Deeper Dive into Cypher.PL

Executable semantics of Cypher 9 written in logic

{jan.posiadala,pawel.susicki}@gmail.com

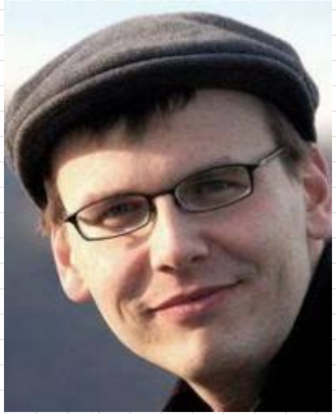
Cypher.PL academia advisor

Prof. at University of Warsaw

Institute of Informatics

Fields of research

- Logic in Computer Science
- Databases
  - semi-structured databases
  - XML Schema mappings
- Knowledge representation
  - description logics
  - ontology-mediated query answering
- Automata Theory
  - tree automata



Filip Murlak

Cypher.PL

What is Cypher.PL

- executable specification
- of declarative query language (Cypher)
- in formal declarative language of logic (Prolog)
- as close to the semantics as possible
- as far from the implementation issues as possible
- a tool for collective designing, verification, validation
- without losing of preciseness and explicitness

Why Prolog?

- declarative langauge
- with built-in unification...
- ...which is more general than pattern matching
- super-native data (structures) representation
- multiple solutions/evident ambiguity
- easy constraint verification
- DCG: notation for grammars
- meta-programming
- generative aspect of nature

Policy Paper
Specifications Are (Preferably) Executable
by Norbert E. Fuchs, Software Engineering Journal, September 1992

Series of symposiums 1980/90:
Programming Language Implementation and Logic Programming

Current status

Specifies semantics of Cypher 9+

- 100% compatibility to TCK test set
- reflects semantics ambiguity due to driving table order
- graph values support

Language extension: support for graph values (also known as ‘g-records', graph-records or graphlets) and operations on them:

- binary functions: gunion, gintersection, gminus
- aggregation functions: agunion, agintersection

As graph value generalizes node, relationship and path values, all new listed functions accept arguments of those types.

Basic Concepts

Cypher.PL basic concepts:

I. Query intermediate representation
II. Environment: Tables × Graphs
III. General schema of clause evaluation

One-slide tutorial of logic programming

```
% Logical connectives

?- (true;false),(false;true).
true.

?- false;false.
false.

% Unification

?- member(X,[1,2]).
X = 1 ;
X = 2.

% Rules

father(X,Y) :- parent(X,Y), man(X).
mother(X,Y) :- parent(X,Y), woman(X).
```

```
% Meta-programming
?- call(member(X,[1,2])). % member is callable term
X = 1 ;
X = 2.

% Partial application
?- call(member(X),[1,2]).
X = 1 ;
X = 2.

?- call(member,X,[1,2]).
X = 1 ;
X = 2.

%Multiple solutions
?-  findall(X,member(X,[1,2]),Xs).
Xs = [1, 2].

% Lambda expressions
?- maplist([X]>>between(1, 5, X),[1,2,3,5]).
true.
```

CYPHER.PL

Intermediate Representation: just a tree

```
match p1=(a:A),p2=(b:B)
```

```
                                   singleQuery
                                        |
                                       LIST
                                        |
                                      clause
                                        |
                                      match
     ___________________________________|___________________________________
    /                                   |                                   \
no_modifier                          pattern                             no_where
                                        |
                                      LIST     or set?
                    ____________________|____________________
                   /                                         \
            patternElement                              patternElement
          _________|_________                         _________|_________
         /                   \                       /                   \
     variable               LIST                 variable               LIST
        |                     |                     |                     |
   symbolicName          nodePattern           symbolicName          nodePattern
        |            _________|_________            |            _________|_________
        |           /         |         \           |           /         |         \
       p1      variable   nodeLabels  properties   p2      variable   nodeLabels  properties
                  |           |           |                    |           |           |
            symbolicName    LIST     mapLiteral          symbolicName    LIST     mapLiteral
                  |           |           |                    |           |           |
                  a           A          []                    b           B          []
```

Syntax-sugar-free query

Base for semantics definition

Machine-oriented

- *Verbose*
- *Explicit*
- *Unambiguous*

Planner-friendly

- *Minimal ordering constraints*
- *Unique variable names*
- *Human-friendly*

Mainly for debugging, not a primary goal

IR.pl: just a tree definition

```
single_query(singleQuery(Clauses)) :- maplist(clause,Clauses).

%%%%% Clauses %%%%%
clause(clause(Clause)) :- match(Clause).
%...
clause(clause(Clause)) :- with(Clause).
clause(clause(Clause)) :- return(Clause).

%%%%% Match %%%%%

match(Clause)
:-
Clause = match(MatchModifier,Pattern,Where),
match_modifier(MatchModifier),
pattern(Pattern),
where(Where).

match_modifier(no_modifier).
match_modifier(optional).
%match_modifier(mandatory). %in the future

pattern(pattern(PatternElements)) :-
maplist(pattern_element,PatternElements).

pattern_element(patternElement(Variable,Patterns)) :-
variable(Variable),
patterns(Patterns).

node_pattern(nodePattern(Variable,NodeLabels,Properties)) :-
variable(Variable),
node_labels(NodeLabels),
properties(Properties).

node_labels(nodeLabels(Labels)) :- maplist(name,Labels).
```

```
relationship_pattern(relationshipPattern(Variable
                                        ,Direction
                                        ,RelationshipTypes
                                        ,RelationshipRange
                                        ,Properties))
:-
variable(Variable),
direction(Direction),
relationship_types(RelationshipTypes),
relationship_range(RelationshipRange),
properties(Properties).

direction(direction(left)).
direction(direction(right)).
direction(direction(both)).

relationship_types(relationshipTypes(Types)) :- maplist(name,Types).

relationship_range(relationshipRange(one_one)).
relationship_range(relationshipRange(L,U)) :-
(L=unlimited;integer(L)),%L >= 0,
(U=unlimited;integer(U)) %U >= 0
.
properties(properties(mapLiteral(Properties))) :-
maplist([(propertyKeyName(PropertyName),Expression)]>>(schema_name(PropertyName),expression(Expression)),
        Properties).

where(where(Expression)) :- expression(Expression).
where(no_where).
```

Environment: driving table…

```
create p=(a:A {int: 1, string: 'S', boolean : true, float: 1.1 })-[r:RELTYPE]->(b)
return p, [a,r] as l, {b : a.boolean, f : a.float, n : a.n} as m, gunion(p,p) as g
```

```
                                                                                     LIST
                                                                                       |
                                                                                       |
                                                                                   tablerow
                                                                                       |
                                                                                       |
                                                                                     LIST
                _______________________________________________________________________|________________________________________________________________________
               /                                              |                                         |                                                       \
             bind                                           bind                                     bind                                                     bind
    ___________|____________                           _______|______                       ____________|___________                                ____________|____________
   /                        \                         /              \                      /                       \                              /                         \
   p                   cypher_path                    l          cypher_list                m                 cypher_map                           g                  cypher_graphlet
                             |                                        |                                              |                                            ____________|__________
                             |                                        |                                              |                                            /                      \
                          LIST                                      LIST                                          LIST                                         LIST                    LIST
            _________________|_______________                  _______|________                     _________________|_________________                     ______|_____                  |
            /                |               \                /                \                    /                |               \                    /            \                 |
      cypher_node  cypher_relationship  cypher_node      cypher_node  cypher_relationship           ,                 ,                ,              cypher_node  cypher_node  cypher_relationship
            |                |                |               |                 |               ____|___          ____|___          ___|__                 |             |                |
            |                |                |               |                 |              /        \         /       \        /      \                |             |                |
          143              145              144             143              145              b cypher_boolean   f  cypher_float  n  cypher_null         143          144              145
                                                                                                         |                 |
                                                                                                         |                 |
                                                                                                      TRUE              1.1
```

Environment: driving table…

```
is_row_of_driving_table(tablerow(TableRow)) :-
is_list(TableRow),
maplist([bind(Name,Value)]>>(name(Name),cypher_value(Value)),TableRow).

    %primitives:
    %           cypher_null,
    %           cypher_string(Value),
    %           cypher_integer(Value),
    %           cypher_float(Value),
    %           cypher_boolean(Value),
    %entities:
    %           cypher_node(NodeId),
    %           cypher_relationship(RelationshipId),
    %           cypher_path(NodesRelationshipsAlernatingList),
    %           extended with cypher_graphlet(CypherNodesList,CypherRelationshipsList)
    %structures:
    %            cypher_list(CypherValuesList),
    %           cypher_map(NamesCypherValuesPairsList)
```

Environment: ... and property graph

```
                                              LIST
 _______________________________________________|_______________________________________________
/     |     |        |          |          |             |             |             |            \
node  node  label  relationship  type     property       property       property      property      property
|     |    _|     ____|___     __|__    _____|______    ____|_____     ____|_____     _____|_____     _____|_____
|     |   /  \   /    |    \   /    \   /     |      \   /    |    \   /    |    \   /    |     \     /    |     \
137   138 137 A  137  139  138 139 RELTYPE 137 boolean cypher_boolean 137 float cypher_float 137 int cypher_integer 137 list cypher_list 137 string cypher_string
                                                          |                   |                   |                   |                      |
                                                          |                   |                   |                   |                      |
                                                         TRUE                1.1                  1                 LIST                     S
                                                                                                                ______|______
                                                                                                               /             \
                                                                                                     cypher_integer   cypher_integer
                                                                                                           |                |
                                                                                                           |                |
                                                                                                           1                2
```

```
%read (matching) "api"
node(NodeId,Graph) :- member(node(NodeId),Graph).
relationship(NodeStartId,RelationId,NodeEndId,Graph) :- member(relationship(NodeStartId,RelationId,NodeEndId),Graph).
property(NorRId,Key,Value,Graph) :- member(property(NorRId,Key,Value),Graph).
label(NodeId,Label,Graph) :- member(label(NodeId,Label),Graph).
type(RelationshipId,Type,Graph) :- member(type(RelationshipId,Type),Graph).

%write "api"
create_node(NodeId,Graph,ResultGraph) :-....
delete_node(NodeId,Graph,ResultGraph) :-....

set_label(NodeId,LabelName,Graph,ResultGraph) :-....
remove_label(NodeId,LabelName,Graph,ResultGraph) :-....

create_relationship(NodeStartId,NodeEndId,RelationshipType,RelationshipId,Graph,ResultGraph) :-....
delete_relationship(RelationshipId,Graph,ResultGraph) :-....

set_property(Id,Key,Value,Graph,ResultGraph) :-....
remove_property(Id,Key,Graph,ResultGraph) :-....
```

General schema of clause evaluation

```
eval_single_query(Graph,
                  singleQuery(Clauses),
                  environment(ResultTable,ResultGraph))
:-
foldl(eval_clause,
      Clauses,
      environment([tablerow([])],Graph),
      environment(ResultTable,ResultGraph)).


eval_clause(clause(Clause), %clause(match(MatchModifier,Pattern,Where))
                             %clause(with(WithModifier,ReturnBody,Where))
                             %clause(unwind(Expression,Variable))
                             %...
            environment(Table,Graph),
            environment(ResultTable,ResultGraph))
:-
%particular clause semantics definition
```

CIR’s with Cypher.PL’s contribution

- #219 Grouping key selection for aggregating subqueries
- #264 Expressions allowed in WHERE subclause of WITH clause with aggregations

J. Posiadała, P. Susicki *Semantics of implied group by clause with mixed expressions. A simple use case of Cypher.PL – an executable specification of Cypher query language.* (withdrawn from GraphSM 2018 due to date conflict).

- #263 Node/relationship accessibility after DELETE operations
- #315 Ambiguity in parsing [value in list]
- <u>#296 Resolving MERGE being dependant on evaluation order</u>

As the source of ambiguity in openCyper is generally reducing semantics of unordered bag to semantics of ordered list, now in Cypher.PL driving table has no order.

#296 Ambiguity due to evaluation order

```
create
  ({num : 1}),({num : 2})
with
  *
match
  (n)
return
  (collect(n)[1]).num as nn


+--------+
|   nn   |
+--------+
|    1   |
+--------+


+--------+
|   nn   |
+--------+
|    2   |
+--------+
```

```
CREATE
  (:A{x:1}), (:A{x:2}),(:B)
WITH
  *
MATCH
  (a:A), (b:B)
SET
  b.x = a.x
return
  b.x
+-----+
|  b  |
+-----+
|  1  |
+-----+
|  1  |
+-----+


+-----+
|  b  |
+-----+
|  2  |
+-----+
|  2  |
+-----+
```

```
create
   (a {num: 1})-[r1:T]->(b {num: 2})-[r2:T]->(c {num: 3})
match
   (x)-[z]->(y)
set
   x.num = maximum([(k)-->(l) | l.num]) + 10
return
   x.num as xnum  //ambiugous result


+--------+
|  xnum  |
+--------+
|   23   |
+--------+
|   13   |
+--------+


+--------+
|  xnum  |
+--------+
|   13   |
+--------+
|   13   |
+--------+
```

Ambiguity handling in Cypher.PL

```
eval_clause(clause(Clause),environment(Table,Graph), environment(ResultTable,ResultGraph))
:-
scall(set(Table),
      eval_clause_(Clause,Graph),
      environment_eq,
      environment(ResultTable,ResultGraph)).

scall(set(Set),Goal,EqGoal,Solution) :-
findall(PermutationSolution
      ,(permutation_(Set,Permutation),call(Goal,Permutation,PermutationSolution))
      ,PermutationSolutions),
gr_by(EqGoal,PermutationSolutions,SolutionsGroups),
member([Solution|_],SolutionsGroups).

eval_clause_(Clause,Graph,Table,environment(ResultTable,ResultGraph)) :- %particular clause semantics definition

environment_eq(environment(Table1,Graph1),environment(Table2,Graph2))
:-
permutation(Table1,Table2),
isomorphic(Graph1,Graph2).
```

Intuitiveness (1/3)

boggle commented on 2 Feb • edited ▾ | Contributor

That makes me wonder if it might be (conceptually possible) to express `MERGE` using subquery operators like this:

```
MATCH (a)
MATCH {
  MATCH {
    MATCH (a)-[-[:X]->(m {prop: n.prop})
    RETURN n, m
  }
  OTHERWISE // query-level xor that has been discussed in the past
  MATCH {
    CREATE (a)-[:X]->(m {:prop: n.prop}
    RETURN n, m
  }
}
```

This shows where the problem is: It still would create duplicates and the only way to emulate `MERGE` that I could see would be a graph-level squashing operation of similarly looking entities. Event that would still not be the same, giving a real argument why `MERGE` is a core feature.

Intuitiveness (2/3)

```
eval_merge(merge(patternElement(Variable,Patterns), MergeActions),
           Graph,
           TableRow,
           environment(ResultMatchTable,ResultMergeGraph))
:-
eval_clause(clause(match(no_modifier,pattern([patternElement(Variable,Patterns)]),no_where)),
            environment([TableRow],Graph),
            environment(ResultMatchTable,Graph)),
not(ResultMatchTable = []),
!,
convlist([onMatch(Set),Set]>>true, MergeActions,OnMatchActions),
foldl(eval_merge_actions,
      OnMatchActions,
      environment(ResultMatchTable,Graph),
      environment(ResultMatchTable,ResultMergeGraph)).
```

Intuitiveness (3/3)

```
eval_merge(merge(patternElement(Variable,Patterns), MergeActions),
           Graph,
           TableRow,
           environment(ResultCreateTable,ResultMergeGraph))
:-
eval_clause(clause(create(pattern([patternElement(Variable,Patterns)]))),
            environment([TableRow],Graph),
            environment(ResultCreateTable,ResultCreateGraph)),
convlist([onCreate(Set),Set]>>true, MergeActions,OnCreateActions),
foldl(eval_merge_actions,
      OnCreateActions,
      environment(ResultCreateTable,ResultCreateGraph),
      environment(ResultCreateTable,ResultMergeGraph)).
```

Future directions

Cypher 9:

- examining TCK test set in ambiguity sensitive mode
- define rules of errors rising as explicitly as possible with preservation of distinction between 'compile time and 'runtime'
- to specify node/relationship accessibility after DELETE operation

Cypher 10/CAPS draft specification including:

- multiple graphs family features
- RPQ
- configurable morphism of pattern matching

Other languages: G-Core, GQL

- executable semantics

Current version is accessible to experiment in console mode:

```
$ ssh cypherpl/cypherpl@185.25.216.85
```

Bibliography

[1] M. A. AlTurki, J. Meseguer, **Executable rewriting logic semantics of Orc and formal analysis of Orc programs**, Journal of Logical and Algebraic Methods in Programming, Volume 84, Issue 4, Pages 505-533, 2015,

[2] K. R. Apt. **From Logic Programming to Prolog.** Prentice-Hall, Inc., Upper Saddle River, NJ, USA. 1996

[3] C. Ellison and G. Rosu. **An executable formal semantics of C with applications**. SIGPLAN Not. 47, 1 533-544, (January 2012).

[4] M. P. J. Fromherz, **A Survey of Executable Specification Methodologies**,Technical Report 89.05, Department of Computer Science, University of Zurich

[5] Norbert E. Fuchs, **Specifications Are (Preferably) Executable**, Software Engineering Journal, September 1992

[6] International Organization for Standardization. **ISO/IEC 9075:2016: Information technology – Database languages – SQL**, 2016.

[7] G. Gupta, E. Pontelli, **Specification, Implementation, and Verification of Domain Specific Languages: A Logic Programming-Based Approach**. In: Kakas A.C., Sadri F. (eds) Computational Logic: Logic Programming and Beyond. Lecture Notesin Computer Science, vol 2407. Springer, Berlin, Heidelberg, 2002

[8] R. A. Kowalski, **The relation between logic programming and logic specification**, in: C. A. R. Hoare, J. C. Shepherdson (Eds.), Mathematical Logic and Programming Languages, Prentice-Hall International, 1985

[9] Neo Technlogy Inc. **openCypher project**, http://www.opencypher.org/ 2018.01.01

[10] U. Nilsson and J. Maluszynski. **Logic, Programming, and PROLOG** (2nd ed.). John Wiley & Sons, Inc., New York, NY, USA. 1995.

[11] I. Robinson, J. Webber, and E. Eifrem. **Graph Databases**. O'Reilly Media, Inc. 2013

[12] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager, **SWI-Prolog. Theory and Practice of Logic Programming**, 12(1-2), 67-96. doi:10.1017/S1471068411000494, 2012.