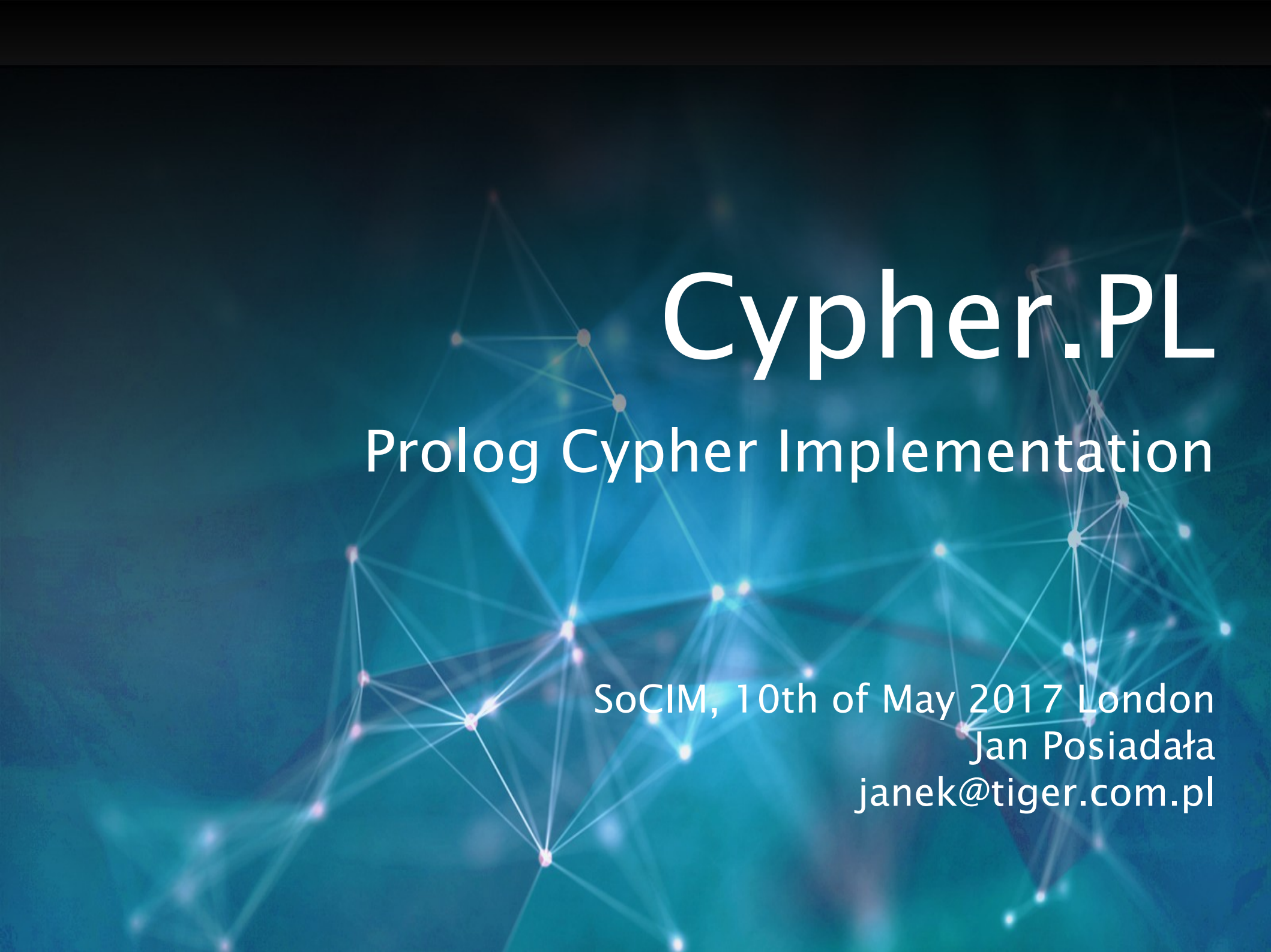

Cypher.PL
Prolog Cypher Implementation
SoCIM, 10th of May 2017 London
Jan Posiadała
janek@tiger.com.pl

Prolog Implementation

Cypher implementation in SWI-Prolog:

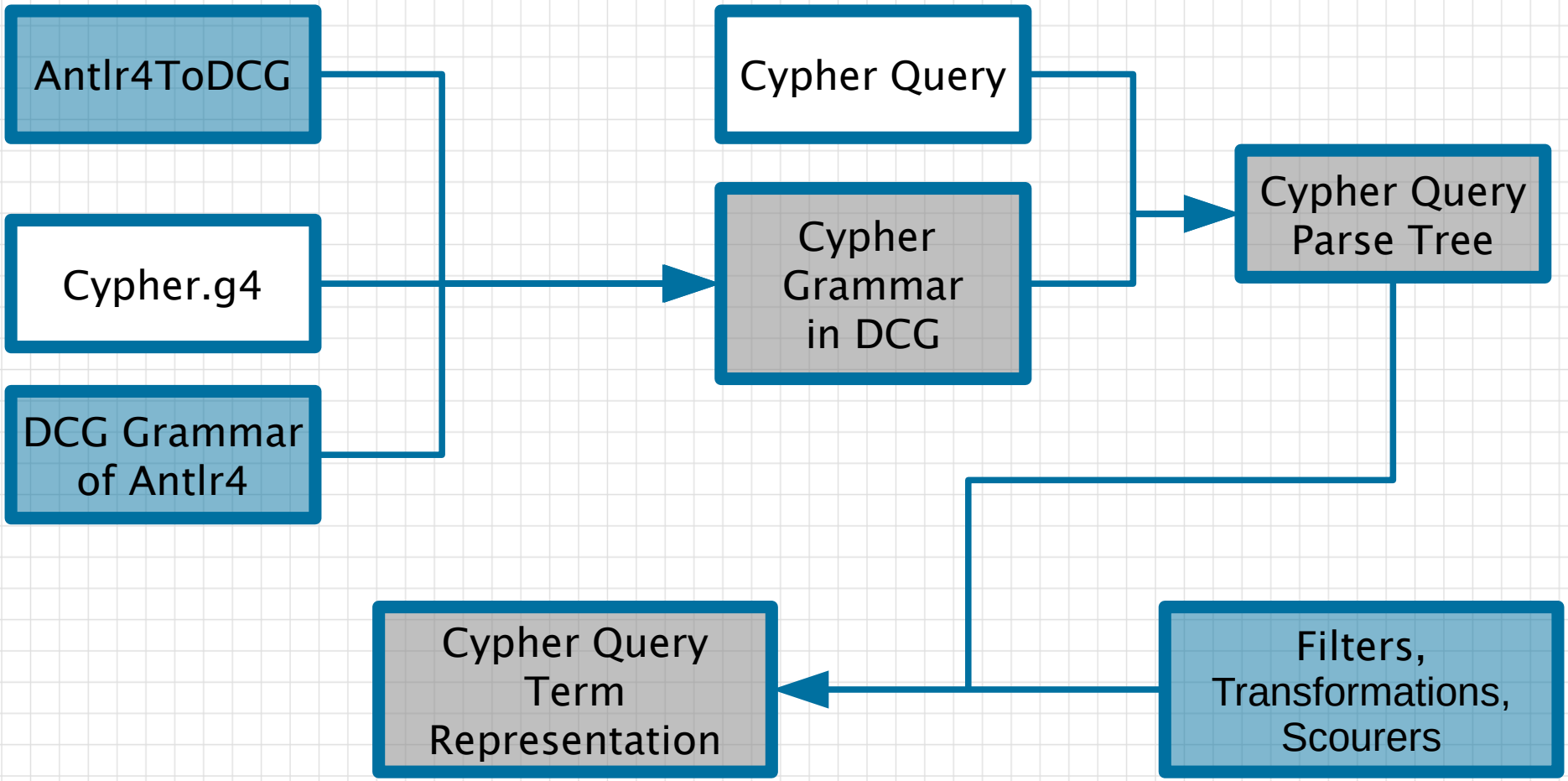
- formal implementation...
- ...or rather executable specification
- as close to the semantics as possible
- as far from the implementation issues as possible
- tool for designing, verification, validation

Why Prolog?

Prolog's enticements:

- built-in unification...
- ...which is more general than pattern matching
- super-native data (structures) representation
- evident ambiguity
- easy constraint verification
- DCG: notation for grammars
- meta-programming

openCypher To Cypher.PL



Graph Representation

Database as facts:

```
node(NodeId).

relationship(NodeStartId,RelationshipId,NodeEndId).

property(NorRId,Key,Value). %Value = cypherType(RawValue)

typeOrLabel(NorRId,TypeOrLabels). % always list
```

Graph Representation

Neo4j property graph via Cypher to Prolog facts.

```
match (a)-[r]->(b)
return 'relationship(' + id(a) + ',' + id(r) + ',' + id(b) + ').' as fact
union
match (a)
return 'node(' + id(a) + ').' as fact
union
match (m)
unwind keys(m) as key
return 'property(' + id(m) + ',\'' + key + '\',\'' + m[key] + '\').'  as fact
//almost: no type extraction
union
match ()-[m]->()
unwind keys(m) as key
return 'property(' + id(m) + ',\'' + key + '\',\'' + m[key] + '\').' as fact
//almost: no type extraction
union
match (m)
with reduce(s = "", x IN labels(m) | s  + ',\'' + x + '\'') as labels, m as m
return 'typeOrLabel(' + id(m) + ',' + substring(labels,1,size(labels) - 1) + ').' as fact
union
match ()-[m]->()
return 'typeOrLabel(' + id(m) + ',[\'' + type(m) + '\']).' as fact
```

Query Intermediate Representation

Prolog Term

```
OPTIONAL MATCH (a:Label1:Label1 {x:1,y:2})-[r]->(b),(c)-[r:Type*1..7]-(d)
WHERE true
```

```
match(OPTIONAL,pattern([patternPart(patternElement([nodePattern(variable(symbolicName(a)),nodeLabels([nodeLabe
l(labelName(symbolicName(Label1))),nodeLabel(labelName(symbolicName(Label2))))]),properties(mapLiteral([(proper
tyKeyName(symoolicName(x)),expression(atom(literal(numberLiteral(1))))),
(propertyKeyName(symbolicName(y)),expression(atom(literal(numberLiteral(2))))))])))),relationshipPattern(relatio
nshipDetail(variable(symbolicName(r)),relationshipTypes([]),relationshipRange(empty_one_one),properties(mapLit
eral([]))),right),nodePattern(variable(symbolicName(b)),nodeLabels([]),properties(mapLiteral([])))])),patternP
art(patternElement([nodePattern(variable(symbolicName(c)),nodeLabels([]),properties(mapLiteral([]))),relations
hipPattern(relationshipDetail(variable(symbolicName(r)),relationshipTypes([relTypeName(symbolicName(Type))]),r
elationshipRange(1,7),properties(mapLiteral([])),both),nodePattern(variable(symbolicName(d)),nodeLabels([]),p
roperties(mapLiteral([])))])]),where(expression(atom(literal(booleanLiteral(TRUE))))))
```

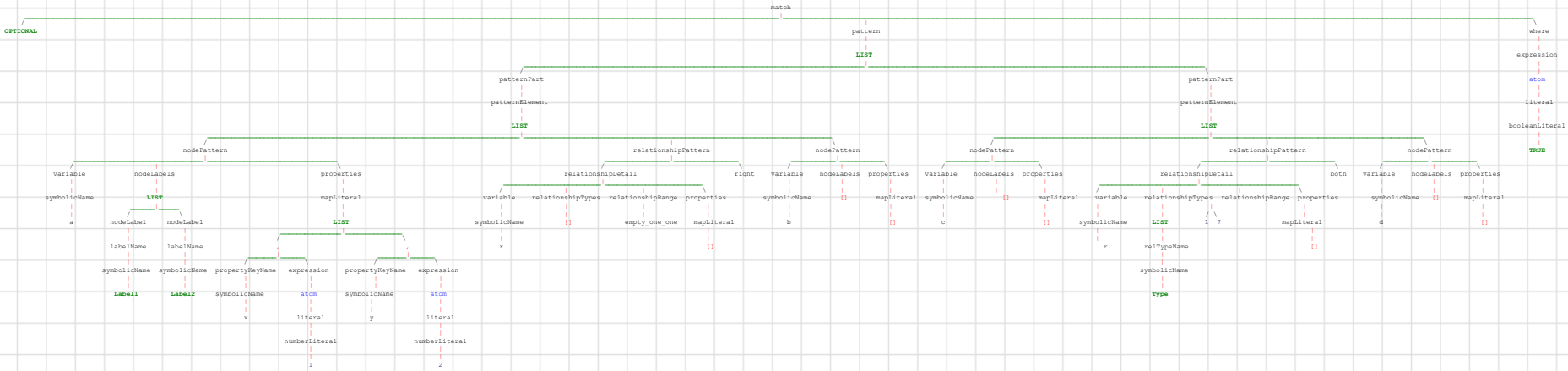
Machine-oriented version
- Verbose
- Explicit
- Unambiguous

Planner-friendly
- Minimal ordering constraints
- Unique variable names

* **orange** is stolen from Stefan's FoCIM slides

Query Intermediate Representation

IR: Prolog term (generically written)

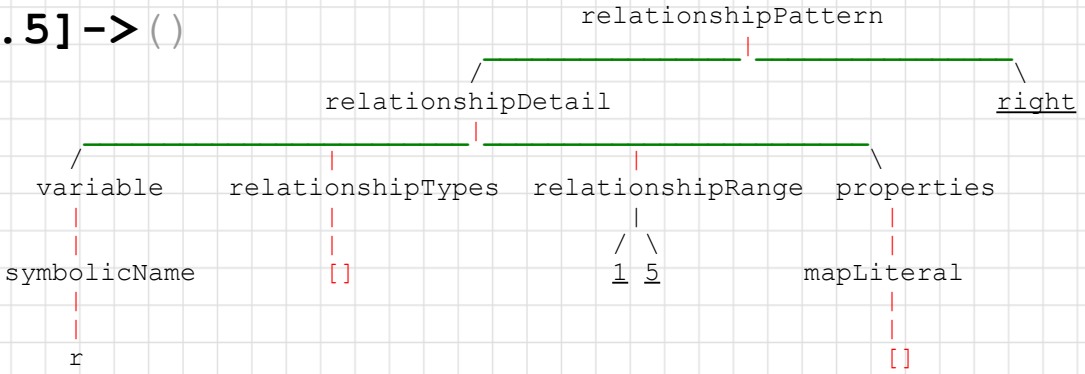


Human-friendly

- Mainly for debugging, not a primary goal

Executable Specification

```
()-[r*1..5]->()
```

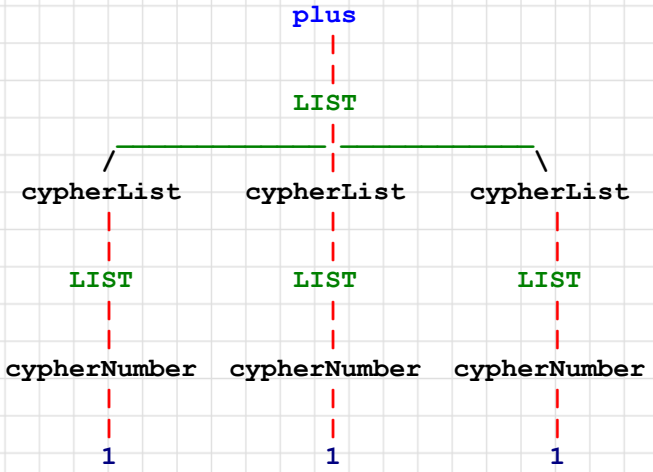


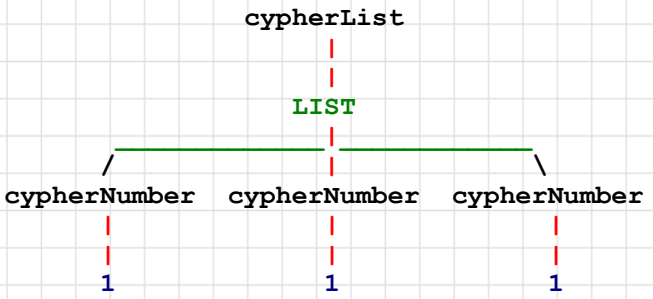

```
connects(_,X,guard,X)  :- !.
connects(both,X,Y,X)   :- connects(rigth,X,Y,X).
connects(both,X,Y,X)   :- connects(left,X,Y,X).
connects(rigth,X,Y,X)  :- relationship(_,X,N),relationship(N,Y,_).
connects(left,X,Y,X)   :- relationship(N,X,_),relationship(_,Y,N).

evalRelationshipPattern(LowerLimit,
                        UpperLimit,
                        Direction,
                        PatternPathRelationshipIds)
                        :-
                        findall(RelationshipId,
                                relationship(_,RelationshipId,_),
                                RelationshipIds),
                        csubset(RelationshipIds,PatternRelationshipIds),
                        length(PatternRelationshipIds,PatternRelationshipIdsLength),
                        (PatternRelationshipIdsLength =< UpperLimit;UpperLimit=nolim),
                        (PatternRelationshipIdsLength >= LowerLimit;LowerLimit=nolim),
                        permutation(PatternRelationshipIds,PatternPathRelationshipIds),
                        foldl(connects(Direction),PatternPathRelationshipIds,guard,_).
```

Semantic Exercises

```
[1] + [1] + [1]
```

```
plus([cypherList([cypherNumber(1)]), cypherList([cypherNumber(1)]), cypherList([cypherNumber(1)])])
```






```
%listPlus for lists
listPlus(Context,cypherList(X),cypherList(Y),cypherList(P)) :- append(X, Y, P).

eval(Context,plus(L),EE) :- maplist(eval(M),L,EL), %recurrent eval
                            foldl(listPlus(Context), EL, cypherList([]), EE),!.
```

Semantic Exercises

```
[1] + 1 + 1
```

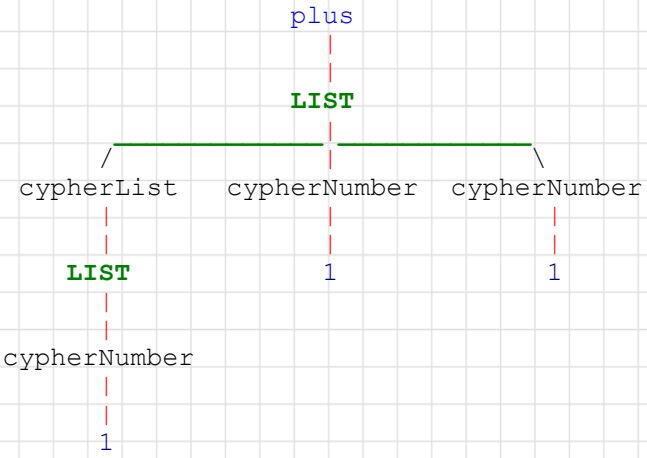
```
plus([cypherList([cypherNumber(1)]), cypherNumber(1), cypherNumber(1)])
```

```
%binary listPlus
eval(Context,plus([X,Y]),cypherList(P)) :- eval(Context,X,cypherList(XL)),
                                           eval(Context,Y,cypherList(YL)),
                                           append(XL, YL, P),!.

eval(Context,plus([X,Y]),cypherList(P)) :- eval(Context,X,cypherList(XL)),
                                           eval(Context,Y,cypherNumber(YN)),
                                           append(XL, [cypherNumber(YN)], P),!.

eval(Context,plus([X,Y]),cypherList(P)) :- eval(Context,X,cypherNumber(XN)),
                                           eval(Context,Y,cypherList(YL)),
                                           append([cypherNumber(XN)], YL, P),!.
```

Semantic Exercises

```
[1] + 1 + 1
```

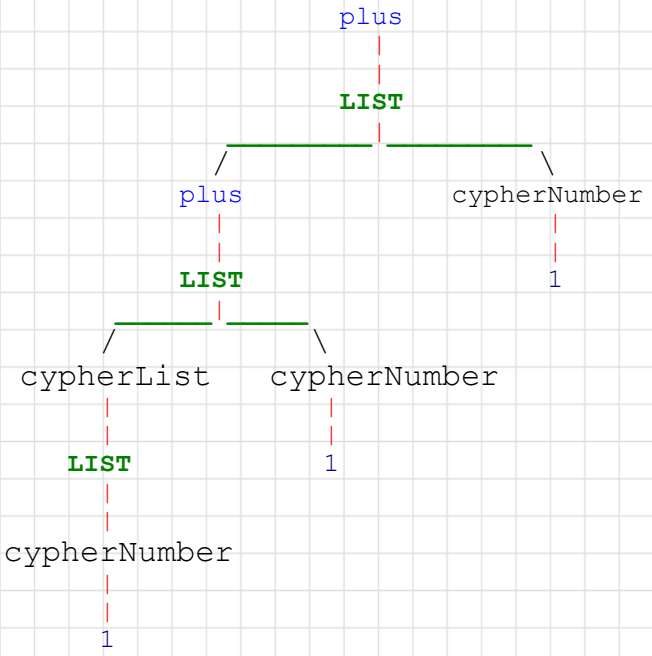
```
plus([cypherList([cypherNumber(1)]), cypherNumber(1), cypherNumber(1)])
```



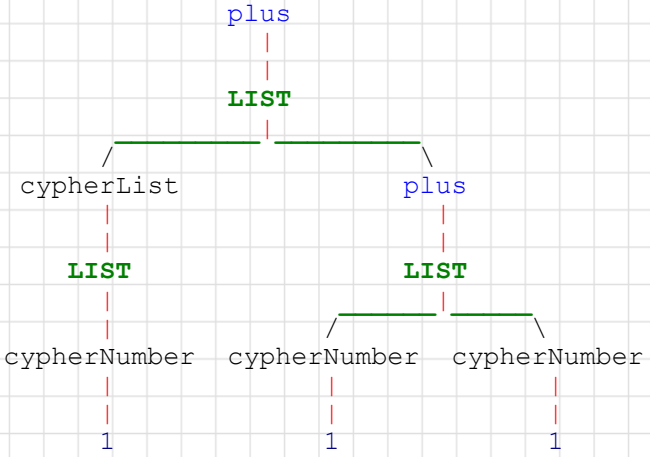

```
%plus on list to binary tree plus
list_to_tree(_,[X],X).
list_to_tree(TermName,[X,Y],Term) :- Term =.. [TermName,[X,Y]], !.
list_to_tree(TermName,L,Term) :- append(L1,L2,L),not(L1=[]),not(L2=[]),
                                 list_to_tree(TermName,L1,TermL1),
                                 list_to_tree(TermName,L2,TermL2),
                                 Term =.. [TermName,[TermL1,TermL2]].
```

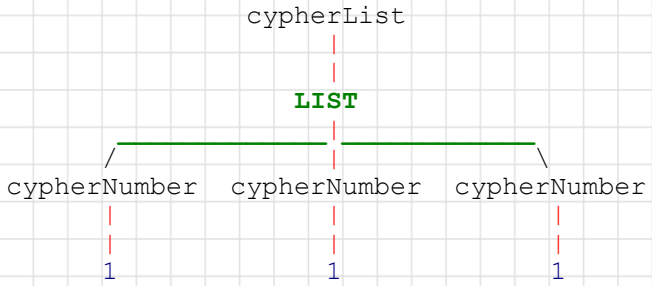
Semantic Exercises

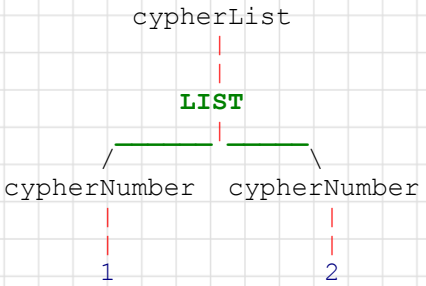
([1] + 1) + 1



[1] + (1 + 1)







Result of overloading of ”+” operator combined with implicit type conversion.

Executable Specification of Cypher

Definition:

Compact Representation of the Informational Content of a Query

Simple model for query planner

- grounded in property graph model
- easy formal treatment

Point of collaboration between implementers

- language agnostic
- engine agnostic
- discuss impact of language changes / extensions

Q & A