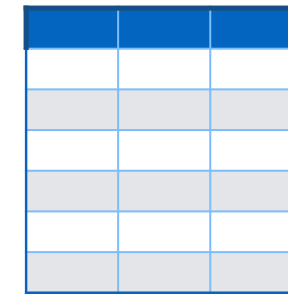
+ extras
(e.g., combining graph and table)

Not yet

The Core: Graph Pattern Matching



graph



relation

Pattern calculus in a nutshell: PODS 23

Pattern calculus in a nutshell: PODS 23

Node pattern $\nu := (x : \ell)$

Pattern calculus in a nutshell: PODS 23

Node pattern

$\nu := (x : \ell)$

match an ℓ -labeled node, assign to a variable

Pattern calculus in a nutshell: PODS 23

Node pattern

$$\nu := (x : \ell)$$

match an ℓ -labeled node, assign to a variable

Edge pattern

$$\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \text{----}$$

Pattern calculus in a nutshell: PODS 23

Node pattern

$$\nu := (x : \ell)$$

match an ℓ -labeled node, assign to a variable

Edge pattern

$$\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \text{---} \xrightarrow{x:\ell}$$

ℓ -labeled edge directed left/right/any-directed, assign to a variable

Pattern calculus in a nutshell: PODS 23

Node pattern

$$\nu := (x : \ell)$$

match an ℓ -labeled node, assign to a variable

Both x and ℓ are optional

Edge pattern

$$\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \text{---} \xrightarrow{x:\ell}$$

ℓ -labeled edge directed left/right/any-directed, assign to a variable

Pattern calculus in a nutshell: PODS 23

Node pattern

$$\nu := (x : \ell)$$

match an ℓ -labeled node, assign to a variable

Both x and ℓ are optional

Edge pattern

$$\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \text{---} \xrightarrow{x:\ell}$$

ℓ -labeled edge directed left/right/any-directed, assign to a variable

Patterns

$$\pi := \nu \mid \alpha \mid \pi\pi \mid \pi + \pi \mid \pi^{n..m} \mid \pi\langle\theta\rangle \quad 0 \leq n \leq m \leq \infty$$

Pattern calculus in a nutshell: PODS 23

Node pattern

$$\nu := (x : \ell)$$

match an ℓ -labeled node, assign to a variable

Both x and ℓ are optional

Edge pattern

$$\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \text{---}^{x:\ell}$$

ℓ -labeled edge directed left/right/any-directed, assign to a variable

Patterns

$$\pi := \nu \mid \alpha \mid \pi\pi \mid \pi + \pi \mid \pi^{n..m} \mid \pi\langle\theta\rangle \quad 0 \leq n \leq m \leq \infty$$

node

edge

concatenation

union

repetition
n-to-m times

selection with condition

Pattern calculus in a nutshell: PODS 23

Node pattern

$$\nu := (x : \ell)$$

match an ℓ -labeled node, assign to a variable

Both x and ℓ are optional

Edge pattern

$$\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \text{---}^{x:\ell}$$

ℓ -labeled edge directed left/right/any-directed, assign to a variable

Patterns

$$\pi := \nu \mid \alpha \mid \pi\pi \mid \pi + \pi \mid \pi^{n..m} \mid \pi\langle\theta\rangle \quad 0 \leq n \leq m \leq \infty$$

node

edge

concatenation

union

repetition
n-to-m times

selection with condition

Conditions

$$\theta := x.a = c \mid x.a = y.b \mid \theta \vee \theta \mid \theta \wedge \theta \mid \neg\theta$$

Pattern calculus in a nutshell: PODS 23

Node pattern

$$\nu := (x : \ell)$$

match an ℓ -labeled node, assign to a variable

Both x and ℓ are optional

Edge pattern

$$\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \text{---} \xrightarrow{x:\ell}$$

ℓ -labeled edge directed left/right/any-directed, assign to a variable

Patterns

$$\pi := \nu \mid \alpha \mid \pi\pi \mid \pi + \pi \mid \pi^{n..m} \mid \pi\langle\theta\rangle \quad 0 \leq n \leq m \leq \infty$$

node

edge

concatenation

union

repetition
n-to-m times

selection with condition

Conditions

$$\theta := x.a = c \mid x.a = y.b \mid \theta \vee \theta \mid \theta \wedge \theta \mid \neg\theta$$

key-value comparisons

Boolean combinations

Pattern calculus in a nutshell: PODS 23

Node pattern	$\nu := (x : \ell)$	match an ℓ -labeled node, assign to a variable	Both x and ℓ are optional		
Edge pattern	$\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \text{---} \xrightarrow{x:\ell}$	ℓ -labeled edge directed left/right/any-directed, assign to a variable			
Patterns	$\pi := \nu \mid \alpha \mid \pi\pi \mid \pi + \pi \mid \pi^{n..m} \mid \pi\langle\theta\rangle$	$0 \leq n \leq m \leq \infty$			
	node	edge concatenation	union	repetition n-to-m times	selection with condition
Conditions	$\theta := x.a = c \mid x.a = y.b \mid \theta \vee \theta \mid \theta \wedge \theta \mid \neg\theta$	key-value comparisons	Boolean combinations		
Queries	$Q := \sigma \pi \mid p = \sigma \pi \mid Q, Q$				

Pattern calculus in a nutshell: PODS 23

Node pattern	$\nu := (x : \ell)$	match an ℓ -labeled node, assign to a variable	Both x and ℓ are optional			
Edge pattern	$\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \text{---} \xrightarrow{x:\ell}$	ℓ -labeled edge directed left/right/any-directed, assign to a variable				
Patterns	$\pi := \nu \mid \alpha \mid \pi\pi \mid \pi + \pi \mid \pi^{n..m} \mid \pi\langle\theta\rangle$	$0 \leq n \leq m \leq \infty$				
	node	edge	concatenation	union	repetition n-to-m times	selection with condition
Conditions	$\theta := x.a = c \mid x.a = y.b \mid \theta \vee \theta \mid \theta \wedge \theta \mid \neg\theta$	key-value comparisons	Boolean combinations			
Queries	$Q := \sigma\pi \mid p = \sigma\pi \mid Q, Q$	ensure finitely many paths				

Pattern calculus in a nutshell: PODS 23

Node pattern	$\nu := (x : \ell)$	match an ℓ -labeled node, assign to a variable	Both x and ℓ are optional			
Edge pattern	$\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \text{---} \xrightarrow{x:\ell}$	ℓ -labeled edge directed left/right/any-directed, assign to a variable				
Patterns	$\pi := \nu \mid \alpha \mid \pi\pi \mid \pi + \pi \mid \pi^{n..m} \mid \pi\langle\theta\rangle$	$0 \leq n \leq m \leq \infty$				
	node	edge	concatenation	union	repetition n-to-m times	selection with condition
Conditions	$\theta := x.a = c \mid x.a = y.b \mid \theta \vee \theta \mid \theta \wedge \theta \mid \neg\theta$	key-value comparisons	Boolean combinations			
Queries	$Q := \sigma \pi \mid p = \sigma \pi \mid Q, Q$	ensure finitely many paths	name matched path			

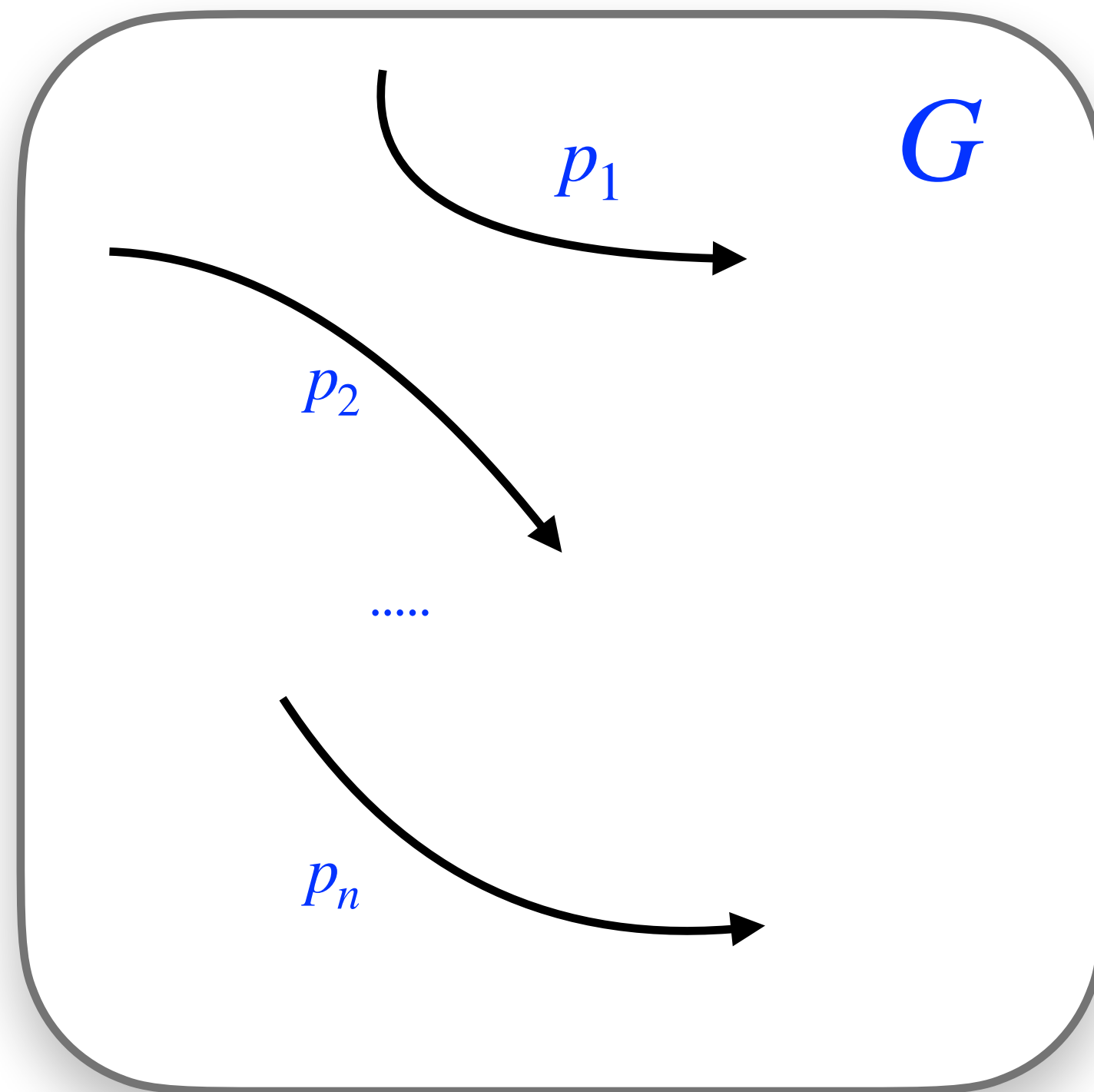
Pattern calculus in a nutshell: PODS 23

Node pattern	$\nu := (x : \ell)$	match an ℓ -labeled node, assign to a variable	Both x and ℓ are optional			
Edge pattern	$\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \text{---} \xrightarrow{x:\ell}$	ℓ -labeled edge directed left/right/any-directed, assign to a variable				
Patterns	$\pi := \nu \mid \alpha \mid \pi\pi \mid \pi + \pi \mid \pi^{n..m} \mid \pi\langle\theta\rangle$	$0 \leq n \leq m \leq \infty$				
	node	edge	concatenation	union	repetition n-to-m times	selection with condition
Conditions	$\theta := x.a = c \mid x.a = y.b \mid \theta \vee \theta \mid \theta \wedge \theta \mid \neg\theta$	key-value comparisons	Boolean combinations			
Queries	$Q := \sigma \pi \mid p = \sigma \pi \mid Q, Q$	ensure finitely many paths	name matched path	join		

Semantics — Idea

$Q = \pi_1, \pi_2, \dots, \pi_n$ with variables x_1, x_2, \dots, x_m

MATCH result: a tuple of paths + a table



x1	x2	xm

GQL and SQL/PGQ only keep the table

What's in the table?

- Graph elements
 - Nodes
 - Edges
- Paths (when named: $x = \pi$)
- **Lists** of graph elements

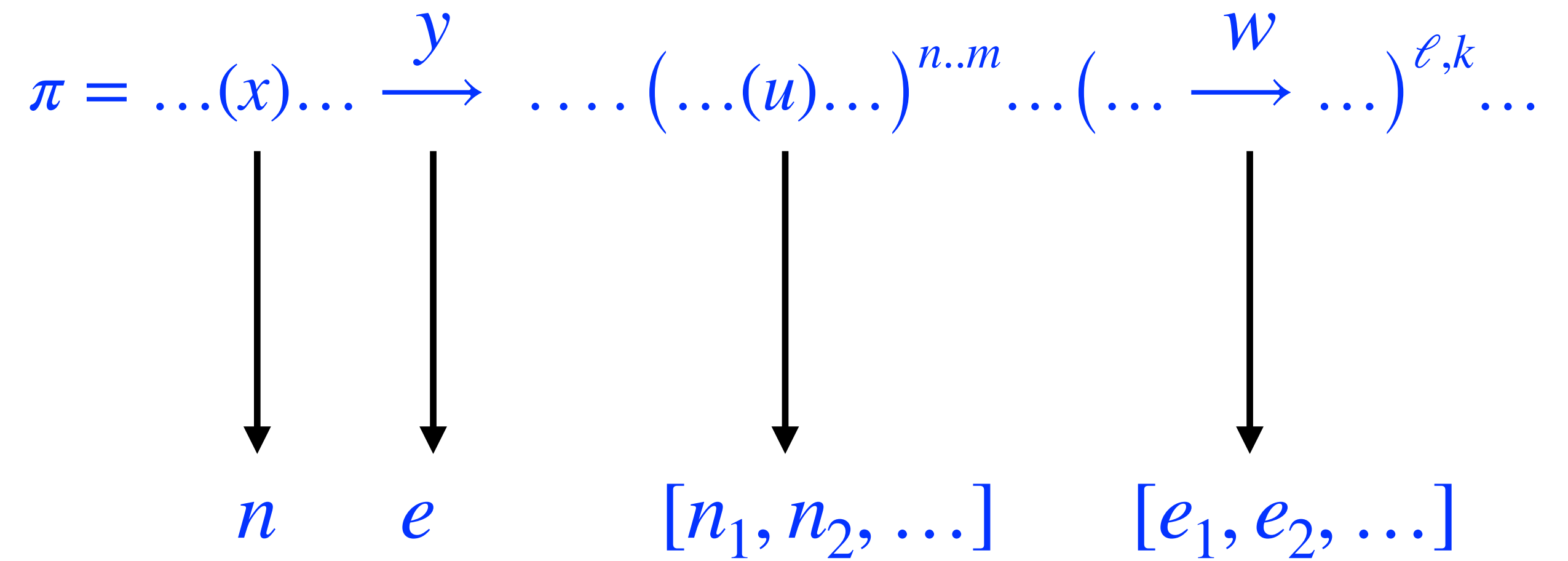
What's in the table?

- Graph elements

- Nodes
- Edges

- Paths (when named: $x = \pi$)

- **Lists** of graph elements



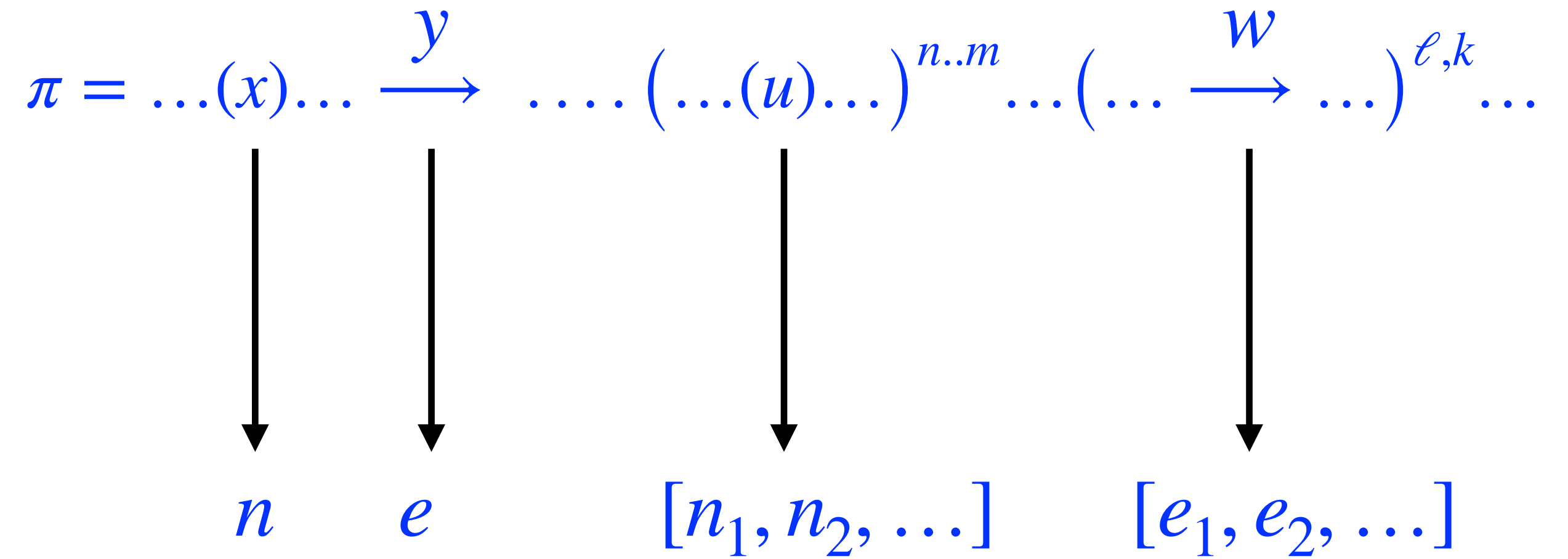
What's in the table?

- Graph elements

- Nodes
- Edges

- Paths (when named: $x = \pi$)

- Lists** of graph elements



Tables may have nulls: $(x) + \xrightarrow{y}$

x	y
n	NULL
NULL	e

What we have done in the PODS paper

- **Formal semantics** for well-typed expressions
- **Type system**: when a variable is assigned:
 - a graph element, or a list, or could be assigned NULL
- **Complexity**
 - **PSPACE** data complexity of enumeration
 - Not surprising: there are many paths
 - Note: Cypher is **NP-hard**. Things may work in practice, but not in theory!
- **Expressivity**
 - Subsumes CRPQs, inverses, unions, nested regular expressions, regular queries

Relational Querying in GQL (streamlined)

Basic Operations on Tables

- **RETURN** (projection)
- **LET** (add columns)
- **FILTER** (selection)
- **FOR** (unnest for lists)
- another **MATCH** (join with the current working table)

Union, Intersection, Difference

If Q_1 and Q_2 are GQL queries, then so are

- Q_1 **UNION** Q_2
- Q_1 **INTERSECT** Q_2
- Q_1 **EXCEPT** Q_2
- Q_1 **OTHERWISE** Q_2

Multiple Graphs

USE G1

MATCH π_1

WHERE θ_1

RETURN L_1

NEXT USE G2

MATCH π_2

WHERE θ_2

RETURN L_2

.....

NEXT USE Gn

MATCH π_n

WHERE θ_n

RETURN L_n

ICDT '23 Paper: “A Researcher’s Digest of GQL”

Idea:

A syntax closer to actual GQL

But still OK for academics to use for research

Syntax: PATTERNS

PATH PATTERN For $x \in \text{Vars}$, $\ell \in \mathcal{L}$, $0 \leq n \leq m \in \mathbb{N}$:

(descriptor)	$\delta := x : \ell$	WHERE θ	x , $: \ell$, and WHERE θ are optional	
(path pattern)	$\pi := (\delta)$		(node pattern)	
	$-\delta->$	$<-\delta-$	$\sim\delta\sim$	(edge pattern)
	$\pi \pi$			(concatenation)
	$\pi \pi$			(union)
	π	WHERE θ		(conditioning)
	$\pi\{n, m\}$			(bounded repetition)
	$\pi\{n, \}$			(unbounded repetition)

EXPRESSION and CONDITION For $x \in \text{Vars}$, $\ell \in \mathcal{L}$, $a \in \mathcal{K}$, $c \in \text{Const}$:

(expression)	$\chi := x$	$x.a$	c
(condition)	$\theta := \chi = \chi$	$\chi < \chi$	χ IS NULL
	$x : \ell$	EXISTS $\{Q\}$	
	θ OR θ	θ AND θ	NOT θ

GRAPH PATTERN For $x \in \text{Vars}$:

(path mode)	$\mu :=$	(ALL ANY)	[SHORTEST]	[TRAIL ACYCLIC]
(graph pattern)	$\Pi := \mu$	[$x =$]	π	Π, Π

Syntax: QUERIES

CLAUSE and **QUERY** For $k \geq 0$, $\ell \geq 1$, and $x, y, x_1, \dots, x_k \in \text{Vars}$, and $G \in \mathbb{G}$:

(clause) $C :=$ **MATCH** Π
 | **LET** $x = \chi$
 | **FOR** x **IN** y
 | **FILTER** θ

(linear query) $L :=$ **USE** G L
 | C L
 | **RETURN** χ_1 **AS** x_1, \dots, χ_k **AS** x_k

(query) $Q :=$ L
 | **USE** G $\{Q_1$ **THEN** Q_2 \dots **THEN** $Q_\ell\}$
 | Q **INTERSECT** Q | Q **UNION** Q | Q **EXCEPT** Q

ICDT Paper: Semantics

1:12 A Researcher's Digest of GQL

$$\begin{aligned} \llbracket [-\square \rightarrow] \rrbracket_G &= \{ (\text{path}(\text{src}(e), e, \text{tgt}(e)), ()) \mid e \in E_d^G \} \\ \llbracket [-[x] \rightarrow] \rrbracket_G &= \{ (\text{path}(\text{src}(e), e, \text{tgt}(e)), (x \mapsto e)) \mid e \in E_d^G \} \\ \llbracket [-[:\ell] \rightarrow] \rrbracket_G &= \left\{ (\text{path}(\text{src}(e), e, \text{tgt}(e)), ()) \mid e \in E_d^G, \ell \in \text{lab}^G(e) \right\} \end{aligned}$$

Other cases of the forward edge patterns are treated by moving the label and conditions outside of the edge pattern, just as for node patterns. Backward edge patterns and undirected edge patterns are treated similarly, with the base cases given below.

$$\begin{aligned} \llbracket \langle -\square - \rangle \rrbracket_G &= \{ (\text{path}(\text{tgt}(e), e, \text{src}(e)), ()) \mid e \in E_d^G \} \\ \llbracket [-\square -] \rrbracket_G &= \left\{ (\text{path}(u_1, e, u_2), ()), (\text{path}(u_2, e, u_1), ()) \mid \begin{array}{l} e \in E_u^G \\ \{u_1, u_2\} = \text{endpoints}^G(e) \end{array} \right\} \end{aligned}$$

Semantics of Concatenation, Union, and Conditioning

$$\llbracket \pi_1 \pi_2 \rrbracket_G \left\{ \begin{array}{l} (p_1 \cdot p_2, \mu_1 \bowtie \mu_2) \\ p_1 \text{ and } p_2 \text{ concatenate} \\ \mu_1 \sim \mu_2 \end{array} \right\} \begin{array}{l} (p_i, \mu_i) \in \llbracket \pi_i \rrbracket_G \text{ for } i = 1, 2 \\ \end{array}$$

Note that since $\pi_1 \pi_2$ is assumed to be well-formed, all variables shared by π_1 and π_2 are singleton variables (Condition 2 in Section 3). In other words, implicit joins over group and optional variables are disallowed; the same remark will also apply for the semantics of joins.

► Remark 9. Consider the pattern

$$(x) \ (-[:\text{Transfer}] \rightarrow () -[:\text{Transfer}] \rightarrow (x)) \{1, \}$$

This pattern is disallowed in GQL because the leftmost x is a singleton variable, whereas the rightmost x is a group variable. In GQL philosophy, the leftmost x will be bound to a node and the rightmost x will be bound to a list of nodes, which is a type mismatch.

$$\llbracket \pi_1 \mid \pi_2 \rrbracket_G = \{ (p, \mu \cup \mu') \mid (p, \mu) \in \llbracket \pi_1 \rrbracket_G \cup \llbracket \pi_2 \rrbracket_G \}$$

where μ' maps every variable in $\text{var}(\pi_1 \mid \pi_2) \setminus \text{Dom}(\mu)$ to null. (Recall that var maps a pattern to the set of variables appearing in it.)

$$\llbracket \pi \text{ WHERE } \theta \rrbracket_G = \{ (p, \mu) \in \llbracket \pi \rrbracket_G \mid \llbracket \theta \rrbracket_G^\mu = \text{true} \}$$

Semantics of Repetition

$$\begin{aligned} \llbracket \pi\{n, m\} \rrbracket_G &= \bigcup_{i=n}^m \llbracket \pi \rrbracket_G^i \\ \llbracket \pi\{n, \}\rrbracket_G &= \bigcup_{i=n}^{\infty} \llbracket \pi \rrbracket_G^i \end{aligned}$$

Above, for a pattern π and a natural number $i \geq 0$, we use $\llbracket \pi \rrbracket_G^i$ to denote the i -th power of $\llbracket \pi \rrbracket_G$, which we define as

$$\llbracket \pi \rrbracket_G^0 = \{ (\text{path}(u), \mu) \mid u \text{ is a node in } G \}$$

where μ binds each variable in $\text{Dom}(\text{sch}(\pi))$ to $\text{list}()$, that is, the empty-list value; and

$$\forall i > 0 \quad \llbracket \pi \rrbracket_G^i = \left\{ (p_1 \cdot \dots \cdot p_i, \mu') \mid \begin{array}{l} (p_1, \mu_1), \dots, (p_i, \mu_i) \in \llbracket \pi \rrbracket_G \\ p_1, \dots, p_i \text{ concatenate} \end{array} \right\}$$

where μ' binds each variable in $\text{Dom}(\text{sch}(\pi))$ to $\text{list}(\mu_1(x), \dots, \mu_i(x))$. Recall that sch is defined in Section 3.

N. Francis et al.

1:13

► Remark 10. Since $\pi\{n, \}$ is assumed to be well-formed, it holds $\|\pi\|_{\min} \geq 1$. A simple induction then yields that each p_i in the definition above has positive length. A second induction then yields that, given a path p , there are finitely many assignments μ such that $(p, \mu) \in \llbracket \pi\{n, m\} \rrbracket_G$. This fact is crucial to have a finite output in the end.

For instance, consider a graph with a single node u and no edges, and the pattern $(\mathbf{a})\{0, \}$ which is not well-formed (the minimal path length of $()$ is 0). For every i , the set $\llbracket (\mathbf{a}) \rrbracket_G^i$ contains $(\text{path}(u), \mu_i)$ where $\mu_i = (a \mapsto \underbrace{\text{list}(u, \dots, u)}_{i \text{ times}})$; hence the union in the definition of

$\llbracket \pi\{n, \} \rrbracket_G$ above would not only yield an infinite number of elements, but all of them would be associated to the same path. As a result a graph pattern such as **ALL SHORTEST** $(\mathbf{a})\{0, \}$ would have infinitely many results.

4.3 Semantics of Graph Patterns

We now define the semantics of graph patterns. We first fully define atomic graph patterns and then define their joins.

$$\llbracket x = \pi \rrbracket_G = \{ (p, \mu \cup \{x \mapsto p\}) \mid (p, \mu) \in \llbracket \pi \rrbracket_G \}$$

In the following we denote by $\tilde{\pi}$ a graph pattern that never uses the “,” operator, hence it is of the form $\mu x = \pi$, where μ is a path mode, x is a variable, π is a path pattern, and “ $x =$ ” is optional.

$$\llbracket \text{TRAIL } \pi \rrbracket_G = \{ (p, \mu) \in \llbracket \pi \rrbracket_G \mid \text{no edge occurs more than once in } p \}$$

$$\llbracket \text{ACYCLIC } \pi \rrbracket_G = \{ (p, \mu) \in \llbracket \pi \rrbracket_G \mid \text{no node occurs more than once in } p \}$$

$$\llbracket \text{SHORTEST } \tilde{\pi} \rrbracket_G = \left\{ (p, \mu) \in \llbracket \tilde{\pi} \rrbracket_G \mid \text{len}(p) = \min \left\{ \text{len}(p') \mid \begin{array}{l} (p', \mu') \in \llbracket \tilde{\pi} \rrbracket_G \\ \text{src}(p') = \text{src}(p) \\ \text{tgt}(p') = \text{tgt}(p) \end{array} \right\} \right\}$$

$$\llbracket \text{ALL } \tilde{\pi} \rrbracket_G = \llbracket \tilde{\pi} \rrbracket_G$$

$$\llbracket \text{ANY } \tilde{\pi} \rrbracket_G = \bigcup_{(s,t) \in X} \{ \text{any}(\{ (p, \mu) \mid (p, \mu) \in \llbracket \tilde{\pi} \rrbracket_G, \text{endpoints}(p) = (s, t) \}) \}$$

where $X = \{ (\text{src}(p), \text{tgt}(p)) \mid (p, \mu) \in \llbracket \tilde{\pi} \rrbracket_G \}$ and any is a procedure that arbitrarily returns one element from a set; any need not be deterministic.

$$\llbracket \Pi_1, \Pi_2 \rrbracket_G = \{ (\bar{p}_1 \times \bar{p}_2, \mu_1 \bowtie \mu_2) \mid (\bar{p}_i, \mu_i) \in \llbracket \Pi_i \rrbracket_G \text{ for } i = 1, 2 \text{ and } \mu_1 \sim \mu_2 \}$$

Here, $\bar{p}_1 = (p_1^1, p_1^2, \dots, p_1^k)$ and $\bar{p}_2 = (p_2^1, p_2^2, \dots, p_2^l)$ are tuples of paths, and $\bar{p}_1 \times \bar{p}_2$ stands for $(p_1^1, p_1^2, \dots, p_1^k, p_2^1, p_2^2, \dots, p_2^l)$. Just as it is the case of concatenation, since Π_1, Π_2 is well-formed, implicit joins can occur over singleton variables only.

4.4 Semantics of Conditions and Expressions

The semantics $\llbracket \chi \rrbracket_G^\mu$ of an expression χ is an element in \mathbb{V} that is computed with respect to a binding μ and a graph G . Intuitively, variables in χ are evaluated with μ and we use G to access the properties of an element. It is formally defined as follows.

$$\begin{aligned} \llbracket c \rrbracket_G^\mu &= c && \text{for } c \in \text{Const} \\ \llbracket x \rrbracket_G^\mu &= \mu(x) && \text{for } x \in \text{Dom}(\mu) \\ \llbracket x.a \rrbracket_G^\mu &= \begin{cases} \text{prop}^G(\mu(x), a) & \text{if } (\mu(x), a) \in \text{Dom}(\text{prop}^G) \\ \text{null} & \text{else if } \mu(x) \in (\mathcal{N} \cup \mathcal{E}_d \cup \mathcal{E}_u) \end{cases} && \text{for } x \in \text{Dom}(\mu), a \in \mathcal{K} \end{aligned}$$

ICDT Paper: Semantics

1:14 A Researcher's Digest of GQL

► **Remark 11.** Recall that different graphs may share nodes and edges. Hence the condition $(\mu(x), a) \in \text{Dom}(\text{prop}^G)$, above, does imply that $\mu(x)$ is a node or an edge in G , but does **not** imply that it was matched in G .

The semantics $\llbracket \theta \rrbracket_G^\mu$ of a condition θ is an element in $\{\text{true}, \text{false}, \text{null}\}$ that is evaluated with respect to a binding μ and a graph G , and is defined as follows:

$$\begin{aligned} \llbracket \chi_1 = \chi_2 \rrbracket_G^\mu &= \begin{cases} \text{null} & \text{if } \llbracket \chi_1 \rrbracket_G^\mu = \text{null} \text{ or } \llbracket \chi_2 \rrbracket_G^\mu = \text{null} \\ \text{true} & \text{if } \llbracket \chi_1 \rrbracket_G^\mu = \llbracket \chi_2 \rrbracket_G^\mu \neq \text{null} \\ \text{false} & \text{otherwise} \end{cases} \\ \llbracket \chi_1 < \chi_2 \rrbracket_G^\mu &= \begin{cases} \text{null} & \text{if } \llbracket \chi_1 \rrbracket_G^\mu = \text{null} \text{ or } \llbracket \chi_2 \rrbracket_G^\mu = \text{null} \\ \text{true} & \text{else if } \llbracket \chi_1 \rrbracket_G^\mu < \llbracket \chi_2 \rrbracket_G^\mu \\ \text{false} & \text{otherwise} \end{cases} \\ \llbracket \chi \text{ IS NULL} \rrbracket_G^\mu &= \begin{cases} \text{true} & \text{if } \llbracket \chi \rrbracket_G^\mu = \text{null} \\ \text{false} & \text{otherwise} \end{cases} \\ \llbracket \chi : \ell \rrbracket_G^\mu &= \begin{cases} \text{true} & \text{if } \llbracket \chi \rrbracket_G^\mu \in N^G \cup E_u^G \cup E_d^G \text{ and } \ell \in \text{lab}^G(\llbracket \chi \rrbracket_G^\mu) \\ \text{false} & \text{else if } \llbracket \chi \rrbracket_G^\mu \in \mathcal{N} \cup \mathcal{E}_d \cup \mathcal{E}_u \end{cases} \\ \llbracket \theta_1 \text{ AND } \theta_2 \rrbracket_G^\mu &= \llbracket \theta_1 \rrbracket_G^\mu \wedge \llbracket \theta_2 \rrbracket_G^\mu \quad (*) \\ \llbracket \theta_1 \text{ OR } \theta_2 \rrbracket_G^\mu &= \llbracket \theta_1 \rrbracket_G^\mu \vee \llbracket \theta_2 \rrbracket_G^\mu \quad (*) \\ \llbracket \text{NOT } \theta \rrbracket_G^\mu &= \neg \llbracket \theta \rrbracket_G^\mu \quad (*) \end{aligned}$$

(*) Operators \wedge , \vee , and \neg are defined as in SQL three-valued logic, e.g. $\text{null} \vee \text{true} = \text{true}$ while $\text{null} \wedge \text{true} = \text{null}$.

$$\llbracket \text{EXISTS } \{Q\} \rrbracket_G^\mu = \begin{cases} \text{true} & \text{if } \llbracket Q \rrbracket_G(\{\mu\}) \text{ is not empty} \\ \text{false} & \text{otherwise} \end{cases}$$

4.5 Semantics of Queries

Clauses and queries are interpreted as functions that operate on tables. These tables are our abstraction of GQL's working tables.

► **Definition 12.** A table T is a set of bindings that have the same domains, referred to as $\text{Dom}(T)$.

Note that tables do not have schemas: two different bindings in a table might associate a variable to values of incompatible types.

Semantics of Clauses

The semantics $\llbracket C \rrbracket_G$ of a clause C is a function that maps tables into tables, and is parametrized by a graph G . Patterns, conditions and expression in a clause are evaluated with respect to that G .

$$\llbracket \text{MATCH } \Pi \rrbracket_G(T) = \bigcup_{\mu \in T} \{\mu \bowtie \mu' \mid (p, \mu') \in \llbracket \Pi \rrbracket_G, \mu \sim \mu'\}$$

Note that if Π uses a variable that already occurs in $\text{Dom}(T)$, a join is performed. Unlike in the case of path patterns and graph patterns, this join can involve variables bound to lists or paths. While this is not problematic mathematically, it might be disallowed in future iterations of GQL.

N. Francis et al.

1:15

If $x \notin \text{Dom}(T)$, then

$$\begin{aligned} \llbracket \text{LET } x = \chi \rrbracket_G(T) &= \bigcup_{\mu \in T} \{\mu \bowtie (x \mapsto \llbracket \chi \rrbracket_G^\mu)\} \\ \llbracket \text{FILTER } \theta \rrbracket_G(T) &= \bigcup_{\mu \in T} \{\mu \mid \llbracket \theta \rrbracket_G^\mu = \text{true}\}. \end{aligned}$$

If $x \notin \text{Dom}(T)$ and, for every $\mu \in T$, $\mu(y)$ is a list or null ,³ then

$$\llbracket \text{FOR } x \text{ IN } y \rrbracket_G(T) = \bigcup_{\mu \in T} \{\mu \bowtie (x \mapsto v) \mid v \in \mu(y)\}.$$

Semantics of Linear Queries

$$\begin{aligned} \llbracket \text{USE } G' \text{ L} \rrbracket_G(T) &= \llbracket \text{L} \rrbracket_{G'}(T) \\ \llbracket C \text{ L} \rrbracket_G(T) &= \llbracket \text{L} \rrbracket_G(\llbracket C \rrbracket_G(T)) \\ \llbracket \text{RETURN } \chi_1 \text{ AS } x_1, \dots, \chi_\ell \text{ AS } x_\ell \rrbracket_G(T) &= \bigcup_{\mu \in T} \{(x_1 \mapsto \llbracket \chi_1 \rrbracket_G^\mu, \dots, x_\ell \mapsto \llbracket \chi_\ell \rrbracket_G^\mu)\} \end{aligned}$$

Semantics of Queries

The *output of a query* Q is defined as

$$\text{Output}(Q) = \llbracket Q \rrbracket_G(\{\{\}\}),$$

where $\{\{\}\}$ is the unit table that consists of the empty binding, and G is the default graph in D . We define the semantics of queries recursively as follows.

$$\llbracket \text{USE } G' \{Q_1 \text{ THEN } Q_2 \dots \text{ THEN } Q_k\} \rrbracket_G(T) = \llbracket Q_k \rrbracket_{G'} \circ \dots \circ \llbracket Q_1 \rrbracket_{G'}(T)$$

If $\text{Dom}(\llbracket Q_1 \rrbracket_G(T)) = \text{Dom}(\llbracket Q_2 \rrbracket_G(T))$, then we let

$$\begin{aligned} \llbracket Q_1 \text{ INTERSECT } Q_2 \rrbracket_G(T) &= \llbracket Q_1 \rrbracket_G(T) \cap \llbracket Q_2 \rrbracket_G(T) \\ \llbracket Q_1 \text{ UNION } Q_2 \rrbracket_G(T) &= \llbracket Q_1 \rrbracket_G(T) \cup \llbracket Q_2 \rrbracket_G(T) \\ \llbracket Q_1 \text{ EXCEPT } Q_2 \rrbracket_G(T) &= \llbracket Q_1 \rrbracket_G(T) \setminus \llbracket Q_2 \rrbracket_G(T) \end{aligned}$$

5 A Few Known Discrepancies with the GQL Standard

In pursuing the goal of introducing the key features of GQL to the research community, we inevitably had to make decisions that resulted in discrepancies between our presentation and the 500+ pages of the forthcoming Standard. In this section, we discuss a non-exhaustive list of differences between the actual GQL Standard and our digest. To start with, in all our formal development we assumed that queries are given by their syntax trees, which result from parsing them. Hence we completely omitted such parsing-related aspects as parentheses, operator precedence etc. Also we note that many GQL features, even those described here, are optional, and not every implementation is obliged to have them all.

³ Note that null is treated just as $\text{list}()$

What Do the Papers Omit?

What Do the Papers Omit?

Bag semantics

Our semantics is correct up to multiplicities

What Do the Papers Omit?

Bag semantics

Our semantics is correct up to multiplicities

Aggregation

- There is vertical aggregation as in SQL
- There is also horizontal aggregation along paths
 - e.g. `SUM(e.weight) < 100`

What Do the Papers Omit?

Bag semantics

Our semantics is correct up to multiplicities

Aggregation

- There is vertical aggregation as in SQL
- There is also horizontal aggregation along paths
 - e.g. `SUM(e.weight) < 100`

Procedure calls

- inlined: `CALL { ... }`
- named: `CALL <proc-name> (<params>)`

What Do the Papers Omit?

Bag semantics

Our semantics is correct up to multiplicities

Aggregation

- There is vertical aggregation as in SQL
- There is also horizontal aggregation along paths
 - e.g. `SUM(e.weight) < 100`

Procedure calls

- inlined: `CALL { ... }`
- named: `CALL <proc-name> (<params>)`

Updates

insert, set, delete

What Do the Papers Omit?

Bag semantics

Our semantics is correct up to multiplicities

Aggregation

- There is vertical aggregation as in SQL
- There is also horizontal aggregation along paths
 - e.g. `SUM(e.weight) < 100`

Procedure calls

- inlined: `CALL { ... }`
- named: `CALL <proc-name> (<params>)`

Updates

insert, set, delete

Extra path modes

e.g. `TRAIL` along several paths π_1, \dots, π_n

What Do the Papers Omit?

Bag semantics

Our semantics is correct up to multiplicities

Aggregation

- There is vertical aggregation as in SQL
- There is also horizontal aggregation along paths
 - e.g. `SUM(e.weight) < 100`

Procedure calls

- inlined: `CALL { ... }`
- named: `CALL <proc-name> (<params>)`

Updates

insert, set, delete

Extra path modes

e.g. `TRAIL` along several paths π_1, \dots, π_n

Catalog operations

Data types and value expressions

Predicates (including handling nulls)

Open Questions

Open Questions

The first version of the standard will be published in 2024

It all makes sense if you take a quick look....

Open Questions



The first version of the standard will be published in 2024
It all makes sense if you take a quick look....

Open Questions



The first version of the standard will be published in 2024
It all makes sense if you take a quick look....

But it has not been scrutinised outside WG3 and LDBC
We need to do research on GQL **now!**
Our work gives you the platform

Open Questions



The first version of the standard will be published in 2024
It all makes sense if you take a quick look....

But it has not been scrutinised outside WG3 and LDBC
We need to do research on GQL **now!**
Our work gives you the platform

Questions:

Is it good? Is it usable? Or the best simply
because everything else is average?

Open Questions

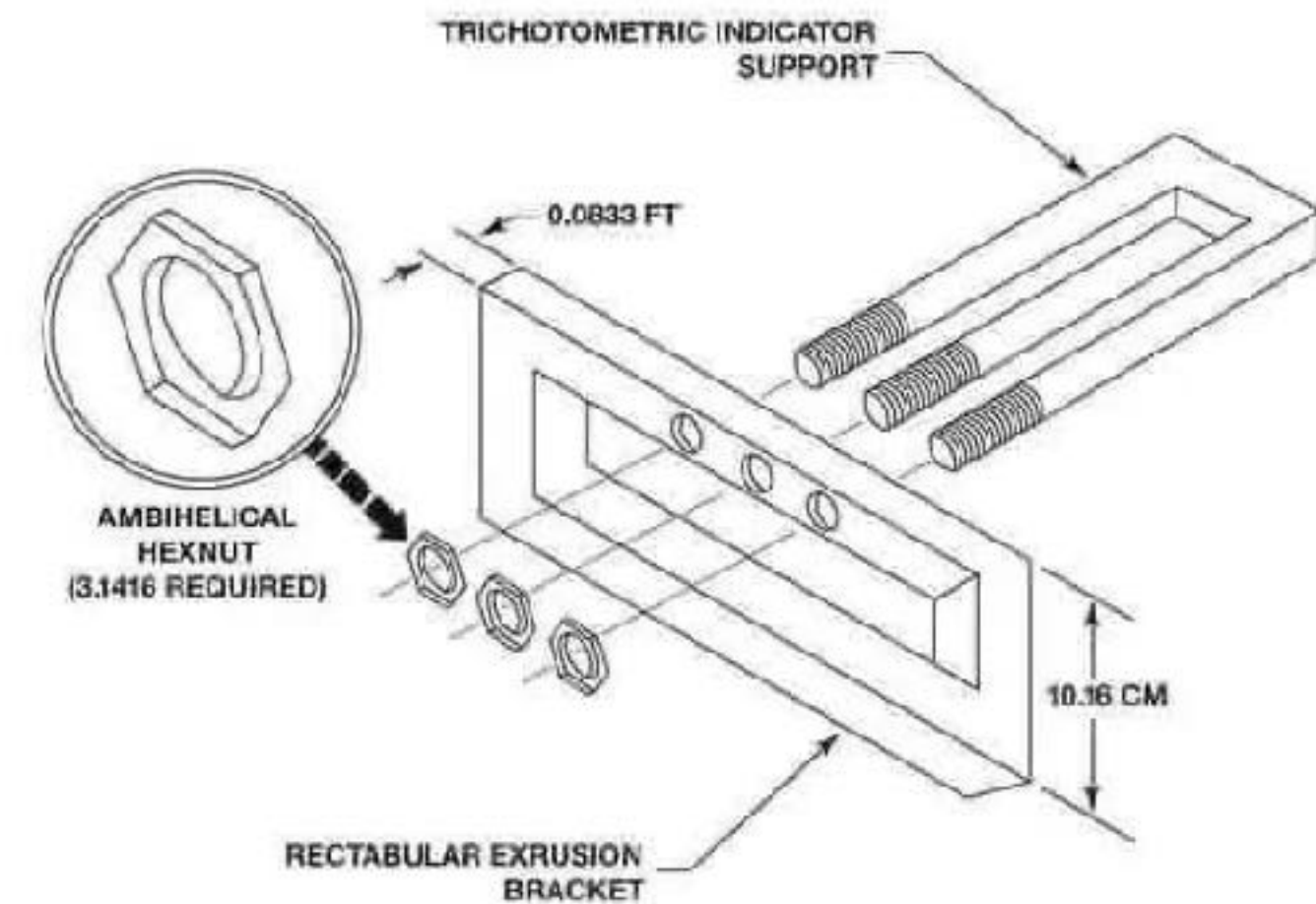


The first version of the standard will be published in 2024
It all makes sense if you take a quick look....

But it has not been scrutinised outside WG3 and LDBC
We need to do research on GQL **now!**
Our work gives you the platform

Questions:

Is it good? Is it usable? Or the best simply because everything else is average?



Directions for Research: Theoretician's Comfort zone

Expressive Power and Complexity

- Clean Language Fragments and Extensions
 - Think of First-Order Logic and everything we know about its power, complexity, and that of CQs, UCQs, Datalog, etc etc
- DB theory folks are really good at this

Directions for Research: Theoretician's Comfort zone

Expressive Power and Complexity

- Clean Language Fragments and Extensions
 - Think of First-Order Logic and everything we know about its power, complexity, and that of CQs, UCQs, Datalog, etc etc
- DB theory folks are really good at this

Query processing and Optimization

- Containment, Equivalence, ...
 - GQL goes much beyond CRPQs
- Practical algorithms, data structures

Directions for Research: Extra features

Directions for Research: Extra features



Directions for Research: Extra features



Updates

- Updating graphs is not a trivial matter
- Many alternative semantics need to be explored (even the case of Cypher was highly problematic)

Directions for Research: Extra features



Updates

- Updating graphs is not a trivial matter
- Many alternative semantics need to be explored (even the case of Cypher was highly problematic)

Design analysis: alternatives, suggestions, holes

- Many examples: e.g., dealing with group variables. Are the current restrictions (e.g., no comparisons) necessary?
- Can variables be used non-locally?
 - E.g. `MATCH (x) (-[y:a]-> WHERE x.k+y.k=10)* (z)`
 - Implications for complexity?

Directions for Research: What is missing



Directions for Research: What is missing

- Graph-to-Graph Queries
- Like Cypher, GQL is an engine for turning graphs into relations
- This has many limitations: how to do views? subqueries?
- Need design principles for graph-to-graph languages.



Directions for Research: What is missing

- Graph-to-Graph Queries
- Like Cypher, GQL is an engine for **turning graphs into relations**
- This has many limitations: how to do views? subqueries?
- Need design principles for graph-to-graph languages.



Schemas and Constraints

- Taken for granted for relational databases
- Much less work on property graphs but it's coming
- **PG-KEYS** (SIGMOD'21), **PG-SCHEMA** (SIGMOD'23)

Final Thoughts

Final Thoughts

After 6 years of work, GQL will become an ISO Standard

Final Thoughts

After 6 years of work, GQL will become an ISO Standard

There is a reason to be happy but:

- *When you jump for joy, beware that no one moves ground under your feet*
- NDL was a standard too but lost to relational

Final Thoughts

After 6 years of work, GQL will become an ISO Standard

There is a reason to be happy but:

- *When you jump for joy, beware that no one moves ground under your feet*
- NDL was a standard too but lost to relational

Relational querying of graphs is very active too

- SQL/PGQ in relational systems
- Relational languages for KG, e.g. **RelationalAI**
- Native Graph Querying vs Relational Graph Querying will be playing out in the next N years
 - [The Register, 6 March 2023](#): *“The Great Graph Debate: Revolutionary concept in databases or niche curiosity”*

Final Thoughts

After 6 years of work, GQL will become an ISO Standard

There is a reason to be happy but:

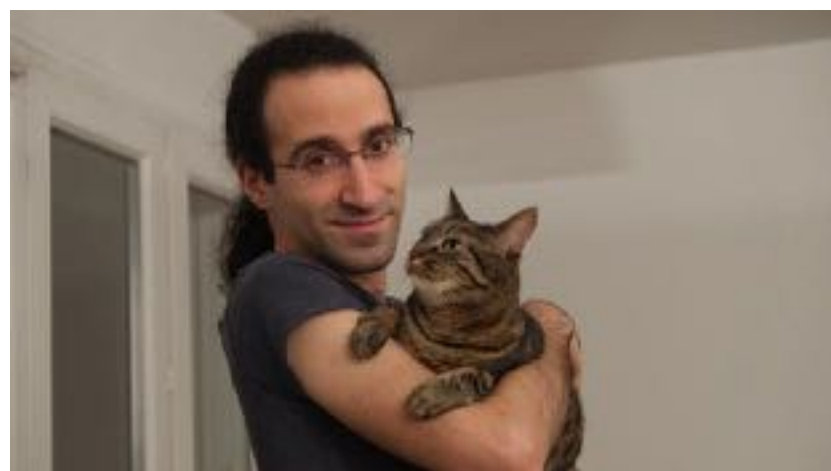
- *When you jump for joy, beware that no one moves ground under your feet*
- NDL was a standard too but lost to relational

Relational querying of graphs is very active too

- SQL/PGQ in relational systems
- Relational languages for KG, e.g. **RelationalAI**
- Native Graph Querying vs Relational Graph Querying will be playing out in the next N years
 - [The Register, 6 March 2023](#): *“The Great Graph Debate: Revolutionary concept in databases or niche curiosity”*

Our community has a lot to offer in this debate — on both fronts

It takes a (cat) team



Nadime Francis



Paolo Guagliardo



Victor Marsault



Filip Murlak



Alexandra Rogova



Amélie Gheerbrant



Leonid Libkin



Wim Martens



Liat Peterfreund



Domagoj Vrgoč