Deeper Dive into Cypher.PL

Executable semantics of Cypher 9
written in logic

{jan.posiadala,pawel.susicki}@gmail.com
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
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 language
- 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.
Cypher.PL basic concepts:

I. Query intermediate representation

II. Environment: Tables × Graphs

III. General schema of clause evaluation
% 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.
Intermediate Representation: just a tree

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

match p1=(a:A), p2=(b:B)
single_query(singleQuery(Clauses)) :- maplist(clause,Clauses).

% Clauses
clause(clause(Clauses)) :- match(Clauses).

clause(clause(Clauses)) :- with(Clauses).
clause(clause(Clauses)) :- return(Clauses).

% Match
match(Clauses) :-
  match(Clauses,Clause),
  pattern(Pattern),
  where(Where).

match(no_modifier).
match(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(Props))) :-
  maplist([mapKeyExpr(PropertyName,Expression)>>schema_name(PropertyName,expression(Expression)),
           Properties).

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

Environment: driving table...
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(NodesRelationshipsAlternatingList),
% extended with cypher_graphlet(CypherNodesList,CypherRelationshipsList)
%structures:
% cypher_list(CypherValuesList),
% cypher_map(NamesCypherValuesPairsList)
Environment: ... and property graph
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]
● #296 Resolving MERGE being dependant on evaluation order

As the source of ambiguity in openCypher 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

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

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

```
create
  (a {num : 1})-[r1:T]->(b {num : 2})-[r2:T]->(c {num : 3})
match
  (x)-[z]->(y)
set
  x.num = maximum(((k)-->(1) | l.num]) + 10
return
  x.num as xnum  //ambiguous result
```
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).
That makes me wonder if it might be (conceptually possible) to express `MERGE` using subquery operators like this:

```cypher
MATCH (a)
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.
eval_merge(merge(patternElement(Variable,Patterns), MergeActions),
   Graph,
   TableRow,
   environment([ResultMatchTable,ResultMergeGraph]))
:-
eval_clause(clause(match(no_modifer,pattern([[patternElement(Variable,Patterns)]],no_where)),
   environment([[TableRow],Graph]),
   environment([ResultMatchTable,Graph])),
not(ResultMatchTable = []),
!,
convlist([[onMatch(\[\]),\[\]>>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],
    fold1(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