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>
```

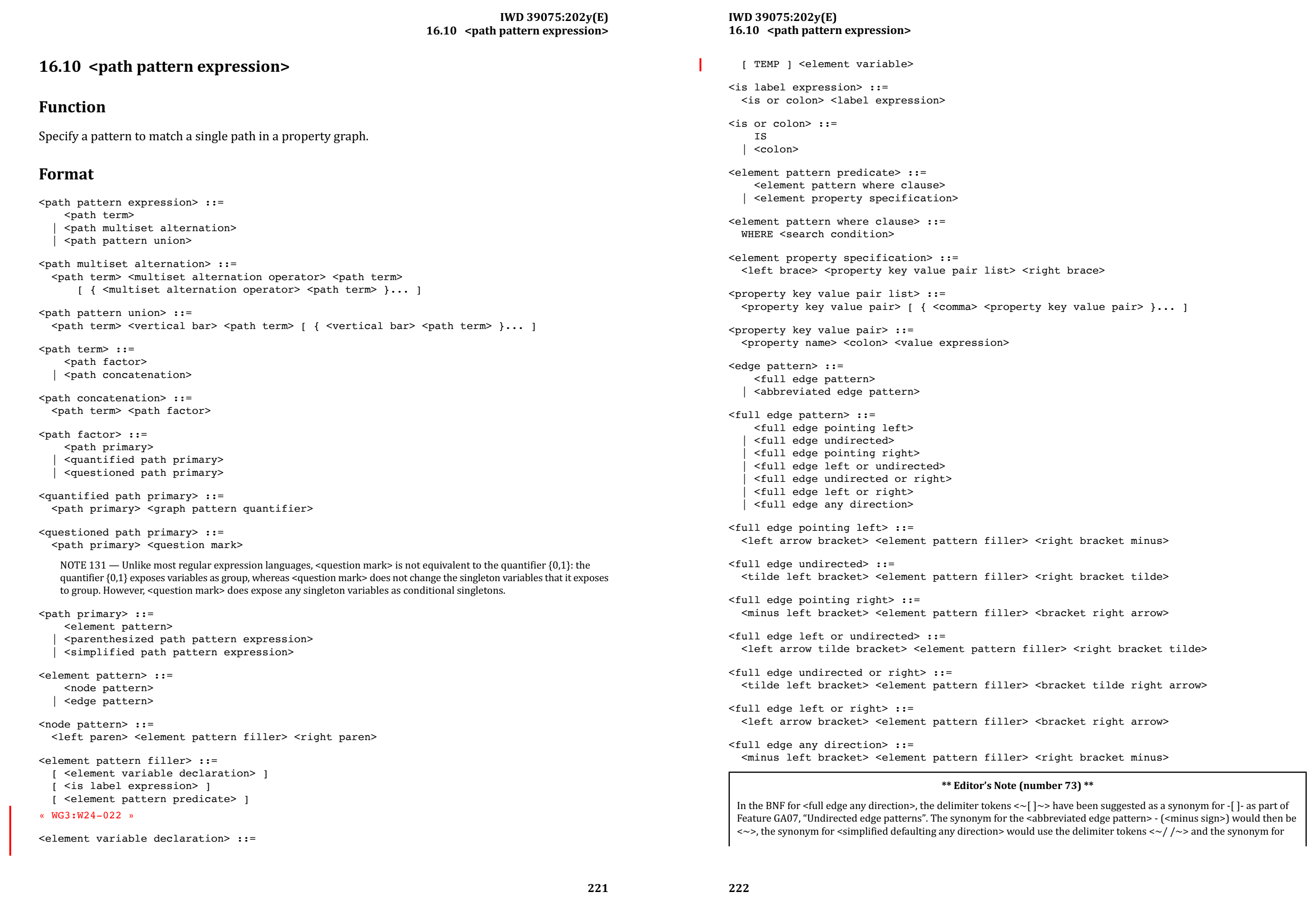
NOTE 131 — Unlike most regular expression languages, <question mark> is not equivalent to the quantifier {0,1}: the quantifier {0,1} exposes variables as group, whereas <question mark> does not change the singleton variables that it exposes to group. However, <question mark> does expose any singleton 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> ]
« WG3:W24-022 »

<element variable declaration> ::=
```

221

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> ]
« WG3:W24-022 »

<element variable declaration> ::=
```

221

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 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 GA07, "Undirected edge patterns". 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

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> ]
« WG3:W24-022 »

<element variable declaration> ::=
```



```
  [ 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 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 GA07, "Undirected edge patterns". 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

<simplified override any direction> would use the tokens <~ and > surrounding a label as originally proposed in WG3:MMX-060. These synonyms might be considered to make the table of edge patterns more harmonious and internally consistent. See Language Opportunity **GQL-212**.

```
<abbreviated edge pattern> ::=
    <left arrow>
  | <tilde>
  | <right arrow>
  | <left arrow tilde>
  | <tilde right arrow>
  | <left minus right>
  | <minus sign>
« WG3:W24-038 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 <token>s: <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 <token>s: <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 <token> from *RIGHTMINUS* followed by a <token> from *LEFTMINUS* without a <separator> between them.

   NOTE 134 — Otherwise, the concatenation of the two tokens would include the sequence of two <minus sign>s, 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>

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> ]
« WG3:W24-022 »

<element variable declaration> ::=
```

221

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 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 GA07, "Undirected edge patterns". 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:MMX-060. These synonyms might be considered to make the table of edge patterns more harmonious and internally consistent. See Language Opportunity **GQL-212**.

```
<abbreviated edge pattern> ::=
    <left arrow>
  | <tilde>
  | <right arrow>
  | <left arrow tilde>
  | <tilde right arrow>
  | <left minus right>
  | <minus sign>
« WG3:W24-038 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 <token>s: <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 <token>s: <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 <token> from *RIGHTMINUS* followed by a <token> from *LEFTMINUS* without a <separator> between them.

   NOTE 134 — Otherwise, the concatenation of the two tokens would include the sequence of two <minus sign>s, 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>

   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. WG3:W01-014 contained a discussion of a way to support aggregates at different depths of aggregation if there are nested quantifiers. See Language Opportunity **GQL-036**.

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 $OPMA_1$, ..., $OPMA_{NOPMA}$ be an enumeration of the operands of *PMA*.

   c) Any <subpath variable>s declared by <subpath variable declaration>s simply contained in the multiset alternation operands of *PMA* shall be mutually distinct.

   d) Let $SOPMA_1$, ..., $SOPMA_{NOPMA}$ 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) $\leq i \leq NOPMA$.

      Case:

      i) If $OPMA_i$ is a <parenthesized path pattern expression> that simply contains a <subpath variable declaration>, then let $OPMAX_i$ be $OPMA_i$.

      ii) Otherwise, let $OPMAX_i$ be the <parenthesized path pattern expression>

      $(SOPMA_i = OPMA_i)$

   f) *PMA* is equivalent to:

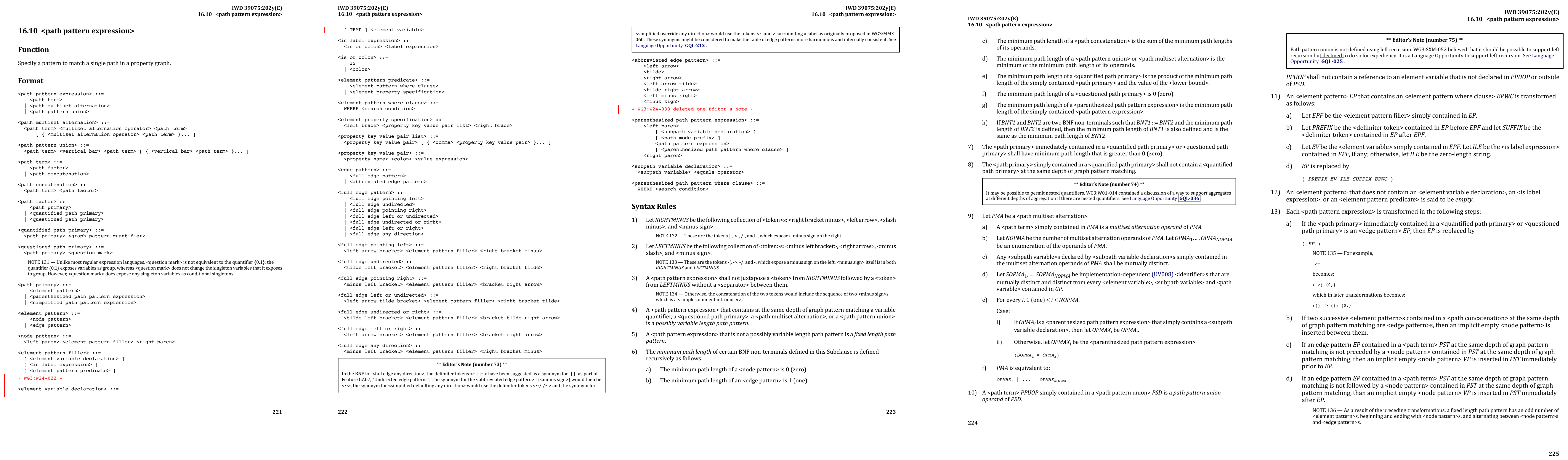
      $OPMAX_1$ | ... | $OPMAX_{NOPMA}$

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

224

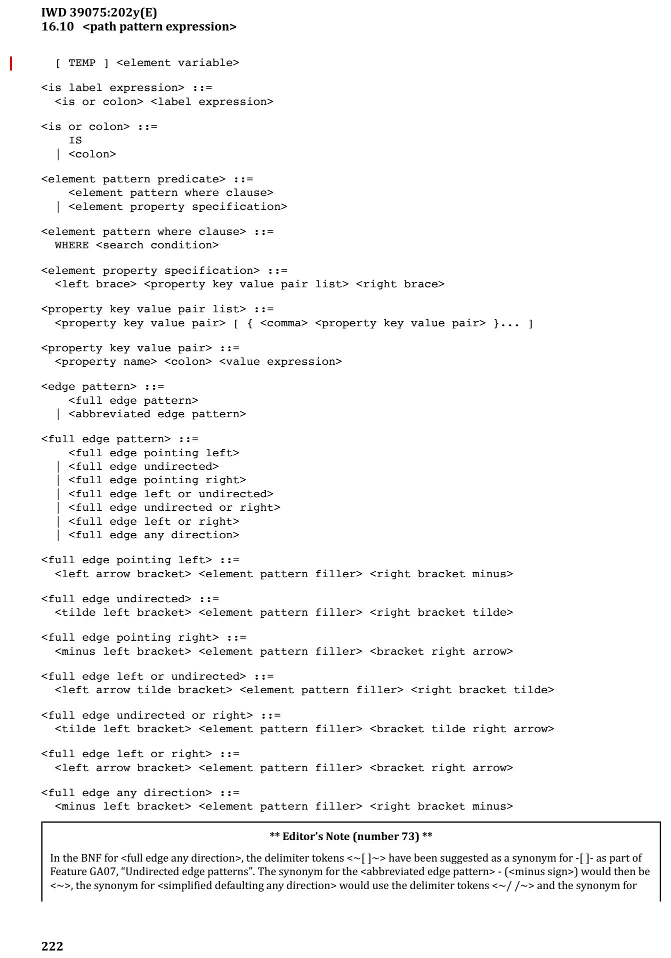
Standards are great but not for academics

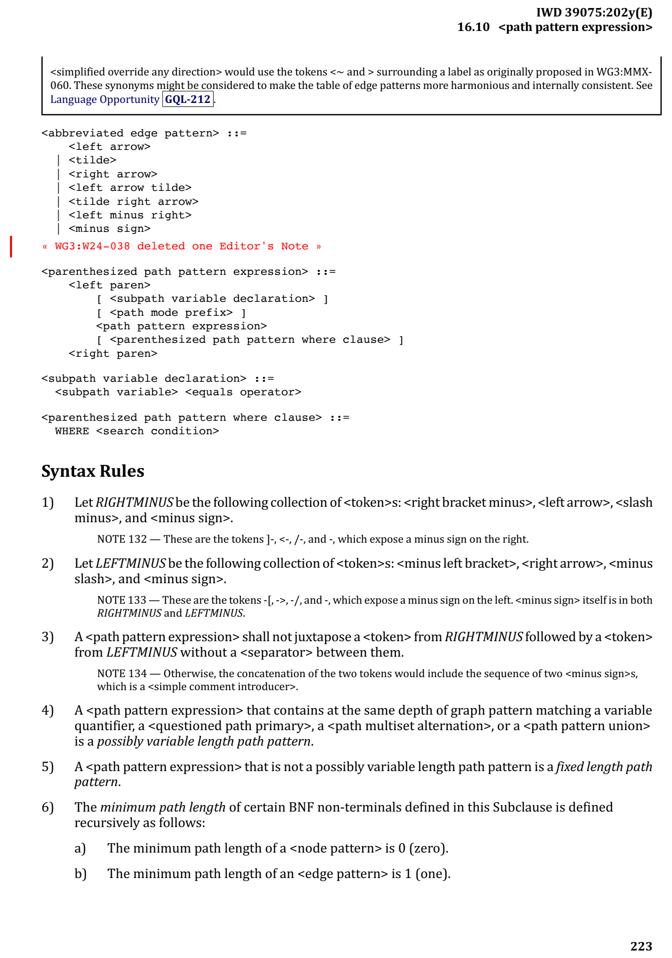
You want to write a paper about pattern matching and start with the syntax

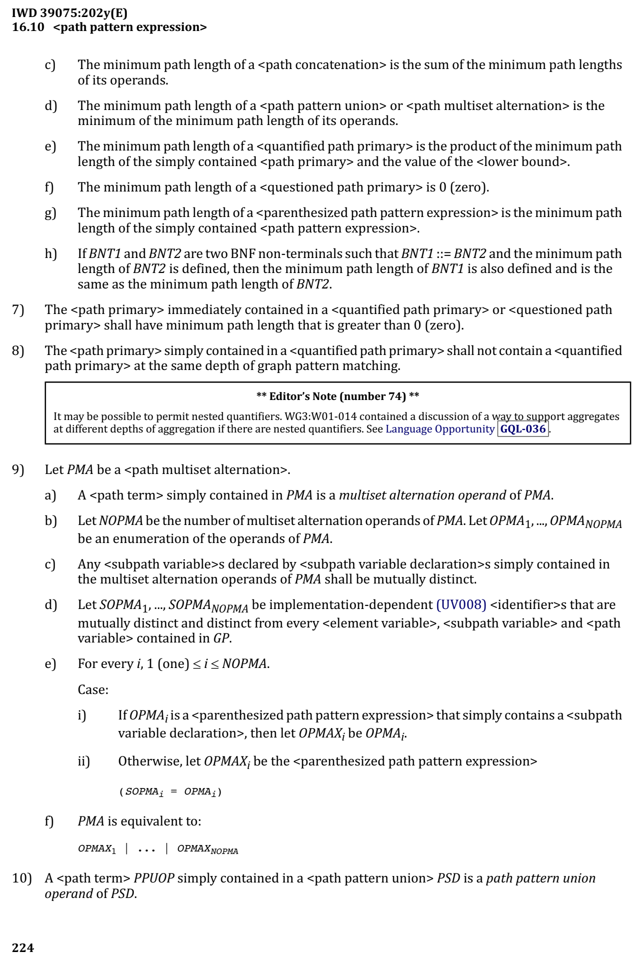


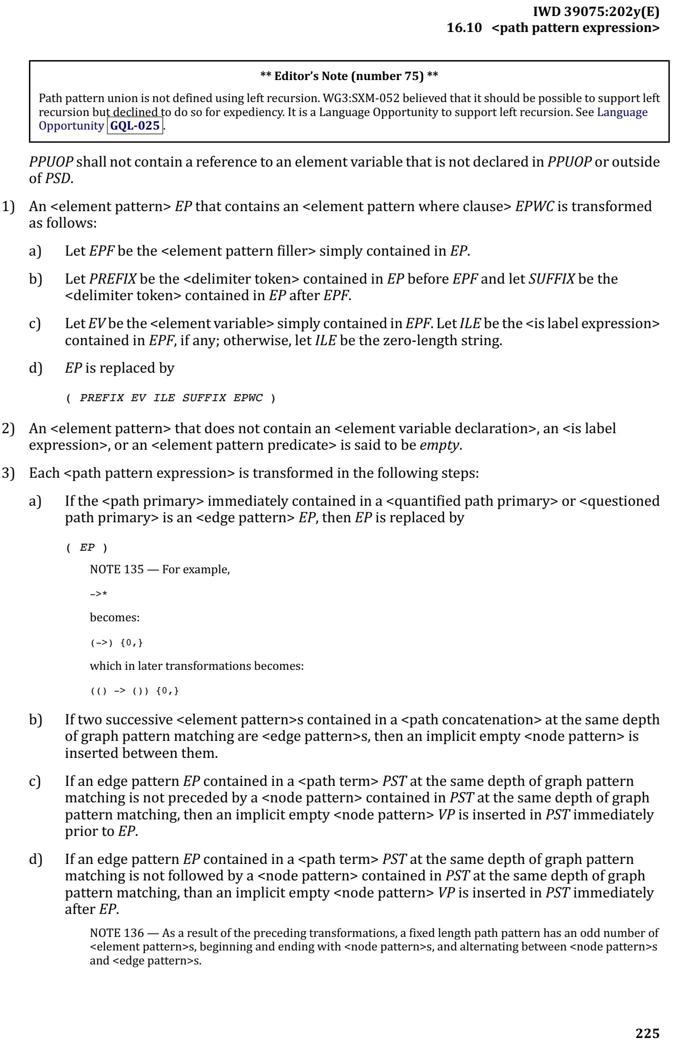
Standards are great but not for academics

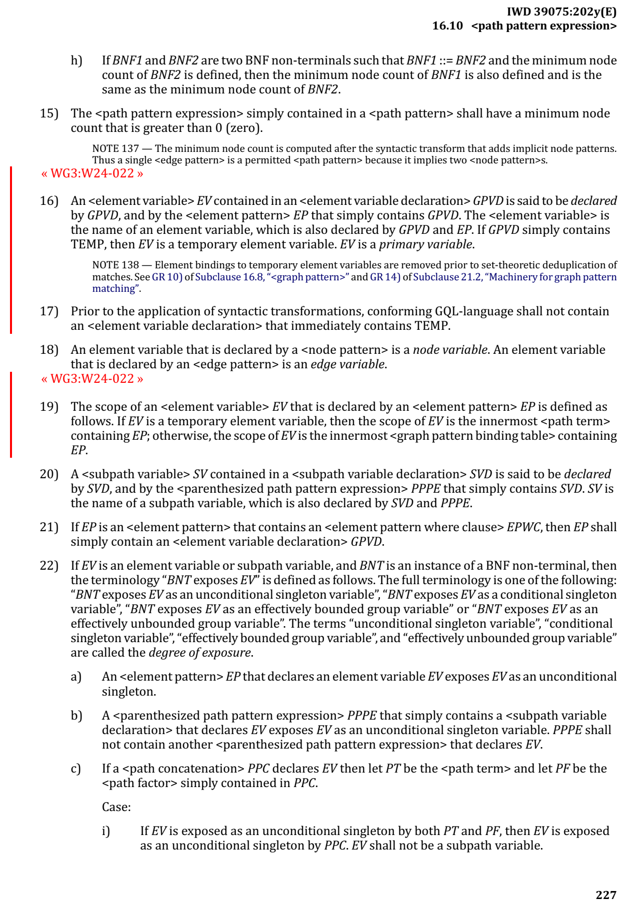
You want to write a paper about pattern matching and start with the syntax





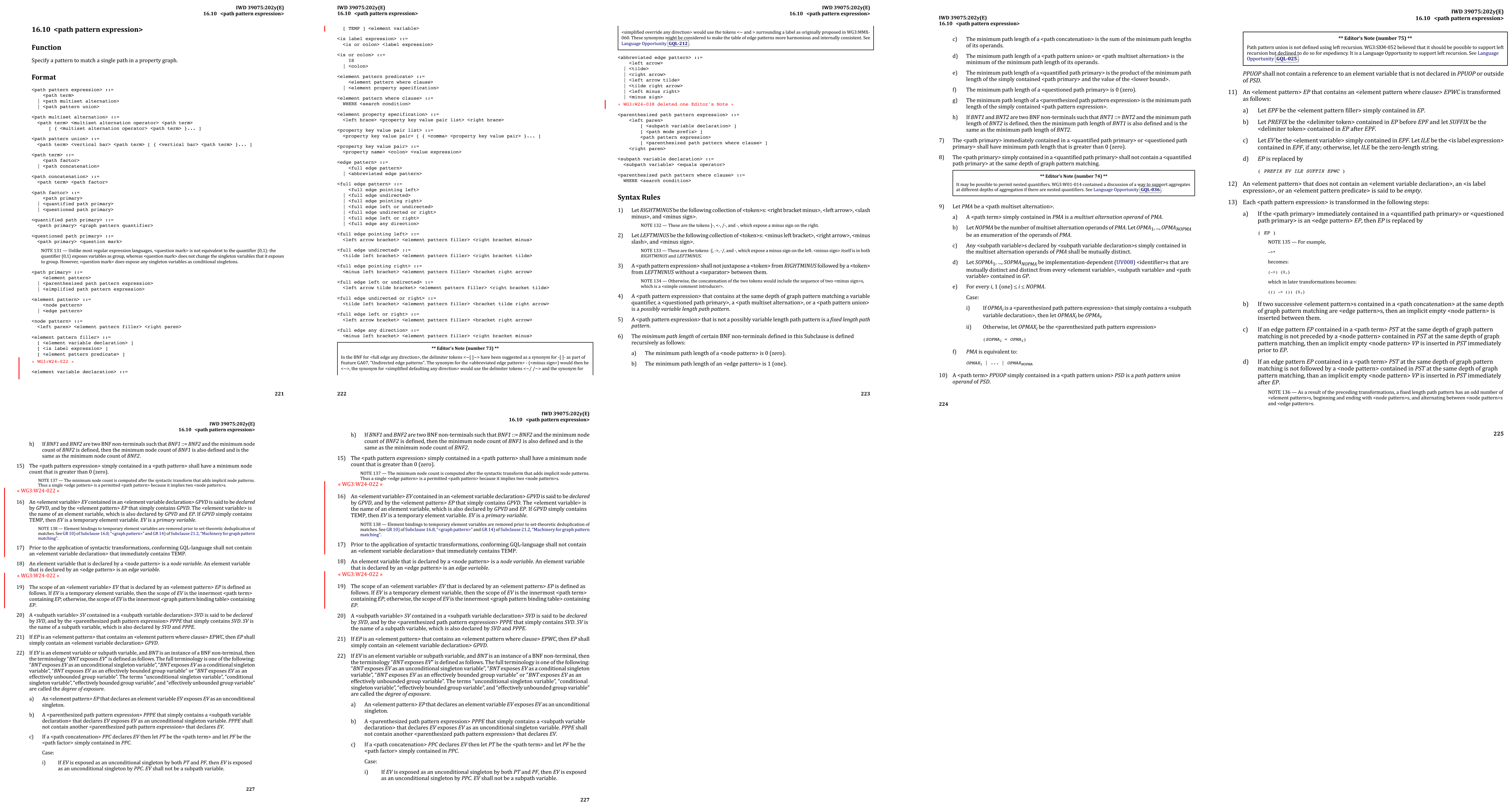






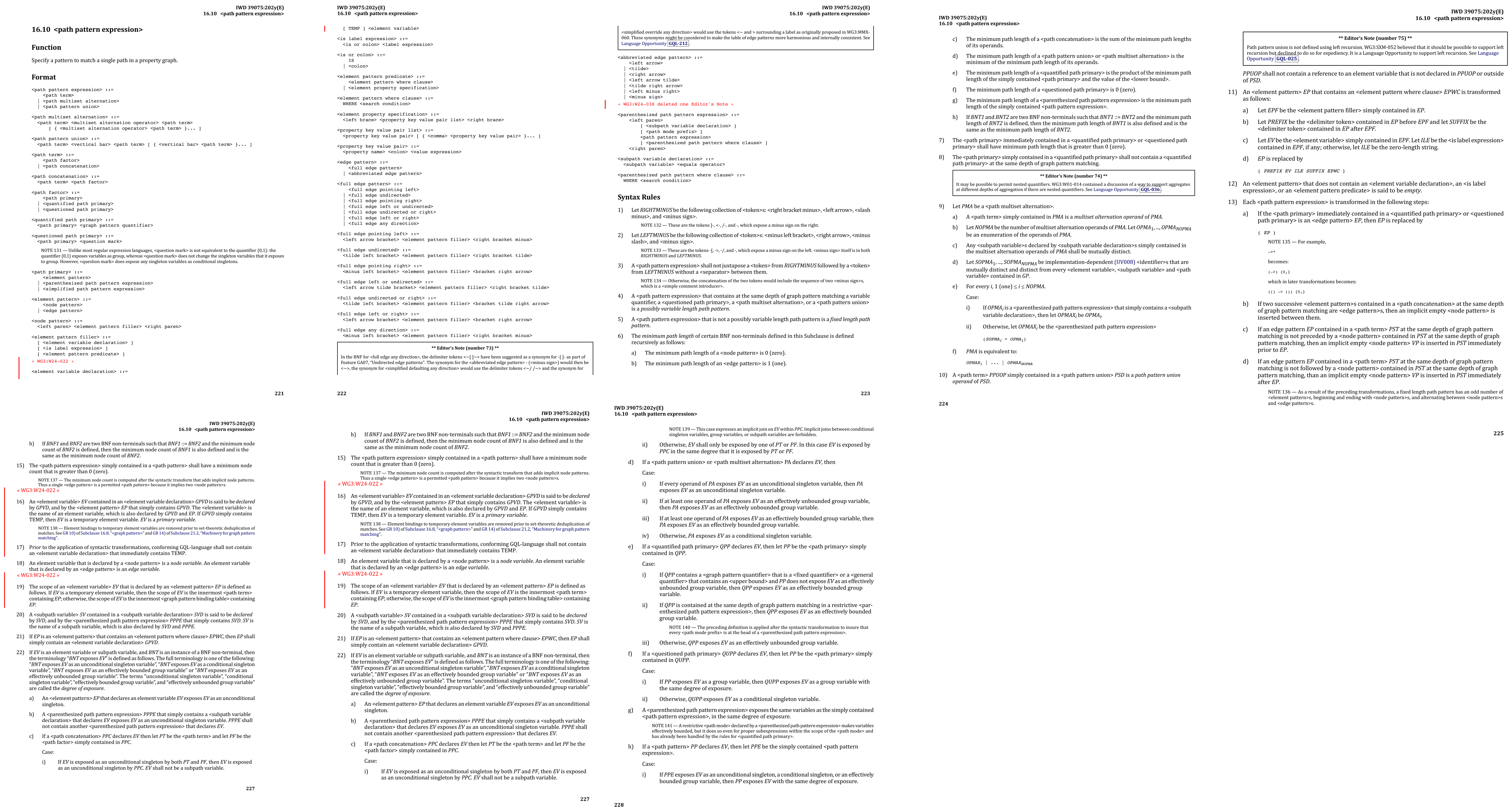
Standards are great but not for academics

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



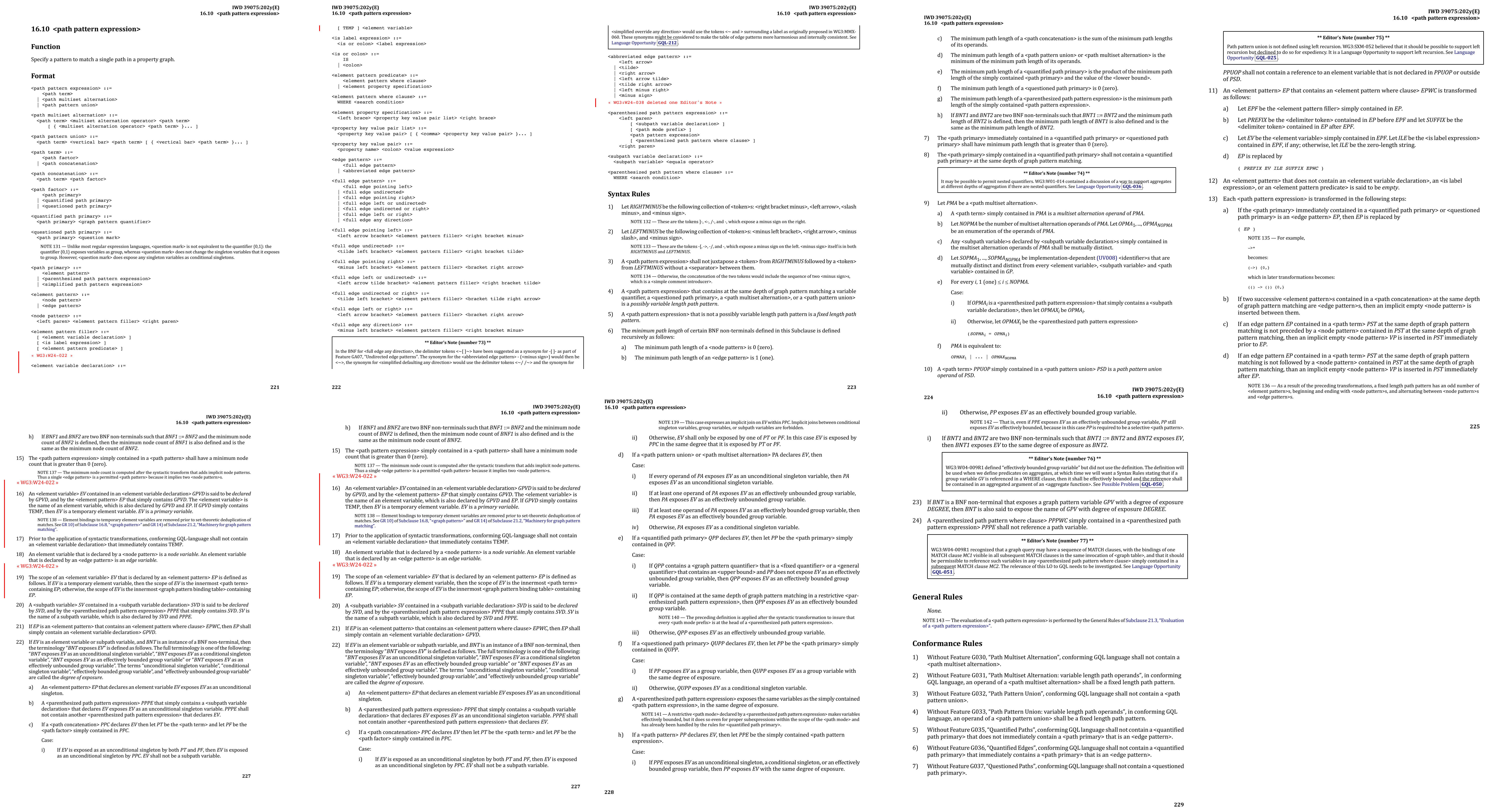
Standards are great but not for academics

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



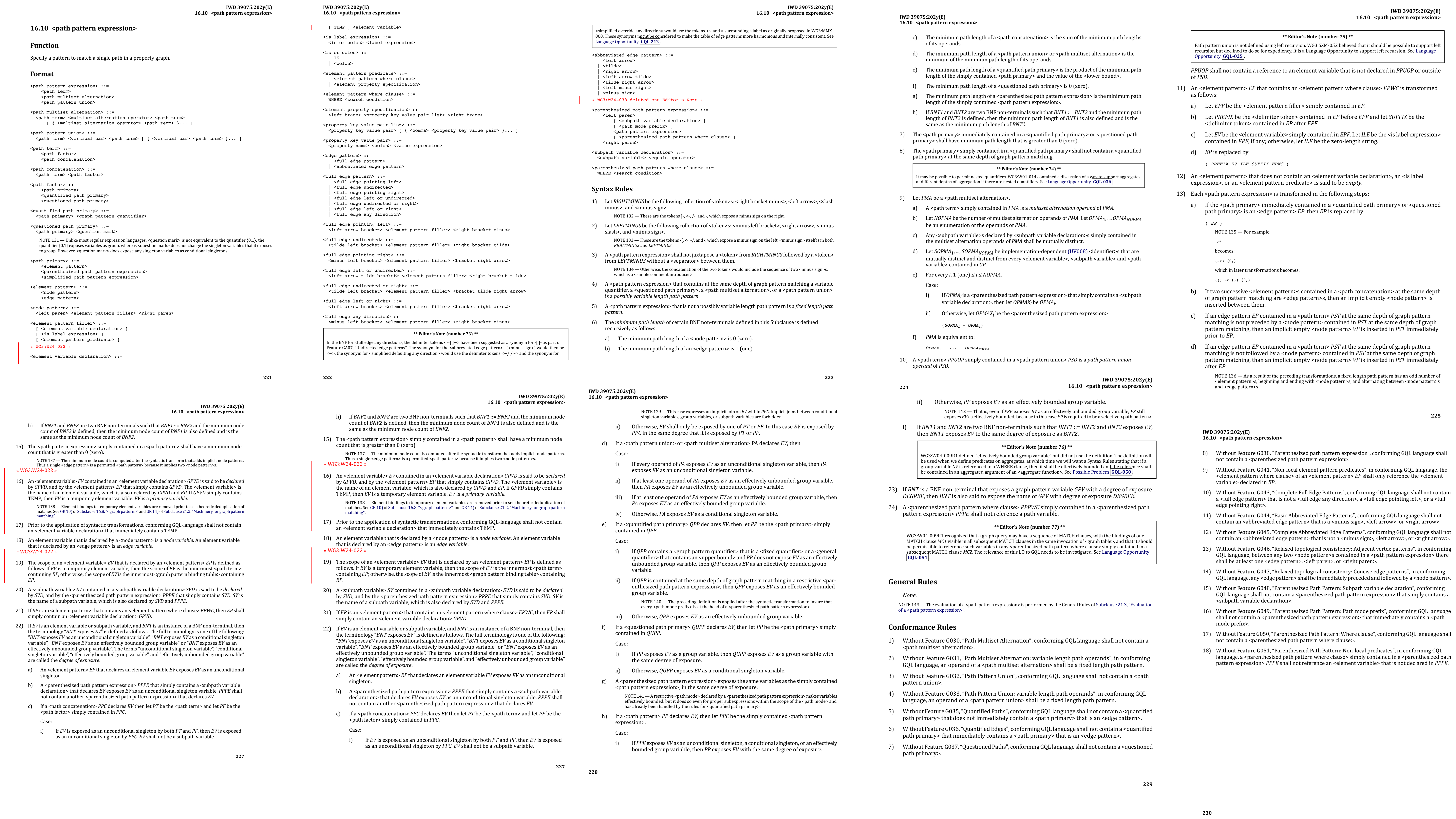
Standards are great but not for academics

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



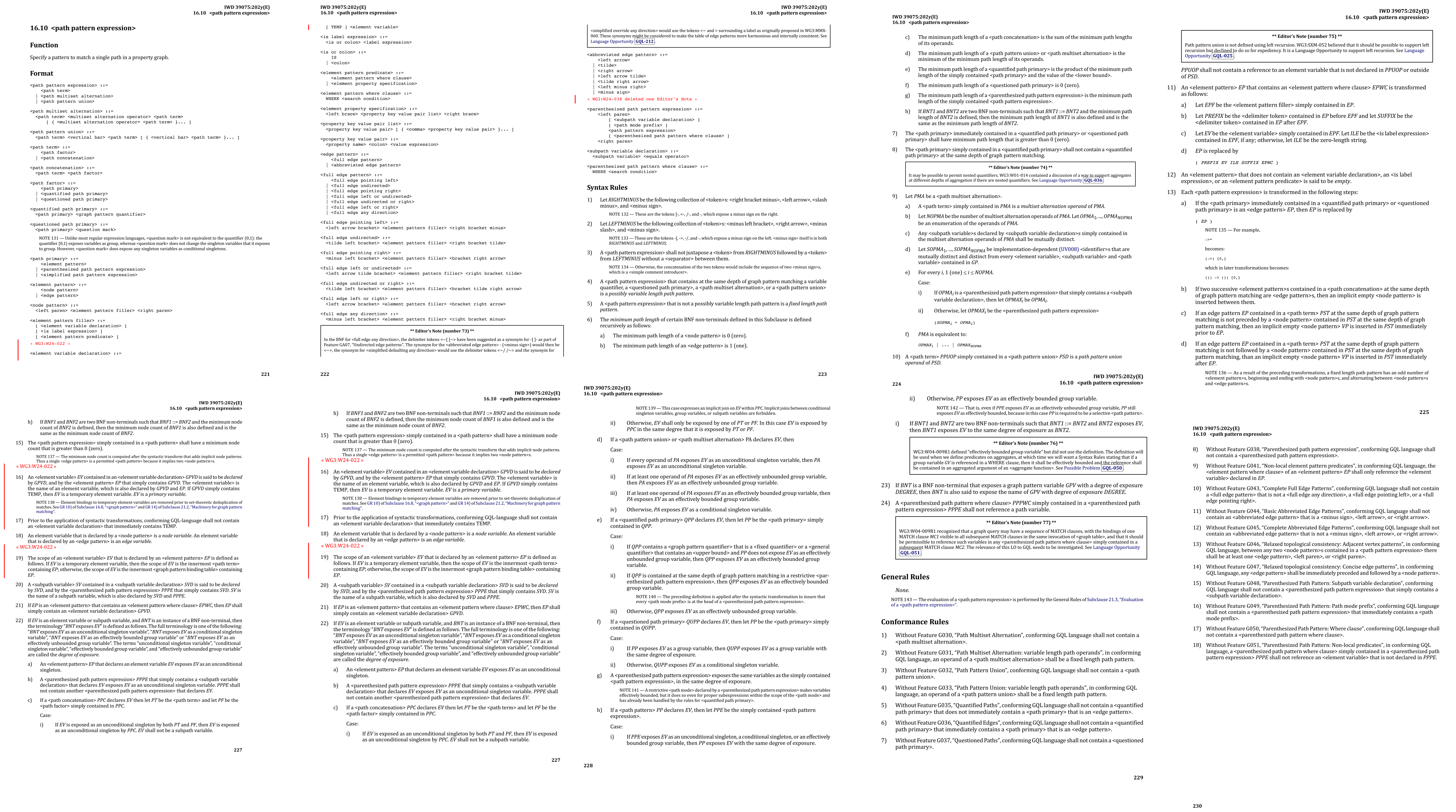
Standards are great but not for academics

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



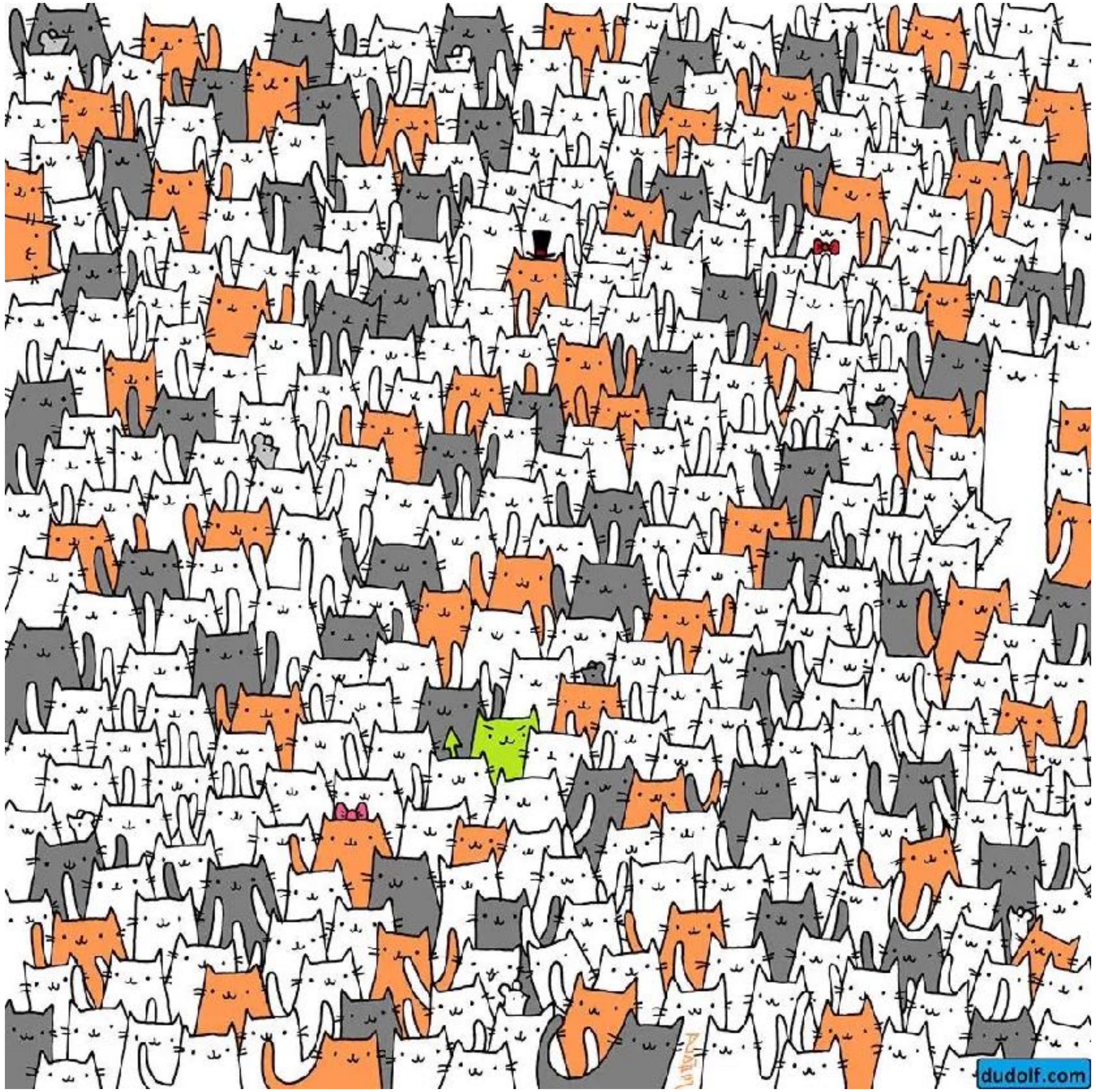
Standards are great but not for academics

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



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

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.....

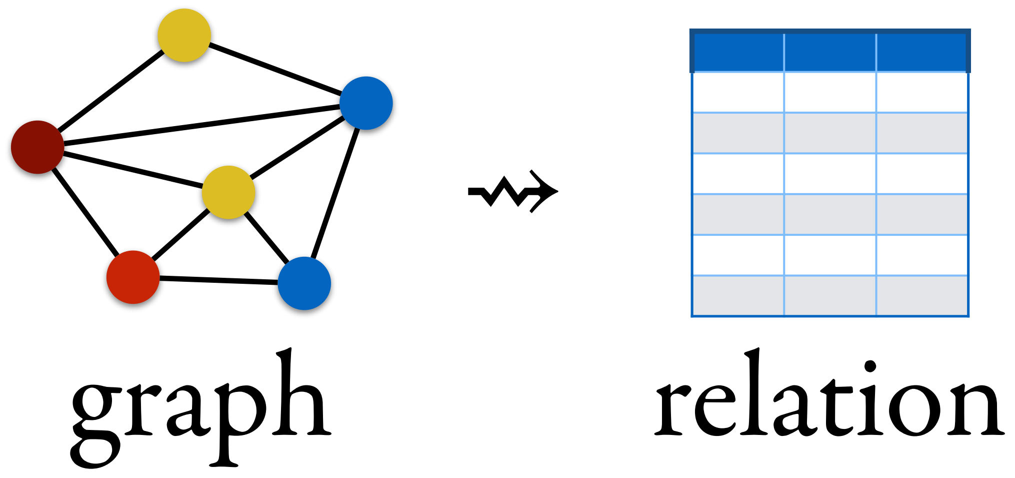
Papers/talks

- 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

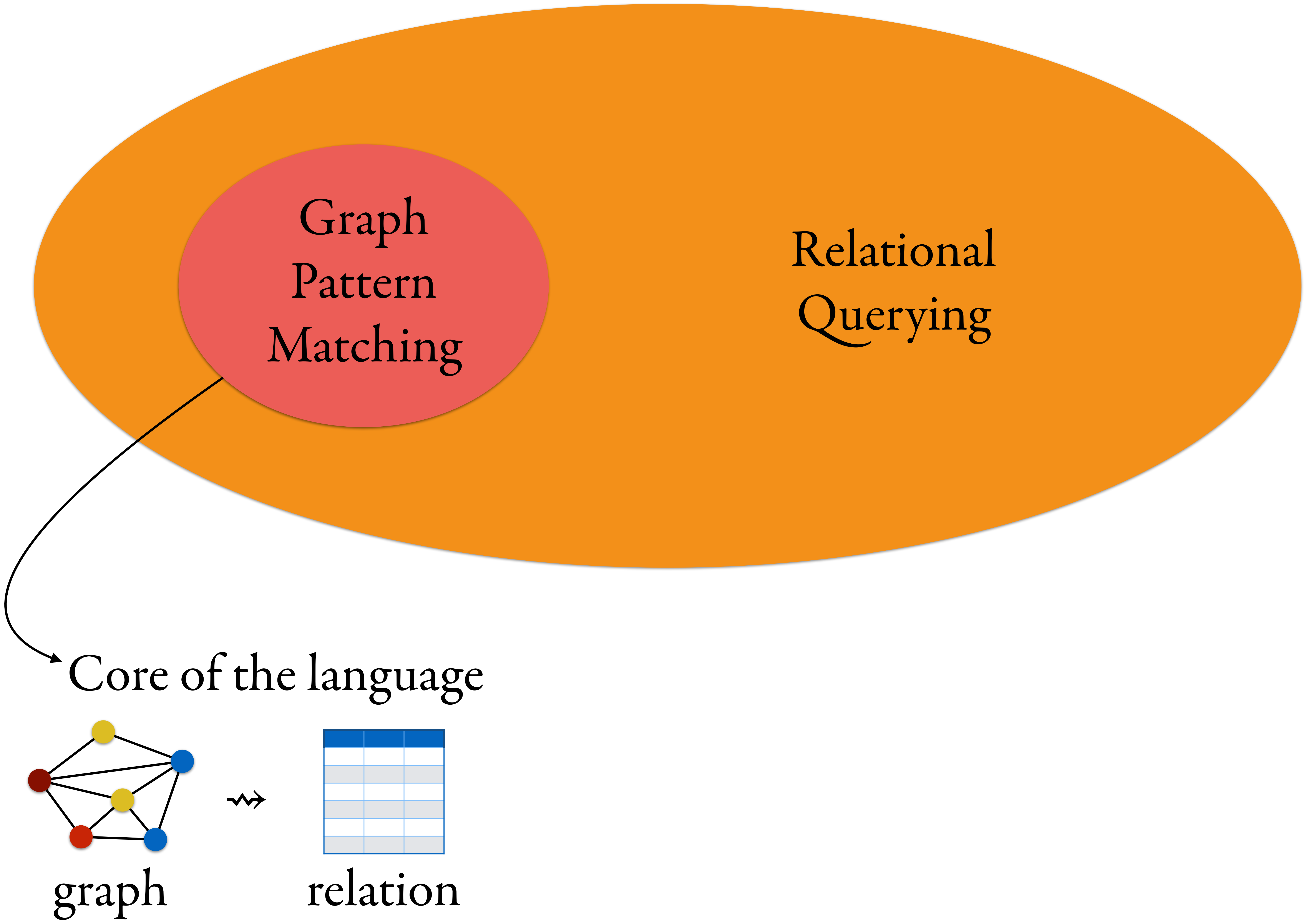
GQL in a Nutshell

Graph
Pattern
Matching

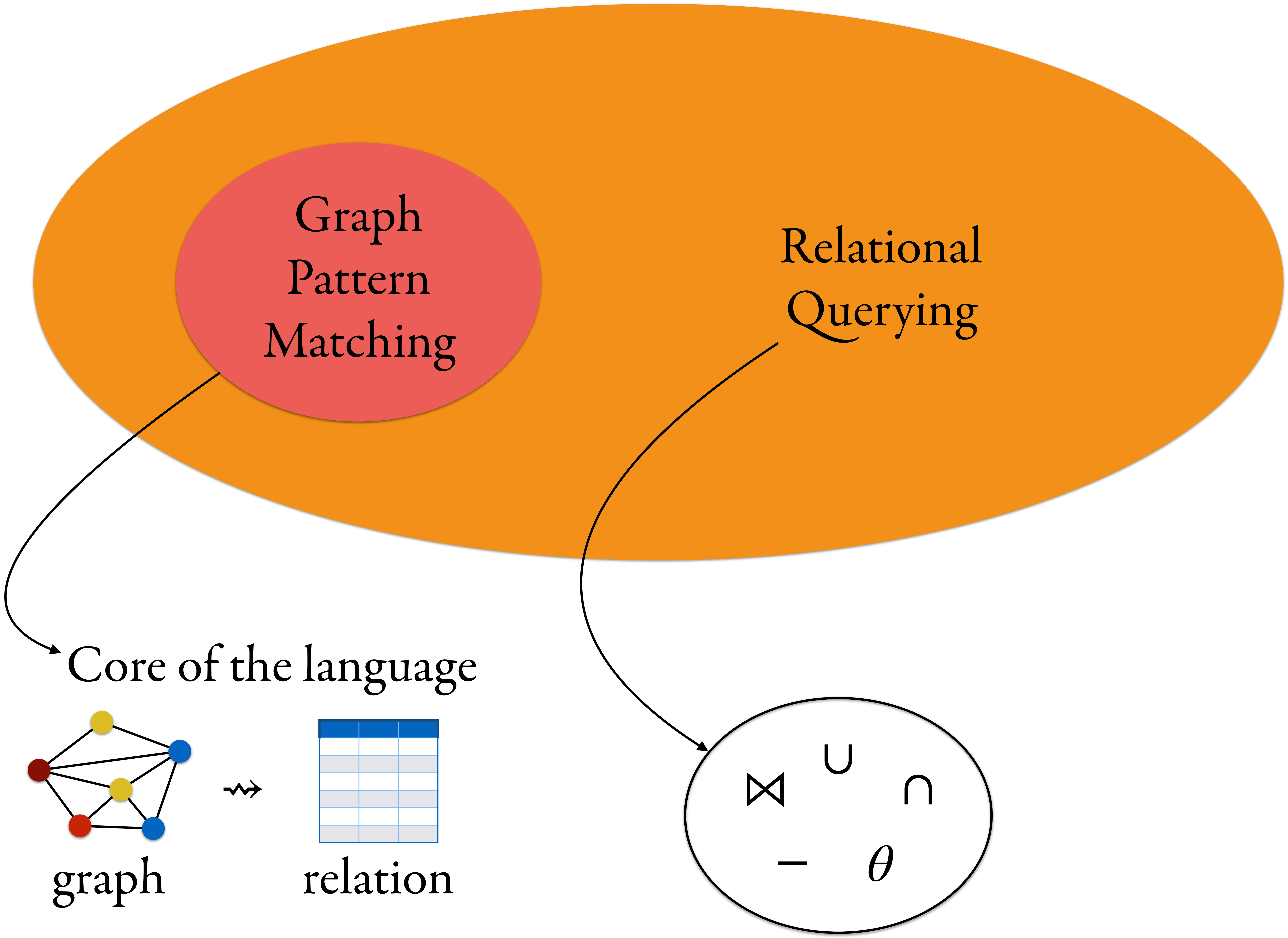
Core of the language



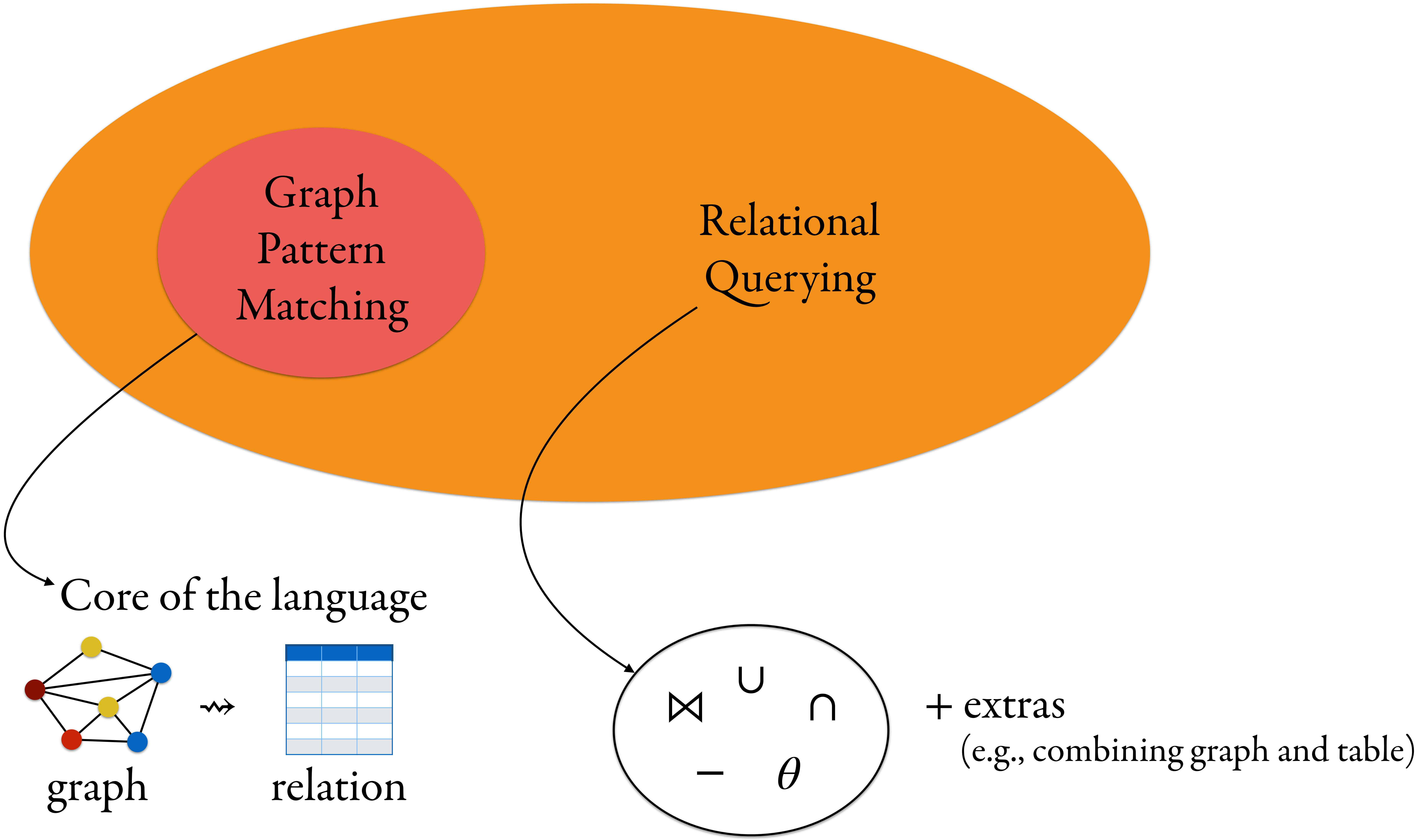
GQL in a Nutshell



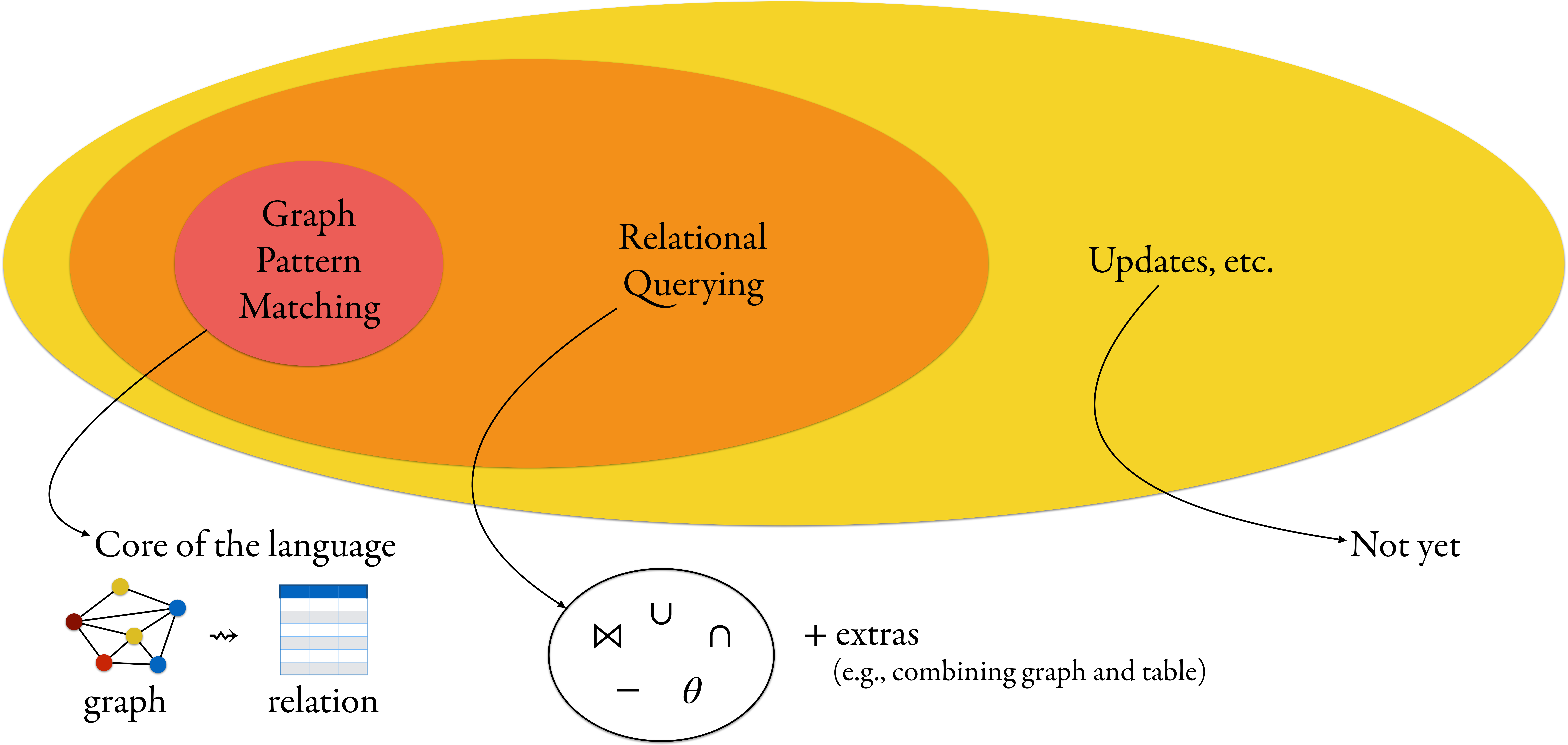
GQL in a Nutshell



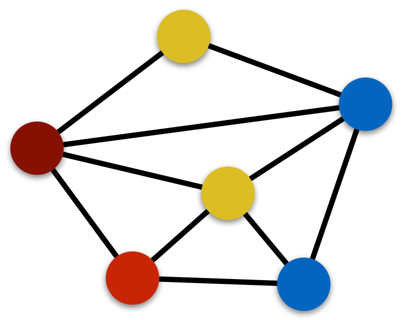
GQL in a Nutshell



GQL in a Nutshell

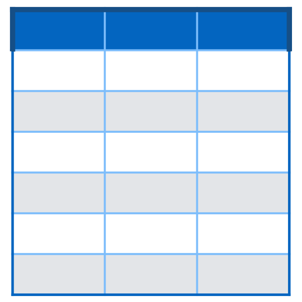


The Core:
Graph Pattern Matching



graph

$\rightsquigarrow$



relation

PODS 2023

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

Pattern calculus in a nutshell: PODS 23

Node pattern $\nu := (x : \ell)$ match an ℓ-labeled node, assign to a variable

Edge pattern $\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \overset{x:\ell}{----}$ 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 $\alpha := \xrightarrow{x : \ell} \mid \xleftarrow{x : \ell} \mid \overset{x : \ell}{\text{----}}$ 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 $\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \overset{x:\ell}{----}$ 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 $\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \overset{x:\ell}{\text{----}}$ 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 $\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \overset{x:\ell}{----}$ 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	$\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \overset{x:\ell}{----}$	ℓ-labeled edge directed left/right/any-directed, assign to a variable

Patterns

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

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 $\alpha := \overset{x:\ell}{\longrightarrow} \mid \overset{x:\ell}{\longleftarrow} \mid \overset{x:\ell}{- - - -}$ 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

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 $\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \overset{x:\ell}{\text{----}}$ ℓ-labeled edge directed left/right/any-directed, assign to a variable

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

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
Edge pattern	$\alpha := \xrightarrow{x:\ell} \mid \xleftarrow{x:\ell} \mid \overset{x:\ell}{----}$	ℓ-labeled edge directed left/right/any-directed, assign to a variable

Both x and ℓ are optional

Patterns

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

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

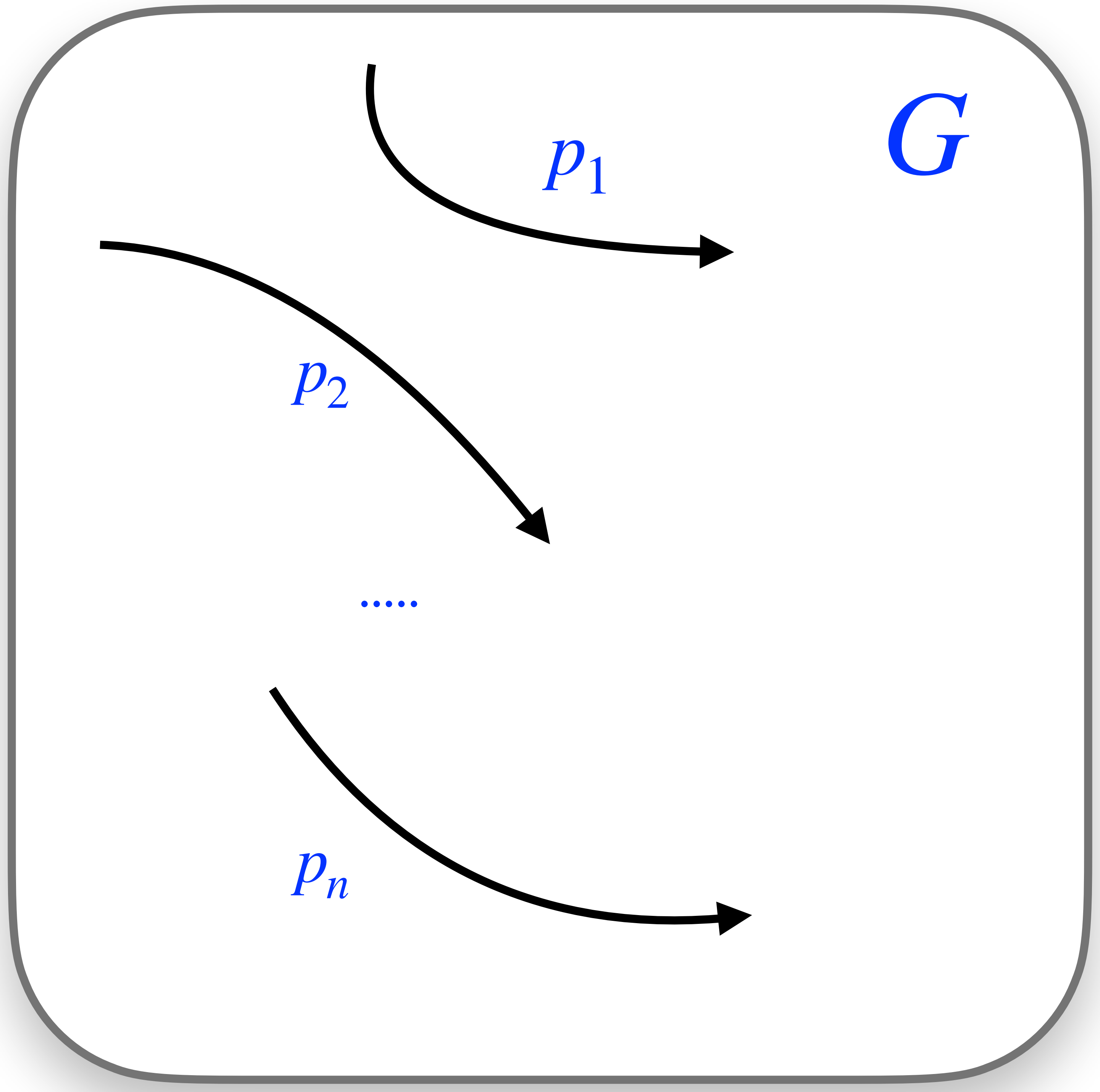
Queries $Q := \sigma\ \pi \mid p = \sigma\ \pi \mid Q, Q$

ensure finitely many paths | name matched path | join

Semantics — Idea

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

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

$$\pi = \ldots(x)\ldots \xrightarrow{y} \ldots.\left(\ldots(u)\ldots\right)^{n..m}\ldots\left(\ldots \xrightarrow{w} \ldots\right)^{\ell,k}\ldots$$

n e $[n_1, n_2, \ldots]$ $[e_1, e_2, \ldots]$

What's in the table?

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

$$\pi = \ldots(x)\ldots \xrightarrow{y} \ldots.(\ldots(u)\ldots)^{n..m}\ldots(\ldots \xrightarrow{w} \ldots)^{\ell,k}\ldots$$

$$n \qquad e \qquad [n_1, n_2, \ldots] \qquad [e_1, e_2, \ldots]$$

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 $x \in \mathsf{Vars}$, $\ell \in \mathcal{L}$, $0 \le n \le m \in \mathbb{N}$:

(descriptor)	$\delta := x$ `:`ℓ `WHERE` θ	x, `:`ℓ, and `WHERE` θ are optional
(path pattern)	$\pi :=$ `(`δ`)`	(node pattern)
	$\mid$ `-[`δ`]->` $\mid$ `<-[`δ`]-` $\mid$ `~[`δ`]~`	(edge pattern)
	$\mid \pi\,\pi$	(concatenation)
	$\mid \pi$ `|` π	(union)
	$\mid \pi$ `WHERE` θ	(conditioning)
	$\mid \pi$`{`n`,`m`}`	(bounded repetition)
	$\mid \pi$`{`n`,}`	(unbounded repetition)

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

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

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

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

Syntax: QUERIES

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

$$
\begin{array}{llcl}
\text{(clause)} & \mathsf{C} & := & \texttt{MATCH}\ \Pi \\
 & & \mid & \texttt{LET}\ x\ \texttt{=}\ \chi \\
 & & \mid & \texttt{FOR}\ x\ \texttt{IN}\ y \\
 & & \mid & \texttt{FILTER}\ \theta \\
\text{(linear query)} & \mathsf{L} & := & \texttt{USE}\ G\ \mathsf{L} \\
 & & \mid & \mathsf{C}\ \mathsf{L} \\
 & & \mid & \texttt{RETURN}\ \chi_1\ \texttt{AS}\ x_1, \ldots, \chi_k\ \texttt{AS}\ x_k \\
\text{(query)} & \mathsf{Q} & := & \mathsf{L} \\
 & & \mid & \texttt{USE}\ G\ \texttt{\{}\mathsf{Q}_1\ \texttt{THEN}\ \mathsf{Q}_2\ \cdots\ \texttt{THEN}\ \mathsf{Q}_\ell\texttt{\}} \\
 & & \mid & \mathsf{Q}\ \texttt{INTERSECT}\ \mathsf{Q} \mid \mathsf{Q}\ \texttt{UNION}\ \mathsf{Q} \mid \mathsf{Q}\ \texttt{EXCEPT}\ \mathsf{Q}
\end{array}
$$

ICDT Paper: Semantics

$$[\![\texttt{-[]->}]\!]_G = \{\, (\mathsf{path}(\mathsf{src}(e), e, \mathsf{tgt}(e)), ()) \mid e \in E_d^G \,\}$$
$$[\![\texttt{-[}x\texttt{]->}]\!]_G = \{\, (\mathsf{path}(\mathsf{src}(e), e, \mathsf{tgt}(e)), (x \mapsto e)) \mid e \in E_d^G \,\}$$
$$[\![\texttt{-[:}\ell\texttt{]->}]\!]_G = \left\{ (\mathsf{path}(\mathsf{src}(e), e, \mathsf{tgt}(e)), ()) \;\middle|\; e \in E_d^G, \ell \in \mathsf{lab}^G(e) \right\}$$

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.

$$[\![\texttt{<-[]-}]\!]_G = \{\, (\mathsf{path}(\mathsf{tgt}(e), e, \mathsf{src}(e)), ()) \mid e \in E_d^G \,\}$$
$$[\![\texttt{\~{}[]\~{}}]\!]_G = \left\{ (\mathsf{path}(u_1, e, u_2), ()), (\mathsf{path}(u_2, e, u_1), ()) \;\middle|\; \begin{array}{l} e \in E_u^G \\ \{u_1, u_2\} = \mathsf{endpoints}^G(e) \end{array} \right\}$$

Semantics of Concatenation, Union, and Conditioning

$$[\![\pi_1\ \pi_2]\!]_G \left\{ (p_1 \cdot p_2, \mu_1 \bowtie \mu_2) \;\middle|\; \begin{array}{l} (p_i, \mu_i) \in [\![\pi_i]\!]_G \text{ for } i = 1,2 \\ p_1 \text{ and } p_2 \text{ concatenate} \\ \mu_1 \sim \mu_2 \end{array} \right\}$$

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) (-[:Transfer]->()-[:Transfer]->(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.

$$[\![\pi_1 \mid \pi_2]\!]_G = \{\, (p, \mu \cup \mu') \mid (p, \mu) \in [\![\pi_1]\!]_G \cup [\![\pi_2]\!]_G \,\}$$

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

$$[\![\pi \texttt{ WHERE } \theta]\!]_G = \{\, (p, \mu) \in [\![\pi]\!]_G \mid [\![\theta]\!]_G^\mu = \mathsf{true} \,\}$$

Semantics of Repetition

$$[\![\pi\texttt{\{}n,m\texttt{\}}]\!]_G = \bigcup_{i=n}^{m} [\![\pi]\!]_G^i$$
$$[\![\pi\texttt{\{}n,\texttt{\}}]\!]_G = \bigcup_{i=n}^{\infty} [\![\pi]\!]_G^i$$

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

$$[\![\pi]\!]_G^0 = \{\, (\mathsf{path}(u), \mu) \mid u \text{ is a node in } G \,\}$$

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

$$\forall i > 0 \quad [\![\pi]\!]_G^i = \left\{ (p_1 \cdot \ldots \cdot p_i, \mu') \;\middle|\; \begin{array}{l} (p_1, \mu_1), \ldots, (p_n, \mu_i) \in [\![\pi]\!]_G \\ p_1, \ldots, p_i \text{ concatenate} \end{array} \right\}$$

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

▶ Remark 10. Since $\pi\texttt{\{}n,\texttt{\}}$ 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 $(p, \mu) \in [\![\pi\texttt{\{}n,m\texttt{\}}]\!]_G$. 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 `(a){0,}` which is not well-formed (the minimal path length of `()` is 0). For every i, the set $[\![\texttt{(a)}]\!]_G^i$ contains $(\mathsf{path}(u), \mu_i)$ where $\mu_i = (a \mapsto \mathsf{list}(\underbrace{u, \ldots, u}_{i \text{ times}}))$; hence the union in the definition of $[\![\pi\texttt{\{}n,\texttt{\}}]\!]_G$ 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 (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.

$$[\![x \texttt{=} \pi]\!]_G = \{ (p, \mu \cup \{x \mapsto p\}) \mid (p, \mu) \in [\![\pi]\!]_G \}$$

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

$$[\![\texttt{TRAIL } \pi]\!]_G = \{\, (p, \mu) \in [\![\pi]\!]_G \mid \text{no edge occurs more than once in } p \,\}$$
$$[\![\texttt{ACYCLIC } \pi]\!]_G = \{\, (p, \mu) \in [\![\pi]\!]_G \mid \text{no node occurs more than once in } p \,\}$$
$$[\![\texttt{SHORTEST } \tilde{\pi}]\!]_G = \left\{ (p, \mu) \in [\![\tilde{\pi}]\!]_G \;\middle|\; \mathsf{len}(p) = \min \left\{ \mathsf{len}(p') \;\middle|\; \begin{array}{l} (p', \mu') \in [\![\tilde{\pi}]\!]_G \\ \mathsf{src}(p') = \mathsf{src}(p) \\ \mathsf{tgt}(p') = \mathsf{tgt}(p) \end{array} \right\} \right\}$$
$$[\![\texttt{ALL } \tilde{\pi}]\!]_G = [\![\tilde{\pi}]\!]_G$$
$$[\![\texttt{ANY } \tilde{\pi}]\!]_G = \bigcup_{(s,t) \in X} \{ \mathsf{any}(\{\, (p, \mu) \mid (p, \mu) \in [\![\tilde{\pi}]\!]_G, \mathsf{endpoints}(p) = (s, t) \,\}) \}$$

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

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

Here, $\bar{p}_1 = (p_1^1, p_1^2, \ldots, p_1^k)$ and $\bar{p}_2 = (p_2^1, p_2^2, \ldots, p_2^l)$ are tuples of paths, and $\bar{p}_1 \times \bar{p}_2$ stands for $(p_1^1, p_1^2, \ldots, p_1^k, p_2^1, p_2^2, \ldots, p_2^l)$. Just as it is the case of concatenation, since $\Pi_1\texttt{,}\ \Pi_2$ is well-formed, implicit joins can occur over singleton variables only.

4.4 Semantics of Conditions and Expressions

The semantics $[\![\chi]\!]_G^\mu$ 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.

$$[\![c]\!]_G^\mu = c \qquad \text{for } c \in \mathsf{Const}$$
$$[\![x]\!]_G^\mu = \mu(x) \qquad \text{for } x \in \mathrm{Dom}(\mu)$$
$$[\![x.a]\!]_G^\mu = \begin{cases} \mathsf{prop}^G(\mu(x), a) & \text{if } (\mu(x), a) \in \mathrm{Dom}(\mathsf{prop}^G) \\ \mathsf{null} & \text{else if } \mu(x) \in (\mathcal{N} \cup \mathcal{E}_\mathsf{d} \cup \mathcal{E}_\mathsf{u}) \end{cases} \qquad \text{for } x \in \mathrm{Dom}(\mu), a \in \mathcal{K}$$

ICDT Paper: Semantics

▶ Remark 11. Recall that different graphs may share nodes and edges. Hence the condition $(\mu(x), a) \in \mathrm{Dom}(\mathsf{prop}^G)$, 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 $[\![\theta]\!]_G^\mu$ of a condition θ is an element in $\{\mathsf{true}, \mathsf{false}, \mathsf{null}\}$ that is evaluated with respect to a binding μ and a graph G, and is defined as follows:

$$[\![\chi_1 \texttt{ = } \chi_2]\!]_G^\mu = \begin{cases} \mathsf{null} & \text{if } [\![\chi_1]\!]_G^\mu = \mathsf{null} \text{ or } [\![\chi_2]\!]_G^\mu = \mathsf{null} \\ \mathsf{true} & \text{if } [\![\chi_1]\!]_G^\mu = [\![\chi_2]\!]_G^\mu \neq \mathsf{null} \\ \mathsf{false} & \text{otherwise} \end{cases}$$

$$[\![\chi_1 \texttt{ < } \chi_2]\!]_G^\mu = \begin{cases} \mathsf{null} & \text{if } [\![\chi_1]\!]_G^\mu = \mathsf{null} \text{ or } [\![\chi_2]\!]_G^\mu = \mathsf{null} \\ \mathsf{true} & \text{else if } [\![\chi_1]\!]_G^\mu < [\![\chi_2]\!]_G^\mu \\ \mathsf{false} & \text{otherwise} \end{cases}$$

$$[\![\chi \texttt{ IS NULL}]\!]_G^\mu = \begin{cases} \mathsf{true} & \text{if } [\![\chi]\!]_G^\mu = \mathsf{null} \\ \mathsf{false} & \text{otherwise} \end{cases}$$

$$[\![\chi \texttt{:} \ell]\!]_G^\mu = \begin{cases} \mathsf{true} & \text{if } [\![\chi]\!]_G^\mu \in N^G \cup E_u^G \cup E_d^G \text{ and } \ell \in \mathsf{lab}^G([\![\chi]\!]_G^\mu) \\ \mathsf{false} & \text{else if } [\![\chi]\!]_G^\mu \in \mathcal{N} \cup \mathcal{E}_\mathsf{d} \cup \mathcal{E}_\mathsf{u} \end{cases}$$

$$[\![\theta_1 \texttt{ AND } \theta_2]\!]_G^\mu = [\![\theta_1]\!]_G^\mu \wedge [\![\theta_2]\!]_G^\mu \quad {}^{(*)}$$

$$[\![\theta_1 \texttt{ OR } \theta_2]\!]_G^\mu = [\![\theta_1]\!]_G^\mu \vee [\![\theta_2]\!]_G^\mu \quad {}^{(*)}$$

$$[\![\texttt{NOT } \theta]\!]_G^\mu = \neg [\![\theta]\!]_G^\mu \quad {}^{(*)}$$

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

$$[\![\texttt{EXISTS \{ } \mathsf{Q} \texttt{ \}}]\!]_G^\mu = \begin{cases} \mathsf{true} & \text{if } [\![\mathsf{Q}]\!]_G(\{\mu\}) \text{ is not empty} \\ \mathsf{false} & \text{otherwise} \end{cases}$$

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* $\mathrm{Dom}(T)$.

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 $[\![\mathsf{C}]\!]_G$ 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.

$$[\![\texttt{MATCH } \Pi]\!]_G(T) = \bigcup_{\mu \in T} \{\mu \bowtie \mu' \mid (p, \mu') \in [\![\Pi]\!]_G,\ \mu \sim \mu'\}$$

Note that if Π uses a variable that already occurs in $\mathrm{Dom}(T)$, 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.

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

$$[\![\texttt{LET } x \texttt{ = } \chi]\!]_G(T) = \bigcup_{\mu \in T} \{\mu \bowtie (x \mapsto [\![\chi]\!]_G^\mu)\}$$

$$[\![\texttt{FILTER } \theta]\!]_G(T) = \bigcup_{\mu \in T} \{\mu \mid [\![\theta]\!]_G^\mu = \mathsf{true}\}\ .$$

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

$$[\![\texttt{FOR } x \texttt{ IN } y]\!]_G(T) = \bigcup_{\mu \in T} \{\mu \bowtie (x \mapsto v) \mid v \in \mu(y)\}\ .$$

Semantics of Linear Queries

$$[\![\texttt{USE } G' \ \mathsf{L}]\!]_G(T) = [\![\mathsf{L}]\!]_{G'}(T)$$

$$[\![C \ \mathsf{L}]\!]_G(T) = [\![\mathsf{L}]\!]_G([\![C]\!]_G(T))$$

$$[\![\texttt{RETURN } \chi_1 \texttt{ AS } x_1, \dots, \chi_\ell \texttt{ AS } x_\ell]\!]_G(T) = \bigcup_{\mu \in T} \left\{(x_1 \mapsto [\![\chi_1]\!]_G^\mu, \dots, x_\ell \mapsto [\![\chi_\ell]\!]_\mu^G)\right\}$$

Semantics of Queries

The *output of a query* Q is defined as

$$\mathsf{Output}(\mathsf{Q}) = [\![\mathsf{Q}]\!]_G(\{()\})\ ,$$

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.

$$[\![\texttt{USE } G' \texttt{ \{}\mathsf{Q}_1 \texttt{ THEN } \mathsf{Q}_2 \cdots \texttt{ THEN } \mathsf{Q}_k\texttt{\}}]\!]_G(T) = [\![\mathsf{Q}_k]\!]_{G'} \circ \cdots \circ [\![\mathsf{Q}_1]\!]_{G'}(T)$$

If $\mathrm{Dom}([\![\mathsf{Q}_1]\!]_G(T)) = \mathrm{Dom}([\![\mathsf{Q}_2]\!]_G(T))$, then we let

$$[\![\mathsf{Q}_1 \texttt{ INTERSECT } \mathsf{Q}_2]\!]_G(T) = [\![\mathsf{Q}_1]\!]_G(T) \cap [\![\mathsf{Q}_2]\!]_G(T)$$

$$[\![\mathsf{Q}_1 \texttt{ UNION } \mathsf{Q}_2]\!]_G(T) = [\![\mathsf{Q}_1]\!]_G(T) \cup [\![\mathsf{Q}_2]\!]_G(T)$$

$$[\![\mathsf{Q}_1 \texttt{ EXCEPT } \mathsf{Q}_2]\!]_G(T) = [\![\mathsf{Q}_1]\!]_G(T) \setminus [\![\mathsf{Q}_2]\!]_G(T)$$

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 $\mathsf{list}()$

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 $\pi_1, \ldots, \pi_n$

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 $\pi_1, \ldots, \pi_n$

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

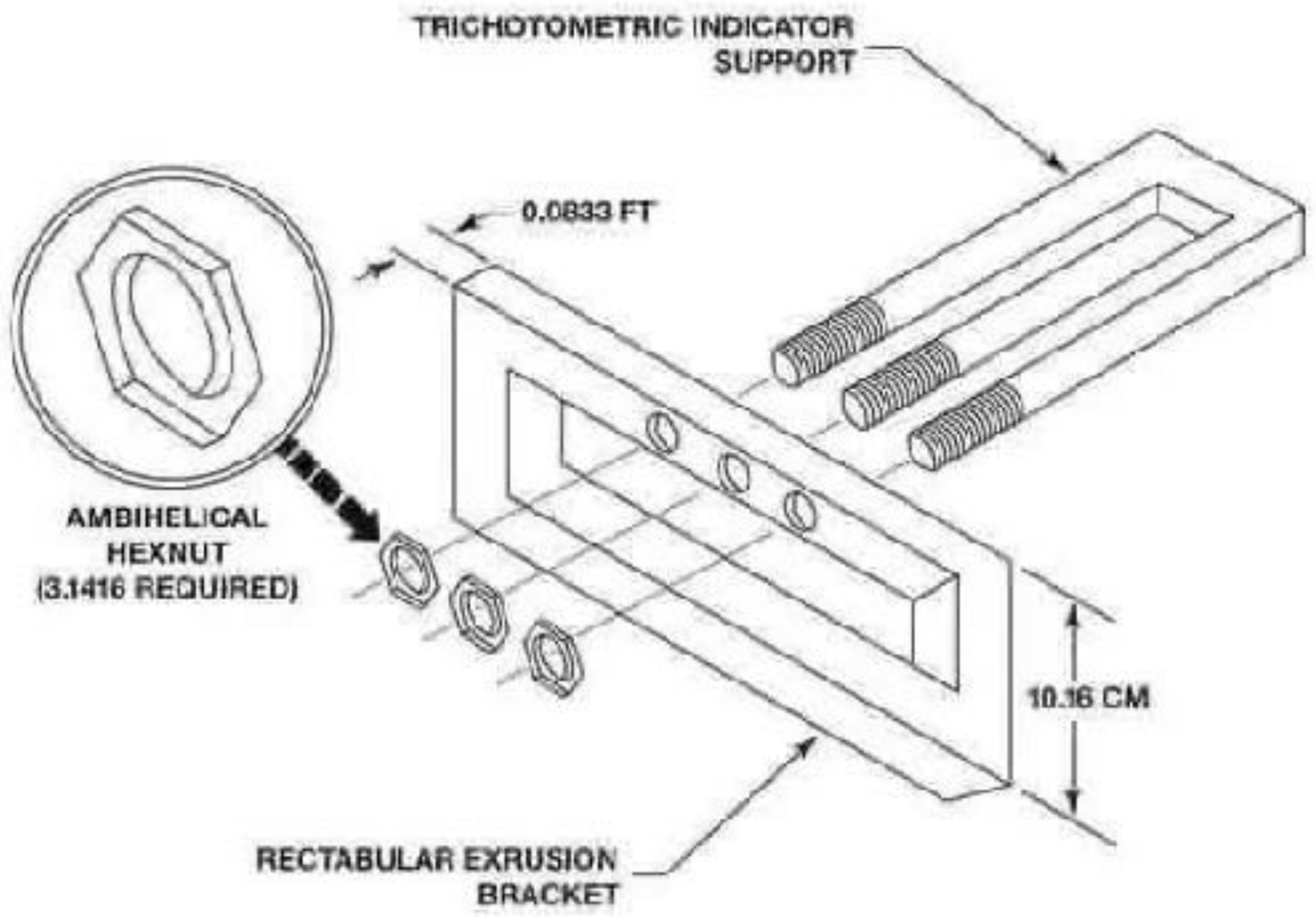
Questions:
Is it good? Is it usable? Or the best simply because everything else is average?

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