openCypher Specification

Gábor Szárnyas, József Marton

2017/03/03 08:59 GMT+0

Contents

1 Theoretical Foundations **15**
1.1 Introduction . . . 15
1.2 Preliminaries . . . 15
1.2.1 Property Graph Data Model . . . 15
1.3 Operators of Relational Algebra . . . 16
1.3.1 Nullary Operators . . . 16
1.3.2 Unary Operators . . . 17
1.3.3 Binary Operators . . . 17
1.3.4 Property Access . . . 18
1.3.5 Summary of Operators . . . 18
1.4 The openCypher Query Language . . . 18
1.5 Mapping openCypher Queries to Relational Graph Algebra . . . 19
1.6 Related Work . . . 20
1.7 Conclusion and Future Work . . . 22

2 Incremental Query Evaluation **23**
2.1 Overview of Rete Networks . . . 23
2.2 Input Nodes . . . 24
2.3 Unary Nodes . . . 24
2.4 Binary Nodes . . . 25
2.4.1 Natural Join Node . . . 25
2.4.2 Antijoin Node . . . 26
2.4.3 Left Outer Join Node . . . 27
2.4.4 Union Node . . . 27
2.5 Production Nodes . . . 27

A TCK Acceptance Tests **30**
A.1 AggregationAcceptance . . . 30
A.1.1 Support multiple divisions in aggregate function . . . 30
A.1.2 Support column renaming for aggregates as well . . . 31
A.1.3 Aggregates inside normal functions . . . 32
A.1.4 Handle aggregates inside non-aggregate expressions . . . 33
A.1.5 Count nodes . . . 33
A.1.6 Sort on aggregate function and normal property . . . 35
A.1.7 Aggregate on property . . . 36
A.1.8 Count non-null values . . . 37
A.1.9 Sum non-null values . . . 38
A.1.10 Handle aggregation on functions . . . 39
A.1.11 Distinct on unbound node . . . 40
A.1.12 Distinct on null . . . 41
A.1.13 Collect distinct nulls . . . 42
A.1.14 Collect distinct values mixed with nulls . . . 43
A.1.15 Aggregate on list values . . . 43
A.1.16 Aggregates with arithmetics . . . 44
A.1.17 Aggregates ordered by arithmetics . . . 45
A.1.18 Multiple aggregates on same variable . . . 47
A.1.19 Simple counting of nodes . . . 47

A.1.20 Aggregation of named paths . . . 49
A.1.21 Aggregation with 'min()' . . . 49
A.1.22 Handle subexpression in aggregation also occurring as standalone expression with nested aggregation in a literal map . . . 49
A.1.23 No overflow during summation . . . 50
A.1.24 Counting with loops . . . 51
A.1.25 'max()' should aggregate strings . . . 52
A.1.26 'min()' should aggregate strings . . . 52
A.2 ColumnNameAcceptance . . . 52
A.2.1 Keeping used expression 1 . . . 53
A.2.2 Keeping used expression 2 . . . 54
A.2.3 Keeping used expression 3 . . . 54
A.2.4 Keeping used expression 4 . . . 54
A.3 Comparability . . . 54
A.3.1 Comparing strings and integers using > in an AND'd predicate . . . 54
A.3.2 Comparing strings and integers using > in a OR'd predicate . . . 55
A.4 ComparisonOperatorAcceptance . . . 55
A.4.1 Handling numerical ranges 1 . . . 55
A.4.2 Handling numerical ranges 2 . . . 55
A.4.3 Handling numerical ranges 3 . . . 55
A.4.4 Handling numerical ranges 4 . . . 56
A.4.5 Handling string ranges 1 . . . 56
A.4.6 Handling string ranges 2 . . . 56
A.4.7 Handling string ranges 3 . . . 56
A.4.8 Handling string ranges 4 . . . 56
A.4.9 Handling empty range . . . 57
A.4.10 Handling long chains of operators . . . 57
A.5 Create . . . 57
A.6 CreateAcceptance . . . 57
A.7 DeleteAcceptance . . . 57
A.8 EqualsAcceptance . . . 57
A.8.1 Number-typed integer comparison . . . 57
A.8.2 Number-typed float comparison . . . 58
A.8.3 Any-typed string comparison . . . 58
A.8.4 Comparing nodes to nodes . . . 58
A.8.5 Comparing relationships to relationships . . . 59
A.9 ExpressionAcceptance . . . 59
A.9.1 IN should work with nested list subscripting . . . 59
A.9.2 IN should work with nested literal list subscripting . . . 60
A.9.3 IN should work with list slices . . . 61
A.9.4 IN should work with literal list slices . . . 62
A.9.5 Execute n[0 . . . 63
A.9.6 Execute n['name' . . . 63
A.9.7 Use dynamic property lookup based on parameters when there is no type information 64
A.9.8 Use dynamic property lookup based on parameters when there is rhs type information 66
A.9.9 Use collection lookup based on parameters when there is no type information . . . 67
A.9.10 Use collection lookup based on parameters when there is lhs type information . . . 68
A.9.11 Use collection lookup based on parameters when there is rhs type information . . . 70
A.10 FunctionsAcceptance . . . 71
A.10.1 Run coalesce . . . 71
A.10.2 Functions should return null if they get path containing unbound . . . 72
A.10.3 'split()' . . . 72
A.10.4 'properties()' on a node . . . 73
A.10.5 'properties()' on a relationship . . . 73
A.10.6 'properties()' on a map . . . 74
A.10.7 'properties()' on null . . . 75
A.10.8 'reverse()' . . . 75
A.10.9 'exists()' with dynamic property lookup . . . 76

A.10.10'percentileDisc()' failing in more involved query 76
A.10.11'type()' 76
A.10.12'type()' on two relationships 77
A.10.13'type()' on null relationship 78
A.10.14'type()' on mixed null and non-null relationships 80
A.10.15'type()' handling Any type 81
A.10.16'labels()' should accept type Any 82
A.10.17'labels()' failing on invalid arguments 83
A.10.18'exists()' is case insensitive 84
A.11 ValueHashJoinAcceptance 85
A.11.1 Find friends of others 85
A.11.2 Should only join when matching 87
A.12 KeysAcceptance 88
A.12.1 Using 'keys()' on a single node, non-empty result 88
A.12.2 Using 'keys()' on multiple nodes, non-empty result 88
A.12.3 Using 'keys()' on a single node, empty result 89
A.12.4 Using 'keys()' on an optionally matched node 89
A.12.5 Using 'keys()' on a relationship, non-empty result 89
A.12.6 Using 'keys()' on a relationship, empty result 89
A.12.7 Using 'keys()' on an optionally matched relationship 90
A.12.8 Using 'keys()' on a literal map 90
A.12.9 Using 'keys()' on a parameter map 90
A.13 LabelsAcceptance 91
A.13.1 Using 'labels()' in return clauses 91
A.14 LargeCreateQuery 92
A.15 LargeIntegerEquality 92
A.15.1 Does not lose precision 92
A.15.2 Handling inlined equality of large integer 93
A.15.3 Handling explicit equality of large integer 94
A.15.4 Handling inlined equality of large integer, non-equal values 95
A.15.5 Handling explicit equality of large integer, non-equal values 96
A.16 ListComprehension 97
A.16.1 Returning a list comprehension 97
A.16.2 Using a list comprehension in a WITH 97
A.16.3 Using a list comprehension in a WHERE 98
A.17 Literals 99
A.17.1 Return an integer 99
A.17.2 Return a float 100
A.17.3 Return a float in exponent form 100
A.17.4 Return a boolean 101
A.17.5 Return a single-quoted string 102
A.17.6 Return a double-quoted string 102
A.17.7 Return null 103
A.17.8 Return an empty list 104
A.17.9 Return a nonempty list 104
A.17.10Return an empty map 105
A.17.11Return a nonempty map 105
A.18 MatchAcceptance 106
A.18.1 Path query should return results in written order 106
A.18.2 Longer path query should return results in written order 106
A.18.3 Use multiple MATCH clauses to do a Cartesian product 107
A.18.4 Use params in pattern matching predicates 108
A.18.5 Filter out based on node prop name 108
A.18.6 Honour the column name for RETURN items 109
A.18.7 Filter based on rel prop name 110
A.18.8 Cope with shadowed variables 111
A.18.9 Get neighbours 111
A.18.10Get two related nodes 112

A.18.11 Get related to related to 113
A.18.12 Handle comparison between node properties 114
A.18.13 Return two subgraphs with bound undirected relationship 116
A.18.14 Return two subgraphs with bound undirected relationship and optional relationship 117
A.18.15 Rel type function works as expected 119
A.18.16 Walk alternative relationships 120
A.18.17 Handle OR in the WHERE clause 122
A.18.18 Return a simple path 123
A.18.19 Return a three node path 123
A.18.20 Do not return anything because path length does not match 123
A.18.21 Pass the path length test 124
A.18.22 Return relationships by fetching them from the path - starting from the end 124
A.18.23 Return relationships by fetching them from the path 124
A.18.24 Return relationships by collecting them as a list - wrong way 124
A.18.25 Return relationships by collecting them as a list - undirected 125
A.18.26 Return relationships by collecting them as a list 126
A.18.27 Return a var length path 128
A.18.28 Return a var length path of length zero 128
A.18.29 Return a named var length path of length zero 128
A.18.30 Accept skip zero 128
A.19 MatchAcceptance2 130
A.19.1 Do not return non-existent nodes 130
A.19.2 Do not return non-existent relationships 130
A.19.3 Do not fail when evaluating predicates with illegal operations if the AND'ed predicate evaluates to false 131
A.19.4 Do not fail when evaluating predicates with illegal operations if the OR'd predicate evaluates to true 133
A.19.5 Aggregation with named paths 133
A.19.6 Zero-length variable length pattern in the middle of the pattern 134
A.19.7 Simple variable length pattern 135
A.19.8 Variable length relationship without lower bound 136
A.19.9 Variable length relationship without bounds 137
A.19.10 Returning bound nodes that are not part of the pattern 137
A.19.11 Two bound nodes pointing to the same node 138
A.19.12 Three bound nodes pointing to the same node 140
A.19.13 Three bound nodes pointing to the same node with extra connections 142
A.19.14 MATCH with OPTIONAL MATCH in longer pattern 144
A.19.15 Optionally matching named paths 146
A.19.16 Optionally matching named paths with single and variable length patterns 146
A.19.17 Optionally matching named paths with variable length patterns 147
A.19.18 Matching variable length patterns from a bound node 147
A.19.19 Excluding connected nodes 147
A.19.20 Do not fail when predicates on optionally matched and missed nodes are invalid . 147
A.19.21 MATCH and OPTIONAL MATCH on same pattern 149
A.19.22 Matching using an undirected pattern 149
A.19.23 Matching all nodes 150
A.19.24 Comparing nodes for equality 150
A.19.25 Matching using self-referencing pattern returns no result 152
A.19.26 Variable length relationship in OPTIONAL MATCH 153
A.19.27 Matching using relationship predicate with multiples of the same type 155
A.19.28 ORDER BY with LIMIT 156
A.19.29 Simple node property predicate 158
A.19.30 Handling direction of named paths 159
A.19.31 Simple OPTIONAL MATCH on empty graph 160
A.19.32 OPTIONAL MATCH with previously bound nodes 161
A.19.33 'collect()' filtering nulls 162
A.19.34 Multiple anonymous nodes in a pattern 163
A.19.35 Matching a relationship pattern using a label predicate 163

A.19.36Matching a relationship pattern using a label predicate on both sides 164
A.19.37Matching nodes using multiple labels 165
A.19.38Returning label predicate expression 166
A.19.39Matching with many predicates and larger pattern 167
A.19.40Matching using a simple pattern with label predicate 167
A.19.41Matching disconnected patterns 168
A.19.42Non-optional matches should not return nulls 170
A.19.43Handling cyclic patterns 172
A.19.44Handling cyclic patterns when separated into two parts 173
A.19.45Handling fixed-length variable length pattern 174
A.19.46Matching from null nodes should return no results owing to finding no matches . . 175
A.19.47Matching from null nodes should return no results owing to matches being filtered out 177
A.19.48Optionally matching from null nodes should return null 179
A.19.49OPTIONAL MATCH returns null 181
A.19.50Zero-length named path 182
A.19.51Variable-length named path 183
A.19.52Matching with aggregation 183
A.19.53Matching using a relationship that is already bound 184
A.19.54Matching using a relationship that is already bound, in conjunction with aggregation186
A.19.55Matching using a relationship that is already bound, in conjunction with aggregation and ORDER BY 188
A.19.56Matching with LIMIT and optionally matching using a relationship that is already bound 191
A.19.57Matching with LIMIT and optionally matching using a relationship and node that are both already bound 192
A.19.58Matching with LIMIT, then matching again using a relationship and node that are both already bound along with an additional predicate 194
A.19.59Matching with LIMIT and predicates, then matching again using a relationship and node that are both already bound along with a duplicate predicate 196
A.19.60Matching twice with conflicting relationship types on same relationship 198
A.19.61Matching twice with duplicate relationship types on same relationship 200
A.19.62Matching relationships into a list and matching variable length using the list . . . 202
A.19.63Matching relationships into a list and matching variable length using the list, with bound nodes 204
A.19.64Matching relationships into a list and matching variable length using the list, with bound nodes, wrong direction 206
A.19.65Matching and optionally matching with bound nodes in reverse direction 208
A.19.66Matching and optionally matching with unbound nodes and equality predicate in reverse direction 210
A.19.67Matching and returning ordered results, with LIMIT 211
A.19.68Counting an empty graph 212
A.19.69Matching variable length pattern with property predicate 213
A.19.70Variable length pattern checking labels on endnodes 214
A.19.71Variable length pattern with label predicate on both sides 216
A.19.72Undirected named path 218
A.19.73Named path with WITH 218
A.19.74Named path with alternating directed/undirected relationships 219
A.19.75Named path with multiple alternating directed/undirected relationships 219
A.19.76Named path with undirected fixed variable length pattern 219
A.19.77Returning a node property value 219
A.19.78Returning a relationship property value 220
A.19.79Projecting nodes and relationships 221
A.19.80Missing node property should become null 222
A.19.81Missing relationship property should become null 223
A.19.82Returning multiple node property values 224
A.19.83Adding a property and a literal in projection 225
A.19.84Adding list properties in projection 226

A.19.85Variable length relationship variables are lists of relationships 227
A.19.86Variable length patterns and nulls 228
A.19.87Projecting a list of nodes and relationships 230
A.19.88Projecting a map of nodes and relationships 231
A.19.89Respecting direction when matching existing path 232
A.19.90Respecting direction when matching non-existent path 232
A.19.91Respecting direction when matching non-existent path with multiple directions . . 233
A.19.92Matching path with both directions should respect other directions 233
A.19.93Matching path with multiple bidirectional relationships 233
A.19.94Matching nodes with many labels 233
A.19.95Matching longer variable length paths 234
A.19.96Matching a self-loop 235
A.20 MatchingSelfRelationships 236
A.20.1 Undirected match in self-relationship graph 236
A.20.2 Undirected match in self-relationship graph, count 237
A.20.3 Undirected match of self-relationship in self-relationship graph 238
A.20.4 Undirected match of self-relationship in self-relationship graph, count 239
A.20.5 Undirected match on simple relationship graph 241
A.20.6 Undirected match on simple relationship graph, count 241
A.20.7 Directed match on self-relationship graph 243
A.20.8 Directed match on self-relationship graph, count 244
A.20.9 Directed match of self-relationship on self-relationship graph 245
A.20.10Directed match of self-relationship on self-relationship graph, count 246
A.20.11Counting undirected self-relationships in self-relationship graph 248
A.20.12Counting distinct undirected self-relationships in self-relationship graph 249
A.20.13Directed match of a simple relationship 250
A.20.14Directed match of a simple relationship, count 251
A.20.15Counting directed self-relationships 252
A.20.16Mixing directed and undirected pattern parts with self-relationship, simple 254
A.20.17Mixing directed and undirected pattern parts with self-relationship, count 255
A.20.18Mixing directed and undirected pattern parts with self-relationship, undirected . . 257
A.20.19Mixing directed and undirected pattern parts with self-relationship, undirected count259
A.21 MergeIntoAcceptance 260
A.22 MergeNodeAcceptance 260
A.23 MergeRelationshipAcceptance 260
A.24 MiscellaneousErrorAcceptance 260
A.25 NullAcceptance 261
A.26 OptionalMatch 261
A.26.1 Satisfies the open world assumption, relationships between same nodes 261
A.26.2 Satisfies the open world assumption, single relationship 262
A.26.3 Satisfies the open world assumption, relationships between different nodes 264
A.27 OptionalMatchAcceptance 266
A.27.1 Return null when no matches due to inline label predicate 266
A.27.2 Return null when no matches due to label predicate in WHERE 267
A.27.3 Respect predicates on the OPTIONAL MATCH 268
A.27.4 Returning label predicate on null node 269
A.27.5 MATCH after OPTIONAL MATCH 270
A.27.6 WITH after OPTIONAL MATCH 272
A.27.7 Named paths in optional matches 274
A.27.8 OPTIONAL MATCH and bound nodes 275
A.27.9 OPTIONAL MATCH with labels on the optional end node 276
A.27.10Named paths inside optional matches with node predicates 277
A.27.11Variable length optional relationships 278
A.27.12Variable length optional relationships with length predicates 279
A.27.13Optionally matching self-loops 281
A.27.14Optionally matching self-loops without matches 282
A.27.15Variable length optional relationships with bound nodes 283
A.27.16Variable length optional relationships with bound nodes, no matches 284

A.27.17Longer pattern with bound nodes 284
A.27.18Longer pattern with bound nodes without matches 286
A.27.19Handling correlated optional matches; first does not match implies second does not match 288
A.27.20Handling optional matches between optionally matched entities 290
A.27.21Handling optional matches between nulls 291
A.27.22OPTIONAL MATCH and 'collect()' 292
A.28 OrderByAcceptance 293
A.28.1 ORDER BY should return results in ascending order 293
A.28.2 ORDER BY DESC should return results in descending order 294
A.28.3 ORDER BY of a column introduced in RETURN should return salient results in ascending order 295
A.28.4 Renaming columns before ORDER BY should return results in ascending order . . 296
A.28.5 Handle projections with ORDER BY - GH no.4937 296
A.28.6 ORDER BY should order booleans in the expected order 296
A.28.7 ORDER BY DESC should order booleans in the expected order 296
A.28.8 ORDER BY should order strings in the expected order 297
A.28.9 ORDER BY DESC should order strings in the expected order 297
A.28.10ORDER BY should order ints in the expected order 297
A.28.11ORDER BY DESC should order ints in the expected order 297
A.28.12ORDER BY should order floats in the expected order 298
A.28.13ORDER BY DESC should order floats in the expected order 298
A.28.14Handle ORDER BY with LIMIT 1 298
A.28.15ORDER BY with LIMIT 0 should not generate errors 299
A.28.16ORDER BY with negative parameter for LIMIT should not generate errors 301
A.29 PatternComprehension 302
A.29.1 Pattern comprehension and ORDER BY 302
A.29.2 Returning a pattern comprehension 302
A.29.3 Returning a pattern comprehension with label predicate 303
A.29.4 Returning a pattern comprehension with bound nodes 303
A.29.5 Using a pattern comprehension in a WITH 303
A.29.6 Using a variable-length pattern comprehension in a WITH 303
A.29.7 Using pattern comprehension in RETURN 303
A.29.8 Aggregating on pattern comprehension 304
A.29.9 Using pattern comprehension to test existence 304
A.29.10Pattern comprehension inside list comprehension 304
A.29.11Get node degree via size of pattern comprehension 304
A.29.12Get node degree via size of pattern comprehension that specifies a relationship type 304
A.29.13Get node degree via size of pattern comprehension that specifies multiple relationship types 305
A.29.14Introducing new node variable in pattern comprehension 305
A.29.15Introducing new relationship variable in pattern comprehension 305
A.30 RemoveAcceptance 305
A.31 ReturnAcceptanceTest 305
A.31.1 Allow addition 306
A.31.2 Limit to two hits 307
A.31.3 Start the result from the second row 308
A.31.4 Start the result from the second row by param 309
A.31.5 Get rows in the middle 310
A.31.6 Get rows in the middle by param 312
A.31.7 Sort on aggregated function 313
A.31.8 Support sort and distinct 315
A.31.9 Support column renaming 316
A.31.10Support ordering by a property after being distinct-ified 317
A.31.11Arithmetic precedence test 319
A.31.12Arithmetic precedence with parenthesis test 320
A.31.13Count star should count everything in scope 320
A.31.14Absolute function 322

A.31.15Return collection size 323
A.32 ReturnAcceptance2 323
A.32.1 Accept valid Unicode literal 324
A.32.2 LIMIT 0 should return an empty result 324
A.32.3 Ordering with aggregation 325
A.32.4 DISTINCT on nullable values 327
A.32.5 Return all variables 328
A.32.6 'sqrt()' returning float values 328
A.32.7 Arithmetic expressions inside aggregation 328
A.32.8 Matching and disregarding output, then matching again 329
A.32.9 Returning a list property 330
A.32.10Returning a projected map 331
A.32.11Returning an expression 332
A.32.12Concatenating and returning the size of literal lists 333
A.32.13Limiting amount of rows when there are fewer left than the LIMIT argument . . . 333
A.32.14'substring()' with default second argument 334
A.32.15Returning all variables with ordering 335
A.32.16Using aliased DISTINCT expression in ORDER BY 336
A.32.17Returned columns do not change from using ORDER BY 338
A.32.18Arithmetic expressions should propagate null values 340
A.32.19Indexing into nested literal lists 340
A.32.20Aliasing expressions 340
A.32.21Projecting an arithmetic expression with aggregation 341
A.32.22Aggregating by a list property has a correct definition of equality 342
A.32.23Reusing variable names 343
A.32.24Concatenating lists of same type 343
A.32.25Appending lists of same type 343
A.32.26DISTINCT inside aggregation should work with lists in maps 343
A.32.27Handling DISTINCT with lists in maps 344
A.32.28DISTINCT inside aggregation should work with nested lists in maps 346
A.32.29DISTINCT inside aggregation should work with nested lists of maps in maps . . . 348
A.33 SemanticErrorAcceptance 349
A.33.1 Handling property access on the Any type 349
A.33.2 Bad arguments for 'range()' 349
A.34 SetAcceptance 350
A.35 SkipLimitAcceptanceTest 350
A.35.1 SKIP with an expression that does not depend on variables 350
A.35.2 LIMIT with an expression that does not depend on variables 350
A.36 StartingPointAcceptance 351
A.36.1 Find all nodes 351
A.36.2 Find labelled nodes 352
A.36.3 Find nodes by property 352
A.37 StartsWithAcceptance 353
A.37.1 Finding exact matches 354
A.37.2 Finding beginning of string 355
A.37.3 Finding end of string 1 356
A.37.4 Finding end of string 2 357
A.37.5 Finding middle of string 358
A.37.6 Finding the empty string 359
A.37.7 Finding when the middle is known 360
A.37.8 Finding strings starting with whitespace 361
A.37.9 Finding strings starting with newline 362
A.37.10Finding strings ending with newline 364
A.37.11Finding strings ending with whitespace 365
A.37.12Finding strings containing whitespace 366
A.37.13Finding strings containing newline 367
A.37.14No string starts with null 368
A.37.15No string does not start with null 370

A.37.16 No string ends with null 371
A.37.17 No string does not end with null 372
A.37.18 No string contains null 373
A.37.19 No string does not contain null 374
A.37.20 Combining string operators 375
A.37.21 NOT with CONTAINS 376
A.37.22 Handling non-string operands for STARTS WITH 377
A.37.23 Handling non-string operands for CONTAINS 378
A.37.24 Handling non-string operands for ENDS WITH 378
A.38 SyntaxErrorAcceptance 378
A.39 TernaryLogicAcceptanceTest 378
A.39.1 The inverse of a null is a null 378
A.39.2 A literal null IS null 379
A.39.3 A literal null is not IS NOT null 380
A.39.4 It is unknown - i.e. null - if a null is equal to a null 380
A.39.5 It is unknown - i.e. null - if a null is not equal to a null 381
A.40 TriadicSelection 381
A.40.1 Handling triadic friend of a friend 381
A.40.2 Handling triadic friend of a friend that is not a friend 382
A.40.3 Handling triadic friend of a friend that is not a friend with different relationship type 382
A.40.4 Handling triadic friend of a friend that is not a friend with superset of relationship type 383
A.40.5 Handling triadic friend of a friend that is not a friend with implicit subset of relationship type 383
A.40.6 Handling triadic friend of a friend that is not a friend with explicit subset of relationship type 383
A.40.7 Handling triadic friend of a friend that is not a friend with same labels 383
A.40.8 Handling triadic friend of a friend that is not a friend with different labels 384
A.40.9 Handling triadic friend of a friend that is not a friend with implicit subset of labels 384
A.40.10 Handling triadic friend of a friend that is not a friend with implicit superset of labels 384
A.40.11 Handling triadic friend of a friend that is a friend 384
A.40.12 Handling triadic friend of a friend that is a friend with different relationship type . 385
A.40.13 Handling triadic friend of a friend that is a friend with superset of relationship type 385
A.40.14 Handling triadic friend of a friend that is a friend with implicit subset of relationship type 385
A.40.15 Handling triadic friend of a friend that is a friend with explicit subset of relationship type 386
A.40.16 Handling triadic friend of a friend that is a friend with same labels 386
A.40.17 Handling triadic friend of a friend that is a friend with different labels 386
A.40.18 Handling triadic friend of a friend that is a friend with implicit subset of labels . . 386
A.40.19 Handling triadic friend of a friend that is a friend with implicit superset of labels . 387
A.41 TypeConversionFunctions 387
A.41.1 'toBoolean()' on valid literal string 387
A.41.2 'toBoolean()' on booleans 388
A.41.3 'toBoolean()' on variables with valid string values 388
A.41.4 'toBoolean()' on invalid strings 388
A.41.5 'toInteger()' 388
A.41.6 'toInteger()' on float 390
A.41.7 'toInteger()' returning null on non-numerical string 390
A.41.8 'toInteger()' handling mixed number types 390
A.41.9 'toInteger()' handling Any type 391
A.41.10 'toInteger()' on a list of strings 392
A.41.11 'toInteger()' on a complex-typed expression 393
A.41.12 'toFloat()' 394
A.41.13 'toFloat()' on mixed number types 395
A.41.14 'toFloat()' returning null on non-numerical string 396
A.41.15 'toFloat()' handling Any type 397
A.41.16 'toFloat()' on a list of strings 398

A.41.17'toString()' . . . 399
A.41.18'toString()' handling boolean properties . . . 401
A.41.19'toString()' handling inlined boolean . . . 402
A.41.20'toString()' handling boolean literal . . . 403
A.41.21'toString()' should work on Any type . . . 403
A.41.22'toString()' on a list of integers . . . 404
A.41.23'toString()' should accept potentially correct types 1 . . . 405
A.41.24'toString()' should accept potentially correct types 2 . . . 406
A.42 UnionAcceptance . . . 406
A.42.1 Should be able to create text output from union queries . . . 406
A.42.2 Two elements, both unique, not distinct . . . 407
A.42.3 Two elements, both unique, distinct . . . 408
A.42.4 Three elements, two unique, distinct . . . 409
A.42.5 Three elements, two unique, not distinct . . . 410
A.43 UnwindAcceptance . . . 411
A.43.1 Unwinding a list . . . 411
A.43.2 Unwinding a range . . . 412
A.43.3 Unwinding a concatenation of lists . . . 412
A.43.4 Unwinding a collected unwound expression . . . 412
A.43.5 Unwinding a collected expression . . . 412
A.43.6 Double unwinding a list of lists . . . 413
A.43.7 Unwinding the empty list . . . 413
A.43.8 Unwinding null . . . 413
A.43.9 Unwinding list with duplicates . . . 413
A.43.10Unwind does not prune context . . . 413
A.43.11Unwind does not remove variables from scope . . . 414
A.43.12Multiple unwinds after each other . . . 414
A.44 VarLengthAcceptance . . . 414
A.44.1 Handling unbounded variable length match . . . 414
A.44.2 Handling explicitly unbounded variable length match . . . 416
A.44.3 Handling single bounded variable length match 1 . . . 417
A.44.4 Handling single bounded variable length match 2 . . . 417
A.44.5 Handling single bounded variable length match 3 . . . 418
A.44.6 Handling upper and lower bounded variable length match 1 . . . 418
A.44.7 Handling upper and lower bounded variable length match 2 . . . 419
A.44.8 Handling symmetrically bounded variable length match, bounds are zero . . . 421
A.44.9 Handling symmetrically bounded variable length match, bounds are one . . . 423
A.44.10Handling symmetrically bounded variable length match, bounds are two . . . 425
A.44.11Handling upper and lower bounded variable length match, empty interval 1 . . . 426
A.44.12Handling upper and lower bounded variable length match, empty interval 2 . . . 428
A.44.13Handling upper bounded variable length match, empty interval . . . 430
A.44.14Handling upper bounded variable length match 1 . . . 432
A.44.15Handling upper bounded variable length match 2 . . . 433
A.44.16Handling lower bounded variable length match 1 . . . 435
A.44.17Handling lower bounded variable length match 2 . . . 437
A.44.18Handling lower bounded variable length match 3 . . . 438
A.44.19Handling a variable length relationship and a standard relationship in chain, zero length 1 . . . 440
A.44.20Handling a variable length relationship and a standard relationship in chain, zero length 2 . . . 440
A.44.21Handling a variable length relationship and a standard relationship in chain, single length 1 . . . 440
A.44.22Handling a variable length relationship and a standard relationship in chain, single length 2 . . . 440
A.44.23Handling a variable length relationship and a standard relationship in chain, longer 1441
A.44.24Handling a variable length relationship and a standard relationship in chain, longer 2441
A.44.25Handling a variable length relationship and a standard relationship in chain, longer 3441
A.44.26Handling mixed relationship patterns and directions 1 . . . 441

A.44.27Handling mixed relationship patterns and directions 2 442
A.44.28Handling mixed relationship patterns 1 442
A.44.29Handling mixed relationship patterns 2 442
A.45 VarLengthAcceptance2 442
A.45.1 Handling relationships that are already bound in variable length paths 442
A.46 WhereAcceptance 443
A.46.1 NOT and false 443
A.47 WithAcceptance 443
A.47.1 Passing on pattern nodes 443
A.47.2 ORDER BY and LIMIT can be used 444
A.47.3 No dependencies between the query parts 446
A.47.4 Aliasing 447
A.47.5 Handle dependencies across WITH 447
A.47.6 Handle dependencies across WITH with SKIP 447
A.47.7 WHERE after WITH should filter results 447
A.47.8 WHERE after WITH can filter on top of an aggregation 448
A.47.9 ORDER BY on an aggregating key 448
A.47.10ORDER BY a DISTINCT column 448
A.47.11WHERE on a DISTINCT column 448
A.47.12A simple pattern with one bound endpoint 449
A.47.13Null handling 451
A.47.14Nested maps 452
A.47.15Connected components succeeding WITH 453
A.47.16Single WITH using a predicate and aggregation 454
A.47.17Multiple WITHs using a predicate and aggregation 455

B Fraud Detection 456
B.1 Queries 456
B.1.1 create 456
B.1.2 financial-risk 458
B.1.3 shared-contact-information 458

C LDBC Social Network Benchmark – Interactive Workload 460
C.1 Queries 460
C.1.1 query-3 460
C.1.2 query-4 461
C.1.3 query-5 461
C.1.4 query-6 462
C.1.5 query-7 464
C.1.6 query-8 465
C.1.7 query-10 466
C.1.8 query-11 467
C.1.9 query-12 467

D LDBC Social Network Benchmark – Business Intelligence Workload 469
D.1 Queries 469

E Movie Database 470
E.1 Queries 470
E.1.1 query-1 470
E.1.2 query-2 471
E.1.3 query-3 473
E.1.4 query-4 473
E.1.5 query-5 474
E.1.6 query-6 474
E.1.7 query-7 474
E.1.8 query-8 475
E.1.9 query-9 476
E.1.10 query-10 479

E.1.11 query-11 . . . 480
E.1.12 query-12 . . . 482
E.1.13 query-13 . . . 483
E.1.14 query-14 . . . 484
E.1.15 query-15 . . . 485
E.1.16 query-16 . . . 486
E.1.17 query-17 . . . 487
E.1.18 query-18 . . . 488
E.1.19 query-19 . . . 489
E.1.20 query-20 . . . 490
E.1.21 query-21 . . . 492
E.1.22 query-22 . . . 492
E.1.23 query-23 . . . 494

F Static Analysis for Java 496
F.1 Queries . . . 496

G Static Analysis for JavaScript 497
G.1 Queries . . . 497
G.1.1 BlockStatement . . . 497
G.1.2 Boolean . . . 497
G.1.3 CallExpressionNoParam . . . 498
G.1.4 CallExpressionParam . . . 498
G.1.5 ExpressionStatement . . . 498
G.1.6 FunctionDeclaration . . . 499
G.1.7 IfStatementAlternate . . . 499
G.1.8 IfStatementNoAlternate . . . 499
G.1.9 Infinity . . . 500
G.1.10 ListNoItem . . . 500
G.1.11 ListWithItem . . . 500
G.1.12 LiteralX . . . 501
G.1.13 LogicalOr . . . 501
G.1.14 Null . . . 501
G.1.15 Numeric . . . 502
G.1.16 Read . . . 502
G.1.17 RegExp . . . 502
G.1.18 String . . . 502
G.1.19 TypeSystem . . . 503
G.1.20 VariableDeclarationStatement . . . 503
G.1.21 VariableDeclarator . . . 504
G.1.22 Write . . . 504
G.1.23 generatecalls . . . 504
G.1.24 getlastcommithash . . . 505
G.1.25 removefile . . . 506
G.1.26 setcommithash . . . 506
G.1.27 typing . . . 506
G.1.28 unusedfunctions . . . 509

H Train Benchmark 510
H.1 Queries . . . 510
H.1.1 ActiveRoute . . . 510
H.1.2 ConnectedSegments . . . 510
H.1.3 PosLength-simple-return . . . 512
H.1.4 PosLength . . . 513
H.1.5 RouteSensor . . . 515
H.1.6 RouteSensorPositive . . . 516
H.1.7 SemaphoreNeighbor . . . 518
H.1.8 SwitchMonitored . . . 520
H.1.9 SwitchSet-simple-return . . . 522

H.1.10 SwitchSet 524

Executive Summary

This document is generated on each commit of the ingraph repository[1].

Structure. Chapter 1 introduces the theoretical foundations of the openCypher language. Chapter 2 presents incremental relational operators.

Appendices. The appendix chapters contain sets of Cypher queries, and their representations as relational algebraic expressions and trees, along with their incremental equivalents.

- Appendix A: the acceptance tests defined in the openCypher Technology Compliance Kit[2].
- Appendix B: fraud detection queries based on the Neo4j white paper.
- **??**: LDBC Social Network Benchmark queries.
- Appendix E: Movie Database queries from the Neo4j tutorials.
- Appendix F: Java static analysis queries.
- Appendix G: JavaScript static analysis queries.
- Appendix H: Train Benchmark queries.

[1] https://github.com/FTSRG/ingraph

[2] https://github.com/opencypher/openCypher/tree/master/tck

Chapter 1

Theoretical Foundations

1.1 Introduction

Context. Graphs are a well-known formalism, widely used for describing and analysing systems. Graphs provide an intuitive formalism for modelling real-world scenarios, as the human mind tends to interpret the world in terms of objects (*vertices*) and their respective relationships to one another (*edges*) [17].

The *property graph* data model [19] extends graphs by adding labels and properties for both vertices and edges. This gives a rich set of features for users to model their specific domain in a natural way. Graph databases are able to store property graphs and query their contents with complex graph patterns, which, otherwise would be are cumbersome to define and/or inefficient to evaluate on traditional relational databases and query technologies.

Neo4j [14], a popular NoSQL property graph database, offers the Cypher [13] query language to specify graph patterns. Cypher is a high-level declarative query language which can be optimised by the query engine. The openCypher project [15] is an initiative of Neo Technology, the company behind Neo4j, to deliver an open specification of Cypher.

Problem and objectives. The openCypher project features a formal specification of the grammar of the query language (Section 1.4) and a set of acceptance tests that define the behaviour of various Cypher features. However, there is no mathematical formalisation for any of the language features. In ambiguous cases, the user is advised to consult Neo4j's Cypher documentation or to experiment with Neo4j's Cypher query engine and follow its behaviour. Our goal is to provide a formal specification for the core features of openCypher.

Contributions. In this paper, we use a formal definition of the property graph data model [9] and an extended version of relational algebra, operating on multisets (bags) and featuring additional operators [8]. These allow us to construct a concise formal specification for the core features in the openCypher grammar, which can then serve as a basis for implementing an openCypher-compliant query engine.

1.2 Preliminaries

This section defines the mathematical concepts used in the paper. Our notation closely follows [9] and is similar to [19][1].

1.2.1 Property Graph Data Model

A *property graph* is defined as $G = (V, E, \mathsf{src\_trg}, L_v, L_e, l_v, l_e, P_v, P_e)$, where V is a set of vertices, E is a set of directed edges, $\mathsf{src\_trg} : E \to V \times V$ assigns the source and target vertices to edges. The graph is labelled (or typed):

- L_v is a set of vertex labels, $l_v : V \to 2^{L_v}$ assigns *a set of labels* to each vertex.
- L_e is a set of edge labels, $l_e : E \to L_e$ assigns *a single label* to each edge.

[1] The formalism presented in [19] lacks the notion of *vertex labels*.

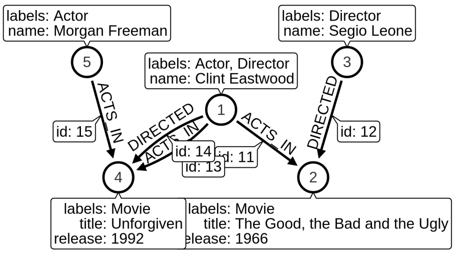


Figure 1.1: Example movie graph.

Furthermore, graph G has properties (*attributed graph*). Let D be a set of atomic domains.

- P_v is a set of vertex properties. A vertex property $p_i \in P_v$ is a function $p_i : V \to D_i \cup \{\mathsf{NULL}\}$, which assigns a property value from a domain $D_i \in D$ to a vertex $v \in V$, if v has property p_i, otherwise $p_i(v)$ returns NULL.
- P_e is a set of edge properties. An edge property $p_j \in P_e$ is a function $p_j : E \to D_j \cup \{\mathsf{NULL}\}$, which assigns a property value from a domain $D_j \in D$ to an edge $e \in E$, if e has property p_j, otherwise $p_j(e)$ returns NULL.

Running example. Figure 1.1 presents an example inspired by the Movie Database dataset[2]. The graph can be represented formally as:
$V = \{1, 2, 3, 4, 5\}; E = \{11, 12, 13, 14, 15\};$
$\mathsf{src\_trg}(11) = \langle 1, 2\rangle; \mathsf{src\_trg}(12) = \langle 3, 2\rangle; \ldots$
$L_v = \{\mathsf{Actor}, \mathsf{Director}, \mathsf{Movie}\};$
$L_e = \{\mathsf{ACTS\_IN}, \mathsf{DIRECTED}\};$
$l_v(1) = \{\mathsf{Actor}, \mathsf{Director}\}; l_v(2) = \{\mathsf{Movie}\}; \ldots;$
$l_e(11) = \mathsf{ACTS\_IN}; l_e(12) = \mathsf{DIRECTED}; \ldots;$
$P_v = \{\mathsf{name}, \mathsf{title}, \mathsf{release}\}; P_e = \{\};$
$\mathrm{name}(1) = \mathsf{'Clint\ Eastwood'}; \mathrm{name}(2) = \mathsf{NULL}; \ldots$
$\mathrm{title}(1) = \mathsf{NULL}; \mathrm{title}(2) = \mathsf{'The\ Good,\ the\ Bad\ and\ the\ Ugly'}; \ldots$
$\mathrm{release}(1) = \mathsf{NULL}; \mathrm{release}(2) = \mathsf{1966}; \ldots$

In the context of this paper, we define a *relation* as a bag (*multiset*) of tuples: a tuple can occur more than once in the relation [8]. Given a property graph G, relation r is a *graph relation* if the following holds:

$$\forall A \in \mathrm{attr}(r) : \mathrm{dom}(A) \subseteq V \cup E \cup D,$$

where $\mathrm{attr}(r)$ is the set of attributes of r, $\mathrm{dom}(A)$ is the domain of attribute A. The schema of r, $\mathrm{sch}(\mathsf{r})$ is a list containing the attribute names. For schema transformations, the *append* operator is denoted by $\|$, the *remove* operator is denoted by $-$.

1.3 Operators of Relational Algebra

For well-known relational algebra operators (e.g. selection, projection, join) and common extensions (e.g. aggregation, left outer join), we only give a brief summary. A more detailed discussion is available in database textbooks, e.g. [8, 4].

We also adapted graph-specific operators from [9][3] and propose new operators.

1.3.1 Nullary Operators

The *get-vertices* nullary operator $\bigcirc_{(\texttt{v: t}_1\wedge\ldots\wedge\texttt{t}_n)}$ returns a graph relation of a single attribute v that contains the ID of all vertices that have *all* of labels $t_1, \ldots, t_n$.

[2] `https://neo4j.com/developer/movie-database/`

[3] The GetNodes operator introduced in [9] and did not support labels. We extended it by allowing the specification of vertex labels and renamed it to *get-vertices* to be consistent with the rest of the definitions. We also extended the ExpandIn and ExpandOut operators to allow it to return a set of edges, and introduced the *expand-both* operator to allow navigation to both directions.

The *get-edges* operator $\Uparrow^{(\mathtt{trg}:\,\mathsf{trgTypes})}_{(\mathtt{src}:\,\mathsf{srcTypes})}$ $[\mathsf{e}\colon \mathsf{labels}]$ operator returns a relation of three attributes. Each row represents an edge and its vertices: the source vertex `src` (with all types of $\mathsf{srcTypes}$), the edge `e` (with a single label of `labels`) and the target vertex `trg` (with all types of $\mathsf{trgTypes}$).

1.3.2 Unary Operators

The *projection* operator π keeps a specific set of attributes in the relation: $t = \pi_{\mathsf{A_1,\ldots,A_n}}\left(r\right)$. Note that the tuples are not deduplicated by default, i.e. the results will have the same number of tuples as the input relation r. The projection operator can also rename the attributes, e.g. $\pi_{\mathsf{v1}\to\mathsf{v2}}\left(r\right)$ renames $\mathsf{v1}$ to $\mathsf{v2}$.

The *selection* operator σ filters the incoming relation according to some criteria. Formally, $t = \sigma_{\theta}\left(r\right)$, where predicate θ is a propositional formula. The operator selects all tuples in r for which θ holds.

The *duplicate-elimination* operator δ eliminates duplicate tuples in a bag.

The *grouping* operator γ groups tuples according to their value in one or more attributes and aggregates the remaining attributes.

The *sorting* operator τ transforms a bag relation of tuples to a list of tuples by ordering them. The ordering is defined by specified attributes of the tuples with an ordering direction (ascending $\uparrow$/descending $\downarrow$) for each attribute, e.g. $\tau_{\uparrow\mathsf{v1},\downarrow\mathsf{v2}}(r)$.

The *top* operator λ_l^s (adapted from [12]) takes a list as its input, skips the top s tuples and returns the next l tuples.[4]

The *expand-both* operator $\updownarrow^{(\mathtt{l_1}\vee\ldots\vee\mathtt{l_k}*\mathtt{min}\ldots\mathtt{max}:\,\mathtt{v})}_{(\mathtt{E})}$ $[\mathtt{w}\colon \mathsf{t_1} \wedge \ldots \wedge \mathsf{t_n} * \mathsf{min} \ldots \mathsf{max}]\,(r)$ adds (1) a new attribute w to r containing the IDs of vertices having *all* labels $t_1, \ldots, t_n$ that can be reached from vertices of attribute v by traversing edges having *any* labels $l_1, \ldots, l_n$, and (2) a new attribute E for the edges of the path from v to w. The operator may use at least min and at most max hops, both defaulting to 1 if omitted. The *expand-in* operator $\downarrow$ and *expand-out* operator $\uparrow$ only consider directed paths from w to v and from v to w, respectively.

The *unwind* operator $\omega_{\mathsf{xs}}^{\mathsf{x}}$ unfolds a list xs to a variable x by introducing additional rows, each containing a single element of the list.

The *all-different* operator $\not\equiv_{\mathsf{E_1,E_2,E_3,\ldots}}\,(r)$ filters r to keep tuples where the variables in $\bigcup_i E_i$ are pairwise different.[5] It can be expressed as a *selection*:

$$\not\equiv_{\mathsf{E_1,E_2,E_3,\ldots}}\,(r) = \sigma_{\bigwedge\limits_{\mathsf{e_1,e_2} \in \bigcup\limits_{\mathsf{i}} \mathsf{E_i} \,\wedge\, \mathsf{e_1} \neq \mathsf{e_2}} \mathsf{r.e_1} \neq \mathsf{r.e_2}}(r)$$

We use the *all-different* operator to guarantee the uniqueness of edges (see the remark on *uniqueness of edges* in Section 1.4).

1.3.3 Binary Operators

The $\cup$ operator produces the set union of two relations, while the $\uplus$ operator produces the *bag union* of two operators, e.g. $\{\langle 1,2\rangle, \langle 1,2\rangle, \langle 3,4\rangle\} \uplus \{\langle 1,2\rangle\} = \{\langle 1,2\rangle, \langle 1,2\rangle, \langle 1,2\rangle, \langle 3,4\rangle\}$. For both the *union* and *bag union* operators, the schema of the operands must have the same number of attributes. Some authors also require that they share a common schema, i.e. have the same set of attributes [8].

The $\times$ operator produces the *Cartesian product*:

$$t = r \times s.$$

The result of the *natural join* operator $\bowtie$ is determined by creating the Cartesian product of the relations, then filtering those tuples which are equal on the attributes that share a common name. The combined tuples are projected: from the attributes present in both of the two input relations, we only keep the ones in r and drop the ones in s. Thus, the join operator is defined as

$$r \bowtie s = \pi_{R\cup S}\left(\sigma_{\mathsf{r.A_1=s.A_1} \,\wedge\, \ldots \,\wedge\, \mathsf{r.A_n=s.A_n}}\left(r \times s\right)\right),$$

where $\{A_1, \ldots, A_n\}$ is the set of attributes that occur both in R and S, i.e. $R \cap S = \{A_1, \ldots, A_n\}$. Note that if the set of common attributes is empty, the *natural join* operator is equivalent to the Cartesian product of the relations. The join operator is both commutative and associative: $r \bowtie s = s \bowtie r$ and $(r \bowtie s) \bowtie t = r \bowtie (s \bowtie t)$, respectively.

[4] SQL implementations offer the `OFFSET` and the `LIMIT`/`TOP` keywords.

[5] Should e.g. E_2 be a set of the single variable e_2, the variable name can be used as a shorthand instead, so $\not\equiv_{\mathsf{E_1,e_2,E_3,\ldots}}\,(r) \equiv \not\equiv_{\mathsf{E_1,\{e_2\},E_3,\ldots}}\,(r)$

The *antijoin* operator $\triangleright$ (also known as *left anti semijoin*) collects the tuples from the left relation r which have no matching pair in the right relation s:

$$t = r \triangleright s = r \setminus \pi_R \left(r \bowtie s \right),$$

where π_R denotes a projection operator, which only keeps the attributes of the schema over relation r. The antijoin operator is not commutative and not associative.

The *left outer join* $⟕$ pads tuples from the left relation that did not match any from the right relation with NULL values and adds them to the result of the *natural join* [21]:

$$t = r ⟕ s = (r \bowtie s) \cup (r \triangleright s) \times \{\mathsf{NULL}, \ldots, \mathsf{NULL}\},$$

where the constant relation $\{\mathsf{NULL}, \ldots, \mathsf{NULL}\}$ is on the schema $S \setminus R$.

1.3.4 Property Access

Assuming that x is an attribute of a graph relation, we use the notation $x.a$ in (1) attribute lists for projections and (2) selection conditions to express the access to the corresponding value of property a in the property graph [9].

1.3.5 Summary of Operators

Table 1.1 provides an overview of the operators of relational graph algebra.

ops	**operator**	**name**	**prop.**	**output for**			**schema**
				set	**bag**	**list**	
0	$\bigcirc_{(\mathtt{v})}$	*get-vertices*	set	set	set	set	$\langle \mathsf{v} \rangle$
1	$\pi_{\mathsf{v}_1, \mathsf{v}_2, \ldots}(r)$	*projection*	i	bag	bag	list	$\langle \mathsf{v}_1, \mathsf{v}_2, \ldots \rangle$
	$\sigma_{\mathsf{condition}}(r)$	*selection*	i	set	bag	list	$\mathrm{sch}(\mathsf{r})$
	$\updownarrow\,^{(\mathtt{w})}_{(\mathtt{v})} [\mathsf{e} * \mathsf{min} \ldots \mathsf{max}]\,(r)$	*expand-both*	–	set	bag	list	$\mathrm{sch}(\mathsf{r}) \parallel \langle \mathsf{e}, \mathsf{w} \rangle$
	$\not\equiv_{\mathsf{variables}}\,(r)$	*all-different*	i	set	bag	list	$\mathrm{sch}(\mathsf{r})$
	$\delta(r)$	*duplicate-elimination*	i	set	set	list	$\mathrm{sch}(\mathsf{r})$
	$\tau_{\downarrow \mathsf{v}_1, \uparrow \mathsf{v}_2, \ldots}(r)$	*sorting*	i	list	list	list	$\mathrm{sch}(\mathsf{r})$
	$\gamma_{\mathsf{v}_1, \mathsf{v}_2, \ldots}(r)$	*grouping*	i	set	set	set	$\langle \mathsf{v}_1, \mathsf{v}_2, \ldots \rangle$
	$\lambda(r)$	*top*	–	list	list	list	$\mathrm{sch}(\mathsf{r})$
2	$r \cup s,\ r \setminus s$	*union, minus*	–	set	set	set	$\mathrm{sch}(\mathsf{r})$
	$r \uplus s$	*bag union*	c, a	bag	bag	bag	$\mathrm{sch}(\mathsf{r})$
	$r \times s$	*Cartesian product*	c, a	set	bag	bag	$\mathrm{sch}(\mathsf{r}) \parallel \mathrm{sch}(\mathsf{s})$
	$r \bowtie s$	*natural join*	c, a	set	bag	bag	$\mathrm{sch}(\mathsf{r}) \parallel (\mathrm{sch}(\mathsf{s}) \setminus \mathrm{sch}(\mathsf{r}))$
	$r ⟕ s$	*left outer join*	–	set	bag	bag	$\mathrm{sch}(\mathsf{r}) \parallel (\mathrm{sch}(\mathsf{s}) \setminus \mathrm{sch}(\mathsf{r}))$
	$r \triangleright s$	*antijoin*	c, a	set	bag	bag	$\mathrm{sch}(\mathsf{r})$

Table 1.1: Properties of relational graph algebra operators. A unary operator α is idempotent (i), iff $\alpha(x) = \alpha(\alpha(x))$ for all inputs. A binary operator β is commutative (c), iff $x\ \beta\ y = y\ \beta\ x$ and associative (a), iff $(x\ \beta\ y)\ \beta\ z = x\ \beta\ (y\ \beta\ z)$.

1.4 The openCypher Query Language

Language. As the primary query language of Neo4j [14], Cypher [13] was designed to read easily. It allows users to specify the graph pattern by a syntax resembling an actual graph. The goal of the openCypher project [15] is to provide a standardised specification of the Cypher language. Listing 1.1 shows an openCypher query, which returns all people who (1) are both actors and directors and (2) have acted in a movie together with Clint Eastwood.

```
MATCH (a1)-[:ACTS_IN]->(:Movie)<-[:ACTS_IN]-(a2:Actor:Director)
WHERE a1.name = 'Clint Eastwood'
RETURN a2
```

Listing 1.1: Get people who are both actors and directors and acted in a movie with Clint Eastwood.

The query returns with a bag of vertices that have both the labels Actor and Director and share a common Movie neighbor through ACTS_IN edges. Cypher guarantees that these edges are only traversed once, so the vertex of Clint Eastwood is not returned (see the section on the uniqueness of edges).

Implementation. While Neo4j uses a parsing expression grammar (PEG) [6] for specifying the grammar rules of Cypher, openCypher aims to achieve an implementation-agnostic specification by only providing a context-free grammar. The parser can be implemented using any capable parser technology, e.g. ANTLR4 [16] or Xtext [5].

Legacy grammar rules. It is not a goal of the openCypher project to fully cover the features of Neo4j's Cypher language: "Not all grammar rules of the Cypher language will be standardised in their current form, meaning that they will not be part of openCypher as-is. Therefore, the openCypher grammar will not include some well-known Cypher constructs; these are called 'legacy'."[6] The *legacy rules* include commands (`CREATE INDEX`, `CREATE UNIQUE CONSTRAINT`, etc.), pre-parser rules (`EXPLAIN`, `PROFILE`) and deprecated constructs (`START`). A detailed description is provided in the openCypher specification. In our work, we focused on the *standard core* of the language and ignored legacy rules.

Uniqueness for edges. In an openCypher query, a `MATCH` clause defines a graph pattern. A query can be composed of multiple patterns spanning multiple `MATCH` clauses. For the matches of a pattern within a single `MATCH` clause, edges are required to be unique. However, matches for multiple `MATCH` clauses can share edges. This uniqueness criterium can be expressed in a compact way with the *all-different* operator introduced in Section 1.3.2. For vertices, this restriction does not apply.

Aggregation. It indeed makes sense to calculate aggregation over graph pattern matches, though, its result will not necessarily be pattern match with vertices and edges. Based on some *grouping criteria*, matches are put into categories, and values for the grouping criteria as well as grouping functions over the groups, the aggregations are evaluated in a single tuple for each and every category. For example, `count`, `avg`, `sum`, `max`, `min`, `stdDev`, `stdDevP`, `collect`. The `collect` function is an exception as it does not return a single scalar value but returns a collection (list).

In the SQL query language, grouping criteria is explicitly given by using the `GROUP BY` clause. In openCypher, however, this is done implicitly in the `RETURN` as well as in `WITH` clauses: vertices, edges and their properties that appear outside the grouping functions become the *grouping criteria*.[7]

Subqueries. One can compose an openCypher query of multiple subqueries. Subqueries, written subsequently, mostly begin by a `MATCH` clause and end at (including) a `RETURN` or `WITH` clause, the latter having an optional `WHERE` clause to follow. The `WITH` and `RETURN` clauses determine the resulting schema of the subquery by specifiying the vertices, edges, attributes and aggregates of the result. When `WITH` has the optional `WHERE` clause, it applies an other filter on the subquery result. [8] The last subquery must be ended by `RETURN`, whereas all the previous ones must be ended by `WITH`. If a query is composed by more than one subqueries, their results are joined together using *natural join* or *left outer join* operators.

1.5 Mapping openCypher Queries to Relational Graph Algebra

In this section, we first give the mapping algorithm of openCypher queries to relational graph algebra, then we give a more detailed listing of the compilation rules for the query language constructs in Table 1.2. We follow the bottom-up approach to build the relational graph algebra tree based on the openCypher query. The algorithm is as follows. Join operations always use all common variables to match the two inputs (see *natural join* in Section 1.3.3).

1. A single pattern is turned left-to-right to a *get-vertices* for the first vertex and a chain of *expand-in*, *expand-out* or *expand-both* operators for inbound, outbound or undirected relationships, respectively.

2. Patterns in the same `MATCH` clause are joined by *natural join*.

[6] https://github.com/opencypher/openCypher/tree/master/grammar

[7] This approach is also used by some SQL code assistant IDEs generating the `GROUP BY` clause for a query.

[8] This is much like the `HAVING` construct of the SQL language with the major difference that it is also allowed in openCypher in case no aggregation has been done.

3. Append an *all-different* operator for all edge variables that appear in the MATCH clause because of the non-repeating edges language rule.
4. Process the WHERE clause. Note that according to the grammar, WHERE is bound to a MATCH clause.
5. Several MATCH clauses are connected to a left deep tree of *natural join*. If MATCH has the OPTIONAL modifier, *left outer join* is used instead of *natural join*.
6. If there is a positive or negative pattern deferred from WHERE processing, append it as a *natural join* or *antijoin* operator, respectively.
7. Append *grouping*, if RETURN or WITH clause has grouping functions inside
8. Append *projection* operator based on the RETURN or WITH clause. This operator will also handle the renaming (i.e. AS).
9. Append *duplicate-elimination* operator, if the RETURN or WITH clause has the DISTINCT modifier.
10. Append a *selection* operator in case the WITH had the optional WHERE clause.
11. If this is not the first subquery, join to the relational graph algebra tree using *natural join* or *left outer join*.
12. Assemble a *union* operation from the query parts[9]. As the *union* operator is technically a binary operator, the *union* of more than two query parts are represented as a left deep tree of UNION operators.

Example. The example query in Listing 1.1 can be formalized as:

$$\pi_{\mathsf{a2}}\left(\sigma_{\mathsf{a1.name}='\mathsf{C.\,E.}'}\left(\not\equiv_{\mathsf{\_e1,\_e2}}\downarrow^{(\mathsf{a2:\ Actor}\wedge\mathsf{Director})}_{(\mathsf{a1})}[\mathsf{\_e1:\ ACTS\_IN}]\uparrow^{(\mathsf{:\ Movie})}_{(\mathsf{a1})}[\mathsf{\_e2:\ ACTS\_IN}]\left(\bigcirc_{(\mathsf{a1:\ Actor})}\right)\right)\right)$$

Note that the $\not\equiv$ guarantees the uniqueness constraint for the edges (Section 1.4), which prevents the query from returning the vertex Clint Eastwood.

Optimisations. Queries with negative conditions for patterns can also be expressed using the *antijoin* operator. For example, MATCH «p1» WHERE NOT «p2» can be formalized as

$$\not\equiv_{\mathsf{edges\ of\ p1}}(\mathsf{p}_1) \rhd \not\equiv_{\mathsf{edges\ of\ p2}}(\mathsf{p}_2)$$

Limitations. Our mapping does not completely cover the openCypher language. As discussed in Section 1.4, some constructs are defined as legacy and thus were omitted. Also, we did not formalize expressions (e.g. conditions in selections), collections (arrays and maps), which are required for both path variables[10] and the UNWIND operator. The mapping does not cover parameters and data manipulation operations, e.g. CREATE, DELETE, SET and MERGE.

1.6 Related Work

The TinkerPop framework [1] aims to provide a standard data model for property graphs, along with Gremlin, a high-level graph-traversal language [18] and the Gremlin Structure API, a low-level programming interface.

Besides property graphs, graph queries can be formalized on different graph-like data models and even relational databases.

EMF. The Eclipse Modeling Framework (EMF) is an object-oriented modelling framework widely used in model-driven engineering. Henshin [2] provides a visual language for defining patterns, while Epsilon [10] and VIATRA Query [3] provide high-level declarative (textual) query languages, Epsilon Pattern Language and VIATRA Query Language.

[9] In this context, query parts refer to those parts of the query connected by the UNION openCypher keyword.

[10] `MATCH p=(:Person)-[:FRIEND*1..2]->(:Person)`

Language construct	**Relational algebra expression**
Vertex, edge and path patterns	
`()`	$\bigcirc_{(\_v)}$
`(:types)`	$\bigcirc_{(\_v:\ \mathsf{types})}$
`(«v»:types)`	$\bigcirc_{(v:\ \mathsf{types})}$
`«v» -[«E»:«labels»]- («w»:types...)`	$\updownarrow^{(w:\ \mathsf{types})}_{(v)} [E:\ \mathsf{labels}]\ (\mathsf{p})$
`«v»<-[«E»:«labels»]->(«w»:types...)`	
`«v» -[«E»:«labels»]->(«w»:types...)`	$\uparrow^{(w:\ \mathsf{types})}_{(v)} [E:\ \mathsf{labels}]\ (\mathsf{p})$
`«v»<-[«E»:«labels»]- («w»:types...)`	$\downarrow^{(w:\ \mathsf{types})}_{(v)} [E:\ \mathsf{labels}]\ (\mathsf{p})$
`«v» -[«E»:«labels»*«min»..«max»]->(«w»:«t2»)`	$\uparrow^{(w:\ \mathsf{types})}_{(v)} [E:\ \mathsf{labels} * \mathsf{min} \ldots \mathsf{max}]\ (\mathsf{p})$
`«v» -[«E»:«labels»*«min»..]->(«w»:«t2»)`	$\uparrow^{(w:\ \mathsf{types})}_{(v)} [E:\ \mathsf{labels} * \mathsf{min} \ldots \infty]\ (\mathsf{p})$
`«v» -[«E»:«labels»*..«max»]->(«w»:«t2»)`	$\uparrow^{(w:\ \mathsf{types})}_{(v)} [E:\ \mathsf{labels} * 1 \ldots \mathsf{max}]\ (\mathsf{p})$
`«v» -[«E»:«labels»*]->(«w»:«t2»)`	$\uparrow^{(w:\ \mathsf{types})}_{(v)} [E:\ \mathsf{labels} * 1 \ldots \infty]\ (\mathsf{p})$
Combining and filtering pattern matches	
`MATCH «p»`	$\not\equiv_{\mathsf{edges\ of\ p}} (\mathsf{p})$
`MATCH «p1», «p2»`	$\not\equiv_{\mathsf{edges\ of\ p1\ and\ p2}} (\mathsf{p}_1 \bowtie \mathsf{p}_2)$

`MATCH «p1»`
`MATCH «p2»`
	$\not\equiv_{\mathsf{edges\ of\ p1}} (\mathsf{p}_1) \bowtie \not\equiv_{\mathsf{edges\ of\ p2}} (\mathsf{p}_2)$

`MATCH «p1»`
`OPTIONAL MATCH «p2»`
	$\not\equiv_{\mathsf{edges\ of\ p1}} (\mathsf{p}_1) ⟕ \not\equiv_{\mathsf{edges\ of\ p2}} (\mathsf{p}_2)$

`MATCH «p»`
`WHERE «condition»`
	$\sigma_{\mathsf{condition}}(r)$, where condition may specify patterns and arithmetic constraints on existing variables

Result and sub-result operations. Rules for `RETURN` also apply to `WITH`.	
`RETURN «variables»`	$\pi_{\mathsf{variables}}(r)$
`RETURN «v1» AS «alias1» ...`	$\pi_{\mathsf{v1} \to \mathsf{alias1}, \ldots}(r)$
`RETURN DISTINCT «variables»`	$\delta\left(\pi_{\mathsf{variables}}(r)\right)$
`RETURN «variables», «aggregates»`	$\gamma_{\mathsf{variables},\mathsf{aggregates}}(r)$
List operations	
`ORDER BY «v1» [ASC|DESC] ...`	$\tau_{\uparrow/\downarrow \mathsf{v1}, \ldots}(r)$
`LIMIT «l»`	$\lambda_l(r)$
`SKIP «s»`	$\lambda^s(r)$
`SKIP «s» LIMIT «l»`	$\lambda^s_l(r)$
Combining results	
`«query1» UNION «query2»`	$r_1 \cup r_2$
`«query1» UNION ALL «query2»`	$r_1 \uplus r_2$

Table 1.2: Mapping from openCypher constructs to relational algebra.

RDF. The Resource Description Framework (RDF) [24] aims to describe entities of the semantic web. RDF assumes sparse, ever-growing and incomplete data stored as triples that can be queried using the SPARQL [25] graph pattern language.

SQL. In general, relational databases offer limited support for graph queries: recursive queries are supported by PostgreSQL using the `WITH RECURSIVE` keyword and by the Oracle Database using the `CONNECT BY` keyword. Graph queries are supported in SAP HANA Graph Scale-Out Extension prototype [20], through a SQL-based language [11].

1.7 Conclusion and Future Work

In this paper, we presented a formal specification for a subset of the openCypher query language. This provides the theoretical foundations to use openCypher as a language for graph query engines. Using the proposed mapping, an openCypher-compliant query engine could be built on any relational database engine to (1) store property graphs as graph relations and to (2) efficiently evaluate the extended operators of relational graph algebra.

As a future work, we will give formal specification of the operators for incremental query evaluation, which requires us to define *maintenance operations* to keep their result in sync with the latest set of changes. Our long-term research objective is to design and prototype a *distributed, incremental graph query engine* [22] for the property graph data model.

Chapter 2

Incremental Query Evaluation

This chapter is based on our paper [23].

In many use cases, queries are continuously evaluated, while the data only changes rarely and to a small degree. The validation queries in MDE are a typical example of such a workload. The goal of *incremental query evaluation* is to speed up such queries, using the (partial) results obtained during the previous executions of the query and only computing the effect of the latest set of changes.

Incremental query evaluation algorithms typically use additional data structures for caching interim results. This implies that they usually consume more memory than non-incremental, search-based algorithms. In other words, they trade memory consumption for execution speed. This approach, called *space–time tradeoff*, is well-known and widely used in computer science.

Numerous algorithms were proposed for incremental pattern matching. Mostly, these algorithms originate from the field of rule-based expert systems. In this paper, we use the *Rete algorithm* [7], which creates a *data flow network* for evaluating relational queries.

2.1 Overview of Rete Networks

Draw a Rete network in tikz

The Rete algorithm constructs a network of three types of nodes. Following Figure 2.2, from bottom to top:

1. *Input nodes* are responsible for indexing the graph, i.e. they store the appropriate tuples for vertices and edges in the graph. They are also responsible for sending *change sets* as *update messages* to worker nodes that are *subscribed* to them.

2. *Worker nodes* perform a relational algebraic operation on their input and propagate the results to other worker nodes or production nodes. Some worker nodes are *stateful*: they store partial query results in their memory to allow incremental reevaluation. Worker nodes have two types: *unary nodes* have a single input slot, *binary nodes* have two input slots.

Figure 2.1: Example railway instance model as a labeled graph

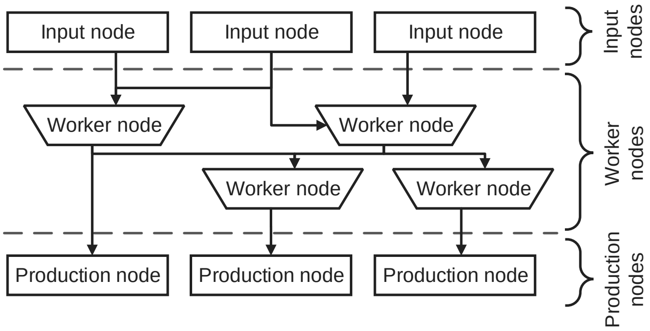


Figure 2.2: The structure of the Rete propagation network.

3. *Production nodes* are terminators that provide an interface for fetching the results.

The Rete network operates as follows. First, the network computes the set of pattern matches in the graph. Then upon a change in the graph, the network is incrementally maintained by propagating *update messages* (also known as *deltas*, denoted with the Δ character). Adding new graph matches to the result set is expressed as *positive update messages*, while removing matches results in *negative update messages*.

In the following, we discuss Rete nodes in detail. For unary and binary nodes, we formulate the *maintenance operations*, which are performed upon receiving an update message. For these operations, we denote the output relation by t, the updated output relation by t', and the propagated update message on the output by Δt. If the *propagated update message* is a positive update, $t' = t \cup \Delta t$, if it is a negative update, $t' = t \setminus \Delta t$.

2.2 Input Nodes

Input nodes provide the relations for each label of the graph. For example, the input node for the requires edge label of example the graph (Figure 2.1) returns tuples that are currently in the *requires* relation: $\{\langle 2, d, 6\rangle, \langle 4, f, 7\rangle\}$. This input node is also responsible for propagating changes to worker nodes in the network:

- If a requires edge 't' is inserted from vertex 2 to 5, the input node sends a positive update message to its subscriber nodes with the change set $\{\langle 2, t, 4\rangle\}$.
- If the edge 'd' between vertices 2 and 6 is deleted, the input node sends a negative update to its subscriber nodes with the change set $\{\langle 2, d, 6\rangle\}$.

The relations contained by input nodes can be defined with nullary operators (Section 1.3.1): input nodes indexing vertices implement the *get-vertices* operator, while input nodes indexing edges implement the *get-edges* operator.

2.3 Unary Nodes

Unary nodes have one input slot. They filter or transform the tuples of the parent node according to certain criteria. In the following, the relation representing the input tuples is denoted with r, the relation representing the output tuples is denoted with t, and the operator processing the input is denoted with α:

$$t = \alpha(r).$$

Maintenance

In the following, we assume that the α operator is *distributive* w.r.t. the union ($\cup$) and set minus ($\setminus$) operators. If a unary node receives an update Δr, it performs the operation and computes the change set. For *positive updates*, the result (t') and the changeset (Δt) are:

$$\begin{aligned} t' &\equiv \alpha\left(r \cup \Delta r\right) \\ &= \alpha\left(r\right) \cup \alpha\left(\Delta r\right) \\ &= t \cup \underbrace{\alpha\left(\Delta r\right)}_{\Delta t} \end{aligned}$$

Similarly, for *negative updates*:

$$\begin{aligned} t' &\equiv \alpha\left(r \setminus \Delta r\right) \\ &= \alpha\left(r\right) \setminus \alpha\left(\Delta r\right) \\ &= t \setminus \underbrace{\alpha\left(\Delta r\right)}_{\Delta t} \end{aligned}$$

Unary nodes are often implemented as *stateless* nodes, i.e. they do not store the results of the previous executions. Instead, these results are cached in their subscribers, e.g. indexers of *binary nodes* (Section 2.4) or *production nodes* (Section 2.5).

As their name suggests, unary nodes implement unary relational algebraic operators (Section 1.3.2):

- The *projection node* performs a projection operation on the input relation.
- The *selection node* performs a selection operation on the input relation.
- The *all-different node* ...
- The *grouping node* ...
- The *unwind node* ...
- The *top node* ...

As all operators are distributive w.r.t. the union and set minus operators , their results can be maintained by performing the operation for the change set Δr.

are they? prove

2.4 Binary Nodes

Binary nodes implement *binary relational operators* (Section 1.3.3) and have two input slots: the *primary* (p) and the *secondary* (s). Binary node implementations typically cache both their input relations in *indexers*.

2.4.1 Natural Join Node

Maintenance

In the following, we define the maintenance operations for natural join nodes. If a natural join node receives a *positive update* Δp on its *primary* input slot, the result (t') and the change set (Δt) are determined as follows:

$$\begin{aligned} t' &\equiv \left(p \cup \Delta p\right) \bowtie s \\ &= \left(p \bowtie s\right) \cup \left(\Delta p \bowtie s\right) \\ &= t \cup \underbrace{\left(\Delta p \bowtie s\right)}_{\Delta t} \end{aligned}$$

If the node receives a *positive update* Δs on its *secondary* input slot, the result (t') and the change set (Δt) are the following:

$$\begin{aligned} t' &\equiv p \bowtie \left(s \cup \Delta s\right) \\ &= \left(p \bowtie s\right) \cup \left(p \bowtie \Delta s\right) \\ &= t \cup \underbrace{\left(p \bowtie \Delta s\right)}_{\Delta t} \end{aligned}$$

For *negative updates*, the changeset is the same, but it is propagated as a *negative update*. The result is $t' = t \setminus (\Delta p \bowtie s)$ and $t' = t \setminus (p \bowtie \Delta s)$, for updates messages on the primary and the secondary input slots, respectively.

2.4.2 Antijoin Node

Maintenance

As the antijoin operator is not commutative, handling update messages requires us to distinguish between the following cases:

- Update on the primary slot.
  - Positive update: send a *positive update* for each incoming tuple for which there is no match on the secondary indexer.
$$\begin{aligned} t' &\equiv (p \cup \Delta p) \rhd s \\ &= (p \rhd s) \cup (\Delta p \rhd s) \\ &= t \cup \underbrace{(\Delta p \rhd s)}_{\Delta t} \end{aligned}$$
  - Negative update: send a *negative update* with the following tuples:
$$\Delta t = \Delta p \rhd s$$
- Update on the secondary slot. This case is more difficult to handle, so we recall the definition of the antijoin operator from Section 1.3.3 for relations p and s:
$$t \equiv p \rhd s = p \setminus (p \bowtie \pi_{P \cap S}(s)),$$
  - For positive updates, the result set can be expressed as:
$$\begin{aligned} t' &\equiv p \rhd (s \cup \Delta s) \\ &= p \setminus (p \bowtie \pi_{P \cap S}(s \cup \Delta s)) \end{aligned}$$
    Positive updates on the secondary indexer result in *negative updates* on the result set, so that $t' = t \setminus \Delta t$, hence $\Delta t = t \setminus t'$.
    For sets $A, B \subseteq C$, the following equality holds: $(C \setminus A) \setminus (C \setminus B) = B \setminus A$. Applying this with $C = p$ and using the distributive property of the natural join operator, the change set can be determined as:
$$\begin{aligned} \Delta t = t \setminus t' &= \overbrace{[p \setminus (p \bowtie \pi_{P \cap S}(s))]}^{t} \setminus \overbrace{[p \setminus (p \bowtie \pi_{P \cap S}(s \cup \Delta s))]}^{t'} \\ &= (p \bowtie \pi_{P \cap S}(s \cup \Delta s)) \setminus (p \bowtie \pi_{P \cap S}(s)) \\ &= p \bowtie (\pi_{P \cap S}(s \cup \Delta s) \setminus \pi_{P \cap S}(s)) \\ &= p \bowtie (\pi_{P \cap S}(s) \cup \pi_{P \cap S}(\Delta s) \setminus \pi_{P \cap S}(s)) \\ &= p \bowtie (\pi_{P \cap S}(\Delta s) \setminus \pi_{P \cap S}(s)) \end{aligned}$$
  - For negative updates, the result set can be expressed as:
$$\begin{aligned} t' &\equiv p \rhd (s \setminus \Delta s) \\ &= p \setminus (p \bowtie \pi_{P \cap S}(s \setminus \Delta s)), \end{aligned}$$
    Negative updates may result in *positive updates* on the result set. Since $t' = t \cup \Delta t$, we can define $\Delta t = t' \setminus t$:
$$\begin{aligned} \Delta t = t' \setminus t &= \overbrace{[p \setminus (p \bowtie \pi_{P \cap S}(s \setminus \Delta s))]}^{t'} \setminus \overbrace{[p \setminus (p \bowtie \pi_{P \cap S}(s))]}^{t} \\ &= \underbrace{(p \bowtie \pi_{P \cap S}(s))}_{x} \setminus \underbrace{(p \bowtie \pi_{P \cap S}(s \setminus \Delta s))}_{y} \end{aligned}$$
    Although this change set may seem difficult to calculate, we point out that both x and y can be maintained incrementally. Furthermore, they only grow linearly in the size of p, as the join operator does not introduce new attributes, hence it can only reduce the number of elements in the relation.

2.4.3 Left Outer Join Node

$$t' \equiv p \mathbin{⟕} s$$

TODO: define maintenance operators

2.4.4 Union Node

$$\begin{aligned} t' &\equiv (p \cup \Delta p) \cup s \\ &= (p \cup s) \cup \Delta p \\ &= t \cup \underbrace{\Delta p}_{\Delta t} \end{aligned}$$

As the union operator is commutative, for updates on the secondary input, the rules are the same, but p and s are changed and Δp is renamed to Δs.

2.5 Production Nodes

Production nodes are terminators that provide an interface for fetching results of a query (the match set) and also propagate the changes introduced by the latest update message.[1]

Maintenance

The change set is defined as:

$$\Delta t \equiv \bigcup_{i=1}^{n} \Delta r_i,$$

where $\Delta r_1, \Delta r_2, \ldots, \Delta r_n$ are the update messages triggered by the last change.

[1] In popular Rete implementations, clients are usually subscribed to the production nodes and notified about the changes in the result set.

Bibliography

[1] Apache. Tinkerpop3. `http://tinkerpop.apache.org/`, 2016.

[2] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer. pages 121–135, 2010.

[3] G. Bergmann et al. Incremental evaluation of model queries over EMF models. In *MODELS*, pages 76–90, 2010.

[4] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems, 3rd Edition.* Addison-Wesley-Longman, 2000.

[5] M. Eysholdt and H. Behrens. Xtext: implement your language faster than the quick and dirty way. In *SIGPLAN, SPLASH/OOPSLA*, pages 307–309, 2010.

[6] B. Ford. Parsing expression grammars: a recognition-based syntactic foundation. In *SIGPLAN-SIGACT, POPL*, pages 111–122, 2004.

[7] C. Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artif. Intell.*, 19(1):17–37, 1982.

[8] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems – The complete book.* Pearson Education, 2nd edition, 2009.

[9] J. Hölsch and M. Grossniklaus. An algebra and equivalences to transform graph patterns in neo4j. In *EDBT/ICDT, GraphX workshop*, 2016.

[10] D. S. Kolovos, R. F. Paige, and F. Polack. The Epsilon transformation language. pages 46–60, 2008.

[11] C. Krause et al. An SQL-based query language and engine for graph pattern matching. In *ICGT*, pages 153–169, 2016.

[12] C. Li, K. C. Chang, I. F. Ilyas, and S. Song. RankSQL: Query algebra and optimization for relational top-k queries. In *SIGMOD*, pages 131–142, 2005.

[13] Neo Technology. Cypher query language. `https://neo4j.com/docs/developer-manual/current/cypher/`, 2016.

[14] Neo Technology. Neo4j. `http://neo4j.org/`, 2016.

[15] Neo Technology. openCypher project. `http://www.opencypher.org/`, 2016.

[16] T. Parr. *The Definitive ANTLR 4 Reference.* Pragmatic Bookshelf, 2nd edition, 2013.

[17] M. A. Rodriguez. A collectively generated model of the world. In *Collective intelligence: creating a prosperous world at peace.* Oakton: Earth Intelligence Network, 2008.

[18] M. A. Rodriguez. The Gremlin graph traversal machine and language (invited talk). DBPL 2015, pages 1–10, New York, NY, USA, 2015. ACM.

[19] M. A. Rodriguez and P. Neubauer. The graph traversal pattern. In *Graph Data Management: Techniques and Applications.*, pages 29–46. 2011.

[20] M. Rudolf, M. Paradies, C. Bornhövd, and W. Lehner. The graph story of the SAP HANA database. In *BTW*, pages 403–420, 2013.

[21] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts, 5th Edition*. McGraw-Hill Book Company, 2005.

[22] G. Szárnyas et al. IncQuery-D: A distributed incremental model query framework in the cloud. In *MODELS*, pages 653–669, 2014.

[23] G. Szárnyas, J. Maginecz, and D. Varró. Evaluation of optimization strategies for incremental graph queries. *Periodica Polytechnica, Electrical Engineering and Computer Science*, 01/2017 2017. Accepted.

[24] World Wide Web Consortium. Resource Description Framework (RDF). `http://www.w3.org/standards/techs/rdf/`.

[25] World Wide Web Consortium. SPARQL query language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`.

Appendix A

TCK Acceptance Tests

Total progress:

294 of 496

A.1 AggregationAcceptance

Progress:

16 of 26

A.1.1 Support multiple divisions in aggregate function [regression test]

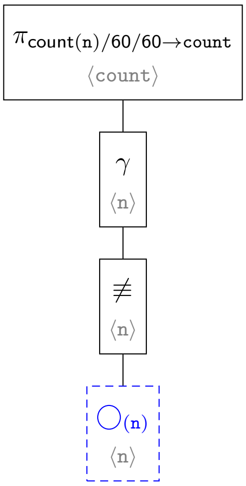
Query specification (Support multiple divisions in aggregate function)

```
MATCH (n)
RETURN count(n) / 60 / 60 AS count
```

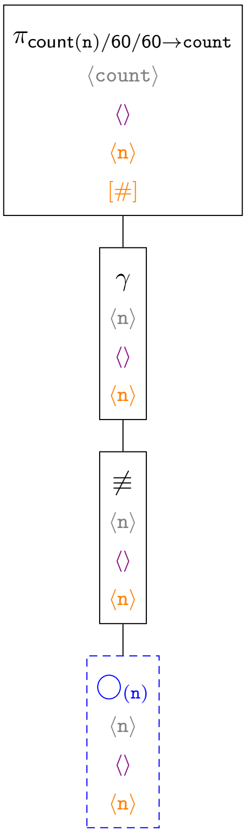
Relational algebra expression for search-based evaluation (Support multiple divisions in aggregate function)

$$r = \pi_{\texttt{count(n)/60/60}\rightarrow\texttt{count}}\Big(\gamma\Big(\not\equiv\Big(\bigcirc_{\texttt{(n)}}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Support multiple divisions in aggregate function)


$\pi_{\texttt{count(n)/60/60}\rightarrow\texttt{count}}$
⟨count⟩
γ
⟨n⟩
$\not\equiv$
⟨n⟩
$\bigcirc_{\texttt{(n)}}$
⟨n⟩

Incremental relational algebra tree (Support multiple divisions in aggregate function)



A.1.2 Support column renaming for aggregates as well

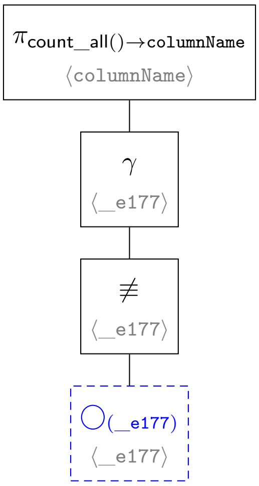
Query specification (Support column renaming for aggregates as well)

```
MATCH ()
RETURN count(*) AS columnName
```

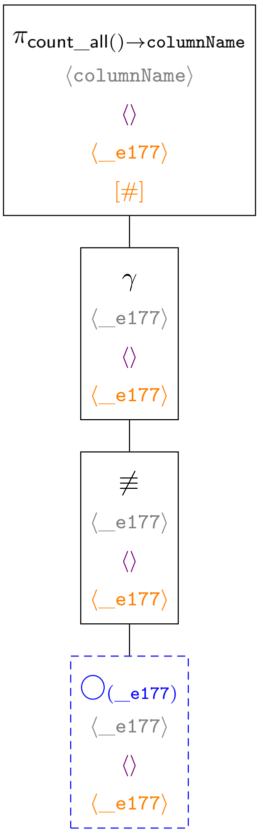
Relational algebra expression for search-based evaluation (Support column renaming for aggregates as well)

$$r = \pi_{\texttt{count\_all()}\rightarrow\texttt{columnName}}\Big(\gamma\Big(\not\equiv\Big(\bigcirc_{(\_e177)}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Support column renaming for aggregates as well)



Incremental relational algebra tree (Support column renaming for aggregates as well)



A.1.3 Aggregates inside normal functions

Query specification (Aggregates inside normal functions)

```
MATCH (a)
RETURN size(collect(a))
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.1.4 Handle aggregates inside non-aggregate expressions

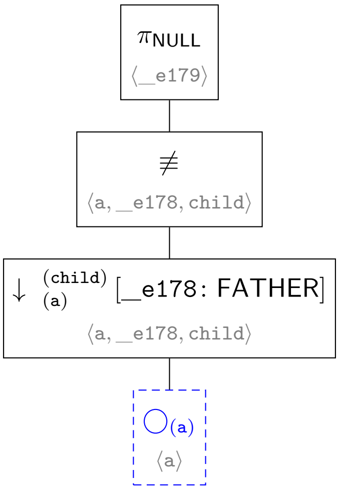
Query specification (Handle aggregates inside non-aggregate expressions)

```
MATCH (a {name: 'Andres'})<-[:FATHER]-(child)
RETURN {foo: a.name='Andres', kids: collect(child.name)}
```

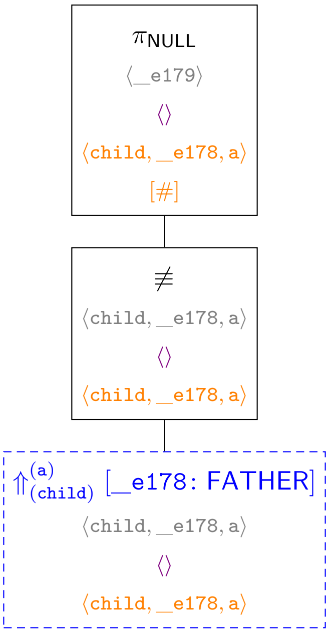
Relational algebra expression for search-based evaluation (Handle aggregates inside non-aggregate expressions)

$$r = \pi_{\mathsf{NULL}}\left(\not\equiv\left(\downarrow^{(\mathtt{child})}_{(\mathtt{a})}[\_e178\colon \mathsf{FATHER}]\left(\bigcirc_{(\mathtt{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Handle aggregates inside non-aggregate expressions)



Incremental relational algebra tree (Handle aggregates inside non-aggregate expressions)



A.1.5 Count nodes

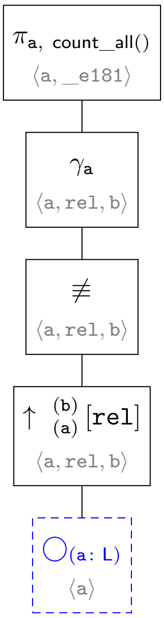
Query specification (Count nodes)

```
MATCH (a:L)-[rel]->(b)
RETURN a, count(*)
```

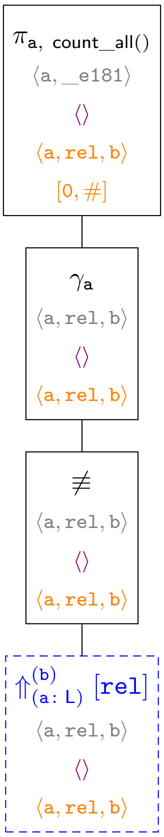
Relational algebra expression for search-based evaluation (Count nodes)

$$r = \pi_{\mathsf{a},\ \mathsf{count\_all}()}\left(\gamma_{\mathsf{a}}\left(\not\equiv\left(\uparrow\ ^{(\mathsf{b})}_{(\mathsf{a})}[\mathtt{rel}]\left(\bigcirc_{(\mathsf{a}:\ \mathsf{L})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Count nodes)


$\pi_{\mathsf{a},\ \mathsf{count\_all}()}$
$\langle \mathsf{a}, \_\mathsf{e181} \rangle$
γ_{a}
$\langle \mathsf{a}, \mathtt{rel}, \mathsf{b} \rangle$
$\not\equiv$
$\langle \mathsf{a}, \mathtt{rel}, \mathsf{b} \rangle$
$\uparrow\ ^{(\mathsf{b})}_{(\mathsf{a})}[\mathtt{rel}]$
$\langle \mathsf{a}, \mathtt{rel}, \mathsf{b} \rangle$
$\bigcirc_{(\mathsf{a}:\ \mathsf{L})}$
$\langle \mathsf{a} \rangle$

Incremental relational algebra tree (Count nodes)


$\pi_{\mathsf{a},\ \mathsf{count\_all}()}$
$\langle \mathsf{a}, \_\mathsf{e181} \rangle$
$\langle \rangle$
$\langle \mathsf{a}, \mathtt{rel}, \mathsf{b} \rangle$
$[0, \#]$
γ_{a}
$\langle \mathsf{a}, \mathtt{rel}, \mathsf{b} \rangle$
$\langle \rangle$
$\langle \mathsf{a}, \mathtt{rel}, \mathsf{b} \rangle$
$\not\equiv$
$\langle \mathsf{a}, \mathtt{rel}, \mathsf{b} \rangle$
$\langle \rangle$
$\langle \mathsf{a}, \mathtt{rel}, \mathsf{b} \rangle$
$\Uparrow^{(\mathsf{b})}_{(\mathsf{a}:\ \mathsf{L})}[\mathtt{rel}]$
$\langle \mathsf{a}, \mathtt{rel}, \mathsf{b} \rangle$
$\langle \rangle$
$\langle \mathsf{a}, \mathtt{rel}, \mathsf{b} \rangle$

A.1.6 Sort on aggregate function and normal property

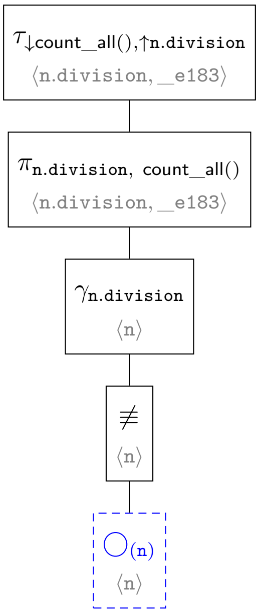
Query specification (Sort on aggregate function and normal property)

```
MATCH (n)
RETURN n.division, count(*)
ORDER BY count(*) DESC, n.division ASC
```

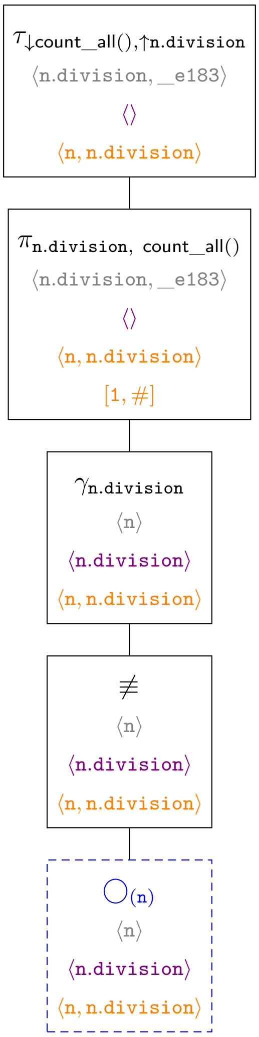
Relational algebra expression for search-based evaluation (Sort on aggregate function and normal property)

$$r = \tau_{\downarrow \mathsf{count\_all()}, \uparrow \mathtt{n.division}}\left(\pi_{\mathtt{n.division},\ \mathsf{count\_all()}}\left(\gamma_{\mathtt{n.division}}\left(\not\equiv\left(\bigcirc_{(\mathtt{n})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Sort on aggregate function and normal property)



Incremental relational algebra tree (Sort on aggregate function and normal property)



A.1.7 Aggregate on property

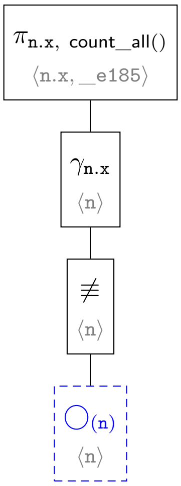
Query specification (Aggregate on property)

```
MATCH (n)
RETURN n.x, count(*)
```

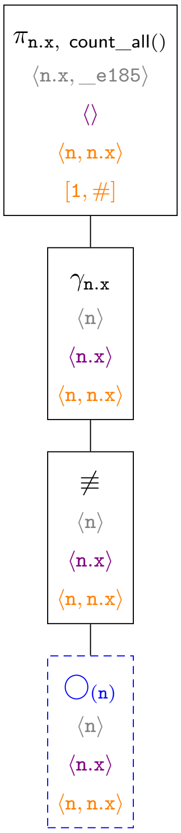
Relational algebra expression for search-based evaluation (Aggregate on property)

$$r = \pi_{\text{n.x, count\_all()}}\left(\gamma_{\text{n.x}}\left(\not\equiv\left(\bigcirc_{\text{(n)}}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Aggregate on property)



Incremental relational algebra tree (Aggregate on property)



A.1.8 Count non-null values

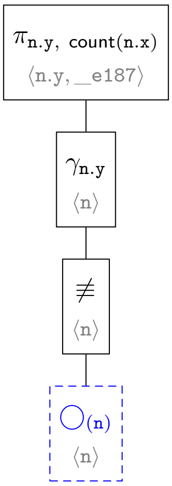
Query specification (Count non-null values)

```
MATCH (n)
RETURN n.y, count(n.x)
```

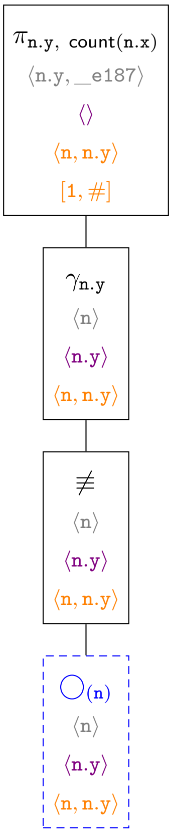
Relational algebra expression for search-based evaluation (Count non-null values)

$$r = \pi_{\texttt{n.y},\ \mathsf{count}(\texttt{n.x})}\Big(\gamma_{\texttt{n.y}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{n})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Count non-null values)



Incremental relational algebra tree (Count non-null values)



A.1.9 Sum non-null values

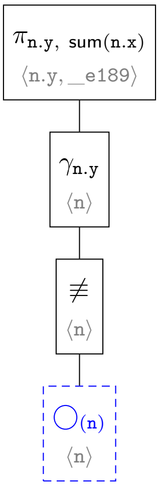
Query specification (Sum non-null values)

```
MATCH (n)
RETURN n.y, sum(n.x)
```

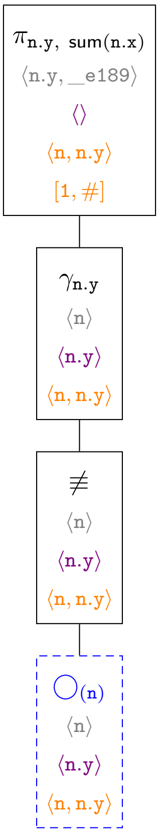
Relational algebra expression for search-based evaluation (Sum non-null values)

$$r = \pi_{\mathtt{n.y},\ \mathsf{sum}(\mathtt{n.x})}\Big(\gamma_{\mathtt{n.y}}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Sum non-null values)



Incremental relational algebra tree (Sum non-null values)



A.1.10 Handle aggregation on functions

Query specification (Handle aggregation on functions)

```
MATCH p=(a:L)-[*]->(b)
RETURN b, avg(length(p))
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.1.11 Distinct on unbound node

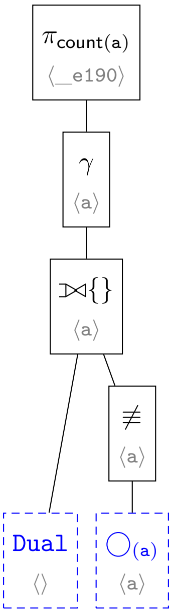
Query specification (Distinct on unbound node)

```
OPTIONAL MATCH (a)
RETURN count(DISTINCT a)
```

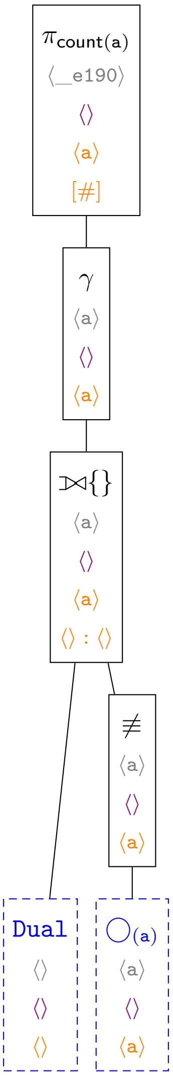
Relational algebra expression for search-based evaluation (Distinct on unbound node)

$$r = \pi_{\mathsf{count(a)}}\Big(\gamma\Big(\mathtt{Dual} ⟕ \not\equiv \Big(\bigcirc_{(\mathsf{a})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Distinct on unbound node)



Incremental relational algebra tree (Distinct on unbound node)



A.1.12 Distinct on null

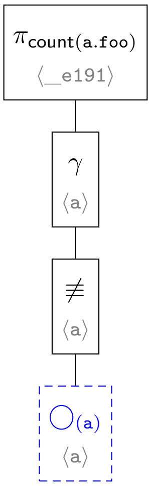
Query specification (Distinct on null)

```
MATCH (a)
RETURN count(DISTINCT a.foo)
```

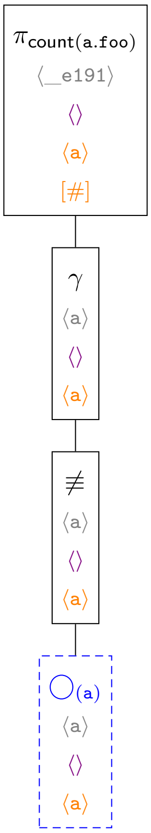
Relational algebra expression for search-based evaluation (Distinct on null)

$$r = \pi_{\text{count(a.foo)}}\Big(\gamma\Big(\not\equiv\Big(\bigcirc_{\text{(a)}}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Distinct on null)



Incremental relational algebra tree (Distinct on null)



A.1.13 Collect distinct nulls

Query specification (Collect distinct nulls)

```
UNWIND [null, null] AS x
RETURN collect(DISTINCT x) AS c
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.1.14 Collect distinct values mixed with nulls

Query specification (Collect distinct values mixed with nulls)

```
UNWIND [null, 1, null] AS x
RETURN collect(DISTINCT x) AS c
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.1.15 Aggregate on list values

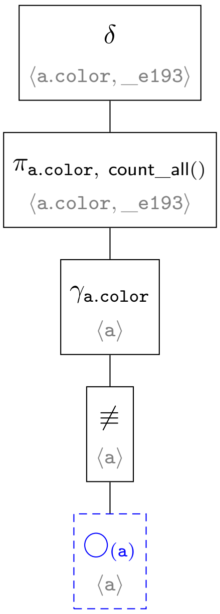
Query specification (Aggregate on list values)

```
MATCH (a)
RETURN DISTINCT a.color, count(*)
```

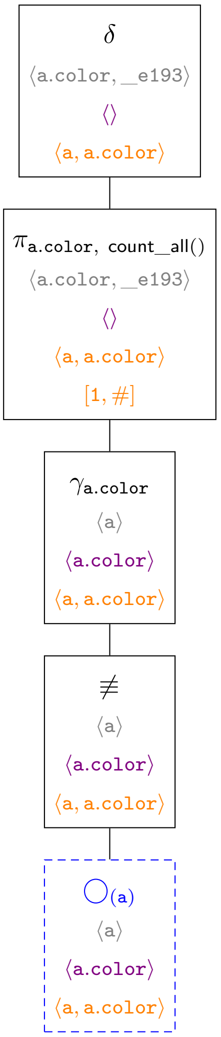
Relational algebra expression for search-based evaluation (Aggregate on list values)

$$r = \delta\left(\pi_{\texttt{a.color},\ \mathsf{count\_all()}}\left(\gamma_{\texttt{a.color}}\left(\not\equiv\left(\bigcirc_{(\texttt{a})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Aggregate on list values)



Incremental relational algebra tree (Aggregate on list values)



A.1.16 Aggregates with arithmetics

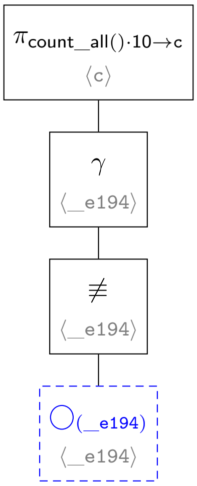
Query specification (Aggregates with arithmetics)

```
MATCH ()
RETURN count(*) * 10 AS c
```

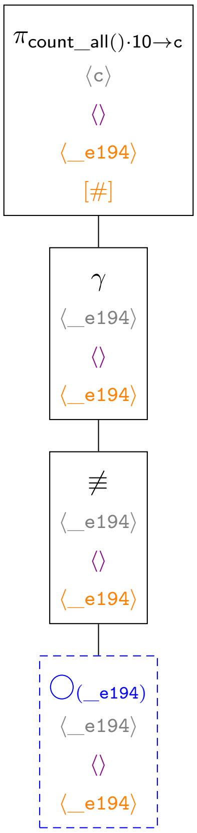
Relational algebra expression for search-based evaluation (Aggregates with arithmetics)

$$r = \pi_{\mathsf{count\_all()} \cdot 10 \to \mathsf{c}}\Big(\gamma\Big(\not\equiv\Big(\bigcirc_{(\mathsf{\_e194})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Aggregates with arithmetics)



Incremental relational algebra tree (Aggregates with arithmetics)



A.1.17 Aggregates ordered by arithmetics

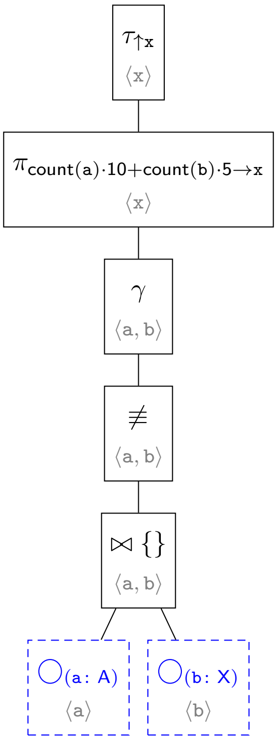
Query specification (Aggregates ordered by arithmetics)

```
MATCH (a:A), (b:X)
RETURN count(a) * 10 + count(b) * 5 AS x
ORDER BY x
```

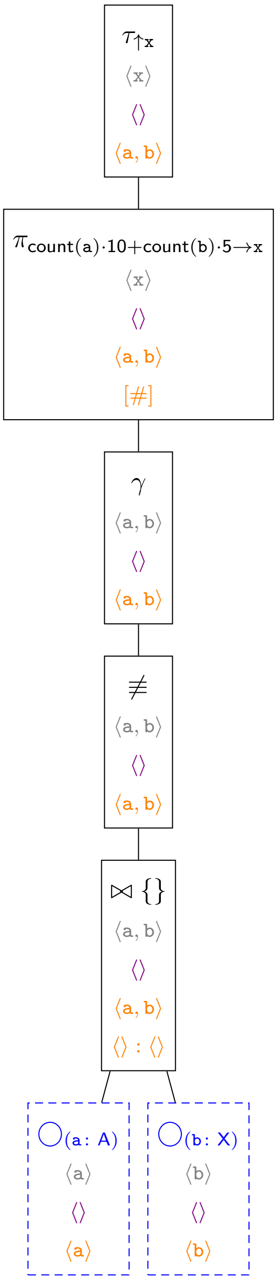
Relational algebra expression for search-based evaluation (Aggregates ordered by arithmetics)

$$r = \tau_{\uparrow \mathtt{x}}\Big(\pi_{\mathsf{count}(\mathtt{a})\cdot 10+\mathsf{count}(\mathtt{b})\cdot 5\rightarrow \mathtt{x}}\Big(\gamma\Big(\not\equiv\Big(\bigcirc_{(\mathtt{a}:\ \mathsf{A})} \bowtie \bigcirc_{(\mathtt{b}:\ \mathsf{X})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Aggregates ordered by arithmetics)


$\tau_{\uparrow \mathtt{x}}$
⟨x⟩
$\pi_{\mathsf{count}(\mathtt{a})\cdot 10+\mathsf{count}(\mathtt{b})\cdot 5\rightarrow \mathtt{x}}$
⟨x⟩
γ
⟨a, b⟩
$\not\equiv$
⟨a, b⟩
$\bowtie$ {}
⟨a, b⟩
$\bigcirc_{(\mathtt{a}:\ \mathsf{A})}$
⟨a⟩
$\bigcirc_{(\mathtt{b}:\ \mathsf{X})}$
⟨b⟩

Incremental relational algebra tree (Aggregates ordered by arithmetics)



A.1.18 Multiple aggregates on same variable

Query specification (Multiple aggregates on same variable)

```
MATCH (n)
RETURN count(n), collect(n)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.1.19 Simple counting of nodes

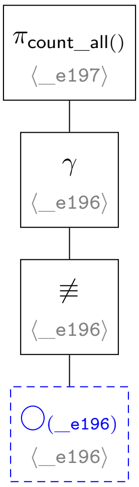
Query specification (Simple counting of nodes)

```
MATCH ()
RETURN count(*)
```

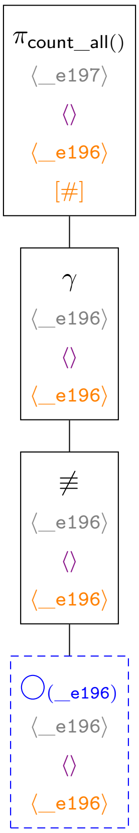
Relational algebra expression for search-based evaluation (Simple counting of nodes)

$$r = \pi_{\mathsf{count\_all()}}\Big(\gamma\Big(\not\equiv\Big(\bigcirc_{(\mathsf{\_e196})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Simple counting of nodes)



Incremental relational algebra tree (Simple counting of nodes)



A.1.20 Aggregation of named paths

Query specification (Aggregation of named paths)

```
MATCH p = (a)-[*]->(b)
RETURN collect(nodes(p)) AS paths, length(p) AS l
ORDER BY l
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.1.21 Aggregation with 'min()'

Query specification (Aggregation with 'min()')

```
MATCH p = (a:T {name: 'a'})-[:R*]->(other:T)
WHERE other <> a
WITH a, other, min(length(p)) AS len
RETURN a.name AS name, collect(other.name) AS others, len
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.1.22 Handle subexpression in aggregation also occurring as standalone expression with nested aggregation in a literal map

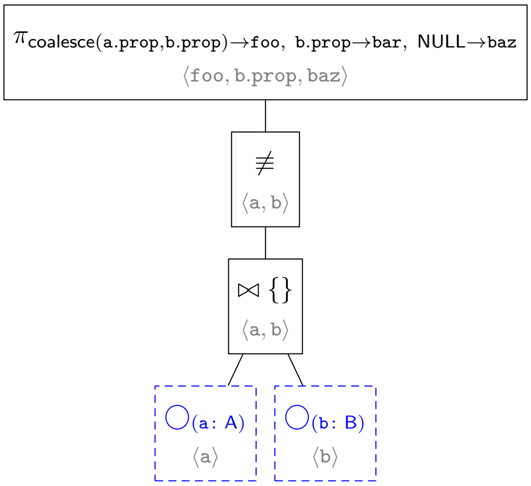
Query specification (Handle subexpression in aggregation also occurring as standalone expression with nested aggregation in a literal map)

```
MATCH (a:A), (b:B)
RETURN coalesce(a.prop, b.prop) AS foo,
b.prop AS bar,
{y: count(b)} AS baz
```

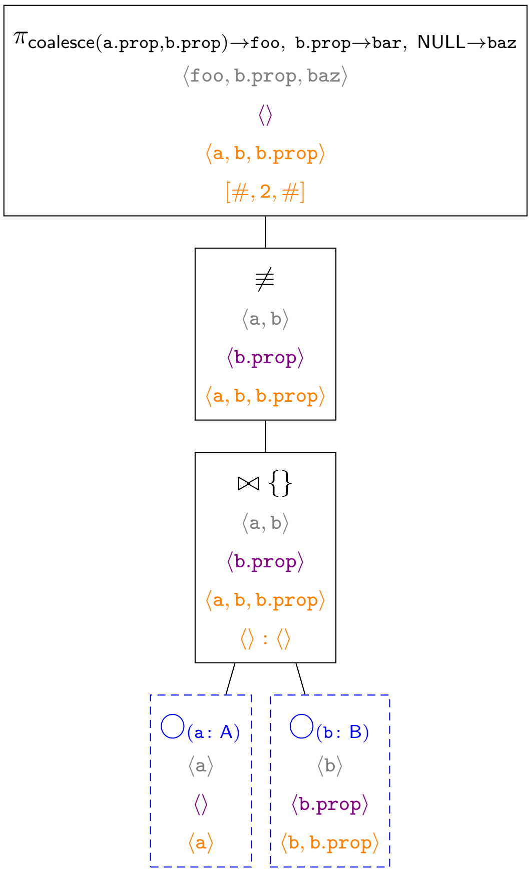
Relational algebra expression for search-based evaluation (Handle subexpression in aggregation also occurring as standalone expression with nested aggregation in a literal map)

$$r = \pi_{\mathsf{coalesce}(\mathtt{a.prop},\mathtt{b.prop})\rightarrow\mathtt{foo},\ \mathtt{b.prop}\rightarrow\mathtt{bar},\ \mathsf{NULL}\rightarrow\mathtt{baz}}\left(\not\equiv\left(\bigcirc_{(\mathtt{a}:\ \mathsf{A})}\bowtie\bigcirc_{(\mathtt{b}:\ \mathsf{B})}\right)\right)$$

Relational algebra tree for search-based evaluation (Handle subexpression in aggregation also occurring as standalone expression with nested aggregation in a literal map)



Incremental relational algebra tree (Handle subexpression in aggregation also occurring as standalone expression with nested aggregation in a literal map)



A.1.23 No overflow during summation

Query specification (No overflow during summation)

```
UNWIND range(1000000, 2000000) AS i
WITH i
LIMIT 3000
RETURN sum(i)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.1.24 Counting with loops

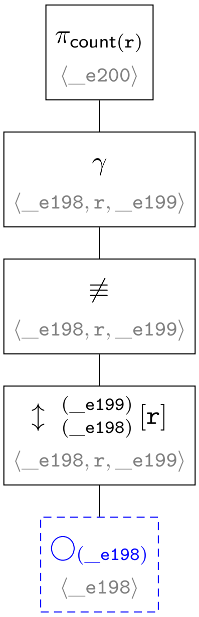
Query specification (Counting with loops)

```
MATCH ()-[r]-()
RETURN count(r)
```

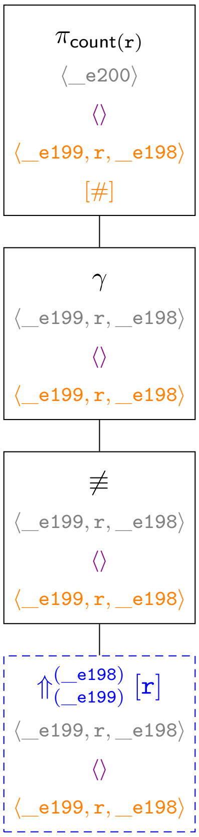
Relational algebra expression for search-based evaluation (Counting with loops)

$$r = \pi_{\mathsf{count}(\mathtt{r})}\Big(\gamma\Big(\not\equiv\Big(\updownarrow^{(\_\mathsf{e199})}_{(\_\mathsf{e198})}[\mathtt{r}]\Big(\bigcirc_{(\_\mathsf{e198})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Counting with loops)


$\pi_{\mathsf{count}(\mathtt{r})}$
⟨_e200⟩
γ
⟨_e198, r, _e199⟩
$\not\equiv$
⟨_e198, r, _e199⟩
$\updownarrow^{(\_\mathsf{e199})}_{(\_\mathsf{e198})}[\mathtt{r}]$
⟨_e198, r, _e199⟩
$\bigcirc_{(\_\mathsf{e198})}$
⟨_e198⟩

Incremental relational algebra tree (Counting with loops)



A.1.25 'max()' should aggregate strings

Query specification ('max()' should aggregate strings)

```
UNWIND ['a', 'b', 'B', null, 'abc', 'abc1'] AS i
RETURN max(i)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.1.26 'min()' should aggregate strings

Query specification ('min()' should aggregate strings)

```
UNWIND ['a', 'b', 'B', null, 'abc', 'abc1'] AS i
RETURN min(i)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.2 ColumnNameAcceptance

Progress:

1 of 4

A.2.1 Keeping used expression 1

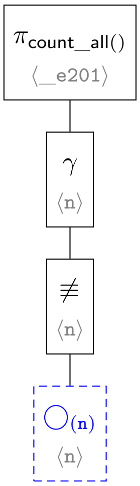
Query specification (Keeping used expression 1)

```
MATCH (n)
RETURN cOuNt( * )
```

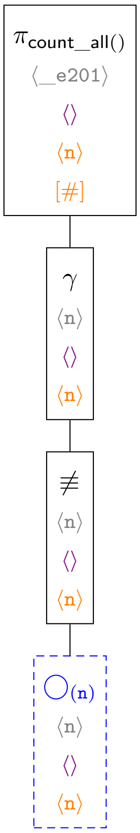
Relational algebra expression for search-based evaluation (Keeping used expression 1)

$$r = \pi_{\mathsf{count\_all()}}\Big(\gamma\Big(\not\equiv\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Keeping used expression 1)



Incremental relational algebra tree (Keeping used expression 1)



A.2.2 Keeping used expression 2

Query specification (Keeping used expression 2)

```
MATCH p = (n)-->(b)
RETURN nOdEs( p )
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.2.3 Keeping used expression 3

Query specification (Keeping used expression 3)

```
MATCH p = (n)-->(b)
RETURN coUnt( dIstInct p )
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.2.4 Keeping used expression 4

Query specification (Keeping used expression 4)

```
MATCH p = (n)-->(b)
RETURN aVg(    n.aGe     )
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.3 Comparability

Progress:

0 of 2

A.3.1 Comparing strings and integers using > in an AND'd predicate

Query specification (Comparing strings and integers using > in an AND'd predicate)

```
MATCH (:Root)-->(i:Child)
WHERE exists(i.id) AND i.id > 'x'
RETURN i.id
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.3.2 Comparing strings and integers using > in a OR’d predicate

Query specification **(Comparing strings and integers using > in a OR’d predicate)**

```
MATCH (:Root)-->(i:Child)
WHERE NOT exists(i.id) OR i.id > 'x'
RETURN i.id
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.4 ComparisonOperatorAcceptance

Progress:

0 of 10

A.4.1 Handling numerical ranges 1

Query specification **(Handling numerical ranges 1)**

```
MATCH (n)
WHERE 1 < n.value < 3
RETURN n.value
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.4.2 Handling numerical ranges 2

Query specification **(Handling numerical ranges 2)**

```
MATCH (n)
WHERE 1 < n.value <= 3
RETURN n.value
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.4.3 Handling numerical ranges 3

Query specification **(Handling numerical ranges 3)**

```
MATCH (n)
WHERE 1 <= n.value < 3
RETURN n.value
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.4.4 Handling numerical ranges 4

Query specification **(Handling numerical ranges 4)**

```
MATCH (n)
WHERE 1 <= n.value <= 3
RETURN n.value
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.4.5 Handling string ranges 1

Query specification **(Handling string ranges 1)**

```
MATCH (n)
WHERE 'a' < n.value < 'c'
RETURN n.value
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.4.6 Handling string ranges 2

Query specification **(Handling string ranges 2)**

```
MATCH (n)
WHERE 'a' < n.value <= 'c'
RETURN n.value
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.4.7 Handling string ranges 3

Query specification **(Handling string ranges 3)**

```
MATCH (n)
WHERE 'a' <= n.value < 'c'
RETURN n.value
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.4.8 Handling string ranges 4

Query specification **(Handling string ranges 4)**

```
MATCH (n)
WHERE 'a' <= n.value <= 'c'
RETURN n.value
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.4.9 Handling empty range

Query specification (Handling empty range)

```
MATCH (n)
WHERE 10 < n.value <= 3
RETURN n.value
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.4.10 Handling long chains of operators

Query specification (Handling long chains of operators)

```
MATCH (n)-->(m)
WHERE n.prop1 < m.prop1 = n.prop2 <> m.prop2
RETURN labels(m)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.5 Create

Progress:

0 of 0

A.6 CreateAcceptance

Progress:

0 of 0

A.7 DeleteAcceptance

Progress:

0 of 0

A.8 EqualsAcceptance

Progress:

0 of 5

A.8.1 Number-typed integer comparison

Query specification (Number-typed integer comparison)

```
WITH collect([0, 0.0]) AS numbers
UNWIND numbers AS arr
WITH arr[0] AS expected
MATCH (n) WHERE toInteger(n.id) = expected
RETURN n
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.8.2 Number-typed float comparison

Query specification (Number-typed float comparison)

```
WITH collect([0.5, 0]) AS numbers
UNWIND numbers AS arr
WITH arr[0] AS expected
MATCH (n) WHERE toInteger(n.id) = expected
RETURN n
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.8.3 Any-typed string comparison

Query specification (Any-typed string comparison)

```
WITH collect(['0', 0]) AS things
UNWIND things AS arr
WITH arr[0] AS expected
MATCH (n) WHERE toInteger(n.id) = expected
RETURN n
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.8.4 Comparing nodes to nodes

Query specification (Comparing nodes to nodes)

```
MATCH (a)
WITH a
MATCH (b)
WHERE a = b
RETURN count(b)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.8.5 Comparing relationships to relationships

Query specification (Comparing relationships to relationships)

```
MATCH ()-[a]->()
WITH a
MATCH ()-[b]->()
WHERE a = b
RETURN count(b)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.9 ExpressionAcceptance

Progress:

11 of 11

A.9.1 IN should work with nested list subscripting

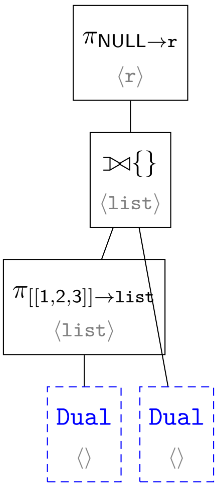
Query specification (IN should work with nested list subscripting)

```
WITH [[1, 2, 3]] AS list
RETURN 3 IN list[0] AS r
```

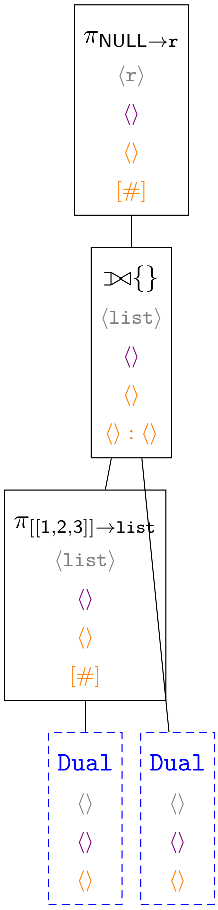
Relational algebra expression for search-based evaluation (IN should work with nested list subscripting)

$$r = \pi_{\mathsf{NULL}\rightarrow\mathtt{r}}\left(\pi_{[[1,2,3]]\rightarrow\mathtt{list}}\left(\mathtt{Dual}\right) \mathbin{⟕} \mathtt{Dual}\right)$$

Relational algebra tree for search-based evaluation (IN should work with nested list subscripting)


$\pi_{\mathsf{NULL}\rightarrow\mathtt{r}}$
⟨r⟩
⟕{}
⟨list⟩
$\pi_{[[1,2,3]]\rightarrow\mathtt{list}}$
⟨list⟩
Dual
⟨⟩
Dual
⟨⟩

Incremental relational algebra tree (IN should work with nested list subscripting)



$\pi_{\text{NULL}\rightarrow\texttt{r}}$
⟨r⟩
⟨⟩
⟨⟩
[#]
⋈{}
⟨list⟩
⟨⟩
⟨⟩
⟨⟩ : ⟨⟩
$\pi_{[[1,2,3]]\rightarrow\texttt{list}}$
⟨list⟩
⟨⟩
⟨⟩
[#]
Dual
⟨⟩
⟨⟩
⟨⟩
Dual
⟨⟩
⟨⟩
⟨⟩

A.9.2 IN should work with nested literal list subscripting

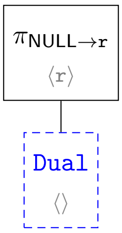
Query specification (IN should work with nested literal list subscripting)

```
RETURN 3 IN [[1, 2, 3]][0] AS r
```

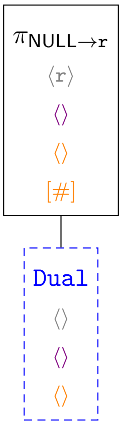
Relational algebra expression for search-based evaluation (IN should work with nested literal list subscripting)

$$r = \pi_{\text{NULL}\rightarrow\texttt{r}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (IN should work with nested literal list subscripting)



Incremental relational algebra tree (IN should work with nested literal list subscripting)



A.9.3 IN should work with list slices

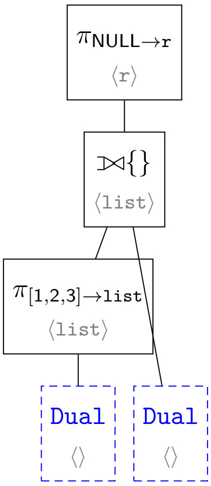
Query specification (IN should work with list slices)

```
WITH [1, 2, 3] AS list
RETURN 3 IN list[0..1] AS r
```

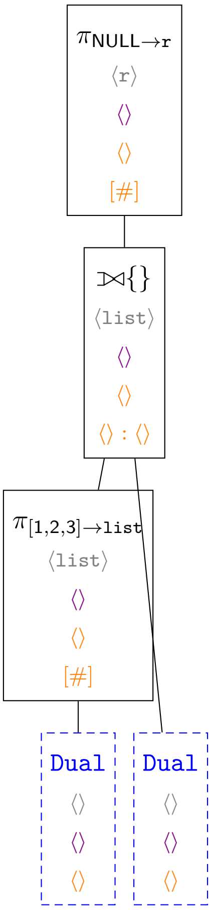
Relational algebra expression for search-based evaluation (IN should work with list slices)

$$r = \pi_{\text{NULL}\rightarrow\texttt{r}}\left(\pi_{[1,2,3]\rightarrow\texttt{list}}\left(\texttt{Dual}\right) \rtimes \texttt{Dual}\right)$$

Relational algebra tree for search-based evaluation (IN should work with list slices)


$\pi_{\text{NULL}\rightarrow\texttt{r}}$
⟨r⟩
$\rtimes\{\}$
⟨list⟩
$\pi_{[1,2,3]\rightarrow\texttt{list}}$
⟨list⟩
Dual
⟨⟩
Dual
⟨⟩

Incremental relational algebra tree (IN should work with list slices)


$\pi_{\mathsf{NULL}\rightarrow\mathtt{r}}$
⟨r⟩
⟨⟩
⟨⟩
[#]
⋈{}
⟨list⟩
⟨⟩
⟨⟩
⟨⟩ : ⟨⟩
$\pi_{[1,2,3]\rightarrow\mathtt{list}}$
⟨list⟩
⟨⟩
⟨⟩
[#]
Dual
⟨⟩
⟨⟩
⟨⟩
Dual
⟨⟩
⟨⟩
⟨⟩

A.9.4 IN should work with literal list slices

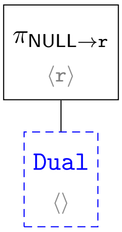
Query specification (IN should work with literal list slices)

```
RETURN 3 IN [1, 2, 3][0..1] AS r
```

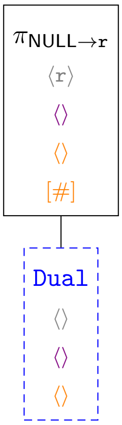
Relational algebra expression for search-based evaluation (IN should work with literal list slices)

$$r = \pi_{\mathsf{NULL}\rightarrow\mathtt{r}}\Big(\mathtt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (IN should work with literal list slices)



Incremental relational algebra tree (IN should work with literal list slices)



A.9.5]

Execute n[0]

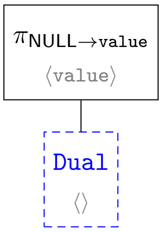
Query specification (Execute n[0])

```
RETURN [1, 2, 3][0] AS value
```

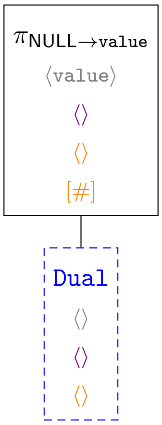
Relational algebra expression for search-based evaluation (Execute n[0])

$$r = \pi_{\mathsf{NULL}\rightarrow\mathtt{value}}\Big(\mathtt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Execute n[0])



Incremental relational algebra tree (Execute n[0])



A.9.6 i

n read queries]Execute n['name'] in read queries

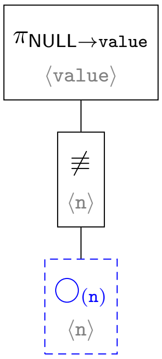
Query specification (Execute n['name'] in read queries)

```
MATCH (n {name: 'Apa'})
RETURN n['nam' + 'e'] AS value
```

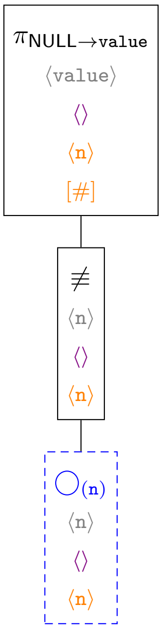
Relational algebra expression for search-based evaluation (Execute n['name'] in read queries)

$$r = \pi_{\mathsf{NULL} \rightarrow \mathtt{value}} \left(\not\equiv \left(\bigcirc_{(\mathtt{n})} \right) \right)$$

Relational algebra tree for search-based evaluation (Execute n['name'] in read queries)


$\pi_{\mathsf{NULL} \rightarrow \mathtt{value}}$
⟨value⟩
$\not\equiv$
⟨n⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩

Incremental relational algebra tree (Execute n['name'] in read queries)


$\pi_{\mathsf{NULL} \rightarrow \mathtt{value}}$
⟨value⟩
⟨⟩
⟨n⟩
[#]
$\not\equiv$
⟨n⟩
⟨⟩
⟨n⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩
⟨⟩
⟨n⟩

A.9.7 Use dynamic property lookup based on parameters when there is no type information

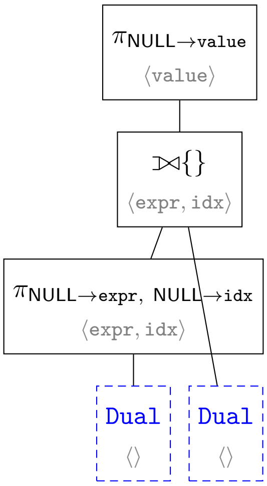
Query specification (Use dynamic property lookup based on parameters when there is no type information)

```
WITH $expr AS expr, $idx AS idx
RETURN expr[idx] AS value
```

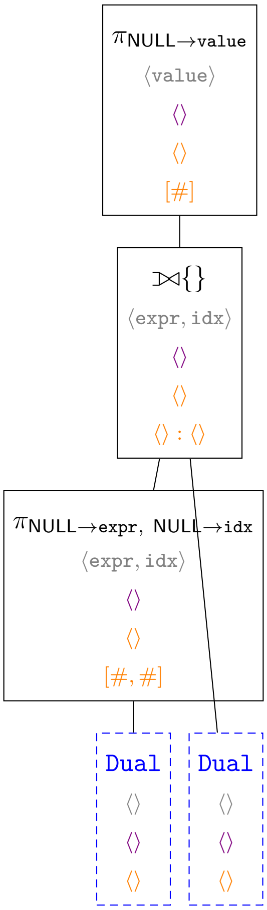
Relational algebra expression for search-based evaluation (Use dynamic property lookup based on parameters when there is no type information)

$$r = \pi_{\mathsf{NULL}\rightarrow\mathtt{value}}\left(\pi_{\mathsf{NULL}\rightarrow\mathtt{expr},\ \mathsf{NULL}\rightarrow\mathtt{idx}}\left(\mathtt{Dual}\right) \rtimes\!\!\!\ltimes \mathtt{Dual}\right)$$

Relational algebra tree for search-based evaluation (Use dynamic property lookup based on parameters when there is no type information)


$\pi_{\mathsf{NULL}\rightarrow\mathtt{value}}$
⟨value⟩
⋊⋉{}
⟨expr, idx⟩
$\pi_{\mathsf{NULL}\rightarrow\mathtt{expr},\ \mathsf{NULL}\rightarrow\mathtt{idx}}$
⟨expr, idx⟩
Dual
⟨⟩
Dual
⟨⟩

Incremental relational algebra tree (Use dynamic property lookup based on parameters when there is no type information)


$\pi_{\mathsf{NULL}\rightarrow\mathtt{value}}$
⟨value⟩
⟨⟩
⟨⟩
[#]
⋊⋉{}
⟨expr, idx⟩
⟨⟩
⟨⟩
⟨⟩ : ⟨⟩
$\pi_{\mathsf{NULL}\rightarrow\mathtt{expr},\ \mathsf{NULL}\rightarrow\mathtt{idx}}$
⟨expr, idx⟩
⟨⟩
⟨⟩
[#, #]
Dual
⟨⟩
⟨⟩
⟨⟩
Dual
⟨⟩
⟨⟩
⟨⟩

A.9.8 Use dynamic property lookup based on parameters when there is rhs type information

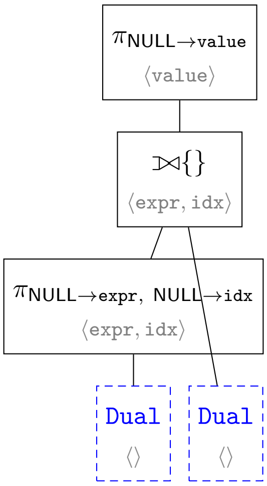
Query specification (Use dynamic property lookup based on parameters when there is rhs type information)

```
WITH $expr AS expr, $idx AS idx
RETURN expr[toString(idx)] AS value
```

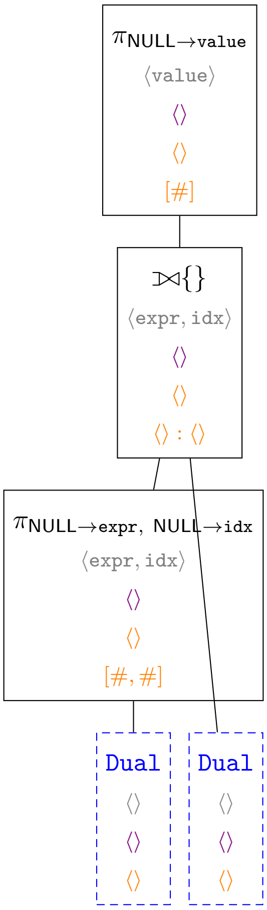
Relational algebra expression for search-based evaluation (Use dynamic property lookup based on parameters when there is rhs type information)

$$r = \pi_{\mathsf{NULL}\rightarrow\mathtt{value}}\left(\pi_{\mathsf{NULL}\rightarrow\mathtt{expr},\ \mathsf{NULL}\rightarrow\mathtt{idx}}\left(\mathtt{Dual}\right) \bowtie \mathtt{Dual}\right)$$

Relational algebra tree for search-based evaluation (Use dynamic property lookup based on parameters when there is rhs type information)


$\pi_{\mathsf{NULL}\rightarrow\mathtt{value}}$
⟨value⟩
$\bowtie\{\}$
⟨expr, idx⟩
$\pi_{\mathsf{NULL}\rightarrow\mathtt{expr},\ \mathsf{NULL}\rightarrow\mathtt{idx}}$
⟨expr, idx⟩
Dual
⟨⟩
Dual
⟨⟩

Incremental relational algebra tree (Use dynamic property lookup based on parameters when there is rhs type information)


$\pi_{\mathsf{NULL} \rightarrow \mathtt{value}}$
$\langle \mathtt{value} \rangle$
$\langle \rangle$
$\langle \rangle$
$[\#]$
$\bowtie \{\}$
$\langle \mathtt{expr}, \mathtt{idx} \rangle$
$\langle \rangle$
$\langle \rangle$
$\langle \rangle : \langle \rangle$
$\pi_{\mathsf{NULL} \rightarrow \mathtt{expr},\ \mathsf{NULL} \rightarrow \mathtt{idx}}$
$\langle \mathtt{expr}, \mathtt{idx} \rangle$
$\langle \rangle$
$\langle \rangle$
$[\#, \#]$
Dual
$\langle \rangle$
$\langle \rangle$
$\langle \rangle$
Dual
$\langle \rangle$
$\langle \rangle$
$\langle \rangle$

A.9.9 Use collection lookup based on parameters when there is no type information

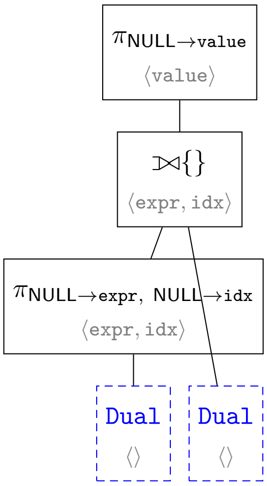
Query specification (Use collection lookup based on parameters when there is no type information)

```
WITH $expr AS expr, $idx AS idx
RETURN expr[idx] AS value
```

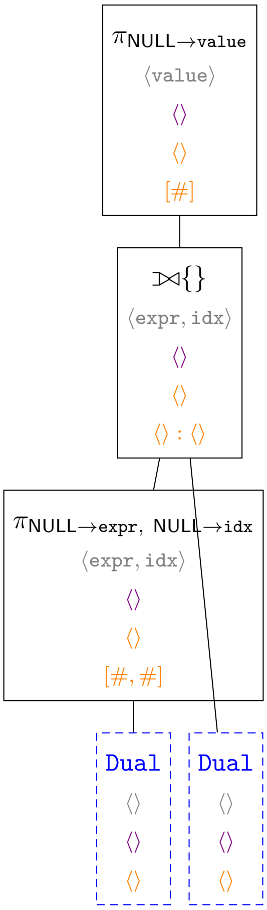
Relational algebra expression for search-based evaluation (Use collection lookup based on parameters when there is no type information)

$$r = \pi_{\mathsf{NULL} \rightarrow \mathtt{value}}\left(\pi_{\mathsf{NULL} \rightarrow \mathtt{expr},\ \mathsf{NULL} \rightarrow \mathtt{idx}}\left(\mathtt{Dual}\right) \bowtie \mathtt{Dual}\right)$$

Relational algebra tree for search-based evaluation (Use collection lookup based on parameters when there is no type information)


$\pi_{\text{NULL}\to\texttt{value}}$
$\langle\texttt{value}\rangle$
$\bowtie\{\}$
$\langle\texttt{expr}, \texttt{idx}\rangle$
$\pi_{\text{NULL}\to\texttt{expr},\ \text{NULL}\to\texttt{idx}}$
$\langle\texttt{expr}, \texttt{idx}\rangle$
Dual
$\langle\rangle$
Dual
$\langle\rangle$

Incremental relational algebra tree (Use collection lookup based on parameters when there is no type information)


$\pi_{\text{NULL}\to\texttt{value}}$
$\langle\texttt{value}\rangle$
$\langle\rangle$
$\langle\rangle$
[#]
$\bowtie\{\}$
$\langle\texttt{expr}, \texttt{idx}\rangle$
$\langle\rangle$
$\langle\rangle$
$\langle\rangle : \langle\rangle$
$\pi_{\text{NULL}\to\texttt{expr},\ \text{NULL}\to\texttt{idx}}$
$\langle\texttt{expr}, \texttt{idx}\rangle$
$\langle\rangle$
$\langle\rangle$
[#, #]
Dual
$\langle\rangle$
$\langle\rangle$
$\langle\rangle$
Dual
$\langle\rangle$
$\langle\rangle$
$\langle\rangle$

A.9.10 Use collection lookup based on parameters when there is lhs type information

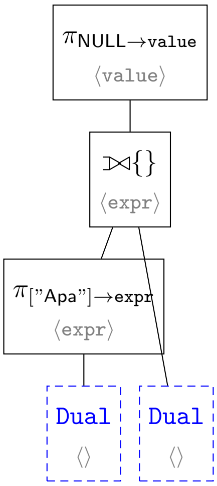
Query specification (Use collection lookup based on parameters when there is lhs type information)

```
WITH ['Apa'] AS expr
RETURN expr[$idx] AS value
```

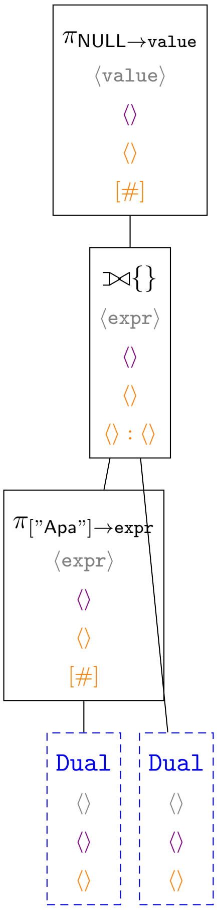
Relational algebra expression for search-based evaluation (Use collection lookup based on parameters when there is lhs type information)

$$r = \pi_{\mathsf{NULL}\rightarrow\texttt{value}}\left(\pi_{[\text{"}\mathsf{Apa}\text{"}]\rightarrow\texttt{expr}}\left(\texttt{Dual}\right) \bowtie \texttt{Dual}\right)$$

Relational algebra tree for search-based evaluation (Use collection lookup based on parameters when there is lhs type information)



Incremental relational algebra tree (Use collection lookup based on parameters when there is lhs type information)



A.9.11 Use collection lookup based on parameters when there is rhs type information

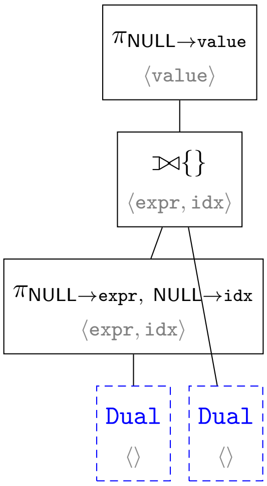
Query specification (Use collection lookup based on parameters when there is rhs type information)

```
WITH $expr AS expr, $idx AS idx
RETURN expr[toInteger(idx)] AS value
```

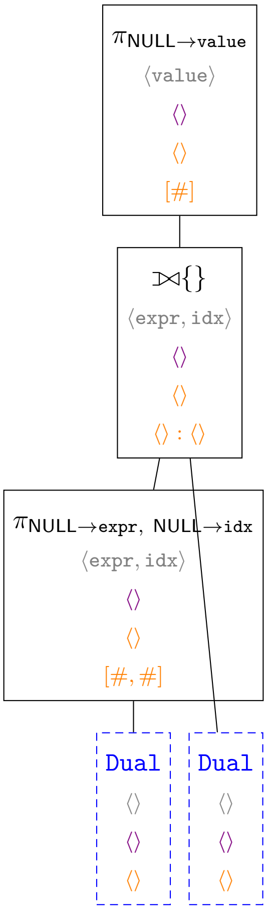
Relational algebra expression for search-based evaluation (Use collection lookup based on parameters when there is rhs type information)

$$r = \pi_{\mathsf{NULL} \rightarrow \mathtt{value}} \left(\pi_{\mathsf{NULL} \rightarrow \mathtt{expr},\ \mathsf{NULL} \rightarrow \mathtt{idx}} \left(\mathtt{Dual} \right) \bowtie \mathtt{Dual} \right)$$

Relational algebra tree for search-based evaluation (Use collection lookup based on parameters when there is rhs type information)


$\pi_{\mathsf{NULL} \rightarrow \mathtt{value}}$
⟨value⟩
$\bowtie\{\}$
⟨expr, idx⟩
$\pi_{\mathsf{NULL} \rightarrow \mathtt{expr},\ \mathsf{NULL} \rightarrow \mathtt{idx}}$
⟨expr, idx⟩
Dual
⟨⟩
Dual
⟨⟩

Incremental relational algebra tree (Use collection lookup based on parameters when there is rhs type information)



A.10 FunctionsAcceptance

Progress:

14 of 18

A.10.1 Run coalesce

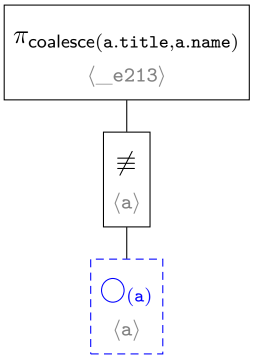
Query specification (Run coalesce)

```
MATCH (a)
RETURN coalesce(a.title, a.name)
```

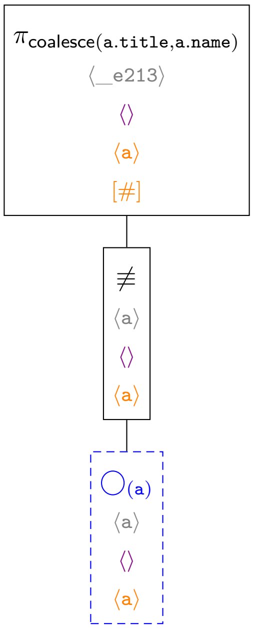
Relational algebra expression for search-based evaluation (Run coalesce)

$$r = \pi_{\text{coalesce(a.title,a.name)}}\left(\not\equiv\left(\bigcirc_{(a)}\right)\right)$$

Relational algebra tree for search-based evaluation (Run coalesce)



Incremental relational algebra tree (Run coalesce)



A.10.2 Functions should return null if they get path containing unbound

Query specification (Functions should return null if they get path containing unbound)

```
WITH null AS a
OPTIONAL MATCH p = (a)-[r]->()
RETURN length(nodes(p)), type(r), nodes(p), relationships(p)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.10.3 'split()'

Query specification ('split()')

```
UNWIND split('one1two', '1') AS item
RETURN count(item) AS item
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.10.4 'properties()' on a node

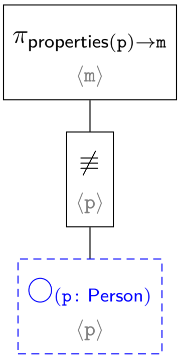
Query specification ('properties()' on a node)

```
MATCH (p:Person)
RETURN properties(p) AS m
```

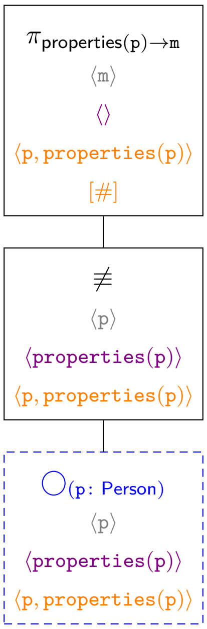
Relational algebra expression for search-based evaluation ('properties()' on a node)

$$r = \pi_{\text{properties(p)} \rightarrow \text{m}} \left(\not\equiv \left(\bigcirc_{(\text{p}:\ \text{Person})} \right) \right)$$

Relational algebra tree for search-based evaluation ('properties()' on a node)


$\pi_{\text{properties(p)} \rightarrow \text{m}}$
⟨m⟩
≢
⟨p⟩
$\bigcirc_{(\text{p}:\ \text{Person})}$
⟨p⟩

Incremental relational algebra tree ('properties()' on a node)



A.10.5 'properties()' on a relationship

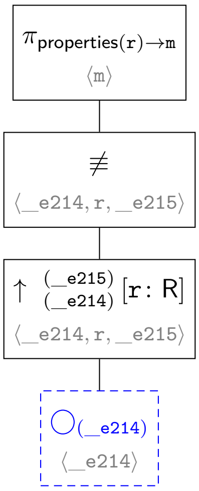
Query specification ('properties()' on a relationship)

```
MATCH ()-[r:R]->()
RETURN properties(r) AS m
```

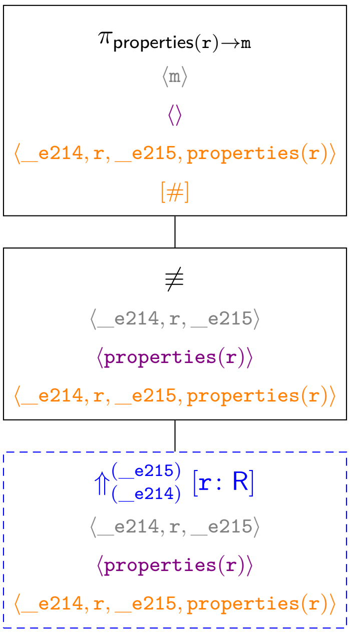
Relational algebra expression for search-based evaluation ('properties()' on a relationship)

$$r = \pi_{\mathsf{properties(r)} \rightarrow \mathtt{m}} \Big(\not\equiv \Big(\uparrow^{(\_e215)}_{(\_e214)} [\mathtt{r}: \mathsf{R}] \Big(\bigcirc_{(\_e214)} \Big) \Big) \Big)$$

Relational algebra tree for search-based evaluation ('properties()' on a relationship)


$\pi_{\mathsf{properties(r)} \rightarrow \mathtt{m}}$
$\langle \mathtt{m} \rangle$
$\not\equiv$
$\langle \_e214, \mathtt{r}, \_e215 \rangle$
$\uparrow^{(\_e215)}_{(\_e214)} [\mathtt{r}: \mathsf{R}]$
$\langle \_e214, \mathtt{r}, \_e215 \rangle$
$\bigcirc_{(\_e214)}$
$\langle \_e214 \rangle$

Incremental relational algebra tree ('properties()' on a relationship)


$\pi_{\mathsf{properties(r)} \rightarrow \mathtt{m}}$
$\langle \mathtt{m} \rangle$
$\langle \rangle$
$\langle \_e214, \mathtt{r}, \_e215, \mathtt{properties(r)} \rangle$
[#]
$\not\equiv$
$\langle \_e214, \mathtt{r}, \_e215 \rangle$
$\langle \mathtt{properties(r)} \rangle$
$\langle \_e214, \mathtt{r}, \_e215, \mathtt{properties(r)} \rangle$
$\Uparrow^{(\_e215)}_{(\_e214)} [\mathtt{r}: \mathsf{R}]$
$\langle \_e214, \mathtt{r}, \_e215 \rangle$
$\langle \mathtt{properties(r)} \rangle$
$\langle \_e214, \mathtt{r}, \_e215, \mathtt{properties(r)} \rangle$

A.10.6 'properties()' on a map

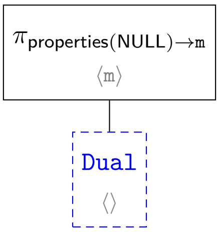
Query specification ('properties()' on a map)

```
RETURN properties({name: 'Popeye', level: 9001}) AS m
```

Relational algebra expression for search-based evaluation ('properties()' on a map)

$$r = \pi_{\mathsf{properties(NULL)} \rightarrow \mathtt{m}} \Big(\mathtt{Dual} \Big)$$

Relational algebra tree for search-based evaluation (’properties()’ on a map)



Incremental relational algebra tree (’properties()’ on a map)

Cannot visualize incremental tree.

A.10.7 ’properties()’ on null

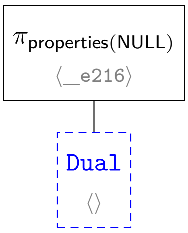
Query specification (’properties()’ on null)

```
RETURN properties(null)
```

Relational algebra expression for search-based evaluation (’properties()’ on null)

$$r = \pi_{\mathsf{properties(NULL)}}\left(\mathtt{Dual}\right)$$

Relational algebra tree for search-based evaluation (’properties()’ on null)



Incremental relational algebra tree (’properties()’ on null)

Cannot visualize incremental tree.

A.10.8 ’reverse()’

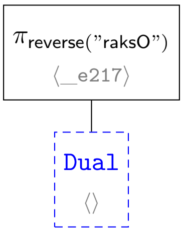
Query specification (’reverse()’)

```
RETURN reverse('raksO')
```

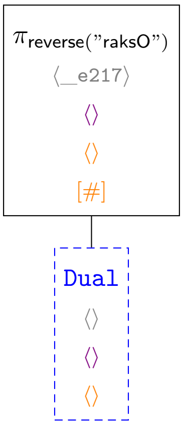
Relational algebra expression for search-based evaluation (’reverse()’)

$$r = \pi_{\mathsf{reverse(”raksO”)}}\left(\mathtt{Dual}\right)$$

Relational algebra tree for search-based evaluation (’reverse()’)



Incremental relational algebra tree ('reverse()')



A.10.9 'exists()' with dynamic property lookup

Query specification ('exists()' with dynamic property lookup)

```
MATCH (n:Person)
WHERE exists(n['prop'])
RETURN n
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.10.10 'percentileDisc()' failing in more involved query

Query specification ('percentileDisc()' failing in more involved query)

```
MATCH (n:S)
WITH n, size([(n)-->() | 1]) AS deg
WHERE deg > 2
WITH deg
LIMIT 100
RETURN percentileDisc(0.90, deg), deg
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.10.11 'type()'

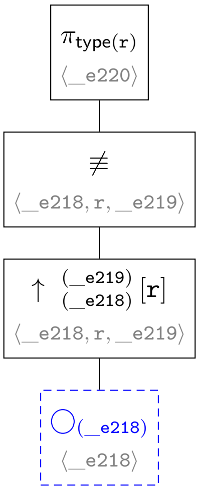
Query specification ('type()')

```
MATCH ()-[r]->()
RETURN type(r)
```

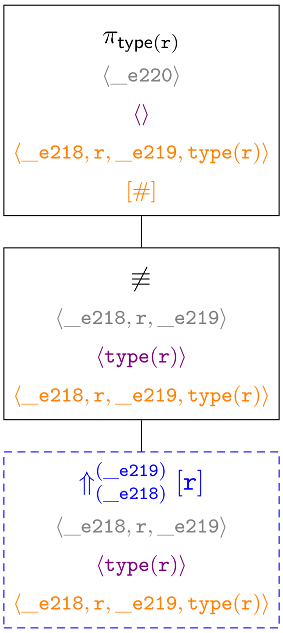
Relational algebra expression for search-based evaluation ('type()')

$$r = \pi_{\text{type(r)}}\left(\not\equiv \left(\uparrow\, {}^{(\_\text{e219})}_{(\_\text{e218})} [\mathtt{r}] \left(\bigcirc_{(\_\text{e218})} \right)\right)\right)$$

Relational algebra tree for search-based evaluation ('type()')



Incremental relational algebra tree ('type()')



$\pi_{\texttt{type(r)}}$
⟨_e220⟩
⟨⟩
⟨_e218, r, _e219, type(r)⟩
[#]
$\not\equiv$
⟨_e218, r, _e219⟩
⟨type(r)⟩
⟨_e218, r, _e219, type(r)⟩
$\Uparrow^{(\_e219)}_{(\_e218)}$ [r]
⟨_e218, r, _e219⟩
⟨type(r)⟩
⟨_e218, r, _e219, type(r)⟩

A.10.12 'type()' on two relationships

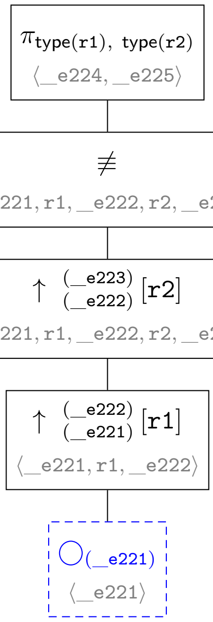
Query specification ('type()' on two relationships)

```
MATCH ()-[r1]->()-[r2]->()
RETURN type(r1), type(r2)
```

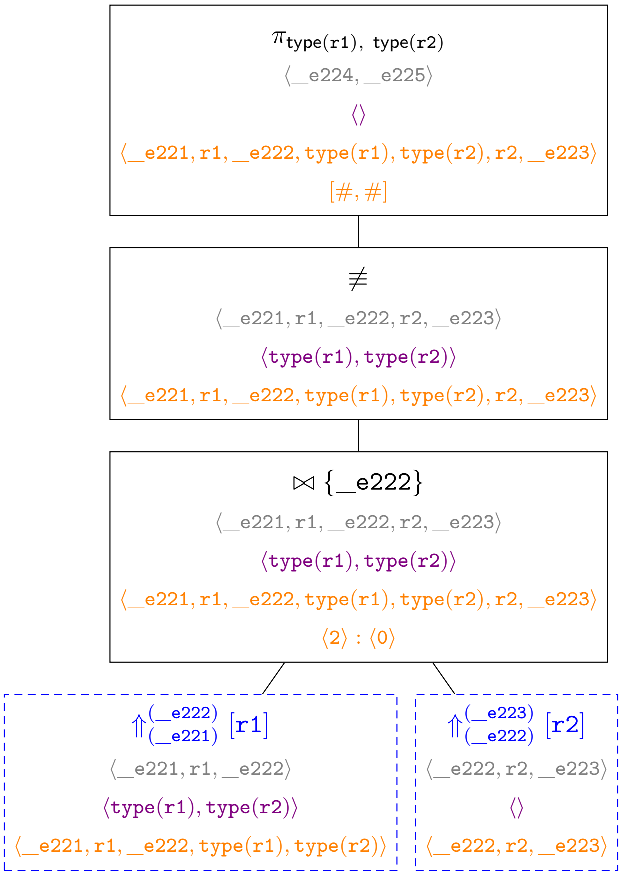
Relational algebra expression for search-based evaluation ('type()' on two relationships)

$$r = \pi_{\texttt{type(r1)},\ \texttt{type(r2)}} \Big(\not\equiv \Big(\uparrow^{(\_e223)}_{(\_e222)} [\texttt{r2}] \Big(\uparrow^{(\_e222)}_{(\_e221)} [\texttt{r1}] \Big(\bigcirc_{(\_e221)} \Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation ('type()' on two relationships)



Incremental relational algebra tree ('type()' on two relationships)


π type(r1), type(r2)
⟨_e224, _e225⟩
⟨⟩
⟨_e221, r1, _e222, type(r1), type(r2), r2, _e223⟩
[#, #]
≢
⟨_e221, r1, _e222, r2, _e223⟩
⟨type(r1), type(r2)⟩
⟨_e221, r1, _e222, type(r1), type(r2), r2, _e223⟩
⋈ {_e222}
⟨_e221, r1, _e222, r2, _e223⟩
⟨type(r1), type(r2)⟩
⟨_e221, r1, _e222, type(r1), type(r2), r2, _e223⟩
⟨2⟩ : ⟨0⟩
⇑ (_e222) (_e221) [r1]
⟨_e221, r1, _e222⟩
⟨type(r1), type(r2)⟩
⟨_e221, r1, _e222, type(r1), type(r2)⟩
⇑ (_e223) (_e222) [r2]
⟨_e222, r2, _e223⟩
⟨⟩
⟨_e222, r2, _e223⟩

A.10.13 'type()' on null relationship

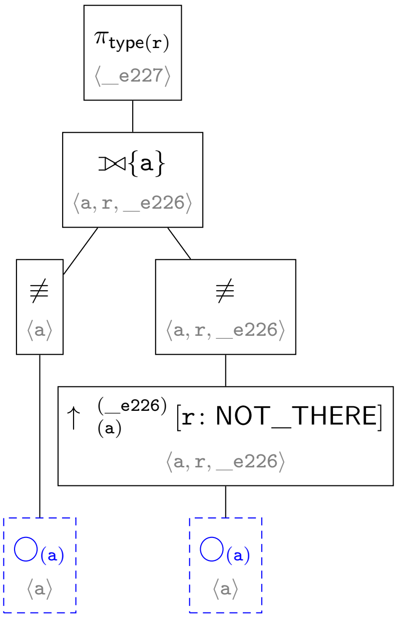
Query specification ('type()' on null relationship)

```
MATCH (a)
OPTIONAL MATCH (a)-[r:NOT_THERE]->()
RETURN type(r)
```

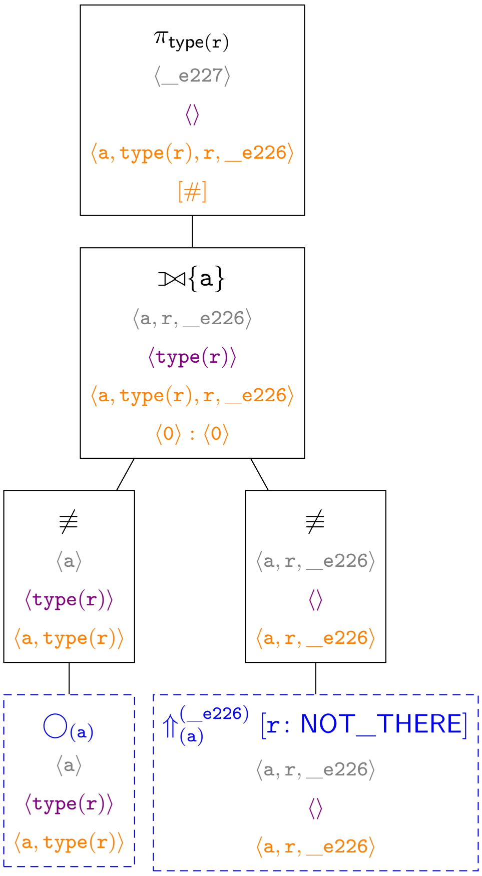
Relational algebra expression for search-based evaluation ('type()' on null relationship)

$$r = \pi_{\mathsf{type}(\mathtt{r})}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{a})}\Big)\text{⟕}\not\equiv\Big(\uparrow^{(\_\mathsf{e226})}_{(\mathtt{a})}[\mathtt{r}\colon\mathsf{NOT\_THERE}]\Big(\bigcirc_{(\mathtt{a})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation ('type()' on null relationship)



Incremental relational algebra tree ('type()' on null relationship)



A.10.14 'type()' on mixed null and non-null relationships

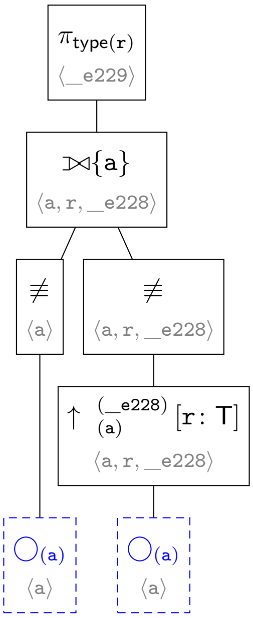
Query specification ('type()' on mixed null and non-null relationships)

```
MATCH (a)
OPTIONAL MATCH (a)-[r:T]->()
RETURN type(r)
```

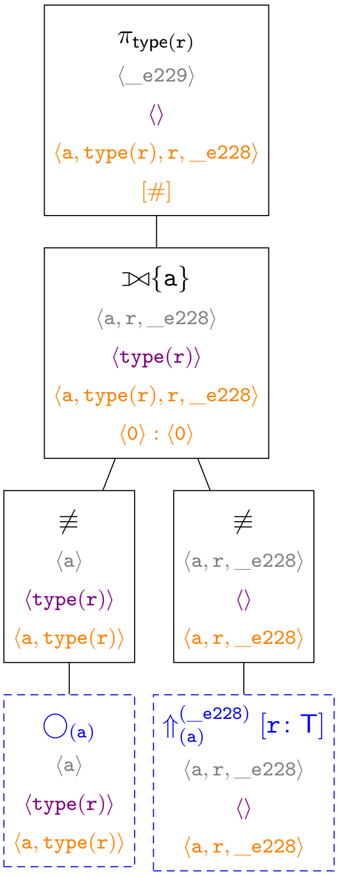
Relational algebra expression for search-based evaluation ('type()' on mixed null and non-null relationships)

$$r = \pi_{\mathtt{type(r)}}\left(\not\equiv\left(\bigcirc_{(\mathtt{a})}\right) ⟕ \not\equiv\left(\uparrow\,{}^{(\_\mathsf{e228})}_{(\mathtt{a})}\,[\mathtt{r}\colon\mathsf{T}]\left(\bigcirc_{(\mathtt{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation ('type()' on mixed null and non-null relationships)



Incremental relational algebra tree ('type()' on mixed null and non-null relationships)



A.10.15 'type()' handling Any type

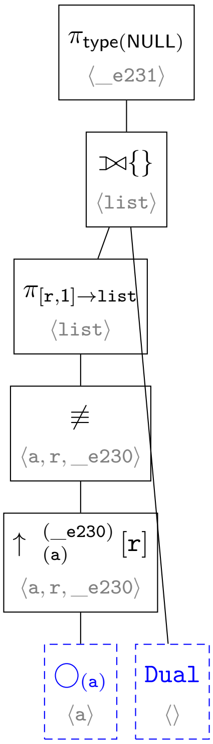
Query specification ('type()' handling Any type)

```
MATCH (a)-[r]->()
WITH [r, 1] AS list
RETURN type(list[0])
```

Relational algebra expression for search-based evaluation ('type()' handling Any type)

$$r = \pi_{\mathsf{type(NULL)}}\left(\pi_{[\mathtt{r},1]\rightarrow\mathtt{list}}\left(\not\equiv\left(\uparrow\,^{(\_\mathsf{e230})}_{(\mathtt{a})}\,[\mathtt{r}]\left(\bigcirc_{(\mathtt{a})}\right)\right)\right) ⟕ \,\mathtt{Dual}\right)$$

Relational algebra tree for search-based evaluation (’type()’ handling Any type)


$\pi_{\mathsf{type(NULL)}}$
⟨_e231⟩
⟕{}
⟨list⟩
$\pi_{[\mathtt{r},1]\rightarrow\mathtt{list}}$
⟨list⟩
≢
⟨a, r, _e230⟩
$\uparrow^{(\_e230)}_{(a)}[\mathtt{r}]$
⟨a, r, _e230⟩
$\bigcirc_{(a)}$
⟨a⟩
Dual
⟨⟩

Incremental relational algebra tree (’type()’ handling Any type)

Cannot visualize incremental tree.

A.10.16 ’labels()’ should accept type Any

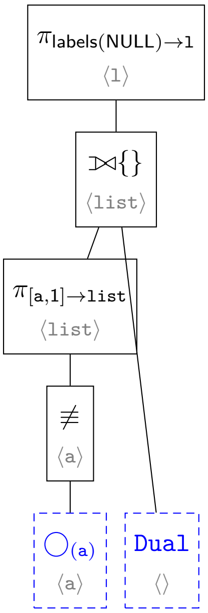
Query specification (’labels()’ should accept type Any)

```
MATCH (a)
WITH [a, 1] AS list
RETURN labels(list[0]) AS l
```

Relational algebra expression for search-based evaluation (’labels()’ should accept type Any)

$$r = \pi_{\mathsf{labels(NULL)}\rightarrow\mathtt{l}}\left(\pi_{[\mathsf{a},1]\rightarrow\mathtt{list}}\left(\not\equiv\left(\bigcirc_{(\mathsf{a})}\right)\right) ⟕ \mathtt{Dual}\right)$$

Relational algebra tree for search-based evaluation ('labels()' should accept type Any)


$\pi_{\text{labels(NULL)}\rightarrow\texttt{l}}$
⟨l⟩
⋈{}
⟨list⟩
$\pi_{[\text{a},1]\rightarrow\texttt{list}}$
⟨list⟩
≢
⟨a⟩
○(a)
⟨a⟩
Dual
⟨⟩

Incremental relational algebra tree ('labels()' should accept type Any)

Cannot visualize incremental tree.

A.10.17 'labels()' failing on invalid arguments

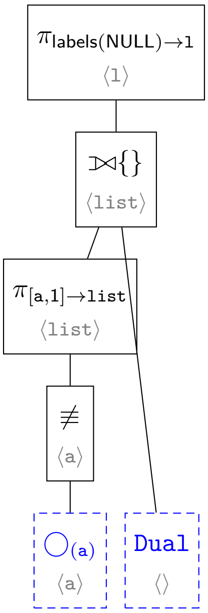
Query specification ('labels()' failing on invalid arguments)

```
MATCH (a)
WITH [a, 1] AS list
RETURN labels(list[1]) AS l
```

Relational algebra expression for search-based evaluation ('labels()' failing on invalid arguments)

$$r = \pi_{\text{labels(NULL)}\rightarrow\texttt{l}}\left(\pi_{[\text{a},1]\rightarrow\texttt{list}}\left(\not\equiv\left(\bigcirc_{(\text{a})}\right)\right)\bowtie\texttt{Dual}\right)$$

Relational algebra tree for search-based evaluation ('labels()' failing on invalid arguments)


$\pi_{\text{labels(NULL)}\rightarrow\texttt{l}}$
⟨l⟩
⋈{}
⟨list⟩
$\pi_{[\texttt{a},\texttt{1}]\rightarrow\texttt{list}}$
⟨list⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩
Dual
⟨⟩

Incremental relational algebra tree ('labels()' failing on invalid arguments)

Cannot visualize incremental tree.

A.10.18 'exists()' is case insensitive

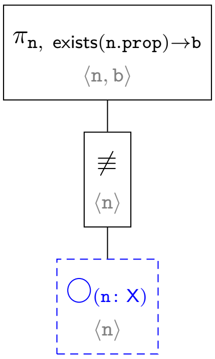
Query specification ('exists()' is case insensitive)

```
MATCH (n:X)
RETURN n, EXIsTS(n.prop) AS b
```

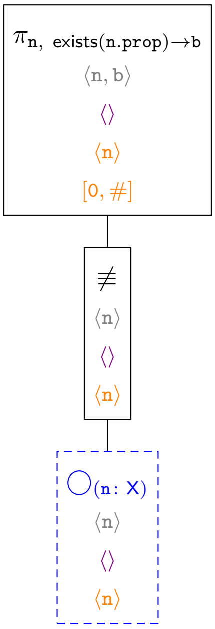
Relational algebra expression for search-based evaluation ('exists()' is case insensitive)

$$r = \pi_{\texttt{n},\ \text{exists}(\texttt{n.prop})\rightarrow\texttt{b}}\left(\not\equiv \left(\bigcirc_{(\texttt{n}:\ \textsf{X})} \right) \right)$$

Relational algebra tree for search-based evaluation ('exists()' is case insensitive)


$\pi_{\texttt{n},\ \text{exists}(\texttt{n.prop})\rightarrow\texttt{b}}$
⟨n, b⟩
$\not\equiv$
⟨n⟩
$\bigcirc_{(\texttt{n}:\ \textsf{X})}$
⟨n⟩

Incremental relational algebra tree ('exists()' is case insensitive)


$\pi_{\texttt{n, exists(n.prop)}\rightarrow\texttt{b}}$
⟨n, b⟩
⟨⟩
⟨n⟩
[0, #]
$\not\equiv$
⟨n⟩
⟨⟩
⟨n⟩
$\bigcirc_{(\texttt{n: X})}$
⟨n⟩
⟨⟩
⟨n⟩

A.11 ValueHashJoinAcceptance

Progress:

2 of 2

A.11.1 Find friends of others

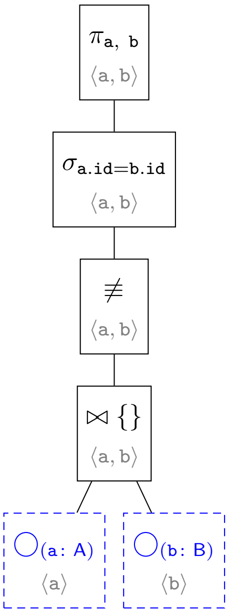
Query specification (Find friends of others)

```
MATCH (a:A), (b:B)
WHERE a.id = b.id
RETURN a, b
```

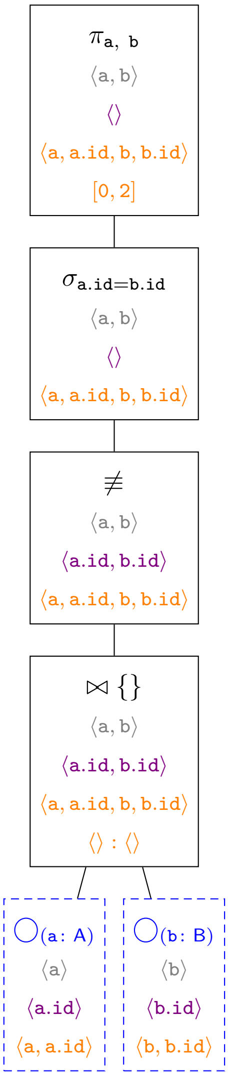
Relational algebra expression for search-based evaluation (Find friends of others)

$$r = \pi_{\texttt{a, b}}\Big(\sigma_{\texttt{a.id=b.id}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{a: A})} \bowtie \bigcirc_{(\texttt{b: B})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Find friends of others)



Incremental relational algebra tree (Find friends of others)



A.11.2 Should only join when matching

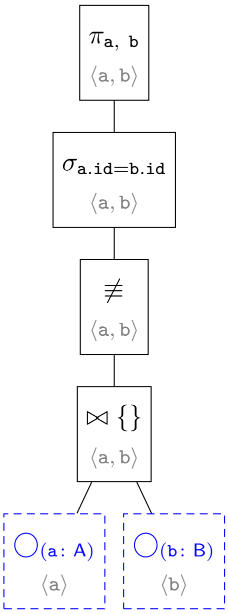
Query specification (Should only join when matching)

```
MATCH (a:A), (b:B)
WHERE a.id = b.id
RETURN a, b
```

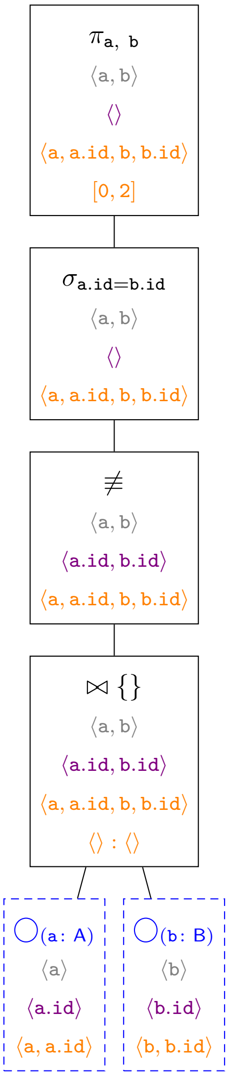
Relational algebra expression for search-based evaluation (Should only join when matching)

$$r = \pi_{\texttt{a, b}}\Big(\sigma_{\texttt{a.id=b.id}}\Big(\not\equiv\Big(\bigcirc_{\texttt{(a: A)}} \bowtie \bigcirc_{\texttt{(b: B)}}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Should only join when matching)



Incremental relational algebra tree (Should only join when matching)



A.12 KeysAcceptance

Progress:

2 of 9

A.12.1 Using 'keys()' on a single node, non-empty result

Query specification (Using 'keys()' on a single node, non-empty result)

```
MATCH (n)
UNWIND keys(n) AS x
RETURN DISTINCT x AS theProps
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.12.2 Using 'keys()' on multiple nodes, non-empty result

Query specification (Using 'keys()' on multiple nodes, non-empty result)

```
MATCH (n)
UNWIND keys(n) AS x
RETURN DISTINCT x AS theProps
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.12.3 Using 'keys()' on a single node, empty result

Query specification (Using 'keys()' on a single node, empty result)

```
MATCH (n)
UNWIND keys(n) AS x
RETURN DISTINCT x AS theProps
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.12.4 Using 'keys()' on an optionally matched node

Query specification (Using 'keys()' on an optionally matched node)

```
OPTIONAL MATCH (n)
UNWIND keys(n) AS x
RETURN DISTINCT x AS theProps
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.12.5 Using 'keys()' on a relationship, non-empty result

Query specification (Using 'keys()' on a relationship, non-empty result)

```
MATCH ()-[r:KNOWS]-()
UNWIND keys(r) AS x
RETURN DISTINCT x AS theProps
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.12.6 Using 'keys()' on a relationship, empty result

Query specification (Using 'keys()' on a relationship, empty result)

```
MATCH ()-[r:KNOWS]-()
UNWIND keys(r) AS x
RETURN DISTINCT x AS theProps
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.12.7 Using 'keys()' on an optionally matched relationship

Query specification (Using 'keys()' on an optionally matched relationship)

```
OPTIONAL MATCH ()-[r:KNOWS]-()
UNWIND keys(r) AS x
RETURN DISTINCT x AS theProps
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.12.8 Using 'keys()' on a literal map

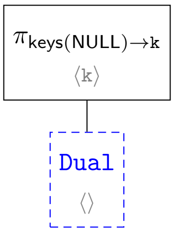
Query specification (Using 'keys()' on a literal map)

```
RETURN keys({name: 'Alice', age: 38, address: {city: 'London', residential: true}}) AS k
```

Relational algebra expression for search-based evaluation (Using 'keys()' on a literal map)

$$r = \pi_{\mathsf{keys(NULL)} \rightarrow \mathsf{k}}\Big(\mathtt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Using 'keys()' on a literal map)



Incremental relational algebra tree (Using 'keys()' on a literal map)

Cannot visualize incremental tree.

A.12.9 Using 'keys()' on a parameter map

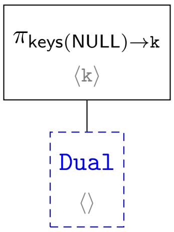
Query specification (Using 'keys()' on a parameter map)

```
RETURN keys($param) AS k
```

Relational algebra expression for search-based evaluation (Using 'keys()' on a parameter map)

$$r = \pi_{\mathsf{keys(NULL)} \rightarrow \mathsf{k}}\Big(\mathtt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Using 'keys()' on a parameter map)



Incremental relational algebra tree (Using 'keys()' on a parameter map)

Cannot visualize incremental tree.

A.13 LabelsAcceptance

Progress:

1 of 1

A.13.1 Using 'labels()' in return clauses

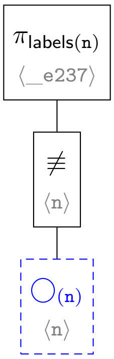
Query specification (Using 'labels()' in return clauses)

```
MATCH (n)
RETURN labels(n)
```

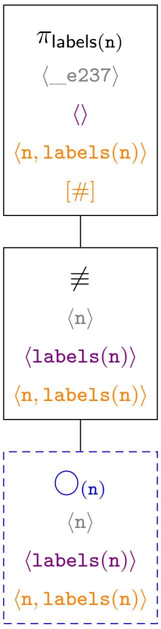
Relational algebra expression for search-based evaluation (Using 'labels()' in return clauses)

$$r = \pi_{\mathsf{labels}(\mathtt{n})}\left(\not\equiv\left(\bigcirc_{(\mathtt{n})}\right)\right)$$

Relational algebra tree for search-based evaluation (Using 'labels()' in return clauses)


$\pi_{\mathsf{labels}(\mathtt{n})}$
⟨_e237⟩
$\not\equiv$
⟨n⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩

Incremental relational algebra tree (Using ’labels()’ in return clauses)



A.14 LargeCreateQuery

Progress:

0 of 0

A.15 LargeIntegerEquality

Progress:

5 of 5

A.15.1 Does not lose precision

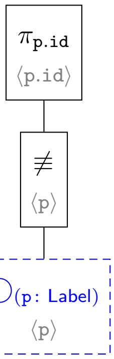
Query specification (Does not lose precision)

```
MATCH (p:Label)
RETURN p.id
```

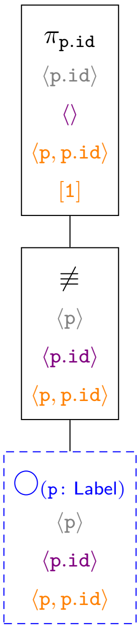
Relational algebra expression for search-based evaluation (Does not lose precision)

$$r = \pi_{\mathtt{p.id}}\Big(\not\equiv \Big(\bigcirc_{(\mathtt{p}:\ \mathsf{Label})} \Big)\Big)$$

Relational algebra tree for search-based evaluation (Does not lose precision)



Incremental relational algebra tree (Does not lose precision)



A.15.2 Handling inlined equality of large integer

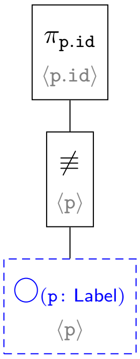
Query specification (Handling inlined equality of large integer)

```
MATCH (p:Label {id: 4611686018427387905})
RETURN p.id
```

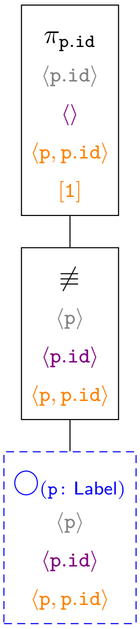
Relational algebra expression for search-based evaluation (Handling inlined equality of large integer)

$$r = \pi_{\mathtt{p.id}}\Big(\not\equiv \Big(\bigcirc_{(\mathtt{p}:\ \mathsf{Label})} \Big)\Big)$$

Relational algebra tree for search-based evaluation (Handling inlined equality of large integer)



Incremental relational algebra tree (Handling inlined equality of large integer)



A.15.3 Handling explicit equality of large integer

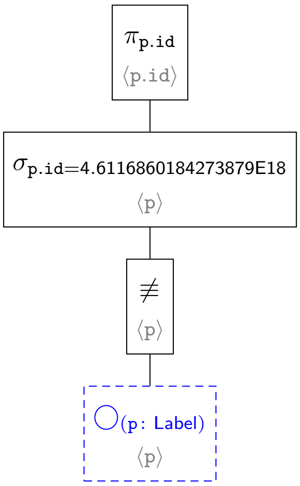
Query specification (Handling explicit equality of large integer)

```
MATCH (p:Label)
WHERE p.id = 4611686018427387905
RETURN p.id
```

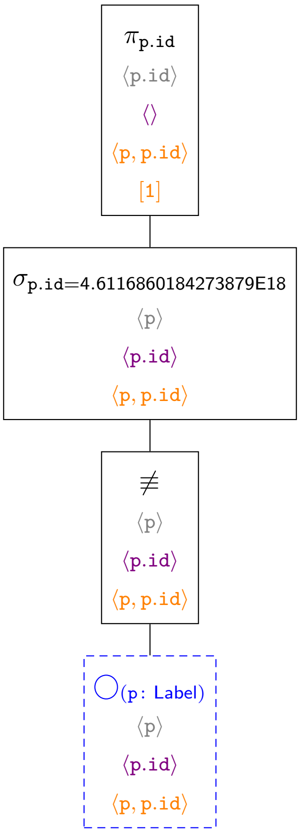
Relational algebra expression for search-based evaluation (Handling explicit equality of large integer)

$$r = \pi_{\mathtt{p.id}}\left(\sigma_{\mathtt{p.id}=\mathsf{4.6116860184273879E18}}\left(\not\equiv\left(\bigcirc_{(\mathtt{p}:\ \mathsf{Label})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Handling explicit equality of large integer)



Incremental relational algebra tree (Handling explicit equality of large integer)



A.15.4 Handling inlined equality of large integer, non-equal values

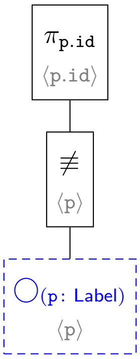
Query specification (Handling inlined equality of large integer, non-equal values)

```
MATCH (p:Label {id : 4611686018427387900})
RETURN p.id
```

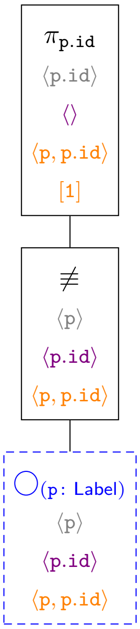
Relational algebra expression for search-based evaluation (Handling inlined equality of large integer, non-equal values)

$$r = \pi_{\texttt{p.id}}\Big(\not\equiv \Big(\bigcirc_{(\texttt{p}:\ \textsf{Label})} \Big)\Big)$$

Relational algebra tree for search-based evaluation (Handling inlined equality of large integer, non-equal values)



Incremental relational algebra tree (Handling inlined equality of large integer, non-equal values)



A.15.5 Handling explicit equality of large integer, non-equal values

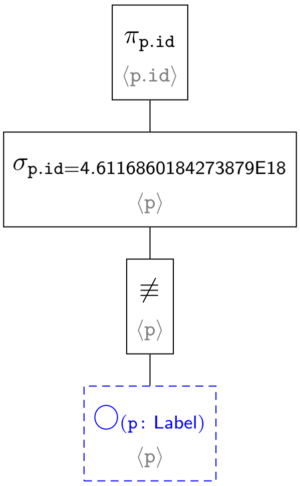
Query specification (Handling explicit equality of large integer, non-equal values)

```
MATCH (p:Label)
WHERE p.id = 4611686018427387900
RETURN p.id
```

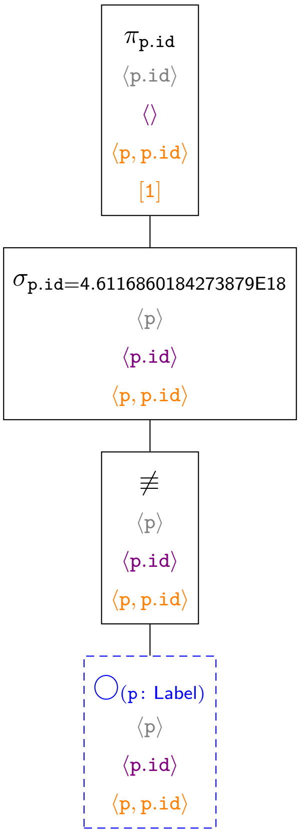
Relational algebra expression for search-based evaluation (Handling explicit equality of large integer, non-equal values)

$$r = \pi_{\texttt{p.id}}\left(\sigma_{\texttt{p.id}=\textsf{4.6116860184273879E18}}\left(\not\equiv\left(\bigcirc_{(\texttt{p}:\ \textsf{Label})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Handling explicit equality of large integer, non-equal values)


$\pi_{\texttt{p.id}}$
⟨p.id⟩
$\sigma_{\texttt{p.id}=\textsf{4.6116860184273879E18}}$
⟨p⟩
$\not\equiv$
⟨p⟩
$\bigcirc_{(\texttt{p}:\ \textsf{Label})}$
⟨p⟩

Incremental relational algebra tree (Handling explicit equality of large integer, non-equal values)



A.16 ListComprehension

Progress:

1 of 3

A.16.1 Returning a list comprehension

Query specification (Returning a list comprehension)

```
MATCH p = (n)-->()
RETURN [x IN collect(p) | head(nodes(x))] AS p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.16.2 Using a list comprehension in a WITH

Query specification (Using a list comprehension in a WITH)

```
MATCH p = (n:A)-->()
WITH [x IN collect(p) | head(nodes(x))] AS p, count(n) AS c
RETURN p, c
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.16.3 Using a list comprehension in a WHERE

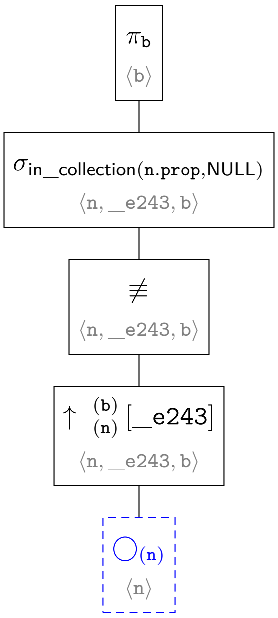
Query specification (Using a list comprehension in a WHERE)

```
MATCH (n)-->(b)
WHERE n.prop IN [x IN labels(b) | lower(x)]
RETURN b
```

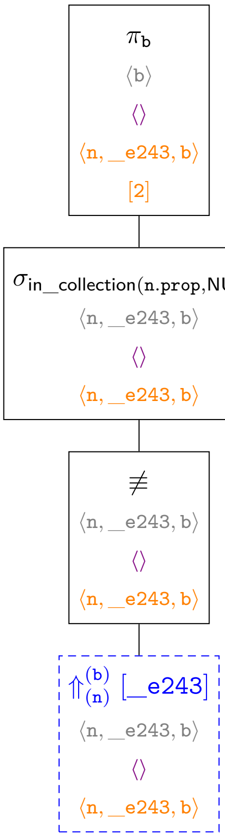
Relational algebra expression for search-based evaluation (Using a list comprehension in a WHERE)

$$r = \pi_{\mathsf{b}}\Big(\sigma_{\mathsf{in\_collection}(\mathtt{n.prop},\mathsf{NULL})}\Big(\not\equiv\Big(\uparrow\ ^{(\mathtt{b})}_{(\mathtt{n})}[\_\mathsf{e243}]\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Using a list comprehension in a WHERE)


π_{b}
⟨b⟩
$\sigma_{\mathsf{in\_collection}(\mathtt{n.prop},\mathsf{NULL})}$
⟨n, _e243, b⟩
≢
⟨n, _e243, b⟩
$\uparrow\ ^{(\mathtt{b})}_{(\mathtt{n})}[\_\mathsf{e243}]$
⟨n, _e243, b⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩

Incremental relational algebra tree (Using a list comprehension in a WHERE)



A.17 Literals

Progress:

A.17.1 Return an integer

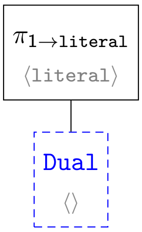
Query specification (Return an integer)

```
RETURN 1 AS literal
```

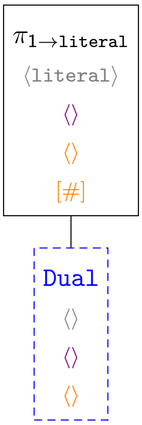
Relational algebra expression for search-based evaluation (Return an integer)

$$r = \pi_{1 \rightarrow \texttt{literal}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Return an integer)



Incremental relational algebra tree (Return an integer)



A.17.2 Return a float

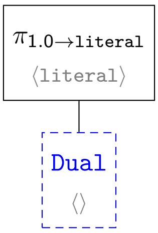
Query specification (Return a float)

```
RETURN 1.0 AS literal
```

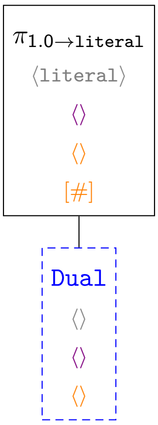
Relational algebra expression for search-based evaluation (Return a float)

$$r = \pi_{1.0 \rightarrow \texttt{literal}} \left(\texttt{Dual} \right)$$

Relational algebra tree for search-based evaluation (Return a float)



Incremental relational algebra tree (Return a float)



A.17.3 Return a float in exponent form

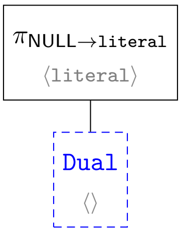
Query specification (Return a float in exponent form)

```
RETURN -1e-9 AS literal
```

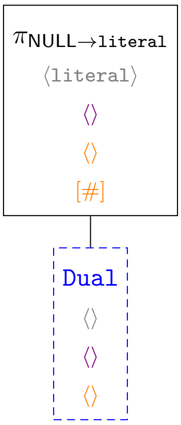
Relational algebra expression for search-based evaluation (Return a float in exponent form)

$$r = \pi_{\textsf{NULL} \rightarrow \texttt{literal}} \Big(\texttt{Dual} \Big)$$

Relational algebra tree for search-based evaluation (Return a float in exponent form)



Incremental relational algebra tree (Return a float in exponent form)



A.17.4 Return a boolean

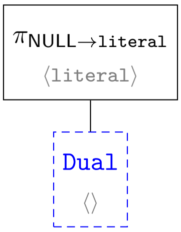
Query specification (Return a boolean)

```
RETURN true AS literal
```

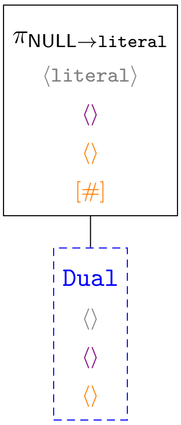
Relational algebra expression for search-based evaluation (Return a boolean)

$$r = \pi_{\textsf{NULL} \rightarrow \texttt{literal}} \Big(\texttt{Dual} \Big)$$

Relational algebra tree for search-based evaluation (Return a boolean)



Incremental relational algebra tree (Return a boolean)



A.17.5 Return a single-quoted string

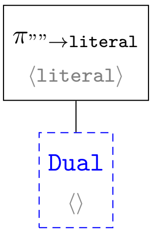
Query specification (Return a single-quoted string)

```
RETURN '' AS literal
```

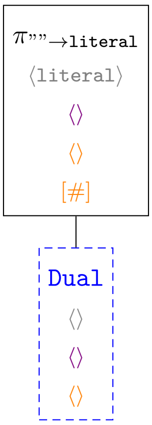
Relational algebra expression for search-based evaluation (Return a single-quoted string)

$$r = \pi_{\text{""}\rightarrow\texttt{literal}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Return a single-quoted string)



Incremental relational algebra tree (Return a single-quoted string)



A.17.6 Return a double-quoted string

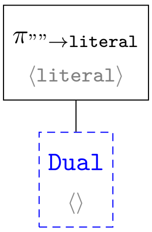
Query specification (Return a double-quoted string)

```
RETURN "" AS literal
```

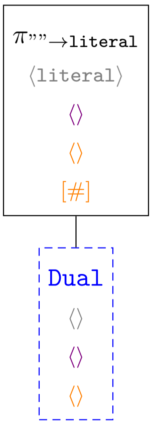
Relational algebra expression for search-based evaluation (Return a double-quoted string)

$$r = \pi_{\text{””}\rightarrow\texttt{literal}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Return a double-quoted string)



Incremental relational algebra tree (Return a double-quoted string)



A.17.7 Return null

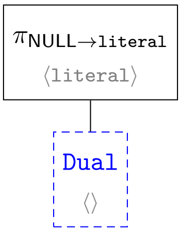
Query specification (Return null)

```
RETURN null AS literal
```

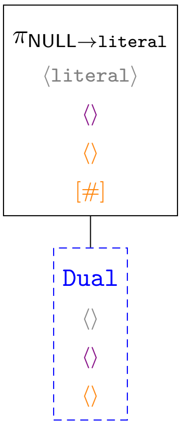
Relational algebra expression for search-based evaluation (Return null)

$$r = \pi_{\mathsf{NULL}\rightarrow\texttt{literal}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Return null)



Incremental relational algebra tree (Return null)



A.17.8 Return an empty list

Query specification (Return an empty list)

```
RETURN [] AS literal
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.17.9 Return a nonempty list

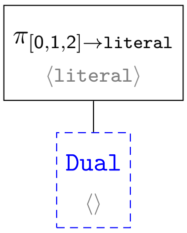
Query specification (Return a nonempty list)

```
RETURN [0, 1, 2] AS literal
```

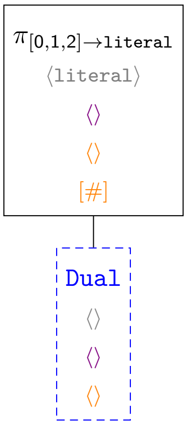
Relational algebra expression for search-based evaluation (Return a nonempty list)

$$r = \pi_{[0,1,2] \rightarrow \texttt{literal}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Return a nonempty list)



Incremental relational algebra tree (Return a nonempty list)



A.17.10 Return an empty map

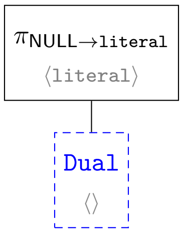
Query specification (Return an empty map)

```
RETURN {} AS literal
```

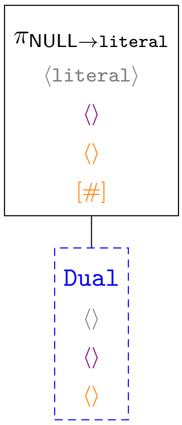
Relational algebra expression for search-based evaluation (Return an empty map)

$$r = \pi_{\mathsf{NULL} \rightarrow \texttt{literal}}\left(\texttt{Dual}\right)$$

Relational algebra tree for search-based evaluation (Return an empty map)



Incremental relational algebra tree (Return an empty map)



A.17.11 Return a nonempty map

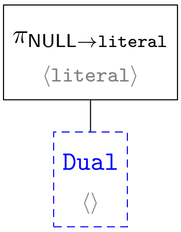
Query specification (Return a nonempty map)

```
RETURN {k1: 0, k2: 'string'} AS literal
```

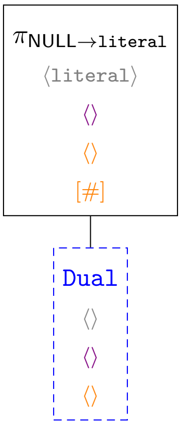
Relational algebra expression for search-based evaluation (Return a nonempty map)

$$r = \pi_{\mathsf{NULL}\rightarrow\texttt{literal}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Return a nonempty map)

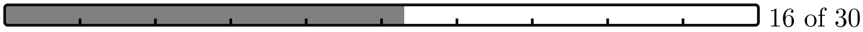


Incremental relational algebra tree (Return a nonempty map)



A.18 MatchAcceptance

Progress:



A.18.1 Path query should return results in written order

Query specification (Path query should return results in written order)

```
MATCH p = (a:Label1)<--(:Label2)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.2 Longer path query should return results in written order

Query specification (Longer path query should return results in written order)

```
MATCH p = (a:Label1)<--(:Label2)--()
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.3 Use multiple MATCH clauses to do a Cartesian product

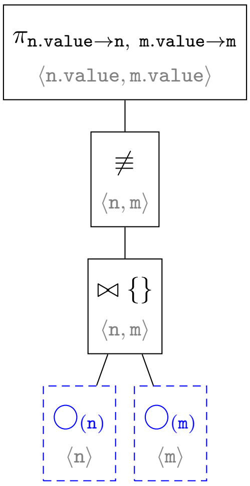
Query specification (Use multiple MATCH clauses to do a Cartesian product)

```
MATCH (n), (m)
RETURN n.value AS n, m.value AS m
```

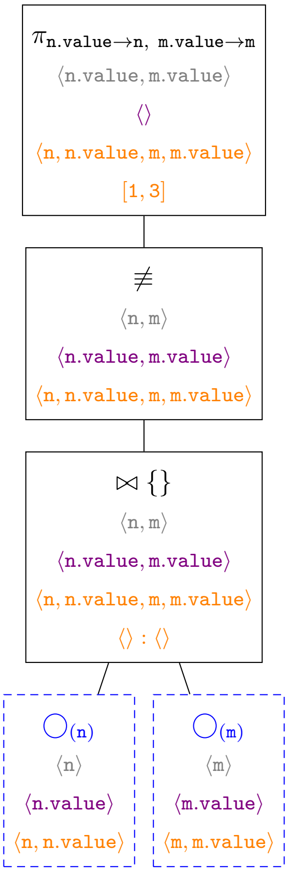
Relational algebra expression for search-based evaluation (Use multiple MATCH clauses to do a Cartesian product)

$$r = \pi_{\texttt{n.value}\to\texttt{n, m.value}\to\texttt{m}}\left(\not\equiv\left(\bigcirc_{(\texttt{n})} \bowtie \bigcirc_{(\texttt{m})}\right)\right)$$

Relational algebra tree for search-based evaluation (Use multiple MATCH clauses to do a Cartesian product)



Incremental relational algebra tree (Use multiple MATCH clauses to do a Cartesian product)



A.18.4 Use params in pattern matching predicates

Query specification (Use params in pattern matching predicates)

```
MATCH (a)-[r]->(b)
WHERE r.foo = $param
RETURN b
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.5 Filter out based on node prop name

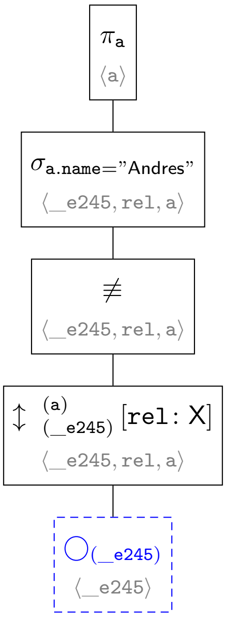
Query specification (Filter out based on node prop name)

```
MATCH ()-[rel:X]-(a)
WHERE a.name = 'Andres'
RETURN a
```

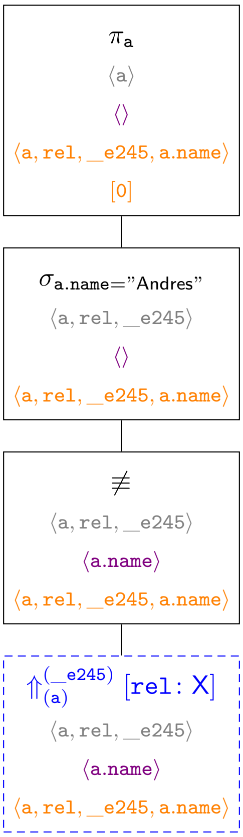
Relational algebra expression for search-based evaluation (Filter out based on node prop name)

$$r = \pi_{\mathsf{a}}\Big(\sigma_{\mathsf{a.name}=\text{"Andres"}}\Big(\not\equiv\Big(\updownarrow\,{}^{(\mathsf{a})}_{(\_\mathsf{e245})}\,[\texttt{rel}\colon \mathsf{X}]\Big(\bigcirc_{(\_\mathsf{e245})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Filter out based on node prop name)



Incremental relational algebra tree (Filter out based on node prop name)



A.18.6 Honour the column name for RETURN items

Query specification (Honour the column name for RETURN items)

```
MATCH (a)
WITH a.name AS a
RETURN a
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.7 Filter based on rel prop name

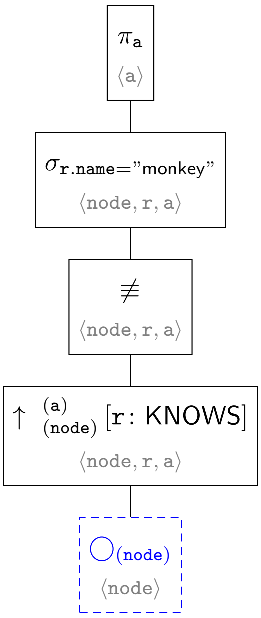
Query specification (Filter based on rel prop name)

```
MATCH (node)-[r:KNOWS]->(a)
WHERE r.name = 'monkey'
RETURN a
```

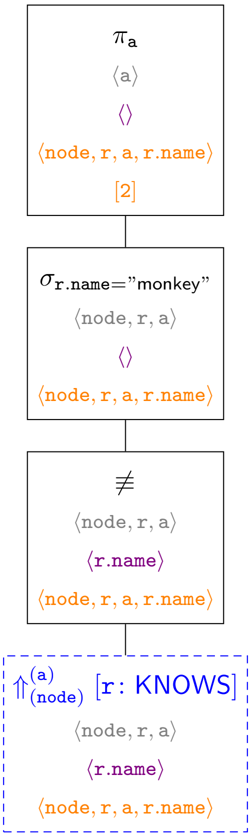
Relational algebra expression for search-based evaluation (Filter based on rel prop name)

$$r = \pi_{\mathtt{a}}\Big(\sigma_{\mathtt{r.name}=\text{"monkey"}}\Big(\not\equiv\Big(\uparrow^{(\mathtt{a})}_{(\mathtt{node})}[\mathtt{r}\colon \textsf{KNOWS}]\Big(\bigcirc_{(\mathtt{node})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Filter based on rel prop name)



Incremental relational algebra tree (Filter based on rel prop name)



A.18.8 Cope with shadowed variables

Query specification (Cope with shadowed variables)

```
MATCH (n)
WITH n.name AS n
RETURN n
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.9 Get neighbours

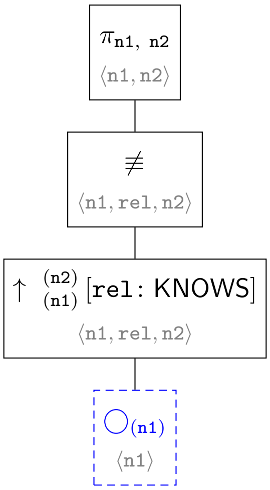
Query specification (Get neighbours)

```
MATCH (n1)-[rel:KNOWS]->(n2)
RETURN n1, n2
```

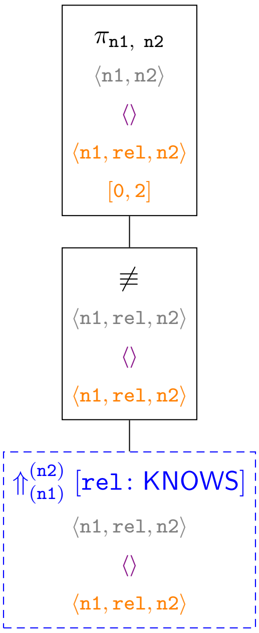
Relational algebra expression for search-based evaluation (Get neighbours)

$$r = \pi_{\texttt{n1},\ \texttt{n2}}\Big(\not\equiv\Big(\uparrow\,^{(\texttt{n2})}_{(\texttt{n1})}\,[\texttt{rel}\colon \mathsf{KNOWS}]\,\Big(\bigcirc_{(\texttt{n1})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Get neighbours)



Incremental relational algebra tree (Get neighbours)



A.18.10 Get two related nodes

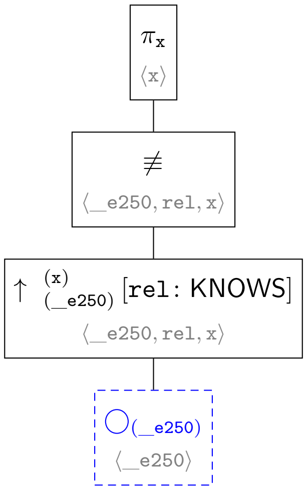
Query specification (Get two related nodes)

```
MATCH ()-[rel:KNOWS]->(x)
RETURN x
```

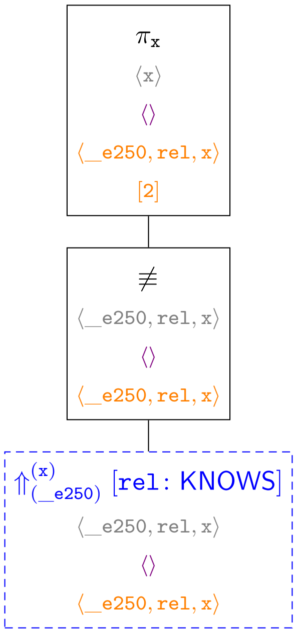
Relational algebra expression for search-based evaluation (Get two related nodes)

$$r = \pi_{\mathtt{x}}\Big(\not\equiv\Big(\uparrow^{(\mathtt{x})}_{(\_\mathsf{e250})}[\mathtt{rel}\colon \mathsf{KNOWS}]\Big(\bigcirc_{(\_\mathsf{e250})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Get two related nodes)



Incremental relational algebra tree (Get two related nodes)



A.18.11 Get related to related to

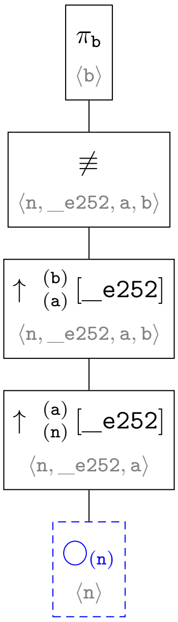
Query specification (Get related to related to)

```
MATCH (n)-->(a)-->(b)
RETURN b
```

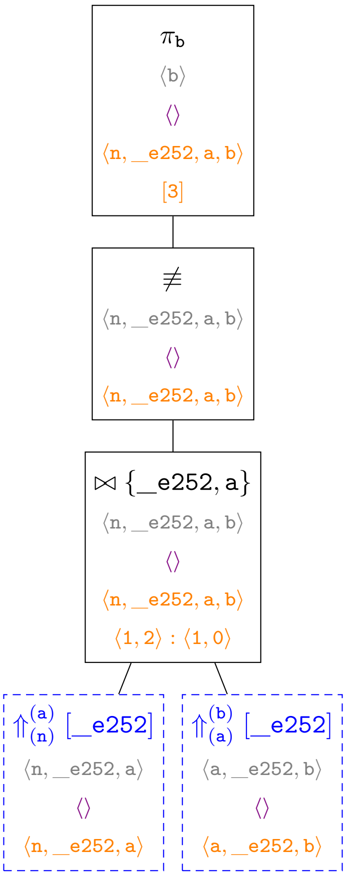
Relational algebra expression for search-based evaluation (Get related to related to)

$$r = \pi_{\mathtt{b}}\Big(\not\equiv \Big(\uparrow\, {}^{(\mathtt{b})}_{(\mathtt{a})} [\_\mathsf{e252}] \Big(\uparrow\, {}^{(\mathtt{a})}_{(\mathtt{n})} [\_\mathsf{e252}] \Big(\bigcirc_{(\mathtt{n})} \Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Get related to related to)



Incremental relational algebra tree (Get related to related to)



A.18.12 Handle comparison between node properties

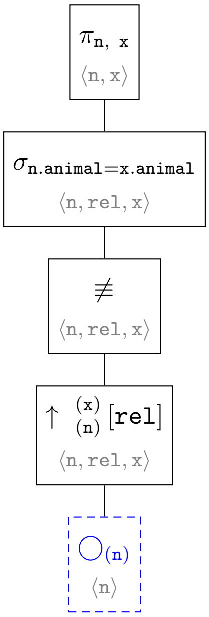
Query specification (Handle comparison between node properties)

```
MATCH (n)-[rel]->(x)
WHERE n.animal = x.animal
RETURN n, x
```

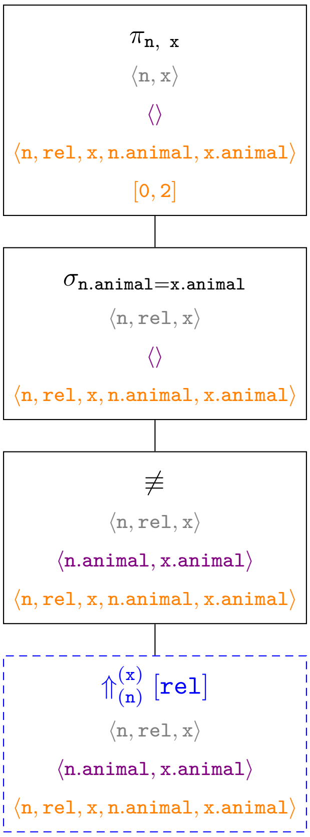
Relational algebra expression for search-based evaluation **(Handle comparison between node properties)**

$$r = \pi_{\texttt{n, x}}\Big(\sigma_{\texttt{n.animal}=\texttt{x.animal}}\Big(\not\equiv\Big(\uparrow\,{}^{(\texttt{x})}_{(\texttt{n})}\,[\texttt{rel}]\Big(\bigcirc_{(\texttt{n})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation **(Handle comparison between node properties)**


$\pi_{\texttt{n, x}}$
$\langle \texttt{n}, \texttt{x} \rangle$
$\sigma_{\texttt{n.animal}=\texttt{x.animal}}$
$\langle \texttt{n}, \texttt{rel}, \texttt{x} \rangle$
$\not\equiv$
$\langle \texttt{n}, \texttt{rel}, \texttt{x} \rangle$
$\uparrow\,{}^{(\texttt{x})}_{(\texttt{n})}\,[\texttt{rel}]$
$\langle \texttt{n}, \texttt{rel}, \texttt{x} \rangle$
$\bigcirc_{(\texttt{n})}$
$\langle \texttt{n} \rangle$

Incremental relational algebra tree (Handle comparison between node properties)


$\pi_{\texttt{n, x}}$
⟨n, x⟩
⟨⟩
⟨n, rel, x, n.animal, x.animal⟩
[0, 2]
$\sigma_{\texttt{n.animal=x.animal}}$
⟨n, rel, x⟩
⟨⟩
⟨n, rel, x, n.animal, x.animal⟩
$\not\equiv$
⟨n, rel, x⟩
⟨n.animal, x.animal⟩
⟨n, rel, x, n.animal, x.animal⟩
$\Uparrow^{(\texttt{x})}_{(\texttt{n})}[\texttt{rel}]$
⟨n, rel, x⟩
⟨n.animal, x.animal⟩
⟨n, rel, x, n.animal, x.animal⟩

A.18.13 Return two subgraphs with bound undirected relationship

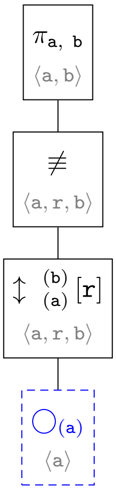
Query specification (Return two subgraphs with bound undirected relationship)

```
MATCH (a)-[r {name: 'r'}]-(b)
RETURN a, b
```

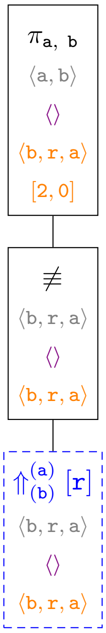
Relational algebra expression for search-based evaluation (Return two subgraphs with bound undirected relationship)

$$r = \pi_{\texttt{a, b}}\left(\not\equiv\left(\updownarrow\,{}^{(\texttt{b})}_{(\texttt{a})}\,[\texttt{r}]\left(\bigcirc_{(\texttt{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Return two subgraphs with bound undirected relationship)



Incremental relational algebra tree (Return two subgraphs with bound undirected relationship)



A.18.14 Return two subgraphs with bound undirected relationship and optional relationship

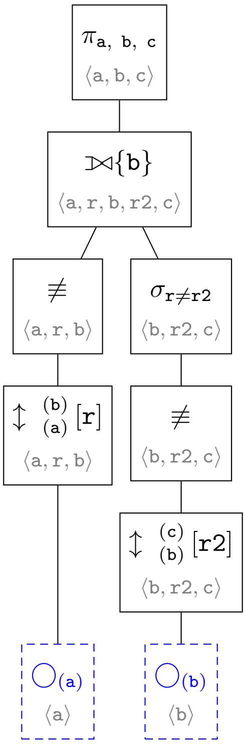
Query specification (Return two subgraphs with bound undirected relationship and optional relationship)

```
MATCH (a)-[r {name: 'r1'}]-(b)
OPTIONAL MATCH (b)-[r2]-(c)
WHERE r <> r2
RETURN a, b, c
```

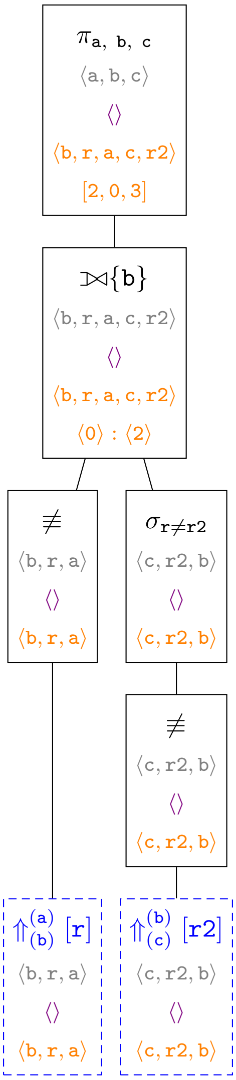
Relational algebra expression for search-based evaluation (Return two subgraphs with bound undirected relationship and optional relationship)

$$r = \pi_{\texttt{a, b, c}}\Big(\not\equiv\Big(\updownarrow\,{}^{(\texttt{b})}_{(\texttt{a})}\,[\texttt{r}]\Big(\bigcirc_{(\texttt{a})}\Big)\Big) ⟕ \sigma_{\texttt{r}\neq\texttt{r2}}\Big(\not\equiv\Big(\updownarrow\,{}^{(\texttt{c})}_{(\texttt{b})}\,[\texttt{r2}]\Big(\bigcirc_{(\texttt{b})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Return two subgraphs with bound undirected relationship and optional relationship)



Incremental relational algebra tree (Return two subgraphs with bound undirected relationship and optional relationship)



A.18.15 Rel type function works as expected

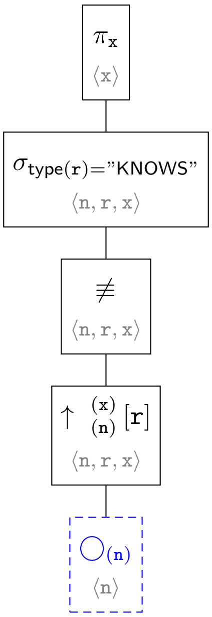
Query specification (Rel type function works as expected)

```
MATCH (n {name: 'A'})-[r]->(x)
WHERE type(r) = 'KNOWS'
RETURN x
```

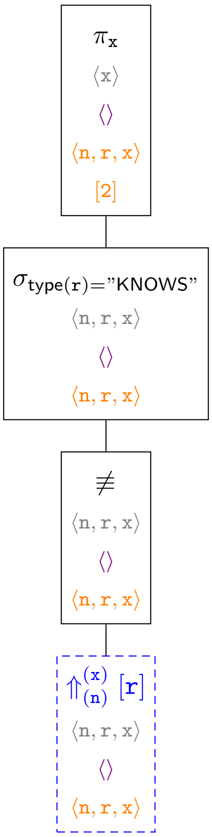
Relational algebra expression for search-based evaluation (Rel type function works as expected)

$$r = \pi_{\mathtt{x}}\left(\sigma_{\mathsf{type(r)}=\text{"KNOWS"}}\left(\not\equiv\left(\uparrow^{(\mathtt{x})}_{(\mathtt{n})}[\mathtt{r}]\left(\bigcirc_{(\mathtt{n})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Rel type function works as expected)



Incremental relational algebra tree (Rel type function works as expected)



A.18.16 Walk alternative relationships

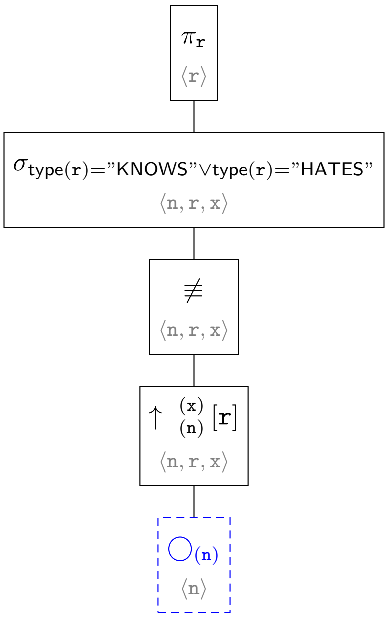
Query specification (Walk alternative relationships)

```
MATCH (n)-[r]->(x)
WHERE type(r) = 'KNOWS' OR type(r) = 'HATES'
RETURN r
```

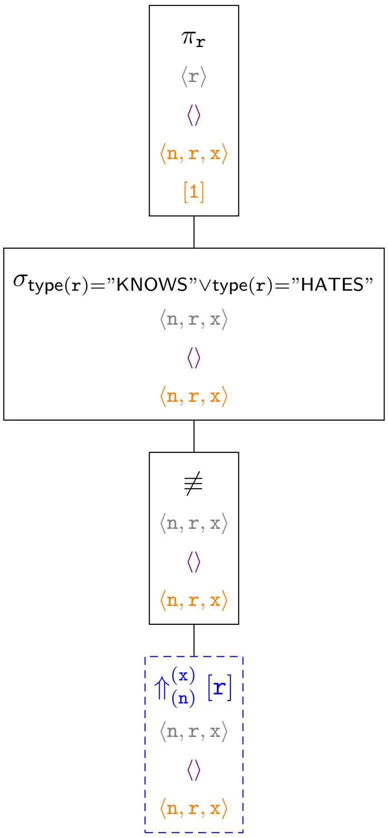
Relational algebra expression for search-based evaluation (Walk alternative relationships)

$$r = \pi_{\mathtt{r}}\left(\sigma_{\mathsf{type}(\mathtt{r})=\text{"KNOWS"}\vee\mathsf{type}(\mathtt{r})=\text{"HATES"}}\left(\not\equiv\left(\uparrow\,{}^{(\mathtt{x})}_{(\mathtt{n})}\,[\mathtt{r}]\left(\bigcirc_{(\mathtt{n})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Walk alternative relationships)


$\pi_{\mathtt{r}}$
⟨r⟩
$\sigma_{\mathsf{type}(\mathtt{r})=\text{"KNOWS"}\vee\mathsf{type}(\mathtt{r})=\text{"HATES"}}$
⟨n, r, x⟩
≢
⟨n, r, x⟩
$\uparrow\,{}^{(\mathtt{x})}_{(\mathtt{n})}\,[\mathtt{r}]$
⟨n, r, x⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩

Incremental relational algebra tree (Walk alternative relationships)


$\pi_{\mathtt{r}}$
⟨r⟩
⟨⟩
⟨n, r, x⟩
[1]
$\sigma_{\mathsf{type}(\mathtt{r})=\text{"KNOWS"}\vee\mathsf{type}(\mathtt{r})=\text{"HATES"}}$
⟨n, r, x⟩
⟨⟩
⟨n, r, x⟩
≢
⟨n, r, x⟩
⟨⟩
⟨n, r, x⟩
$\Uparrow^{(\mathtt{x})}_{(\mathtt{n})}\,[\mathtt{r}]$
⟨n, r, x⟩
⟨⟩
⟨n, r, x⟩

A.18.17 Handle OR in the WHERE clause

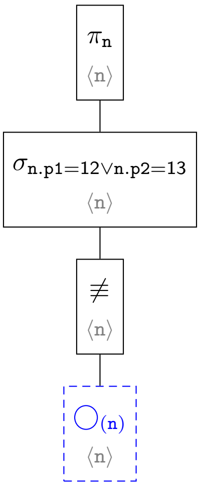
Query specification **(Handle OR in the WHERE clause)**

```
MATCH (n)
WHERE n.p1 = 12 OR n.p2 = 13
RETURN n
```

Relational algebra expression for search-based evaluation **(Handle OR in the WHERE clause)**

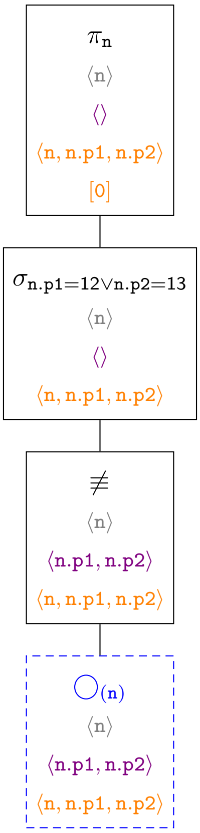
$$r = \pi_{\mathtt{n}}\Big(\sigma_{\mathtt{n.p1}=12 \lor \mathtt{n.p2}=13}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation **(Handle OR in the WHERE clause)**



$\pi_{\mathtt{n}}$
⟨n⟩
$\sigma_{\mathtt{n.p1}=12 \lor \mathtt{n.p2}=13}$
⟨n⟩
$\not\equiv$
⟨n⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩

Incremental relational algebra tree (Handle OR in the WHERE clause)



A.18.18 Return a simple path

Query specification (Return a simple path)

```
MATCH p = (a {name: 'A'})-->(b)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.19 Return a three node path

Query specification (Return a three node path)

```
MATCH p = (a {name: 'A'})-[rel1]->(b)-[rel2]->(c)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.20 Do not return anything because path length does not match

Query specification (Do not return anything because path length does not match)

```
MATCH p = (n)-->(x)
WHERE length(p) = 10
RETURN x
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.21 Pass the path length test

Query specification (Pass the path length test)

```
MATCH p = (n)-->(x)
WHERE length(p) = 1
RETURN x
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.22 Return relationships by fetching them from the path - starting from the end

Query specification (Return relationships by fetching them from the path - starting from the end)

```
MATCH p = (a)-[:REL*2..2]->(b:End)
RETURN relationships(p)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.23 Return relationships by fetching them from the path

Query specification (Return relationships by fetching them from the path)

```
MATCH p = (a:Start)-[:REL*2..2]->(b)
RETURN relationships(p)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.24 Return relationships by collecting them as a list - wrong way

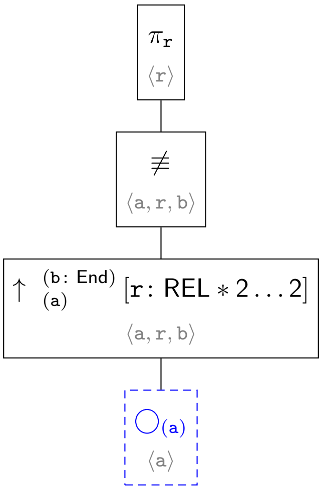
Query specification (Return relationships by collecting them as a list - wrong way)

```
MATCH (a)-[r:REL*2..2]->(b:End)
RETURN r
```

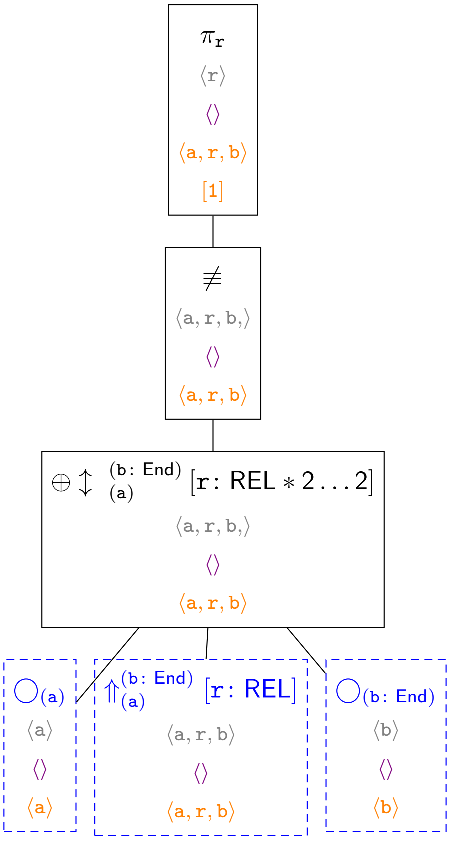
Relational algebra expression for search-based evaluation (Return relationships by collecting them as a list - wrong way)

$$r = \pi_{\mathtt{r}}\Big(\not\equiv \Big(\uparrow\,^{(\mathtt{b}\colon \mathsf{End})}_{(\mathtt{a})} [\mathtt{r}\colon \mathsf{REL} * 2 \ldots 2] \Big(\bigcirc_{(\mathtt{a})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Return relationships by collecting them as a list - wrong way)



Incremental relational algebra tree (Return relationships by collecting them as a list - wrong way)



A.18.25 Return relationships by collecting them as a list - undirected

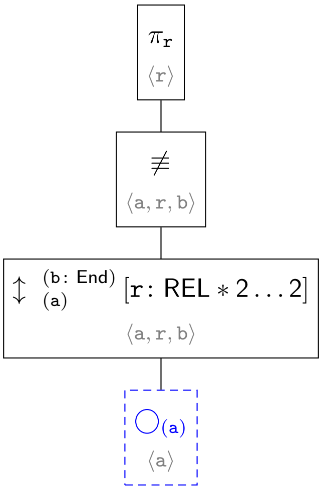
Query specification (Return relationships by collecting them as a list - undirected)

```
MATCH (a)-[r:REL*2..2]-(b:End)
RETURN r
```

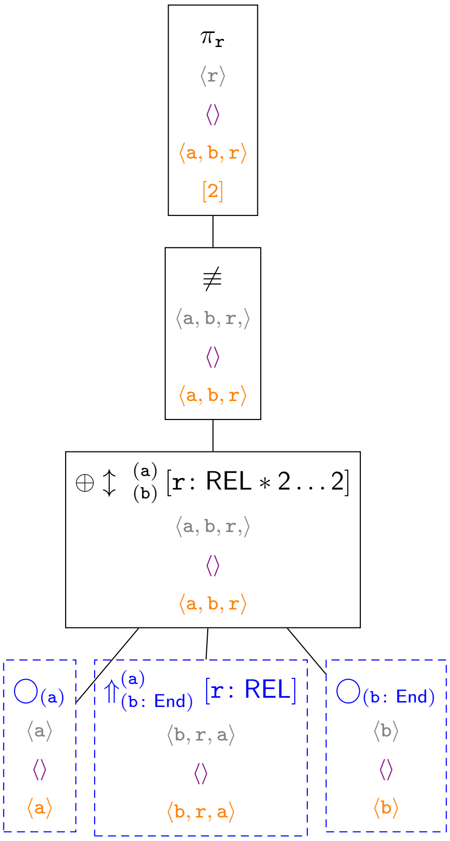
Relational algebra expression for search-based evaluation (Return relationships by collecting them as a list - undirected)

$$r = \pi_{\mathtt{r}}\left(\not\equiv\left(\updownarrow^{(\mathtt{b}:\,\mathsf{End})}_{(\mathtt{a})}[\mathtt{r}\colon \mathsf{REL}*2\ldots2]\left(\bigcirc_{(\mathtt{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Return relationships by collecting them as a list - undirected)


$\pi_{\mathtt{r}}$
⟨r⟩
≢
⟨a, r, b⟩
$\updownarrow^{(\mathtt{b}:\,\mathsf{End})}_{(\mathtt{a})}[\mathtt{r}\colon \mathsf{REL}*2\ldots2]$
⟨a, r, b⟩
$\bigcirc_{(\mathtt{a})}$
⟨a⟩

Incremental relational algebra tree (Return relationships by collecting them as a list - undirected)


$\pi_{\mathtt{r}}$
⟨r⟩
⟨⟩
⟨a, b, r⟩
[2]
≢
⟨a, b, r,⟩
⟨⟩
⟨a, b, r⟩
$\oplus\updownarrow^{(\mathtt{a})}_{(\mathtt{b})}[\mathtt{r}\colon \mathsf{REL}*2\ldots2]$
⟨a, b, r,⟩
⟨⟩
⟨a, b, r⟩
$\bigcirc_{(\mathtt{a})}$
⟨a⟩
⟨⟩
⟨a⟩
$\Uparrow^{(\mathtt{a})}_{(\mathtt{b}:\,\mathsf{End})}[\mathtt{r}\colon \mathsf{REL}]$
⟨b, r, a⟩
⟨⟩
⟨b, r, a⟩
$\bigcirc_{(\mathtt{b}:\,\mathsf{End})}$
⟨b⟩
⟨⟩
⟨b⟩

A.18.26 Return relationships by collecting them as a list

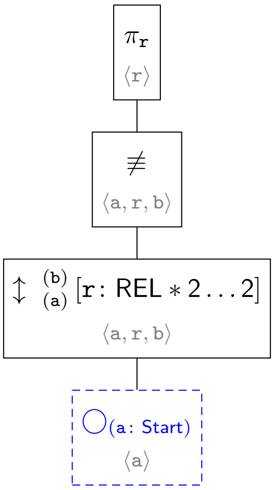
Query specification (Return relationships by collecting them as a list)

```
MATCH (a:Start)-[r:REL*2..2]-(b)
RETURN r
```

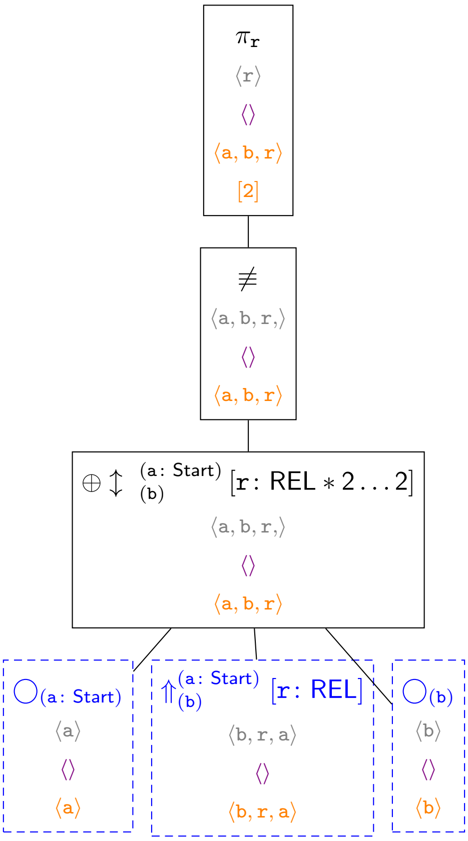
Relational algebra expression for search-based evaluation (Return relationships by collecting them as a list)

$$r = \pi_{\mathtt{r}}\left(\not\equiv\left(\updownarrow^{(\mathtt{b})}_{(\mathtt{a})}[\mathtt{r}\colon \mathsf{REL}*2\ldots2]\left(\bigcirc_{(\mathtt{a}\colon \mathsf{Start})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Return relationships by collecting them as a list)



Incremental relational algebra tree (Return relationships by collecting them as a list)



A.18.27 Return a var length path

Query specification (Return a var length path)

```
MATCH p = (n {name: 'A'})-[:KNOWS*1..2]->(x)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.28 Return a var length path of length zero

Query specification (Return a var length path of length zero)

```
MATCH p = (a)-[*0..1]->(b)
RETURN a, b, length(p) AS l
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.29 Return a named var length path of length zero

Query specification (Return a named var length path of length zero)

```
MATCH p = (a {name: 'A'})-[:KNOWS*0..1]->(b)-[:FRIEND*0..1]->(c)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.18.30 Accept skip zero

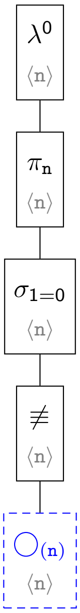
Query specification (Accept skip zero)

```
MATCH (n)
WHERE 1 = 0
RETURN n SKIP 0
```

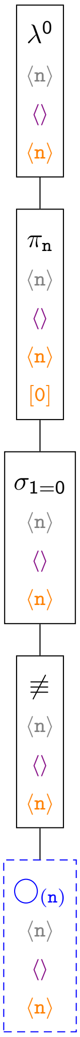
Relational algebra expression for search-based evaluation (Accept skip zero)

$$r = \lambda^{0}\Big(\pi_{\mathtt{n}}\Big(\sigma_{1=0}\Big(\not\equiv \Big(\bigcirc_{(\mathtt{n})} \Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Accept skip zero)



Incremental relational algebra tree (Accept skip zero)



A.19 MatchAcceptance2

Progress:

68 of 96

A.19.1 Do not return non-existent nodes

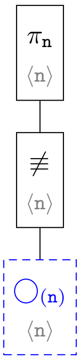
Query specification (Do not return non-existent nodes)

```
MATCH (n)
RETURN n
```

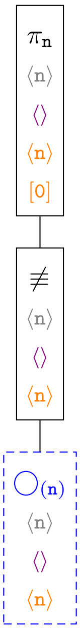
Relational algebra expression for search-based evaluation (Do not return non-existent nodes)

$$r = \pi_{\mathtt{n}}\Big(\not\equiv \Big(\bigcirc_{(\mathtt{n})} \Big)\Big)$$

Relational algebra tree for search-based evaluation (Do not return non-existent nodes)



Incremental relational algebra tree (Do not return non-existent nodes)



A.19.2 Do not return non-existent relationships

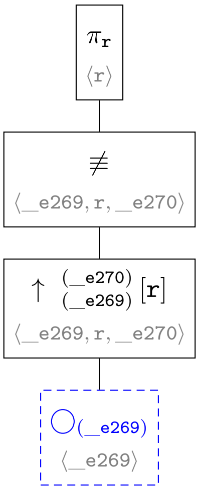
Query specification (Do not return non-existent relationships)

```
MATCH ()-[r]->()
RETURN r
```

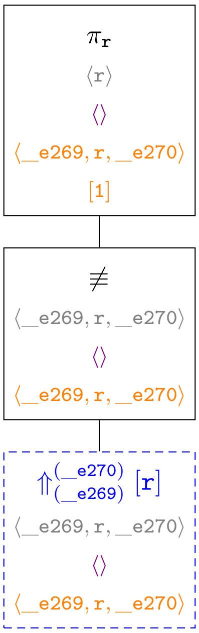
Relational algebra expression for search-based evaluation (Do not return non-existent relationships)

$$r = \pi_{\mathtt{r}}\Big(\not\equiv \Big(\uparrow^{(\_e270)}_{(\_e269)} [\mathtt{r}] \Big(\bigcirc_{(\_e269)} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Do not return non-existent relationships)



Incremental relational algebra tree (Do not return non-existent relationships)



A.19.3 Do not fail when evaluating predicates with illegal operations if the AND'ed predicate evaluates to false

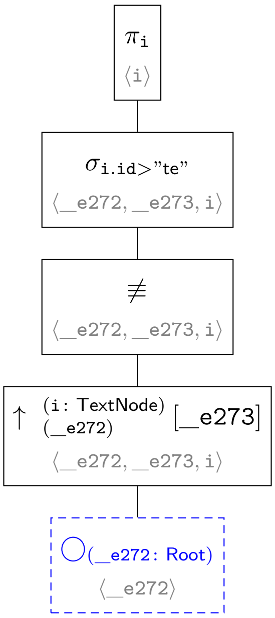
Query specification (Do not fail when evaluating predicates with illegal operations if the AND'ed predicate evaluates to false)

```
MATCH (:Root {name: 'x'})-->(i:TextNode)
WHERE i.id > 'te'
RETURN i
```

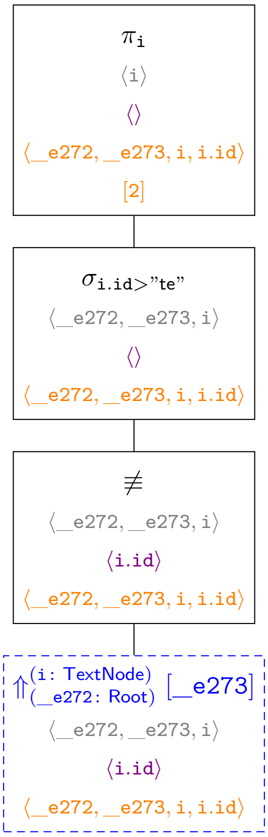
Relational algebra expression for search-based evaluation (Do not fail when evaluating predicates with illegal operations if the AND'ed predicate evaluates to false)

$$r = \pi_{\mathtt{i}}\left(\sigma_{\mathtt{i.id}>\text{"te"}}\left(\not\equiv\left(\uparrow^{(\mathtt{i}:\,\mathsf{TextNode})}_{(\_\mathsf{e272})}[\_\mathsf{e273}]\left(\bigcirc_{(\_\mathsf{e272}:\,\mathsf{Root})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Do not fail when evaluating predicates with illegal operations if the AND'ed predicate evaluates to false)


$\pi_{\mathtt{i}}$
⟨i⟩
$\sigma_{\mathtt{i.id}>\text{"te"}}$
⟨_e272, _e273, i⟩
$\not\equiv$
⟨_e272, _e273, i⟩
$\uparrow^{(\mathtt{i}:\,\mathsf{TextNode})}_{(\_\mathsf{e272})}[\_\mathsf{e273}]$
⟨_e272, _e273, i⟩
$\bigcirc_{(\_\mathsf{e272}:\,\mathsf{Root})}$
⟨_e272⟩

Incremental relational algebra tree (Do not fail when evaluating predicates with illegal operations if the AND'ed predicate evaluates to false)



A.19.4 Do not fail when evaluating predicates with illegal operations if the OR'd predicate evaluates to true

Query specification (Do not fail when evaluating predicates with illegal operations if the OR'd predicate evaluates to true)

```
MATCH (:Root {name: 'x'})-->(i)
WHERE exists(i.id) OR i.id > 'te'
RETURN i
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.5 Aggregation with named paths

Query specification (Aggregation with named paths)

```
MATCH p = ()-[*]->()
WITH count(*) AS count, p AS p
WITH nodes(p) AS nodes
RETURN *
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.6 Zero-length variable length pattern in the middle of the pattern

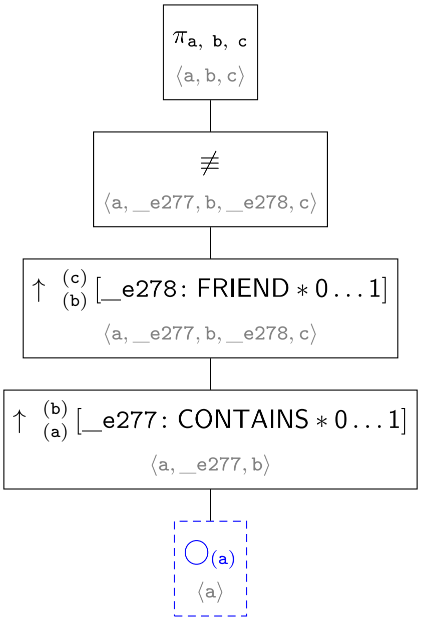
Query specification (Zero-length variable length pattern in the middle of the pattern)

```
MATCH (a {name: 'A'})-[:CONTAINS*0..1]->(b)-[:FRIEND*0..1]->(c)
RETURN a, b, c
```

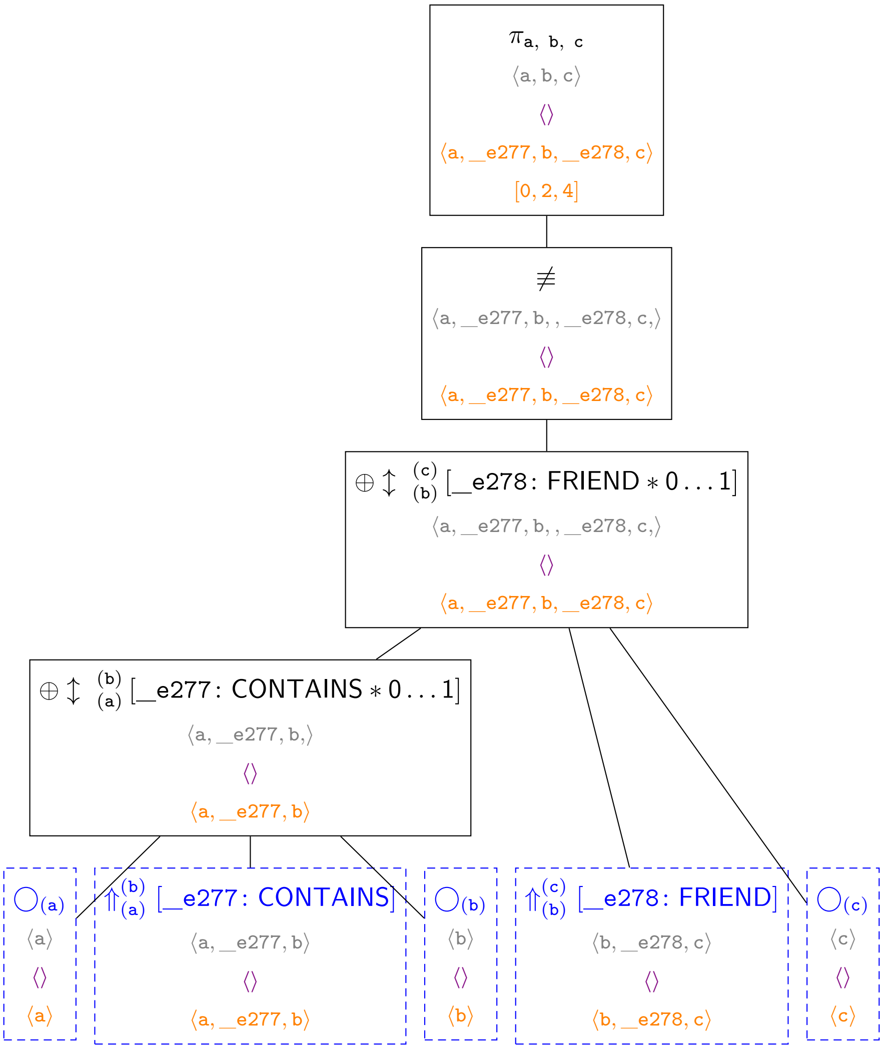
Relational algebra expression for search-based evaluation (Zero-length variable length pattern in the middle of the pattern)

$$r = \pi_{\texttt{a, b, c}}\Big(\not\equiv\Big(\uparrow\, {}^{(\texttt{c})}_{(\texttt{b})}\, [\_\mathsf{e278}\colon \mathsf{FRIEND} * 0\ldots 1]\Big(\uparrow\, {}^{(\texttt{b})}_{(\texttt{a})}\, [\_\mathsf{e277}\colon \mathsf{CONTAINS} * 0\ldots 1]\Big(\bigcirc_{(\texttt{a})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Zero-length variable length pattern in the middle of the pattern)



Incremental relational algebra tree (Zero-length variable length pattern in the middle of the pattern)



A.19.7 Simple variable length pattern

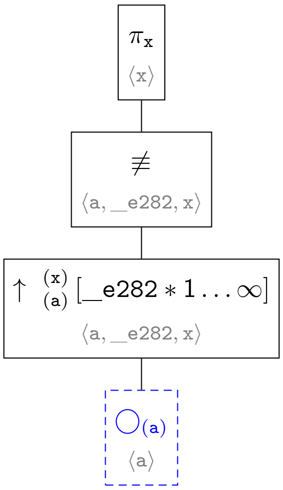
Query specification (Simple variable length pattern)

```
MATCH (a {name: 'A'})-[*]->(x)
RETURN x
```

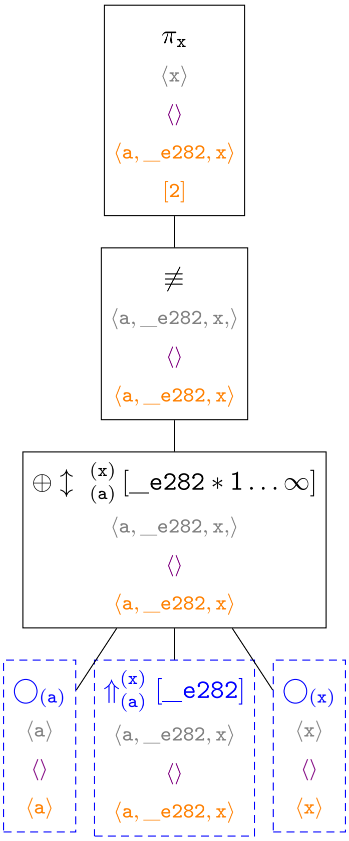
Relational algebra expression for search-based evaluation (Simple variable length pattern)

$$r = \pi_{\mathtt{x}}\Big(\not\equiv \Big(\uparrow \, {}^{(\mathtt{x})}_{(\mathtt{a})} [\_e282 * 1 \ldots \infty] \Big(\bigcirc_{(\mathtt{a})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Simple variable length pattern)



Incremental relational algebra tree (Simple variable length pattern)



A.19.8 Variable length relationship without lower bound

Query specification (Variable length relationship without lower bound)

```
MATCH p = ({name: 'A'})-[:KNOWS*..2]->()
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.9 Variable length relationship without bounds

Query specification (Variable length relationship without bounds)

```
MATCH p = ({name: 'A'})-[:KNOWS*..]->()
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.10 Returning bound nodes that are not part of the pattern

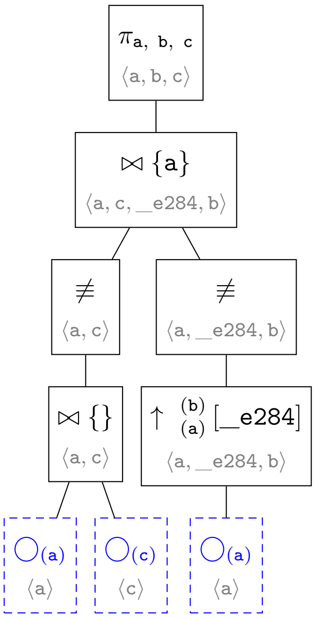
Query specification (Returning bound nodes that are not part of the pattern)

```
MATCH (a {name: 'A'}), (c {name: 'C'})
MATCH (a)-->(b)
RETURN a, b, c
```

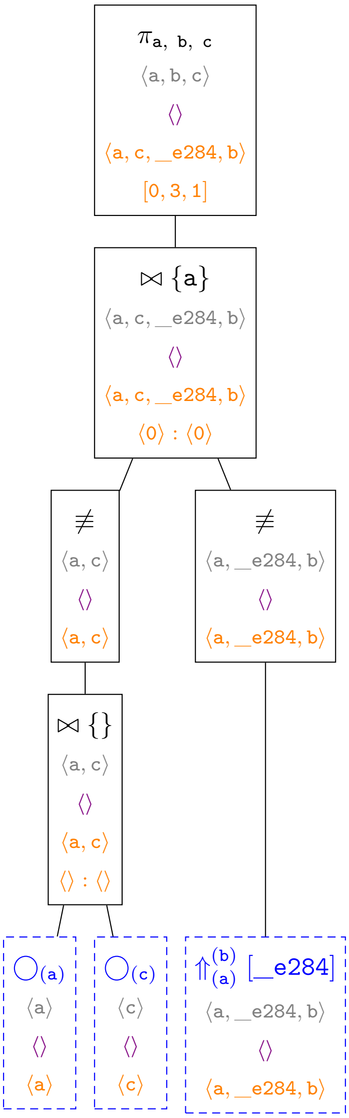
Relational algebra expression for search-based evaluation (Returning bound nodes that are not part of the pattern)

$$r = \pi_{\texttt{a, b, c}}\left(\not\equiv\left(\bigcirc_{(\texttt{a})} \bowtie \bigcirc_{(\texttt{c})}\right) \bowtie\not\equiv \left(\uparrow {}^{(\texttt{b})}_{(\texttt{a})} [\_\mathsf{e284}] \left(\bigcirc_{(\texttt{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Returning bound nodes that are not part of the pattern)



Incremental relational algebra tree (Returning bound nodes that are not part of the pattern)



A.19.11 Two bound nodes pointing to the same node

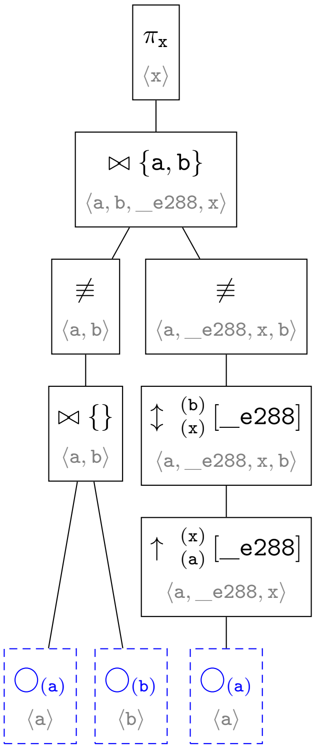
Query specification (Two bound nodes pointing to the same node)

```
MATCH (a {name: 'A'}), (b {name: 'B'})
MATCH (a)-->(x)<-->(b)
RETURN x
```

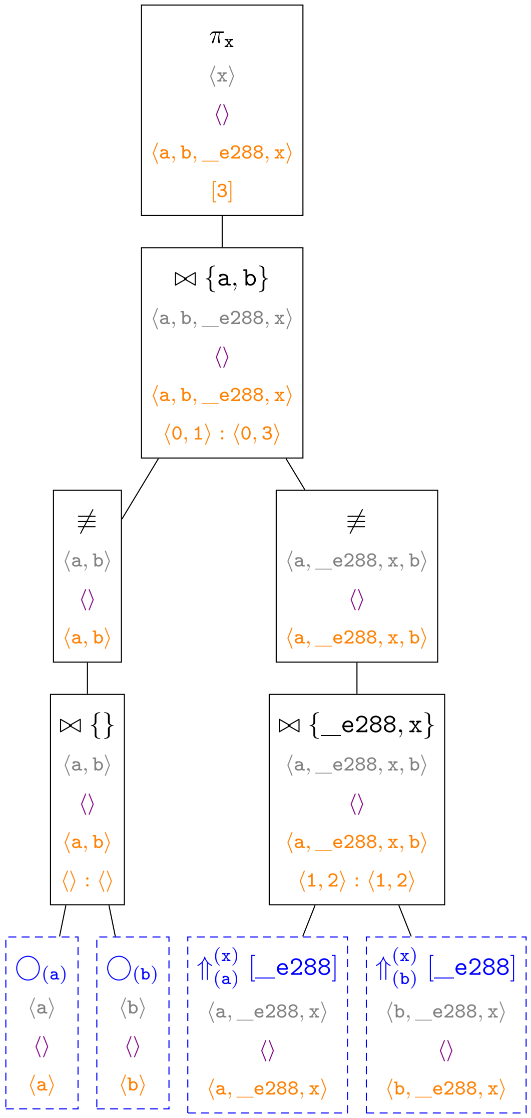
Relational algebra expression for search-based evaluation (Two bound nodes pointing to the same node)

$$r = \pi_{\mathtt{x}}\Big(\not\equiv \big(\bigcirc_{(\mathtt{a})} \bowtie \bigcirc_{(\mathtt{b})}\big) \bowtie_{\not\equiv} \big(\updownarrow {}^{(\mathtt{b})}_{(\mathtt{x})} [\_e288] \big(\uparrow {}^{(\mathtt{x})}_{(\mathtt{a})} [\_e288] \big(\bigcirc_{(\mathtt{a})} \big)\big)\big)\Big)$$

Relational algebra tree for search-based evaluation **(Two bound nodes pointing to the same node)**



Incremental relational algebra tree (Two bound nodes pointing to the same node)



A.19.12 Three bound nodes pointing to the same node

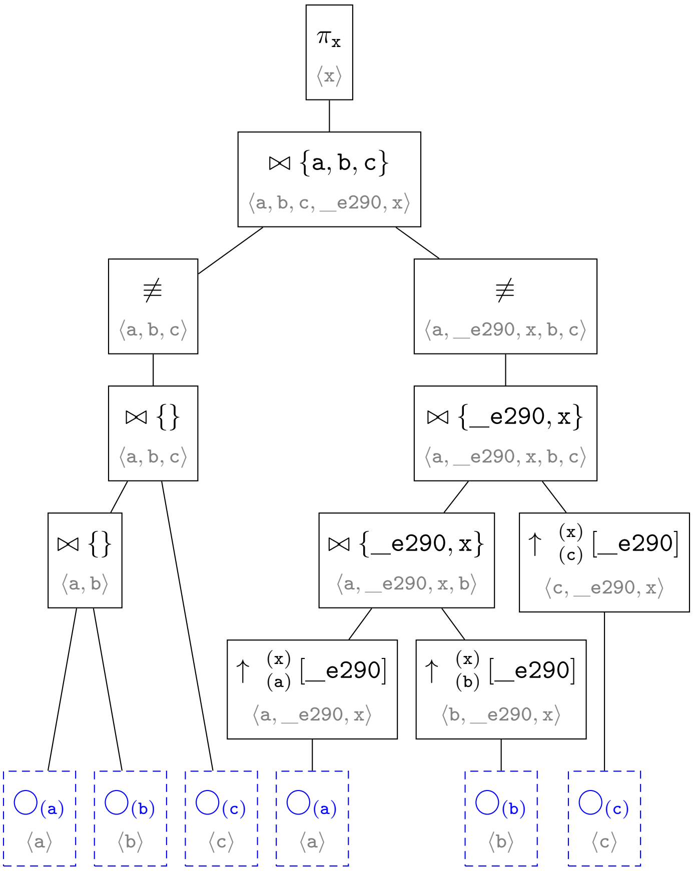
Query specification (Three bound nodes pointing to the same node)

```
MATCH (a {name: 'A'}), (b {name: 'B'}), (c {name: 'C'})
MATCH (a)-->(x), (b)-->(x), (c)-->(x)
RETURN x
```

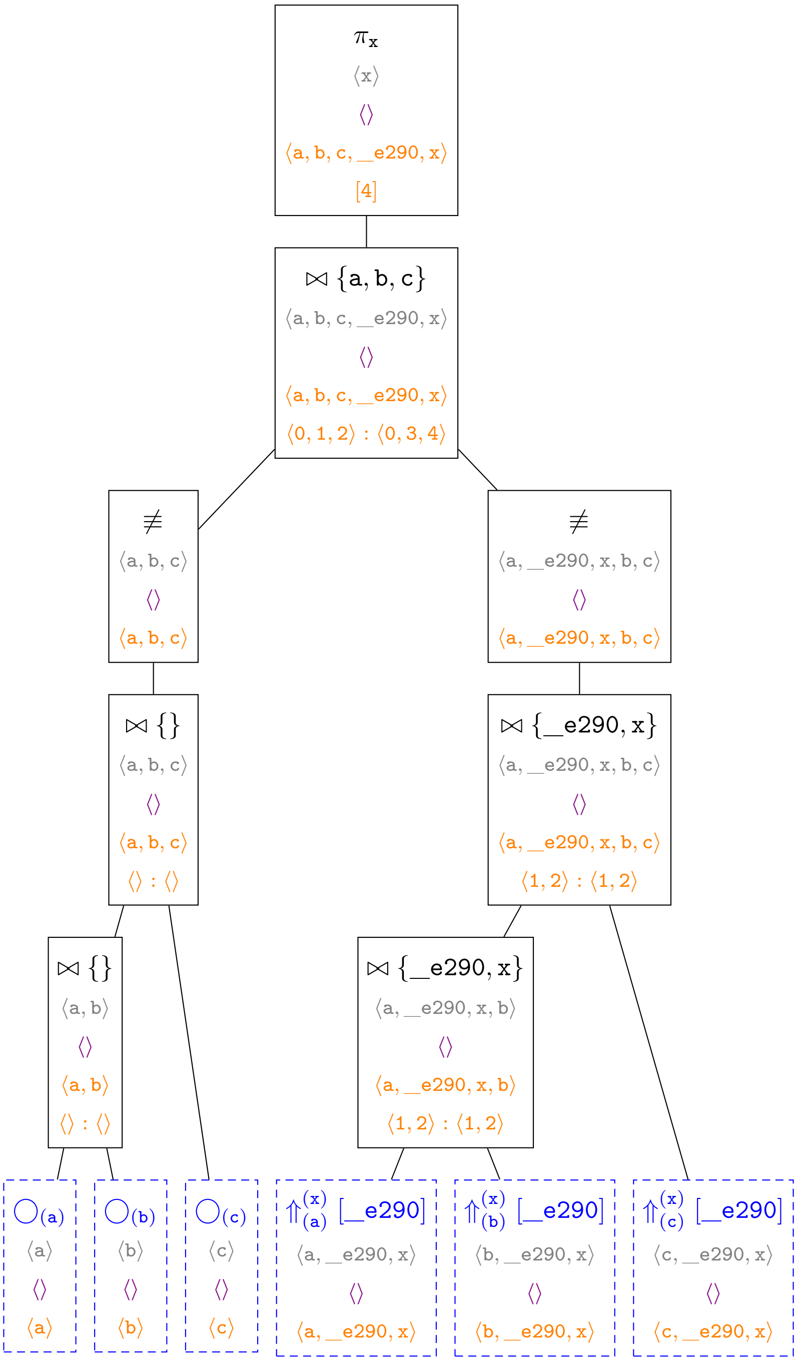
Relational algebra expression for search-based evaluation (Three bound nodes pointing to the same node)

$$r = \pi_{x}\Big(\not\equiv \Big(\bigcirc_{(a)} \bowtie \bigcirc_{(b)} \bowtie \bigcirc_{(c)} \Big) \bowtie_{\not\equiv} \Big(\uparrow {}^{(x)}_{(a)} [\_e290] \Big(\bigcirc_{(a)} \Big) \bowtie_{\uparrow} {}^{(x)}_{(b)} [\_e290] \Big(\bigcirc_{(b)} \Big) \bowtie_{\uparrow} {}^{(x)}_{(c)} [\_e290] \Big(\bigcirc_{(c)} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Three bound nodes pointing to the same node)



Incremental relational algebra tree (Three bound nodes pointing to the same node)



A.19.13 Three bound nodes pointing to the same node with extra connections

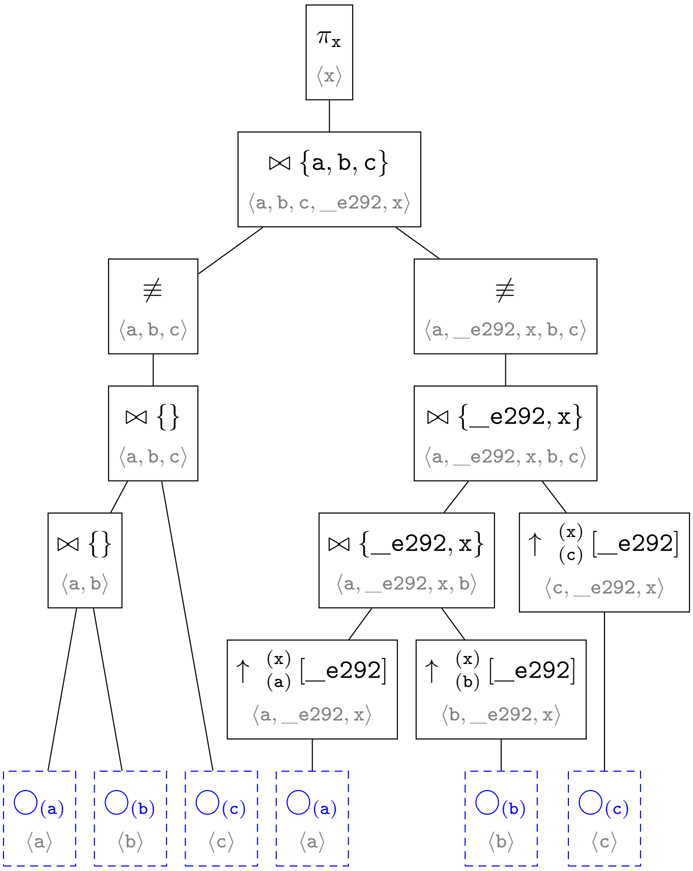
Query specification (Three bound nodes pointing to the same node with extra connections)

```
MATCH (a {name: 'a'}), (b {name: 'b'}), (c {name: 'c'})
MATCH (a)-->(x), (b)-->(x), (c)-->(x)
RETURN x
```

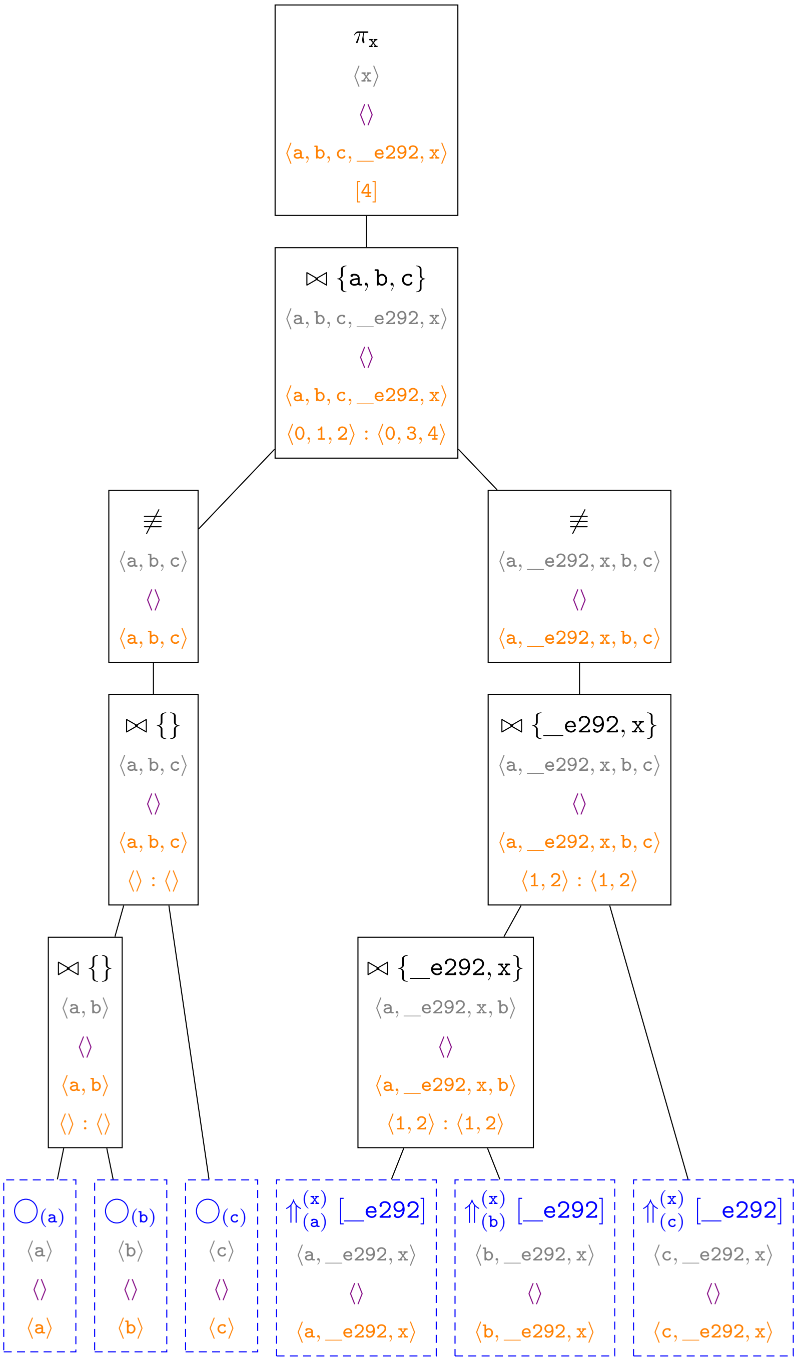
Relational algebra expression for search-based evaluation (Three bound nodes pointing to the same node with extra connections)

$$r = \pi_{\mathtt{x}}\Bigg(\not\equiv \Big(\bigcirc_{(\mathtt{a})} \bowtie \bigcirc_{(\mathtt{b})} \bowtie \bigcirc_{(\mathtt{c})} \Big) \bowtie\!\not\equiv \Big(\uparrow \, {}^{(\mathtt{x})}_{(\mathtt{a})} \, [\_\mathsf{e292}] \Big(\bigcirc_{(\mathtt{a})} \Big) \bowtie\!\uparrow \, {}^{(\mathtt{x})}_{(\mathtt{b})} \, [\_\mathsf{e292}] \Big(\bigcirc_{(\mathtt{b})} \Big) \bowtie\!\uparrow \, {}^{(\mathtt{x})}_{(\mathtt{c})} \, [\_\mathsf{e292}] \Big(\bigcirc_{(\mathtt{c})} \Big)\Big)\Bigg)$$

Relational algebra tree for search-based evaluation (Three bound nodes pointing to the same node with extra connections)



Incremental relational algebra tree (Three bound nodes pointing to the same node with extra connections)



A.19.14 MATCH with OPTIONAL MATCH in longer pattern

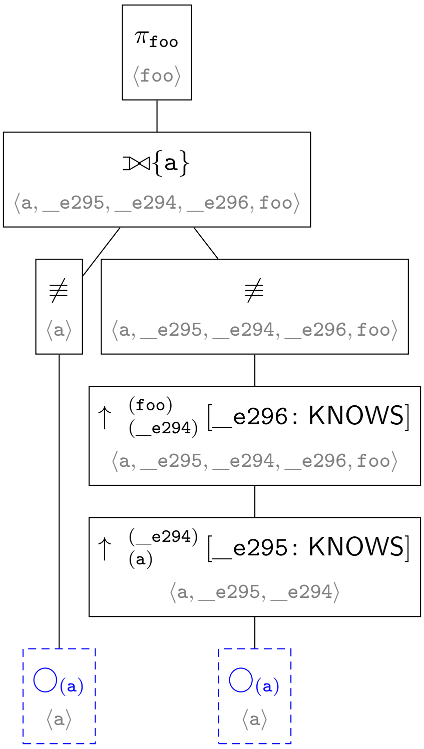
Query specification (MATCH with OPTIONAL MATCH in longer pattern)

```
MATCH (a {name: 'A'})
OPTIONAL MATCH (a)-[:KNOWS]->()-[:KNOWS]->(foo)
RETURN foo
```

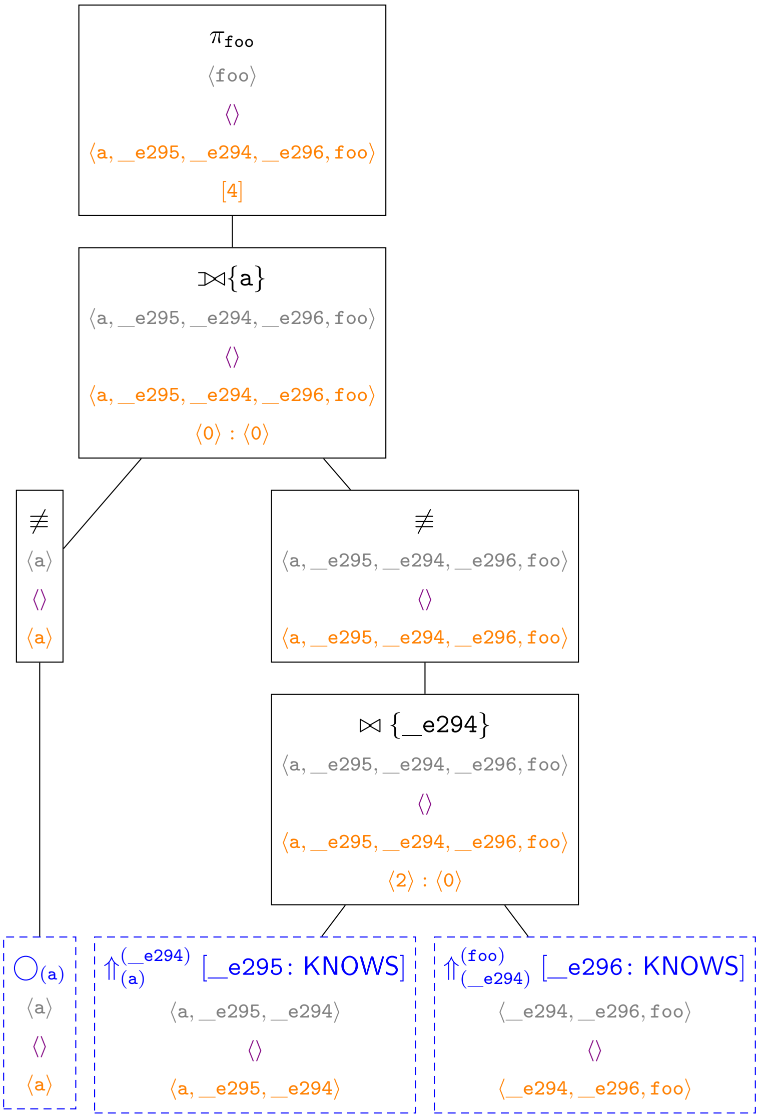
Relational algebra expression for search-based evaluation (MATCH with OPTIONAL MATCH in longer pattern)

$$r = \pi_{\texttt{foo}}\Big(\not\equiv \Big(\bigcirc_{(\texttt{a})} \Big) \mathbin{⟕} \not\equiv \Big(\uparrow\,{}^{(\texttt{foo})}_{(\_\text{e294})} [\_\mathsf{e296}\colon \mathsf{KNOWS}] \Big(\uparrow\,{}^{(\_\text{e294})}_{(\texttt{a})} [\_\mathsf{e295}\colon \mathsf{KNOWS}] \Big(\bigcirc_{(\texttt{a})} \Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (MATCH with OPTIONAL MATCH in longer pattern)


π foo
⟨foo⟩
⟕{a}
⟨a, _e295, _e294, _e296, foo⟩
≢
⟨a⟩
≢
⟨a, _e295, _e294, _e296, foo⟩
↑ (foo) (_e294) [_e296: KNOWS]
⟨a, _e295, _e294, _e296, foo⟩
↑ (_e294) (a) [_e295: KNOWS]
⟨a, _e295, _e294⟩
○(a)
⟨a⟩
○(a)
⟨a⟩

Incremental relational algebra tree (MATCH with OPTIONAL MATCH in longer pattern)


$\pi_{\texttt{foo}}$
⟨foo⟩
⟨⟩
⟨a, _e295, _e294, _e296, foo⟩
[4]
⟕{a}
⟨a, _e295, _e294, _e296, foo⟩
⟨⟩
⟨a, _e295, _e294, _e296, foo⟩
⟨0⟩ : ⟨0⟩
≢
⟨a⟩
⟨⟩
⟨a⟩
≢
⟨a, _e295, _e294, _e296, foo⟩
⟨⟩
⟨a, _e295, _e294, _e296, foo⟩
⋈ {_e294}
⟨a, _e295, _e294, _e296, foo⟩
⟨⟩
⟨a, _e295, _e294, _e296, foo⟩
⟨2⟩ : ⟨0⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩
⟨⟩
⟨a⟩
$\Uparrow^{(\_e294)}_{(a)}$ [_e295 : KNOWS]
⟨a, _e295, _e294⟩
⟨⟩
⟨a, _e295, _e294⟩
$\Uparrow^{(\texttt{foo})}_{(\_e294)}$ [_e296 : KNOWS]
⟨_e294, _e296, foo⟩
⟨⟩
⟨_e294, _e296, foo⟩

A.19.15 Optionally matching named paths

Query specification (Optionally matching named paths)

```
MATCH (a {name: 'A'}), (x)
WHERE x.name IN ['B', 'C']
OPTIONAL MATCH p = (a)-->(x)
RETURN x, p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.16 Optionally matching named paths with single and variable length patterns

Query specification (Optionally matching named paths with single and variable length patterns)

```
MATCH (a {name: 'A'})
OPTIONAL MATCH p = (a)-->(b)-[*]->(c)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.17 Optionally matching named paths with variable length patterns

Query specification **(Optionally matching named paths with variable length patterns)**

```
MATCH (a {name: 'A'}), (x)
WHERE x.name IN ['B', 'C']
OPTIONAL MATCH p = (a)-[r*]->(x)
RETURN r, x, p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.18 Matching variable length patterns from a bound node

Query specification **(Matching variable length patterns from a bound node)**

```
MATCH (a:A)
MATCH (a)-[r*2]->()
RETURN r
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.19 Excluding connected nodes

Query specification **(Excluding connected nodes)**

```
MATCH (a:A), (other:B)
OPTIONAL MATCH (a)-[r]->(other)
WITH other WHERE r IS NULL
RETURN other
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.20 Do not fail when predicates on optionally matched and missed nodes are invalid

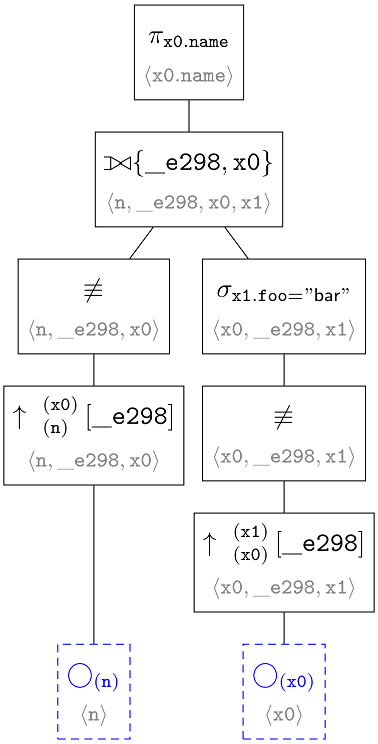
Query specification **(Do not fail when predicates on optionally matched and missed nodes are invalid)**

```
MATCH (n)-->(x0)
OPTIONAL MATCH (x0)-->(x1)
WHERE x1.foo = 'bar'
RETURN x0.name
```

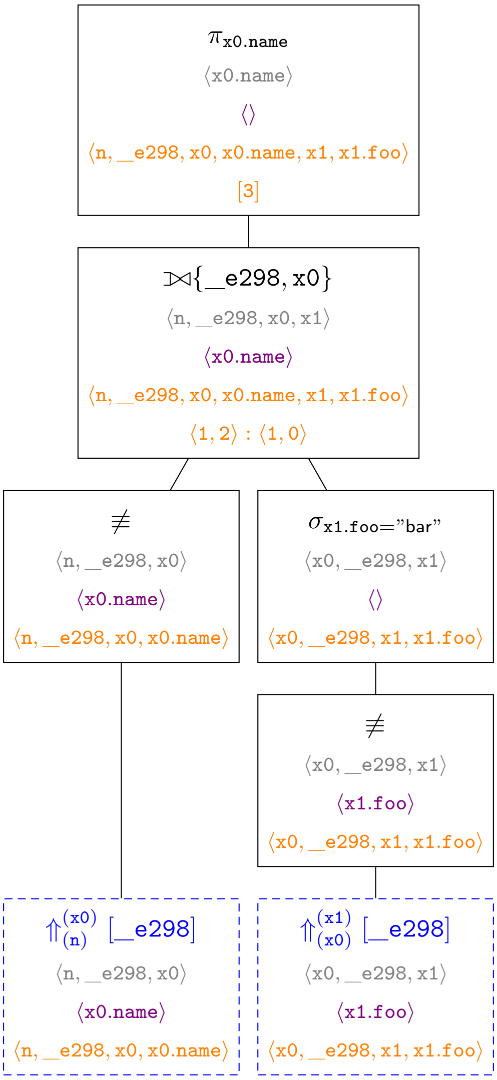
Relational algebra expression for search-based evaluation (Do not fail when predicates on optionally matched and missed nodes are invalid)

$$r = \pi_{\texttt{x0.name}}\left(\not\equiv \left(\uparrow \,^{(\texttt{x0})}_{(\texttt{n})} [\_\mathsf{e298}] \left(\bigcirc_{(\texttt{n})} \right) \right) \mathbin{⟕} \sigma_{\texttt{x1.foo}=\text{"bar"}} \left(\not\equiv \left(\uparrow \,^{(\texttt{x1})}_{(\texttt{x0})} [\_\mathsf{e298}] \left(\bigcirc_{(\texttt{x0})} \right) \right) \right) \right)$$

Relational algebra tree for search-based evaluation (Do not fail when predicates on optionally matched and missed nodes are invalid)



Incremental relational algebra tree (Do not fail when predicates on optionally matched and missed nodes are invalid)



A.19.21 MATCH and OPTIONAL MATCH on same pattern

Query specification (MATCH and OPTIONAL MATCH on same pattern)

```
MATCH (a)-->(b)
WHERE b:B
OPTIONAL MATCH (a)-->(c)
WHERE c:C
RETURN a.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.22 Matching using an undirected pattern

Query specification (Matching using an undirected pattern)

```
MATCH (a)-[:ADMIN]-(b)
WHERE a:A
RETURN a.id, b.id
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.23 Matching all nodes

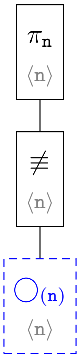
Query specification (Matching all nodes)

```
MATCH (n)
RETURN n
```

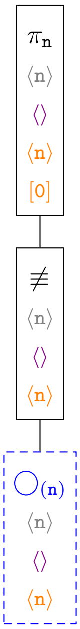
Relational algebra expression for search-based evaluation (Matching all nodes)

$$r = \pi_{\mathtt{n}}\Big(\not\equiv \Big(\bigcirc_{(\mathtt{n})} \Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching all nodes)



Incremental relational algebra tree (Matching all nodes)



A.19.24 Comparing nodes for equality

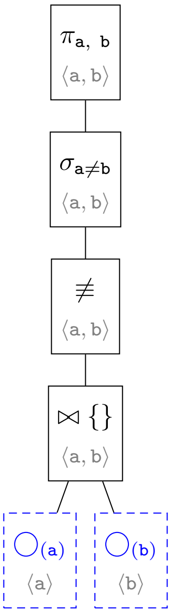
Query specification (Comparing nodes for equality)

```
MATCH (a), (b)
WHERE a <> b
RETURN a, b
```

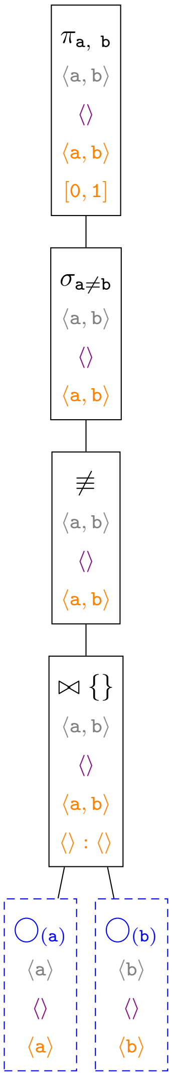
Relational algebra expression for search-based evaluation (Comparing nodes for equality)

$$r = \pi_{\texttt{a, b}}\Big(\sigma_{\texttt{a}\neq\texttt{b}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{a})} \bowtie \bigcirc_{(\texttt{b})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Comparing nodes for equality)


$\pi_{\texttt{a, b}}$
⟨a, b⟩
$\sigma_{\texttt{a}\neq\texttt{b}}$
⟨a, b⟩
$\not\equiv$
⟨a, b⟩
$\bowtie \{\}$
⟨a, b⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩
$\bigcirc_{(\texttt{b})}$
⟨b⟩

Incremental relational algebra tree (Comparing nodes for equality)



A.19.25 Matching using self-referencing pattern returns no result

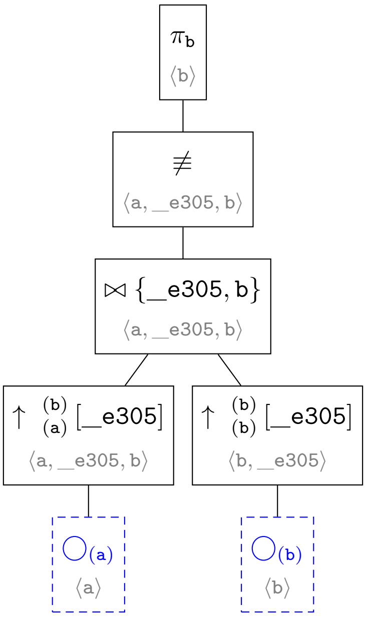
Query specification (Matching using self-referencing pattern returns no result)

```
MATCH (a)-->(b), (b)-->(b)
RETURN b
```

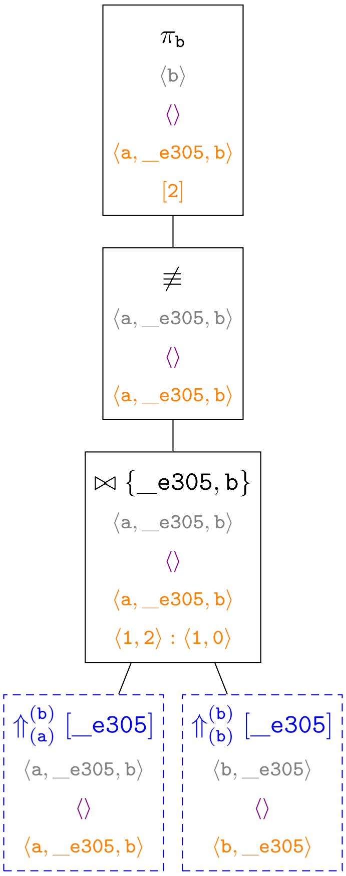
Relational algebra expression for search-based evaluation (Matching using self-referencing pattern returns no result)

$$r = \pi_{\mathtt{b}}\Big(\not\equiv \Big(\uparrow {}^{(\mathtt{b})}_{(\mathtt{a})} [\_\mathsf{e305}] \Big(\bigcirc_{(\mathtt{a})} \Big) \bowtie \uparrow {}^{(\mathtt{b})}_{(\mathtt{b})} [\_\mathsf{e305}] \Big(\bigcirc_{(\mathtt{b})} \Big) \Big) \Big)$$

Relational algebra tree for search-based evaluation (Matching using self-referencing pattern returns no result)



Incremental relational algebra tree (Matching using self-referencing pattern returns no result)



A.19.26 Variable length relationship in OPTIONAL MATCH

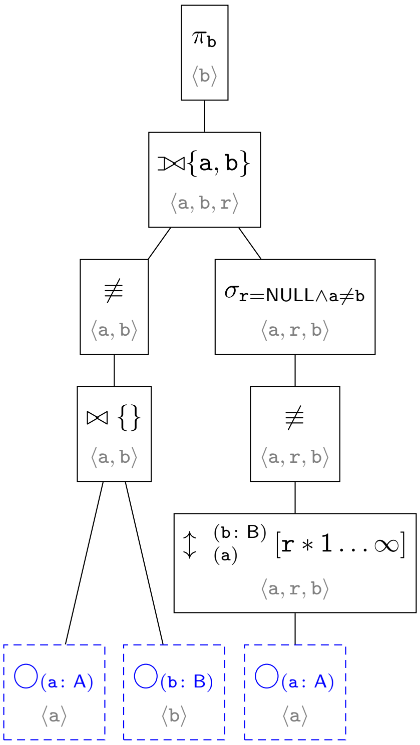
Query specification (Variable length relationship in OPTIONAL MATCH)

```
MATCH (a:A), (b:B)
OPTIONAL MATCH (a)-[r*]-(b)
WHERE r IS NULL
AND a <> b
RETURN b
```

Relational algebra expression for search-based evaluation (Variable length relationship in OPTIONAL MATCH)

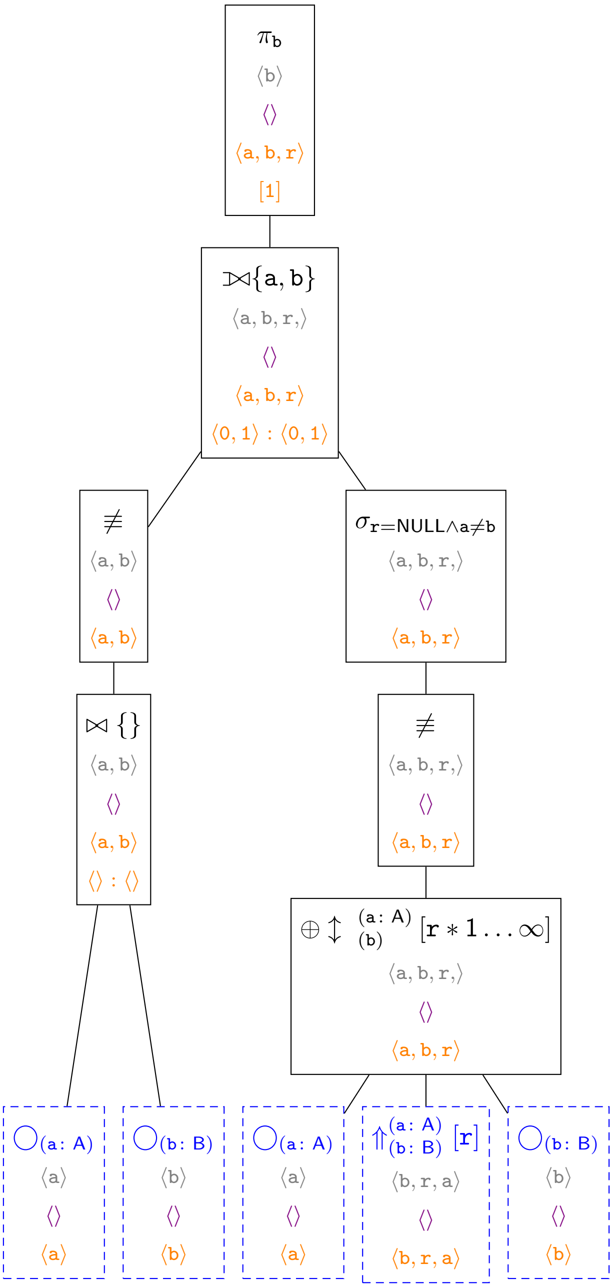
$$r = \pi_{\mathtt{b}}\Big(\not\equiv \Big(\bigcirc_{(\mathtt{a}:\ \mathtt{A})} \bowtie \bigcirc_{(\mathtt{b}:\ \mathtt{B})} \Big) \mathbin{⟕} \sigma_{\mathtt{r}=\mathtt{NULL} \land \mathtt{a} \neq \mathtt{b}} \Big(\not\equiv \Big(\updownarrow {}^{(\mathtt{b}:\ \mathtt{B})}_{(\mathtt{a})} [\mathtt{r} * 1 \ldots \infty] \Big(\bigcirc_{(\mathtt{a}:\ \mathtt{A})} \Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Variable length relationship in OPTIONAL MATCH)



$\pi_{\mathtt{b}}$
⟨b⟩
$\mathbin{⟕}\{\mathtt{a}, \mathtt{b}\}$
⟨a, b, r⟩
$\not\equiv$
⟨a, b⟩
$\sigma_{\mathtt{r}=\mathtt{NULL} \land \mathtt{a} \neq \mathtt{b}}$
⟨a, r, b⟩
$\bowtie \{\}$
⟨a, b⟩
$\not\equiv$
⟨a, r, b⟩
$\updownarrow {}^{(\mathtt{b}:\ \mathtt{B})}_{(\mathtt{a})} [\mathtt{r} * 1 \ldots \infty]$
⟨a, r, b⟩
$\bigcirc_{(\mathtt{a}:\ \mathtt{A})}$
⟨a⟩
$\bigcirc_{(\mathtt{b}:\ \mathtt{B})}$
⟨b⟩
$\bigcirc_{(\mathtt{a}:\ \mathtt{A})}$
⟨a⟩

Incremental relational algebra tree (Variable length relationship in OPTIONAL MATCH)



A.19.27 Matching using relationship predicate with multiples of the same type

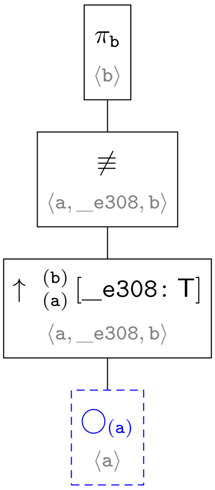
Query specification (Matching using relationship predicate with multiples of the same type)

```
MATCH (a)-[:T|:T]->(b)
RETURN b
```

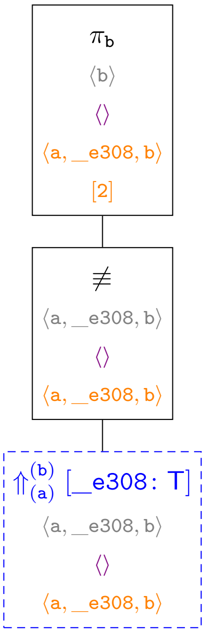
Relational algebra expression for search-based evaluation (Matching using relationship predicate with multiples of the same type)

$$r = \pi_{\mathsf{b}}\Big(\not\equiv \Big(\uparrow\,{}^{(\mathsf{b})}_{(\mathsf{a})}\, [\_\mathsf{e308}\colon \mathsf{T}] \Big(\bigcirc_{(\mathsf{a})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching using relationship predicate with multiples of the same type)



Incremental relational algebra tree (Matching using relationship predicate with multiples of the same type)



A.19.28 ORDER BY with LIMIT

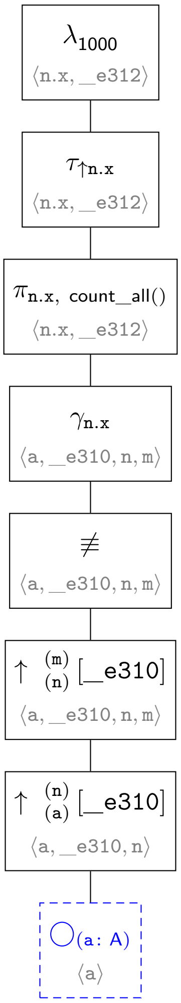
Query specification (ORDER BY with LIMIT)

```
MATCH (a:A)-->(n)-->(m)
RETURN n.x, count(*)
ORDER BY n.x
LIMIT 1000
```

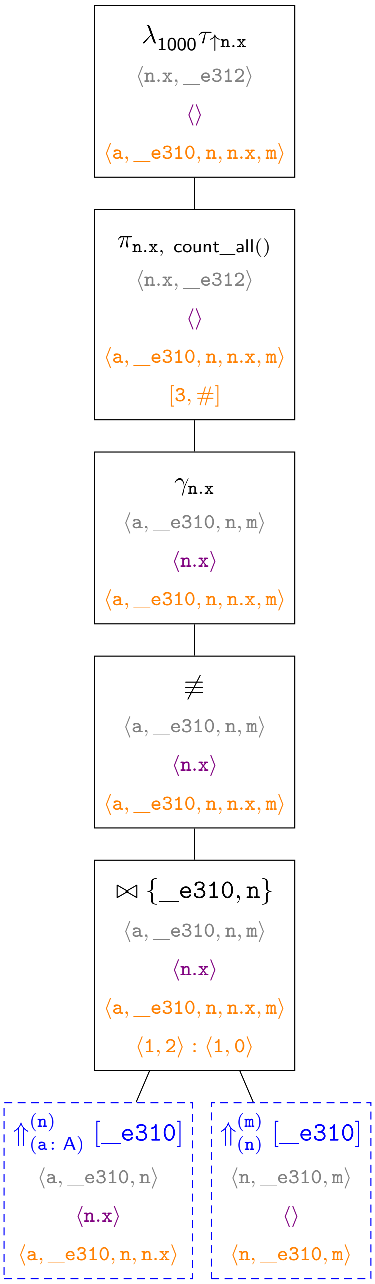
Relational algebra expression for search-based evaluation (ORDER BY with LIMIT)

$$r = \lambda_{1000}\Big(\tau_{\uparrow \mathtt{n.x}}\Big(\pi_{\mathtt{n.x},\ \mathsf{count\_all()}}\Big(\gamma_{\mathtt{n.x}}\Big(\not\equiv\Big(\uparrow {}^{(\mathtt{m})}_{(\mathtt{n})}\,[\_\mathsf{e310}]\Big(\uparrow {}^{(\mathtt{n})}_{(\mathtt{a})}\,[\_\mathsf{e310}]\Big(\bigcirc_{(\mathtt{a}:\ \mathsf{A})}\Big)\Big)\Big)\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (ORDER BY with LIMIT)


λ_{1000}
$\langle$n.x, __e312$\rangle$
$\tau_{\uparrow \texttt{n.x}}$
$\langle$n.x, __e312$\rangle$
$\pi_{\texttt{n.x},\ \text{count\_all}()}$
$\langle$n.x, __e312$\rangle$
$\gamma_{\texttt{n.x}}$
$\langle$a, __e310, n, m$\rangle$
$\not\equiv$
$\langle$a, __e310, n, m$\rangle$
$\uparrow$ (m) (n) [__e310]
$\langle$a, __e310, n, m$\rangle$
$\uparrow$ (n) (a) [__e310]
$\langle$a, __e310, n$\rangle$
$\bigcirc_{\texttt{(a: A)}}$
$\langle$a$\rangle$

Incremental relational algebra tree (ORDER BY with LIMIT)



A.19.29 Simple node property predicate

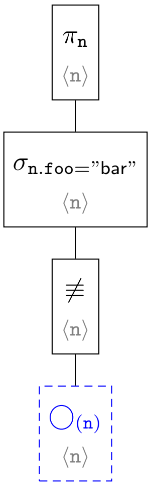
Query specification (Simple node property predicate)

```
MATCH (n)
WHERE n.foo = 'bar'
RETURN n
```

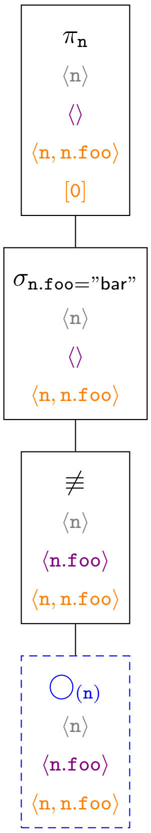
Relational algebra expression for search-based evaluation (Simple node property predicate)

$$r = \pi_{n}\Big(\sigma_{n.foo=\text{"bar"}}\Big(\not\equiv\Big(\bigcirc_{(n)}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Simple node property predicate)



Incremental relational algebra tree (Simple node property predicate)



A.19.30 Handling direction of named paths

Query specification (Handling direction of named paths)

```
MATCH p = (b)<--(a)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.31 Simple OPTIONAL MATCH on empty graph

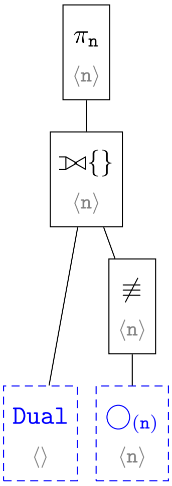
Query specification (Simple OPTIONAL MATCH on empty graph)

```
OPTIONAL MATCH (n)
RETURN n
```

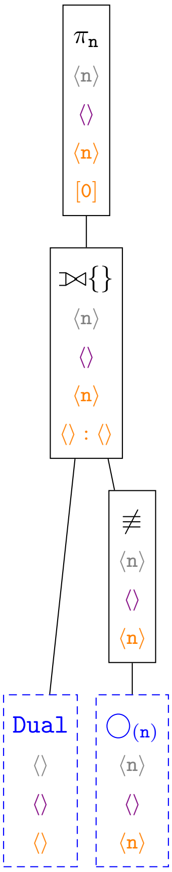
Relational algebra expression for search-based evaluation (Simple OPTIONAL MATCH on empty graph)

$$r = \pi_{\mathtt{n}}\Big(\mathtt{Dual} ⟕ \not\equiv \Big(\bigcirc_{(\mathtt{n})}\Big)\Big)$$

Relational algebra tree for search-based evaluation (Simple OPTIONAL MATCH on empty graph)



Incremental relational algebra tree (Simple OPTIONAL MATCH on empty graph)



A.19.32 OPTIONAL MATCH with previously bound nodes

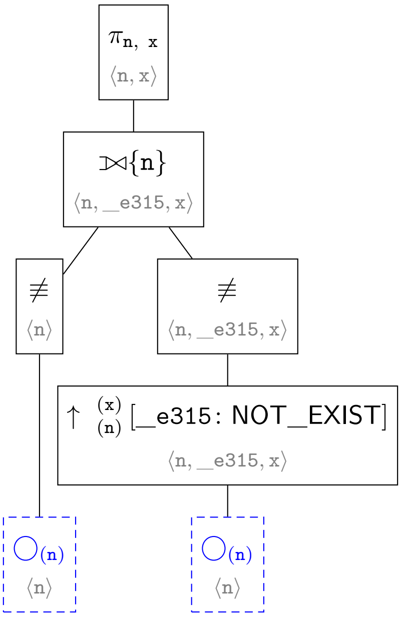
Query specification (OPTIONAL MATCH with previously bound nodes)

```
MATCH (n)
OPTIONAL MATCH (n)-[:NOT_EXIST]->(x)
RETURN n, x
```

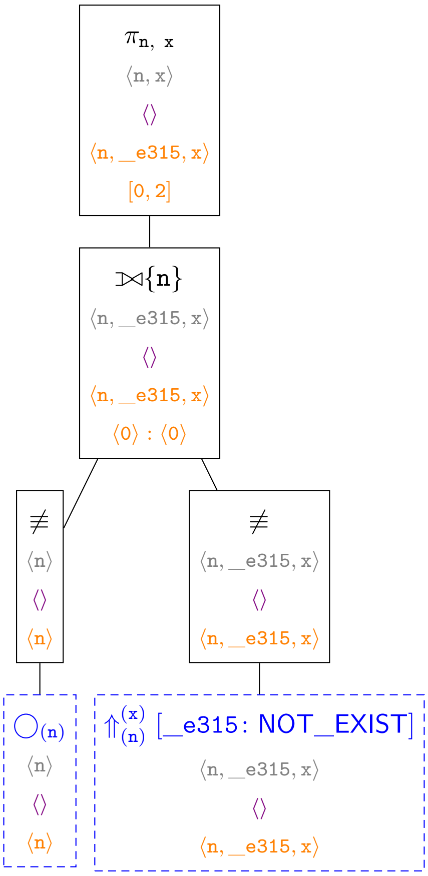
Relational algebra expression for search-based evaluation (OPTIONAL MATCH with previously bound nodes)

$$r = \pi_{\mathtt{n},\ \mathtt{x}}\left(\not\equiv \left(\bigcirc_{(\mathtt{n})} \right) ⟕ \not\equiv \left(\uparrow^{(\mathtt{x})}_{(\mathtt{n})} [\_\mathsf{e315}\colon \mathsf{NOT\_EXIST}] \left(\bigcirc_{(\mathtt{n})} \right) \right) \right)$$

Relational algebra tree for search-based evaluation (OPTIONAL MATCH with previously bound nodes)



Incremental relational algebra tree (OPTIONAL MATCH with previously bound nodes)



A.19.33 'collect()' filtering nulls

Query specification ('collect()' filtering nulls)

```
MATCH (n)
OPTIONAL MATCH (n)-[:NOT_EXIST]->(x)
```

```
RETURN n, collect(x)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.34 Multiple anonymous nodes in a pattern

Query specification (Multiple anonymous nodes in a pattern)

```
MATCH (a)<--()<--(b)-->()-->(c)
WHERE a:A
RETURN c
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.35 Matching a relationship pattern using a label predicate

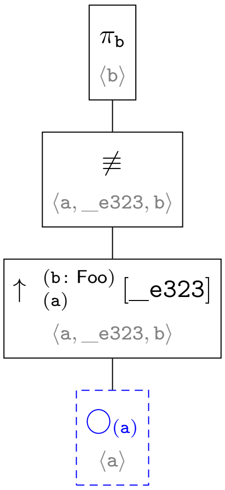
Query specification (Matching a relationship pattern using a label predicate)

```
MATCH (a)-->(b:Foo)
RETURN b
```

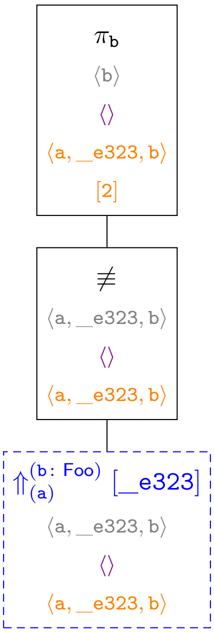
Relational algebra expression for search-based evaluation (Matching a relationship pattern using a label predicate)

$$r = \pi_{\mathsf{b}}\Big(\not\equiv \Big(\uparrow^{(\mathsf{b}:\,\mathsf{Foo})}_{(\mathsf{a})} [\_\mathsf{e323}] \Big(\bigcirc_{(\mathsf{a})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching a relationship pattern using a label predicate)


π_{b}
⟨b⟩
≢
⟨a, _e323, b⟩
$\uparrow^{(\mathsf{b}:\,\mathsf{Foo})}_{(\mathsf{a})} [\_\mathsf{e323}]$
⟨a, _e323, b⟩
$\bigcirc_{(\mathsf{a})}$
⟨a⟩

Incremental relational algebra tree (Matching a relationship pattern using a label predicate)



A.19.36 Matching a relationship pattern using a label predicate on both sides

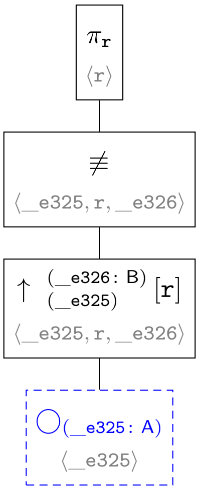
Query specification (Matching a relationship pattern using a label predicate on both sides)

```
MATCH (:A)-[r]->(:B)
RETURN r
```

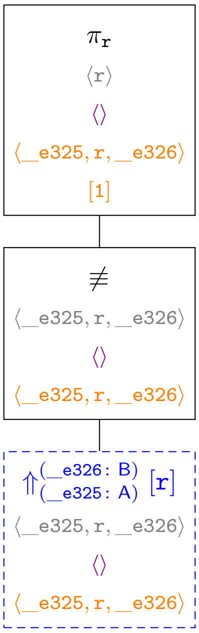
Relational algebra expression for search-based evaluation (Matching a relationship pattern using a label predicate on both sides)

$$r = \pi_{\mathtt{r}}\left(\not\equiv\left(\uparrow^{(\mathsf{\_e326:\ B})}_{(\mathsf{\_e325})}[\mathtt{r}]\left(\bigcirc_{(\mathsf{\_e325:\ A})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Matching a relationship pattern using a label predicate on both sides)


$\pi_{\mathtt{r}}$
⟨r⟩
$\not\equiv$
⟨_e325, r, _e326⟩
$\uparrow^{(\mathsf{\_e326:\ B})}_{(\mathsf{\_e325})}$ [r]
⟨_e325, r, _e326⟩
$\bigcirc_{(\mathsf{\_e325:\ A})}$
⟨_e325⟩

Incremental relational algebra tree (Matching a relationship pattern using a label predicate on both sides)



A.19.37 Matching nodes using multiple labels

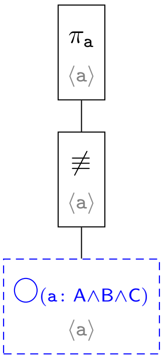
Query specification (Matching nodes using multiple labels)

```
MATCH (a:A:B:C)
RETURN a
```

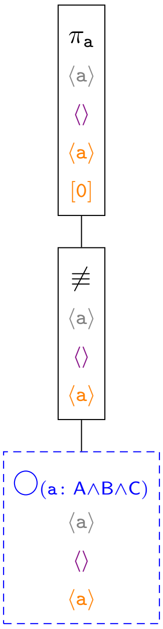
Relational algebra expression for search-based evaluation (Matching nodes using multiple labels)

$$r = \pi_{a}\Big(\not\equiv \Big(\bigcirc_{(a:\ A \wedge B \wedge C)} \Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching nodes using multiple labels)



Incremental relational algebra tree (Matching nodes using multiple labels)



A.19.38 Returning label predicate expression

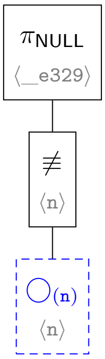
Query specification (Returning label predicate expression)

```
MATCH (n)
RETURN (n:Foo)
```

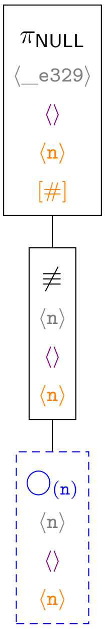
Relational algebra expression for search-based evaluation (Returning label predicate expression)

$$r = \pi_{\mathsf{NULL}}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)$$

Relational algebra tree for search-based evaluation (Returning label predicate expression)



Incremental relational algebra tree (Returning label predicate expression)



A.19.39 Matching with many predicates and larger pattern

Query specification (Matching with many predicates and larger pattern)

```
MATCH (advertiser)-[:ADV_HAS_PRODUCT]->(out)-[:AP_HAS_VALUE]->(red)<-[:AA_HAS_VALUE]-(a)
WHERE advertiser.id = $1
AND a.id = $2
AND red.name = 'red'
AND out.name = 'product1'
RETURN out.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.40 Matching using a simple pattern with label predicate

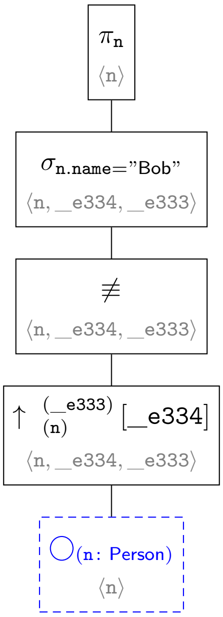
Query specification (Matching using a simple pattern with label predicate)

```
MATCH (n:Person)-->()
WHERE n.name = 'Bob'
RETURN n
```

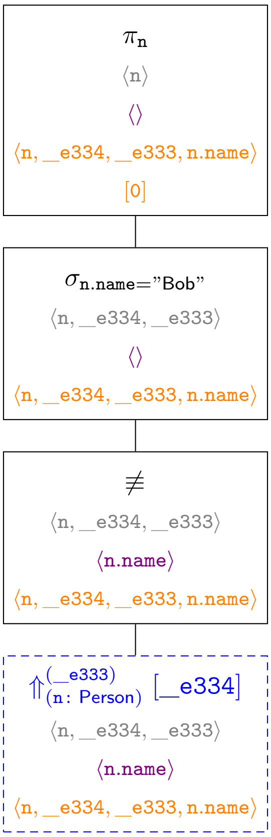
Relational algebra expression for search-based evaluation (Matching using a simple pattern with label predicate)

$$r = \pi_{\mathtt{n}}\left(\sigma_{\mathtt{n.name}=\text{"Bob"}}\left(\not\equiv\left(\uparrow\,{}^{(\text{\_e333})}_{(\mathtt{n})}\,[\text{\_e334}]\left(\bigcirc_{(\mathtt{n}:\ \text{Person})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Matching using a simple pattern with label predicate)



Incremental relational algebra tree (Matching using a simple pattern with label predicate)


$\pi_{\texttt{n}}$
⟨n⟩
⟨⟩
⟨n, _e334, _e333, n.name⟩
[0]
$\sigma_{\texttt{n.name}=\text{"Bob"}}$
⟨n, _e334, _e333⟩
⟨⟩
⟨n, _e334, _e333, n.name⟩
≢
⟨n, _e334, _e333⟩
⟨n.name⟩
⟨n, _e334, _e333, n.name⟩
⇑ (_e333) (n: Person) [_e334]
⟨n, _e334, _e333⟩
⟨n.name⟩
⟨n, _e334, _e333, n.name⟩

A.19.41 Matching disconnected patterns

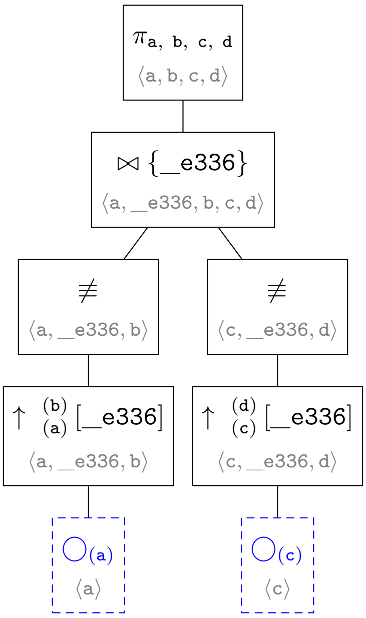
Query specification (Matching disconnected patterns)

```
MATCH (a)-->(b)
MATCH (c)-->(d)
RETURN a, b, c, d
```

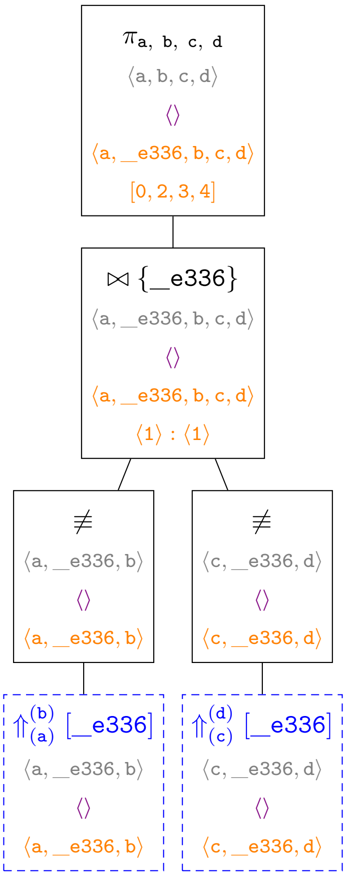
Relational algebra expression for search-based evaluation (Matching disconnected patterns)

$$r = \pi_{\mathtt{a,\ b,\ c,\ d}}\left(\not\equiv\left(\uparrow\ ^{(\mathtt{b})}_{(\mathtt{a})}[\_\mathsf{e336}]\left(\bigcirc_{(\mathtt{a})}\right)\right)\bowtie\not\equiv\left(\uparrow\ ^{(\mathtt{d})}_{(\mathtt{c})}[\_\mathsf{e336}]\left(\bigcirc_{(\mathtt{c})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Matching disconnected patterns)



Incremental relational algebra tree (Matching disconnected patterns)



A.19.42 Non-optional matches should not return nulls

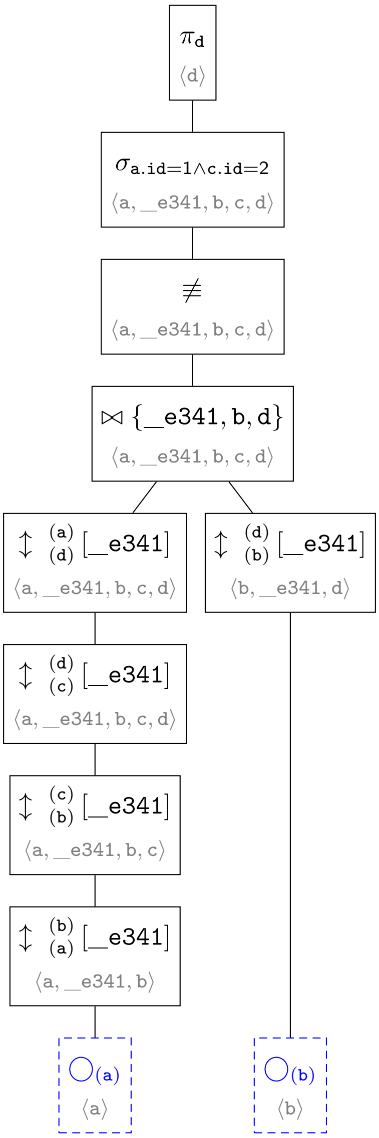
Query specification (Non-optional matches should not return nulls)

```
MATCH (a)--(b)--(c)--(d)--(a), (b)--(d)
WHERE a.id = 1
AND c.id = 2
RETURN d
```

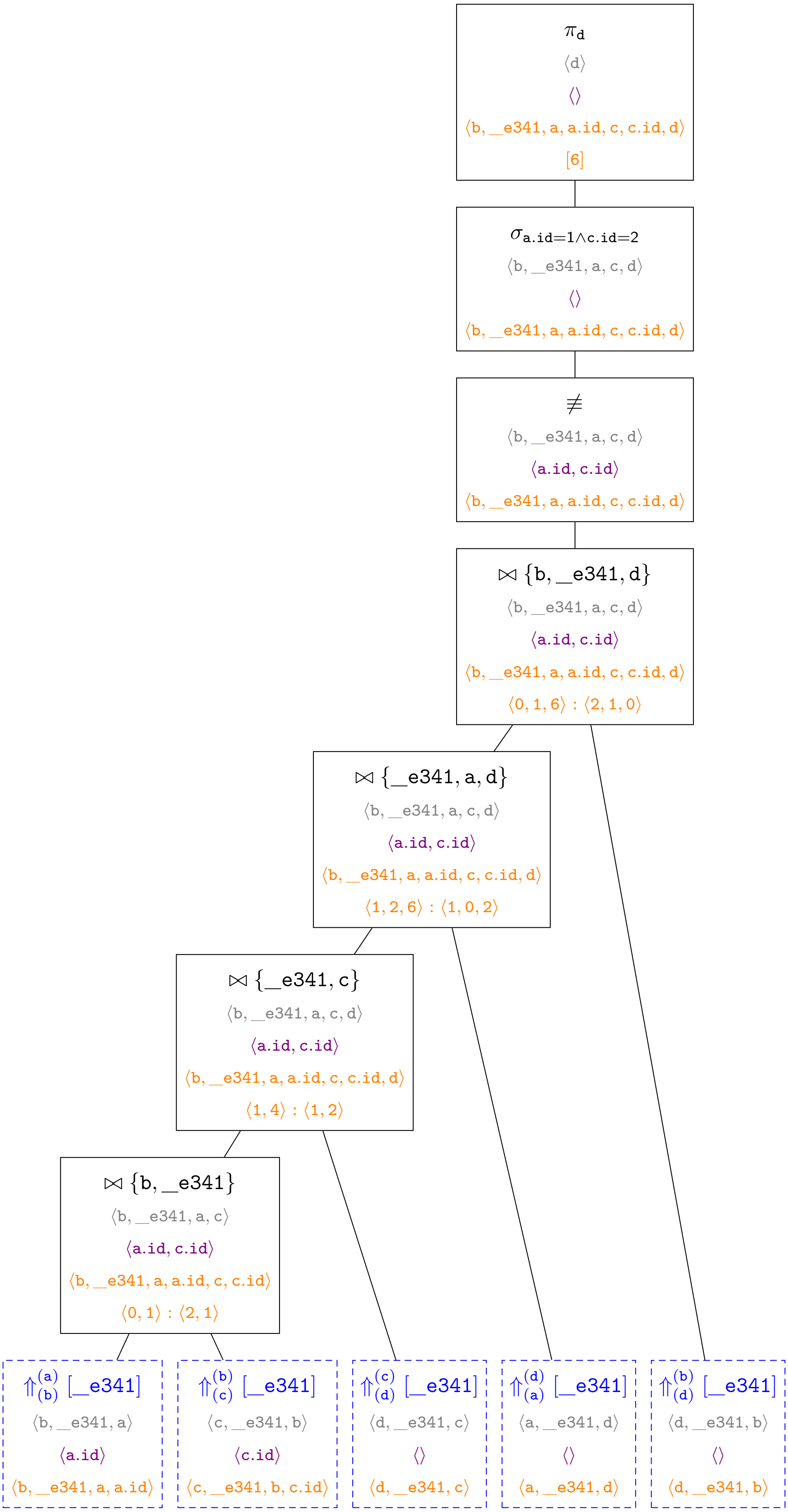
Relational algebra expression for search-based evaluation (Non-optional matches should not return nulls)

$$r = \pi_{\mathtt{d}}\Bigg(\sigma_{\mathtt{a.id}=1\wedge\mathtt{c.id}=2}\Bigg(\not\equiv\Bigg(\updownarrow\,{}^{(\mathtt{a})}_{(\mathtt{d})}\,[\_\mathsf{e341}]\Bigg(\updownarrow\,{}^{(\mathtt{d})}_{(\mathtt{c})}\,[\_\mathsf{e341}]\Bigg(\updownarrow\,{}^{(\mathtt{c})}_{(\mathtt{b})}\,[\_\mathsf{e341}]\Bigg(\updownarrow\,{}^{(\mathtt{b})}_{(\mathtt{a})}\,[\_\mathsf{e341}]\Bigg(\bigcirc_{(\mathtt{a})}\Bigg)\Bigg)\Bigg)\Bigg)\bowtie \updownarrow\,{}^{(\mathtt{d})}_{(\mathtt{b})}\,[\_\mathsf{e341}]\Bigg(\bigcirc_{(\mathtt{b})}\Bigg)\Bigg)\Bigg)\Bigg)$$

Relational algebra tree for search-based evaluation (Non-optional matches should not return nulls)


$\pi_{\mathtt{d}}$
⟨d⟩
$\sigma_{\mathtt{a.id}=1\wedge\mathtt{c.id}=2}$
⟨a, _e341, b, c, d⟩
$\not\equiv$
⟨a, _e341, b, c, d⟩
$\bowtie \{\_\mathsf{e341}, \mathsf{b}, \mathsf{d}\}$
⟨a, _e341, b, c, d⟩
$\updownarrow\,{}^{(\mathtt{a})}_{(\mathtt{d})}\,[\_\mathsf{e341}]$
⟨a, _e341, b, c, d⟩
$\updownarrow\,{}^{(\mathtt{d})}_{(\mathtt{b})}\,[\_\mathsf{e341}]$
⟨b, _e341, d⟩
$\updownarrow\,{}^{(\mathtt{d})}_{(\mathtt{c})}\,[\_\mathsf{e341}]$
⟨a, _e341, b, c, d⟩
$\updownarrow\,{}^{(\mathtt{c})}_{(\mathtt{b})}\,[\_\mathsf{e341}]$
⟨a, _e341, b, c⟩
$\updownarrow\,{}^{(\mathtt{b})}_{(\mathtt{a})}\,[\_\mathsf{e341}]$
⟨a, _e341, b⟩
$\bigcirc_{(\mathtt{a})}$
⟨a⟩
$\bigcirc_{(\mathtt{b})}$
⟨b⟩

Incremental relational algebra tree (Non-optional matches should not return nulls)


$\pi_{\mathtt{d}}$
⟨d⟩
⟨⟩
⟨b, _e341, a, a.id, c, c.id, d⟩
[6]
$\sigma_{\mathtt{a.id}=1 \land \mathtt{c.id}=2}$
⟨b, _e341, a, c, d⟩
⟨⟩
⟨b, _e341, a, a.id, c, c.id, d⟩
≢
⟨b, _e341, a, c, d⟩
⟨a.id, c.id⟩
⟨b, _e341, a, a.id, c, c.id, d⟩
⋈ {b, _e341, d}
⟨b, _e341, a, c, d⟩
⟨a.id, c.id⟩
⟨b, _e341, a, a.id, c, c.id, d⟩
⟨0, 1, 6⟩ : ⟨2, 1, 0⟩
⋈ {_e341, a, d}
⟨b, _e341, a, c, d⟩
⟨a.id, c.id⟩
⟨b, _e341, a, a.id, c, c.id, d⟩
⟨1, 2, 6⟩ : ⟨1, 0, 2⟩
⋈ {_e341, c}
⟨b, _e341, a, c, d⟩
⟨a.id, c.id⟩
⟨b, _e341, a, a.id, c, c.id, d⟩
⟨1, 4⟩ : ⟨1, 2⟩
⋈ {b, _e341}
⟨b, _e341, a, c⟩
⟨a.id, c.id⟩
⟨b, _e341, a, a.id, c, c.id⟩
⟨0, 1⟩ : ⟨2, 1⟩
$\Uparrow^{(\mathtt{a})}_{(\mathtt{b})}$ [_e341]
⟨b, _e341, a⟩
⟨a.id⟩
⟨b, _e341, a, a.id⟩
$\Uparrow^{(\mathtt{b})}_{(\mathtt{c})}$ [_e341]
⟨c, _e341, b⟩
⟨c.id⟩
⟨c, _e341, b, c.id⟩
$\Uparrow^{(\mathtt{c})}_{(\mathtt{d})}$ [_e341]
⟨d, _e341, c⟩
⟨⟩
⟨d, _e341, c⟩
$\Uparrow^{(\mathtt{d})}_{(\mathtt{a})}$ [_e341]
⟨a, _e341, d⟩
⟨⟩
⟨a, _e341, d⟩
$\Uparrow^{(\mathtt{b})}_{(\mathtt{d})}$ [_e341]
⟨d, _e341, b⟩
⟨⟩
⟨d, _e341, b⟩

A.19.43 Handling cyclic patterns

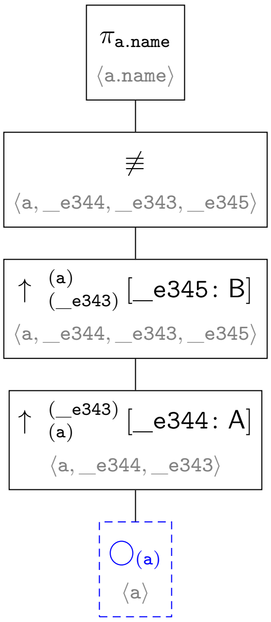
Query specification (Handling cyclic patterns)

```
MATCH (a)-[:A]->()-[:B]->(a)
RETURN a.name
```

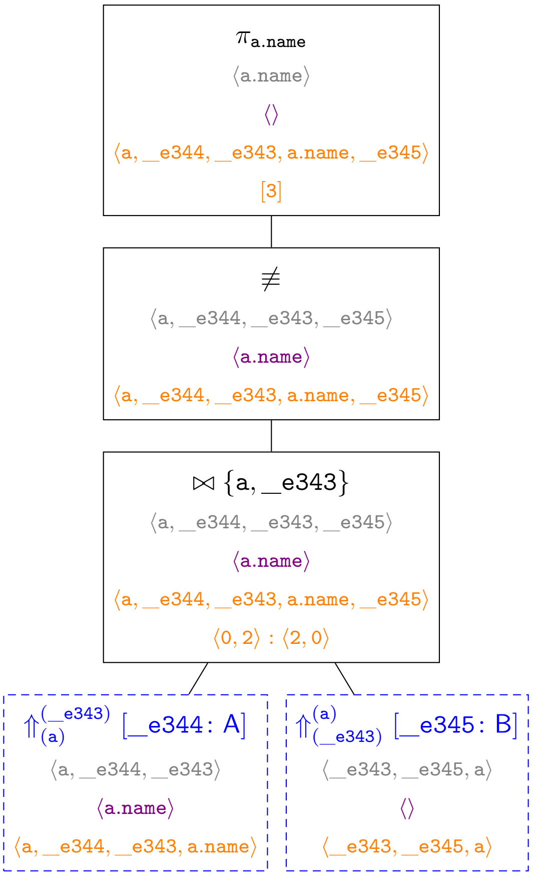
Relational algebra expression for search-based evaluation (Handling cyclic patterns)

$$r = \pi_{\texttt{a.name}}\Big(\not\equiv\Big(\uparrow\,^{(\texttt{a})}_{(\_\texttt{e343})}\,[\_\mathsf{e345}\colon\mathsf{B}]\Big(\uparrow\,^{(\_\texttt{e343})}_{(\texttt{a})}\,[\_\mathsf{e344}\colon\mathsf{A}]\Big(\bigcirc_{(\texttt{a})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Handling cyclic patterns)



Incremental relational algebra tree (Handling cyclic patterns)



A.19.44 Handling cyclic patterns when separated into two parts

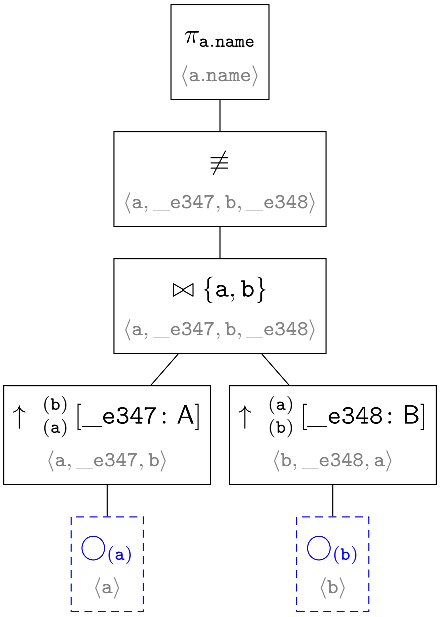
Query specification (Handling cyclic patterns when separated into two parts)

```
MATCH (a)-[:A]->(b), (b)-[:B]->(a)
RETURN a.name
```

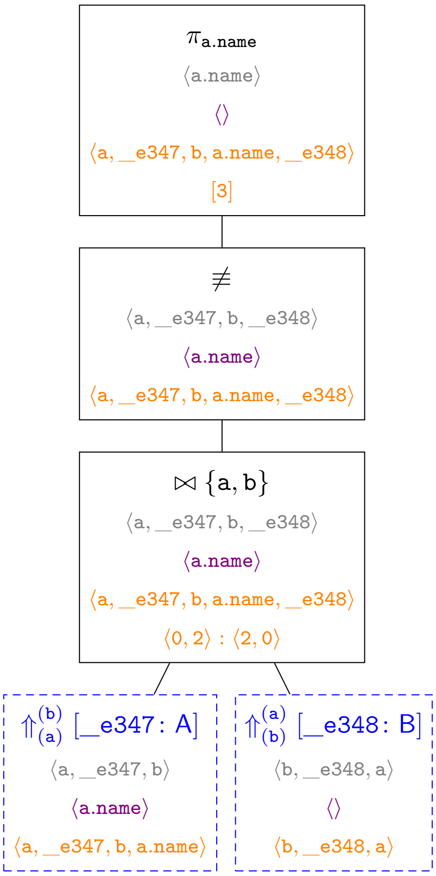
Relational algebra expression for search-based evaluation (Handling cyclic patterns when separated into two parts)

$$r = \pi_{\texttt{a.name}}\Big(\not\equiv\Big(\uparrow\, {}^{(\texttt{b})}_{(\texttt{a})}\,[\_e347\colon \mathsf{A}]\Big(\bigcirc_{(\texttt{a})}\Big) \bowtie\!\uparrow\, {}^{(\texttt{a})}_{(\texttt{b})}\,[\_e348\colon \mathsf{B}]\Big(\bigcirc_{(\texttt{b})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Handling cyclic patterns when separated into two parts)



Incremental relational algebra tree (Handling cyclic patterns when separated into two parts)



A.19.45 Handling fixed-length variable length pattern

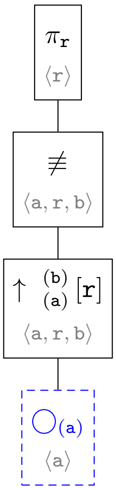
Query specification (Handling fixed-length variable length pattern)

```
MATCH (a)-[r*1..1]->(b)
RETURN r
```

Relational algebra expression for search-based evaluation (Handling fixed-length variable length pattern)

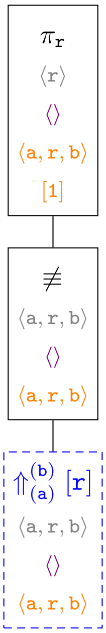
$$r = \pi_{\mathtt{r}}\left(\not\equiv \left(\uparrow {}^{(\mathtt{b})}_{(\mathtt{a})} [\mathtt{r}] \left(\bigcirc_{(\mathtt{a})} \right)\right)\right)$$

Relational algebra tree for search-based evaluation (Handling fixed-length variable length pattern)



$\pi_{\mathtt{r}}$
⟨r⟩
≢
⟨a, r, b⟩
$\uparrow {}^{(\mathtt{b})}_{(\mathtt{a})} [\mathtt{r}]$
⟨a, r, b⟩
$\bigcirc_{(\mathtt{a})}$
⟨a⟩

Incremental relational algebra tree (Handling fixed-length variable length pattern)



A.19.46 Matching from null nodes should return no results owing to finding no matches

Query specification (Matching from null nodes should return no results owing to finding no matches)

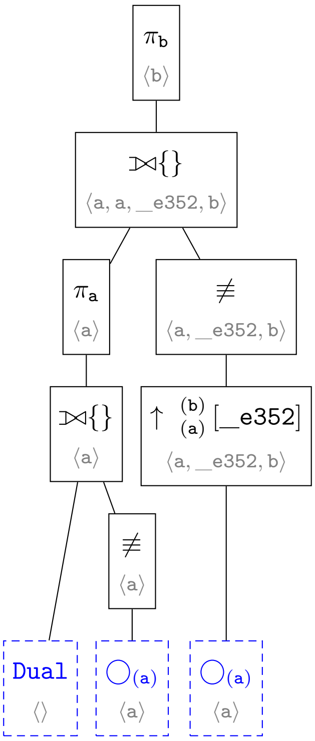
```
OPTIONAL MATCH (a)
WITH a
MATCH (a)-->(b)
```

```
RETURN b
```

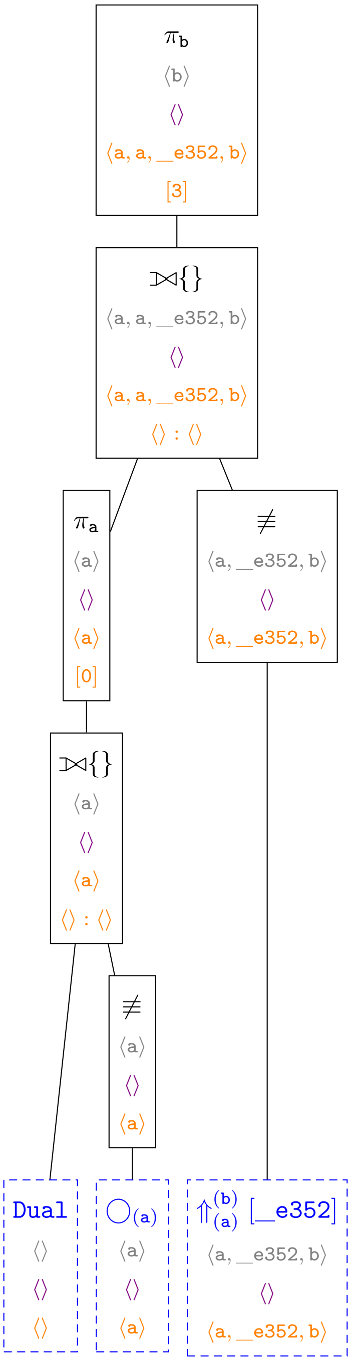
Relational algebra expression for search-based evaluation (Matching from null nodes should return no results owing to finding no matches)

$$r = \pi_{\mathtt{b}}\Big(\pi_{\mathtt{a}}\Big(\mathtt{Dual} ⟕ \not\equiv \Big(\bigcirc_{(\mathtt{a})}\Big)\Big) ⟕ \not\equiv \Big(\uparrow \,{}^{(\mathtt{b})}_{(\mathtt{a})}\,[\_\mathtt{e352}]\Big(\bigcirc_{(\mathtt{a})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching from null nodes should return no results owing to finding no matches)


$\pi_{\mathtt{b}}$
⟨b⟩
⟕{}
⟨a, a, _e352, b⟩
$\pi_{\mathtt{a}}$
⟨a⟩
≢
⟨a, _e352, b⟩
⟕{}
⟨a⟩
$\uparrow {}^{(\mathtt{b})}_{(\mathtt{a})} [\_\mathtt{e352}]$
⟨a, _e352, b⟩
≢
⟨a⟩
Dual
⟨⟩
$\bigcirc_{(\mathtt{a})}$
⟨a⟩
$\bigcirc_{(\mathtt{a})}$
⟨a⟩

Incremental relational algebra tree (Matching from null nodes should return no results owing to finding no matches)



A.19.47 Matching from null nodes should return no results owing to matches being filtered out

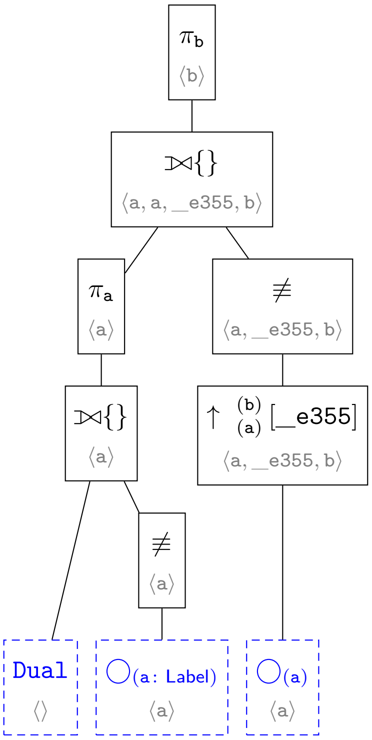
Query specification (Matching from null nodes should return no results owing to matches being filtered out)

```
OPTIONAL MATCH (a:Label)
WITH a
MATCH (a)-->(b)
RETURN b
```

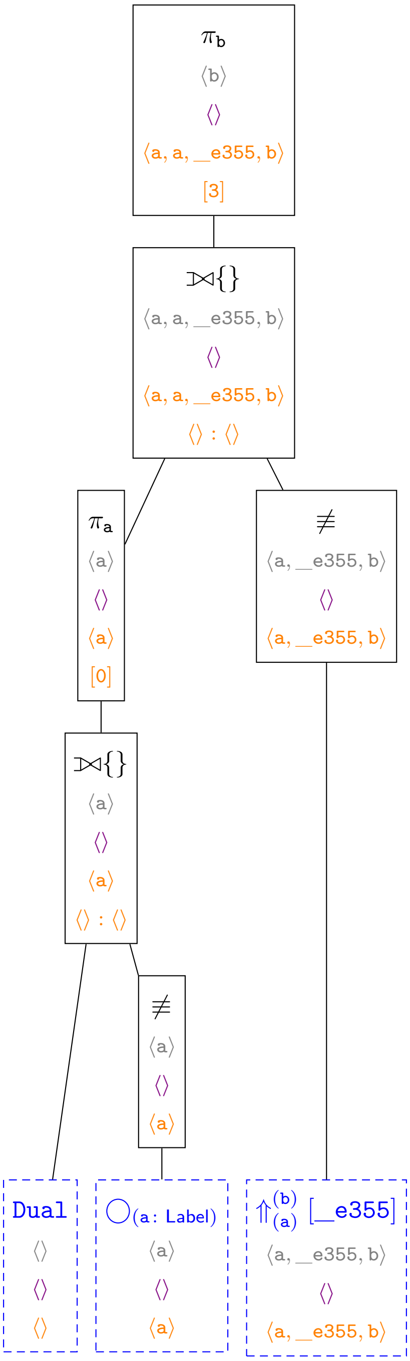
Relational algebra expression for search-based evaluation (Matching from null nodes should return no results owing to matches being filtered out)

$$r = \pi_{\mathtt{b}}\Big(\pi_{\mathtt{a}}\Big(\mathtt{Dual} ⟕ \not\equiv \Big(\bigcirc_{(\mathtt{a}:\ \mathsf{Label})}\Big)\Big) ⟕ \not\equiv \Big(\uparrow^{(\mathtt{b})}_{(\mathtt{a})}[\_\mathsf{e355}]\Big(\bigcirc_{(\mathtt{a})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching from null nodes should return no results owing to matches being filtered out)



Incremental relational algebra tree (Matching from null nodes should return no results owing to matches being filtered out)



A.19.48 Optionally matching from null nodes should return null

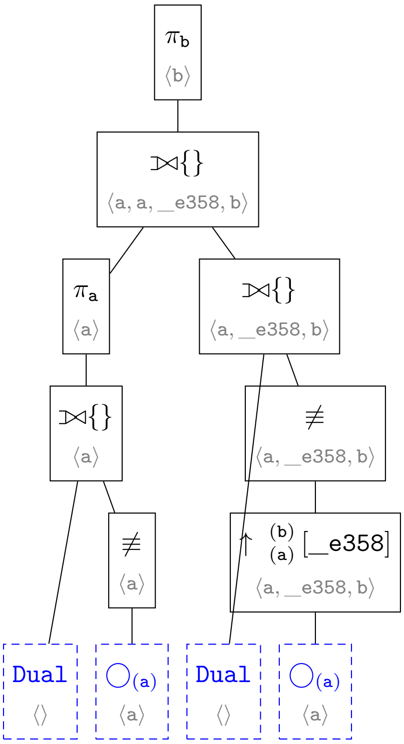
Query specification (Optionally matching from null nodes should return null)

```
OPTIONAL MATCH (a)
WITH a
OPTIONAL MATCH (a)-->(b)
RETURN b
```

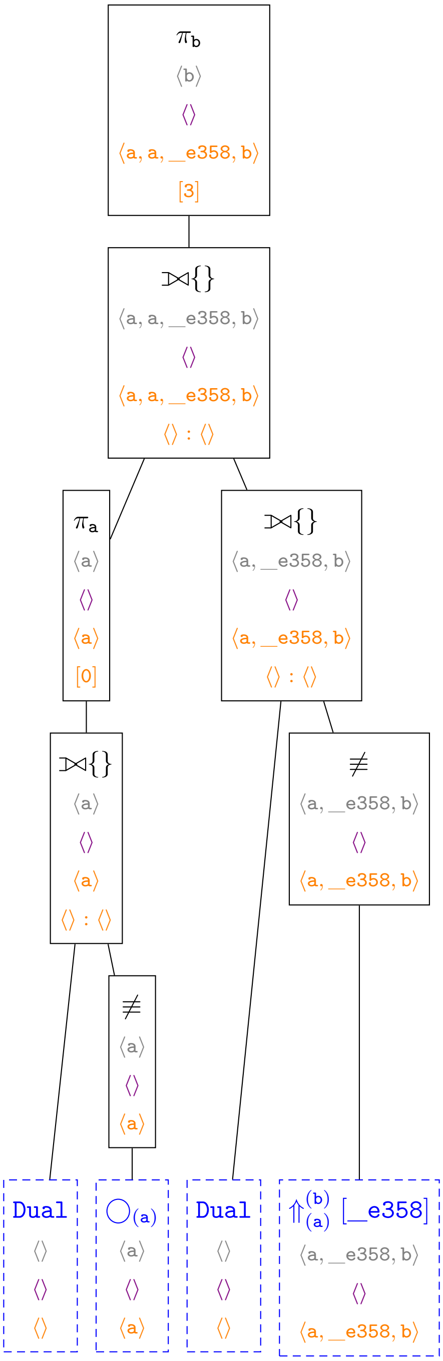
Relational algebra expression for search-based evaluation (Optionally matching from null nodes should return null)

$$r = \pi_{\mathtt{b}}\Big(\pi_{\mathtt{a}}\Big(\mathtt{Dual} ⟕ \not\equiv \Big(\bigcirc_{(\mathtt{a})}\Big)\Big) \bowtie \mathtt{Dual} ⟕ \not\equiv \Big(\uparrow {}^{(\mathtt{b})}_{(\mathtt{a})} [\_\mathtt{e358}] \Big(\bigcirc_{(\mathtt{a})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Optionally matching from null nodes should return null)



Incremental relational algebra tree (Optionally matching from null nodes should return null)



A.19.49 OPTIONAL MATCH returns null

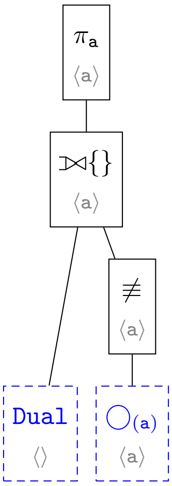
Query specification (OPTIONAL MATCH returns null)

```
OPTIONAL MATCH (a)
RETURN a
```

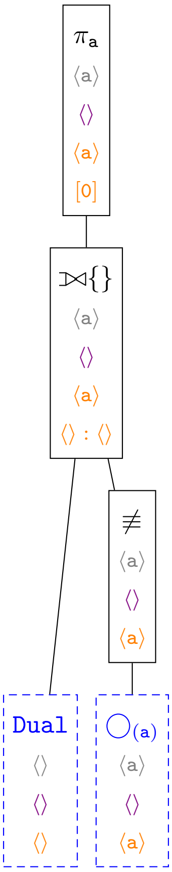
Relational algebra expression for search-based evaluation (OPTIONAL MATCH returns null)

$$r = \pi_{a}\Big(\texttt{Dual} \rtimes\!\!\!\ltimes \not\equiv \Big(\bigcirc_{(a)}\Big)\Big)$$

Relational algebra tree for search-based evaluation (OPTIONAL MATCH returns null)



Incremental relational algebra tree (OPTIONAL MATCH returns null)



A.19.50 Zero-length named path

Query specification (Zero-length named path)

```
MATCH p = (a)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.51 Variable-length named path

Query specification (Variable-length named path)

```
MATCH p = ()-[*0..]->()
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.52 Matching with aggregation

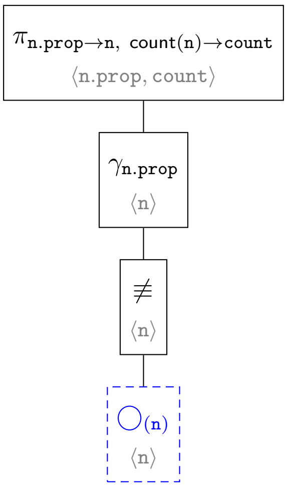
Query specification (Matching with aggregation)

```
MATCH (n)
RETURN n.prop AS n, count(n) AS count
```

Relational algebra expression for search-based evaluation (Matching with aggregation)

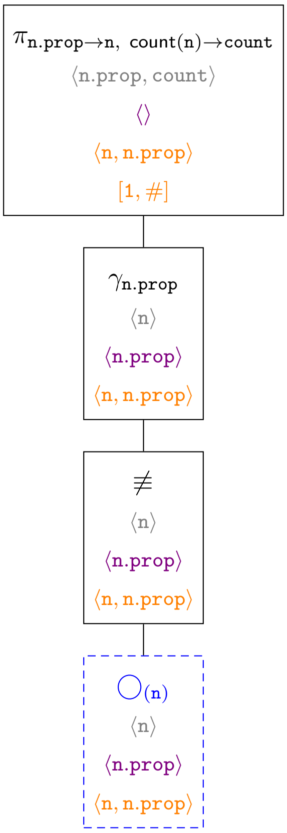
$$r = \pi_{\mathtt{n.prop} \rightarrow \mathtt{n},\ \mathsf{count}(\mathtt{n}) \rightarrow \mathsf{count}}\left(\gamma_{\mathtt{n.prop}}\left(\not\equiv\left(\bigcirc_{(\mathtt{n})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Matching with aggregation)



$\pi_{\mathtt{n.prop} \rightarrow \mathtt{n},\ \mathsf{count}(\mathtt{n}) \rightarrow \mathsf{count}}$
⟨n.prop, count⟩
$\gamma_{\mathtt{n.prop}}$
⟨n⟩
$\not\equiv$
⟨n⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩

Incremental relational algebra tree (Matching with aggregation)



A.19.53 Matching using a relationship that is already bound

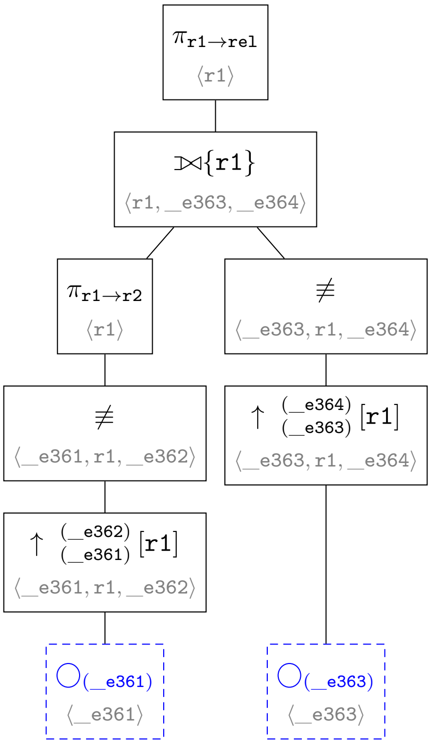
Query specification (Matching using a relationship that is already bound)

```
MATCH ()-[r1]->()
WITH r1 AS r2
MATCH ()-[r2]->()
RETURN r2 AS rel
```

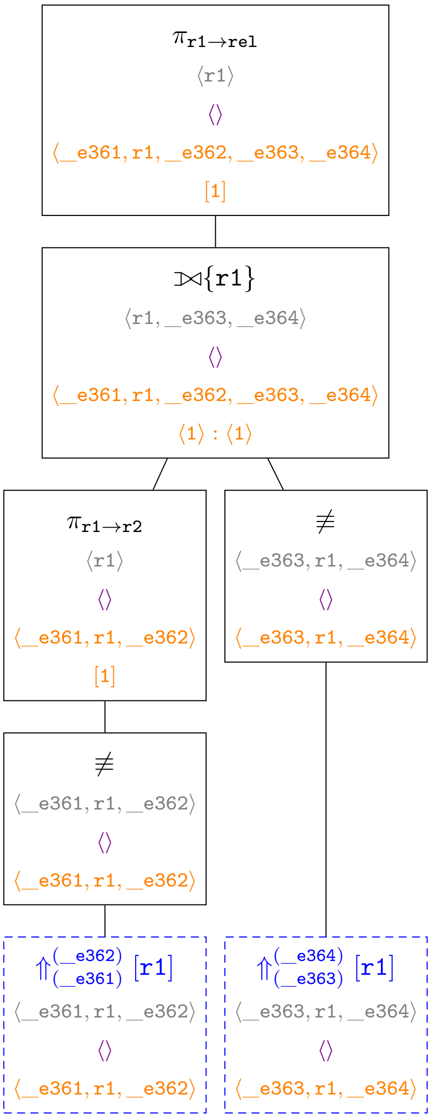
Relational algebra expression for search-based evaluation (Matching using a relationship that is already bound)

$$r = \pi_{\texttt{r1}\rightarrow\texttt{rel}}\left(\pi_{\texttt{r1}\rightarrow\texttt{r2}}\left(\not\equiv\left(\uparrow^{(\_\text{e362})}_{(\_\text{e361})}[\texttt{r1}]\left(\bigcirc_{(\_\text{e361})}\right)\right)\right)\bowtie\not\equiv\left(\uparrow^{(\_\text{e364})}_{(\_\text{e363})}[\texttt{r1}]\left(\bigcirc_{(\_\text{e363})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Matching using a relationship that is already bound)



Incremental relational algebra tree (Matching using a relationship that is already bound)



A.19.54 Matching using a relationship that is already bound, in conjunction with aggregation

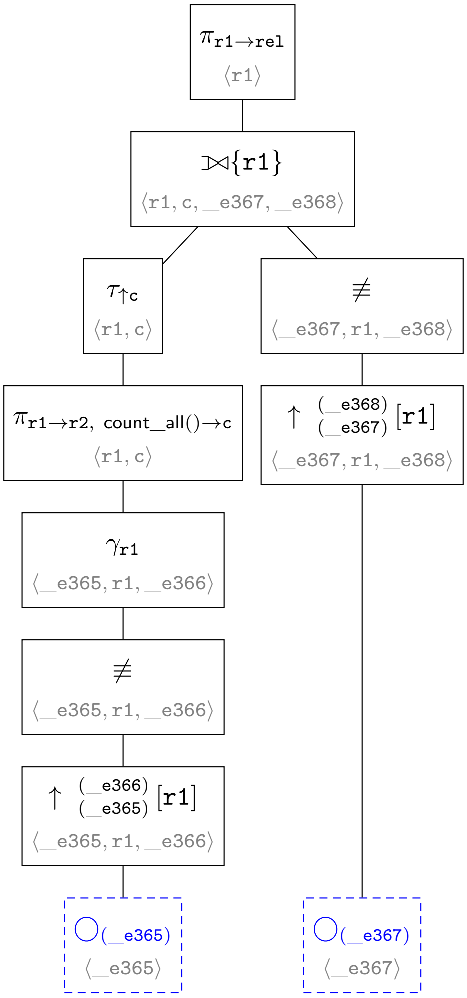
Query specification (Matching using a relationship that is already bound, in conjunction with aggregation)

```
MATCH ()-[r1]->()
WITH r1 AS r2, count(*) AS c
ORDER BY c
MATCH ()-[r2]->()
RETURN r2 AS rel
```

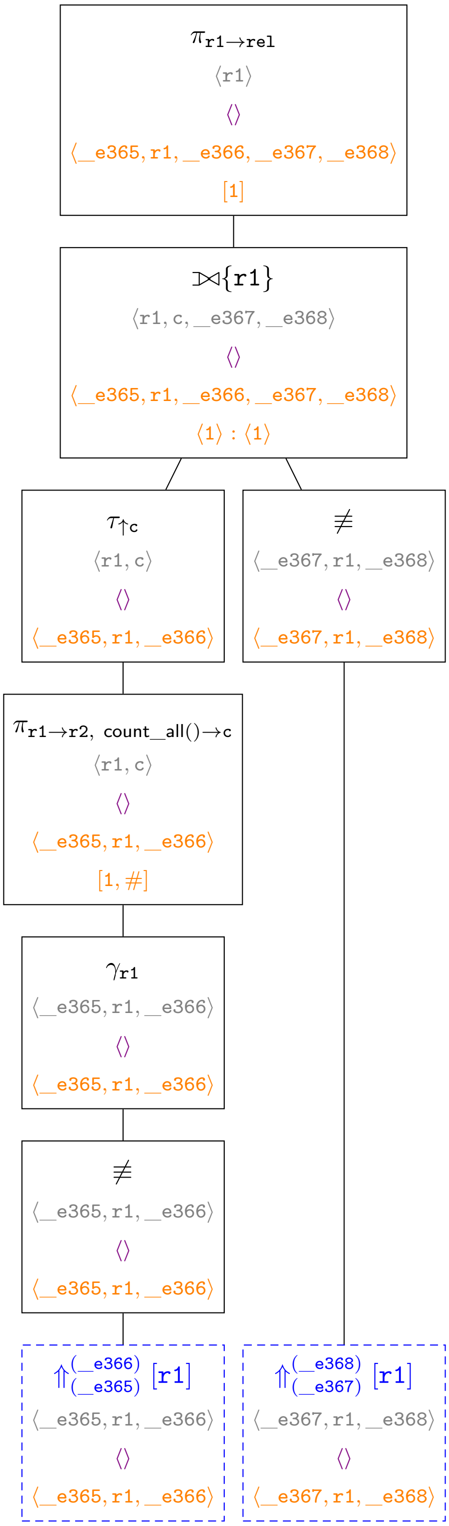
Relational algebra expression for search-based evaluation (Matching using a relationship that is already bound, in conjunction with aggregation)

$$r = \pi_{\texttt{r1}\rightarrow\texttt{rel}}\Big(\tau_{\uparrow\mathsf{c}}\Big(\pi_{\texttt{r1}\rightarrow\texttt{r2},\ \mathsf{count\_all()}\rightarrow\mathsf{c}}\Big(\gamma_{\texttt{r1}}\Big(\not\equiv\Big(\uparrow\,{}^{(\_\mathsf{e366})}_{(\_\mathsf{e365})}\,[\texttt{r1}]\Big(\bigcirc_{(\_\mathsf{e365})}\Big)\Big)\Big)\Big)\Big)\Big)\bowtie\not\equiv\Big(\uparrow\,{}^{(\_\mathsf{e368})}_{(\_\mathsf{e367})}\,[\texttt{r1}]\Big(\bigcirc_{(\_\mathsf{e367})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching using a relationship that is already bound, in conjunction with aggregation)


$\pi_{\mathtt{r1} \rightarrow \mathtt{rel}}$
⟨r1⟩
$⋊\{\mathtt{r1}\}$
⟨r1, c, _e367, _e368⟩
$\tau_{\uparrow c}$
⟨r1, c⟩
≢
⟨_e367, r1, _e368⟩
$\pi_{\mathtt{r1} \rightarrow \mathtt{r2},\ \mathsf{count\_all}() \rightarrow c}$
⟨r1, c⟩
$\uparrow\ {}^{(\_e368)}_{(\_e367)}\ [\mathtt{r1}]$
⟨_e367, r1, _e368⟩
$\gamma_{\mathtt{r1}}$
⟨_e365, r1, _e366⟩
≢
⟨_e365, r1, _e366⟩
$\uparrow\ {}^{(\_e366)}_{(\_e365)}\ [\mathtt{r1}]$
⟨_e365, r1, _e366⟩
○(_e365)
⟨_e365⟩
○(_e367)
⟨_e367⟩

Incremental relational algebra tree (Matching using a relationship that is already bound, in conjunction with aggregation)



A.19.55 Matching using a relationship that is already bound, in conjunction with aggregation and ORDER BY

Query specification (Matching using a relationship that is already bound, in conjunction with aggregation and ORDER BY)

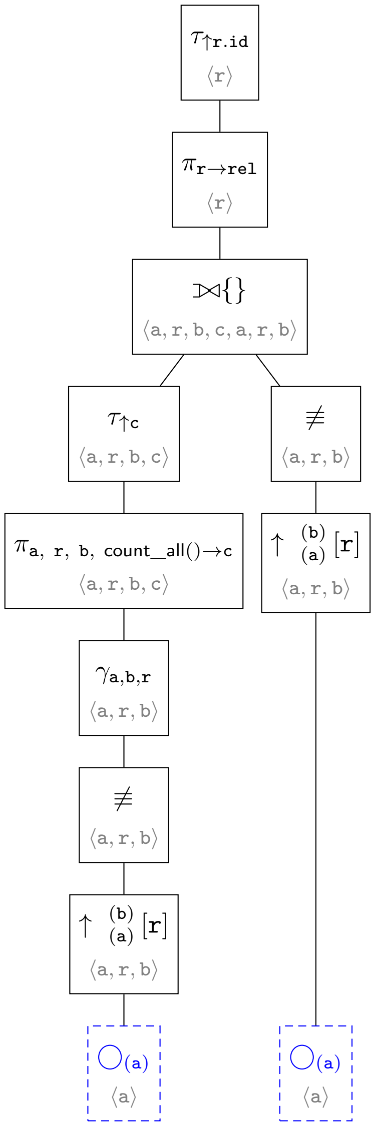
```
MATCH (a)-[r]->(b)
```

```
WITH a, r, b, count(*) AS c
ORDER BY c
MATCH (a)-[r]->(b)
RETURN r AS rel
ORDER BY rel.id
```

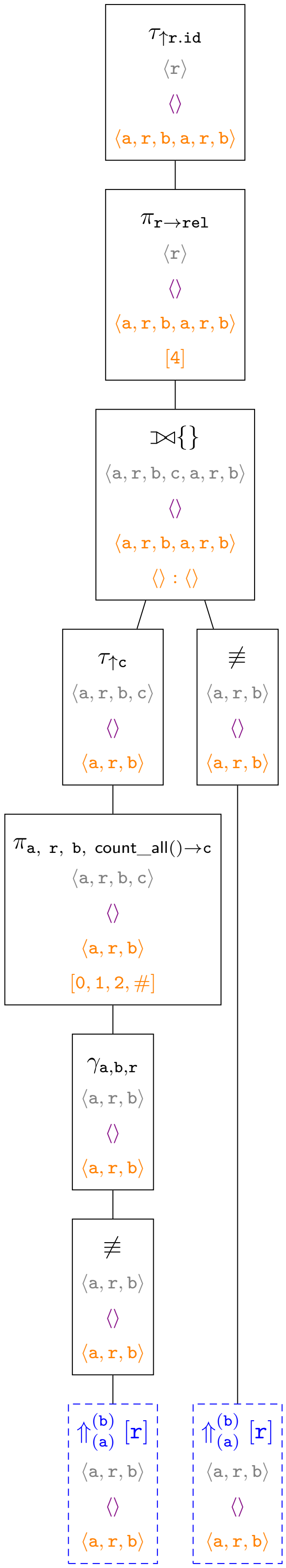
Relational algebra expression for search-based evaluation (Matching using a relationship that is already bound, in conjunction with aggregation and ORDER BY)

$$r = \tau_{\uparrow \mathtt{r.id}}\Bigg(\pi_{\mathtt{r} \rightarrow \mathtt{rel}}\Bigg(\tau_{\uparrow \mathtt{c}}\Bigg(\pi_{\mathtt{a},\ \mathtt{r},\ \mathtt{b},\ \mathsf{count\_all()} \rightarrow \mathtt{c}}\Bigg(\gamma_{\mathtt{a,b,r}}\Bigg(\not\equiv\Bigg(\uparrow \ \substack{\mathtt{(b)}\\ \mathtt{(a)}} [\mathtt{r}] \Bigg(\bigcirc_{(\mathtt{a})}\Bigg)\Bigg)\Bigg)\Bigg)\Bigg) \Join \not\equiv\Bigg(\uparrow \ \substack{\mathtt{(b)}\\ \mathtt{(a)}} [\mathtt{r}] \Bigg(\bigcirc_{(\mathtt{a})}\Bigg)\Bigg)\Bigg)$$
$$\Bigg)$$

Relational algebra tree for search-based evaluation (Matching using a relationship that is already bound, in conjunction with aggregation and ORDER BY)



Incremental relational algebra tree (Matching using a relationship that is already bound, in conjunction with aggregation and ORDER BY)



A.19.56 Matching with LIMIT and optionally matching using a relationship that is already bound

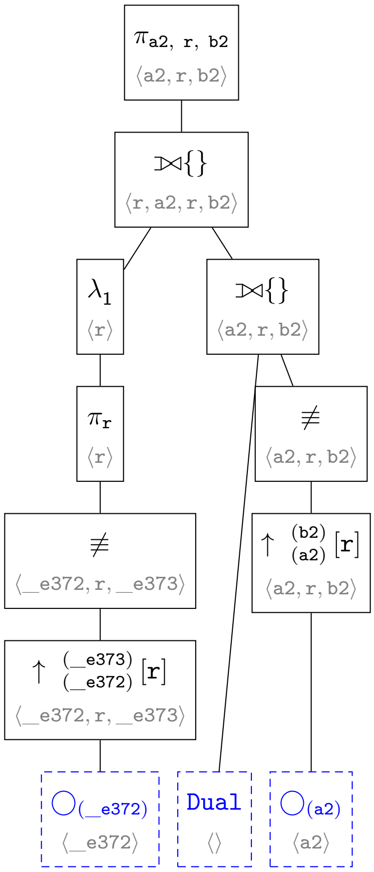
Query specification (Matching with LIMIT and optionally matching using a relationship that is already bound)

```
MATCH ()-[r]->()
WITH r
LIMIT 1
OPTIONAL MATCH (a2)-[r]->(b2)
RETURN a2, r, b2
```

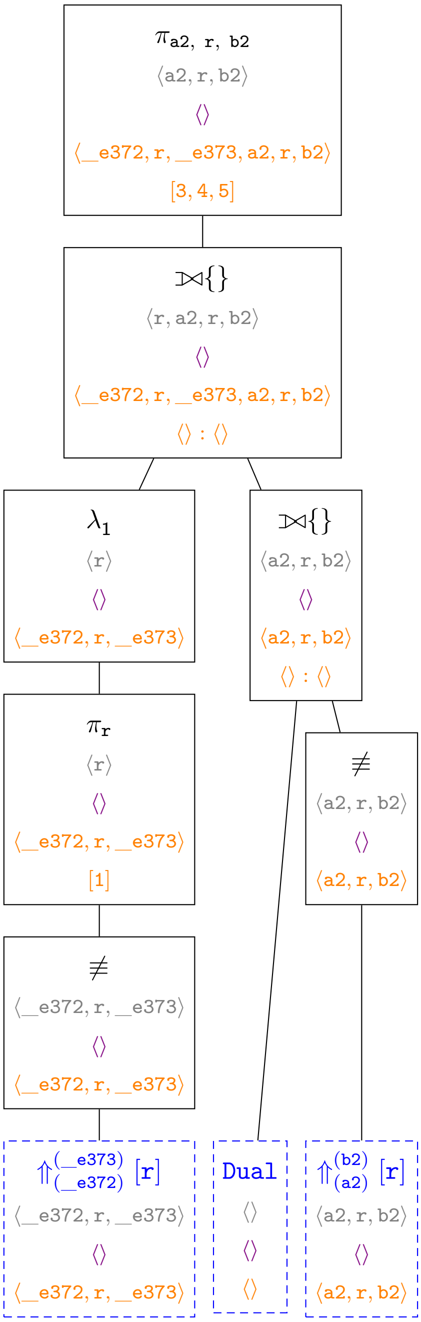
Relational algebra expression for search-based evaluation (Matching with LIMIT and optionally matching using a relationship that is already bound)

$$r = \pi_{\texttt{a2, r, b2}}\Big(\lambda_1\Big(\pi_{\texttt{r}}\Big(\not\equiv\Big(\uparrow\,^{(\texttt{\_e373})}_{(\texttt{\_e372})}[\texttt{r}]\Big(\bigcirc_{(\texttt{\_e372})}\Big)\Big)\Big)\Big)\Big) ⟕ \texttt{Dual} ⟕ \not\equiv\Big(\uparrow\,^{(\texttt{b2})}_{(\texttt{a2})}[\texttt{r}]\Big(\bigcirc_{(\texttt{a2})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching with LIMIT and optionally matching using a relationship that is already bound)



Incremental relational algebra tree (Matching with LIMIT and optionally matching using a relationship that is already bound)



A.19.57 Matching with LIMIT and optionally matching using a relationship and node that are both already bound

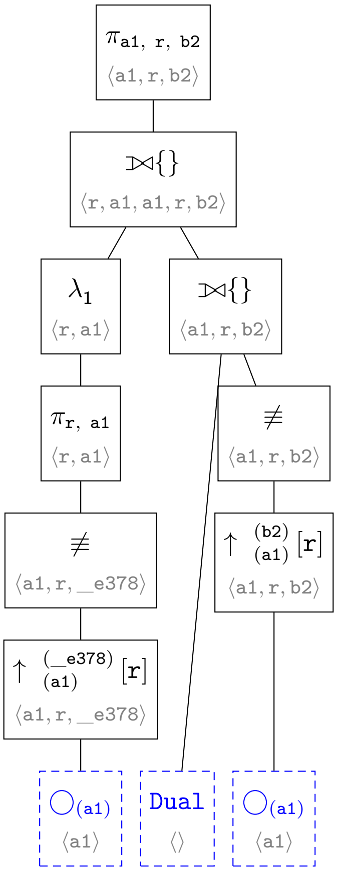
Query specification (Matching with LIMIT and optionally matching using a relationship and node that are both already bound)

```
MATCH (a1)-[r]->()
WITH r, a1
LIMIT 1
OPTIONAL MATCH (a1)-[r]->(b2)
RETURN a1, r, b2
```

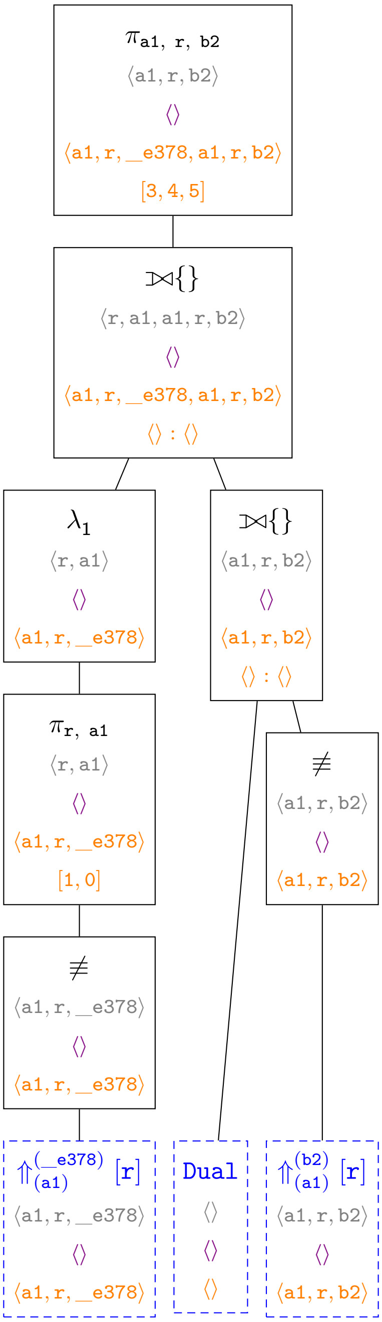
Relational algebra expression for search-based evaluation (Matching with LIMIT and optionally matching using a relationship and node that are both already bound)

$$r = \pi_{\texttt{a1, r, b2}}\Big(\lambda_1\Big(\pi_{\texttt{r, a1}}\Big(\not\equiv\Big(\uparrow\,{}^{(\texttt{\_e378})}_{(\texttt{a1})}\,[\texttt{r}]\Big(\bigcirc_{(\texttt{a1})}\Big)\Big)\Big)\Big)\Big) \mathbin{\unicode{x27D5}} \texttt{Dual} \mathbin{\unicode{x27D5}} \not\equiv\Big(\uparrow\,{}^{(\texttt{b2})}_{(\texttt{a1})}\,[\texttt{r}]\Big(\bigcirc_{(\texttt{a1})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching with LIMIT and optionally matching using a relationship and node that are both already bound)



Incremental relational algebra tree (Matching with LIMIT and optionally matching using a relationship and node that are both already bound)



A.19.58 Matching with LIMIT, then matching again using a relationship and node that are both already bound along with an additional predicate

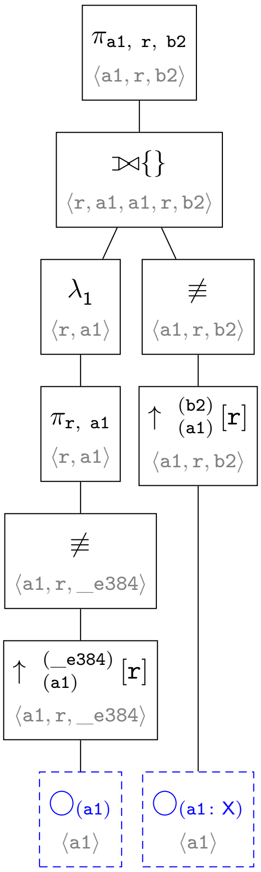
Query specification (Matching with LIMIT, then matching again using a relationship and node that are both already bound along with an additional predicate)

```
MATCH (a1)-[r]->()
WITH r, a1
LIMIT 1
MATCH (a1:X)-[r]->(b2)
RETURN a1, r, b2
```

Relational algebra expression for search-based evaluation (Matching with LIMIT, then matching again using a relationship and node that are both already bound along with an additional predicate)

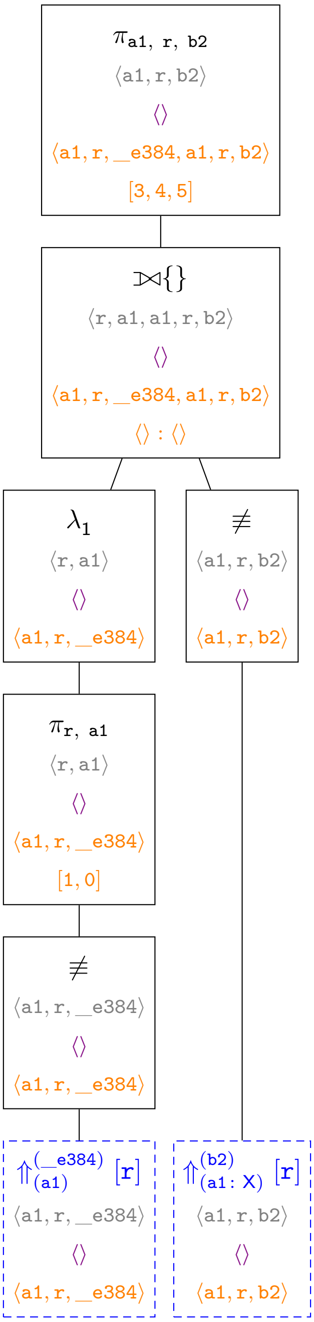
$$r = \pi_{\texttt{a1, r, b2}}\Big(\lambda_1\Big(\pi_{\texttt{r, a1}}\Big(\not\equiv\Big(\uparrow\,^{(\texttt{\_e384})}_{(\texttt{a1})}\,[\texttt{r}]\Big(\bigcirc_{(\texttt{a1})}\Big)\Big)\Big)\Big) ⟕ \not\equiv\Big(\uparrow\,^{(\texttt{b2})}_{(\texttt{a1})}\,[\texttt{r}]\Big(\bigcirc_{(\texttt{a1: X})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching with LIMIT, then matching again using a relationship and node that are both already bound along with an additional predicate)



$\pi_{\texttt{a1, r, b2}}$
⟨a1, r, b2⟩
⟕{}
⟨r, a1, a1, r, b2⟩
λ_1
⟨r, a1⟩
$\not\equiv$
⟨a1, r, b2⟩
$\pi_{\texttt{r, a1}}$
⟨r, a1⟩
$\uparrow\,^{(\texttt{b2})}_{(\texttt{a1})}\,[\texttt{r}]$
⟨a1, r, b2⟩
$\not\equiv$
⟨a1, r, _e384⟩
$\uparrow\,^{(\texttt{\_e384})}_{(\texttt{a1})}\,[\texttt{r}]$
⟨a1, r, _e384⟩
$\bigcirc_{(\texttt{a1})}$
⟨a1⟩
$\bigcirc_{(\texttt{a1: X})}$
⟨a1⟩

Incremental relational algebra tree (Matching with LIMIT, then matching again using a relationship and node that are both already bound along with an additional predicate)



A.19.59 Matching with LIMIT and predicates, then matching again using a relationship and node that are both already bound along with a duplicate predicate

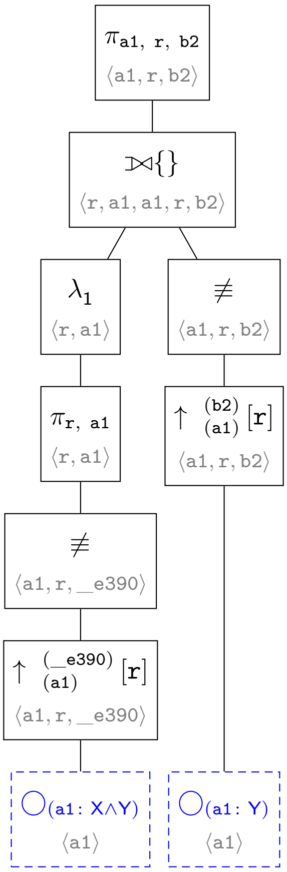
Query specification (Matching with LIMIT and predicates, then matching again using a relationship and node that are both already bound along with a duplicate predicate)

```
MATCH (a1:X:Y)-[r]->()
WITH r, a1
LIMIT 1
MATCH (a1:Y)-[r]->(b2)
RETURN a1, r, b2
```

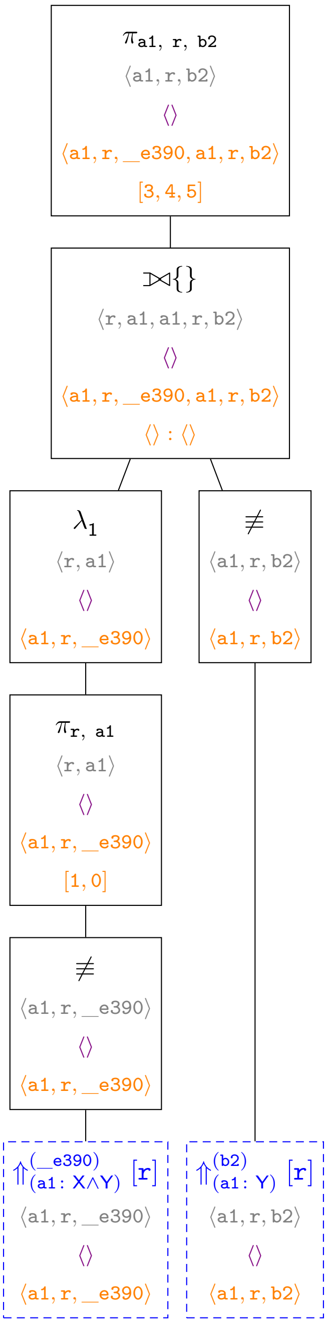
Relational algebra expression for search-based evaluation (Matching with LIMIT and predicates, then matching again using a relationship and node that are both already bound along with a duplicate predicate)

$$r = \pi_{\texttt{a1, r, b2}}\Big(\lambda_1\Big(\pi_{\texttt{r, a1}}\Big(\not\equiv\Big(\uparrow\,{}^{(\texttt{\_e390})}_{(\texttt{a1})}\,[\texttt{r}]\Big(\bigcirc_{(\texttt{a1}:\,\texttt{X}\wedge\texttt{Y})}\Big)\Big)\Big)\Big) ⟕ \not\equiv\Big(\uparrow\,{}^{(\texttt{b2})}_{(\texttt{a1})}\,[\texttt{r}]\Big(\bigcirc_{(\texttt{a1}:\,\texttt{Y})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching with LIMIT and predicates, then matching again using a relationship and node that are both already bound along with a duplicate predicate)



Incremental relational algebra tree (Matching with LIMIT and predicates, then matching again using a relationship and node that are both already bound along with a duplicate predicate)



A.19.60 Matching twice with conflicting relationship types on same relationship

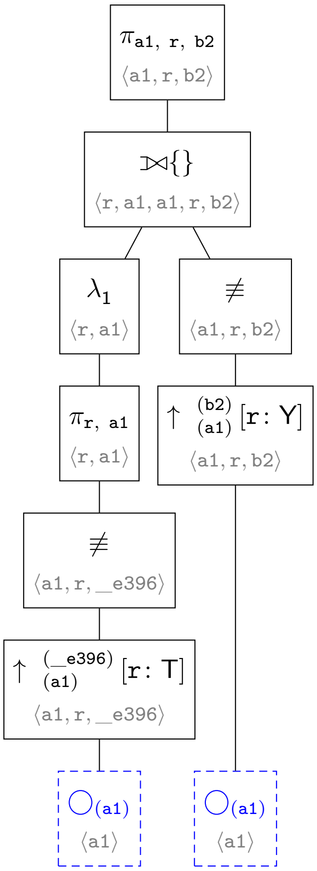
Query specification (Matching twice with conflicting relationship types on same relationship)

```
MATCH (a1)-[r:T]->()
WITH r, a1
LIMIT 1
MATCH (a1)-[r:Y]->(b2)
RETURN a1, r, b2
```

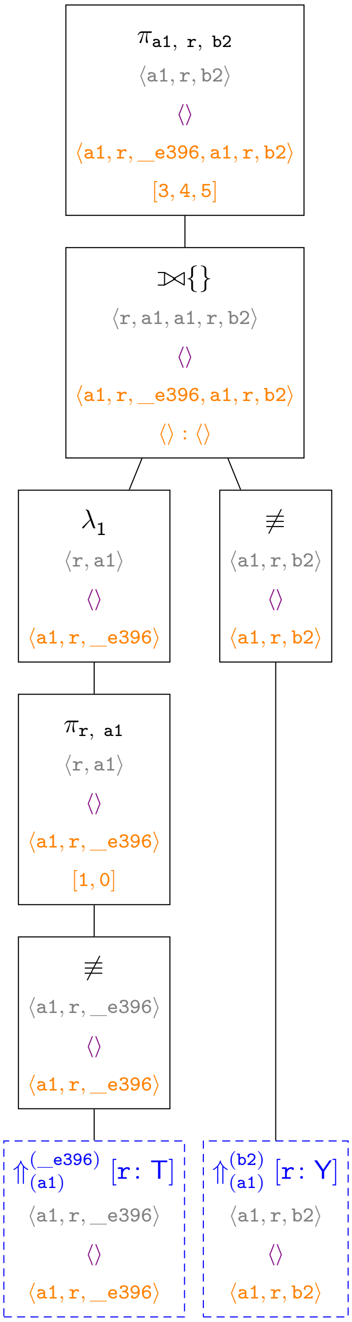
Relational algebra expression for search-based evaluation (Matching twice with conflicting relationship types on same relationship)

$$r = \pi_{\texttt{a1, r, b2}}\Big(\lambda_1\Big(\pi_{\texttt{r, a1}}\Big(\not\equiv\Big(\uparrow^{(\_\text{e396})}_{(\texttt{a1})}[\texttt{r}\colon \mathsf{T}]\Big(\bigcirc_{(\texttt{a1})}\Big)\Big)\Big)\Big)\mathbin{⟕}\not\equiv\Big(\uparrow^{(\texttt{b2})}_{(\texttt{a1})}[\texttt{r}\colon \mathsf{Y}]\Big(\bigcirc_{(\texttt{a1})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching twice with conflicting relationship types on same relationship)



Incremental relational algebra tree (Matching twice with conflicting relationship types on same relationship)



A.19.61 Matching twice with duplicate relationship types on same relationship

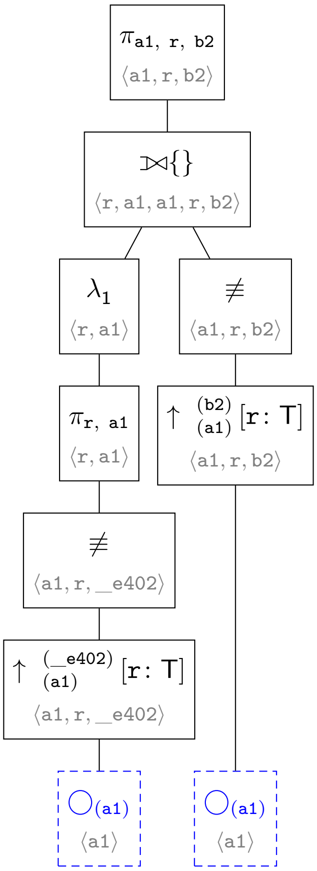
Query specification (Matching twice with duplicate relationship types on same relationship)

```
MATCH (a1)-[r:T]->() WITH r, a1
LIMIT 1
MATCH (a1)-[r:T]->(b2)
RETURN a1, r, b2
```

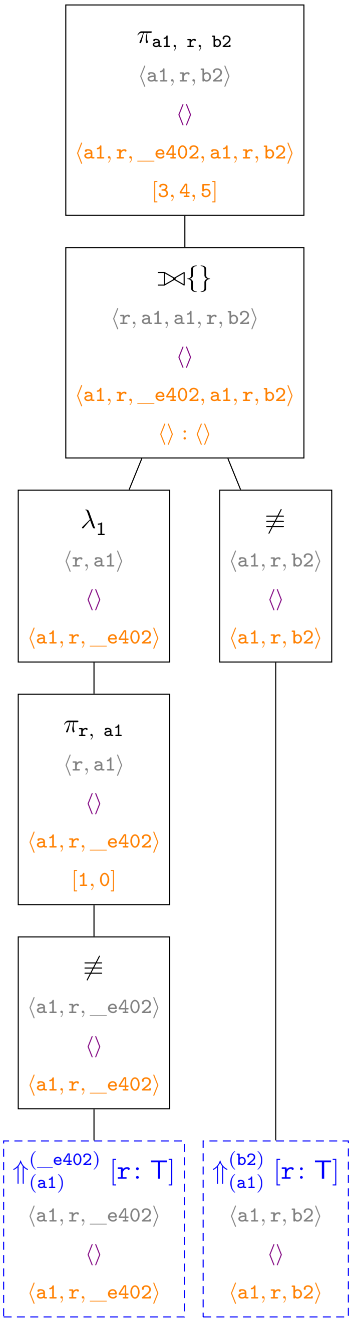
Relational algebra expression for search-based evaluation (Matching twice with duplicate relationship types on same relationship)

$$r = \pi_{\texttt{a1, r, b2}}\Big(\lambda_1\Big(\pi_{\texttt{r, a1}}\Big(\not\equiv\Big(\uparrow\,{}^{(\texttt{\_e402})}_{(\texttt{a1})}\,[\texttt{r}\colon\mathsf{T}]\Big(\bigcirc_{(\texttt{a1})}\Big)\Big)\Big)\Big)\Big)\rtimes\!\!\!\ltimes \not\equiv\Big(\uparrow\,{}^{(\texttt{b2})}_{(\texttt{a1})}\,[\texttt{r}\colon\mathsf{T}]\Big(\bigcirc_{(\texttt{a1})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching twice with duplicate relationship types on same relationship)



Incremental relational algebra tree (Matching twice with duplicate relationship types on same relationship)



A.19.62 Matching relationships into a list and matching variable length using the list

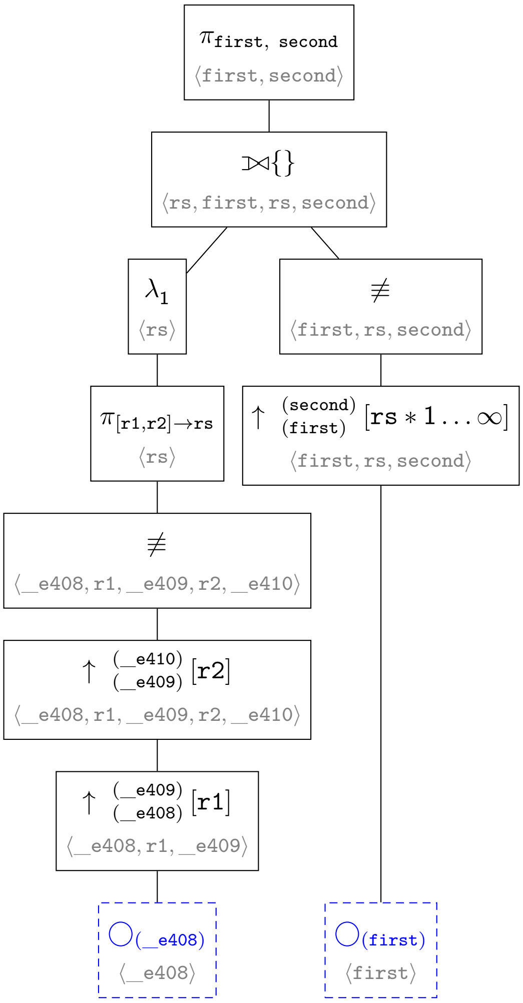
Query specification (Matching relationships into a list and matching variable length using the list)

```
MATCH ()-[r1]->()-[r2]->()
WITH [r1, r2] AS rs
LIMIT 1
MATCH (first)-[rs*]->(second)
RETURN first, second
```

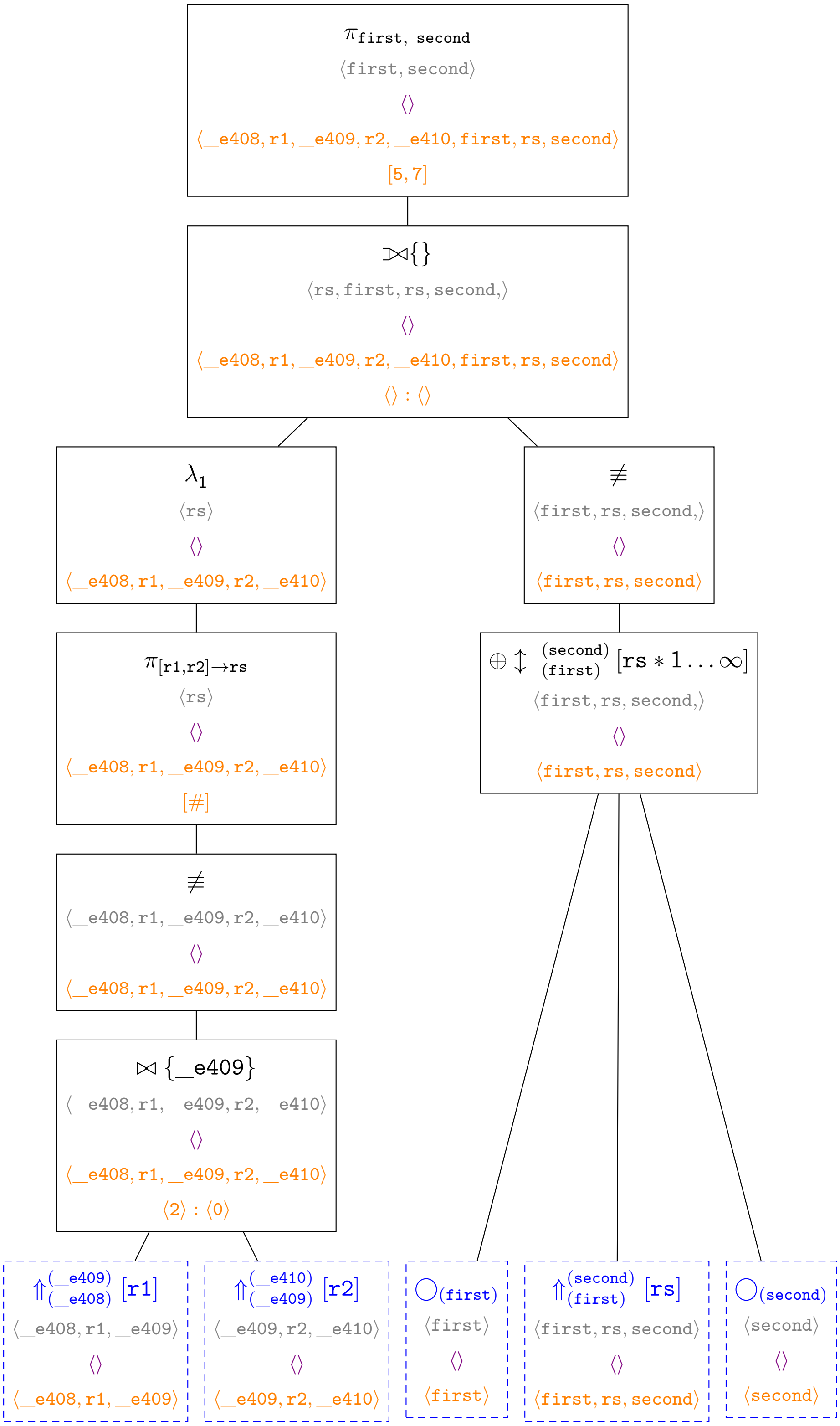
Relational algebra expression for search-based evaluation (Matching relationships into a list and matching variable length using the list)

$$r = \pi_{\texttt{first},\ \texttt{second}}\Bigg(\lambda_1\Bigg(\pi_{[\texttt{r1},\texttt{r2}]\rightarrow\texttt{rs}}\Bigg(\not\equiv\Bigg(\uparrow\,{}^{(\_\text{e410})}_{(\_\text{e409})}\,[\texttt{r2}]\Bigg(\uparrow\,{}^{(\_\text{e409})}_{(\_\text{e408})}\,[\texttt{r1}]\Bigg(\bigcirc_{(\_\text{e408})}\Bigg)\Bigg)\Bigg)\Bigg)\Bigg)\Bigg)\text{⟕}\not\equiv \\ \Bigg(\uparrow\,{}^{(\texttt{second})}_{(\texttt{first})}\,[\texttt{rs} * 1 \ldots \infty]\Bigg(\bigcirc_{(\texttt{first})}\Bigg)\Bigg)\Bigg)$$

Relational algebra tree for search-based evaluation (Matching relationships into a list and matching variable length using the list)


$\pi_{\texttt{first, second}}$
⟨first, second⟩
⟕{}
⟨rs, first, rs, second⟩
λ_1
⟨rs⟩
$\not\equiv$
⟨first, rs, second⟩
$\pi_{[\texttt{r1},\texttt{r2}]\rightarrow\texttt{rs}}$
⟨rs⟩
$\uparrow\,{}^{(\texttt{second})}_{(\texttt{first})}\,[\texttt{rs} * 1 \ldots \infty]$
⟨first, rs, second⟩
$\not\equiv$
⟨_e408, r1, _e409, r2, _e410⟩
$\uparrow\,{}^{(\_\text{e410})}_{(\_\text{e409})}\,[\texttt{r2}]$
⟨_e408, r1, _e409, r2, _e410⟩
$\uparrow\,{}^{(\_\text{e409})}_{(\_\text{e408})}\,[\texttt{r1}]$
⟨_e408, r1, _e409⟩
$\bigcirc_{(\_\text{e408})}$
⟨_e408⟩
$\bigcirc_{(\texttt{first})}$
⟨first⟩

Incremental relational algebra tree (Matching relationships into a list and matching variable length using the list)


$\pi_{\texttt{first, second}}$
⟨first, second⟩
⟨⟩
⟨_e408, r1, _e409, r2, _e410, first, rs, second⟩
[5, 7]
⋈{}
⟨rs, first, rs, second,⟩
⟨⟩
⟨_e408, r1, _e409, r2, _e410, first, rs, second⟩
⟨⟩ : ⟨⟩
λ_1
⟨rs⟩
⟨⟩
⟨_e408, r1, _e409, r2, _e410⟩
≢
⟨first, rs, second,⟩
⟨⟩
⟨first, rs, second⟩
$\pi_{[\texttt{r1,r2}] \to \texttt{rs}}$
⟨rs⟩
⟨⟩
⟨_e408, r1, _e409, r2, _e410⟩
[#]
$\oplus \updownarrow^{(\texttt{second})}_{(\texttt{first})} [\texttt{rs} * 1 \ldots \infty]$
⟨first, rs, second,⟩
⟨⟩
⟨first, rs, second⟩
≢
⟨_e408, r1, _e409, r2, _e410⟩
⟨⟩
⟨_e408, r1, _e409, r2, _e410⟩
⋈ {_e409}
⟨_e408, r1, _e409, r2, _e410⟩
⟨⟩
⟨_e408, r1, _e409, r2, _e410⟩
⟨2⟩ : ⟨0⟩
$\Uparrow^{(\_e409)}_{(\_e408)} [\texttt{r1}]$
⟨_e408, r1, _e409⟩
⟨⟩
⟨_e408, r1, _e409⟩
$\Uparrow^{(\_e410)}_{(\_e409)} [\texttt{r2}]$
⟨_e409, r2, _e410⟩
⟨⟩
⟨_e409, r2, _e410⟩
$\bigcirc_{(\texttt{first})}$
⟨first⟩
⟨⟩
⟨first⟩
$\Uparrow^{(\texttt{second})}_{(\texttt{first})} [\texttt{rs}]$
⟨first, rs, second⟩
⟨⟩
⟨first, rs, second⟩
$\bigcirc_{(\texttt{second})}$
⟨second⟩
⟨⟩
⟨second⟩

A.19.63 Matching relationships into a list and matching variable length using the list, with bound nodes

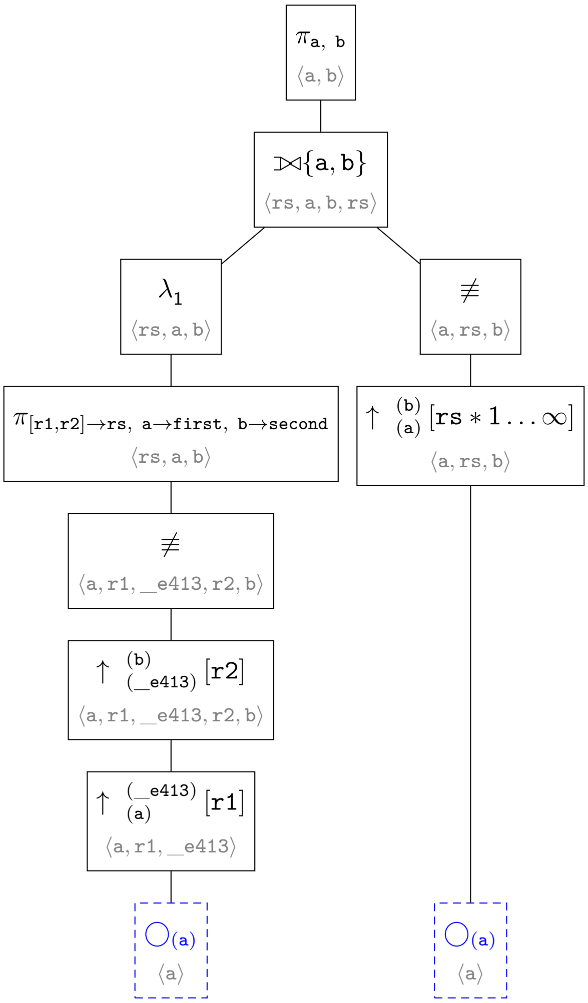
Query specification (Matching relationships into a list and matching variable length using the list, with bound nodes)

```
MATCH (a)-[r1]->()-[r2]->(b)
WITH [r1, r2] AS rs, a AS first, b AS second
LIMIT 1
MATCH (first)-[rs*]->(second)
RETURN first, second
```

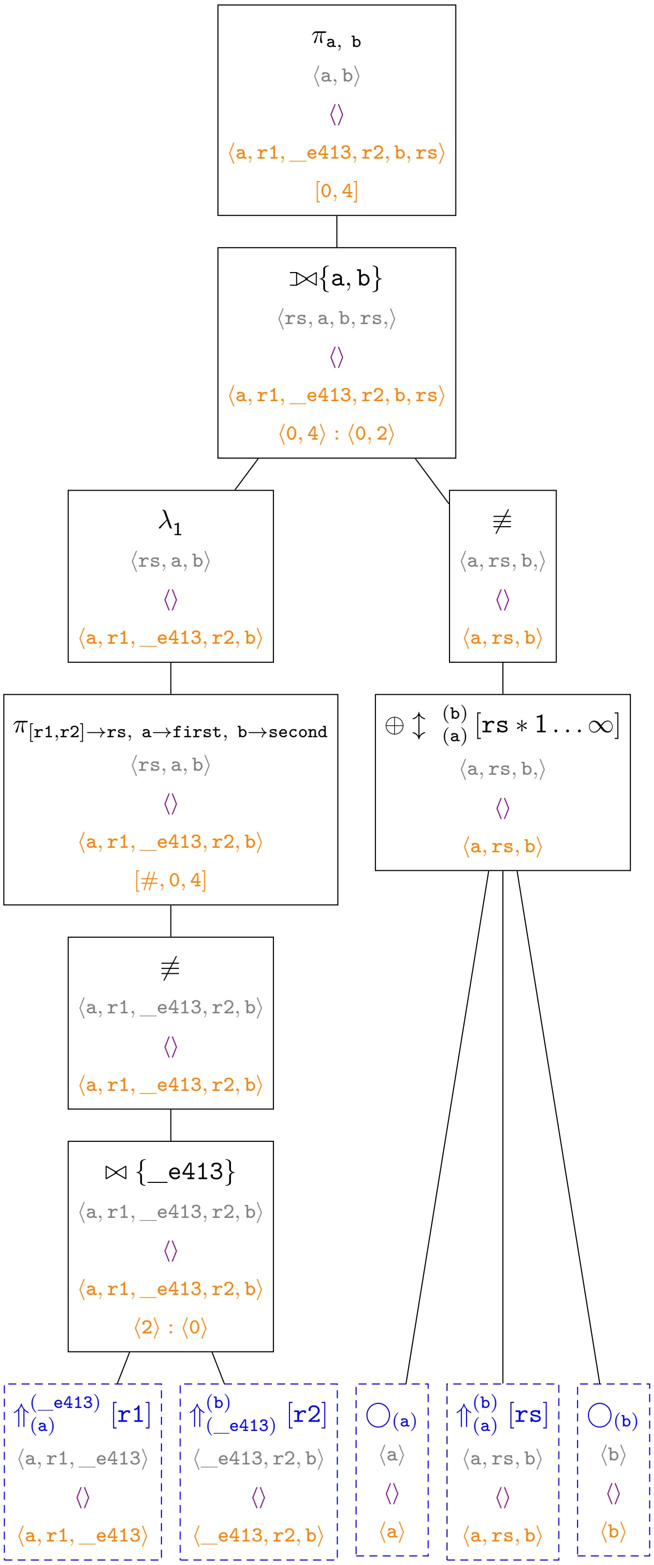
Relational algebra expression for search-based evaluation (Matching relationships into a list and matching variable length using the list, with bound nodes)

$$r = \pi_{\texttt{a, b}}\Bigg(\lambda_1\Bigg(\pi_{[\texttt{r1},\texttt{r2}]\rightarrow\texttt{rs, a}\rightarrow\texttt{first, b}\rightarrow\texttt{second}}\Bigg(\not\equiv\Bigg(\uparrow\,^{(\texttt{b})}_{(\texttt{\_e413})}\,[\texttt{r2}]\Bigg(\uparrow\,^{(\texttt{\_e413})}_{(\texttt{a})}\,[\texttt{r1}]\Bigg(\bigcirc_{(\texttt{a})}\Bigg)\Bigg)\Bigg)\Bigg)\Bigg)\text{⟕}\not\equiv\Bigg(\uparrow\,^{(\texttt{b})}_{(\texttt{a})}\,[\texttt{rs} * 1 \ldots \infty]\Bigg(\bigcirc_{(\texttt{a})}\Bigg)\Bigg)\Bigg)$$

Relational algebra tree for search-based evaluation (Matching relationships into a list and matching variable length using the list, with bound nodes)


$\pi_{\texttt{a, b}}$
⟨a, b⟩
⟕{a, b}
⟨rs, a, b, rs⟩
λ_1
⟨rs, a, b⟩
$\not\equiv$
⟨a, rs, b⟩
$\pi_{[\texttt{r1},\texttt{r2}]\rightarrow\texttt{rs, a}\rightarrow\texttt{first, b}\rightarrow\texttt{second}}$
⟨rs, a, b⟩
$\uparrow\,^{(\texttt{b})}_{(\texttt{a})}\,[\texttt{rs} * 1 \ldots \infty]$
⟨a, rs, b⟩
$\not\equiv$
⟨a, r1, _e413, r2, b⟩
$\uparrow\,^{(\texttt{b})}_{(\texttt{\_e413})}\,[\texttt{r2}]$
⟨a, r1, _e413, r2, b⟩
$\uparrow\,^{(\texttt{\_e413})}_{(\texttt{a})}\,[\texttt{r1}]$
⟨a, r1, _e413⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩

Incremental relational algebra tree (Matching relationships into a list and matching variable length using the list, with bound nodes)



A.19.64 Matching relationships into a list and matching variable length using the list, with bound nodes, wrong direction

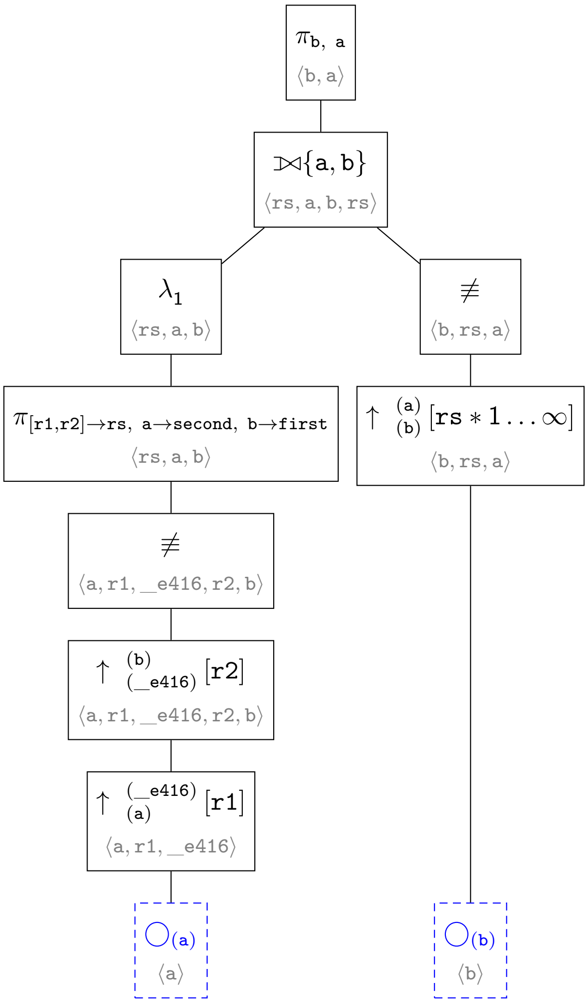
Query specification (Matching relationships into a list and matching variable length using the list, with bound nodes, wrong direction)

```
MATCH (a)-[r1]->()-[r2]->(b)
WITH [r1, r2] AS rs, a AS second, b AS first
LIMIT 1
MATCH (first)-[rs*]->(second)
RETURN first, second
```

Relational algebra expression for search-based evaluation (Matching relationships into a list and matching variable length using the list, with bound nodes, wrong direction)

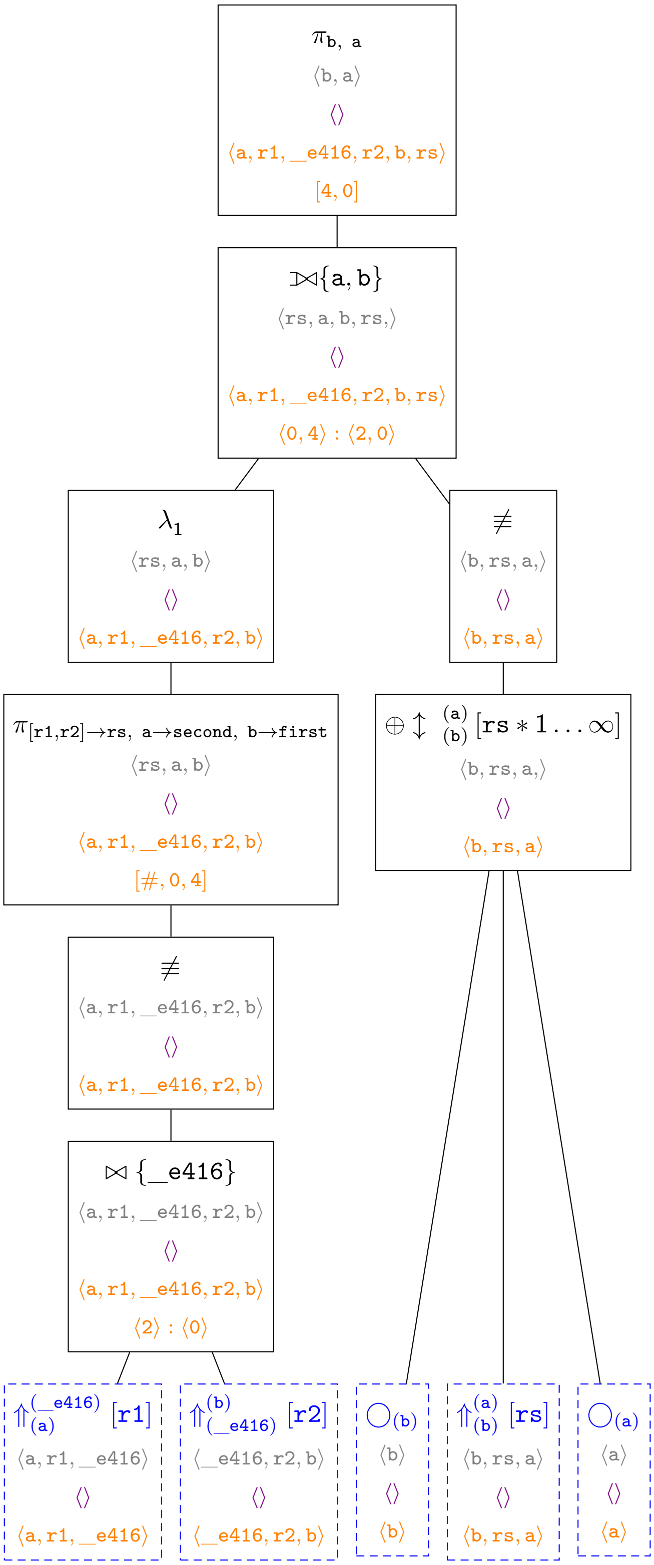
$$r = \pi_{\texttt{b, a}}\Big(\lambda_1\Big(\pi_{\texttt{[r1,r2]}\rightarrow\texttt{rs, a}\rightarrow\texttt{second, b}\rightarrow\texttt{first}}\Big(\not\equiv\Big(\uparrow\,^{(\texttt{b})}_{(\texttt{\_e416})}\,[\texttt{r2}]\Big(\uparrow\,^{(\texttt{\_e416})}_{(\texttt{a})}\,[\texttt{r1}]\Big(\bigcirc_{(\texttt{a})}\Big)\Big)\Big)\Big)\Big) ⟕ \not\equiv \Big(\uparrow\,^{(\texttt{a})}_{(\texttt{b})}\,[\texttt{rs} * 1\ldots\infty]\Big(\bigcirc_{(\texttt{b})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching relationships into a list and matching variable length using the list, with bound nodes, wrong direction)



$\pi_{\texttt{b, a}}$
⟨b, a⟩
⟕{a, b}
⟨rs, a, b, rs⟩
λ_1
⟨rs, a, b⟩
$\not\equiv$
⟨b, rs, a⟩
$\pi_{\texttt{[r1,r2]}\rightarrow\texttt{rs, a}\rightarrow\texttt{second, b}\rightarrow\texttt{first}}$
⟨rs, a, b⟩
$\uparrow\,^{(\texttt{a})}_{(\texttt{b})}\,[\texttt{rs} * 1\ldots\infty]$
⟨b, rs, a⟩
$\not\equiv$
⟨a, r1, _e416, r2, b⟩
$\uparrow\,^{(\texttt{b})}_{(\texttt{\_e416})}\,[\texttt{r2}]$
⟨a, r1, _e416, r2, b⟩
$\uparrow\,^{(\texttt{\_e416})}_{(\texttt{a})}\,[\texttt{r1}]$
⟨a, r1, _e416⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩
$\bigcirc_{(\texttt{b})}$
⟨b⟩

Incremental relational algebra tree (Matching relationships into a list and matching variable length using the list, with bound nodes, wrong direction)



A.19.65 Matching and optionally matching with bound nodes in reverse direction

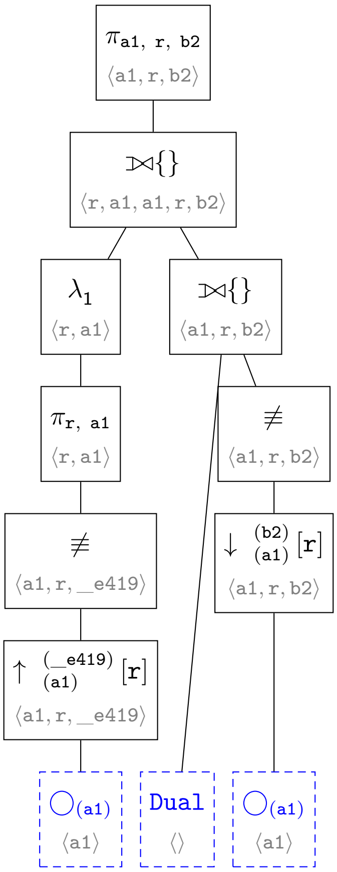
Query specification (Matching and optionally matching with bound nodes in reverse direction)

```
MATCH (a1)-[r]->()
WITH r, a1
LIMIT 1
OPTIONAL MATCH (a1)<-[r]-(b2)
RETURN a1, r, b2
```

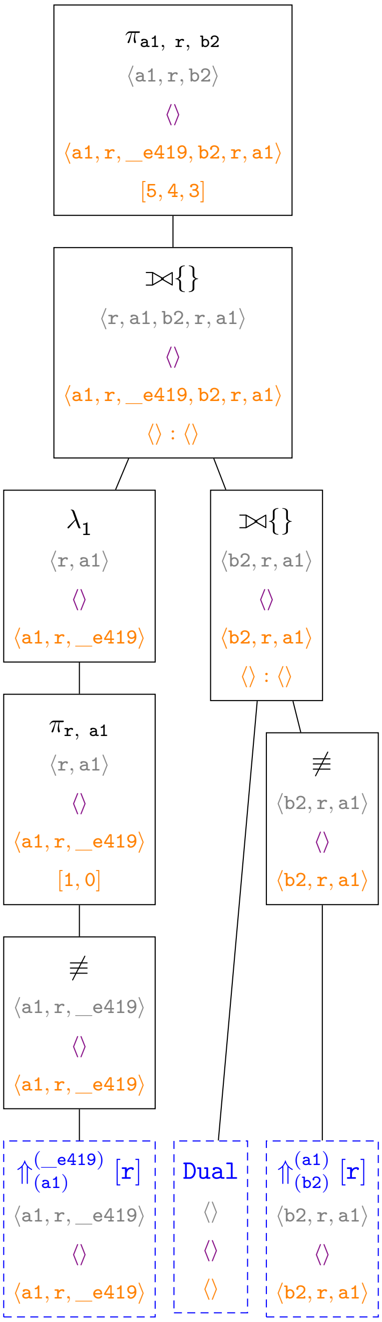
Relational algebra expression for search-based evaluation (Matching and optionally matching with bound nodes in reverse direction)

$$r = \pi_{\texttt{a1, r, b2}}\Big(\lambda_1\Big(\pi_{\texttt{r, a1}}\Big(\not\equiv\Big(\uparrow{}^{(\_\texttt{e419})}_{(\texttt{a1})}[\texttt{r}]\Big(\bigcirc_{(\texttt{a1})}\Big)\Big)\Big)\Big) ⟕ \texttt{Dual} ⟕ \not\equiv\Big(\downarrow{}^{(\texttt{b2})}_{(\texttt{a1})}[\texttt{r}]\Big(\bigcirc_{(\texttt{a1})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching and optionally matching with bound nodes in reverse direction)



Incremental relational algebra tree (Matching and optionally matching with bound nodes in reverse direction)



A.19.66 Matching and optionally matching with unbound nodes and equality predicate in reverse direction

Query specification (Matching and optionally matching with unbound nodes and equality predicate in reverse direction)

```
MATCH (a1)-[r]->()
WITH r, a1
LIMIT 1
OPTIONAL MATCH (a2)<-[r]-(b2)
WHERE a1 = a2
RETURN a1, r, b2, a2
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.67 Matching and returning ordered results, with LIMIT

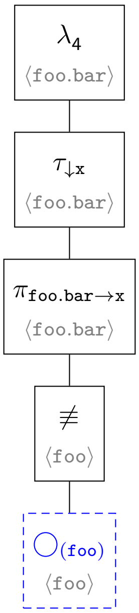
Query specification (Matching and returning ordered results, with LIMIT)

```
MATCH (foo)
RETURN foo.bar AS x
ORDER BY x DESC
LIMIT 4
```

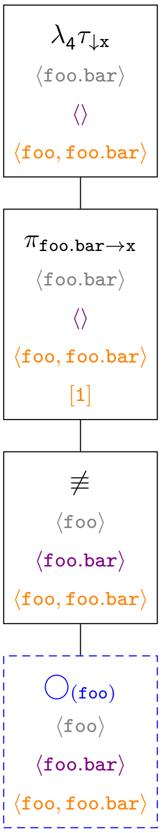
Relational algebra expression for search-based evaluation (Matching and returning ordered results, with LIMIT)

$$r = \lambda_4\Big(\tau_{\downarrow \mathtt{x}}\Big(\pi_{\mathtt{foo.bar}\rightarrow\mathtt{x}}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{foo})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching and returning ordered results, with LIMIT)


λ_4
⟨foo.bar⟩
$\tau_{\downarrow \mathtt{x}}$
⟨foo.bar⟩
$\pi_{\mathtt{foo.bar}\rightarrow\mathtt{x}}$
⟨foo.bar⟩
$\not\equiv$
⟨foo⟩
$\bigcirc_{(\mathtt{foo})}$
⟨foo⟩

Incremental relational algebra tree (Matching and returning ordered results, with LIMIT)



A.19.68 Counting an empty graph

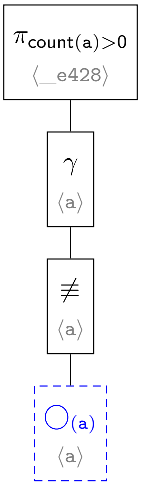
Query specification (Counting an empty graph)

```
MATCH (a)
RETURN count(a) > 0
```

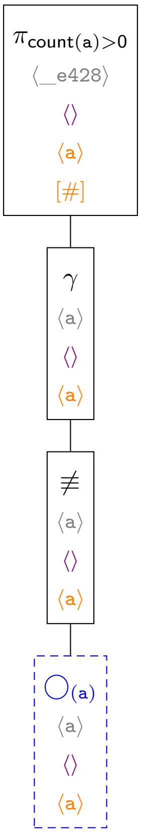
Relational algebra expression for search-based evaluation (Counting an empty graph)

$$r = \pi_{\mathsf{count(a)>0}}\Big(\gamma\Big(\not\equiv\Big(\bigcirc_{\mathsf{(a)}}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Counting an empty graph)



Incremental relational algebra tree (Counting an empty graph)



A.19.69 Matching variable length pattern with property predicate

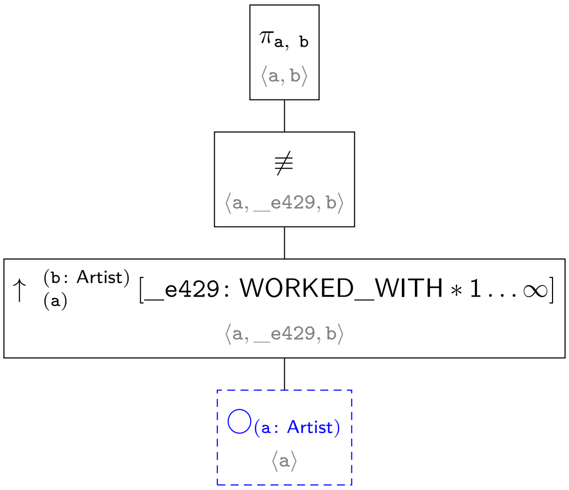
Query specification (Matching variable length pattern with property predicate)

```
MATCH (a:Artist)-[:WORKED_WITH* {year: 1988}]->(b:Artist)
RETURN *
```

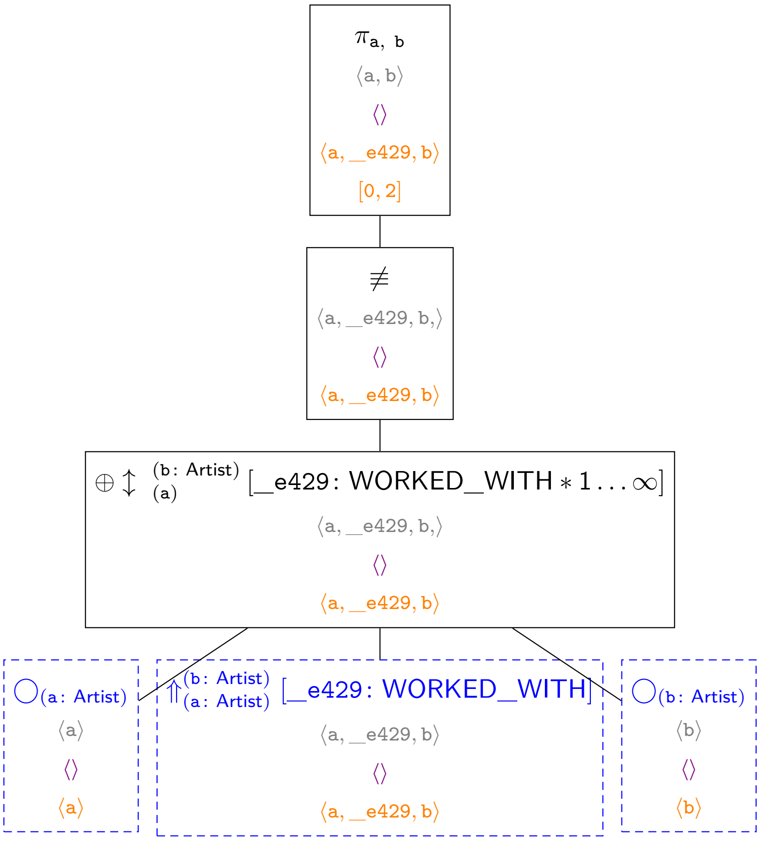
Relational algebra expression for search-based evaluation (Matching variable length pattern with property predicate)

$$r = \pi_{\text{a, b}}\left(\not\equiv\left(\uparrow^{(\text{b: Artist})}_{(\text{a})}[\text{\_e429: WORKED\_WITH} * 1\ldots\infty]\left(\bigcirc_{(\text{a: Artist})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Matching variable length pattern with property predicate)



Incremental relational algebra tree (Matching variable length pattern with property predicate)



A.19.70 Variable length pattern checking labels on endnodes

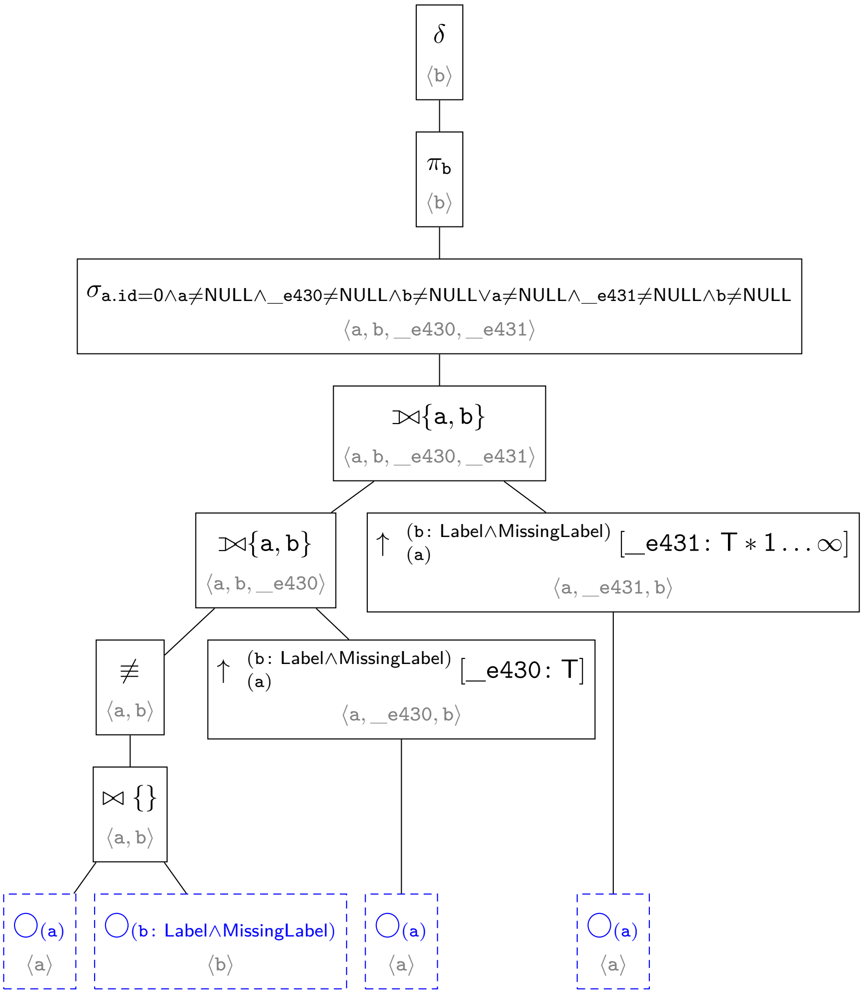
Query specification (Variable length pattern checking labels on endnodes)

```
MATCH (a), (b)
WHERE a.id = 0
AND (a)-[:T]->(b:Label)
OR (a)-[:T*]->(b:MissingLabel)
RETURN DISTINCT b
```

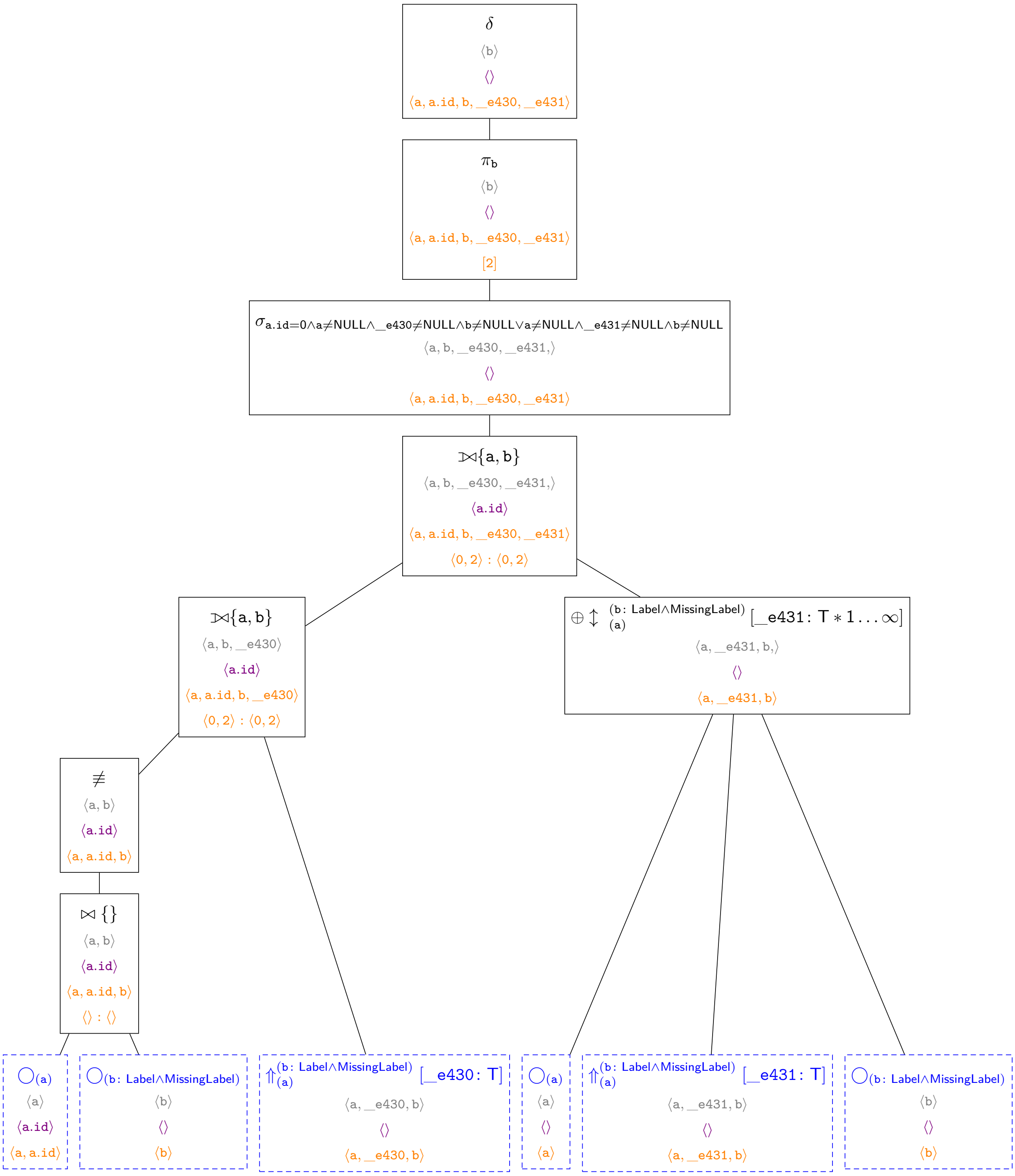
Relational algebra expression for search-based evaluation (Variable length pattern checking labels on endnodes)

$$r = \delta\Big(\pi_{\mathtt{b}}\Big(\sigma_{\mathtt{a.id}=0\wedge\mathtt{a}\neq\mathsf{NULL}\wedge\_\mathsf{e430}\neq\mathsf{NULL}\wedge\mathtt{b}\neq\mathsf{NULL}\vee\mathtt{a}\neq\mathsf{NULL}\wedge\_\mathsf{e431}\neq\mathsf{NULL}\wedge\mathtt{b}\neq\mathsf{NULL}}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{a})} \bowtie \bigcirc_{(\mathtt{b}:\ \mathsf{Label}\wedge\mathsf{MissingLabel})}\Big) ⟕ \uparrow^{(\mathtt{b}:\ \mathsf{Label}\wedge\mathsf{MissingLabel})}_{(\mathtt{a})} [\_\mathsf{e430}\colon \mathsf{T}]\Big(\bigcirc_{(\mathtt{a})}\Big) ⟕ \uparrow^{(\mathtt{b}:\ \mathsf{Label}\wedge\mathsf{MissingLabel})}_{(\mathtt{a})} [\_\mathsf{e431}\colon \mathsf{T} * 1\ldots\infty]\Big(\bigcirc_{(\mathtt{a})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Variable length pattern checking labels on endnodes)



Incremental relational algebra tree (Variable length pattern checking labels on endnodes)



A.19.71 Variable length pattern with label predicate on both sides

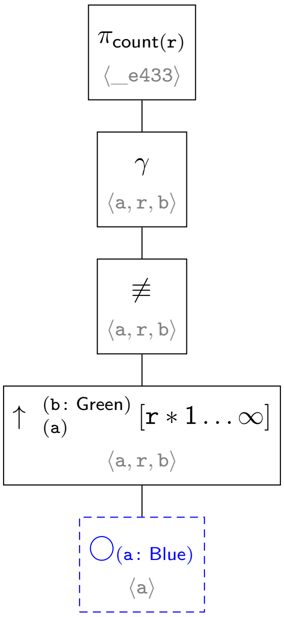
Query specification (Variable length pattern with label predicate on both sides)

```
MATCH (a:Blue)-[r*]->(b:Green)
RETURN count(r)
```

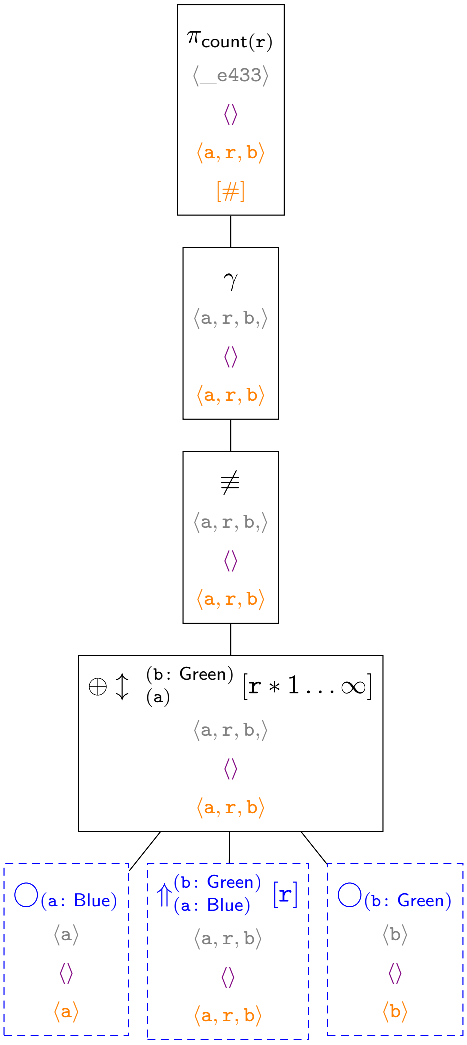
Relational algebra expression for search-based evaluation (Variable length pattern with label predicate on both sides)

$$r = \pi_{\mathsf{count}(\mathtt{r})}\Big(\gamma\Big(\not\equiv\Big(\uparrow\,^{(\mathtt{b}\colon\ \mathsf{Green})}_{(\mathtt{a})}\,[\mathtt{r} * 1 \ldots \infty]\Big(\bigcirc_{(\mathtt{a}\colon\ \mathsf{Blue})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Variable length pattern with label predicate on both sides)


$\pi_{\mathsf{count}(\mathtt{r})}$
⟨_e433⟩
γ
⟨a, r, b⟩
$\not\equiv$
⟨a, r, b⟩
$\uparrow\,^{(\mathtt{b}\colon\ \mathsf{Green})}_{(\mathtt{a})}\,[\mathtt{r} * 1 \ldots \infty]$
⟨a, r, b⟩
$\bigcirc_{(\mathtt{a}\colon\ \mathsf{Blue})}$
⟨a⟩

Incremental relational algebra tree (Variable length pattern with label predicate on both sides)



A.19.72 Undirected named path

Query specification (Undirected named path)

```
MATCH p = (n:Movie)--(m)
RETURN p
LIMIT 1
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.73 Named path with WITH

Query specification (Named path with WITH)

```
MATCH p = (a)
WITH p
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.74 Named path with alternating directed/undirected relationships

Query specification (Named path with alternating directed/undirected relationships)

```
MATCH p = (n)-->(m)--(o)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.75 Named path with multiple alternating directed/undirected relationships

Query specification (Named path with multiple alternating directed/undirected relationships)

```
MATCH path = (n)-->(m)--(o)--(p)
RETURN path
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.76 Named path with undirected fixed variable length pattern

Query specification (Named path with undirected fixed variable length pattern)

```
MATCH topRoute = (:Start)<-[:CONNECTED_TO]-()-[:CONNECTED_TO*3..3]-(:End)
RETURN topRoute
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.77 Returning a node property value

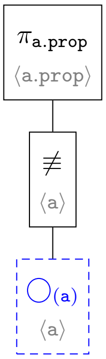
Query specification (Returning a node property value)

```
MATCH (a)
RETURN a.prop
```

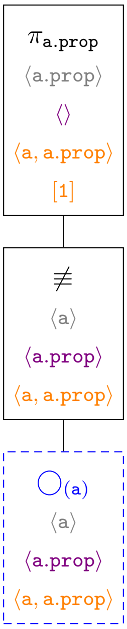
Relational algebra expression for search-based evaluation (Returning a node property value)

$$r = \pi_{\texttt{a.prop}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{a})}\Big)\Big)$$

Relational algebra tree for search-based evaluation (Returning a node property value)



Incremental relational algebra tree (Returning a node property value)



A.19.78 Returning a relationship property value

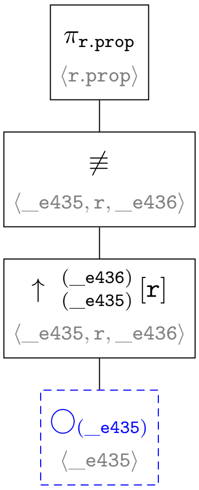
Query specification (Returning a relationship property value)

```
MATCH ()-[r]->()
RETURN r.prop
```

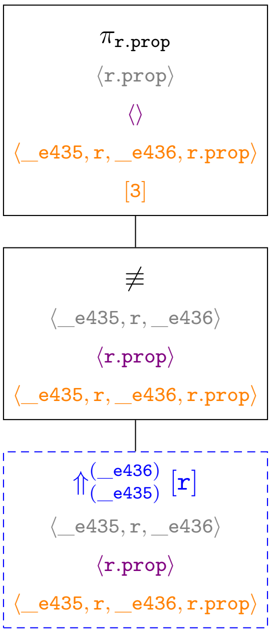
Relational algebra expression for search-based evaluation (Returning a relationship property value)

$$r = \pi_{\texttt{r.prop}}\left(\not\equiv\left(\uparrow\, ^{(\texttt{\_e436})}_{(\texttt{\_e435})}\, [\texttt{r}]\left(\bigcirc_{(\texttt{\_e435})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Returning a relationship property value)



Incremental relational algebra tree (Returning a relationship property value)


$\pi_{\texttt{r.prop}}$
⟨r.prop⟩
⟨⟩
⟨_e435, r, _e436, r.prop⟩
[3]
$\not\equiv$
⟨_e435, r, _e436⟩
⟨r.prop⟩
⟨_e435, r, _e436, r.prop⟩
$\Uparrow^{(\_e436)}_{(\_e435)}$ [r]
⟨_e435, r, _e436⟩
⟨r.prop⟩
⟨_e435, r, _e436, r.prop⟩

A.19.79 Projecting nodes and relationships

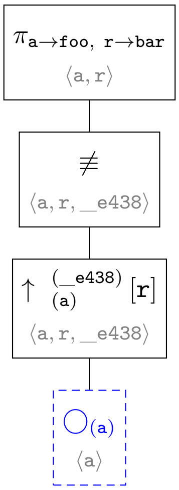
Query specification (Projecting nodes and relationships)

```
MATCH (a)-[r]->()
RETURN a AS foo, r AS bar
```

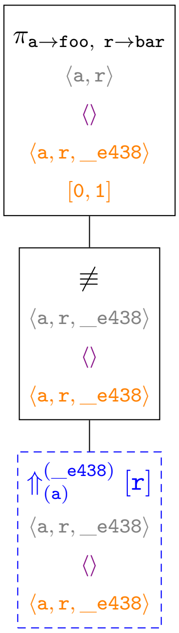
Relational algebra expression for search-based evaluation (Projecting nodes and relationships)

$$r = \pi_{\texttt{a}\rightarrow\texttt{foo},\ \texttt{r}\rightarrow\texttt{bar}}\left(\not\equiv\left(\uparrow^{(\_\texttt{e438})}_{(\texttt{a})}[\texttt{r}]\left(\bigcirc_{(\texttt{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Projecting nodes and relationships)



Incremental relational algebra tree (Projecting nodes and relationships)



A.19.80 Missing node property should become null

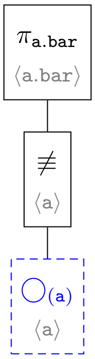
Query specification (Missing node property should become null)

```
MATCH (a)
RETURN a.bar
```

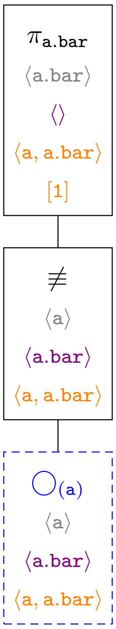
Relational algebra expression for search-based evaluation (Missing node property should become null)

$$r = \pi_{\texttt{a.bar}}\Big(\not\equiv\Big(\bigcirc_{\texttt{(a)}}\Big)\Big)$$

Relational algebra tree for search-based evaluation (Missing node property should become null)



Incremental relational algebra tree (Missing node property should become null)



A.19.81 Missing relationship property should become null

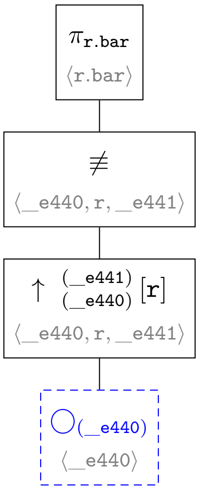
Query specification (Missing relationship property should become null)

```
MATCH ()-[r]->()
RETURN r.bar
```

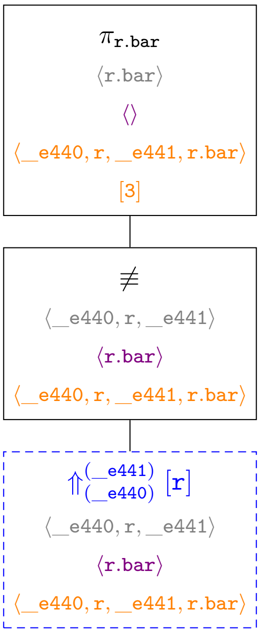
Relational algebra expression for search-based evaluation (Missing relationship property should become null)

$$r = \pi_{\texttt{r.bar}}\left(\not\equiv\left(\uparrow^{(\_\texttt{e441})}_{(\_\texttt{e440})}[\texttt{r}]\left(\bigcirc_{(\_\texttt{e440})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Missing relationship property should become null)



Incremental relational algebra tree (Missing relationship property should become null)



A.19.82 Returning multiple node property values

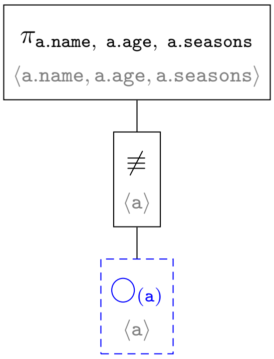
Query specification (Returning multiple node property values)

```
MATCH (a)
RETURN a.name, a.age, a.seasons
```

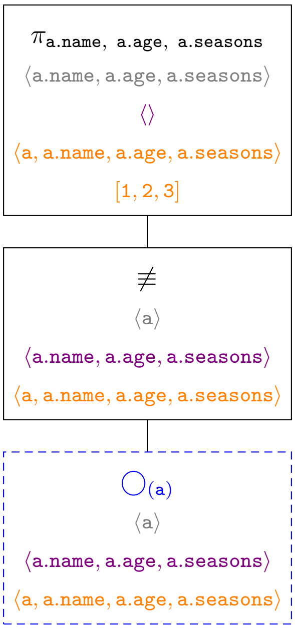
Relational algebra expression for search-based evaluation (Returning multiple node property values)

$$r = \pi_{\texttt{a.name, a.age, a.seasons}}\left(\not\equiv\left(\bigcirc_{\texttt{(a)}}\right)\right)$$

Relational algebra tree for search-based evaluation (Returning multiple node property values)



Incremental relational algebra tree (Returning multiple node property values)


π a.name, a.age, a.seasons
⟨a.name, a.age, a.seasons⟩
⟨⟩
⟨a, a.name, a.age, a.seasons⟩
[1, 2, 3]
≢
⟨a⟩
⟨a.name, a.age, a.seasons⟩
⟨a, a.name, a.age, a.seasons⟩
○ (a)
⟨a⟩
⟨a.name, a.age, a.seasons⟩
⟨a, a.name, a.age, a.seasons⟩

A.19.83 Adding a property and a literal in projection

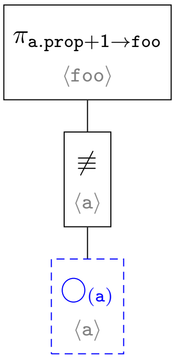
Query specification (Adding a property and a literal in projection)

```
MATCH (a)
RETURN a.prop + 1 AS foo
```

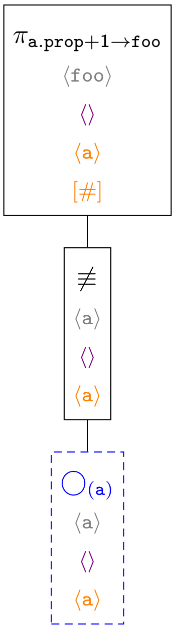
Relational algebra expression for search-based evaluation (Adding a property and a literal in projection)

$$r = \pi_{\texttt{a.prop+1}\rightarrow\texttt{foo}}\left(\not\equiv\left(\bigcirc_{(\texttt{a})}\right)\right)$$

Relational algebra tree for search-based evaluation (Adding a property and a literal in projection)


$\pi_{\texttt{a.prop+1}\rightarrow\texttt{foo}}$
⟨foo⟩
≢
⟨a⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩

Incremental relational algebra tree (Adding a property and a literal in projection)


$\pi_{\texttt{a.prop+1}\rightarrow\texttt{foo}}$
⟨foo⟩
⟨⟩
⟨a⟩
[#]
≢
⟨a⟩
⟨⟩
⟨a⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩
⟨⟩
⟨a⟩

A.19.84 Adding list properties in projection

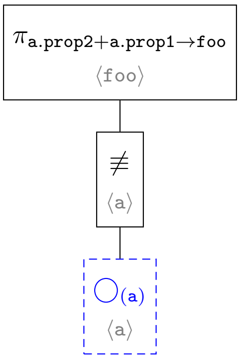
Query specification (Adding list properties in projection)

```
MATCH (a)
RETURN a.prop2 + a.prop1 AS foo
```

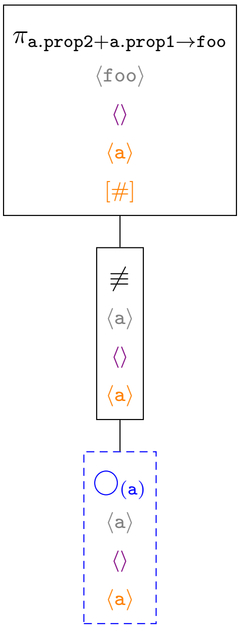
Relational algebra expression for search-based evaluation (Adding list properties in projection)

$$r = \pi_{\texttt{a.prop2+a.prop1}\rightarrow\texttt{foo}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{a})}\Big)\Big)$$

Relational algebra tree for search-based evaluation (Adding list properties in projection)



Incremental relational algebra tree (Adding list properties in projection)



A.19.85 Variable length relationship variables are lists of relationships

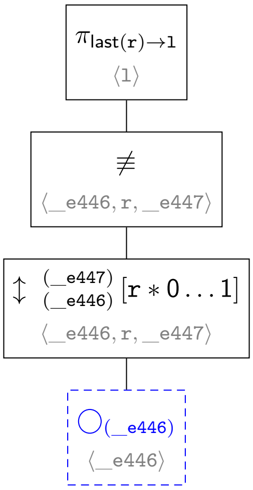
Query specification (Variable length relationship variables are lists of relationships)

```
MATCH ()-[r*0..1]-()
RETURN last(r) AS l
```

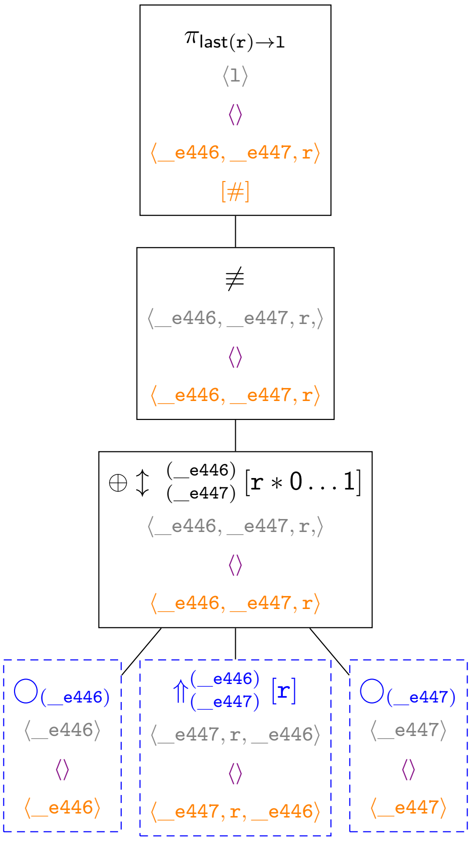
Relational algebra expression for search-based evaluation (Variable length relationship variables are lists of relationships)

$$r = \pi_{\mathsf{last(r)}\rightarrow \mathtt{l}}\left(\not\equiv\left(\updownarrow^{(\_\mathsf{e447})}_{(\_\mathsf{e446})}[\mathtt{r} * 0 \ldots 1]\left(\bigcirc_{(\_\mathsf{e446})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Variable length relationship variables are lists of relationships)


$\pi_{\text{last}(r)\rightarrow l}$
⟨l⟩
≢
⟨_e446, r, _e447⟩
↕ (_e447) (_e446) [r ∗ 0 . . . 1]
⟨_e446, r, _e447⟩
○(_e446)
⟨_e446⟩

Incremental relational algebra tree (Variable length relationship variables are lists of relationships)



A.19.86 Variable length patterns and nulls

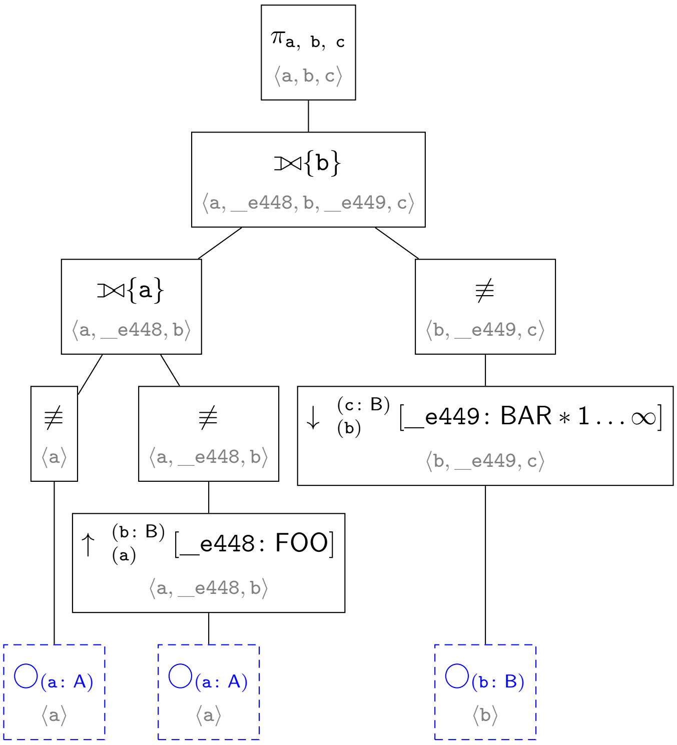
Query specification (Variable length patterns and nulls)

```
MATCH (a:A)
OPTIONAL MATCH (a)-[:FOO]->(b:B)
OPTIONAL MATCH (b)<-[:BAR*]-(c:B)
RETURN a, b, c
```

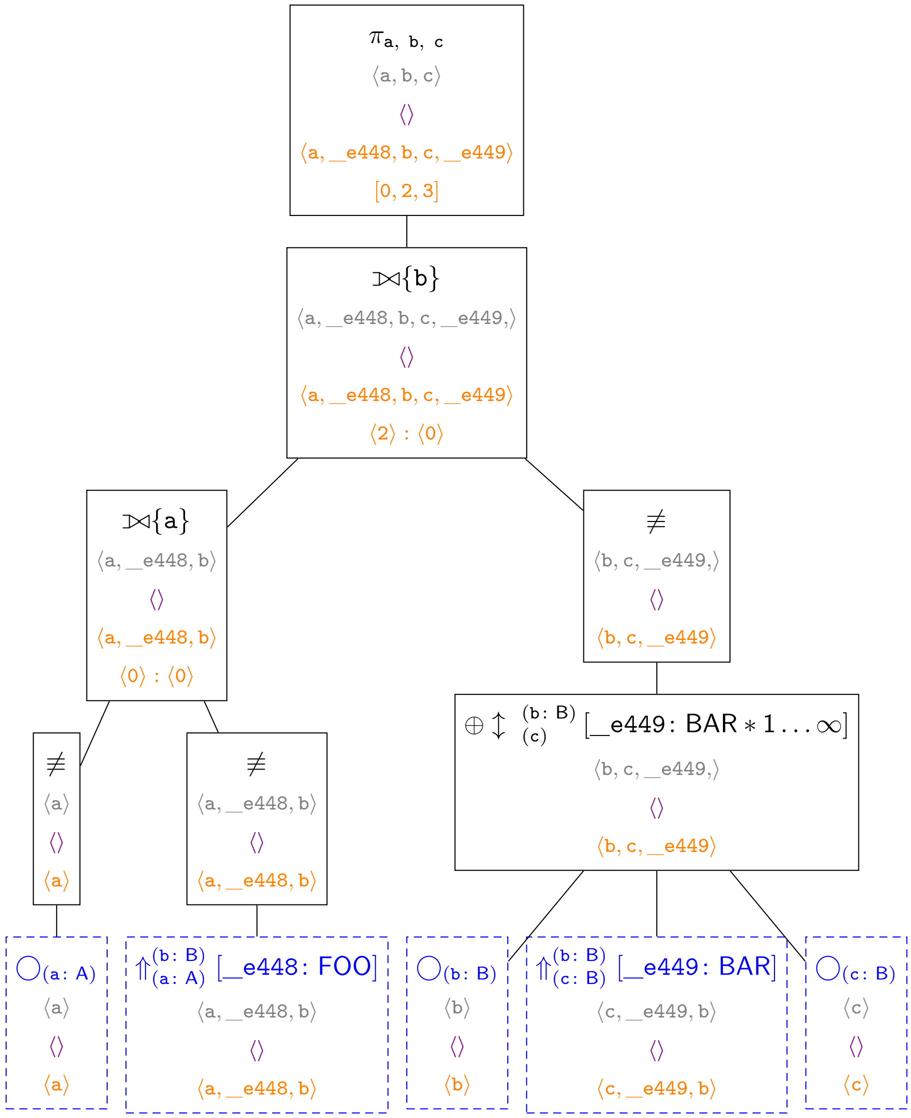
Relational algebra expression for search-based evaluation **(Variable length patterns and nulls)**

$$r = \pi_{\texttt{a, b, c}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{a: A})}\Big) ⟕ \not\equiv\Big(\uparrow^{(\texttt{b: B})}_{(\texttt{a})}[\_\mathsf{e448: FOO}]\Big(\bigcirc_{(\texttt{a: A})}\Big)\Big) ⟕ \not\equiv \Big(\downarrow^{(\texttt{c: B})}_{(\texttt{b})}[\_\mathsf{e449: BAR} * 1 \ldots \infty]\Big(\bigcirc_{(\texttt{b: B})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation **(Variable length patterns and nulls)**



Incremental relational algebra tree (Variable length patterns and nulls)



A.19.87 Projecting a list of nodes and relationships

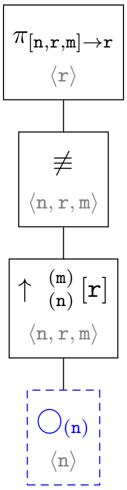
Query specification (Projecting a list of nodes and relationships)

```
MATCH (n)-[r]->(m)
RETURN [n, r, m] AS r
```

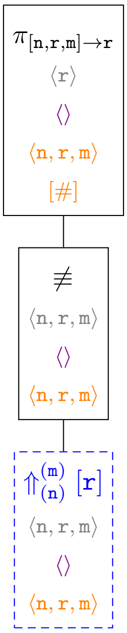
Relational algebra expression for search-based evaluation (Projecting a list of nodes and relationships)

$$r = \pi_{[\mathtt{n},\mathtt{r},\mathtt{m}]\rightarrow\mathtt{r}}\left(\not\equiv\left(\uparrow {}^{(\mathtt{m})}_{(\mathtt{n})} [\mathtt{r}]\left(\bigcirc_{(\mathtt{n})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Projecting a list of nodes and relationships)


$\pi_{\texttt{[n,r,m]}\rightarrow\texttt{r}}$
⟨r⟩
≢
⟨n, r, m⟩
$\uparrow\ {}^{\texttt{(m)}}_{\texttt{(n)}}\ [\texttt{r}]$
⟨n, r, m⟩
$\bigcirc_{\texttt{(n)}}$
⟨n⟩

Incremental relational algebra tree (Projecting a list of nodes and relationships)


$\pi_{\texttt{[n,r,m]}\rightarrow\texttt{r}}$
⟨r⟩
⟨⟩
⟨n, r, m⟩
[#]
≢
⟨n, r, m⟩
⟨⟩
⟨n, r, m⟩
$\Uparrow{}^{\texttt{(m)}}_{\texttt{(n)}}\ [\texttt{r}]$
⟨n, r, m⟩
⟨⟩
⟨n, r, m⟩

A.19.88 Projecting a map of nodes and relationships

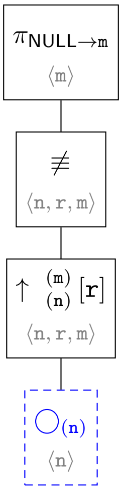
Query specification (Projecting a map of nodes and relationships)

```
MATCH (n)-[r]->(m)
RETURN {node1: n, rel: r, node2: m} AS m
```

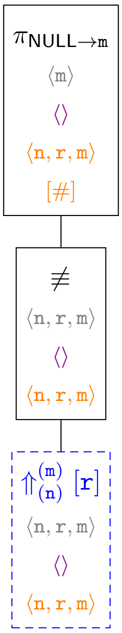
Relational algebra expression for search-based evaluation (Projecting a map of nodes and relationships)

$$r = \pi_{\textsf{NULL}\rightarrow\texttt{m}}\Big(\not\equiv\Big(\uparrow\ {}^{\texttt{(m)}}_{\texttt{(n)}}\ [\texttt{r}]\Big(\bigcirc_{\texttt{(n)}}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Projecting a map of nodes and relationships)



Incremental relational algebra tree (Projecting a map of nodes and relationships)



A.19.89 Respecting direction when matching existing path

Query specification (Respecting direction when matching existing path)

```
MATCH p = ({prop: 'a'})-->({prop: 'b'})
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.90 Respecting direction when matching non-existent path

Query specification (Respecting direction when matching non-existent path)

```
MATCH p = ({prop: 'a'})<--({prop: 'b'})
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.91 Respecting direction when matching non-existent path with multiple directions

Query specification (Respecting direction when matching non-existent path with multiple directions)

```
MATCH p = (n)-->(k)<--(n)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.92 Matching path with both directions should respect other directions

Query specification (Matching path with both directions should respect other directions)

```
MATCH p = (n)<-->(k)<--(n)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.93 Matching path with multiple bidirectional relationships

Query specification (Matching path with multiple bidirectional relationships)

```
MATCH p=(n)<-->(k)<-->(n)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.19.94 Matching nodes with many labels

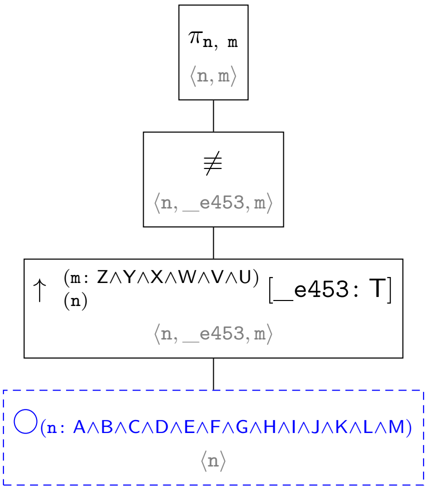
Query specification (Matching nodes with many labels)

```
MATCH (n:A:B:C:D:E:F:G:H:I:J:K:L:M)-[:T]->(m:Z:Y:X:W:V:U)
RETURN n, m
```

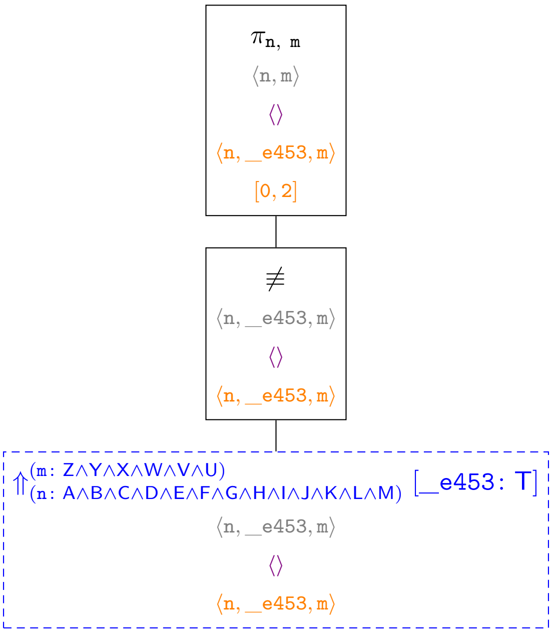
Relational algebra expression for search-based evaluation (Matching nodes with many labels)

$$r = \pi_{\mathtt{n},\ \mathtt{m}}\Big(\not\equiv \Big(\uparrow\ {}^{(\mathtt{m}:\ \mathsf{Z} \wedge \mathsf{Y} \wedge \mathsf{X} \wedge \mathsf{W} \wedge \mathsf{V} \wedge \mathsf{U})}_{(\mathtt{n})}\ [\_\mathsf{e453}\colon \mathsf{T}] \Big(\bigcirc_{(\mathtt{n}:\ \mathsf{A} \wedge \mathsf{B} \wedge \mathsf{C} \wedge \mathsf{D} \wedge \mathsf{E} \wedge \mathsf{F} \wedge \mathsf{G} \wedge \mathsf{H} \wedge \mathsf{I} \wedge \mathsf{J} \wedge \mathsf{K} \wedge \mathsf{L} \wedge \mathsf{M})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching nodes with many labels)


π n, m
⟨n, m⟩
≢
⟨n, _e453, m⟩
↑ (m: Z∧Y∧X∧W∧V∧U) (n) [_e453: T]
⟨n, _e453, m⟩
◯(n: A∧B∧C∧D∧E∧F∧G∧H∧I∧J∧K∧L∧M)
⟨n⟩

Incremental relational algebra tree (Matching nodes with many labels)



A.19.95 Matching longer variable length paths

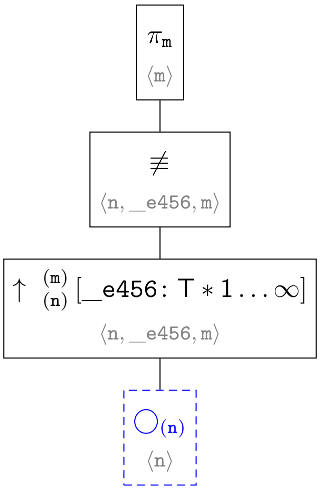
Query specification (Matching longer variable length paths)

```
MATCH (n {prop: 'start'})-[:T*]->(m {prop: 'end'})
RETURN m
```

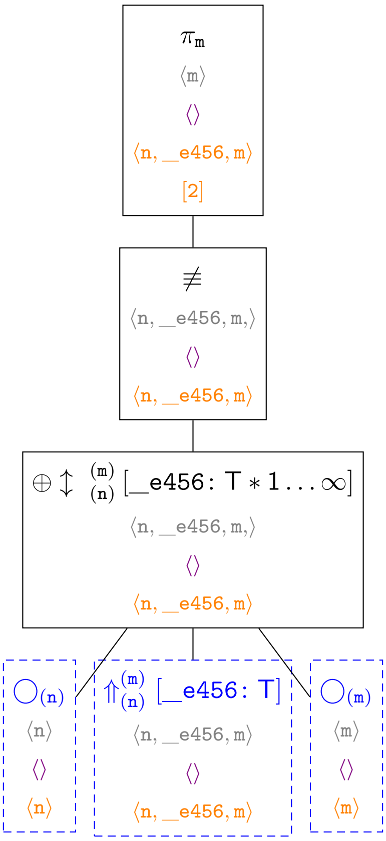
Relational algebra expression for search-based evaluation (Matching longer variable length paths)

$$r = \pi_{\mathtt{m}}\Big(\not\equiv \Big(\uparrow {}^{(\mathtt{m})}_{(\mathtt{n})} [\_e456\colon \mathsf{T} * 1 \ldots \infty] \Big(\bigcirc_{(\mathtt{n})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching longer variable length paths)



Incremental relational algebra tree (Matching longer variable length paths)



A.19.96 Matching a self-loop

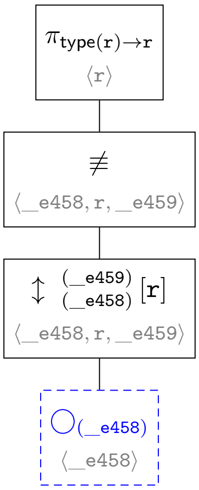
Query specification (Matching a self-loop)

```
MATCH ()-[r]-()
RETURN type(r) AS r
```

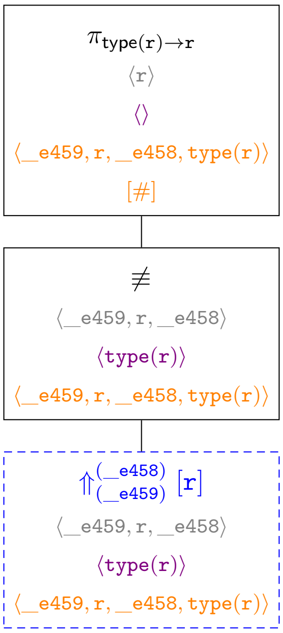
Relational algebra expression for search-based evaluation (Matching a self-loop)

$$r = \pi_{\mathsf{type(r)}\rightarrow\mathsf{r}}\left(\not\equiv \left(\updownarrow \,{}^{(\_\mathsf{e459})}_{(\_\mathsf{e458})} \,[\mathtt{r}] \left(\bigcirc_{(\_\mathsf{e458})} \right)\right)\right)$$

Relational algebra tree for search-based evaluation (Matching a self-loop)


$\pi_{\texttt{type(r)}\rightarrow\texttt{r}}$
⟨r⟩
$\not\equiv$
⟨_e458, r, _e459⟩
$\updownarrow^{(\texttt{\_e459})}_{(\texttt{\_e458})}$ [r]
⟨_e458, r, _e459⟩
$\bigcirc_{(\texttt{\_e458})}$
⟨_e458⟩

Incremental relational algebra tree (Matching a self-loop)


$\pi_{\texttt{type(r)}\rightarrow\texttt{r}}$
⟨r⟩
⟨⟩
⟨_e459, r, _e458, type(r)⟩
[#]
$\not\equiv$
⟨_e459, r, _e458⟩
⟨type(r)⟩
⟨_e459, r, _e458, type(r)⟩
$\Uparrow^{(\texttt{\_e458})}_{(\texttt{\_e459})}$ [r]
⟨_e459, r, _e458⟩
⟨type(r)⟩
⟨_e459, r, _e458, type(r)⟩

A.20 MatchingSelfRelationships

Progress:

19 of 19

A.20.1 Undirected match in self-relationship graph

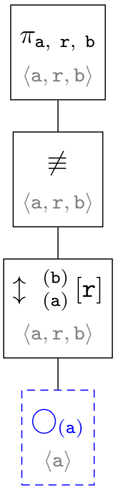
Query specification (Undirected match in self-relationship graph)

```
MATCH (a)-[r]-(b)
RETURN a, r, b
```

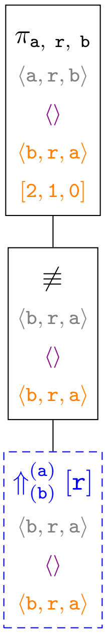
Relational algebra expression for search-based evaluation (Undirected match in self-relationship graph)

$$r = \pi_{\texttt{a, r, b}}\Big(\not\equiv\Big(\updownarrow^{(\texttt{b})}_{(\texttt{a})}[\texttt{r}]\Big(\bigcirc_{(\texttt{a})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Undirected match in self-relationship graph)



Incremental relational algebra tree (Undirected match in self-relationship graph)



A.20.2 Undirected match in self-relationship graph, count

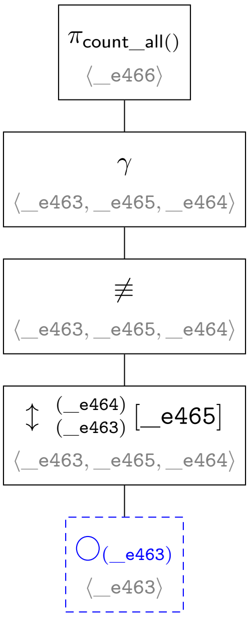
Query specification (Undirected match in self-relationship graph, count)

```
MATCH ()--()
RETURN count(*)
```

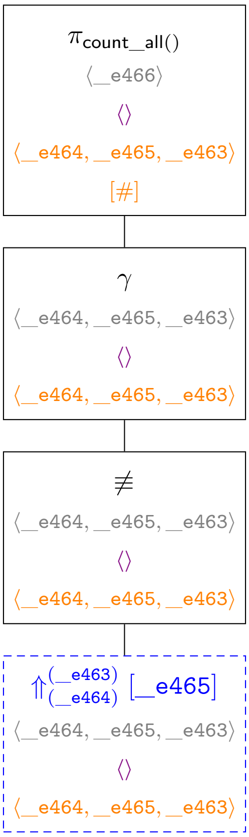
Relational algebra expression for search-based evaluation (Undirected match in self-relationship graph, count)

$$r = \pi_{\text{count\_all()}}\Big(\gamma\Big(\not\equiv\Big(\updownarrow^{(\_\text{e464})}_{(\_\text{e463})}[\_\text{e465}]\Big(\bigcirc_{(\_\text{e463})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Undirected match in self-relationship graph, count)



Incremental relational algebra tree (Undirected match in self-relationship graph, count)



A.20.3 Undirected match of self-relationship in self-relationship graph

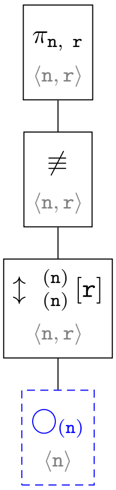
Query specification (Undirected match of self-relationship in self-relationship graph)

```
MATCH (n)-[r]-(n)
RETURN n, r
```

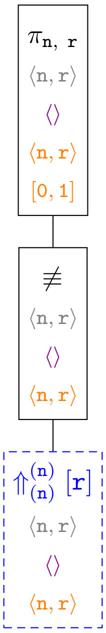
Relational algebra expression for search-based evaluation (Undirected match of self-relationship in self-relationship graph)

$$r = \pi_{\texttt{n, r}}\Big(\not\equiv\Big(\updownarrow\,^{\texttt{(n)}}_{\texttt{(n)}}[\texttt{r}]\Big(\bigcirc_{\texttt{(n)}}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Undirected match of self-relationship in self-relationship graph)


$\pi_{\texttt{n, r}}$
⟨n, r⟩
≢
⟨n, r⟩
$\updownarrow\,^{\texttt{(n)}}_{\texttt{(n)}}[\texttt{r}]$
⟨n, r⟩
$\bigcirc_{\texttt{(n)}}$
⟨n⟩

Incremental relational algebra tree (Undirected match of self-relationship in self-relationship graph)



A.20.4 Undirected match of self-relationship in self-relationship graph, count

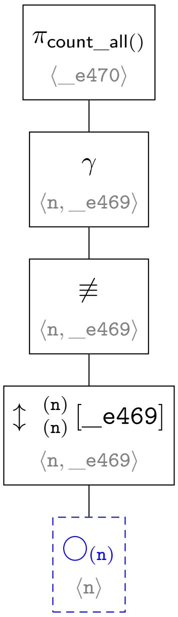
Query specification (Undirected match of self-relationship in self-relationship graph, count)

```
MATCH (n)--(n)
RETURN count(*)
```

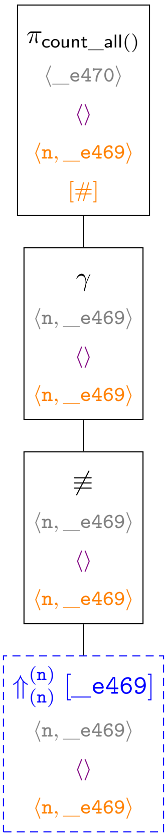
Relational algebra expression for search-based evaluation (Undirected match of self-relationship in self-relationship graph, count)

$$r = \pi_{\text{count\_all()}}\Big(\gamma\Big(\not\equiv\Big(\updownarrow^{(\mathtt{n})}_{(\mathtt{n})}[\text{\_e469}]\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Undirected match of self-relationship in self-relationship graph, count)


$\pi_{\text{count\_all()}}$
⟨_e470⟩
γ
⟨n, _e469⟩
$\not\equiv$
⟨n, _e469⟩
$\updownarrow^{(\mathtt{n})}_{(\mathtt{n})}$ [_e469]
⟨n, _e469⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩

Incremental relational algebra tree (Undirected match of self-relationship in self-relationship graph, count)



A.20.5 Undirected match on simple relationship graph

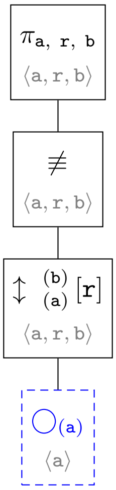
Query specification (Undirected match on simple relationship graph)

```
MATCH (a)-[r]-(b)
RETURN a, r, b
```

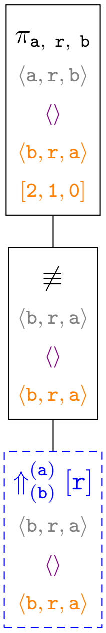
Relational algebra expression for search-based evaluation (Undirected match on simple relationship graph)

$$r = \pi_{\texttt{a, r, b}}\left(\not\equiv \left(\updownarrow\,{}^{(\texttt{b})}_{(\texttt{a})}\,[\texttt{r}] \left(\bigcirc_{(\texttt{a})} \right)\right)\right)$$

Relational algebra tree for search-based evaluation (Undirected match on simple relationship graph)


$\pi_{\texttt{a, r, b}}$
⟨a, r, b⟩
≢
⟨a, r, b⟩
$\updownarrow\,{}^{(\texttt{b})}_{(\texttt{a})}\,[\texttt{r}]$
⟨a, r, b⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩

Incremental relational algebra tree (Undirected match on simple relationship graph)



A.20.6 Undirected match on simple relationship graph, count

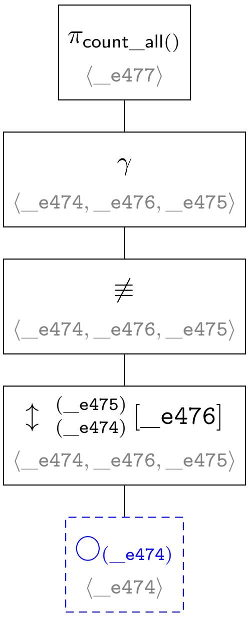
Query specification (Undirected match on simple relationship graph, count)

```
MATCH ()--()
RETURN count(*)
```

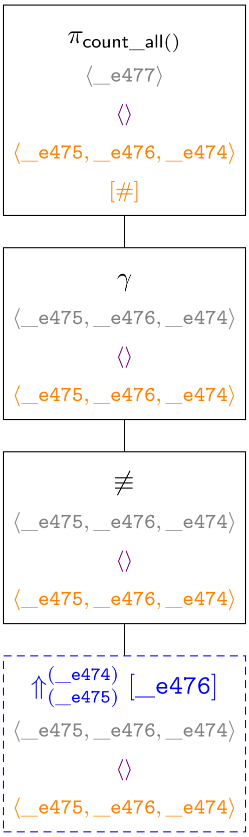
Relational algebra expression for search-based evaluation (Undirected match on simple relationship graph, count)

$$r = \pi_{\mathsf{count\_all()}}\Big(\gamma\Big(\neq\Big(\updownarrow^{(\_\mathsf{e475})}_{(\_\mathsf{e474})}[\_\mathsf{e476}]\Big(\bigcirc_{(\_\mathsf{e474})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Undirected match on simple relationship graph, count)


$\pi_{\mathsf{count\_all()}}$
⟨_e477⟩
γ
⟨_e474, _e476, _e475⟩
$\neq$
⟨_e474, _e476, _e475⟩
$\updownarrow^{(\_\mathsf{e475})}_{(\_\mathsf{e474})}[\_\mathsf{e476}]$
⟨_e474, _e476, _e475⟩
$\bigcirc_{(\_\mathsf{e474})}$
⟨_e474⟩

Incremental relational algebra tree (Undirected match on simple relationship graph, count)



A.20.7 Directed match on self-relationship graph

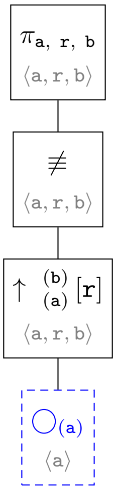
Query specification (Directed match on self-relationship graph)

```
MATCH (a)-[r]->(b)
RETURN a, r, b
```

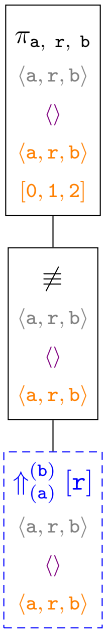
Relational algebra expression for search-based evaluation (Directed match on self-relationship graph)

$$r = \pi_{\texttt{a, r, b}}\Big(\not\equiv\Big(\uparrow^{(\texttt{b})}_{(\texttt{a})}[\texttt{r}]\Big(\bigcirc_{(\texttt{a})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Directed match on self-relationship graph)


$\pi_{\texttt{a, r, b}}$
⟨a, r, b⟩
≢
⟨a, r, b⟩
$\uparrow^{(\texttt{b})}_{(\texttt{a})}[\texttt{r}]$
⟨a, r, b⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩

Incremental relational algebra tree (Directed match on self-relationship graph)



A.20.8 Directed match on self-relationship graph, count

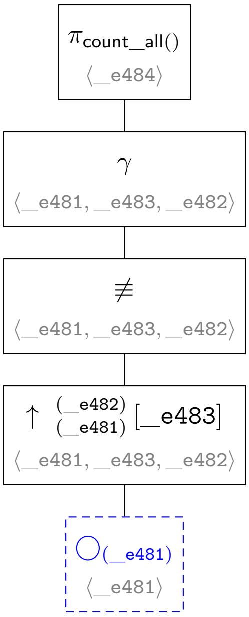
Query specification (Directed match on self-relationship graph, count)

```
MATCH ()-->()
RETURN count(*)
```

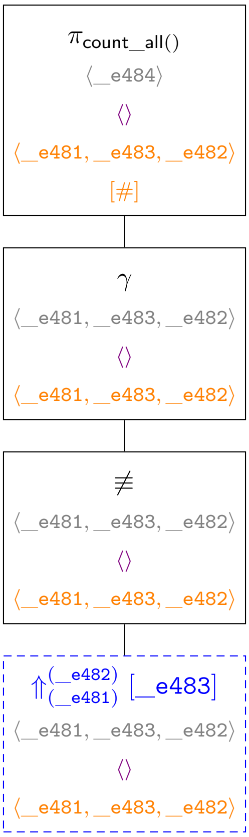
Relational algebra expression for search-based evaluation (Directed match on self-relationship graph, count)

$$r = \pi_{\mathsf{count\_all()}}\Big(\gamma\Big(\not\equiv\Big(\uparrow^{(\mathsf{\_e482})}_{(\mathsf{\_e481})}[\mathsf{\_e483}]\Big(\bigcirc_{(\mathsf{\_e481})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Directed match on self-relationship graph, count)



Incremental relational algebra tree (Directed match on self-relationship graph, count)



A.20.9 Directed match of self-relationship on self-relationship graph

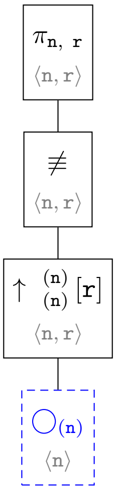
Query specification (Directed match of self-relationship on self-relationship graph)

```
MATCH (n)-[r]->(n)
RETURN n, r
```

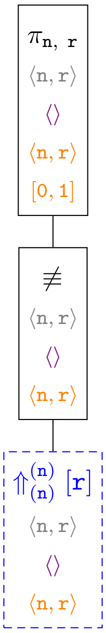
Relational algebra expression for search-based evaluation (Directed match of self-relationship on self-relationship graph)

$$r = \pi_{\texttt{n, r}} \Big(\not\equiv \Big(\uparrow {}^{(\texttt{n})}_{(\texttt{n})} [\texttt{r}] \Big(\bigcirc_{(\texttt{n})} \Big) \Big) \Big)$$

Relational algebra tree for search-based evaluation (Directed match of self-relationship on self-relationship graph)


$\pi_{\texttt{n, r}}$
⟨n, r⟩
$\not\equiv$
⟨n, r⟩
$\uparrow {}^{(\texttt{n})}_{(\texttt{n})} [\texttt{r}]$
⟨n, r⟩
$\bigcirc_{(\texttt{n})}$
⟨n⟩

Incremental relational algebra tree (Directed match of self-relationship on self-relationship graph)



A.20.10 Directed match of self-relationship on self-relationship graph, count

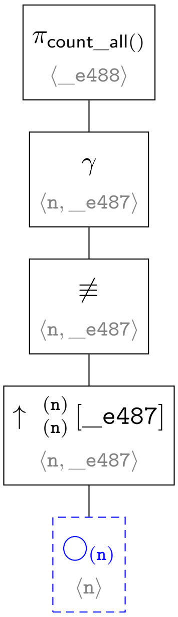
Query specification (Directed match of self-relationship on self-relationship graph, count)

```
MATCH (n)-->(n)
RETURN count(*)
```

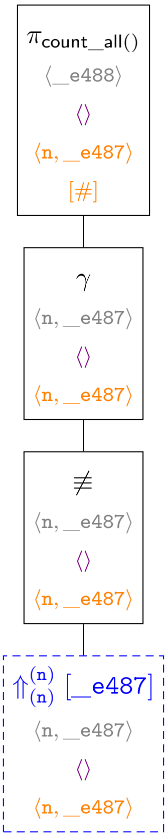
Relational algebra expression for search-based evaluation (Directed match of self-relationship on self-relationship graph, count)

$$r = \pi_{\mathsf{count\_all()}}\Big(\gamma\Big(\not\equiv\Big(\uparrow\ {}^{(\mathtt{n})}_{(\mathtt{n})}\,[\mathsf{\_e487}]\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Directed match of self-relationship on self-relationship graph, count)


$\pi_{\mathsf{count\_all()}}$
⟨_e488⟩
γ
⟨n, _e487⟩
$\not\equiv$
⟨n, _e487⟩
$\uparrow\ {}^{(\mathtt{n})}_{(\mathtt{n})}\,[\mathsf{\_e487}]$
⟨n, _e487⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩

Incremental relational algebra tree (Directed match of self-relationship on self-relationship graph, count)



A.20.11 Counting undirected self-relationships in self-relationship graph

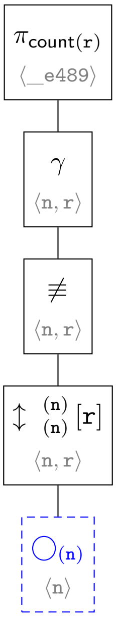
Query specification (Counting undirected self-relationships in self-relationship graph)

```
MATCH (n)-[r]-(n)
RETURN count(r)
```

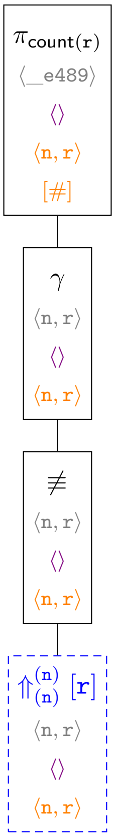
Relational algebra expression for search-based evaluation (Counting undirected self-relationships in self-relationship graph)

$$r = \pi_{\mathsf{count(r)}}\Big(\gamma\Big(\neq\Big(\updownarrow\,{}^{(\mathtt{n})}_{(\mathtt{n})}\,[\mathtt{r}]\,\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Counting undirected self-relationships in self-relationship graph)


$\pi_{\mathsf{count(r)}}$
⟨_e489⟩
γ
⟨n, r⟩
$\neq$
⟨n, r⟩
$\updownarrow\,{}^{(\mathtt{n})}_{(\mathtt{n})}\,[\mathtt{r}]$
⟨n, r⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩

Incremental relational algebra tree (Counting undirected self-relationships in self-relationship graph)



A.20.12 Counting distinct undirected self-relationships in self-relationship graph

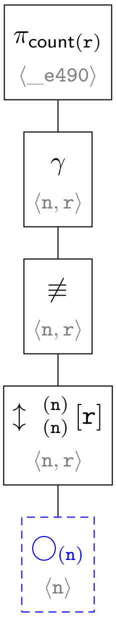
Query specification (Counting distinct undirected self-relationships in self-relationship graph)

```
MATCH (n)-[r]-(n)
RETURN count(DISTINCT r)
```

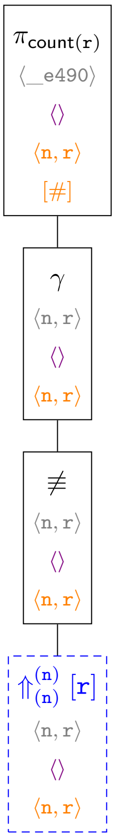
Relational algebra expression for search-based evaluation (Counting distinct undirected self-relationships in self-relationship graph)

$$r = \pi_{\mathsf{count(r)}}\Big(\gamma\Big(\not\equiv\Big(\updownarrow\,{}^{(\mathtt{n})}_{(\mathtt{n})}\,[\mathtt{r}]\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Counting distinct undirected self-relationships in self-relationship graph)



Incremental relational algebra tree (Counting distinct undirected self-relationships in self-relationship graph)



A.20.13 Directed match of a simple relationship

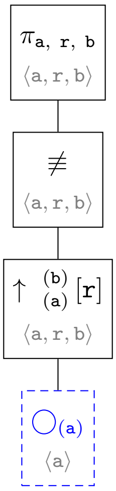
Query specification (Directed match of a simple relationship)

```
MATCH (a)-[r]->(b)
RETURN a, r, b
```

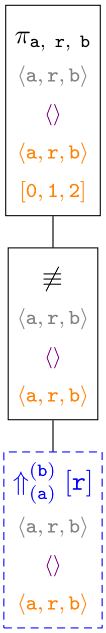
Relational algebra expression for search-based evaluation (Directed match of a simple relationship)

$$r = \pi_{\texttt{a, r, b}}\Big(\not\equiv\Big(\uparrow^{(\texttt{b})}_{(\texttt{a})}[\texttt{r}]\Big(\bigcirc_{(\texttt{a})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Directed match of a simple relationship)



Incremental relational algebra tree (Directed match of a simple relationship)



A.20.14 Directed match of a simple relationship, count

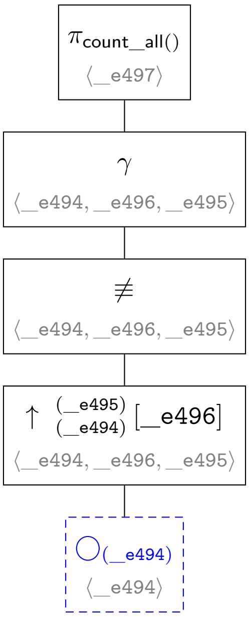
Query specification (Directed match of a simple relationship, count)

```
MATCH ()-->()
RETURN count(*)
```

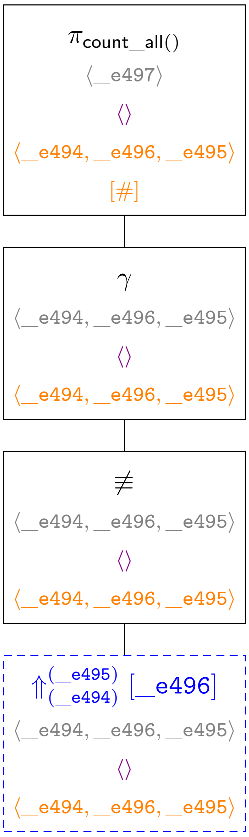
Relational algebra expression for search-based evaluation (Directed match of a simple relationship, count)

$$r = \pi_{\mathsf{count\_all()}}\Big(\gamma\Big(\not\equiv\Big(\uparrow^{(\mathsf{\_e495})}_{(\mathsf{\_e494})}[\mathsf{\_e496}]\Big(\bigcirc_{(\mathsf{\_e494})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Directed match of a simple relationship, count)



Incremental relational algebra tree (Directed match of a simple relationship, count)



A.20.15 Counting directed self-relationships

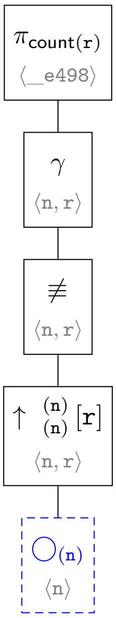
Query specification (Counting directed self-relationships)

```
MATCH (n)-[r]->(n)
RETURN count(r)
```

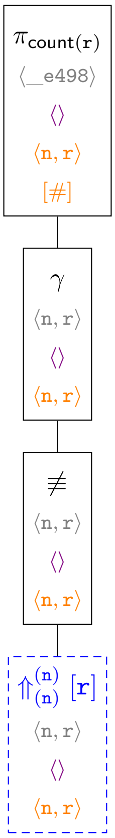
Relational algebra expression for search-based evaluation (Counting directed self-relationships)

$$r = \pi_{\mathsf{count}(\mathtt{r})}\Big(\gamma\Big(\not\equiv\Big(\uparrow\ {}^{(\mathtt{n})}_{(\mathtt{n})}\,[\mathtt{r}]\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Counting directed self-relationships)



Incremental relational algebra tree (Counting directed self-relationships)



A.20.16 Mixing directed and undirected pattern parts with self-relationship, simple

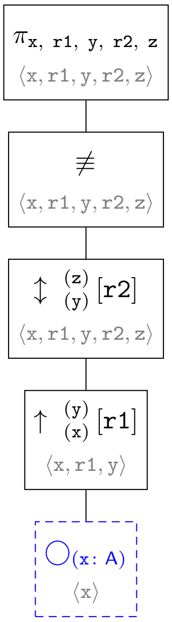
Query specification (Mixing directed and undirected pattern parts with self-relationship, simple)

```
MATCH (x:A)-[r1]->(y)-[r2]-(z)
RETURN x, r1, y, r2, z
```

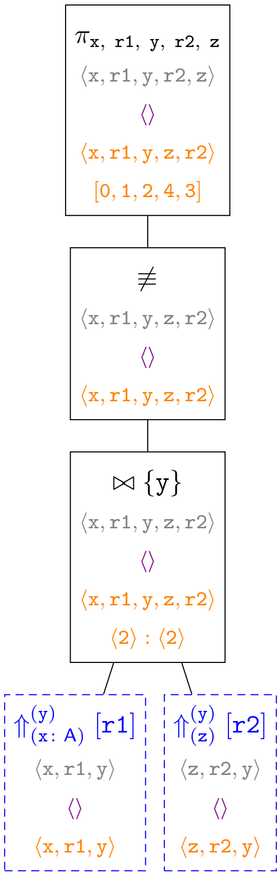
Relational algebra expression for search-based evaluation (Mixing directed and undirected pattern parts with self-relationship, simple)

$$r = \pi_{\texttt{x, r1, y, r2, z}}\left(\not\equiv \left(\updownarrow\,{}^{(\texttt{z})}_{(\texttt{y})}\,[\texttt{r2}] \left(\uparrow\,{}^{(\texttt{y})}_{(\texttt{x})}\,[\texttt{r1}] \left(\bigcirc_{(\texttt{x: A})} \right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Mixing directed and undirected pattern parts with self-relationship, simple)



Incremental relational algebra tree (Mixing directed and undirected pattern parts with self-relationship, simple)



A.20.17 Mixing directed and undirected pattern parts with self-relationship, count

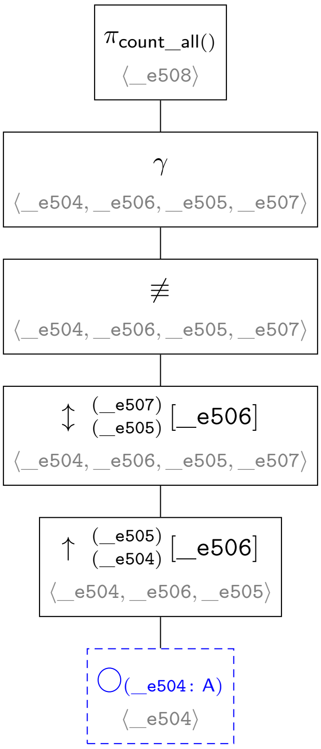
Query specification (Mixing directed and undirected pattern parts with self-relationship, count)

```
MATCH (:A)-->()--()
RETURN count(*)
```

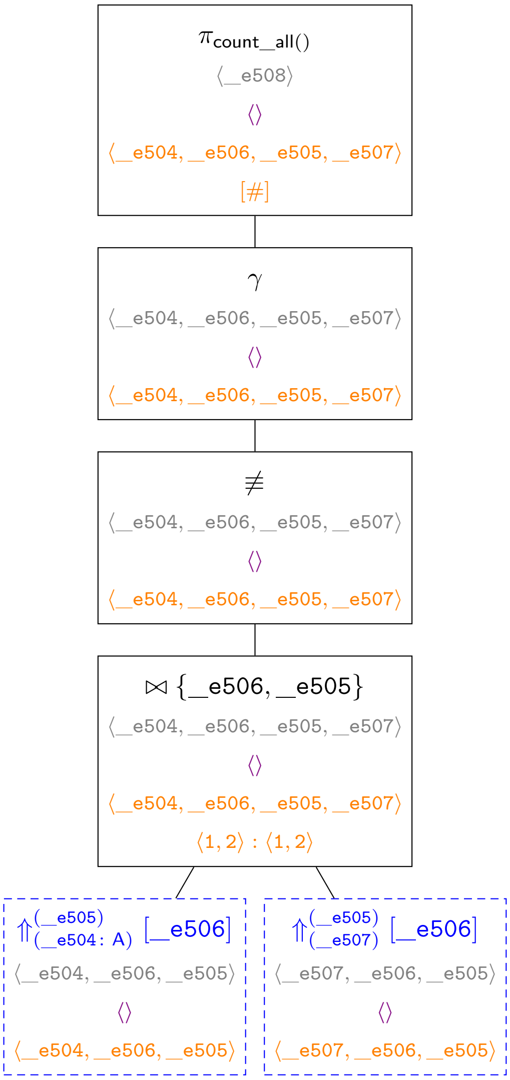
Relational algebra expression for search-based evaluation (Mixing directed and undirected pattern parts with self-relationship, count)

$$r = \pi_{\mathsf{count\_all()}}\left(\gamma\left(\not\equiv\left(\updownarrow\,^{(\mathsf{\_e507})}_{(\mathsf{\_e505})}\,[\mathsf{\_e506}]\left(\uparrow\,^{(\mathsf{\_e505})}_{(\mathsf{\_e504})}\,[\mathsf{\_e506}]\left(\bigcirc_{(\mathsf{\_e504:\ A})}\right)\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Mixing directed and undirected pattern parts with self-relationship, count)


$\pi_{\text{count\_all()}}$
⟨_e508⟩
γ
⟨_e504, _e506, _e505, _e507⟩
$\neq$
⟨_e504, _e506, _e505, _e507⟩
$\updownarrow^{(\_e507)}_{(\_e505)}$ [_e506]
⟨_e504, _e506, _e505, _e507⟩
$\uparrow^{(\_e505)}_{(\_e504)}$ [_e506]
⟨_e504, _e506, _e505⟩
○(_e504 : A)
⟨_e504⟩

Incremental relational algebra tree (Mixing directed and undirected pattern parts with self-relationship, count)


π count_all()
⟨_e508⟩
⟨⟩
⟨_e504, _e506, _e505, _e507⟩
[#]
γ
⟨_e504, _e506, _e505, _e507⟩
⟨⟩
⟨_e504, _e506, _e505, _e507⟩
≢
⟨_e504, _e506, _e505, _e507⟩
⟨⟩
⟨_e504, _e506, _e505, _e507⟩
⋈ {_e506, _e505}
⟨_e504, _e506, _e505, _e507⟩
⟨⟩
⟨_e504, _e506, _e505, _e507⟩
⟨1, 2⟩ : ⟨1, 2⟩
⇑(_e505) (_e504: A) [_e506]
⟨_e504, _e506, _e505⟩
⟨⟩
⟨_e504, _e506, _e505⟩
⇑(_e505) (_e507) [_e506]
⟨_e507, _e506, _e505⟩
⟨⟩
⟨_e507, _e506, _e505⟩

A.20.18 Mixing directed and undirected pattern parts with self-relationship, undirected

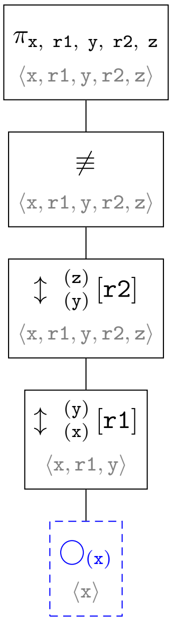
Query specification (Mixing directed and undirected pattern parts with self-relationship, undirected)

```
MATCH (x)-[r1]-(y)-[r2]-(z)
RETURN x, r1, y, r2, z
```

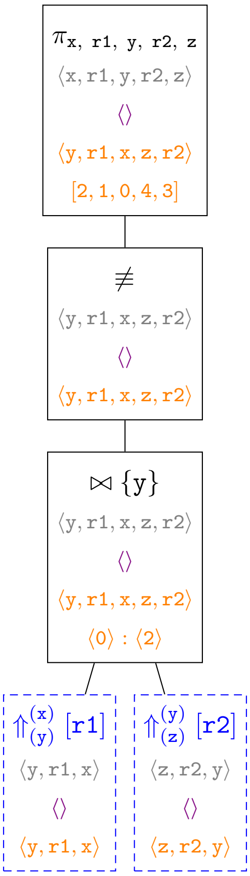
Relational algebra expression for search-based evaluation (Mixing directed and undirected pattern parts with self-relationship, undirected)

$$r = \pi_{\texttt{x, r1, y, r2, z}}\left(\not\equiv\left(\updownarrow{}^{(\texttt{z})}_{(\texttt{y})}[\texttt{r2}]\left(\updownarrow{}^{(\texttt{y})}_{(\texttt{x})}[\texttt{r1}]\left(\bigcirc_{(\texttt{x})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Mixing directed and undirected pattern parts with self-relationship, undirected)



Incremental relational algebra tree (Mixing directed and undirected pattern parts with self-relationship, undirected)



A.20.19 Mixing directed and undirected pattern parts with self-relationship, undirected count

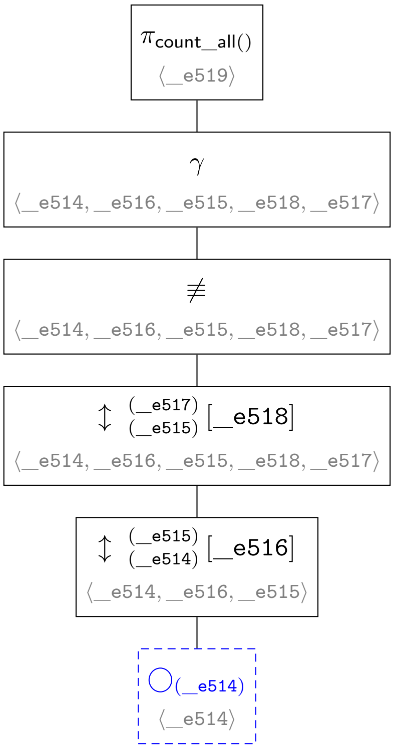
Query specification (Mixing directed and undirected pattern parts with self-relationship, undirected count)

```
MATCH ()-[]-()-[]-()
RETURN count(*)
```

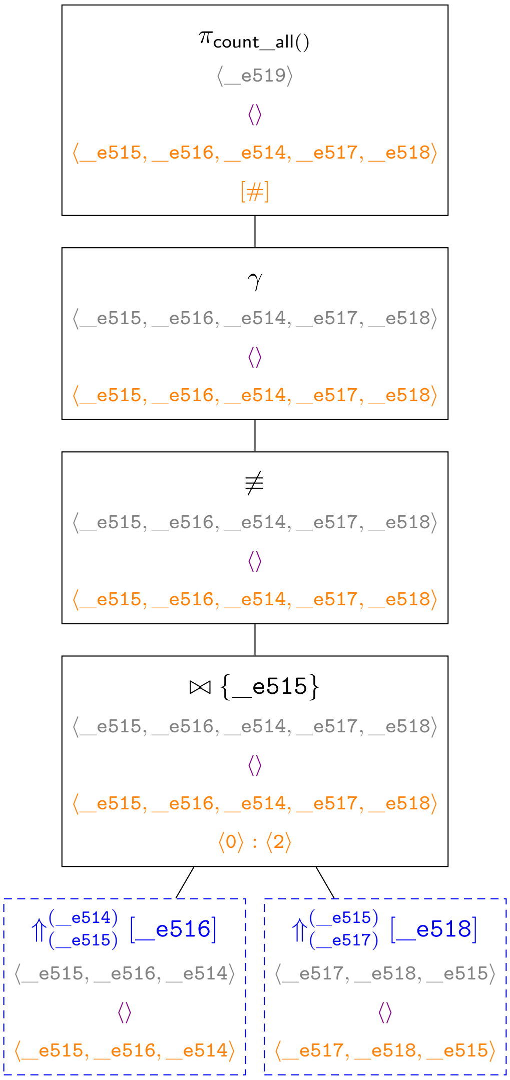
Relational algebra expression for search-based evaluation (Mixing directed and undirected pattern parts with self-relationship, undirected count)

$$r = \pi_{\text{count\_all()}}\left(\gamma\left(\not\equiv\left(\updownarrow^{(\_\text{e517})}_{(\_\text{e515})}[\_\text{e518}]\left(\updownarrow^{(\_\text{e515})}_{(\_\text{e514})}[\_\text{e516}]\left(\bigcirc_{(\_\text{e514})}\right)\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Mixing directed and undirected pattern parts with self-relationship, undirected count)


π count_all()
⟨_e519⟩
γ
⟨_e514, _e516, _e515, _e518, _e517⟩
≢
⟨_e514, _e516, _e515, _e518, _e517⟩
↕ (_e517) (_e515) [_e518]
⟨_e514, _e516, _e515, _e518, _e517⟩
↕ (_e515) (_e514) [_e516]
⟨_e514, _e516, _e515⟩
○(_e514)
⟨_e514⟩

Incremental relational algebra tree (Mixing directed and undirected pattern parts with self-relationship, undirected count)


$\pi_{\text{count\_all()}}$
⟨_e519⟩
⟨⟩
⟨_e515, _e516, _e514, _e517, _e518⟩
[#]
γ
⟨_e515, _e516, _e514, _e517, _e518⟩
⟨⟩
⟨_e515, _e516, _e514, _e517, _e518⟩
$\neq$
⟨_e515, _e516, _e514, _e517, _e518⟩
⟨⟩
⟨_e515, _e516, _e514, _e517, _e518⟩
⋈ {_e515}
⟨_e515, _e516, _e514, _e517, _e518⟩
⟨⟩
⟨_e515, _e516, _e514, _e517, _e518⟩
⟨0⟩ : ⟨2⟩
$\Uparrow^{(\_e514)}_{(\_e515)}$ [_e516]
⟨_e515, _e516, _e514⟩
⟨⟩
⟨_e515, _e516, _e514⟩
$\Uparrow^{(\_e515)}_{(\_e517)}$ [_e518]
⟨_e517, _e518, _e515⟩
⟨⟩
⟨_e517, _e518, _e515⟩

A.21 MergeIntoAcceptance

Progress:

0 of 0

A.22 MergeNodeAcceptance

Progress:

0 of 0

A.23 MergeRelationshipAcceptance

Progress:

0 of 0

A.24 MiscellaneousErrorAcceptance

Progress:

0 of 0

A.25 NullAcceptance

Progress:

0 of 0

A.26 OptionalMatch

Progress:

3 of 3

A.26.1 Satisfies the open world assumption, relationships between same nodes

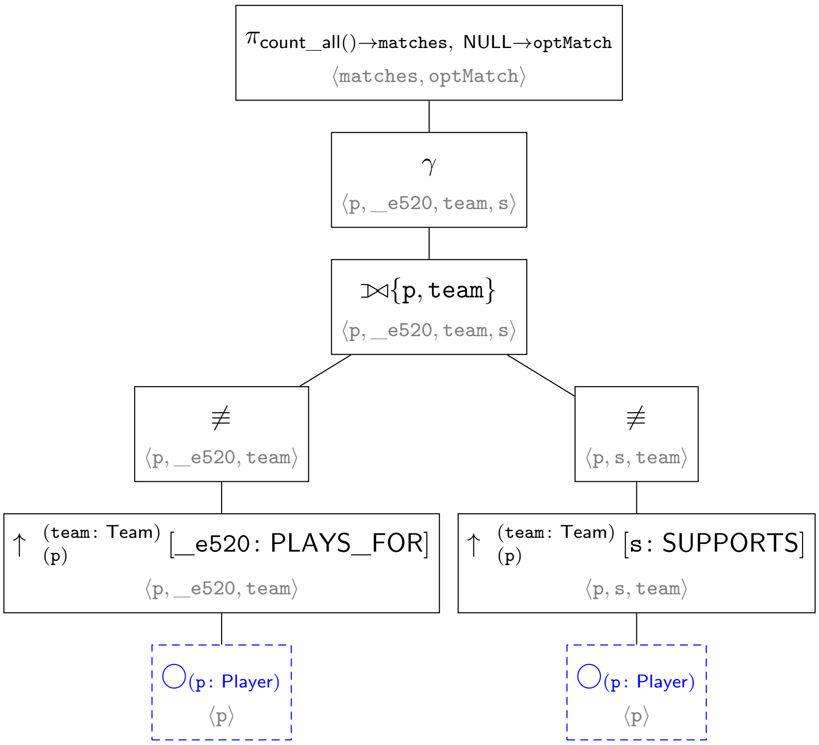
Query specification (Satisfies the open world assumption, relationships between same nodes)

```
MATCH (p:Player)-[:PLAYS_FOR]->(team:Team)
OPTIONAL MATCH (p)-[s:SUPPORTS]->(team)
RETURN count(*) AS matches, s IS NULL AS optMatch
```

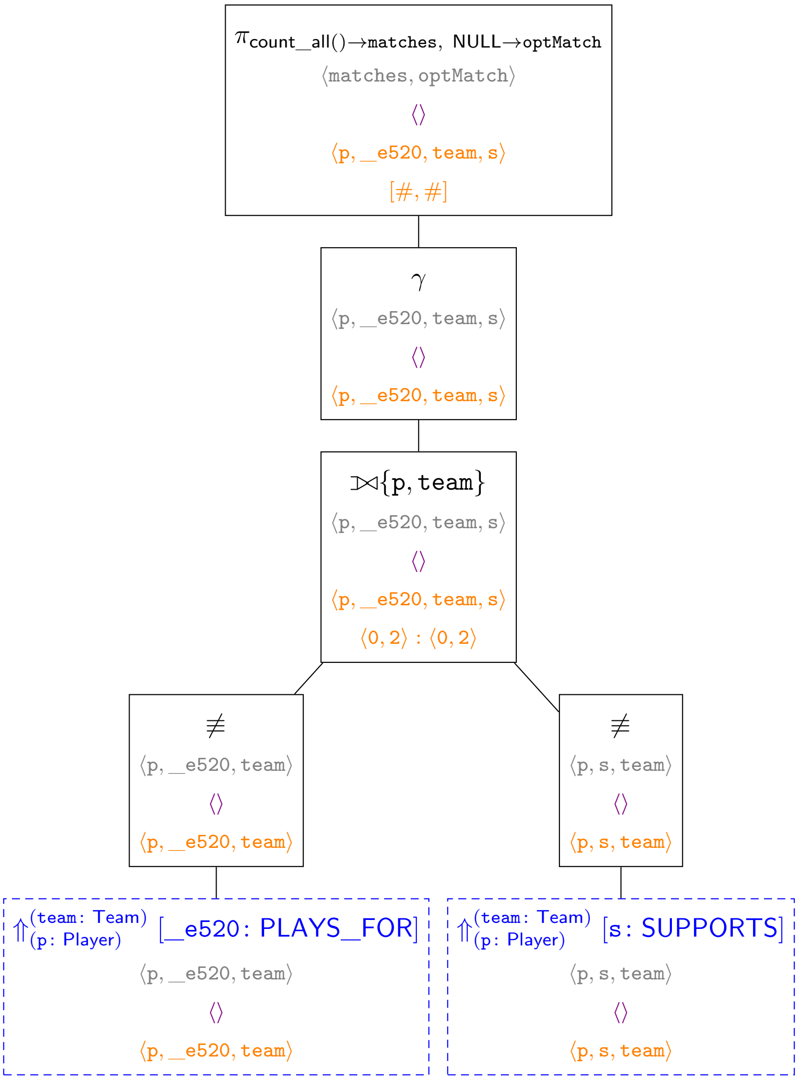
Relational algebra expression for search-based evaluation (Satisfies the open world assumption, relationships between same nodes)

$$r = \pi_{\mathsf{count\_all}()\rightarrow\mathtt{matches},\ \mathsf{NULL}\rightarrow\mathtt{optMatch}}\Bigg(\gamma\Bigg(\not\equiv\Bigg(\uparrow^{(\mathtt{team}:\ \mathsf{Team})}_{(\mathtt{p})}[\mathsf{\_e520}:\ \mathsf{PLAYS\_FOR}]\Bigg(\bigcirc_{(\mathtt{p}:\ \mathsf{Player})}\Bigg)\Bigg)\leftouterjoin\not\equiv\Bigg(\uparrow^{(\mathtt{team}:\ \mathsf{Team})}_{(\mathtt{p})}[\mathsf{s}:\ \mathsf{SUPPORTS}]\Bigg(\bigcirc_{(\mathtt{p}:\ \mathsf{Player})}\Bigg)\Bigg)\Bigg)\Bigg)$$

Relational algebra tree for search-based evaluation (Satisfies the open world assumption, relationships between same nodes)



Incremental relational algebra tree (Satisfies the open world assumption, relationships between same nodes)



A.26.2 Satisfies the open world assumption, single relationship

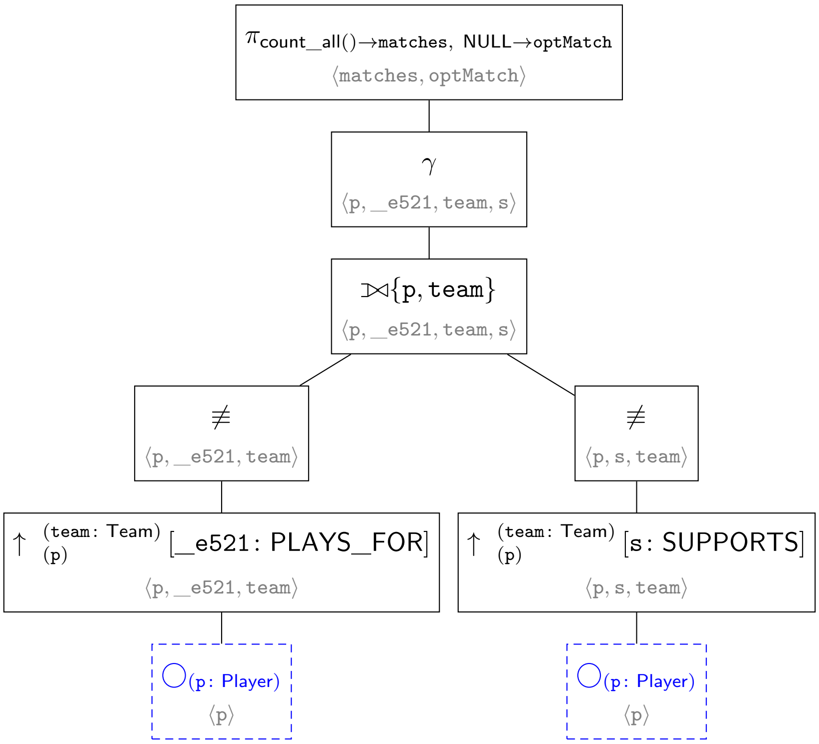
Query specification (Satisfies the open world assumption, single relationship)

```
MATCH (p:Player)-[:PLAYS_FOR]->(team:Team)
OPTIONAL MATCH (p)-[s:SUPPORTS]->(team)
RETURN count(*) AS matches, s IS NULL AS optMatch
```

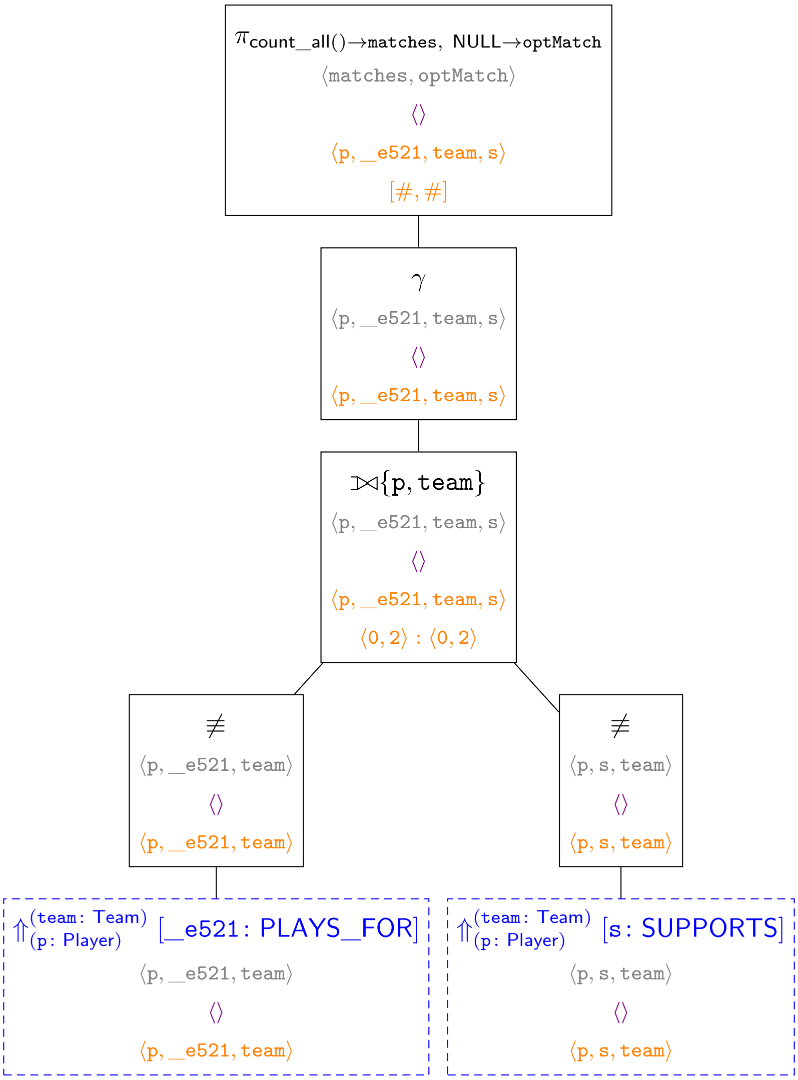
Relational algebra expression for search-based evaluation (Satisfies the open world assumption, single relationship)

$$r = \pi_{\mathsf{count\_all()}\rightarrow\mathtt{matches},\ \mathsf{NULL}\rightarrow\mathtt{optMatch}}\Bigg(\gamma\Bigg(\not\equiv\Bigg(\uparrow\,{}^{(\mathtt{team}:\ \mathsf{Team})}_{(\mathtt{p})}\,[\mathsf{\_e521}:\ \mathsf{PLAYS\_FOR}]\Bigg(\bigcirc_{(\mathtt{p}:\ \mathsf{Player})}\Bigg)\Bigg)\mathbin{⟕}\not\equiv\Bigg(\uparrow\,{}^{(\mathtt{team}:\ \mathsf{Team})}_{(\mathtt{p})}\,[\mathsf{s}:\ \mathsf{SUPPORTS}]\Bigg(\bigcirc_{(\mathtt{p}:\ \mathsf{Player})}\Bigg)\Bigg)\Bigg)\Bigg)$$

Relational algebra tree for search-based evaluation (Satisfies the open world assumption, single relationship)



Incremental relational algebra tree (Satisfies the open world assumption, single relationship)



A.26.3 Satisfies the open world assumption, relationships between different nodes

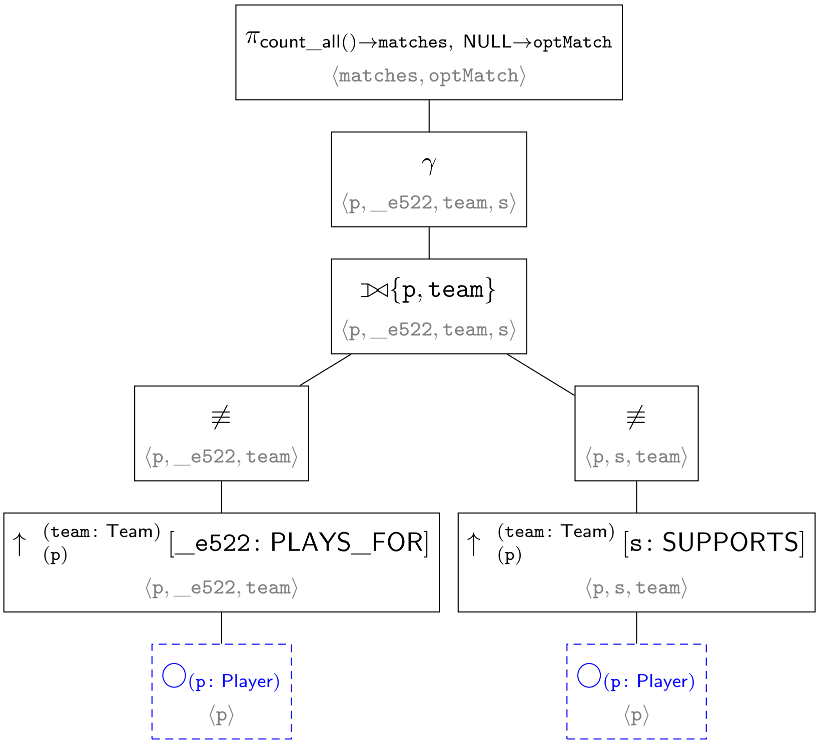
Query specification (Satisfies the open world assumption, relationships between different nodes)

```
MATCH (p:Player)-[:PLAYS_FOR]->(team:Team)
OPTIONAL MATCH (p)-[s:SUPPORTS]->(team)
RETURN count(*) AS matches, s IS NULL AS optMatch
```

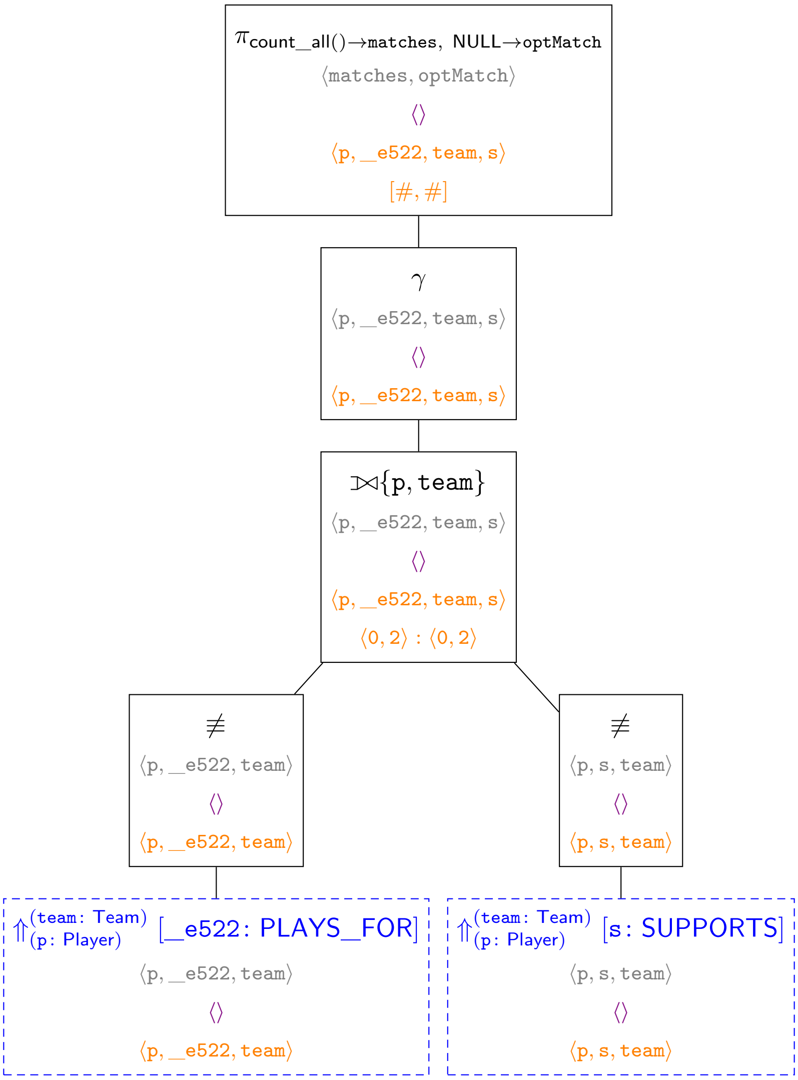
Relational algebra expression for search-based evaluation (Satisfies the open world assumption, relationships between different nodes)

$$r = \pi_{\text{count\_all()}\rightarrow\texttt{matches},\ \text{NULL}\rightarrow\texttt{optMatch}}\left(\gamma\left(\not\equiv\left(\uparrow^{(\text{team: Team})}_{(\text{p})}[\text{\_e522: PLAYS\_FOR}]\left(\bigcirc_{(\text{p: Player})}\right)\right)⟕\not\equiv\left(\uparrow^{(\text{team: Team})}_{(\text{p})}[\text{s: SUPPORTS}]\left(\bigcirc_{(\text{p: Player})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation **(Satisfies the open world assumption, relationships between different nodes)**


$\pi_{\text{count\_all()}\rightarrow\texttt{matches},\ \text{NULL}\rightarrow\texttt{optMatch}}$
⟨matches, optMatch⟩
γ
⟨p, _e522, team, s⟩
⋊⋉{p, team}
⟨p, _e522, team, s⟩
≠
⟨p, _e522, team⟩
≠
⟨p, s, team⟩
↑ (team: Team) (p) [_e522: PLAYS_FOR]
⟨p, _e522, team⟩
↑ (team: Team) (p) [s: SUPPORTS]
⟨p, s, team⟩
○(p: Player)
⟨p⟩
○(p: Player)
⟨p⟩

Incremental relational algebra tree (Satisfies the open world assumption, relationships between different nodes)



A.27 OptionalMatchAcceptance

Progress:

15 of 22

A.27.1 Return null when no matches due to inline label predicate

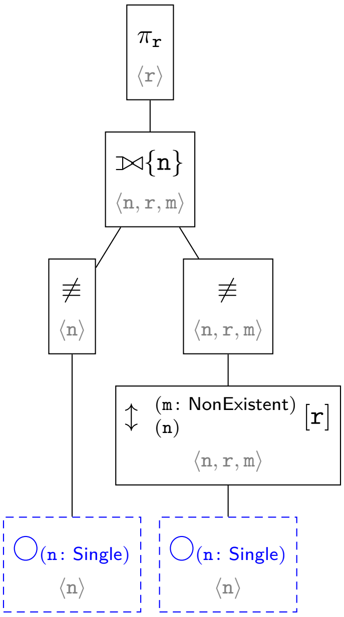
Query specification (Return null when no matches due to inline label predicate)

```
MATCH (n:Single)
OPTIONAL MATCH (n)-[r]-(m:NonExistent)
RETURN r
```

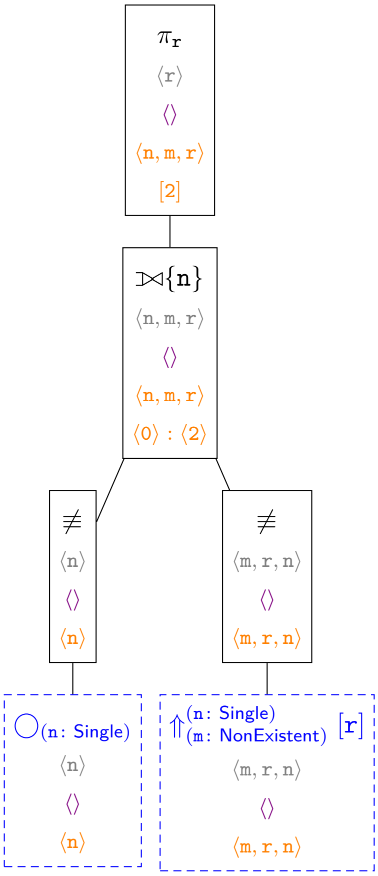
Relational algebra expression for search-based evaluation (Return null when no matches due to inline label predicate)

$$r = \pi_{\mathtt{r}}\Big(\not\equiv \Big(\bigcirc_{(\mathtt{n}:\ \mathsf{Single})} \Big) ⟕ \not\equiv \Big(\updownarrow^{(\mathtt{m}:\ \mathsf{NonExistent})}_{(\mathtt{n})} [\mathtt{r}] \Big(\bigcirc_{(\mathtt{n}:\ \mathsf{Single})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Return null when no matches due to inline label predicate)



Incremental relational algebra tree (Return null when no matches due to inline label predicate)



A.27.2 Return null when no matches due to label predicate in WHERE

Query specification (Return null when no matches due to label predicate in WHERE)

```
MATCH (n:Single)
OPTIONAL MATCH (n)-[r]-(m)
WHERE m:NonExistent
RETURN r
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.27.3 Respect predicates on the OPTIONAL MATCH

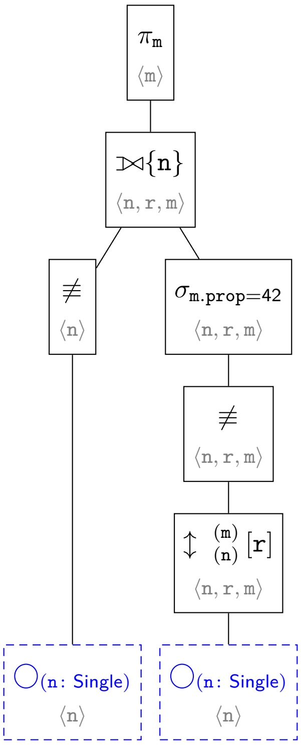
Query specification (Respect predicates on the OPTIONAL MATCH)

```
MATCH (n:Single)
OPTIONAL MATCH (n)-[r]-(m)
WHERE m.prop = 42
RETURN m
```

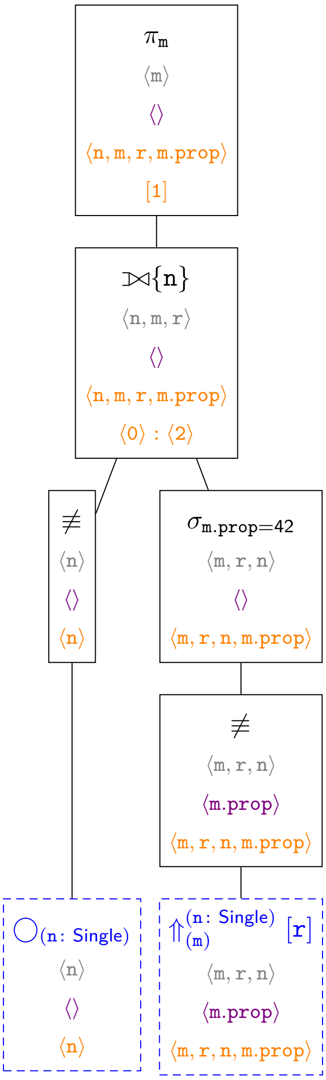
Relational algebra expression for search-based evaluation (Respect predicates on the OPTIONAL MATCH)

$$r = \pi_{\mathtt{m}}\Big(\not\equiv \Big(\bigcirc_{(\mathtt{n}:\ \mathsf{Single})} \Big) ⟕ \sigma_{\mathtt{m.prop}=42}\Big(\not\equiv \Big(\updownarrow {}^{(\mathtt{m})}_{(\mathtt{n})} [\mathtt{r}] \Big(\bigcirc_{(\mathtt{n}:\ \mathsf{Single})} \Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Respect predicates on the OPTIONAL MATCH)



Incremental relational algebra tree (Respect predicates on the OPTIONAL MATCH)



A.27.4 Returning label predicate on null node

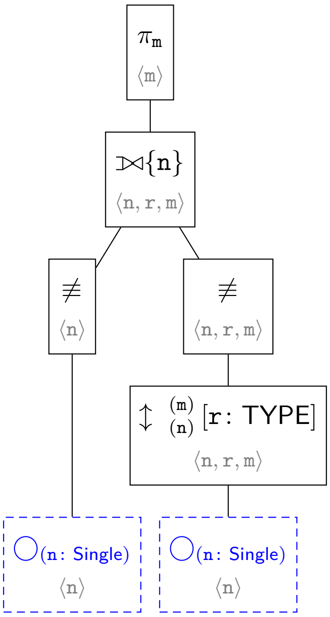
Query specification (Returning label predicate on null node)

```
MATCH (n:Single)
OPTIONAL MATCH (n)-[r:TYPE]-(m)
RETURN m:TYPE
```

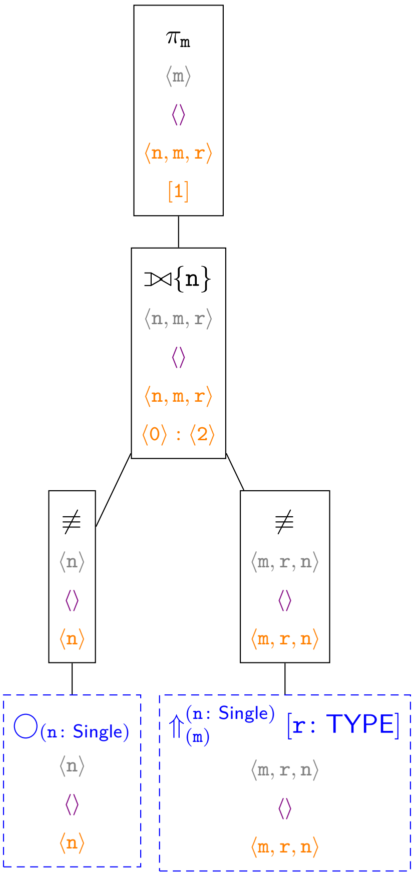
Relational algebra expression for search-based evaluation (Returning label predicate on null node)

$$r = \pi_{\mathtt{m}}\left(\not\equiv \left(\bigcirc_{(\mathtt{n}:\ \mathsf{Single})} \right) ⟕ \not\equiv \left(\updownarrow\, ^{(\mathtt{m})}_{(\mathtt{n})} [\mathtt{r}:\ \mathsf{TYPE}] \left(\bigcirc_{(\mathtt{n}:\ \mathsf{Single})} \right) \right) \right)$$

Relational algebra tree for search-based evaluation (Returning label predicate on null node)



Incremental relational algebra tree (Returning label predicate on null node)



A.27.5 MATCH after OPTIONAL MATCH

Query specification (MATCH after OPTIONAL MATCH)

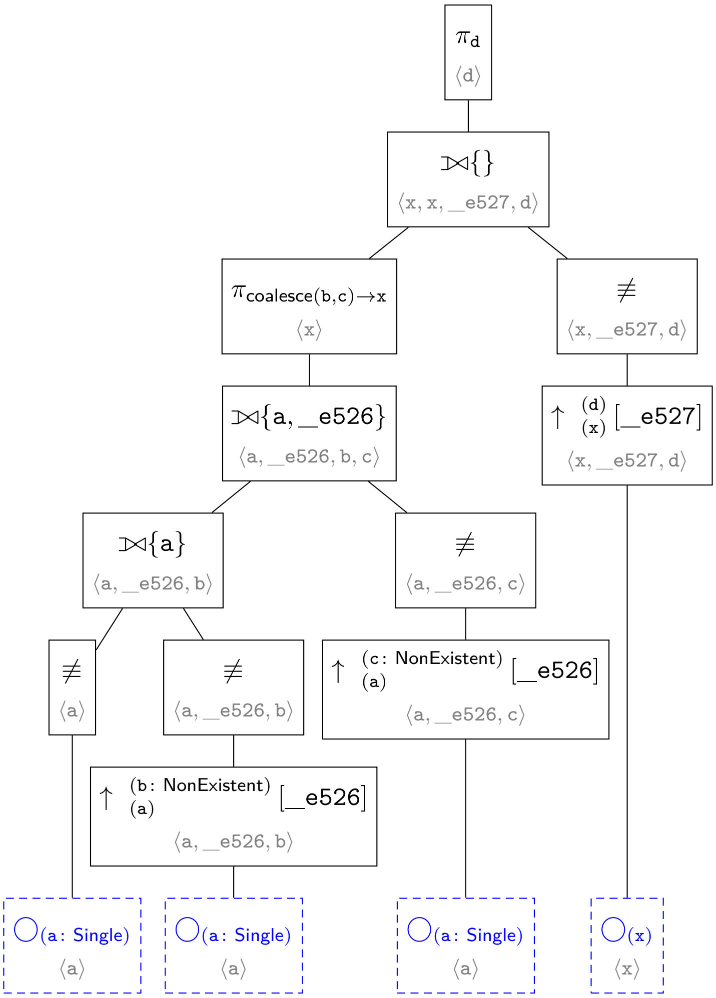
```
MATCH (a:Single)
OPTIONAL MATCH (a)-->(b:NonExistent)
OPTIONAL MATCH (a)-->(c:NonExistent)
```

```
WITH coalesce(b, c) AS x
MATCH (x)-->(d)
RETURN d
```

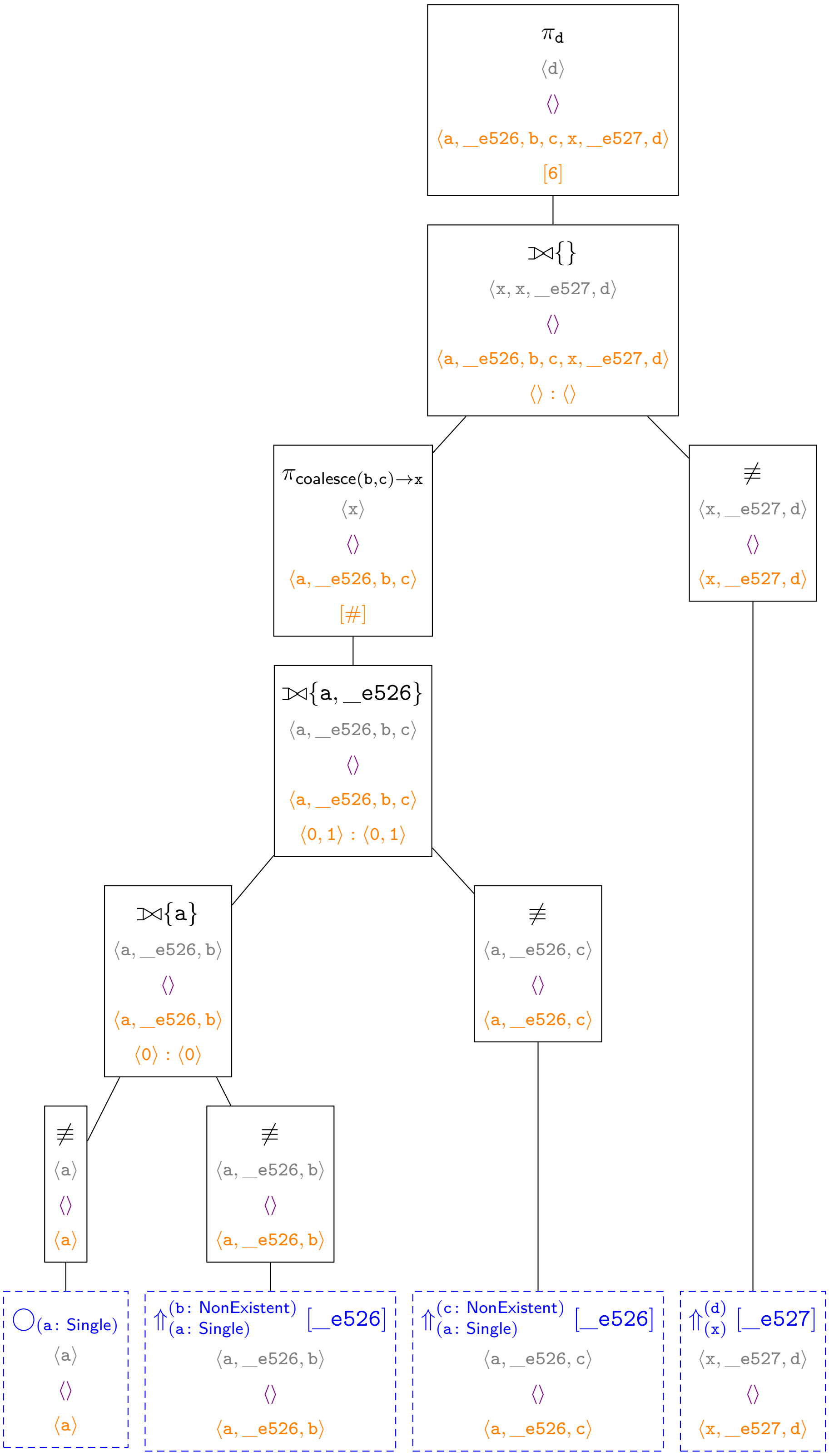
Relational algebra expression for search-based evaluation (MATCH after OPTIONAL MATCH)

$$r = \pi_{\mathtt{d}}\Big(\pi_{\mathsf{coalesce}(\mathtt{b},\mathtt{c})\rightarrow\mathtt{x}}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{a}:\ \mathsf{Single})}\Big)\text{⟕}\not\equiv\Big(\uparrow^{(\mathtt{b}:\ \mathsf{NonExistent})}_{(\mathtt{a})}[\_\mathsf{e526}]\Big(\bigcirc_{(\mathtt{a}:\ \mathsf{Single})}\Big)\Big)\text{⟕}\not\equiv\Big(\uparrow^{(\mathtt{c}:\ \mathsf{NonExistent})}_{(\mathtt{a})}[\_\mathsf{e526}]\Big(\bigcirc_{(\mathtt{a}:\ \mathsf{Single})}\Big)\Big)\Big)\text{⟕}\not\equiv\Big(\uparrow^{(\mathtt{d})}_{(\mathtt{x})}[\_\mathsf{e527}]\Big(\bigcirc_{(\mathtt{x})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (MATCH after OPTIONAL MATCH)



Incremental relational algebra tree (MATCH after OPTIONAL MATCH)



A.27.6 WITH after OPTIONAL MATCH

Query specification (WITH after OPTIONAL MATCH)

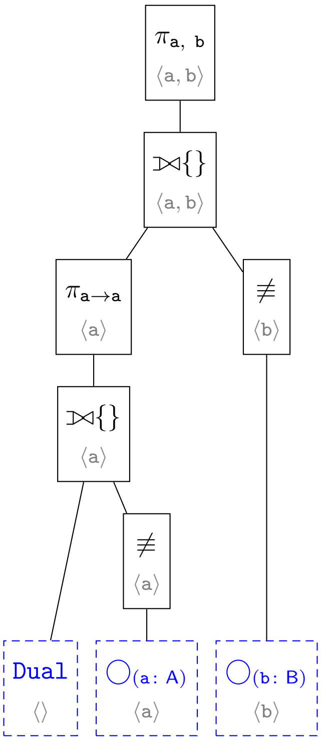
```
OPTIONAL MATCH (a:A)
WITH a AS a
```

```
MATCH (b:B)
RETURN a, b
```

Relational algebra expression for search-based evaluation **(WITH after OPTIONAL MATCH)**

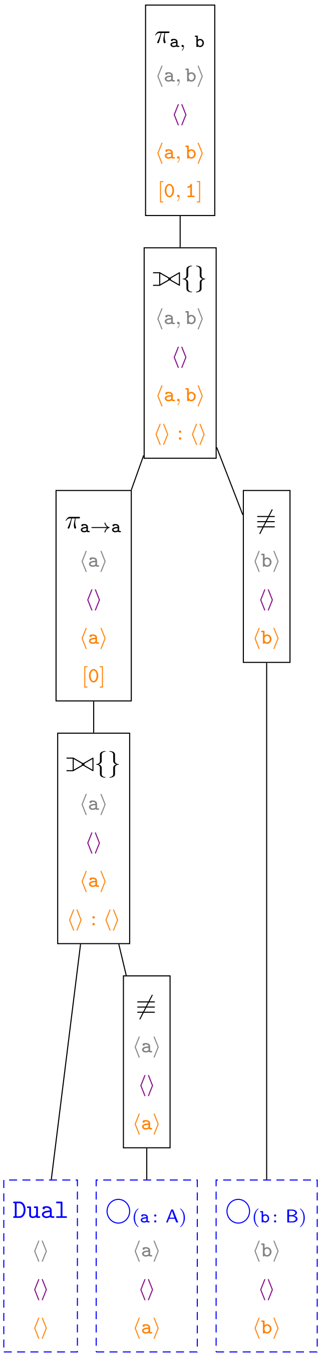
$$r = \pi_{\texttt{a, b}}\Big(\pi_{\texttt{a}\rightarrow\texttt{a}}\Big(\texttt{Dual}\,⟕\!\not\equiv\Big(\bigcirc_{(\texttt{a: A})}\Big)\Big)⟕\!\not\equiv\Big(\bigcirc_{(\texttt{b: B})}\Big)\Big)$$

Relational algebra tree for search-based evaluation **(WITH after OPTIONAL MATCH)**



$\pi_{\texttt{a, b}}$
⟨a, b⟩
⟕{}
⟨a, b⟩
$\pi_{\texttt{a}\rightarrow\texttt{a}}$
⟨a⟩
≢
⟨b⟩
⟕{}
⟨a⟩
≢
⟨a⟩
Dual
⟨⟩
$\bigcirc_{(\texttt{a: A})}$
⟨a⟩
$\bigcirc_{(\texttt{b: B})}$
⟨b⟩

Incremental relational algebra tree (WITH after OPTIONAL MATCH)



A.27.7 Named paths in optional matches

Query specification (Named paths in optional matches)

```
MATCH (a:A)
OPTIONAL MATCH p = (a)-[:X]->(b)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.27.8 OPTIONAL MATCH and bound nodes

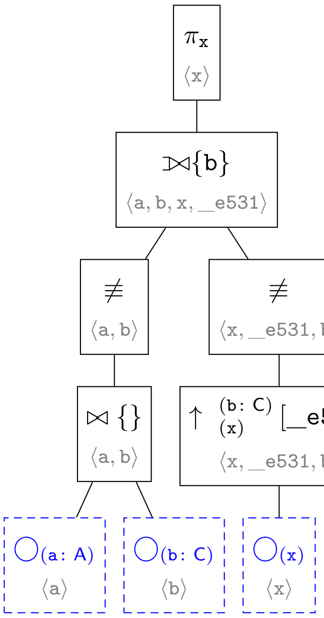
Query specification (OPTIONAL MATCH and bound nodes)

```
MATCH (a:A), (b:C)
OPTIONAL MATCH (x)-->(b)
RETURN x
```

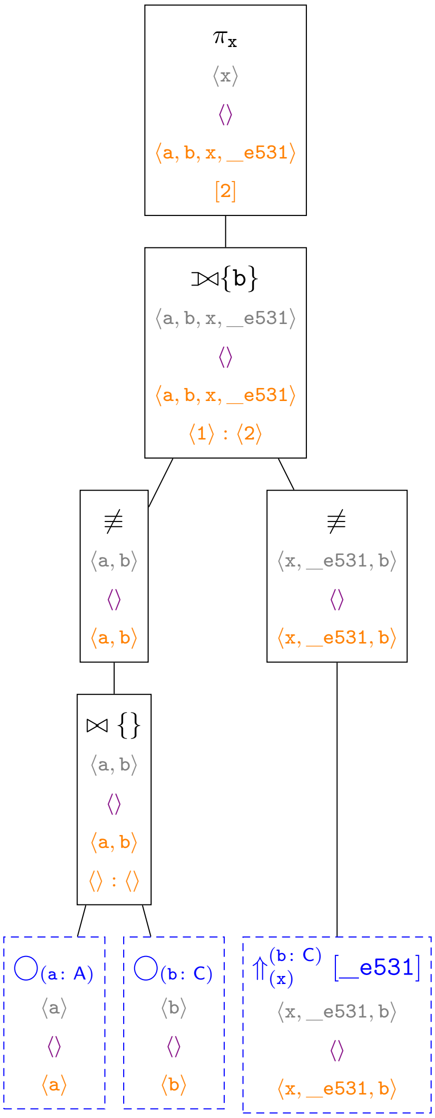
Relational algebra expression for search-based evaluation (OPTIONAL MATCH and bound nodes)

$$r = \pi_{\mathtt{x}}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{a}:\ \mathtt{A})} \bowtie \bigcirc_{(\mathtt{b}:\ \mathtt{C})}\Big) \text{⟕} \not\equiv\Big(\uparrow^{(\mathtt{b}:\ \mathtt{C})}_{(\mathtt{x})}[\_\mathtt{e531}]\Big(\bigcirc_{(\mathtt{x})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (OPTIONAL MATCH and bound nodes)


$\pi_{\mathtt{x}}$
⟨x⟩
⟕{b}
⟨a, b, x, _e531⟩
≢
⟨a, b⟩
≢
⟨x, _e531, b⟩
⋈ {}
⟨a, b⟩
$\uparrow^{(\mathtt{b}:\ \mathtt{C})}_{(\mathtt{x})}[\_\mathtt{e531}]$
⟨x, _e531, b⟩
$\bigcirc_{(\mathtt{a}:\ \mathtt{A})}$
⟨a⟩
$\bigcirc_{(\mathtt{b}:\ \mathtt{C})}$
⟨b⟩
$\bigcirc_{(\mathtt{x})}$
⟨x⟩

Incremental relational algebra tree (OPTIONAL MATCH and bound nodes)



A.27.9 OPTIONAL MATCH with labels on the optional end node

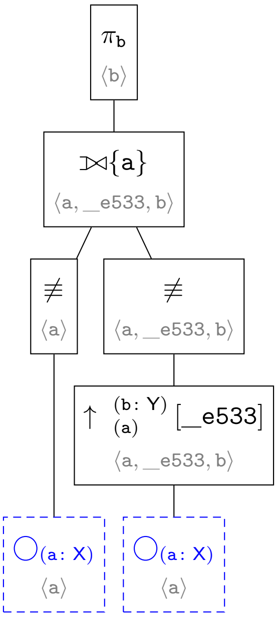
Query specification (OPTIONAL MATCH with labels on the optional end node)

```
MATCH (a:X)
OPTIONAL MATCH (a)-->(b:Y)
RETURN b
```

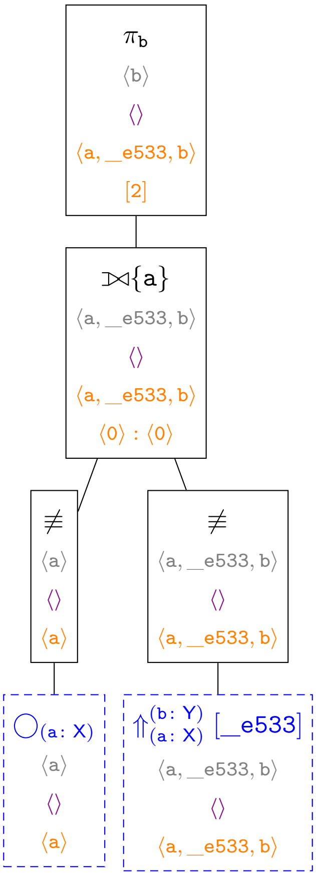
Relational algebra expression for search-based evaluation (OPTIONAL MATCH with labels on the optional end node)

$$r = \pi_{\mathtt{b}}\Big(\not\equiv \Big(\bigcirc_{(\mathtt{a}:\,\mathsf{X})} \Big) ⟕ \not\equiv \Big(\uparrow\,{}^{(\mathtt{b}:\,\mathsf{Y})}_{(\mathtt{a})}\,[\_\mathsf{e533}] \Big(\bigcirc_{(\mathtt{a}:\,\mathsf{X})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (OPTIONAL MATCH with labels on the optional end node)



Incremental relational algebra tree (OPTIONAL MATCH with labels on the optional end node)



A.27.10 Named paths inside optional matches with node predicates

Query specification (Named paths inside optional matches with node predicates)

```
MATCH (a:A), (b:B)
OPTIONAL MATCH p = (a)-[:X]->(b)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.27.11 Variable length optional relationships

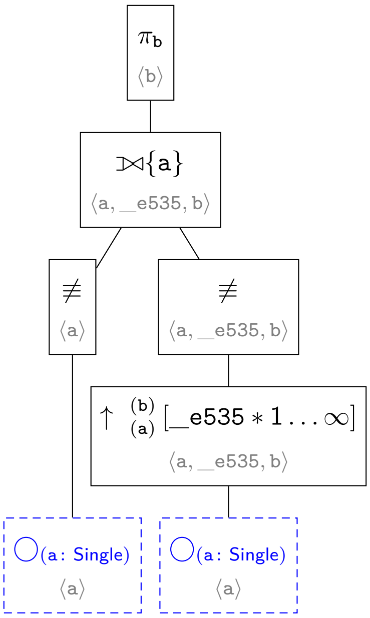
Query specification (Variable length optional relationships)

```
MATCH (a:Single)
OPTIONAL MATCH (a)-[*]->(b)
RETURN b
```

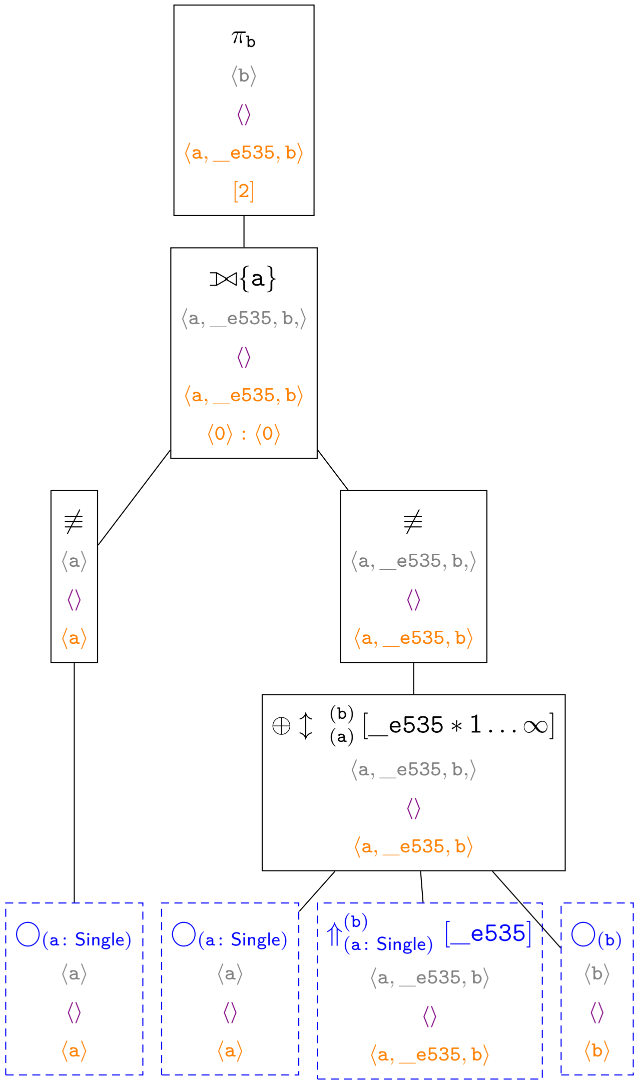
Relational algebra expression for search-based evaluation (Variable length optional relationships)

$$r = \pi_{\mathsf{b}}\Big(\not\equiv\Big(\bigcirc_{(\mathsf{a}:\ \mathsf{Single})}\Big) ⟕ \not\equiv\Big(\uparrow\ ^{(\mathsf{b})}_{(\mathsf{a})}\left[\_\mathsf{e535} * 1 \ldots \infty\right]\Big(\bigcirc_{(\mathsf{a}:\ \mathsf{Single})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Variable length optional relationships)


π_{b}
⟨b⟩
⟕{a}
⟨a, __e535, b⟩
≢
⟨a⟩
≢
⟨a, __e535, b⟩
$\uparrow\ ^{(\mathsf{b})}_{(\mathsf{a})}\left[\_\mathsf{e535} * 1 \ldots \infty\right]$
⟨a, __e535, b⟩
$\bigcirc_{(\mathsf{a}:\ \mathsf{Single})}$
⟨a⟩
$\bigcirc_{(\mathsf{a}:\ \mathsf{Single})}$
⟨a⟩

Incremental relational algebra tree (Variable length optional relationships)



A.27.12 Variable length optional relationships with length predicates

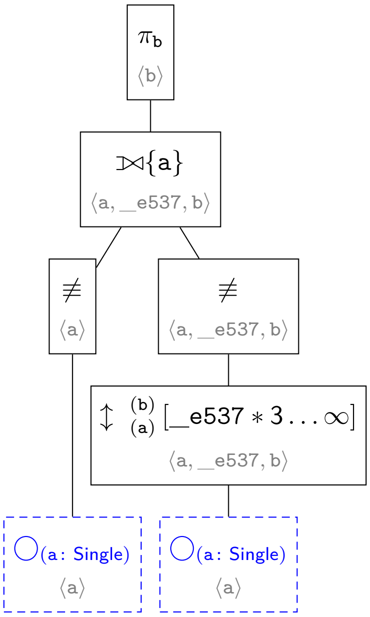
Query specification (Variable length optional relationships with length predicates)

```
MATCH (a:Single)
OPTIONAL MATCH (a)-[*3..]-(b)
RETURN b
```

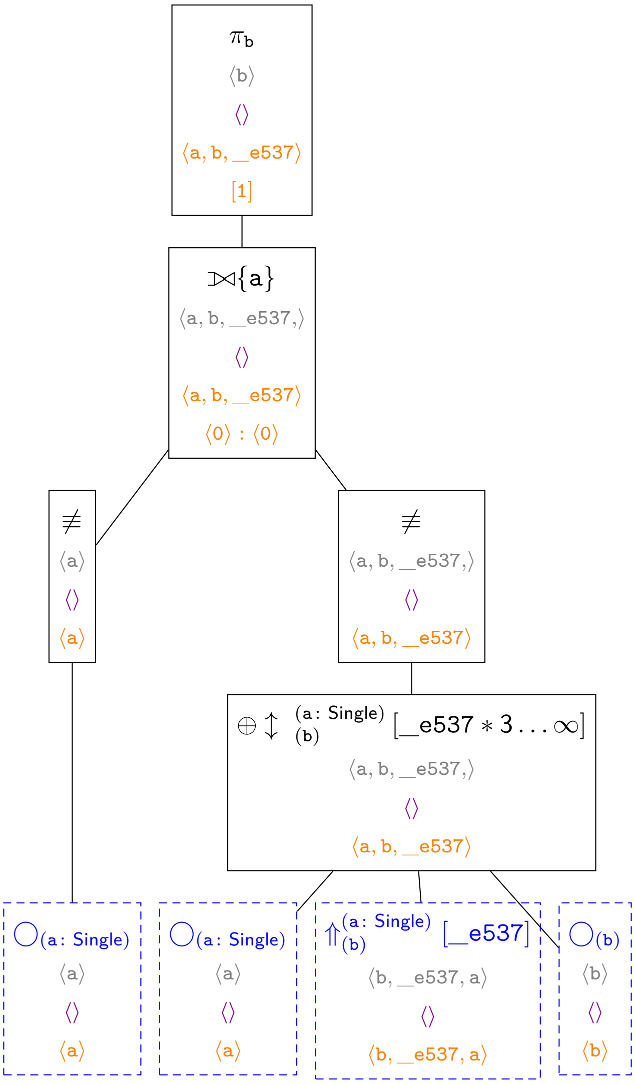
Relational algebra expression for search-based evaluation (Variable length optional relationships with length predicates)

$$r = \pi_{\mathsf{b}}\Big(\not\equiv \Big(\bigcirc_{(\mathsf{a}:\ \mathsf{Single})} \Big) \bowtie \not\equiv \Big(\updownarrow\ {}^{(\mathsf{b})}_{(\mathsf{a})}\, [\_\mathsf{e537} * 3 \ldots \infty] \Big(\bigcirc_{(\mathsf{a}:\ \mathsf{Single})} \Big) \Big) \Big)$$

Relational algebra tree for search-based evaluation (Variable length optional relationships with length predicates)



Incremental relational algebra tree (Variable length optional relationships with length predicates)



A.27.13 Optionally matching self-loops

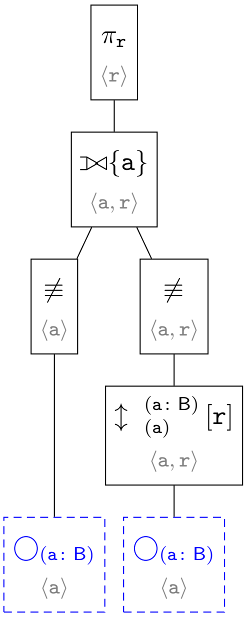
Query specification (Optionally matching self-loops)

```
MATCH (a:B)
OPTIONAL MATCH (a)-[r]-(a)
RETURN r
```

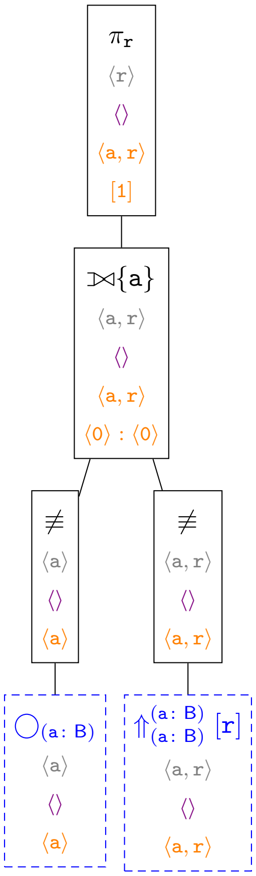
Relational algebra expression for search-based evaluation (Optionally matching self-loops)

$$r = \pi_{\mathtt{r}}\Big(\not\equiv \Big(\bigcirc_{(\mathtt{a}:\ \mathtt{B})} \Big) ⟕ \not\equiv \Big(\updownarrow \, {}^{(\mathtt{a}:\ \mathtt{B})}_{(\mathtt{a})} \, [\mathtt{r}] \Big(\bigcirc_{(\mathtt{a}:\ \mathtt{B})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Optionally matching self-loops)



Incremental relational algebra tree (Optionally matching self-loops)



A.27.14 Optionally matching self-loops without matches

Query specification (Optionally matching self-loops without matches)

```
MATCH (a)
WHERE NOT (a:B)
OPTIONAL MATCH (a)-[r]->(a)
```

```
RETURN r
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.27.15 Variable length optional relationships with bound nodes

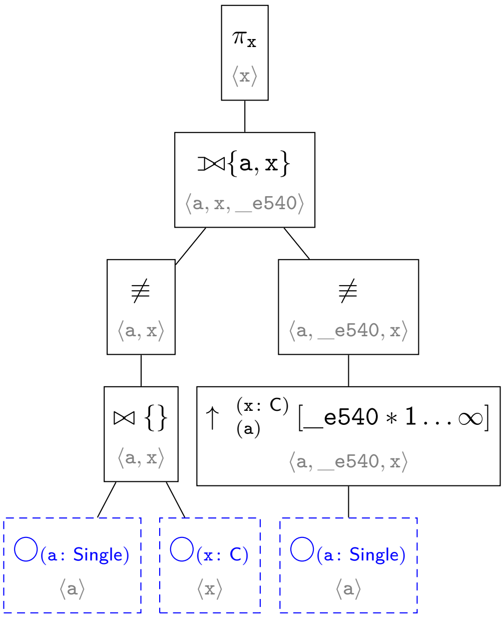
Query specification (Variable length optional relationships with bound nodes)

```
MATCH (a:Single), (x:C)
OPTIONAL MATCH (a)-[*]->(x)
RETURN x
```

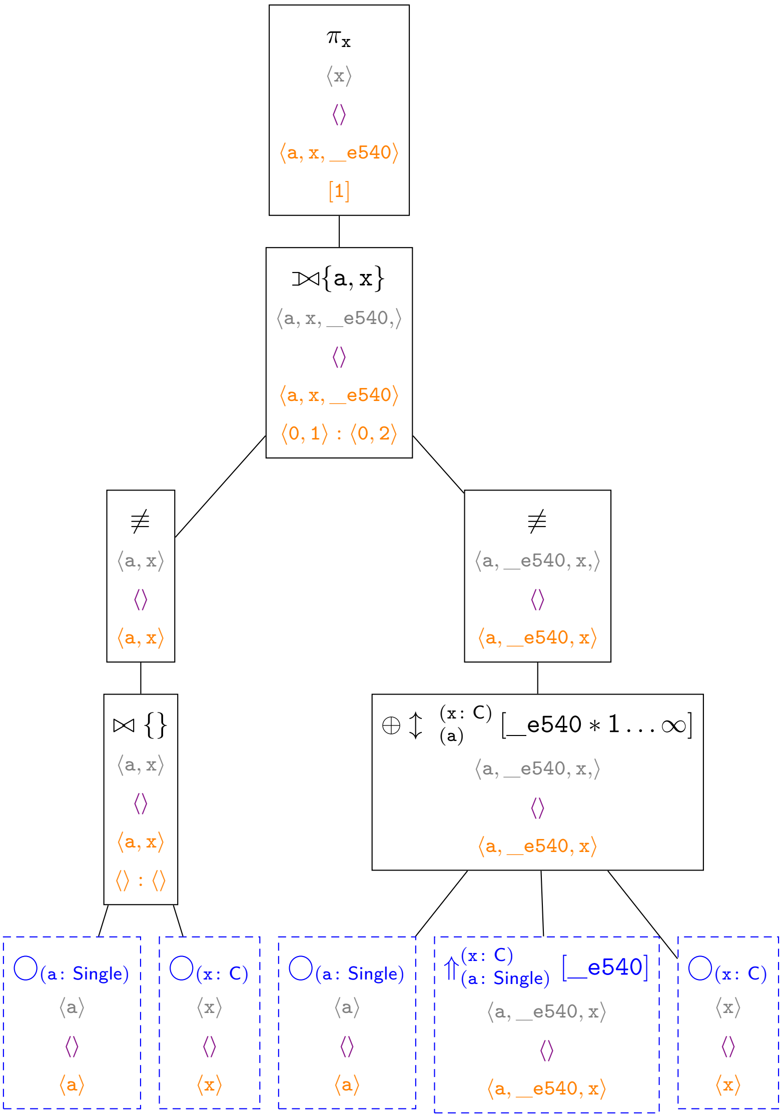
Relational algebra expression for search-based evaluation (Variable length optional relationships with bound nodes)

$$r = \pi_{\mathrm{x}}\Big(\not\equiv \Big(\bigcirc_{(\mathtt{a}:\ \mathsf{Single})} \bowtie \bigcirc_{(\mathtt{x}:\ \mathsf{C})}\Big) ⟕ \not\equiv \Big(\uparrow {}^{(\mathtt{x}:\ \mathsf{C})}_{(\mathtt{a})} [\_\mathsf{e540} * 1 \ldots \infty] \Big(\bigcirc_{(\mathtt{a}:\ \mathsf{Single})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Variable length optional relationships with bound nodes)



Incremental relational algebra tree (Variable length optional relationships with bound nodes)



A.27.16 Variable length optional relationships with bound nodes, no matches

Query specification (Variable length optional relationships with bound nodes, no matches)

```
MATCH (a:A), (b:B)
OPTIONAL MATCH p = (a)-[*]->(b)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.27.17 Longer pattern with bound nodes

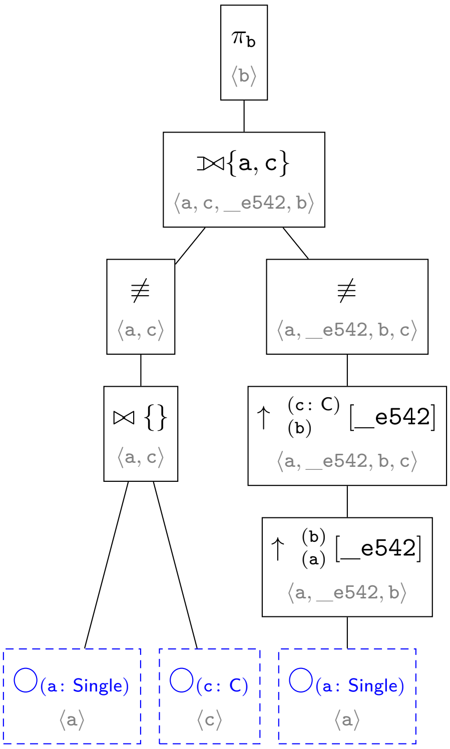
Query specification (Longer pattern with bound nodes)

```
MATCH (a:Single), (c:C)
OPTIONAL MATCH (a)-->(b)-->(c)
RETURN b
```

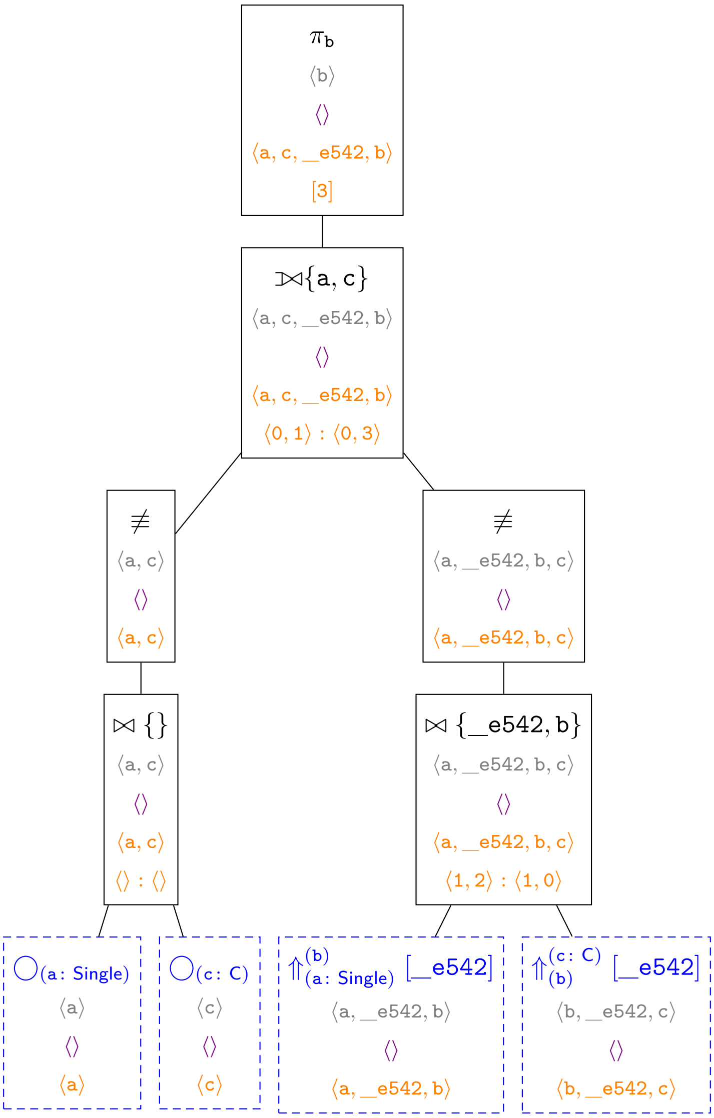
Relational algebra expression for search-based evaluation (Longer pattern with bound nodes)

$$r = \pi_{\mathsf{b}}\Big(\not\equiv \Big(\bigcirc_{(\mathsf{a}\colon \mathsf{Single})} \bowtie \bigcirc_{(\mathsf{c}\colon \mathsf{C})} \Big) ⟕ \not\equiv \Big(\uparrow\, {}^{(\mathsf{c}\colon\,\mathsf{C})}_{(\mathsf{b})} [\_\mathsf{e542}] \Big(\uparrow\, {}^{(\mathsf{b})}_{(\mathsf{a})} [\_\mathsf{e542}] \Big(\bigcirc_{(\mathsf{a}\colon \mathsf{Single})} \Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Longer pattern with bound nodes)



Incremental relational algebra tree (Longer pattern with bound nodes)



A.27.18 Longer pattern with bound nodes without matches

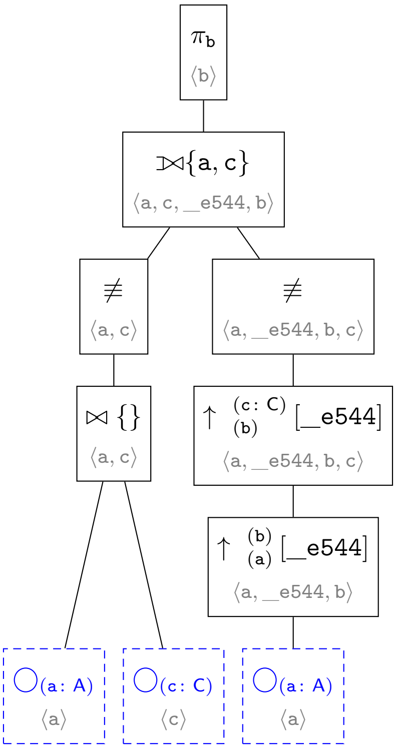
Query specification (Longer pattern with bound nodes without matches)

```
MATCH (a:A), (c:C)
OPTIONAL MATCH (a)-->(b)-->(c)
RETURN b
```

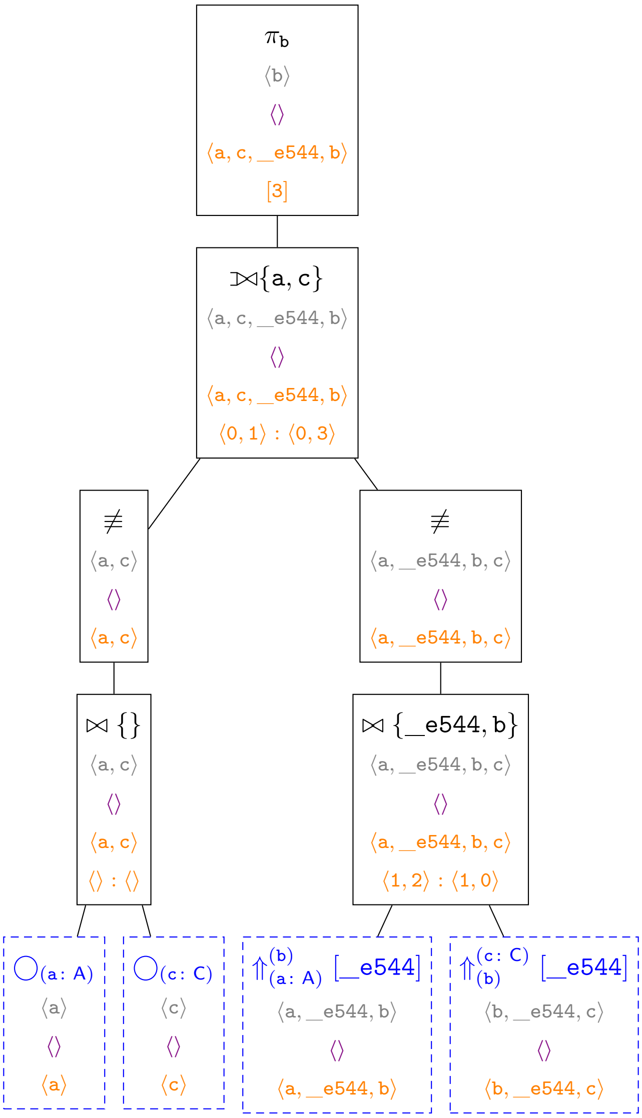
Relational algebra expression for search-based evaluation (Longer pattern with bound nodes without matches)

$$r = \pi_{\mathtt{b}}\Big(\not\equiv \Big(\bigcirc_{(\mathtt{a}:\ \mathtt{A})} \bowtie \bigcirc_{(\mathtt{c}:\ \mathtt{C})} \Big) ⟕ \not\equiv \Big(\uparrow\, {}^{(\mathtt{c}:\ \mathtt{C})}_{(\mathtt{b})} \, [\_\mathtt{e544}] \Big(\uparrow\, {}^{(\mathtt{b})}_{(\mathtt{a})} \, [\_\mathtt{e544}] \Big(\bigcirc_{(\mathtt{a}:\ \mathtt{A})} \Big) \Big) \Big) \Big)$$

Relational algebra tree for search-based evaluation (Longer pattern with bound nodes without matches)



Incremental relational algebra tree (Longer pattern with bound nodes without matches)



A.27.19 Handling correlated optional matches; first does not match implies second does not match

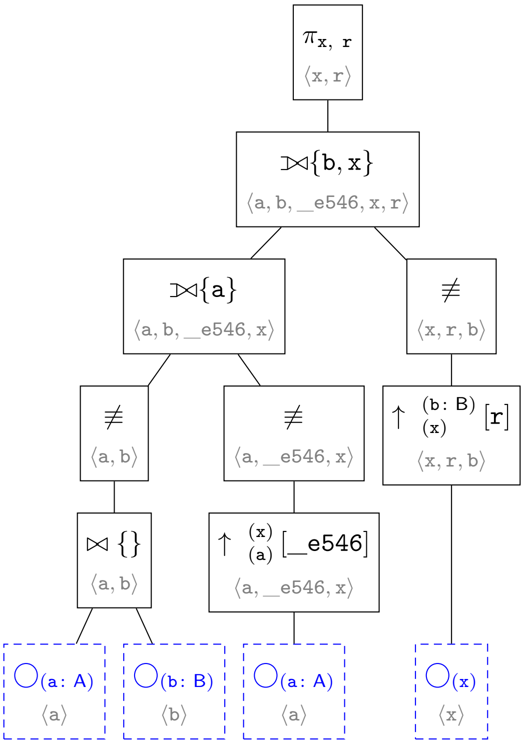
Query specification (Handling correlated optional matches; first does not match implies second does not match)

```
MATCH (a:A), (b:B)
OPTIONAL MATCH (a)-->(x)
OPTIONAL MATCH (x)-[r]->(b)
RETURN x, r
```

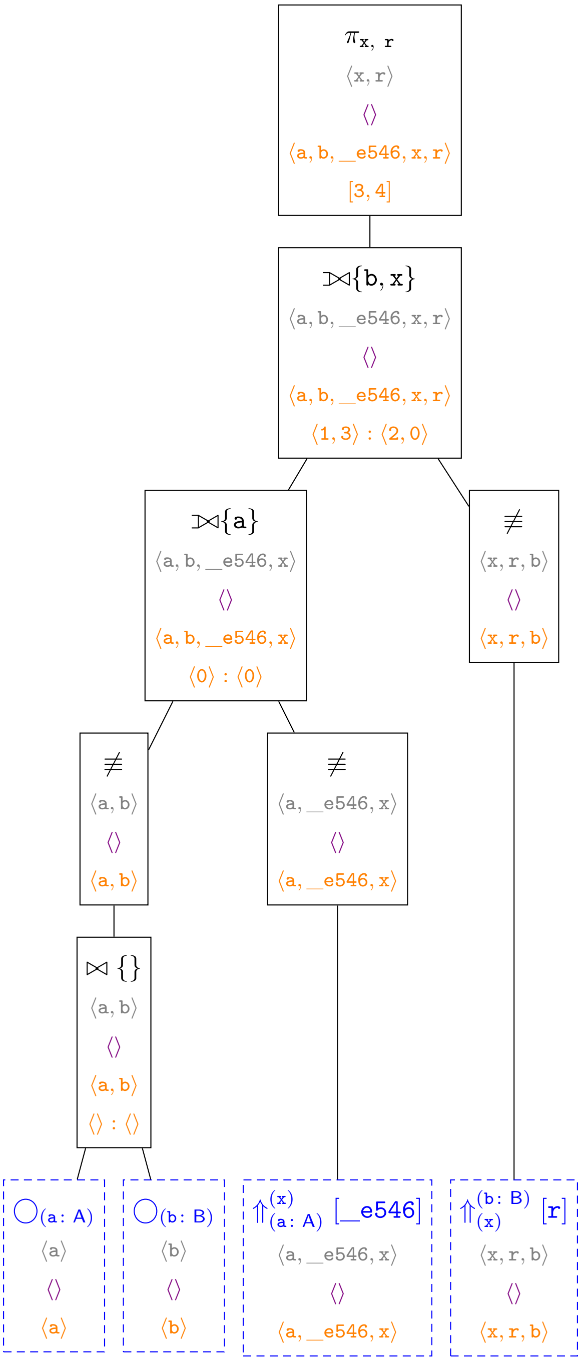
Relational algebra expression for search-based evaluation (Handling correlated optional matches; first does not match implies second does not match)

$$r = \pi_{\mathtt{x},\,\mathtt{r}}\Big(\not\equiv \big(\bigcirc_{(\mathtt{a}:\,\mathtt{A})} \bowtie \bigcirc_{(\mathtt{b}:\,\mathtt{B})} \big) \mathbin{⟕} \not\equiv \Big(\uparrow^{(\mathtt{x})}_{(\mathtt{a})} [\_\mathtt{e546}] \Big(\bigcirc_{(\mathtt{a}:\,\mathtt{A})} \Big) \Big) \mathbin{⟕} \not\equiv \Big(\uparrow^{(\mathtt{b}:\,\mathtt{B})}_{(\mathtt{x})} [\mathtt{r}] \Big(\bigcirc_{(\mathtt{x})} \Big) \Big) \Big)$$

Relational algebra tree for search-based evaluation (Handling correlated optional matches; first does not match implies second does not match)



Incremental relational algebra tree (Handling correlated optional matches; first does not match implies second does not match)



A.27.20 Handling optional matches between optionally matched entities

Query specification (Handling optional matches between optionally matched entities)

```
OPTIONAL MATCH (a:NotThere)
WITH a
MATCH (b:B)
WITH a, b
OPTIONAL MATCH (b)-[r:NOR_THIS]->(a)
RETURN a, b, r
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.27.21 Handling optional matches between nulls

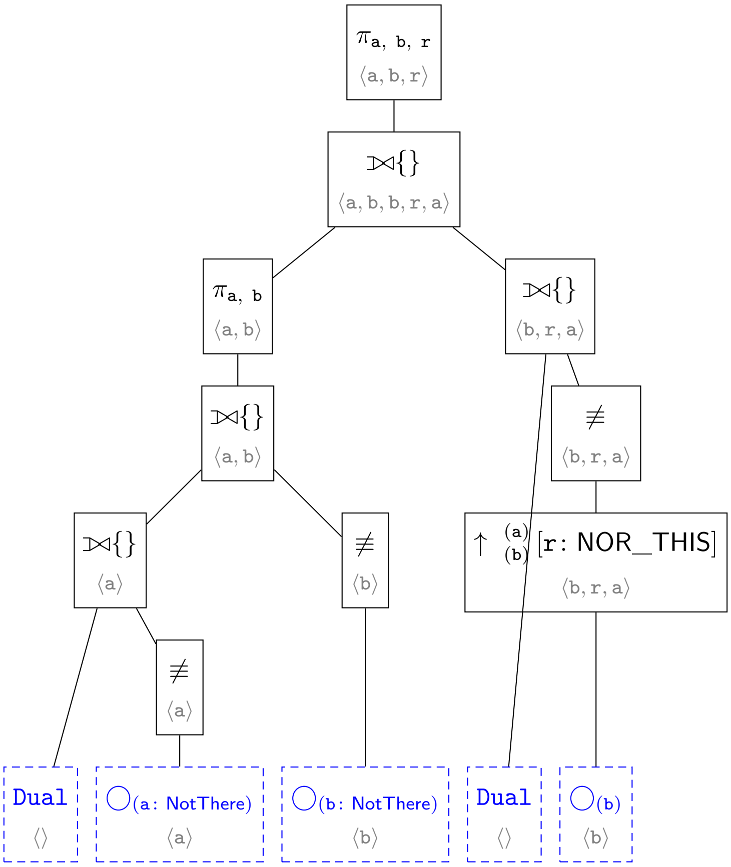
Query specification (Handling optional matches between nulls)

```
OPTIONAL MATCH (a:NotThere)
OPTIONAL MATCH (b:NotThere)
WITH a, b
OPTIONAL MATCH (b)-[r:NOR_THIS]->(a)
RETURN a, b, r
```

Relational algebra expression for search-based evaluation (Handling optional matches between nulls)

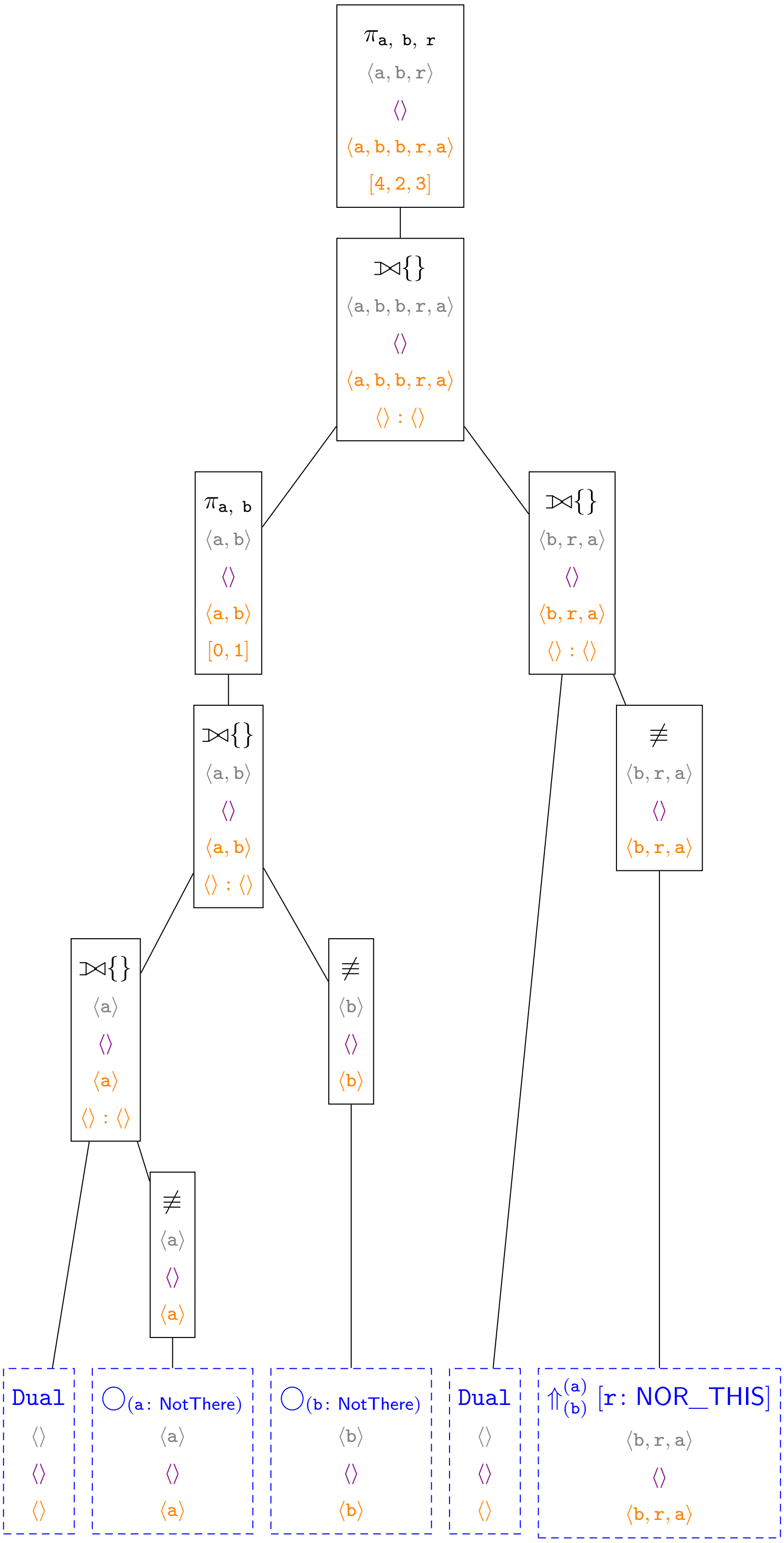
$$r = \pi_{\texttt{a, b, r}}\left(\pi_{\texttt{a, b}}\left(\texttt{Dual} ⟕ \not\equiv \left(\bigcirc_{(\texttt{a}:\ \textsf{NotThere})}\right) ⟕ \not\equiv \left(\bigcirc_{(\texttt{b}:\ \textsf{NotThere})}\right)\right) ⟕ \texttt{Dual} ⟕ \not\equiv \left(\uparrow^{(\texttt{a})}_{(\texttt{b})} [\texttt{r}:\ \textsf{NOR\_THIS}] \left(\bigcirc_{(\texttt{b})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Handling optional matches between nulls)



$\pi_{\texttt{a, b, r}}$
⟨a, b, r⟩
⟕{}
⟨a, b, b, r, a⟩
$\pi_{\texttt{a, b}}$
⟨a, b⟩
⟕{}
⟨b, r, a⟩
⟕{}
⟨a, b⟩
≢
⟨b, r, a⟩
⟕{}
⟨a⟩
≢
⟨b⟩
$\uparrow^{(\texttt{a})}_{(\texttt{b})} [\texttt{r}:\ \textsf{NOR\_THIS}]$
⟨b, r, a⟩
≢
⟨a⟩
Dual
⟨⟩
$\bigcirc_{(\texttt{a}:\ \textsf{NotThere})}$
⟨a⟩
$\bigcirc_{(\texttt{b}:\ \textsf{NotThere})}$
⟨b⟩
Dual
⟨⟩
$\bigcirc_{(\texttt{b})}$
⟨b⟩

Incremental relational algebra tree (Handling optional matches between nulls)



A.27.22 OPTIONAL MATCH and 'collect()'

Query specification (OPTIONAL MATCH and 'collect()')

```
OPTIONAL MATCH (f:DoesExist)
OPTIONAL MATCH (n:DoesNotExist)
```

```
RETURN collect(DISTINCT n.property) AS a, collect(DISTINCT f.property) AS b
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.28 OrderByAcceptance

Progress:

5 of 16

A.28.1 ORDER BY should return results in ascending order

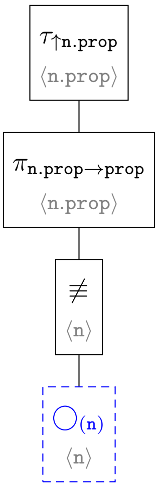
Query specification (ORDER BY should return results in ascending order)

```
MATCH (n)
RETURN n.prop AS prop
ORDER BY n.prop
```

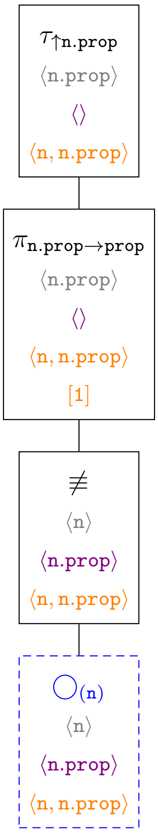
Relational algebra expression for search-based evaluation (ORDER BY should return results in ascending order)

$$r = \tau_{\uparrow \mathtt{n.prop}}\Big(\pi_{\mathtt{n.prop}\rightarrow\mathtt{prop}}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (ORDER BY should return results in ascending order)



Incremental relational algebra tree (ORDER BY should return results in ascending order)


$\tau_{\uparrow \texttt{n.prop}}$
⟨n.prop⟩
⟨⟩
⟨n, n.prop⟩
$\pi_{\texttt{n.prop}\rightarrow\texttt{prop}}$
⟨n.prop⟩
⟨⟩
⟨n, n.prop⟩
[1]
$\not\equiv$
⟨n⟩
⟨n.prop⟩
⟨n, n.prop⟩
$\bigcirc_{(\texttt{n})}$
⟨n⟩
⟨n.prop⟩
⟨n, n.prop⟩

A.28.2 ORDER BY DESC should return results in descending order

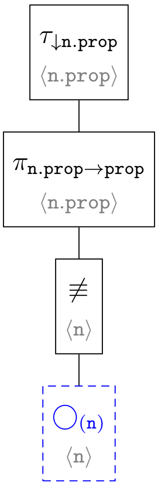
Query specification (ORDER BY DESC should return results in descending order)

```
MATCH (n)
RETURN n.prop AS prop
ORDER BY n.prop DESC
```

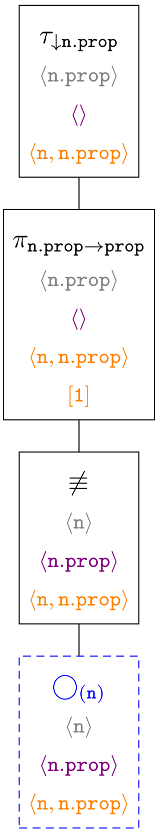
Relational algebra expression for search-based evaluation (ORDER BY DESC should return results in descending order)

$$r = \tau_{\downarrow \texttt{n.prop}}\left(\pi_{\texttt{n.prop}\rightarrow\texttt{prop}}\left(\not\equiv\left(\bigcirc_{(\texttt{n})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (ORDER BY DESC should return results in descending order)



Incremental relational algebra tree (ORDER BY DESC should return results in descending order)



A.28.3 ORDER BY of a column introduced in RETURN should return salient results in ascending order

Query specification (ORDER BY of a column introduced in RETURN should return salient results in ascending order)

```
WITH [0, 1] AS prows, [[2], [3, 4]] AS qrows
UNWIND prows AS p
UNWIND qrows[p] AS q
WITH p, count(q) AS rng
```

```
RETURN p
ORDER BY rng
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.28.4 Renaming columns before ORDER BY should return results in ascending order

Query specification (Renaming columns before ORDER BY should return results in ascending order)

```
MATCH (n)
RETURN n.prop AS n
ORDER BY n + 2
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.28.5 Handle projections with ORDER BY - GH no.4937

Query specification (Handle projections with ORDER BY - GH no.4937)

```
MATCH (c:Crew {name: 'Neo'})
WITH c, 0 AS relevance
RETURN c.rank AS rank
ORDER BY relevance, c.rank
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.28.6 ORDER BY should order booleans in the expected order

Query specification (ORDER BY should order booleans in the expected order)

```
UNWIND [true, false] AS bools
RETURN bools
ORDER BY bools
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.28.7 ORDER BY DESC should order booleans in the expected order

Query specification (ORDER BY DESC should order booleans in the expected order)

```
UNWIND [true, false] AS bools
RETURN bools
ORDER BY bools DESC
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.28.8 ORDER BY should order strings in the expected order

Query specification (ORDER BY should order strings in the expected order)

```
UNWIND ['.*', '', ' ', 'one'] AS strings
RETURN strings
ORDER BY strings
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.28.9 ORDER BY DESC should order strings in the expected order

Query specification (ORDER BY DESC should order strings in the expected order)

```
UNWIND ['.*', '', ' ', 'one'] AS strings
RETURN strings
ORDER BY strings DESC
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.28.10 ORDER BY should order ints in the expected order

Query specification (ORDER BY should order ints in the expected order)

```
UNWIND [1, 3, 2] AS ints
RETURN ints
ORDER BY ints
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.28.11 ORDER BY DESC should order ints in the expected order

Query specification (ORDER BY DESC should order ints in the expected order)

```
UNWIND [1, 3, 2] AS ints
RETURN ints
ORDER BY ints DESC
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.28.12 ORDER BY should order floats in the expected order

Query specification (ORDER BY should order floats in the expected order)

```
UNWIND [1.5, 1.3, 999.99] AS floats
RETURN floats
ORDER BY floats
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.28.13 ORDER BY DESC should order floats in the expected order

Query specification (ORDER BY DESC should order floats in the expected order)

```
UNWIND [1.5, 1.3, 999.99] AS floats
RETURN floats
ORDER BY floats DESC
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.28.14 Handle ORDER BY with LIMIT 1

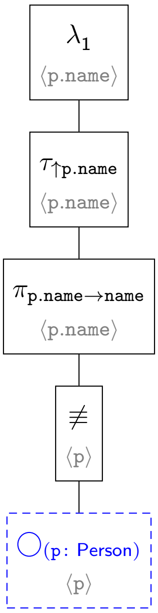
Query specification (Handle ORDER BY with LIMIT 1)

```
MATCH (p:Person)
RETURN p.name AS name
ORDER BY p.name
LIMIT 1
```

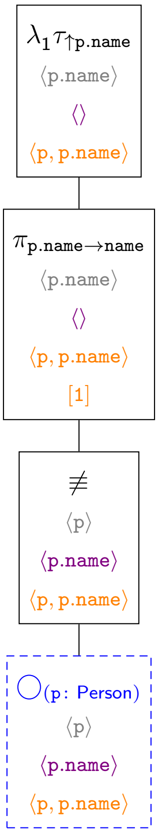
Relational algebra expression for search-based evaluation (Handle ORDER BY with LIMIT 1)

$$r = \lambda_1\left(\tau_{\uparrow \texttt{p.name}}\left(\pi_{\texttt{p.name}\to\texttt{name}}\left(\not\equiv\left(\bigcirc_{(\texttt{p}:\ \textsf{Person})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Handle ORDER BY with LIMIT 1)


λ_1
⟨p.name⟩
$\tau_{\uparrow \texttt{p.name}}$
⟨p.name⟩
$\pi_{\texttt{p.name} \rightarrow \texttt{name}}$
⟨p.name⟩
$\not\equiv$
⟨p⟩
○(p: Person)
⟨p⟩

Incremental relational algebra tree (Handle ORDER BY with LIMIT 1)



A.28.15 ORDER BY with LIMIT 0 should not generate errors

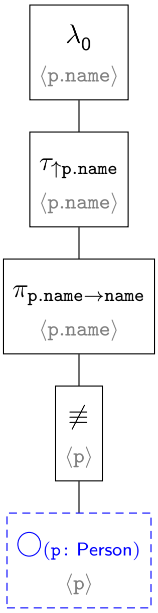
Query specification (ORDER BY with LIMIT 0 should not generate errors)

```
MATCH (p:Person)
RETURN p.name AS name
ORDER BY p.name
LIMIT 0
```

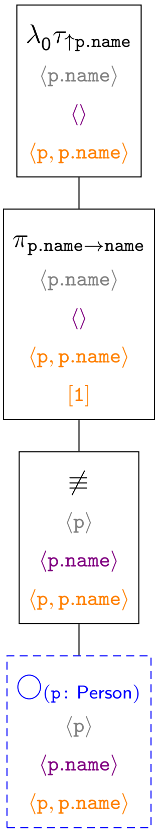
Relational algebra expression for search-based evaluation (ORDER BY with LIMIT 0 should not generate errors)

$$r = \lambda_{0}\Big(\tau_{\uparrow\mathtt{p.name}}\Big(\pi_{\mathtt{p.name}\rightarrow\mathtt{name}}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{p}:\ \mathsf{Person})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (ORDER BY with LIMIT 0 should not generate errors)


λ_0
⟨p.name⟩
$\tau_{\uparrow\mathtt{p.name}}$
⟨p.name⟩
$\pi_{\mathtt{p.name}\rightarrow\mathtt{name}}$
⟨p.name⟩
$\not\equiv$
⟨p⟩
$\bigcirc_{(\mathtt{p}:\ \mathsf{Person})}$
⟨p⟩

Incremental relational algebra tree (ORDER BY with LIMIT 0 should not generate errors)


$\lambda_0\tau_{\uparrow\mathtt{p.name}}$
⟨p.name⟩
⟨⟩
⟨p, p.name⟩
$\pi_{\mathtt{p.name}\rightarrow\mathtt{name}}$
⟨p.name⟩
⟨⟩
⟨p, p.name⟩
[1]
$\not\equiv$
⟨p⟩
⟨p.name⟩
⟨p, p.name⟩
$\bigcirc_{(\mathtt{p}:\ \mathsf{Person})}$
⟨p⟩
⟨p.name⟩
⟨p, p.name⟩

A.28.16 ORDER BY with negative parameter for LIMIT should not generate errors

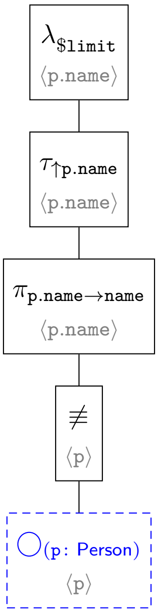
Query specification (ORDER BY with negative parameter for LIMIT should not generate errors)

```
MATCH (p:Person)
RETURN p.name AS name
ORDER BY p.name
LIMIT $limit
```

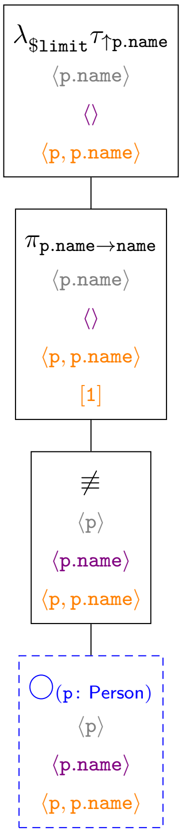
Relational algebra expression for search-based evaluation (ORDER BY with negative parameter for LIMIT should not generate errors)

$$r = \lambda_{\texttt{\$limit}}\left(\tau_{\uparrow\texttt{p.name}}\left(\pi_{\texttt{p.name}\rightarrow\texttt{name}}\left(\not\equiv\left(\bigcirc_{(\texttt{p}:\ \textsf{Person})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (ORDER BY with negative parameter for LIMIT should not generate errors)



Incremental relational algebra tree (ORDER BY with negative parameter for LIMIT should not generate errors)



A.29 PatternComprehension

Progress:

0 of 15

A.29.1 Pattern comprehension and ORDER BY

Query specification (Pattern comprehension and ORDER BY)

```
MATCH (liker)
RETURN [p = (liker)--() | p] AS isNew
ORDER BY liker.time
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.2 Returning a pattern comprehension

Query specification (Returning a pattern comprehension)

```
MATCH (n)
RETURN [p = (n)-->() | p] AS ps
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.3 Returning a pattern comprehension with label predicate

Query specification (Returning a pattern comprehension with label predicate)

```
MATCH (n:A)
RETURN [p = (n)-->(:B) | p]
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.4 Returning a pattern comprehension with bound nodes

Query specification (Returning a pattern comprehension with bound nodes)

```
MATCH (a:A), (b:B)
RETURN [p = (a)-[*]->(b) | p] AS paths
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.5 Using a pattern comprehension in a WITH

Query specification (Using a pattern comprehension in a WITH)

```
MATCH (n)-->(b)
WITH [p = (n)-->() | p] AS ps, count(b) AS c
RETURN ps, c
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.6 Using a variable-length pattern comprehension in a WITH

Query specification (Using a variable-length pattern comprehension in a WITH)

```
MATCH (a:A), (b:B)
WITH [p = (a)-[*]->(b) | p] AS paths, count(a) AS c
RETURN paths, c
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.7 Using pattern comprehension in RETURN

Query specification (Using pattern comprehension in RETURN)

```
MATCH (n:A)
RETURN [p = (n)-[:HAS]->() | p] AS ps
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.8 Aggregating on pattern comprehension

Query specification (Aggregating on pattern comprehension)

```
MATCH (n:A)
RETURN count([p = (n)-[:HAS]->() | p]) AS c
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.9 Using pattern comprehension to test existence

Query specification (Using pattern comprehension to test existence)

```
MATCH (n:X)
RETURN n, size([(n)--() | 1]) > 0 AS b
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.10 Pattern comprehension inside list comprehension

Query specification (Pattern comprehension inside list comprehension)

```
MATCH p = (n:X)-->(b)
RETURN n, [x IN nodes(p) | size([(x)-->(:Y) | 1])] AS list
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.11 Get node degree via size of pattern comprehension

Query specification (Get node degree via size of pattern comprehension)

```
MATCH (a:X)
RETURN size([(a)-->() | 1]) AS length
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.12 Get node degree via size of pattern comprehension that specifies a relationship type

Query specification (Get node degree via size of pattern comprehension that specifies a relationship type)

```
MATCH (a:X)
RETURN size([(a)-[:T]->() | 1]) AS length
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.13 Get node degree via size of pattern comprehension that specifies multiple relationship types

Query specification **(Get node degree via size of pattern comprehension that specifies multiple relationship types)**

```
MATCH (a:X)
RETURN size([(a)-[:T|OTHER]->() | 1]) AS length
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.14 Introducing new node variable in pattern comprehension

Query specification **(Introducing new node variable in pattern comprehension)**

```
MATCH (n)
RETURN [(n)-[:T]->(b) | b.prop] AS list
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.29.15 Introducing new relationship variable in pattern comprehension

Query specification **(Introducing new relationship variable in pattern comprehension)**

```
MATCH (n)
RETURN [(n)-[r:T]->() | r.prop] AS list
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.30 RemoveAcceptance

Progress:

0 of 0

A.31 ReturnAcceptanceTest

Progress:

15 of 15

A.31.1 Allow addition

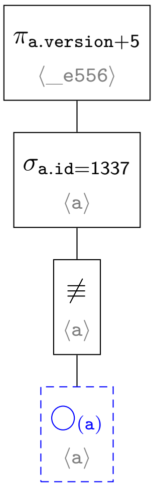
Query specification (Allow addition)

```
MATCH (a)
WHERE a.id = 1337
RETURN a.version + 5
```

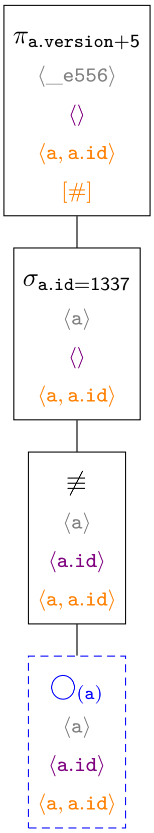
Relational algebra expression for search-based evaluation (Allow addition)

$$r = \pi_{\texttt{a.version}+5}\left(\sigma_{\texttt{a.id}=1337}\left(\not\equiv\left(\bigcirc_{(\texttt{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Allow addition)



Incremental relational algebra tree (Allow addition)



A.31.2 Limit to two hits

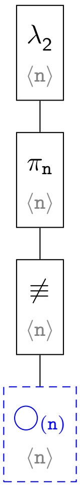
Query specification (Limit to two hits)

```
MATCH (n)
RETURN n
LIMIT 2
```

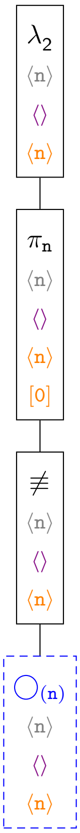
Relational algebra expression for search-based evaluation (Limit to two hits)

$$r = \lambda_2\Big(\pi_{\mathtt{n}}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Limit to two hits)



Incremental relational algebra tree (Limit to two hits)



A.31.3 Start the result from the second row

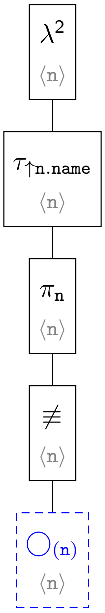
Query specification (Start the result from the second row)

```
MATCH (n)
RETURN n
ORDER BY n.name ASC
SKIP 2
```

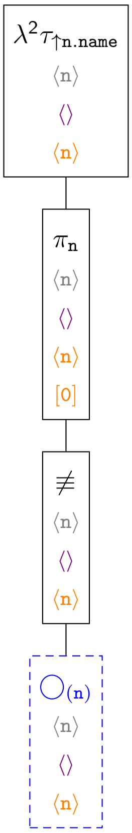
Relational algebra expression for search-based evaluation (Start the result from the second row)

$$r = \lambda^{2}\left(\tau_{\uparrow \texttt{n.name}}\left(\pi_{\texttt{n}}\left(\not\equiv\left(\bigcirc_{\texttt{(n)}}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Start the result from the second row)


λ^{2}
$\langle \texttt{n} \rangle$
$\tau_{\uparrow \texttt{n.name}}$
$\langle \texttt{n} \rangle$
$\pi_{\texttt{n}}$
$\langle \texttt{n} \rangle$
$\not\equiv$
$\langle \texttt{n} \rangle$
$\bigcirc_{\texttt{(n)}}$
$\langle \texttt{n} \rangle$

Incremental relational algebra tree (Start the result from the second row)


$\lambda^2\tau_{\uparrow\texttt{n.name}}$
⟨n⟩
⟨⟩
⟨n⟩
$\pi_{\texttt{n}}$
⟨n⟩
⟨⟩
⟨n⟩
[0]
$\not\equiv$
⟨n⟩
⟨⟩
⟨n⟩
$\bigcirc_{(\texttt{n})}$
⟨n⟩
⟨⟩
⟨n⟩

A.31.4 Start the result from the second row by param

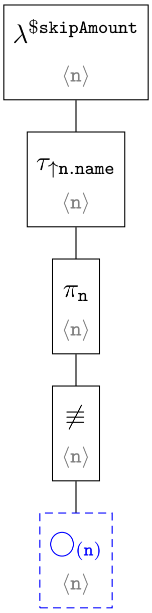
Query specification (Start the result from the second row by param)

```
MATCH (n)
RETURN n
ORDER BY n.name ASC
SKIP $skipAmount
```

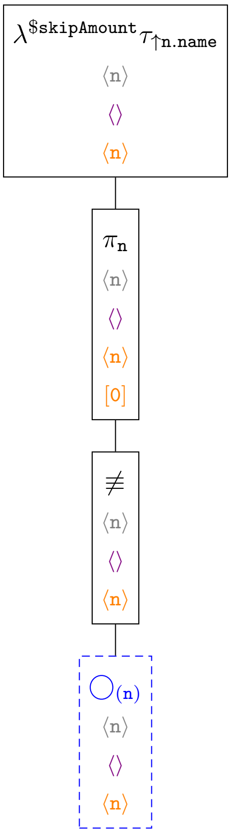
Relational algebra expression for search-based evaluation (Start the result from the second row by param)

$$r = \lambda^{\texttt{\$skipAmount}}\left(\tau_{\uparrow\texttt{n.name}}\left(\pi_{\texttt{n}}\left(\not\equiv\left(\bigcirc_{(\texttt{n})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Start the result from the second row by param)



Incremental relational algebra tree (Start the result from the second row by param)



A.31.5 Get rows in the middle

Query specification (Get rows in the middle)

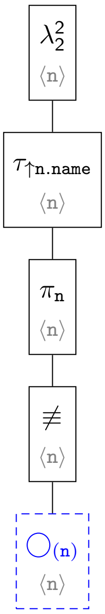
```
MATCH (n)
RETURN n
ORDER BY n.name ASC
```

```
SKIP 2
LIMIT 2
```

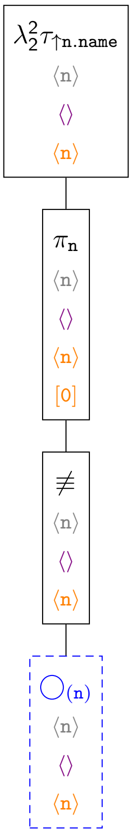
Relational algebra expression for search-based evaluation (Get rows in the middle)

$$r = \lambda_2^2\Big(\tau_{\uparrow\texttt{n.name}}\Big(\pi_{\texttt{n}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{n})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Get rows in the middle)



Incremental relational algebra tree (Get rows in the middle)


$\lambda_2^2\tau_{\uparrow\texttt{n.name}}$
⟨n⟩
⟨⟩
⟨n⟩
$\pi_{\texttt{n}}$
⟨n⟩
⟨⟩
⟨n⟩
[0]
$\not\equiv$
⟨n⟩
⟨⟩
⟨n⟩
$\bigcirc_{(\texttt{n})}$
⟨n⟩
⟨⟩
⟨n⟩

A.31.6 Get rows in the middle by param

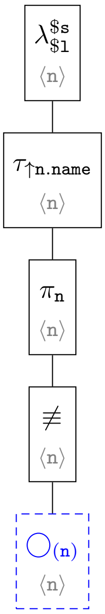
Query specification (Get rows in the middle by param)

```
MATCH (n)
RETURN n
ORDER BY n.name ASC
SKIP $s
LIMIT $l
```

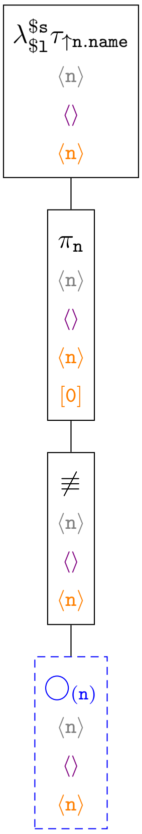
Relational algebra expression for search-based evaluation (Get rows in the middle by param)

$$r = \lambda_{\texttt{\$l}}^{\texttt{\$s}}\left(\tau_{\uparrow\texttt{n.name}}\left(\pi_{\texttt{n}}\left(\not\equiv\left(\bigcirc_{(\texttt{n})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Get rows in the middle by param)


$\lambda_{\texttt{\$l}}^{\texttt{\$s}}$
⟨n⟩
$\tau_{\uparrow\texttt{n.name}}$
⟨n⟩
$\pi_{\texttt{n}}$
⟨n⟩
$\not\equiv$
⟨n⟩
$\bigcirc_{(\texttt{n})}$
⟨n⟩

Incremental relational algebra tree (Get rows in the middle by param)


$\lambda_{\texttt{\$1}}^{\texttt{\$s}}\tau_{\uparrow\texttt{n.name}}$
⟨n⟩
⟨⟩
⟨n⟩
$\pi_{\texttt{n}}$
⟨n⟩
⟨⟩
⟨n⟩
[0]
$\not\equiv$
⟨n⟩
⟨⟩
⟨n⟩
$\bigcirc_{\texttt{(n)}}$
⟨n⟩
⟨⟩
⟨n⟩

A.31.7 Sort on aggregated function

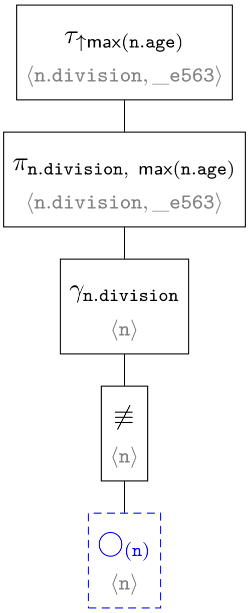
Query specification (Sort on aggregated function)

```
MATCH (n)
RETURN n.division, max(n.age)
ORDER BY max(n.age)
```

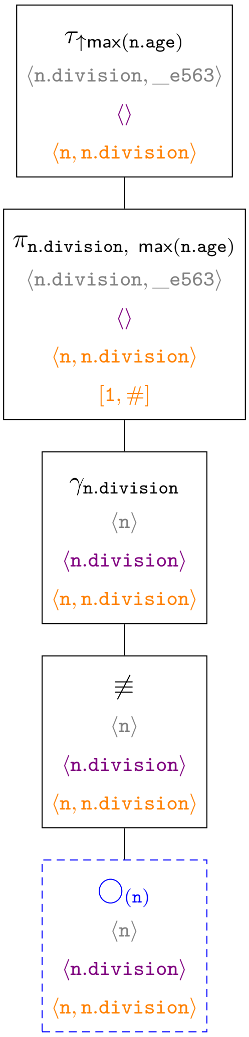
Relational algebra expression for search-based evaluation (Sort on aggregated function)

$$r = \tau_{\uparrow\texttt{max(n.age)}}\left(\pi_{\texttt{n.division, max(n.age)}}\left(\gamma_{\texttt{n.division}}\left(\not\equiv\left(\bigcirc_{\texttt{(n)}}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Sort on aggregated function)


$\tau_{\uparrow\texttt{max(n.age)}}$
⟨n.division, __e563⟩
$\pi_{\texttt{n.division, max(n.age)}}$
⟨n.division, __e563⟩
$\gamma_{\texttt{n.division}}$
⟨n⟩
≢
⟨n⟩
◯(n)
⟨n⟩

Incremental relational algebra tree (Sort on aggregated function)



A.31.8 Support sort and distinct

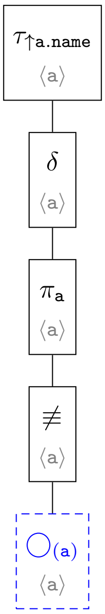
Query specification (Support sort and distinct)

```
MATCH (a)
RETURN DISTINCT a
ORDER BY a.name
```

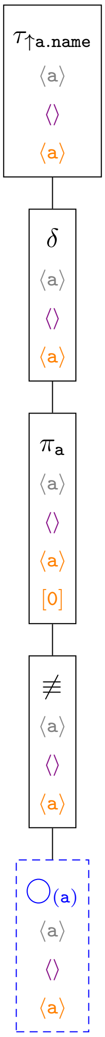
Relational algebra expression for search-based evaluation (Support sort and distinct)

$$r = \tau_{\uparrow \texttt{a.name}}\left(\delta\left(\pi_{\texttt{a}}\left(\not\equiv\left(\bigcirc_{(\texttt{a})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Support sort and distinct)



Incremental relational algebra tree (Support sort and distinct)



A.31.9 Support column renaming

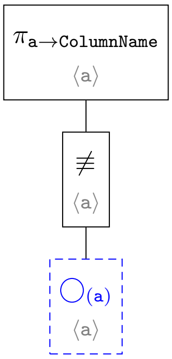
Query specification (Support column renaming)

```
MATCH (a)
RETURN a AS ColumnName
```

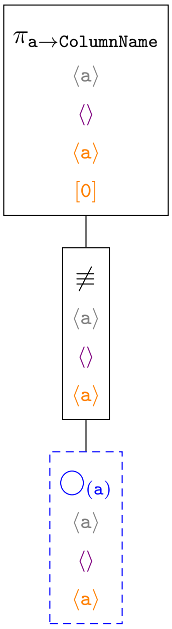
Relational algebra expression for search-based evaluation (Support column renaming)

$$r = \pi_{\texttt{a}\rightarrow\texttt{ColumnName}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{a})}\Big)\Big)$$

Relational algebra tree for search-based evaluation (Support column renaming)



Incremental relational algebra tree (Support column renaming)



A.31.10 Support ordering by a property after being distinct-ified

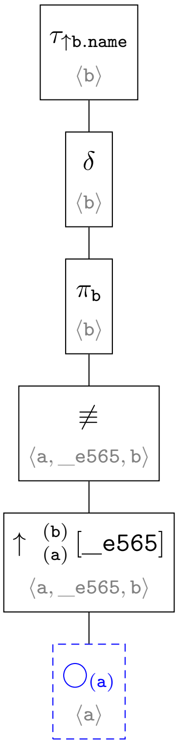
Query specification (Support ordering by a property after being distinct-ified)

```
MATCH (a)-->(b)
RETURN DISTINCT b
ORDER BY b.name
```

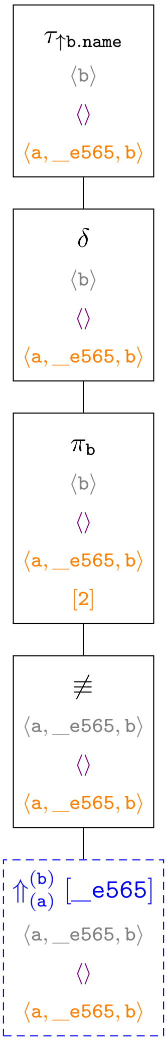
Relational algebra expression for search-based evaluation (Support ordering by a property after being distinct-ified)

$$r = \tau_{\uparrow \mathtt{b.name}}\Big(\delta\Big(\pi_{\mathtt{b}}\Big(\not\equiv\Big(\uparrow{}^{(\mathtt{b})}_{(\mathtt{a})}[\_\mathsf{e565}]\Big(\bigcirc_{(\mathtt{a})}\Big)\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Support ordering by a property after being distinct-ified)


$\tau_{\uparrow \texttt{b.name}}$
⟨b⟩
δ
⟨b⟩
$\pi_{\texttt{b}}$
⟨b⟩
$\not\equiv$
⟨a, _e565, b⟩
$\uparrow\,^{(\texttt{b})}_{(\texttt{a})}$ [_e565]
⟨a, _e565, b⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩

Incremental relational algebra tree (Support ordering by a property after being distinctified)



A.31.11 Arithmetic precedence test

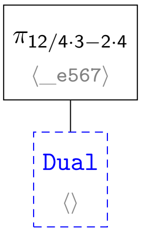
Query specification (Arithmetic precedence test)

```
RETURN 12 / 4 * 3 - 2 * 4
```

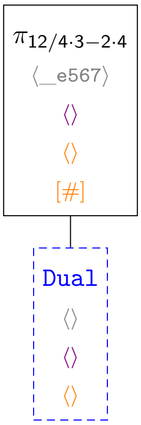
Relational algebra expression for search-based evaluation (Arithmetic precedence test)

$$r = \pi_{12/4\cdot3-2\cdot4}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Arithmetic precedence test)



Incremental relational algebra tree (Arithmetic precedence test)



A.31.12 Arithmetic precedence with parenthesis test

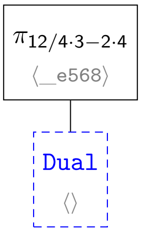
Query specification (Arithmetic precedence with parenthesis test)

```
RETURN 12 / 4 * (3 - 2 * 4)
```

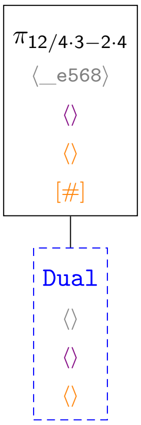
Relational algebra expression for search-based evaluation (Arithmetic precedence with parenthesis test)

$$r = \pi_{12/4 \cdot 3 - 2 \cdot 4}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Arithmetic precedence with parenthesis test)



Incremental relational algebra tree (Arithmetic precedence with parenthesis test)



A.31.13 Count star should count everything in scope

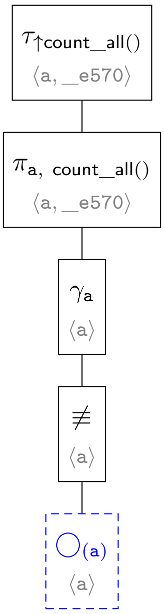
Query specification (Count star should count everything in scope)

```
MATCH (a)
RETURN a, count(*)
ORDER BY count(*)
```

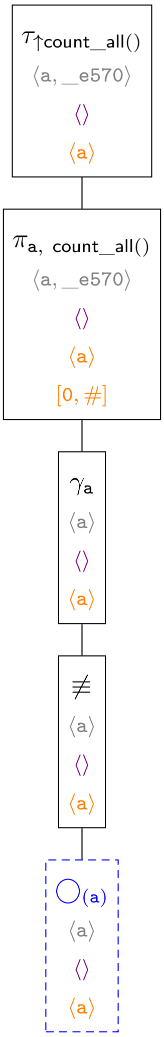
Relational algebra expression for search-based evaluation (Count star should count everything in scope)

$$r = \tau_{\uparrow \mathsf{count\_all()}}\left(\pi_{\mathsf{a},\ \mathsf{count\_all()}}\left(\gamma_{\mathsf{a}}\left(\not\equiv\left(\bigcirc_{(\mathsf{a})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Count star should count everything in scope)


$\tau_{\uparrow \mathsf{count\_all()}}$
⟨a, _e570⟩
$\pi_{\mathsf{a},\ \mathsf{count\_all()}}$
⟨a, _e570⟩
γ_{a}
⟨a⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{(\mathsf{a})}$
⟨a⟩

Incremental relational algebra tree (Count star should count everything in scope)



A.31.14 Absolute function

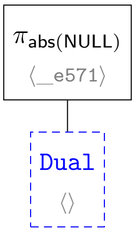
Query specification (Absolute function)

```
RETURN abs(-1)
```

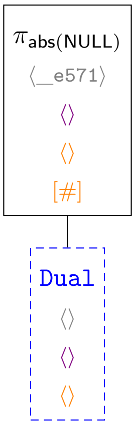
Relational algebra expression for search-based evaluation (Absolute function)

$$r = \pi_{\mathsf{abs(NULL)}}\left(\mathtt{Dual}\right)$$

Relational algebra tree for search-based evaluation (Absolute function)



Incremental relational algebra tree (Absolute function)



A.31.15 Return collection size

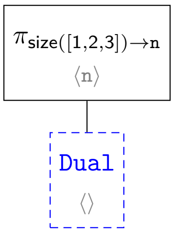
Query specification (Return collection size)

```
RETURN size([1, 2, 3]) AS n
```

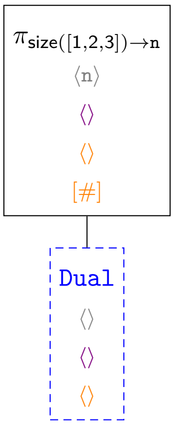
Relational algebra expression for search-based evaluation (Return collection size)

$$r = \pi_{\mathsf{size}([1,2,3]) \rightarrow \mathtt{n}}\Big(\mathtt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Return collection size)



Incremental relational algebra tree (Return collection size)



A.32 ReturnAcceptance2

Progress:

21 of 29

A.32.1 Accept valid Unicode literal

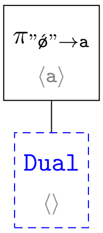
Query specification (Accept valid Unicode literal)

```
RETURN '\u01FF' AS a
```

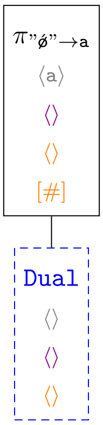
Relational algebra expression for search-based evaluation (Accept valid Unicode literal)

$$r = \pi_{\text{"ǿ"} \rightarrow \mathtt{a}}\Big(\mathtt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Accept valid Unicode literal)



Incremental relational algebra tree (Accept valid Unicode literal)



A.32.2 LIMIT 0 should return an empty result

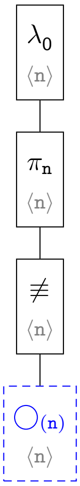
Query specification (LIMIT 0 should return an empty result)

```
MATCH (n)
RETURN n
LIMIT 0
```

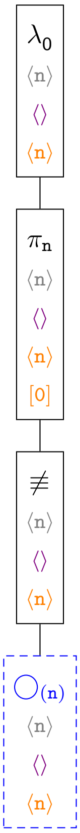
Relational algebra expression for search-based evaluation (LIMIT 0 should return an empty result)

$$r = \lambda_{0}\Big(\pi_{\mathtt{n}}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (LIMIT 0 should return an empty result)



Incremental relational algebra tree (LIMIT 0 should return an empty result)



A.32.3 Ordering with aggregation

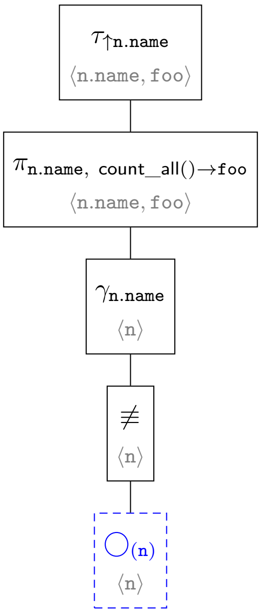
Query specification (Ordering with aggregation)

```
MATCH (n)
RETURN n.name, count(*) AS foo
ORDER BY n.name
```

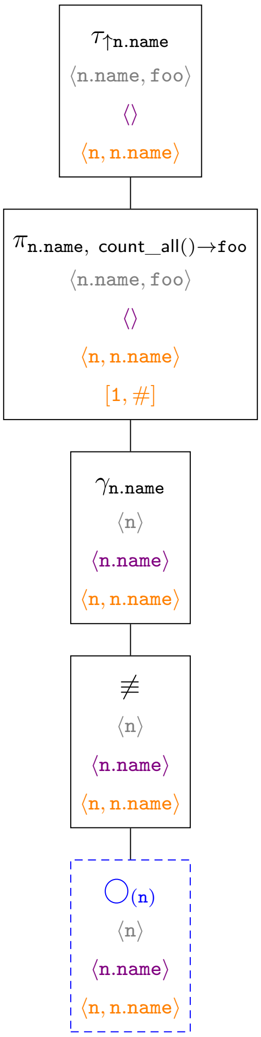
Relational algebra expression for search-based evaluation (Ordering with aggregation)

$$r = \tau_{\uparrow \texttt{n.name}}\left(\pi_{\texttt{n.name},\ \textsf{count\_all}() \rightarrow \texttt{foo}}\left(\gamma_{\texttt{n.name}}\left(\not\equiv\left(\bigcirc_{(\texttt{n})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Ordering with aggregation)


$\tau_{\uparrow \texttt{n.name}}$
⟨n.name, foo⟩
$\pi_{\texttt{n.name, count\_all()}\rightarrow\texttt{foo}}$
⟨n.name, foo⟩
$\gamma_{\texttt{n.name}}$
⟨n⟩
$\not\equiv$
⟨n⟩
$\bigcirc_{(\texttt{n})}$
⟨n⟩

Incremental relational algebra tree (Ordering with aggregation)


$\tau_{\uparrow \texttt{n.name}}$
⟨n.name, foo⟩
⟨⟩
⟨n, n.name⟩
$\pi_{\texttt{n.name, count\_all()}\rightarrow\texttt{foo}}$
⟨n.name, foo⟩
⟨⟩
⟨n, n.name⟩
[1, #]
$\gamma_{\texttt{n.name}}$
⟨n⟩
⟨n.name⟩
⟨n, n.name⟩
$\not\equiv$
⟨n⟩
⟨n.name⟩
⟨n, n.name⟩
$\bigcirc_{(\texttt{n})}$
⟨n⟩
⟨n.name⟩
⟨n, n.name⟩

A.32.4 DISTINCT on nullable values

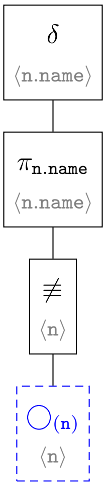
Query specification (DISTINCT on nullable values)

```
MATCH (n)
RETURN DISTINCT n.name
```

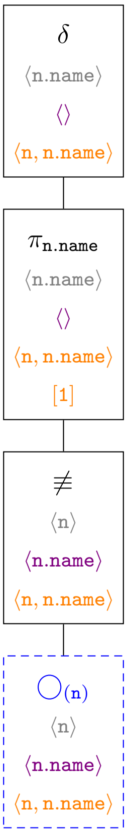
Relational algebra expression for search-based evaluation (DISTINCT on nullable values)

$$r = \delta\Big(\pi_{\texttt{n.name}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{n})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (DISTINCT on nullable values)



Incremental relational algebra tree (DISTINCT on nullable values)



A.32.5 Return all variables

Query specification (Return all variables)

```
MATCH p = (a:Start)-->(b)
RETURN *
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.32.6 'sqrt()' returning float values

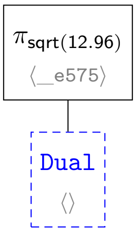
Query specification ('sqrt()' returning float values)

```
RETURN sqrt(12.96)
```

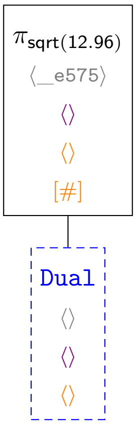
Relational algebra expression for search-based evaluation ('sqrt()' returning float values)

$$r = \pi_{\mathsf{sqrt(12.96)}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation ('sqrt()' returning float values)



Incremental relational algebra tree ('sqrt()' returning float values)



A.32.7 Arithmetic expressions inside aggregation

Query specification (Arithmetic expressions inside aggregation)

```
MATCH (me)-[r1:ATE]->()<-[r2:ATE]-(you)
WHERE me.name = 'Michael'
WITH me, count(DISTINCT r1) AS H1, count(DISTINCT r2) AS H2, you
MATCH (me)-[r1:ATE]->()<-[r2:ATE]-(you)
RETURN me, you, sum((1 - abs(r1.times / H1 - r2.times / H2)) * (r1.times + r2.times) / (H1 + H2)
    ) AS sum
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.32.8 Matching and disregarding output, then matching again

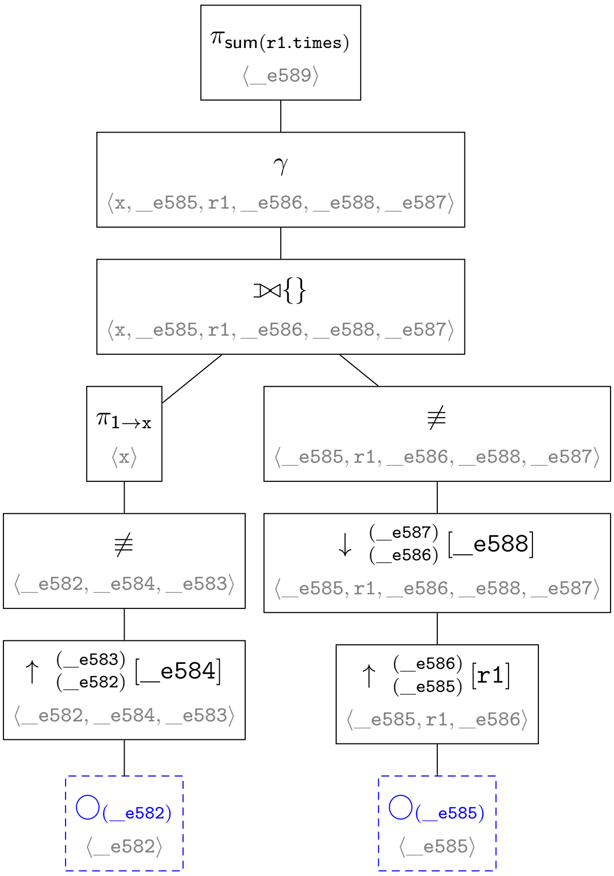
Query specification (Matching and disregarding output, then matching again)

```
MATCH ()-->()
WITH 1 AS x
MATCH ()-[r1]->()<--()
RETURN sum(r1.times)
```

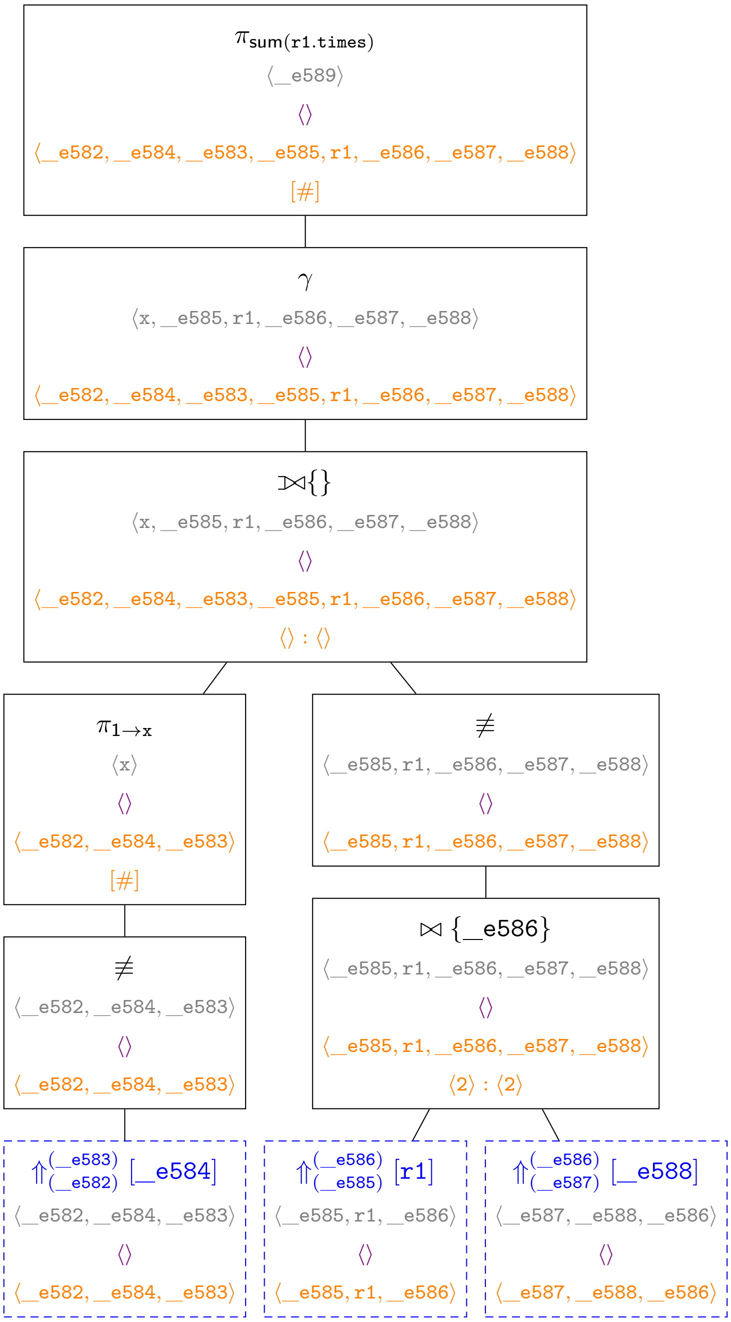
Relational algebra expression for search-based evaluation (Matching and disregarding output, then matching again)

$$r = \pi_{\mathtt{sum(r1.times)}}\Big(\gamma\Big(\pi_{1\rightarrow\mathtt{x}}\Big(\not\equiv\Big(\uparrow^{(\_e583)}_{(\_e582)}[\_e584]\Big(\bigcirc_{(\_e582)}\Big)\Big)\Big) ⟕ \not\equiv\Big(\downarrow^{(\_e587)}_{(\_e586)}[\_e588] \Big(\uparrow^{(\_e586)}_{(\_e585)}[\mathtt{r1}]\Big(\bigcirc_{(\_e585)}\Big)\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Matching and disregarding output, then matching again)


$\pi_{\mathtt{sum(r1.times)}}$
⟨_e589⟩
γ
⟨x, _e585, r1, _e586, _e588, _e587⟩
⟕{}
⟨x, _e585, r1, _e586, _e588, _e587⟩
$\pi_{1\rightarrow\mathtt{x}}$
⟨x⟩
≢
⟨_e585, r1, _e586, _e588, _e587⟩
≢
⟨_e582, _e584, _e583⟩
$\downarrow^{(\_e587)}_{(\_e586)}[\_e588]$
⟨_e585, r1, _e586, _e588, _e587⟩
$\uparrow^{(\_e583)}_{(\_e582)}[\_e584]$
⟨_e582, _e584, _e583⟩
$\uparrow^{(\_e586)}_{(\_e585)}[\mathtt{r1}]$
⟨_e585, r1, _e586⟩
$\bigcirc_{(\_e582)}$
⟨_e582⟩
$\bigcirc_{(\_e585)}$
⟨_e585⟩

Incremental relational algebra tree (Matching and disregarding output, then matching again)


$\pi_{\mathrm{sum(r1.times)}}$
⟨_e589⟩
⟨⟩
⟨_e582, _e584, _e583, _e585, r1, _e586, _e587, _e588⟩
[#]
γ
⟨x, _e585, r1, _e586, _e587, _e588⟩
⟨⟩
⟨_e582, _e584, _e583, _e585, r1, _e586, _e587, _e588⟩
⟖{}
⟨x, _e585, r1, _e586, _e587, _e588⟩
⟨⟩
⟨_e582, _e584, _e583, _e585, r1, _e586, _e587, _e588⟩
⟨⟩ : ⟨⟩
$\pi_{1\to x}$
⟨x⟩
⟨⟩
⟨_e582, _e584, _e583⟩
[#]
≢
⟨_e585, r1, _e586, _e587, _e588⟩
⟨⟩
⟨_e585, r1, _e586, _e587, _e588⟩
⋈ {_e586}
⟨_e585, r1, _e586, _e587, _e588⟩
⟨⟩
⟨_e585, r1, _e586, _e587, _e588⟩
⟨2⟩ : ⟨2⟩
≢
⟨_e582, _e584, _e583⟩
⟨⟩
⟨_e582, _e584, _e583⟩
$\Uparrow^{(\_e583)}_{(\_e582)}$ [_e584]
⟨_e582, _e584, _e583⟩
⟨⟩
⟨_e582, _e584, _e583⟩
$\Uparrow^{(\_e586)}_{(\_e585)}$ [r1]
⟨_e585, r1, _e586⟩
⟨⟩
⟨_e585, r1, _e586⟩
$\Uparrow^{(\_e586)}_{(\_e587)}$ [_e588]
⟨_e587, _e588, _e586⟩
⟨⟩
⟨_e587, _e588, _e586⟩

A.32.9 Returning a list property

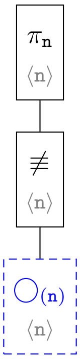
Query specification (Returning a list property)

```
MATCH (n)
RETURN n
```

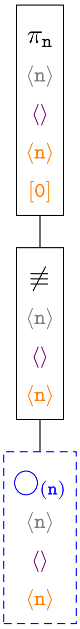
Relational algebra expression for search-based evaluation (Returning a list property)

$$r = \pi_{\mathrm{n}}\Big(\not\equiv \Big(\bigcirc_{(\mathrm{n})} \Big)\Big)$$

Relational algebra tree for search-based evaluation (Returning a list property)



Incremental relational algebra tree (Returning a list property)



A.32.10 Returning a projected map

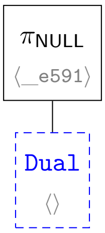
Query specification (Returning a projected map)

```
RETURN {a: 1, b: 'foo'}
```

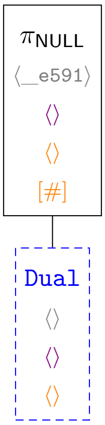
Relational algebra expression for search-based evaluation (Returning a projected map)

$$r = \pi_{\mathsf{NULL}}\Big(\mathtt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Returning a projected map)



Incremental relational algebra tree (Returning a projected map)



A.32.11 Returning an expression

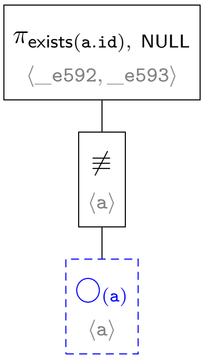
Query specification (Returning an expression)

```
MATCH (a)
RETURN exists(a.id), a IS NOT NULL
```

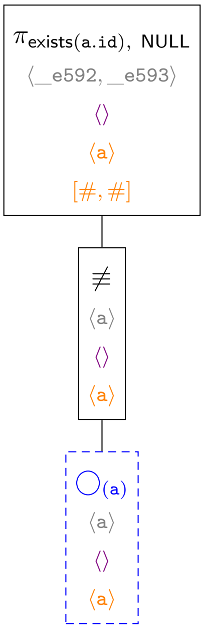
Relational algebra expression for search-based evaluation (Returning an expression)

$$r = \pi_{\text{exists(a.id), NULL}}\Big(\not\equiv\Big(\bigcirc_{\text{(a)}}\Big)\Big)$$

Relational algebra tree for search-based evaluation (Returning an expression)



Incremental relational algebra tree (Returning an expression)


$\pi_{\mathsf{exists(a.id)},\ \mathsf{NULL}}$
⟨_e592, _e593⟩
⟨⟩
⟨a⟩
[#, #]
≢
⟨a⟩
⟨⟩
⟨a⟩
$\bigcirc_{(\mathsf{a})}$
⟨a⟩
⟨⟩
⟨a⟩

A.32.12 Concatenating and returning the size of literal lists

Query specification (Concatenating and returning the size of literal lists)

```
RETURN size([[], []] + [[]]) AS l
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.32.13 Limiting amount of rows when there are fewer left than the LIMIT argument

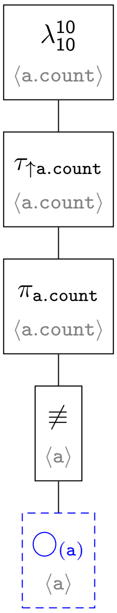
Query specification (Limiting amount of rows when there are fewer left than the LIMIT argument)

```
MATCH (a)
RETURN a.count
ORDER BY a.count
SKIP 10
LIMIT 10
```

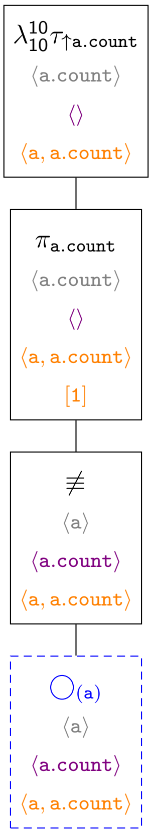
Relational algebra expression for search-based evaluation (Limiting amount of rows when there are fewer left than the LIMIT argument)

$$r = \lambda_{10}^{10}\Big(\tau_{\uparrow \mathtt{a.count}}\Big(\pi_{\mathtt{a.count}}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{a})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Limiting amount of rows when there are fewer left than the LIMIT argument)



Incremental relational algebra tree (Limiting amount of rows when there are fewer left than the LIMIT argument)



A.32.14 'substring()' with default second argument

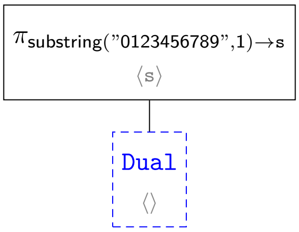
Query specification ('substring()' with default second argument)

```
RETURN substring('0123456789', 1) AS s
```

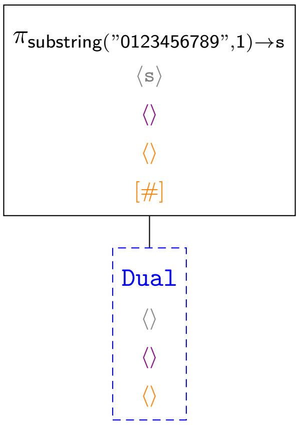
Relational algebra expression for search-based evaluation ('substring()' with default second argument)

$$r = \pi_{\mathsf{substring}(\text{"0123456789"},1) \rightarrow \mathsf{s}}\left(\texttt{Dual}\right)$$

Relational algebra tree for search-based evaluation ('substring()' with default second argument)


$\pi_{\mathsf{substring}(\text{"0123456789"},1) \rightarrow \mathsf{s}}$
⟨s⟩
Dual
⟨⟩

Incremental relational algebra tree ('substring()' with default second argument)


$\pi_{\mathsf{substring}(\text{"0123456789"},1) \rightarrow \mathsf{s}}$
⟨s⟩
⟨⟩
⟨⟩
[#]
Dual
⟨⟩
⟨⟩
⟨⟩

A.32.15 Returning all variables with ordering

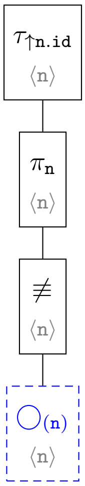
Query specification (Returning all variables with ordering)

```
MATCH (n)
RETURN *
ORDER BY n.id
```

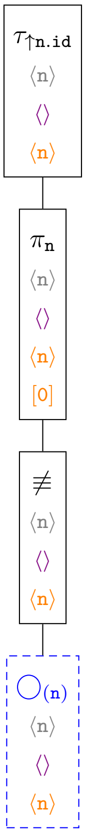
Relational algebra expression for search-based evaluation (Returning all variables with ordering)

$$r = \tau_{\uparrow \texttt{n.id}}\left(\pi_{\texttt{n}}\left(\not\equiv\left(\bigcirc_{(\texttt{n})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Returning all variables with ordering)



Incremental relational algebra tree (Returning all variables with ordering)



A.32.16 Using aliased DISTINCT expression in ORDER BY

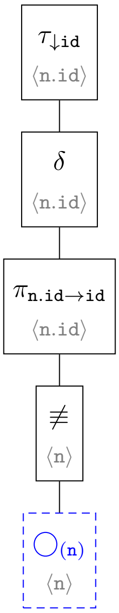
Query specification (Using aliased DISTINCT expression in ORDER BY)

```
MATCH (n)
RETURN DISTINCT n.id AS id
ORDER BY id DESC
```

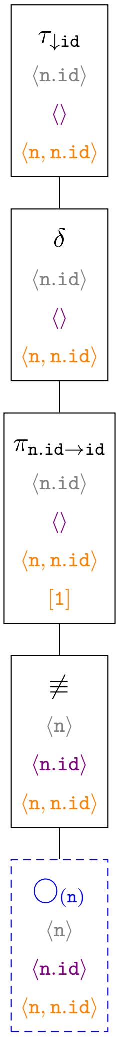
Relational algebra expression for search-based evaluation **(Using aliased DISTINCT expression in ORDER BY)**

$$r = \tau_{\downarrow \texttt{id}}\Big(\delta\Big(\pi_{\texttt{n.id}\rightarrow\texttt{id}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{n})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation **(Using aliased DISTINCT expression in ORDER BY)**


$\tau_{\downarrow \texttt{id}}$
⟨n.id⟩
δ
⟨n.id⟩
$\pi_{\texttt{n.id}\rightarrow\texttt{id}}$
⟨n.id⟩
$\not\equiv$
⟨n⟩
$\bigcirc_{(\texttt{n})}$
⟨n⟩

Incremental relational algebra tree (Using aliased DISTINCT expression in ORDER BY)



A.32.17 Returned columns do not change from using ORDER BY

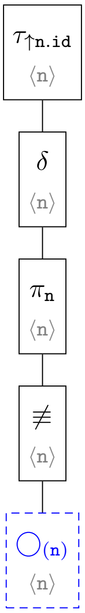
Query specification (Returned columns do not change from using ORDER BY)

```
MATCH (n)
RETURN DISTINCT n
ORDER BY n.id
```

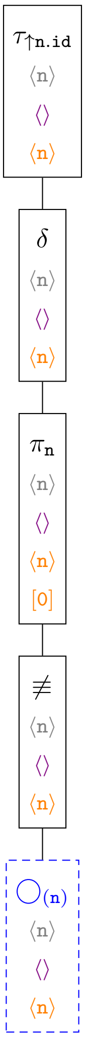
Relational algebra expression for search-based evaluation (Returned columns do not change from using ORDER BY)

$$r = \tau_{\uparrow\texttt{n.id}}\Big(\delta\Big(\pi_{\texttt{n}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{n})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Returned columns do not change from using ORDER BY)



Incremental relational algebra tree (Returned columns do not change from using ORDER BY)



A.32.18 Arithmetic expressions should propagate null values

Query specification (Arithmetic expressions should propagate null values)

```
RETURN 1 + (2 - (3 * (4 / (5 ^ (6 % null))))) AS a
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.32.19 Indexing into nested literal lists

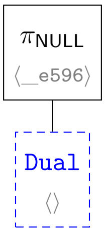
Query specification (Indexing into nested literal lists)

```
RETURN [[1]][0][0]
```

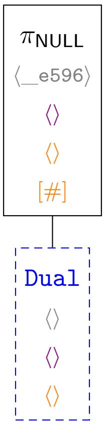
Relational algebra expression for search-based evaluation (Indexing into nested literal lists)

$$r = \pi_{\mathsf{NULL}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Indexing into nested literal lists)



Incremental relational algebra tree (Indexing into nested literal lists)



A.32.20 Aliasing expressions

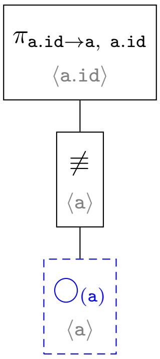
Query specification (Aliasing expressions)

```
MATCH (a)
RETURN a.id AS a, a.id
```

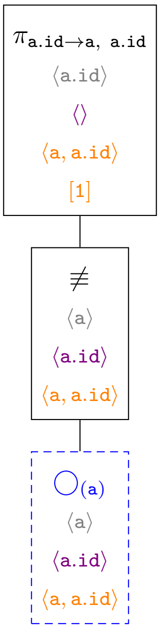
Relational algebra expression for search-based evaluation (Aliasing expressions)

$$r = \pi_{\texttt{a.id}\to\texttt{a},\ \texttt{a.id}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{a})}\Big)\Big)$$

Relational algebra tree for search-based evaluation (Aliasing expressions)


$\pi_{\texttt{a.id}\rightarrow\texttt{a, a.id}}$
⟨a.id⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{\texttt{(a)}}$
⟨a⟩

Incremental relational algebra tree (Aliasing expressions)


$\pi_{\texttt{a.id}\rightarrow\texttt{a, a.id}}$
⟨a.id⟩
⟨⟩
⟨a, a.id⟩
[1]
$\not\equiv$
⟨a⟩
⟨a.id⟩
⟨a, a.id⟩
$\bigcirc_{\texttt{(a)}}$
⟨a⟩
⟨a.id⟩
⟨a, a.id⟩

A.32.21 Projecting an arithmetic expression with aggregation

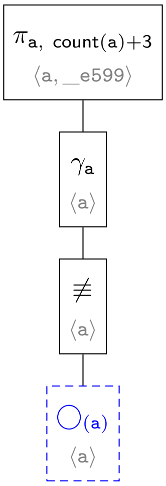
Query specification (Projecting an arithmetic expression with aggregation)

```
MATCH (a)
RETURN a, count(a) + 3
```

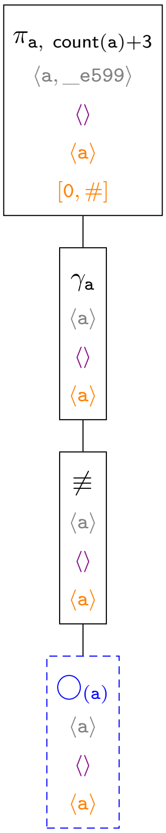
Relational algebra expression for search-based evaluation (Projecting an arithmetic expression with aggregation)

$$r = \pi_{\mathsf{a},\ \mathsf{count(a)+3}}\Big(\gamma_{\mathsf{a}}\Big(\not\equiv\Big(\bigcirc_{\mathsf{(a)}}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Projecting an arithmetic expression with aggregation)



Incremental relational algebra tree (Projecting an arithmetic expression with aggregation)



A.32.22 Aggregating by a list property has a correct definition of equality

Query specification (Aggregating by a list property has a correct definition of equality)

```
MATCH (a)
WITH a.a AS a, count(*) AS count
RETURN count
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.32.23 Reusing variable names

Query specification (Reusing variable names)

```
MATCH (person:Person)<--(message)<-[like]-(:Person)
WITH like.creationDate AS likeTime, person AS person
ORDER BY likeTime, message.id
WITH head(collect({likeTime: likeTime})) AS latestLike, person AS person
RETURN latestLike.likeTime AS likeTime
ORDER BY likeTime
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.32.24 Concatenating lists of same type

Query specification (Concatenating lists of same type)

```
RETURN [1, 10, 100] + [4, 5] AS foo
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.32.25 Appending lists of same type

Query specification (Appending lists of same type)

```
RETURN [false, true] + false AS foo
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.32.26 DISTINCT inside aggregation should work with lists in maps

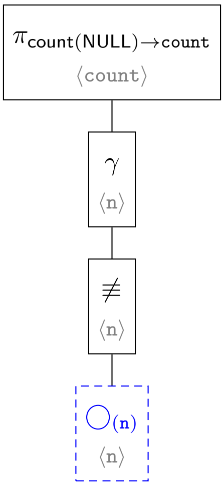
Query specification (DISTINCT inside aggregation should work with lists in maps)

```
MATCH (n)
RETURN count(DISTINCT {foo: n.list}) AS count
```

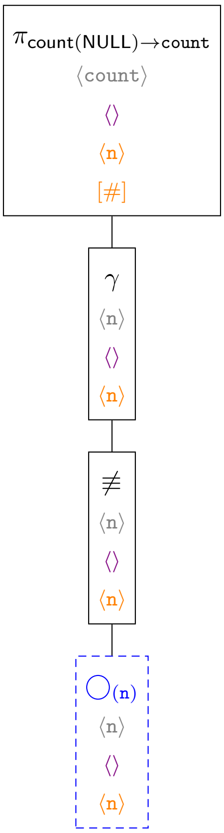
Relational algebra expression for search-based evaluation (DISTINCT inside aggregation should work with lists in maps)

$$r = \pi_{\mathsf{count(NULL)} \rightarrow \mathsf{count}} \Big(\gamma \Big(\not\equiv \Big(\bigcirc_{(\mathtt{n})} \Big) \Big) \Big)$$

Relational algebra tree for search-based evaluation (DISTINCT inside aggregation should work with lists in maps)



Incremental relational algebra tree (DISTINCT inside aggregation should work with lists in maps)



A.32.27 Handling DISTINCT with lists in maps

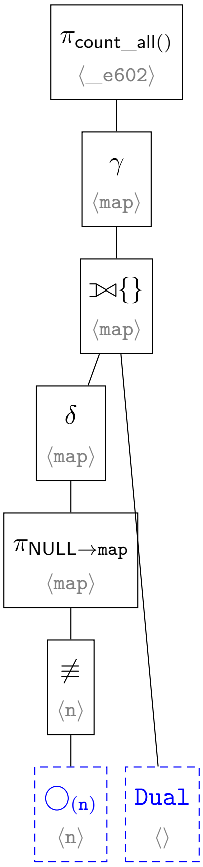
Query specification (Handling DISTINCT with lists in maps)

```
MATCH (n)
WITH DISTINCT {foo: n.list} AS map
RETURN count(*)
```

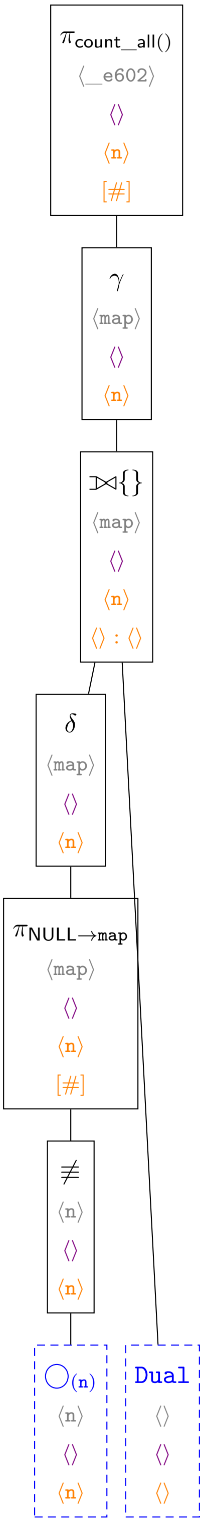
Relational algebra expression for search-based evaluation **(Handling DISTINCT with lists in maps)**

$$r = \pi_{\mathsf{count\_all()}}\left(\gamma\left(\delta\left(\pi_{\mathsf{NULL}\rightarrow\mathtt{map}}\left(\not\equiv\left(\bigcirc_{(\mathtt{n})}\right)\right)\right)\mathbin{⟕}\mathtt{Dual}\right)\right)$$

Relational algebra tree for search-based evaluation **(Handling DISTINCT with lists in maps)**


$\pi_{\mathsf{count\_all()}}$
⟨_e602⟩
γ
⟨map⟩
⟕{}
⟨map⟩
δ
⟨map⟩
$\pi_{\mathsf{NULL}\rightarrow\mathtt{map}}$
⟨map⟩
$\not\equiv$
⟨n⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩
Dual
⟨⟩

Incremental relational algebra tree (Handling DISTINCT with lists in maps)



A.32.28 DISTINCT inside aggregation should work with nested lists in maps

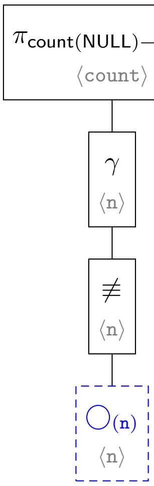
Query specification (DISTINCT inside aggregation should work with nested lists in maps)

```
MATCH (n)
RETURN count(DISTINCT {foo: [[n.list, n.list], [n.list, n.list]]}) AS count
```

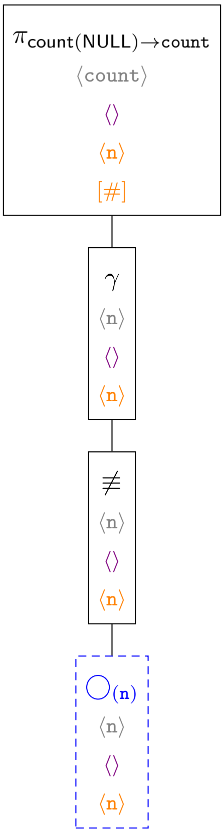
Relational algebra expression for search-based evaluation (DISTINCT inside aggregation should work with nested lists in maps)

$$r = \pi_{\mathsf{count(NULL)} \rightarrow \mathsf{count}} \Big(\gamma \Big(\not\equiv \Big(\bigcirc_{(\mathtt{n})} \Big) \Big) \Big)$$

Relational algebra tree for search-based evaluation (DISTINCT inside aggregation should work with nested lists in maps)


$\pi_{\mathsf{count(NULL)} \rightarrow \mathsf{count}}$
⟨count⟩
γ
⟨n⟩
$\not\equiv$
⟨n⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩

Incremental relational algebra tree (DISTINCT inside aggregation should work with nested lists in maps)


$\pi_{\mathsf{count(NULL)} \rightarrow \mathsf{count}}$
⟨count⟩
⟨⟩
⟨n⟩
[#]
γ
⟨n⟩
⟨⟩
⟨n⟩
$\not\equiv$
⟨n⟩
⟨⟩
⟨n⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩
⟨⟩
⟨n⟩

A.32.29 DISTINCT inside aggregation should work with nested lists of maps in maps

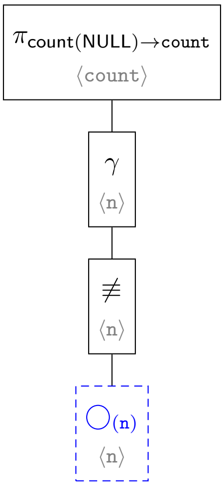
Query specification (DISTINCT inside aggregation should work with nested lists of maps in maps)

```
MATCH (n)
RETURN count(DISTINCT {foo: [{bar: n.list}, {baz: {apa: n.list}}]}) AS count
```

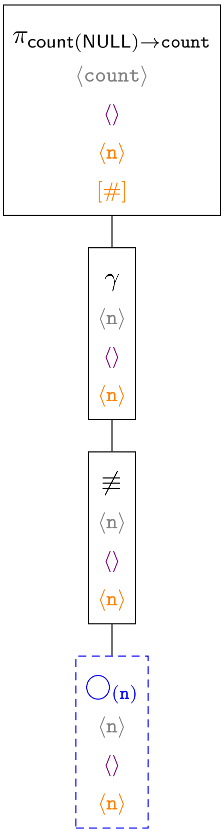
Relational algebra expression for search-based evaluation (DISTINCT inside aggregation should work with nested lists of maps in maps)

$$r = \pi_{\mathsf{count(NULL)} \rightarrow \mathsf{count}}\Big(\gamma\Big(\not\equiv\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (DISTINCT inside aggregation should work with nested lists of maps in maps)


$\pi_{\mathsf{count(NULL)} \rightarrow \mathsf{count}}$
⟨count⟩
γ
⟨n⟩
$\not\equiv$
⟨n⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩

Incremental relational algebra tree (DISTINCT inside aggregation should work with nested lists of maps in maps)



A.33 SemanticErrorAcceptance

Progress:

1 of 2

A.33.1 Handling property access on the Any type

Query specification (Handling property access on the Any type)

```
WITH [{prop: 0}, 1] AS list
RETURN (list[0]).prop
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.33.2 Bad arguments for 'range()'

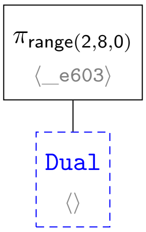
Query specification (Bad arguments for 'range()')

```
RETURN range(2, 8, 0)
```

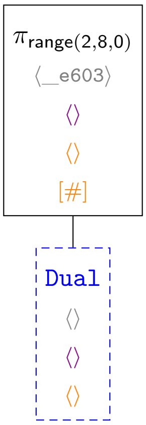
Relational algebra expression for search-based evaluation (Bad arguments for 'range()')

$$r = \pi_{\mathsf{range(2,8,0)}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Bad arguments for 'range()')



Incremental relational algebra tree (Bad arguments for 'range()')



A.34 SetAcceptance

Progress:

0 of 0

A.35 SkipLimitAcceptanceTest

Progress:

0 of 2

A.35.1 SKIP with an expression that does not depend on variables

Query specification (SKIP with an expression that does not depend on variables)

```
MATCH (n)
WITH n SKIP toInteger(rand()*9)
WITH count(*) AS count
RETURN count > 0 AS nonEmpty
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.35.2 LIMIT with an expression that does not depend on variables

Query specification (LIMIT with an expression that does not depend on variables)

```
MATCH (n)
WITH n LIMIT toInteger(ceil(1.7))
RETURN count(*) AS count
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.36 StartingPointAcceptance

Progress:

3 of 3

A.36.1 Find all nodes

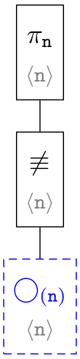
Query specification **(Find all nodes)**

```
MATCH (n)
RETURN n
```

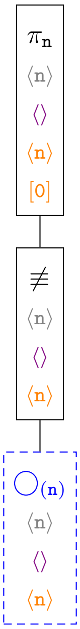
Relational algebra expression for search-based evaluation **(Find all nodes)**

$$r = \pi_{\mathtt{n}}\Big(\not\equiv \Big(\bigcirc_{(\mathtt{n})} \Big)\Big)$$

Relational algebra tree for search-based evaluation **(Find all nodes)**



Incremental relational algebra tree **(Find all nodes)**



A.36.2 Find labelled nodes

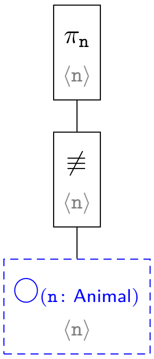
Query specification (Find labelled nodes)

```
MATCH (n:Animal)
RETURN n
```

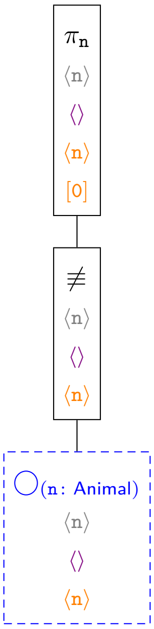
Relational algebra expression for search-based evaluation (Find labelled nodes)

$$r = \pi_{\mathtt{n}}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{n}:\ \mathsf{Animal})}\Big)\Big)$$

Relational algebra tree for search-based evaluation (Find labelled nodes)



Incremental relational algebra tree (Find labelled nodes)



A.36.3 Find nodes by property

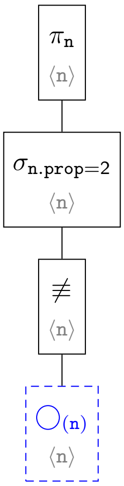
Query specification (Find nodes by property)

```
MATCH (n)
WHERE n.prop = 2
RETURN n
```

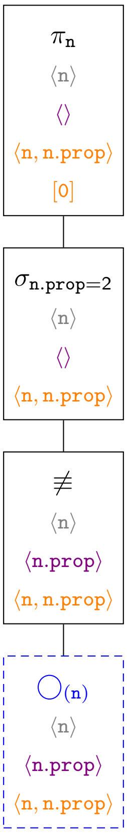
Relational algebra expression for search-based evaluation (Find nodes by property)

$$r = \pi_{\mathtt{n}}\Big(\sigma_{\mathtt{n.prop}=2}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{n})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Find nodes by property)



Incremental relational algebra tree (Find nodes by property)



A.37 StartsWithAcceptance

Progress:

21 of 24

A.37.1 Finding exact matches

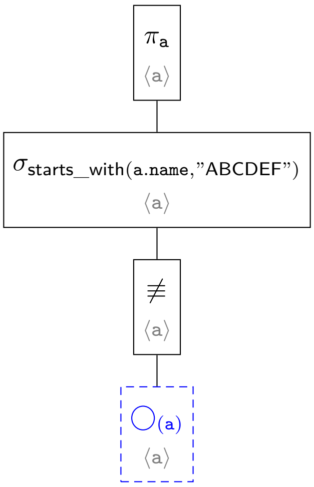
Query specification (Finding exact matches)

```
MATCH (a)
WHERE a.name STARTS WITH 'ABCDEF'
RETURN a
```

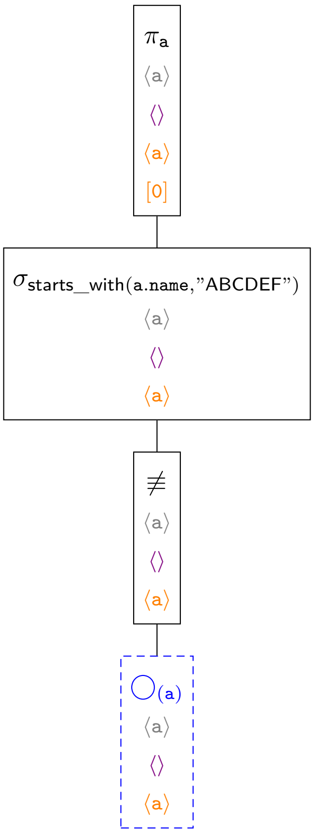
Relational algebra expression for search-based evaluation (Finding exact matches)

$$r = \pi_{\mathsf{a}}\left(\sigma_{\mathsf{starts\_with(a.name,"ABCDEF")}}\left(\not\equiv\left(\bigcirc_{\mathsf{(a)}}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Finding exact matches)


π_{a}
⟨a⟩
$\sigma_{\mathsf{starts\_with(a.name,"ABCDEF")}}$
⟨a⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{\mathsf{(a)}}$
⟨a⟩

Incremental relational algebra tree (Finding exact matches)


π_{a}
⟨a⟩
⟨⟩
⟨a⟩
[0]
$\sigma_{\mathsf{starts\_with(a.name,"ABCDEF")}}$
⟨a⟩
⟨⟩
⟨a⟩
$\not\equiv$
⟨a⟩
⟨⟩
⟨a⟩
$\bigcirc_{\mathsf{(a)}}$
⟨a⟩
⟨⟩
⟨a⟩

A.37.2 Finding beginning of string

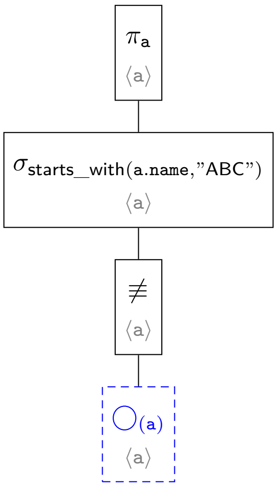
Query specification (Finding beginning of string)

```
MATCH (a)
WHERE a.name STARTS WITH 'ABC'
RETURN a
```

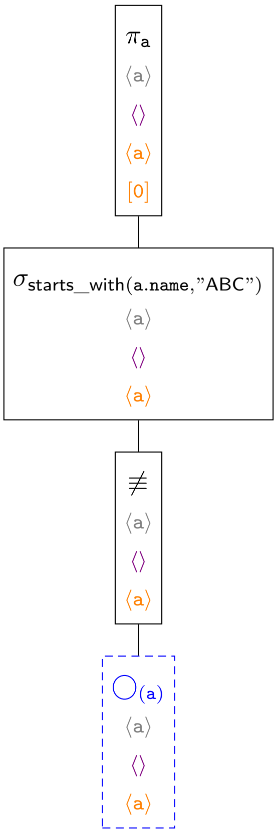
Relational algebra expression for search-based evaluation (Finding beginning of string)

$$r = \pi_{\mathsf{a}}\left(\sigma_{\mathsf{starts\_with(a.name,"ABC")}}\left(\not\equiv\left(\bigcirc_{\mathsf{(a)}}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Finding beginning of string)


π_{a}
⟨a⟩
$\sigma_{\mathsf{starts\_with(a.name,"ABC")}}$
⟨a⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{\mathsf{(a)}}$
⟨a⟩

Incremental relational algebra tree (Finding beginning of string)


π_{a}
⟨a⟩
⟨⟩
⟨a⟩
[0]
$\sigma_{\mathsf{starts\_with(a.name,"ABC")}}$
⟨a⟩
⟨⟩
⟨a⟩
$\not\equiv$
⟨a⟩
⟨⟩
⟨a⟩
$\bigcirc_{\mathsf{(a)}}$
⟨a⟩
⟨⟩
⟨a⟩

A.37.3 Finding end of string 1

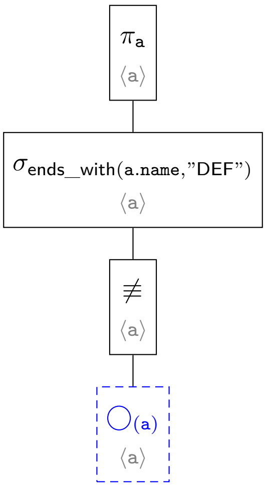
Query specification (Finding end of string 1)

```
MATCH (a)
WHERE a.name ENDS WITH 'DEF'
RETURN a
```

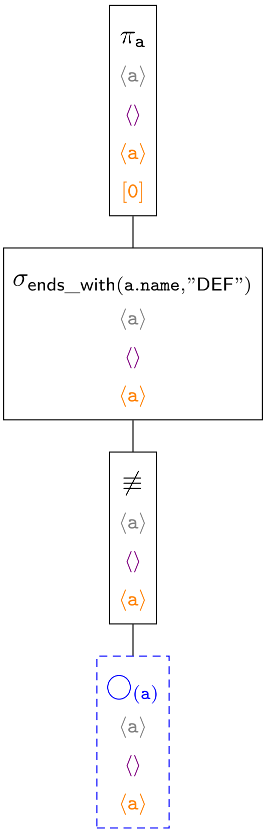
Relational algebra expression for search-based evaluation (Finding end of string 1)

$$r = \pi_{\mathsf{a}}\left(\sigma_{\mathsf{ends\_with(a.name,"DEF")}}\left(\not\equiv\left(\bigcirc_{\mathsf{(a)}}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Finding end of string 1)


π_{a}
⟨a⟩
$\sigma_{\mathsf{ends\_with(a.name,"DEF")}}$
⟨a⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{\mathsf{(a)}}$
⟨a⟩

Incremental relational algebra tree (Finding end of string 1)


π_{a}
⟨a⟩
⟨⟩
⟨a⟩
[0]
$\sigma_{\mathsf{ends\_with(a.name,"DEF")}}$
⟨a⟩
⟨⟩
⟨a⟩
$\not\equiv$
⟨a⟩
⟨⟩
⟨a⟩
$\bigcirc_{\mathsf{(a)}}$
⟨a⟩
⟨⟩
⟨a⟩

A.37.4 Finding end of string 2

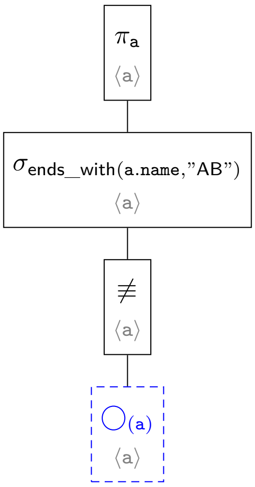
Query specification (Finding end of string 2)

```
MATCH (a)
WHERE a.name ENDS WITH 'AB'
RETURN a
```

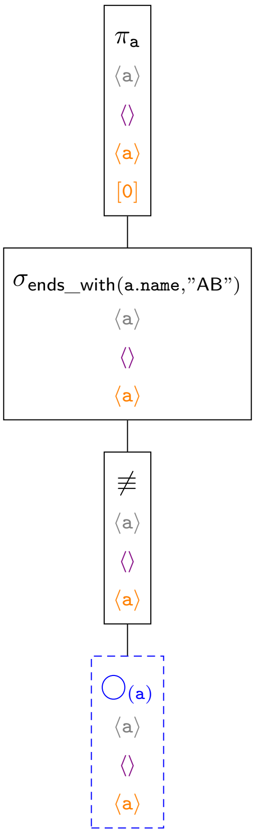
Relational algebra expression for search-based evaluation (Finding end of string 2)

$$r = \pi_{\mathsf{a}}\Big(\sigma_{\mathsf{ends\_with(a.name,"AB")}}\Big(\not\equiv\Big(\bigcirc_{\mathsf{(a)}}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Finding end of string 2)


π_{a}
⟨a⟩
$\sigma_{\mathsf{ends\_with(a.name,"AB")}}$
⟨a⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{\mathsf{(a)}}$
⟨a⟩

Incremental relational algebra tree (Finding end of string 2)


π_{a}
⟨a⟩
⟨⟩
⟨a⟩
[0]
$\sigma_{\mathsf{ends\_with(a.name,"AB")}}$
⟨a⟩
⟨⟩
⟨a⟩
$\not\equiv$
⟨a⟩
⟨⟩
⟨a⟩
$\bigcirc_{\mathsf{(a)}}$
⟨a⟩
⟨⟩
⟨a⟩

A.37.5 Finding middle of string

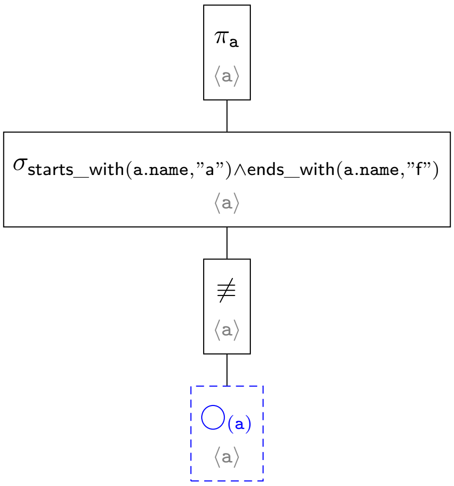
Query specification (Finding middle of string)

```
MATCH (a)
WHERE a.name STARTS WITH 'a'
AND a.name ENDS WITH 'f'
RETURN a
```

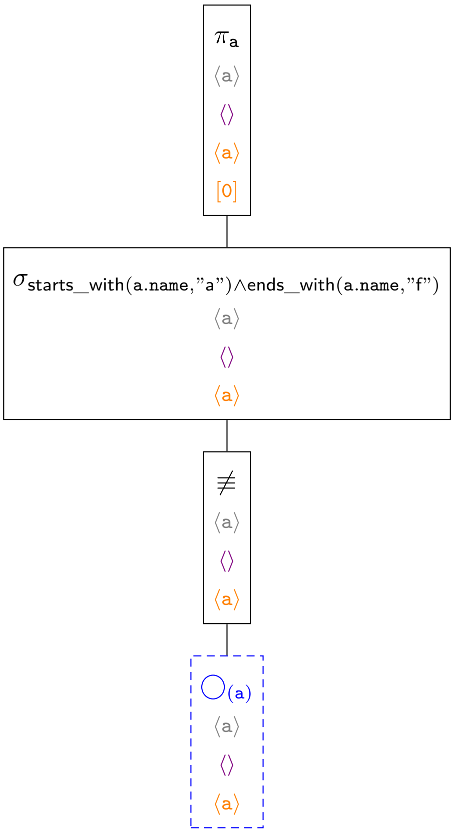
Relational algebra expression for search-based evaluation (Finding middle of string)

$$r = \pi_{\mathsf{a}}\left(\sigma_{\mathsf{starts\_with(a.name,"a")} \land \mathsf{ends\_with(a.name,"f")}}\left(\not\equiv\left(\bigcirc_{(\mathsf{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Finding middle of string)


π_{a}
⟨a⟩
$\sigma_{\mathsf{starts\_with(a.name,"a")} \land \mathsf{ends\_with(a.name,"f")}}$
⟨a⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{(\mathsf{a})}$
⟨a⟩

Incremental relational algebra tree (Finding middle of string)


π_{a}
⟨a⟩
⟨⟩
⟨a⟩
[0]
$\sigma_{\mathsf{starts\_with}(\mathtt{a.name},\text{"a"})\wedge\mathsf{ends\_with}(\mathtt{a.name},\text{"f"})}$
⟨a⟩
⟨⟩
⟨a⟩
$\not\equiv$
⟨a⟩
⟨⟩
⟨a⟩
$\bigcirc_{(\mathsf{a})}$
⟨a⟩
⟨⟩
⟨a⟩

A.37.6 Finding the empty string

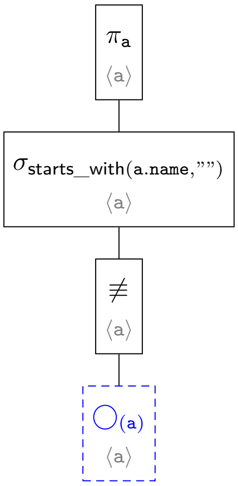
Query specification (Finding the empty string)

```
MATCH (a)
WHERE a.name STARTS WITH ''
RETURN a
```

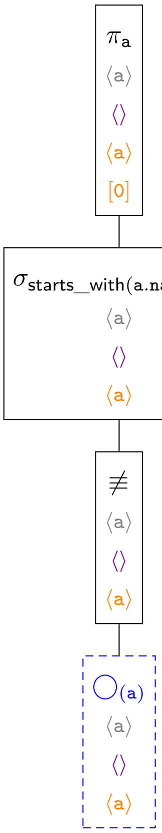
Relational algebra expression for search-based evaluation (Finding the empty string)

$$r = \pi_{\mathsf{a}}\Big(\sigma_{\mathsf{starts\_with}(\mathtt{a.name},\text{""})}\Big(\not\equiv\Big(\bigcirc_{(\mathsf{a})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Finding the empty string)


π_{a}
⟨a⟩
$\sigma_{\mathsf{starts\_with}(\mathtt{a.name},\text{""})}$
⟨a⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{(\mathsf{a})}$
⟨a⟩

Incremental relational algebra tree (Finding the empty string)



A.37.7 Finding when the middle is known

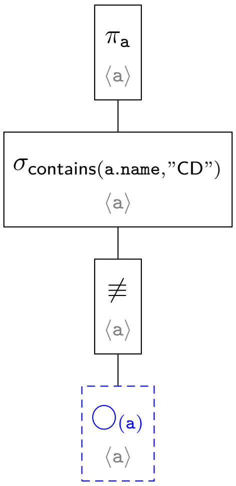
Query specification (Finding when the middle is known)

```
MATCH (a)
WHERE a.name CONTAINS 'CD'
RETURN a
```

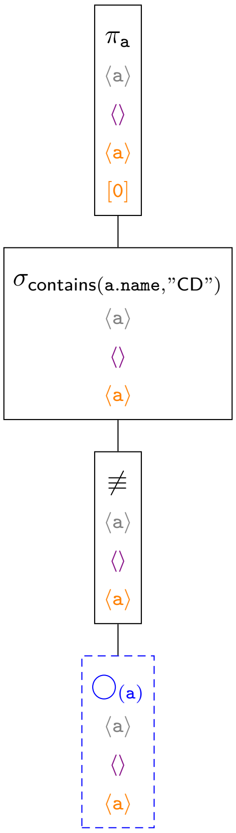
Relational algebra expression for search-based evaluation (Finding when the middle is known)

$$r = \pi_{a}\left(\sigma_{contains(a.name,"CD")}\left(\not\equiv\left(\bigcirc_{(a)}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Finding when the middle is known)



Incremental relational algebra tree (Finding when the middle is known)



A.37.8 Finding strings starting with whitespace

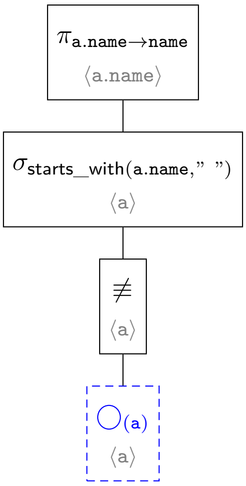
Query specification (Finding strings starting with whitespace)

```
MATCH (a)
WHERE a.name STARTS WITH ' '
RETURN a.name AS name
```

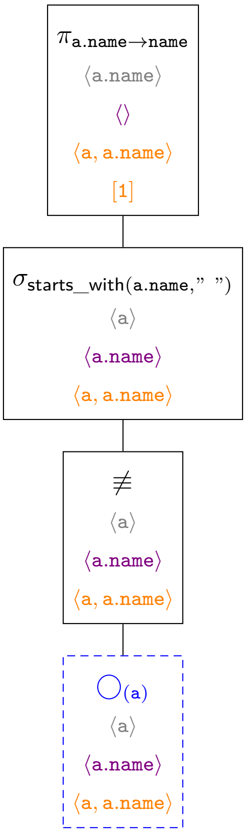
Relational algebra expression for search-based evaluation (Finding strings starting with whitespace)

$$r = \pi_{\texttt{a.name}\rightarrow\texttt{name}}\left(\sigma_{\mathsf{starts\_with}(\texttt{a.name},\text{" "})}\left(\not\equiv\left(\bigcirc_{(\texttt{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Finding strings starting with whitespace)



Incremental relational algebra tree (Finding strings starting with whitespace)



A.37.9 Finding strings starting with newline

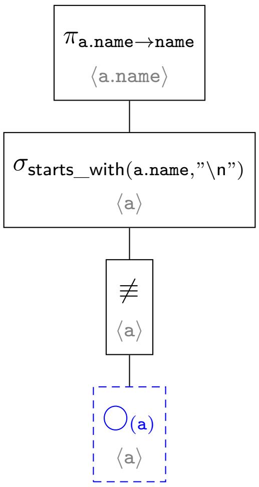
Query specification (Finding strings starting with newline)

```
MATCH (a)
WHERE a.name STARTS WITH '\n'
RETURN a.name AS name
```

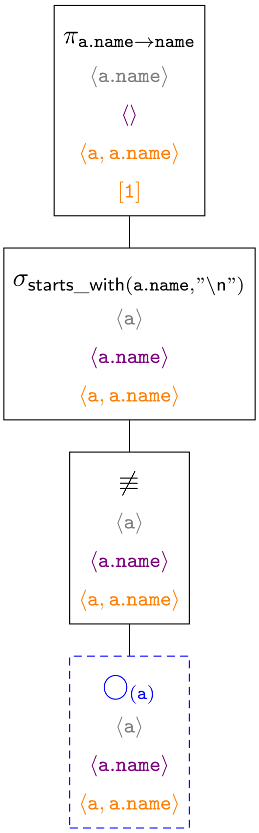
Relational algebra expression for search-based evaluation (Finding strings starting with newline)

$$r = \pi_{\texttt{a.name} \rightarrow \texttt{name}} \left(\sigma_{\mathsf{starts\_with}(\texttt{a.name}, \text{"}\backslash\texttt{n}\text{"})} \left(\not\equiv \left(\bigcirc_{(\texttt{a})} \right) \right) \right)$$

Relational algebra tree for search-based evaluation (Finding strings starting with newline)



Incremental relational algebra tree (Finding strings starting with newline)



A.37.10 Finding strings ending with newline

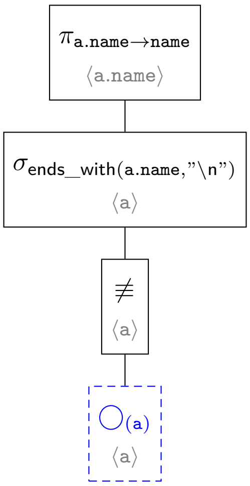
Query specification (Finding strings ending with newline)

```
MATCH (a)
WHERE a.name ENDS WITH '\n'
RETURN a.name AS name
```

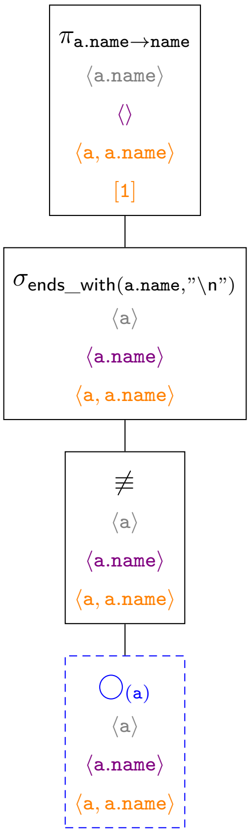
Relational algebra expression for search-based evaluation (Finding strings ending with newline)

$$r = \pi_{\texttt{a.name}\rightarrow\texttt{name}}\left(\sigma_{\mathsf{ends\_with}(\texttt{a.name},"\backslash\texttt{n}")}\left(\not\equiv\left(\bigcirc_{(\texttt{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Finding strings ending with newline)


$\pi_{\texttt{a.name}\rightarrow\texttt{name}}$
⟨a.name⟩
$\sigma_{\mathsf{ends\_with}(\texttt{a.name},"\backslash\texttt{n}")}$
⟨a⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩

Incremental relational algebra tree (Finding strings ending with newline)


$\pi_{\texttt{a.name}\rightarrow\texttt{name}}$
⟨a.name⟩
⟨⟩
⟨a, a.name⟩
[1]
$\sigma_{\text{ends\_with}(\texttt{a.name},\text{"\textbackslash n"})}$
⟨a⟩
⟨a.name⟩
⟨a, a.name⟩
$\not\equiv$
⟨a⟩
⟨a.name⟩
⟨a, a.name⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩
⟨a.name⟩
⟨a, a.name⟩

A.37.11 Finding strings ending with whitespace

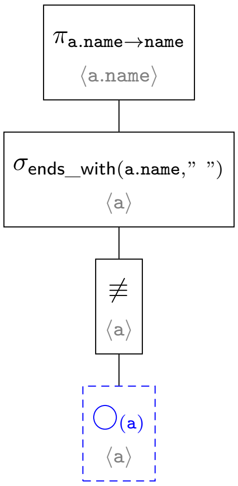
Query specification (Finding strings ending with whitespace)

```
MATCH (a)
WHERE a.name ENDS WITH ' '
RETURN a.name AS name
```

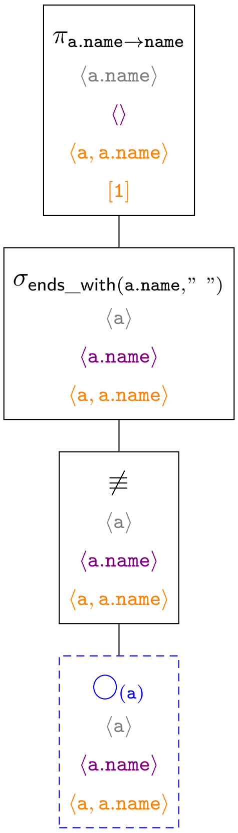
Relational algebra expression for search-based evaluation (Finding strings ending with whitespace)

$$r = \pi_{\texttt{a.name}\rightarrow\texttt{name}}\left(\sigma_{\text{ends\_with}(\texttt{a.name},\text{" "})}\left(\not\equiv\left(\bigcirc_{(\texttt{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Finding strings ending with whitespace)


$\pi_{\texttt{a.name}\rightarrow\texttt{name}}$
⟨a.name⟩
$\sigma_{\text{ends\_with}(\texttt{a.name}, " ")}$
⟨a⟩
≢
⟨a⟩
○(a)
⟨a⟩

Incremental relational algebra tree (Finding strings ending with whitespace)


$\pi_{\texttt{a.name}\rightarrow\texttt{name}}$
⟨a.name⟩
⟨⟩
⟨a, a.name⟩
[1]
$\sigma_{\text{ends\_with}(\texttt{a.name}, " ")}$
⟨a⟩
⟨a.name⟩
⟨a, a.name⟩
≢
⟨a⟩
⟨a.name⟩
⟨a, a.name⟩
○(a)
⟨a⟩
⟨a.name⟩
⟨a, a.name⟩

A.37.12 Finding strings containing whitespace

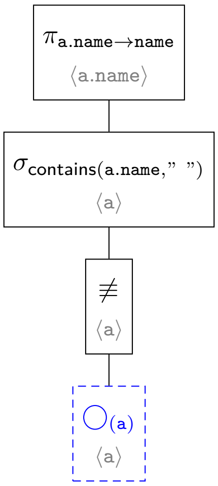
Query specification (Finding strings containing whitespace)

```
MATCH (a)
WHERE a.name CONTAINS ' '
RETURN a.name AS name
```

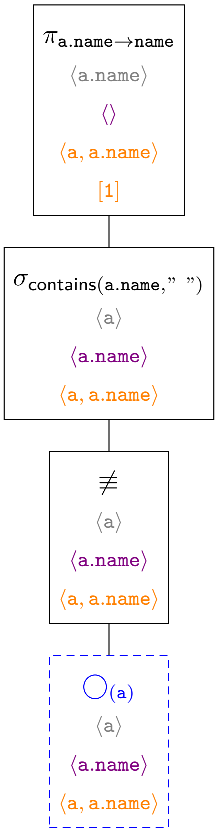
Relational algebra expression for search-based evaluation (Finding strings containing whitespace)

$$r = \pi_{\texttt{a.name} \rightarrow \texttt{name}} \left(\sigma_{\mathsf{contains}(\texttt{a.name}, \text{" "})} \left(\not\equiv \left(\bigcirc_{(\texttt{a})} \right) \right) \right)$$

Relational algebra tree for search-based evaluation (Finding strings containing whitespace)


$\pi_{\texttt{a.name} \rightarrow \texttt{name}}$
⟨a.name⟩
$\sigma_{\mathsf{contains}(\texttt{a.name}, \text{" "})}$
⟨a⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩

Incremental relational algebra tree (Finding strings containing whitespace)


$\pi_{\texttt{a.name} \rightarrow \texttt{name}}$
⟨a.name⟩
⟨⟩
⟨a, a.name⟩
[1]
$\sigma_{\mathsf{contains}(\texttt{a.name}, \text{" "})}$
⟨a⟩
⟨a.name⟩
⟨a, a.name⟩
$\not\equiv$
⟨a⟩
⟨a.name⟩
⟨a, a.name⟩
$\bigcirc_{(\texttt{a})}$
⟨a⟩
⟨a.name⟩
⟨a, a.name⟩

A.37.13 Finding strings containing newline

Query specification (Finding strings containing newline)

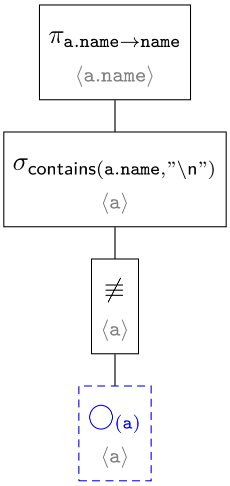
```
MATCH (a)
WHERE a.name CONTAINS '\n'
```

```
RETURN a.name AS name
```

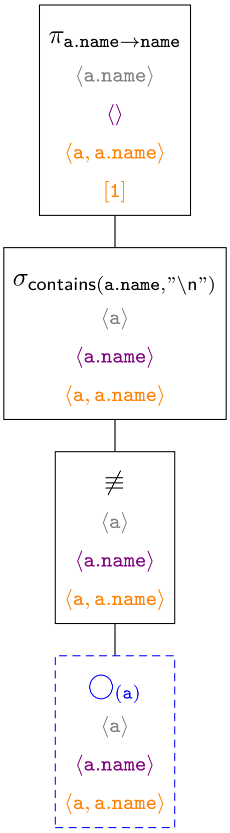
Relational algebra expression for search-based evaluation (Finding strings containing newline)

$$r = \pi_{\texttt{a.name}\rightarrow\texttt{name}}\left(\sigma_{\mathsf{contains}(\texttt{a.name},"\backslash\texttt{n}")}\left(\not\equiv\left(\bigcirc_{(\texttt{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Finding strings containing newline)



Incremental relational algebra tree (Finding strings containing newline)



A.37.14 No string starts with null

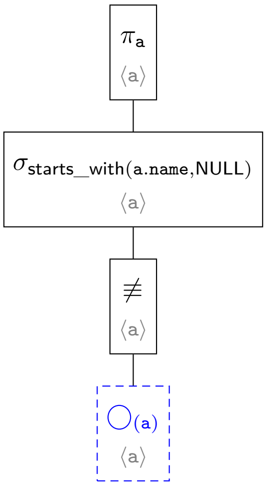
Query specification (No string starts with null)

```
MATCH (a)
WHERE a.name STARTS WITH null
RETURN a
```

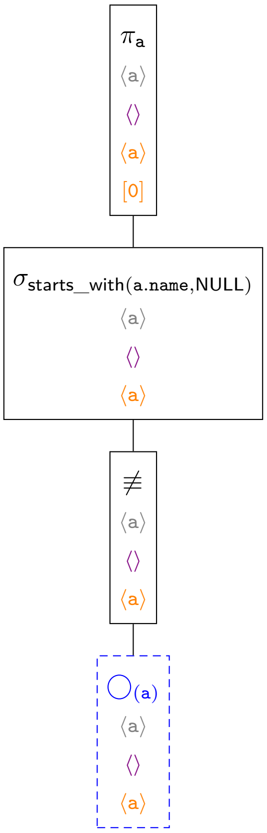
Relational algebra expression for search-based evaluation (No string starts with null)

$$r = \pi_{\mathsf{a}}\left(\sigma_{\mathsf{starts\_with(a.name,NULL)}}\left(\not\equiv\left(\bigcirc_{\mathsf{(a)}}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (No string starts with null)


π_{a}
⟨a⟩
$\sigma_{\mathsf{starts\_with(a.name,NULL)}}$
⟨a⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{\mathsf{(a)}}$
⟨a⟩

Incremental relational algebra tree (No string starts with null)


π_{a}
⟨a⟩
⟨⟩
⟨a⟩
[0]
$\sigma_{\mathsf{starts\_with(a.name,NULL)}}$
⟨a⟩
⟨⟩
⟨a⟩
$\not\equiv$
⟨a⟩
⟨⟩
⟨a⟩
$\bigcirc_{\mathsf{(a)}}$
⟨a⟩
⟨⟩
⟨a⟩

A.37.15 No string does not start with null

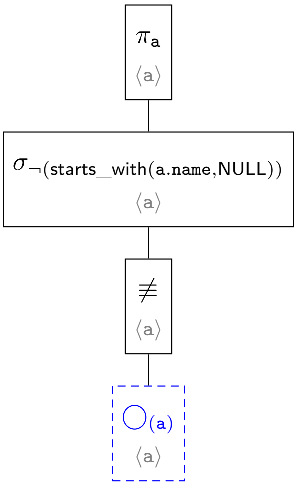
Query specification (No string does not start with null)

```
MATCH (a)
WHERE NOT a.name STARTS WITH null
RETURN a
```

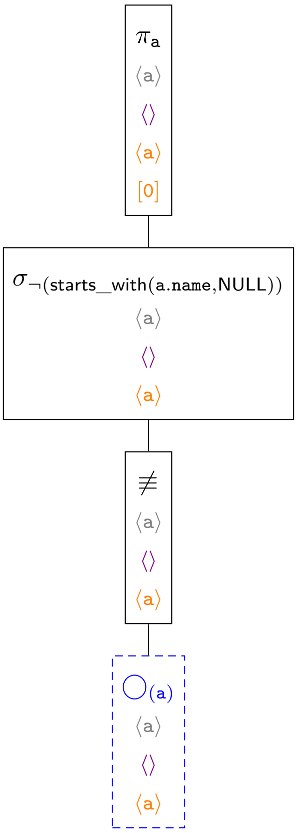
Relational algebra expression for search-based evaluation (No string does not start with null)

$$r = \pi_{\mathsf{a}}\Big(\sigma_{\neg(\mathsf{starts\_with}(\mathsf{a}.\mathtt{name},\mathsf{NULL}))}\Big(\not\equiv\Big(\bigcirc_{(\mathsf{a})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (No string does not start with null)


π_{a}
⟨a⟩
$\sigma_{\neg(\mathsf{starts\_with}(\mathsf{a}.\mathtt{name},\mathsf{NULL}))}$
⟨a⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{(\mathsf{a})}$
⟨a⟩

Incremental relational algebra tree (No string does not start with null)



A.37.16 No string ends with null

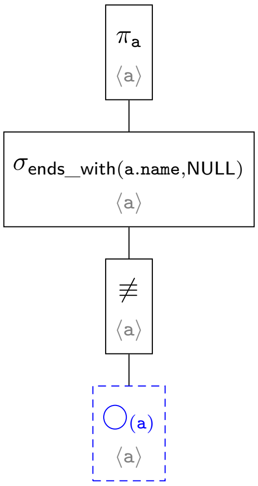
Query specification (No string ends with null)

```
MATCH (a)
WHERE a.name ENDS WITH null
RETURN a
```

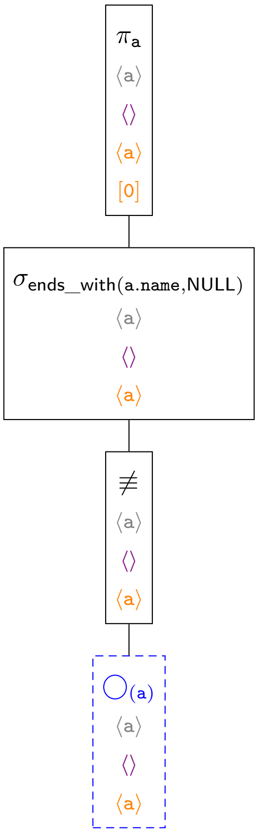
Relational algebra expression for search-based evaluation (No string ends with null)

$$r = \pi_{a}\Big(\sigma_{\text{ends\_with}(a.\text{name},\text{NULL})}\Big(\not\equiv\Big(\bigcirc_{(a)}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (No string ends with null)



Incremental relational algebra tree (No string ends with null)



π_{a}
⟨a⟩
⟨⟩
⟨a⟩
[0]
$\sigma_{\mathsf{ends\_with}(\mathtt{a.name},\mathsf{NULL})}$
⟨a⟩
⟨⟩
⟨a⟩
$\not\equiv$
⟨a⟩
⟨⟩
⟨a⟩
$\bigcirc_{(\mathsf{a})}$
⟨a⟩
⟨⟩
⟨a⟩

A.37.17 No string does not end with null

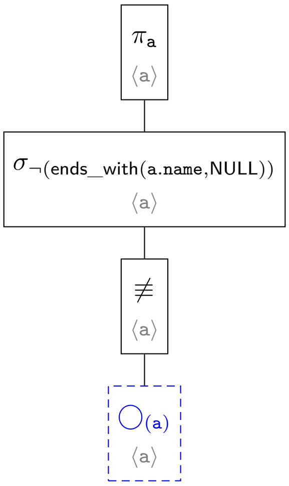
Query specification (No string does not end with null)

```
MATCH (a)
WHERE NOT a.name ENDS WITH null
RETURN a
```

Relational algebra expression for search-based evaluation (No string does not end with null)

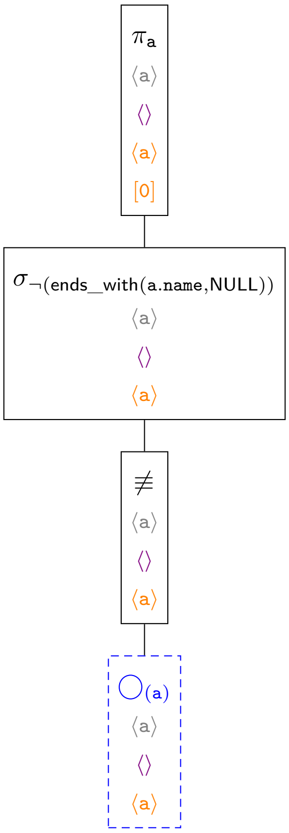
$$r = \pi_{\mathsf{a}}\left(\sigma_{\neg(\mathsf{ends\_with}(\mathtt{a.name},\mathsf{NULL}))}\left(\not\equiv\left(\bigcirc_{(\mathsf{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (No string does not end with null)



π_{a}
⟨a⟩
$\sigma_{\neg(\mathsf{ends\_with}(\mathtt{a.name},\mathsf{NULL}))}$
⟨a⟩
$\not\equiv$
⟨a⟩
$\bigcirc_{(\mathsf{a})}$
⟨a⟩

Incremental relational algebra tree (No string does not end with null)



A.37.18 No string contains null

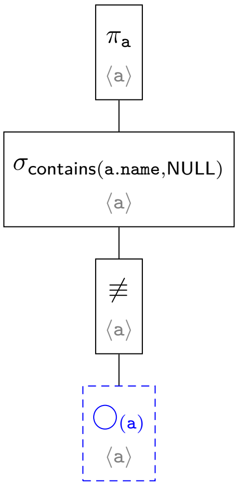
Query specification (No string contains null)

```
MATCH (a)
WHERE a.name CONTAINS null
RETURN a
```

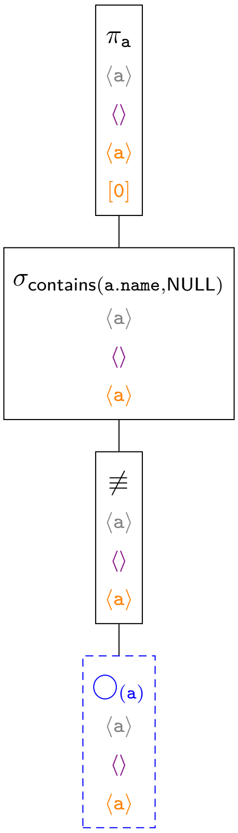
Relational algebra expression for search-based evaluation (No string contains null)

$$r = \pi_{\mathsf{a}}\Big(\sigma_{\mathsf{contains(a.name,NULL)}}\Big(\not\equiv\Big(\bigcirc_{(\mathsf{a})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (No string contains null)



Incremental relational algebra tree (No string contains null)



A.37.19 No string does not contain null

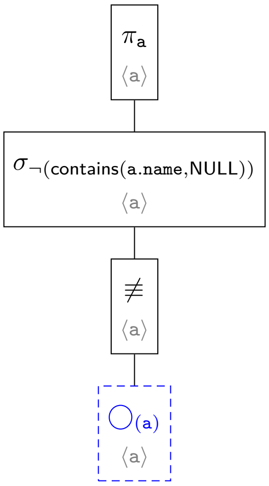
Query specification (No string does not contain null)

```
MATCH (a)
WHERE NOT a.name CONTAINS null
RETURN a
```

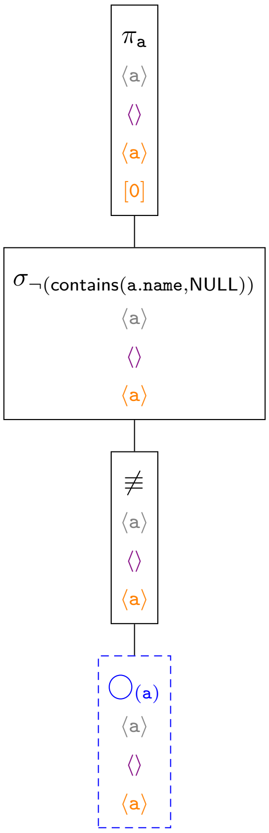
Relational algebra expression for search-based evaluation (No string does not contain null)

$$r = \pi_{a}\left(\sigma_{\neg(\text{contains(a.name,NULL)})}\left(\not\equiv\left(\bigcirc_{(a)}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (No string does not contain null)



Incremental relational algebra tree (No string does not contain null)


π_{a}
⟨a⟩
⟨⟩
⟨a⟩
[0]
$\sigma_{\neg(\mathsf{contains}(\mathsf{a.name},\mathsf{NULL}))}$
⟨a⟩
⟨⟩
⟨a⟩
$\not\equiv$
⟨a⟩
⟨⟩
⟨a⟩
$\bigcirc_{(\mathsf{a})}$
⟨a⟩
⟨⟩
⟨a⟩

A.37.20 Combining string operators

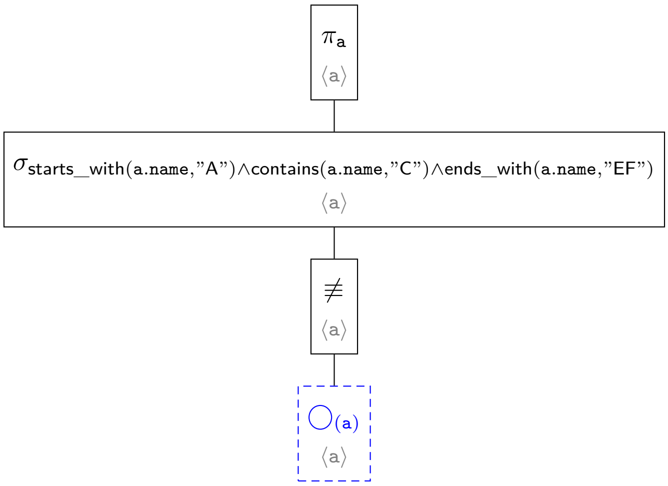
Query specification (Combining string operators)

```
MATCH (a)
WHERE a.name STARTS WITH 'A'
AND a.name CONTAINS 'C'
AND a.name ENDS WITH 'EF'
RETURN a
```

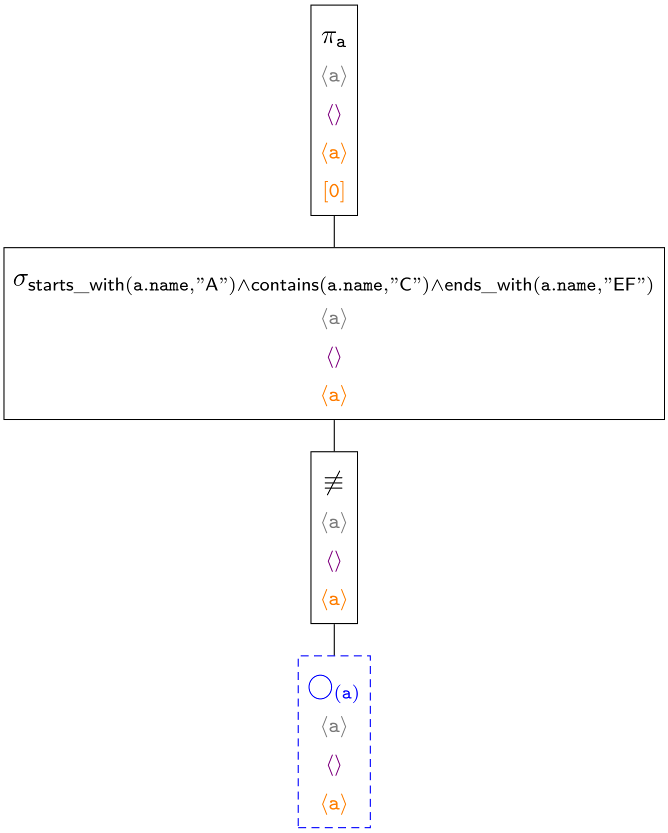
Relational algebra expression for search-based evaluation (Combining string operators)

$$r = \pi_{\mathsf{a}}\left(\sigma_{\mathsf{starts\_with(a.name,"A")} \wedge \mathsf{contains(a.name,"C")} \wedge \mathsf{ends\_with(a.name,"EF")}}\left(\not\equiv\left(\bigcirc_{(\mathsf{a})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Combining string operators)


π_a
⟨a⟩
σ_starts_with(a.name,"A")∧contains(a.name,"C")∧ends_with(a.name,"EF")
⟨a⟩
≢
⟨a⟩
○(a)
⟨a⟩

Incremental relational algebra tree (Combining string operators)


π_a
⟨a⟩
⟨⟩
⟨a⟩
[0]
σ_starts_with(a.name,"A")∧contains(a.name,"C")∧ends_with(a.name,"EF")
⟨a⟩
⟨⟩
⟨a⟩
≢
⟨a⟩
⟨⟩
⟨a⟩
○(a)
⟨a⟩
⟨⟩
⟨a⟩

A.37.21 NOT with CONTAINS

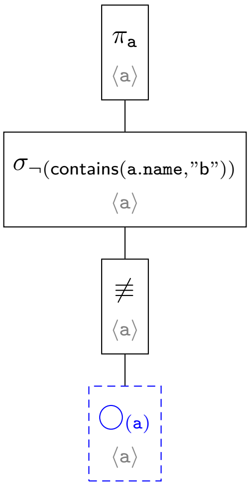
Query specification (NOT with CONTAINS)

```
MATCH (a)
WHERE NOT a.name CONTAINS 'b'
RETURN a
```

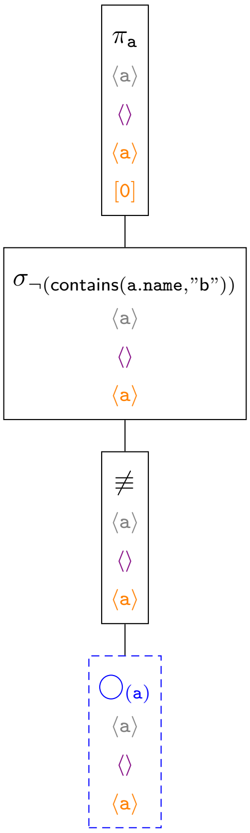
Relational algebra expression for search-based evaluation (NOT with CONTAINS)

$$r = \pi_{\mathsf{a}} \left(\sigma_{\neg(\mathsf{contains}(\mathtt{a.name}, \text{"b"}))} \left(\not\equiv \left(\bigcirc_{(\mathsf{a})} \right) \right) \right)$$

Relational algebra tree for search-based evaluation (NOT with CONTAINS)



Incremental relational algebra tree (NOT with CONTAINS)



A.37.22 Handling non-string operands for STARTS WITH

Query specification (Handling non-string operands for STARTS WITH)

```
WITH [1, 3.14, true, [], {}, null] AS operands
UNWIND operands AS op1
UNWIND operands AS op2
WITH op1 STARTS WITH op2 AS v
RETURN v, count(*)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.37.23 Handling non-string operands for CONTAINS

Query specification (Handling non-string operands for CONTAINS)

```
WITH [1, 3.14, true, [], {}, null] AS operands
UNWIND operands AS op1
UNWIND operands AS op2
WITH op1 STARTS WITH op2 AS v
RETURN v, count(*)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.37.24 Handling non-string operands for ENDS WITH

Query specification (Handling non-string operands for ENDS WITH)

```
WITH [1, 3.14, true, [], {}, null] AS operands
UNWIND operands AS op1
UNWIND operands AS op2
WITH op1 STARTS WITH op2 AS v
RETURN v, count(*)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.38 SyntaxErrorAcceptance

Progress:

0 of 0

A.39 TernaryLogicAcceptanceTest

Progress:

3 of 5

A.39.1 The inverse of a null is a null

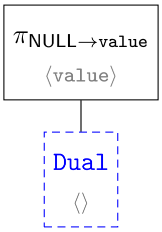
Query specification (The inverse of a null is a null)

```
RETURN NOT null AS value
```

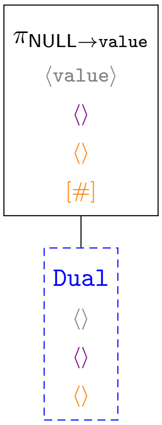
Relational algebra expression for search-based evaluation (The inverse of a null is a null)

$$r = \pi_{\mathsf{NULL} \rightarrow \texttt{value}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (The inverse of a null is a null)


$\pi_{\mathsf{NULL} \rightarrow \texttt{value}}$
$\langle \texttt{value} \rangle$
Dual
$\langle \rangle$

Incremental relational algebra tree (The inverse of a null is a null)



A.39.2 A literal null IS null

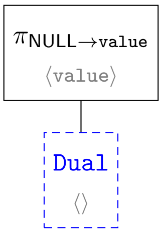
Query specification (A literal null IS null)

```
RETURN null IS NULL AS value
```

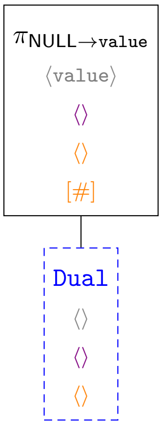
Relational algebra expression for search-based evaluation (A literal null IS null)

$$r = \pi_{\mathsf{NULL} \rightarrow \texttt{value}} \Big(\texttt{Dual} \Big)$$

Relational algebra tree for search-based evaluation (A literal null IS null)


$\pi_{\mathsf{NULL} \rightarrow \texttt{value}}$
$\langle \texttt{value} \rangle$
Dual
$\langle \rangle$

Incremental relational algebra tree (A literal null IS null)



A.39.3 A literal null is not IS NOT null

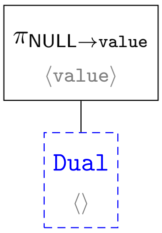
Query specification (A literal null is not IS NOT null)

```
RETURN null IS NOT NULL AS value
```

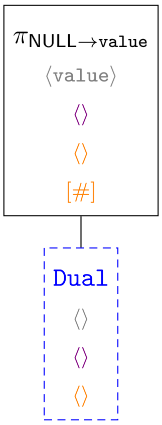
Relational algebra expression for search-based evaluation (A literal null is not IS NOT null)

$$r = \pi_{\mathsf{NULL} \rightarrow \texttt{value}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (A literal null is not IS NOT null)



Incremental relational algebra tree (A literal null is not IS NOT null)



A.39.4 It is unknown - i.e. null - if a null is equal to a null

Query specification (It is unknown - i.e. null - if a null is equal to a null)

```
RETURN null = null AS value
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.39.5 It is unknown - i.e. null - if a null is not equal to a null

Query specification (It is unknown - i.e. null - if a null is not equal to a null)

```
RETURN null <> null AS value
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40 TriadicSelection

Progress:

1 of 19

A.40.1 Handling triadic friend of a friend

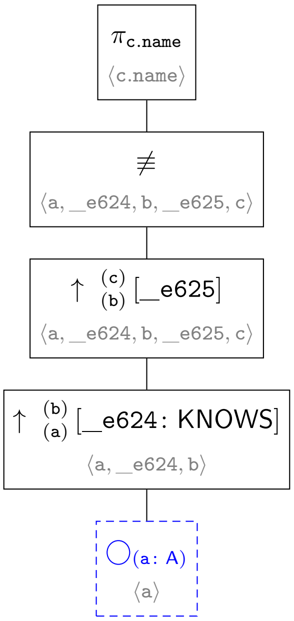
Query specification (Handling triadic friend of a friend)

```
MATCH (a:A)-[:KNOWS]->(b)-->(c)
RETURN c.name
```

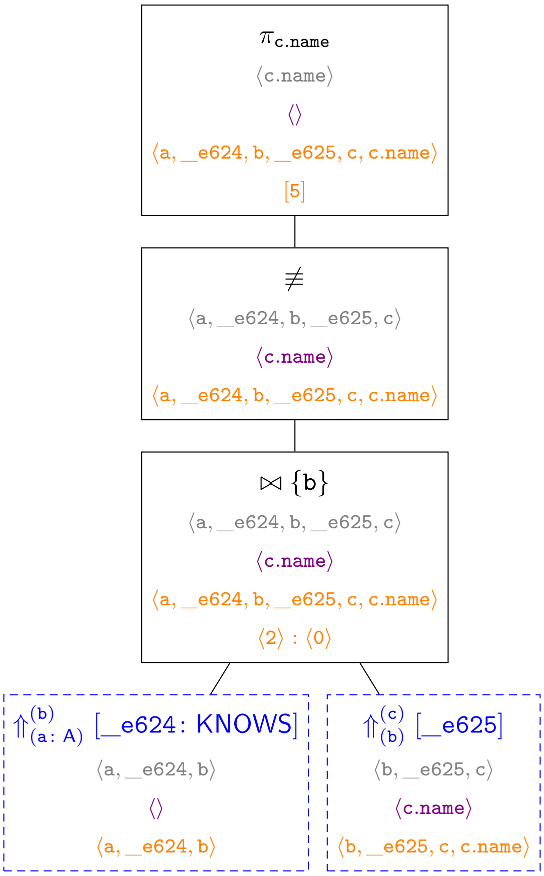
Relational algebra expression for search-based evaluation (Handling triadic friend of a friend)

$$r = \pi_{\texttt{c.name}}\Big(\not\equiv\Big(\uparrow\,{}^{(\mathsf{c})}_{(\mathsf{b})}\,[\_\mathsf{e625}]\,\Big(\uparrow\,{}^{(\mathsf{b})}_{(\mathsf{a})}\,[\_\mathsf{e624}\colon \mathsf{KNOWS}]\,\Big(\bigcirc_{(\mathsf{a}\colon\,\mathsf{A})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Handling triadic friend of a friend)



Incremental relational algebra tree (Handling triadic friend of a friend)



A.40.2 Handling triadic friend of a friend that is not a friend

Query specification (Handling triadic friend of a friend that is not a friend)

```
MATCH (a:A)-[:KNOWS]->(b)-->(c)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.3 Handling triadic friend of a friend that is not a friend with different relationship type

Query specification (Handling triadic friend of a friend that is not a friend with different relationship type)

```
MATCH (a:A)-[:KNOWS]->(b)-->(c)
OPTIONAL MATCH (a)-[r:FOLLOWS]->(c)
WITH c WHERE r IS NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.4 Handling triadic friend of a friend that is not a friend with superset of relationship type

Query specification (Handling triadic friend of a friend that is not a friend with superset of relationship type)

```
MATCH (a:A)-[:KNOWS]->(b)-->(c)
OPTIONAL MATCH (a)-[r]->(c)
WITH c WHERE r IS NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.5 Handling triadic friend of a friend that is not a friend with implicit subset of relationship type

Query specification (Handling triadic friend of a friend that is not a friend with implicit subset of relationship type)

```
MATCH (a:A)-->(b)-->(c)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.6 Handling triadic friend of a friend that is not a friend with explicit subset of relationship type

Query specification (Handling triadic friend of a friend that is not a friend with explicit subset of relationship type)

```
MATCH (a:A)-[:KNOWS|FOLLOWS]->(b)-->(c)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.7 Handling triadic friend of a friend that is not a friend with same labels

Query specification (Handling triadic friend of a friend that is not a friend with same labels)

```
MATCH (a:A)-[:KNOWS]->(b:X)-->(c:X)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.8 Handling triadic friend of a friend that is not a friend with different labels

Query specification (Handling triadic friend of a friend that is not a friend with different labels)

```
MATCH (a:A)-[:KNOWS]->(b:X)-->(c:Y)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.9 Handling triadic friend of a friend that is not a friend with implicit subset of labels

Query specification (Handling triadic friend of a friend that is not a friend with implicit subset of labels)

```
MATCH (a:A)-[:KNOWS]->(b)-->(c:X)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.10 Handling triadic friend of a friend that is not a friend with implicit superset of labels

Query specification (Handling triadic friend of a friend that is not a friend with implicit superset of labels)

```
MATCH (a:A)-[:KNOWS]->(b:X)-->(c)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.11 Handling triadic friend of a friend that is a friend

Query specification (Handling triadic friend of a friend that is a friend)

```
MATCH (a:A)-[:KNOWS]->(b)-->(c)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NOT NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.12 Handling triadic friend of a friend that is a friend with different relationship type

Query specification (Handling triadic friend of a friend that is a friend with different relationship type)

```
MATCH (a:A)-[:KNOWS]->(b)-->(c)
OPTIONAL MATCH (a)-[r:FOLLOWS]->(c)
WITH c WHERE r IS NOT NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.13 Handling triadic friend of a friend that is a friend with superset of relationship type

Query specification (Handling triadic friend of a friend that is a friend with superset of relationship type)

```
MATCH (a:A)-[:KNOWS]->(b)-->(c)
OPTIONAL MATCH (a)-[r]->(c)
WITH c WHERE r IS NOT NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.14 Handling triadic friend of a friend that is a friend with implicit subset of relationship type

Query specification (Handling triadic friend of a friend that is a friend with implicit subset of relationship type)

```
MATCH (a:A)-->(b)-->(c)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NOT NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.15 Handling triadic friend of a friend that is a friend with explicit subset of relationship type

Query specification (Handling triadic friend of a friend that is a friend with explicit subset of relationship type)

```
MATCH (a:A)-[:KNOWS|FOLLOWS]->(b)-->(c)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NOT NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.16 Handling triadic friend of a friend that is a friend with same labels

Query specification (Handling triadic friend of a friend that is a friend with same labels)

```
MATCH (a:A)-[:KNOWS]->(b:X)-->(c:X)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NOT NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.17 Handling triadic friend of a friend that is a friend with different labels

Query specification (Handling triadic friend of a friend that is a friend with different labels)

```
MATCH (a:A)-[:KNOWS]->(b:X)-->(c:Y)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NOT NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.18 Handling triadic friend of a friend that is a friend with implicit subset of labels

Query specification (Handling triadic friend of a friend that is a friend with implicit subset of labels)

```
MATCH (a:A)-[:KNOWS]->(b)-->(c:X)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NOT NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.40.19 Handling triadic friend of a friend that is a friend with implicit superset of labels

Query specification (Handling triadic friend of a friend that is a friend with implicit superset of labels)

```
MATCH (a:A)-[:KNOWS]->(b:X)-->(c)
OPTIONAL MATCH (a)-[r:KNOWS]->(c)
WITH c WHERE r IS NOT NULL
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.41 TypeConversionFunctions

Progress:

15 of 24

A.41.1 'toBoolean()' on valid literal string

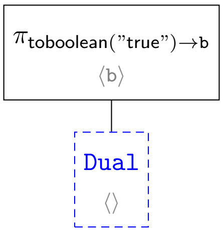
Query specification ('toBoolean()' on valid literal string)

```
RETURN toBoolean('true') AS b
```

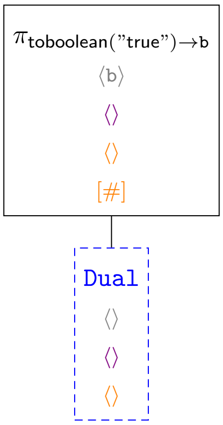
Relational algebra expression for search-based evaluation ('toBoolean()' on valid literal string)

$$r = \pi_{\text{toboolean(”true”)}\rightarrow\text{b}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation ('toBoolean()' on valid literal string)



Incremental relational algebra tree ('toBoolean()' on valid literal string)



A.41.2 'toBoolean()' on booleans

Query specification ('toBoolean()' on booleans)

```
UNWIND [true, false] AS b
RETURN toBoolean(b) AS b
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.41.3 'toBoolean()' on variables with valid string values

Query specification ('toBoolean()' on variables with valid string values)

```
UNWIND ['true', 'false'] AS s
RETURN toBoolean(s) AS b
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.41.4 'toBoolean()' on invalid strings

Query specification ('toBoolean()' on invalid strings)

```
UNWIND [null, '', ' tru ', 'f alse'] AS things
RETURN toBoolean(things) AS b
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.41.5 'toInteger()'

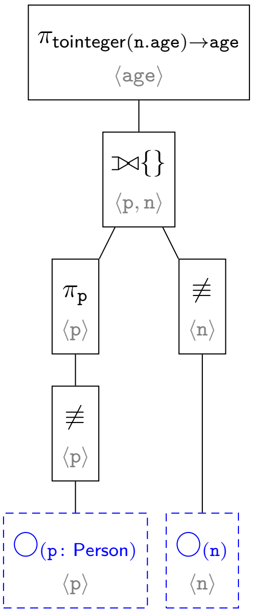
Query specification ('toInteger()')

```
MATCH (p:Person { age: '42' })
WITH *
MATCH (n)
RETURN toInteger(n.age) AS age
```

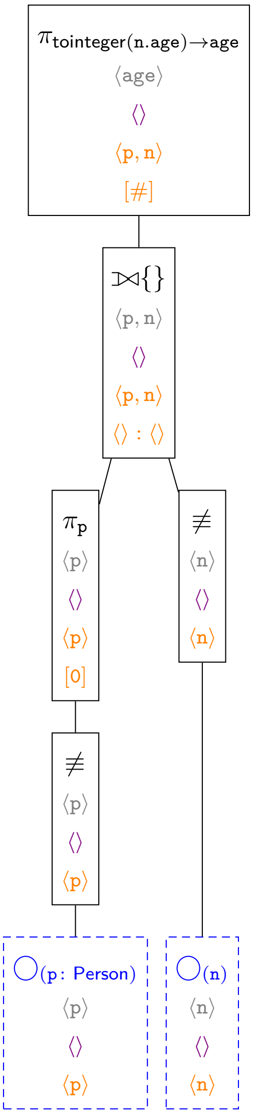
Relational algebra expression for search-based evaluation ('toInteger()')

$$r = \pi_{\mathsf{tointeger(n.age)} \rightarrow \mathsf{age}}\left(\pi_{\mathtt{p}}\left(\not\equiv\left(\bigcirc_{(\mathtt{p}:\ \mathsf{Person})}\right)\right) ⟕ \not\equiv\left(\bigcirc_{(\mathtt{n})}\right)\right)$$

Relational algebra tree for search-based evaluation ('toInteger()')



Incremental relational algebra tree ('toInteger()')



A.41.6 'toInteger()' on float

Query specification ('toInteger()' on float)

```
WITH 82.9 AS weight
RETURN toInteger(weight)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.41.7 'toInteger()' returning null on non-numerical string

Query specification ('toInteger()' returning null on non-numerical string)

```
WITH 'foo' AS foo_string, '' AS empty_string
RETURN toInteger(foo_string) AS foo, toInteger(empty_string) AS empty
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.41.8 'toInteger()' handling mixed number types

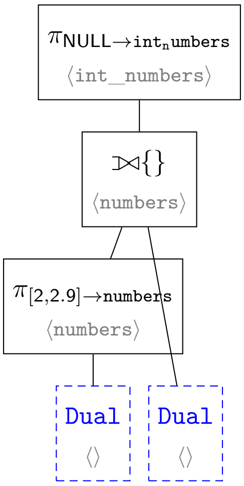
Query specification ('toInteger()' handling mixed number types)

```
WITH [2, 2.9] AS numbers
RETURN [n IN numbers | toInteger(n)] AS int_numbers
```

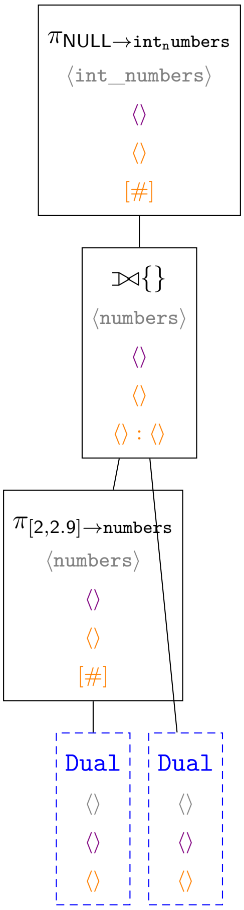
Relational algebra expression for search-based evaluation ('toInteger()' handling mixed number types)

$$r = \pi_{\mathsf{NULL} \rightarrow \mathtt{int}_{\mathtt{n}}\mathtt{umbers}}\left(\pi_{[2,2.9] \rightarrow \mathtt{numbers}}\left(\mathtt{Dual}\right) \bowtie \mathtt{Dual}\right)$$

Relational algebra tree for search-based evaluation ('toInteger()' handling mixed number types)


$\pi_{\mathsf{NULL} \rightarrow \mathtt{int}_{\mathtt{n}}\mathtt{umbers}}$
⟨int_numbers⟩
⋈{}
⟨numbers⟩
$\pi_{[2,2.9] \rightarrow \mathtt{numbers}}$
⟨numbers⟩
Dual
⟨⟩
Dual
⟨⟩

Incremental relational algebra tree ('toInteger()' handling mixed number types)


$\pi_{\text{NULL}\rightarrow\texttt{int}_{\texttt{n}}\texttt{umbers}}$
⟨int_numbers⟩
⟨⟩
⟨⟩
[#]
⟕{}
⟨numbers⟩
⟨⟩
⟨⟩
⟨⟩ : ⟨⟩
$\pi_{[2,2.9]\rightarrow\texttt{numbers}}$
⟨numbers⟩
⟨⟩
⟨⟩
[#]
Dual
⟨⟩
⟨⟩
⟨⟩
Dual
⟨⟩
⟨⟩
⟨⟩

A.41.9 'toInteger()' handling Any type

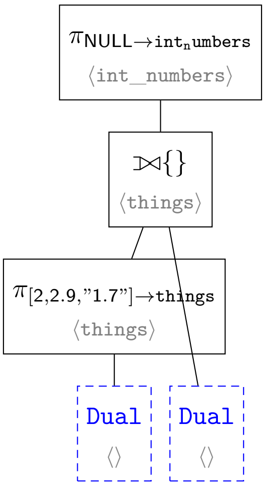
Query specification ('toInteger()' handling Any type)

```
WITH [2, 2.9, '1.7'] AS things
RETURN [n IN things | toInteger(n)] AS int_numbers
```

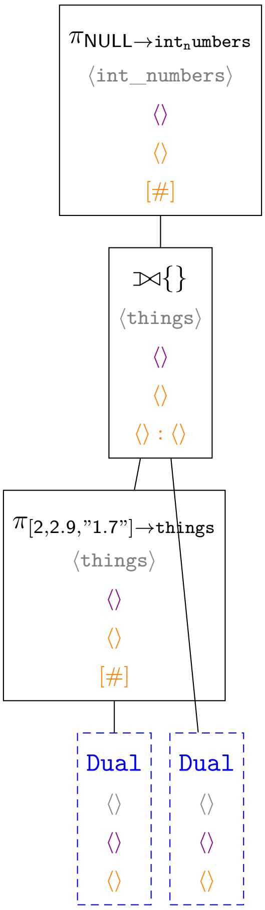
Relational algebra expression for search-based evaluation ('toInteger()' handling Any type)

$$r = \pi_{\text{NULL}\rightarrow\texttt{int}_{\texttt{n}}\texttt{umbers}}\left(\pi_{[2,2.9,"1.7"]\rightarrow\texttt{things}}\left(\texttt{Dual}\right) ⟕ \texttt{Dual}\right)$$

Relational algebra tree for search-based evaluation ('toInteger()' handling Any type)



Incremental relational algebra tree ('toInteger()' handling Any type)



A.41.10 'toInteger()' on a list of strings

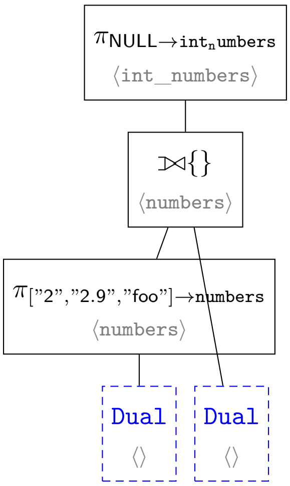
Query specification ('toInteger()' on a list of strings)

```
WITH ['2', '2.9', 'foo'] AS numbers
RETURN [n IN numbers | toInteger(n)] AS int_numbers
```

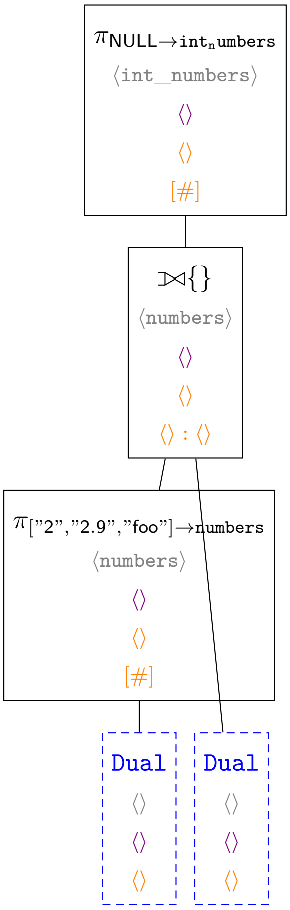
Relational algebra expression for search-based evaluation ('toInteger()' on a list of strings)

$$r = \pi_{\mathsf{NULL}\rightarrow\mathtt{int}_{\mathtt{n}}\mathtt{umbers}}\left(\pi_{[\text{"2","2.9","foo"}]\rightarrow\mathtt{numbers}}\left(\mathtt{Dual}\right) ⟕ \mathtt{Dual}\right)$$

Relational algebra tree for search-based evaluation ('toInteger()' on a list of strings)


$\pi_{\mathsf{NULL}\rightarrow\mathtt{int}_{\mathtt{n}}\mathtt{umbers}}$
⟨int_numbers⟩
⟕{}
⟨numbers⟩
$\pi_{[\text{"2","2.9","foo"}]\rightarrow\mathtt{numbers}}$
⟨numbers⟩
Dual
⟨⟩
Dual
⟨⟩

Incremental relational algebra tree ('toInteger()' on a list of strings)


$\pi_{\mathsf{NULL}\rightarrow\mathtt{int}_{\mathtt{n}}\mathtt{umbers}}$
⟨int_numbers⟩
⟨⟩
⟨⟩
[#]
⟕{}
⟨numbers⟩
⟨⟩
⟨⟩
⟨⟩ : ⟨⟩
$\pi_{[\text{"2","2.9","foo"}]\rightarrow\mathtt{numbers}}$
⟨numbers⟩
⟨⟩
⟨⟩
[#]
Dual
⟨⟩
⟨⟩
⟨⟩
Dual
⟨⟩
⟨⟩
⟨⟩

A.41.11 'toInteger()' on a complex-typed expression

Query specification ('toInteger()' on a complex-typed expression)

```
RETURN toInteger(1 - {param}) AS result
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.41.12 'toFloat()'

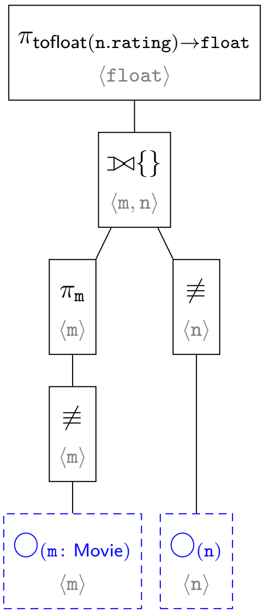
Query specification ('toFloat()')

```
MATCH (m:Movie { rating: 4 })
WITH *
MATCH (n)
RETURN toFloat(n.rating) AS float
```

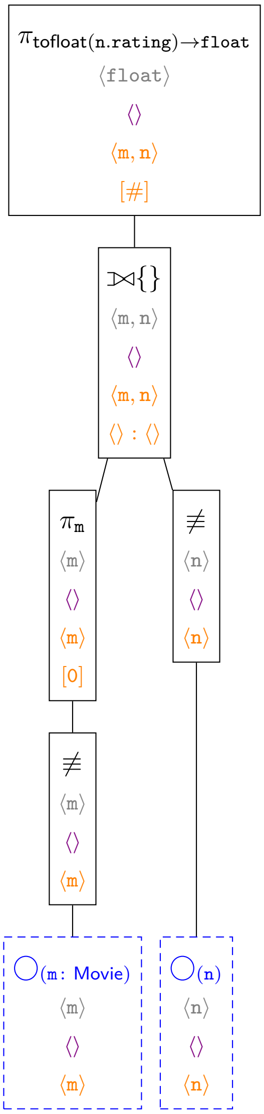
Relational algebra expression for search-based evaluation ('toFloat()')

$$r = \pi_{\mathsf{tofloat}(\mathtt{n.rating}) \rightarrow \mathtt{float}} \left(\pi_{\mathtt{m}} \left(\not\equiv \left(\bigcirc_{(\mathtt{m}:\ \mathsf{Movie})} \right) \right) \mathbin{⟕} \not\equiv \left(\bigcirc_{(\mathtt{n})} \right) \right)$$

Relational algebra tree for search-based evaluation ('toFloat()')


$\pi_{\mathsf{tofloat}(\mathtt{n.rating}) \rightarrow \mathtt{float}}$
⟨float⟩
⟕{}
⟨m, n⟩
$\pi_{\mathtt{m}}$
⟨m⟩
≢
⟨n⟩
≢
⟨m⟩
$\bigcirc_{(\mathtt{m}:\ \mathsf{Movie})}$
⟨m⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩

Incremental relational algebra tree ('toFloat()')



A.41.13 'toFloat()' on mixed number types

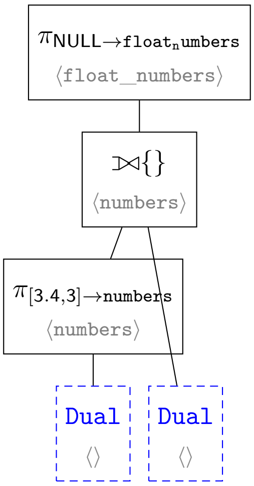
Query specification ('toFloat()' on mixed number types)

```
WITH [3.4, 3] AS numbers
RETURN [n IN numbers | toFloat(n)] AS float_numbers
```

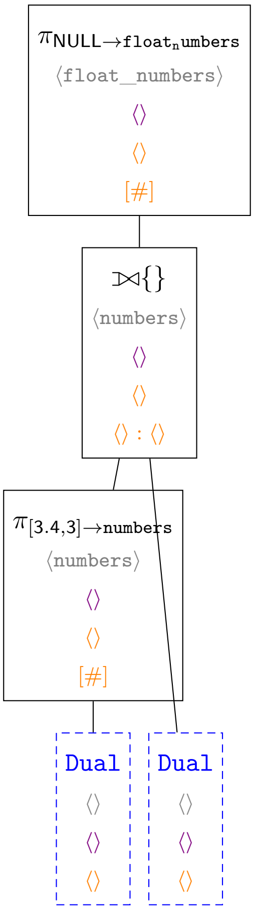
Relational algebra expression for search-based evaluation ('toFloat()' on mixed number types)

$$r = \pi_{\mathsf{NULL}\rightarrow\mathtt{float}_{\mathtt{n}}\mathtt{umbers}}\left(\pi_{[3.4,3]\rightarrow\mathtt{numbers}}\left(\mathtt{Dual}\right)\Join\mathtt{Dual}\right)$$

Relational algebra tree for search-based evaluation (’toFloat()’ on mixed number types)



Incremental relational algebra tree (’toFloat()’ on mixed number types)



A.41.14 ’toFloat()’ returning null on non-numerical string

Query specification (’toFloat()’ returning null on non-numerical string)

```
WITH 'foo' AS foo_string, '' AS empty_string
RETURN toFloat(foo_string) AS foo, toFloat(empty_string) AS empty
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.41.15 'toFloat()' handling Any type

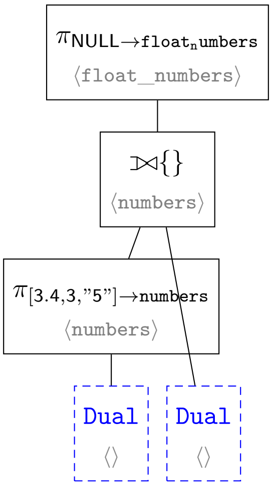
Query specification ('toFloat()' handling Any type)

```
WITH [3.4, 3, '5'] AS numbers
RETURN [n IN numbers | toFloat(n)] AS float_numbers
```

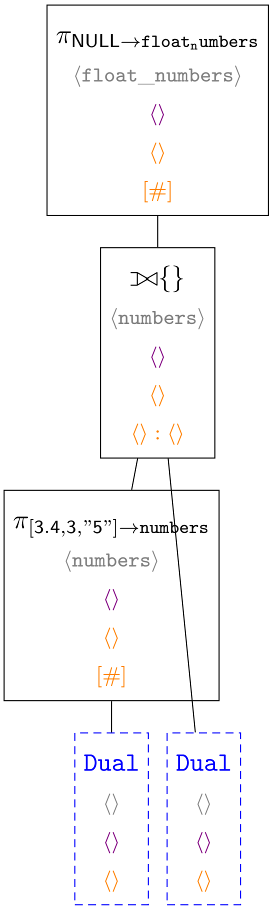
Relational algebra expression for search-based evaluation ('toFloat()' handling Any type)

$$r = \pi_{\mathsf{NULL} \rightarrow \mathtt{float_numbers}}\left(\pi_{[3.4,3,"5"] \rightarrow \mathtt{numbers}}\left(\mathtt{Dual}\right) \bowtie \mathtt{Dual}\right)$$

Relational algebra tree for search-based evaluation ('toFloat()' handling Any type)


$\pi_{\mathsf{NULL} \rightarrow \mathtt{float_numbers}}$
⟨float_numbers⟩
⋈{}
⟨numbers⟩
$\pi_{[3.4,3,"5"] \rightarrow \mathtt{numbers}}$
⟨numbers⟩
Dual
⟨⟩
Dual
⟨⟩

Incremental relational algebra tree ('toFloat()' handling Any type)



A.41.16 'toFloat()' on a list of strings

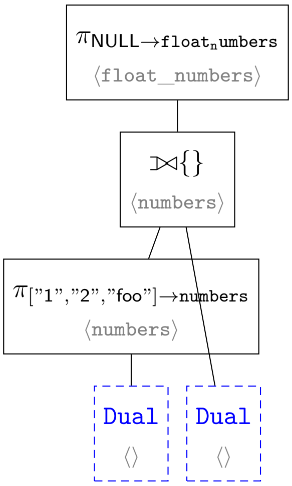
Query specification ('toFloat()' on a list of strings)

```
WITH ['1', '2', 'foo'] AS numbers
RETURN [n IN numbers | toFloat(n)] AS float_numbers
```

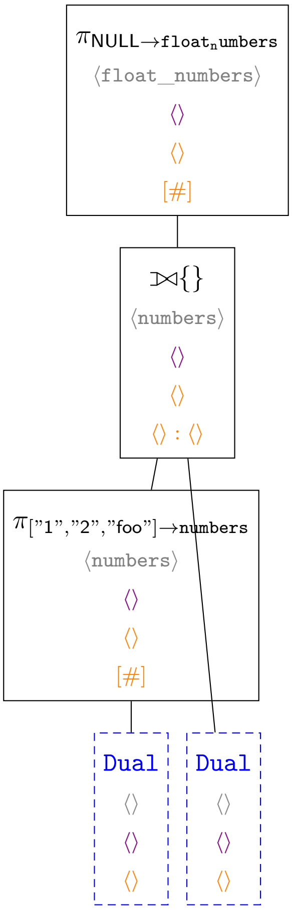
Relational algebra expression for search-based evaluation ('toFloat()' on a list of strings)

$$r = \pi_{\mathsf{NULL}\rightarrow\texttt{float}_{\texttt{n}}\texttt{umbers}}\left(\pi_{[\text{"1","2","foo"}]\rightarrow\texttt{numbers}}\left(\texttt{Dual}\right)\mathbin{⟕}\texttt{Dual}\right)$$

Relational algebra tree for search-based evaluation ('toFloat()' on a list of strings)



Incremental relational algebra tree ('toFloat()' on a list of strings)



A.41.17 'toString()'

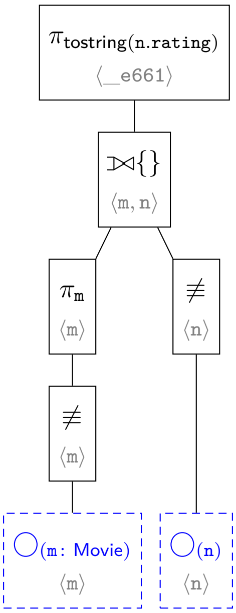
Query specification ('toString()')

```
MATCH (m:Movie { rating: 4 })
WITH *
MATCH (n)
RETURN toString(n.rating)
```

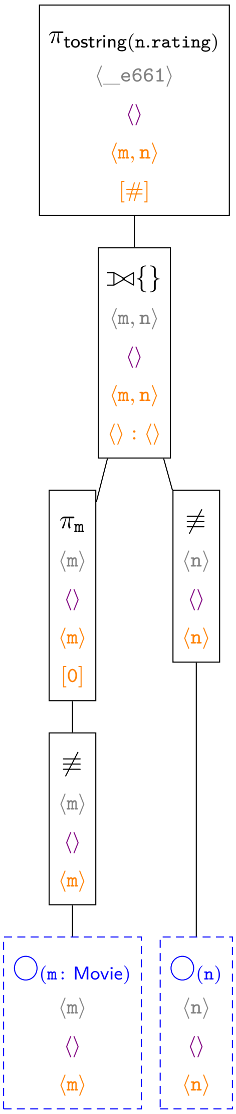
Relational algebra expression for search-based evaluation ('toString()')

$$r = \pi_{\mathsf{tostring}(\mathtt{n.rating})}\left(\pi_{\mathtt{m}}\left(\not\equiv\left(\bigcirc_{(\mathtt{m}:\ \mathsf{Movie})}\right)\right)\text{⟕}\not\equiv\left(\bigcirc_{(\mathtt{n})}\right)\right)$$

Relational algebra tree for search-based evaluation ('toString()')


$\pi_{\mathsf{tostring}(\mathtt{n.rating})}$
⟨_e661⟩
⟕{}
⟨m, n⟩
$\pi_{\mathtt{m}}$
⟨m⟩
≢
⟨n⟩
≢
⟨m⟩
$\bigcirc_{(\mathtt{m}:\ \mathsf{Movie})}$
⟨m⟩
$\bigcirc_{(\mathtt{n})}$
⟨n⟩

Incremental relational algebra tree ('toString()')



A.41.18 'toString()' handling boolean properties

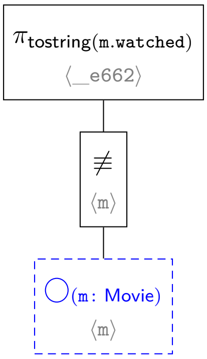
Query specification ('toString()' handling boolean properties)

```
MATCH (m:Movie)
RETURN toString(m.watched)
```

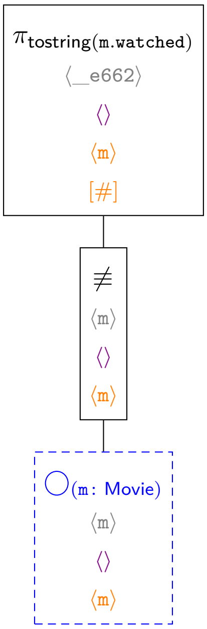
Relational algebra expression for search-based evaluation ('toString()' handling boolean properties)

$$r = \pi_{\text{tostring(m.watched)}}\Big(\not\equiv\Big(\bigcirc_{(\text{m: Movie})}\Big)\Big)$$

Relational algebra tree for search-based evaluation ('toString()' handling boolean properties)



Incremental relational algebra tree ('toString()' handling boolean properties)



A.41.19 'toString()' handling inlined boolean

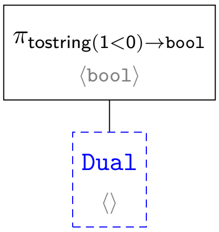
Query specification ('toString()' handling inlined boolean)

```
RETURN toString(1 < 0) AS bool
```

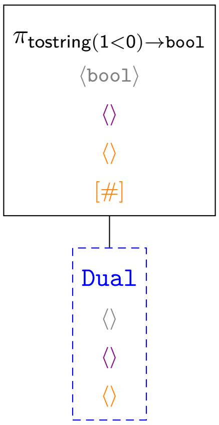
Relational algebra expression for search-based evaluation ('toString()' handling inlined boolean)

$$r = \pi_{\text{tostring}(1<0)\to\texttt{bool}}\Big(\texttt{Dual}\Big)$$

Relational algebra tree for search-based evaluation ('toString()' handling inlined boolean)



Incremental relational algebra tree ('toString()' handling inlined boolean)



A.41.20 'toString()' handling boolean literal

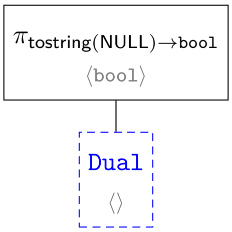
Query specification ('toString()' handling boolean literal)

```
RETURN toString(true) AS bool
```

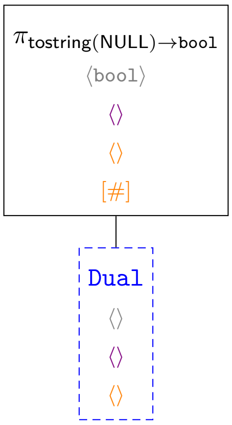
Relational algebra expression for search-based evaluation ('toString()' handling boolean literal)

$$r = \pi_{\mathsf{tostring(NULL)}\to\texttt{bool}}\left(\texttt{Dual}\right)$$

Relational algebra tree for search-based evaluation ('toString()' handling boolean literal)



Incremental relational algebra tree ('toString()' handling boolean literal)



A.41.21 'toString()' should work on Any type

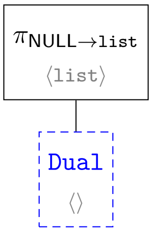
Query specification ('toString()' should work on Any type)

```
RETURN [x IN [1, 2.3, true, 'apa'] | toString(x) ] AS list
```

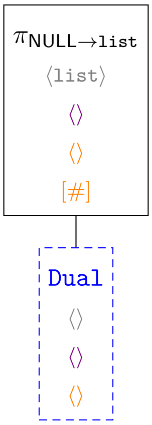
Relational algebra expression for search-based evaluation ('toString()' should work on Any type)

$$r = \pi_{\mathsf{NULL} \rightarrow \texttt{list}} \Big(\texttt{Dual} \Big)$$

Relational algebra tree for search-based evaluation ('toString()' should work on Any type)



Incremental relational algebra tree ('toString()' should work on Any type)



A.41.22 'toString()' on a list of integers

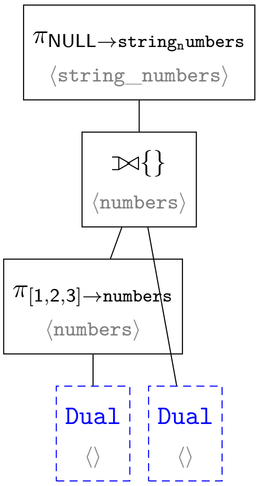
Query specification ('toString()' on a list of integers)

```
WITH [1, 2, 3] AS numbers
RETURN [n IN numbers | toString(n)] AS string_numbers
```

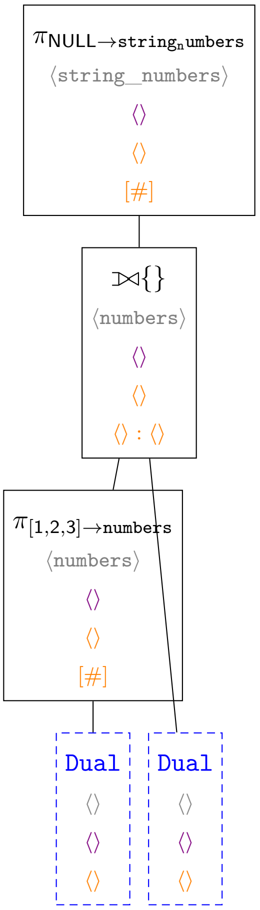
Relational algebra expression for search-based evaluation ('toString()' on a list of integers)

$$r = \pi_{\mathsf{NULL} \rightarrow \texttt{string}_{\texttt{n}}\texttt{umbers}} \Big(\pi_{[1,2,3] \rightarrow \texttt{numbers}} \Big(\texttt{Dual} \Big) \rtimes\!\!\!\ltimes \texttt{Dual} \Big)$$

Relational algebra tree for search-based evaluation (’toString()’ on a list of integers)


$\pi_{\mathsf{NULL} \rightarrow \mathtt{string_numbers}}$
⟨string_numbers⟩
⋈{}
⟨numbers⟩
$\pi_{[1,2,3] \rightarrow \mathtt{numbers}}$
⟨numbers⟩
Dual
⟨⟩
Dual
⟨⟩

Incremental relational algebra tree (’toString()’ on a list of integers)


$\pi_{\mathsf{NULL} \rightarrow \mathtt{string_numbers}}$
⟨string_numbers⟩
⟨⟩
⟨⟩
[#]
⋈{}
⟨numbers⟩
⟨⟩
⟨⟩
⟨⟩ : ⟨⟩
$\pi_{[1,2,3] \rightarrow \mathtt{numbers}}$
⟨numbers⟩
⟨⟩
⟨⟩
[#]
Dual
⟨⟩
⟨⟩
⟨⟩
Dual
⟨⟩
⟨⟩
⟨⟩

A.41.23 ’toString()’ should accept potentially correct types 1

Query specification (’toString()’ should accept potentially correct types 1)

```
UNWIND ['male', 'female', null] AS gen
RETURN coalesce(toString(gen), 'x') AS result
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.41.24 'toString()' should accept potentially correct types 2

Query specification ('toString()' should accept potentially correct types 2)

```
UNWIND ['male', 'female', null] AS gen
RETURN toString(coalesce(gen, 'x')) AS result
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.42 UnionAcceptance

Progress:

5 of 5

A.42.1 Should be able to create text output from union queries

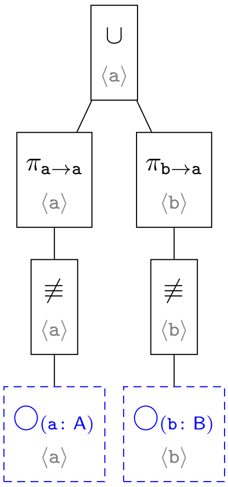
Query specification (Should be able to create text output from union queries)

```
MATCH (a:A)
RETURN a AS a
UNION
MATCH (b:B)
RETURN b AS a
```

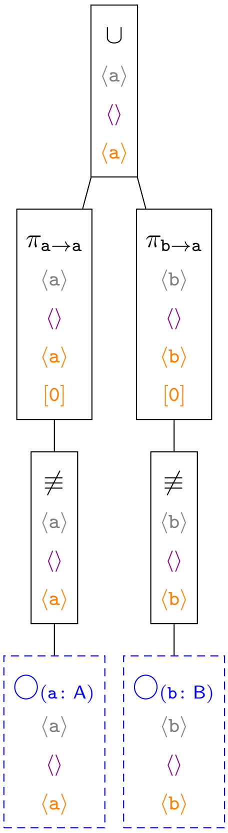
Relational algebra expression for search-based evaluation (Should be able to create text output from union queries)

$$r = \pi_{\mathtt{a}\rightarrow\mathtt{a}}\left(\not\equiv\left(\bigcirc_{(\mathtt{a:\ A})}\right)\right) \cup \pi_{\mathtt{b}\rightarrow\mathtt{a}}\left(\not\equiv\left(\bigcirc_{(\mathtt{b:\ B})}\right)\right)$$

Relational algebra tree for search-based evaluation (Should be able to create text output from union queries)



Incremental relational algebra tree (Should be able to create text output from union queries)



A.42.2 Two elements, both unique, not distinct

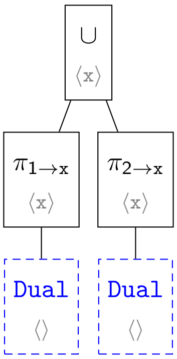
Query specification (Two elements, both unique, not distinct)

```
RETURN 1 AS x
UNION ALL
RETURN 2 AS x
```

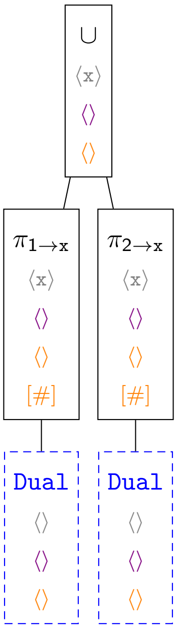
Relational algebra expression for search-based evaluation (Two elements, both unique, not distinct)

$$r = \pi_{1 \to \mathtt{x}}\Big(\mathtt{Dual}\Big) \cup \pi_{2 \to \mathtt{x}}\Big(\mathtt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Two elements, both unique, not distinct)



Incremental relational algebra tree (Two elements, both unique, not distinct)



A.42.3 Two elements, both unique, distinct

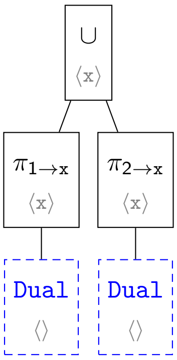
Query specification (Two elements, both unique, distinct)

```
RETURN 1 AS x
UNION
RETURN 2 AS x
```

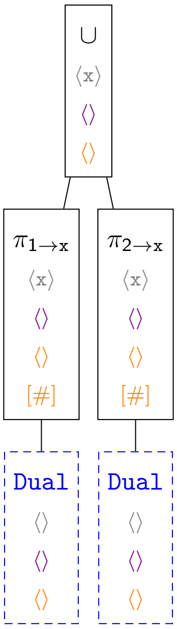
Relational algebra expression for search-based evaluation (Two elements, both unique, distinct)

$$r = \pi_{1 \to \mathrm{x}}\left(\texttt{Dual}\right) \cup \pi_{2 \to \mathrm{x}}\left(\texttt{Dual}\right)$$

Relational algebra tree for search-based evaluation (Two elements, both unique, distinct)



Incremental relational algebra tree (Two elements, both unique, distinct)



A.42.4 Three elements, two unique, distinct

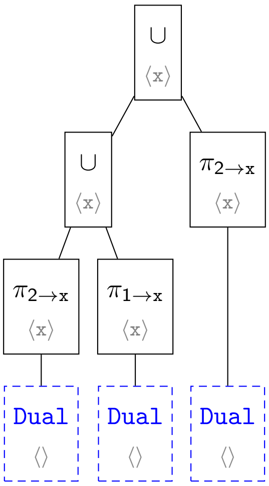
Query specification (Three elements, two unique, distinct)

```
RETURN 2 AS x
UNION
RETURN 1 AS x
UNION
RETURN 2 AS x
```

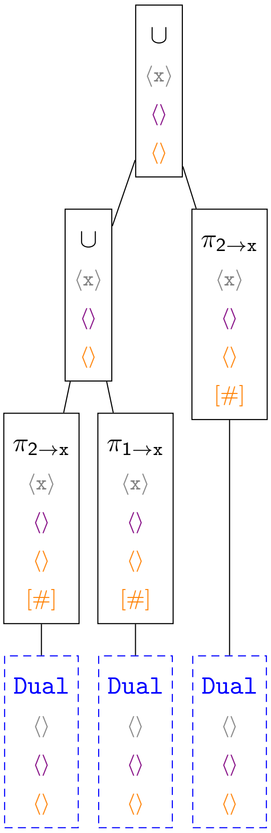
Relational algebra expression for search-based evaluation (Three elements, two unique, distinct)

$$r = \pi_{2\to\mathrm{x}}\Big(\mathtt{Dual}\Big) \cup \pi_{1\to\mathrm{x}}\Big(\mathtt{Dual}\Big) \cup \pi_{2\to\mathrm{x}}\Big(\mathtt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Three elements, two unique, distinct)



Incremental relational algebra tree (Three elements, two unique, distinct)



A.42.5 Three elements, two unique, not distinct

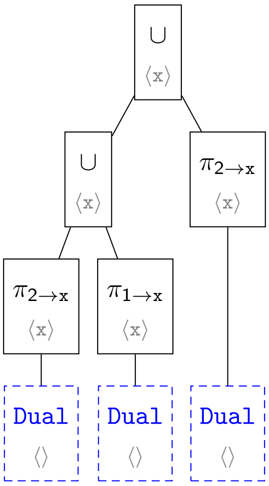
Query specification (Three elements, two unique, not distinct)

```
RETURN 2 AS x
UNION ALL
RETURN 1 AS x
UNION ALL
RETURN 2 AS x
```

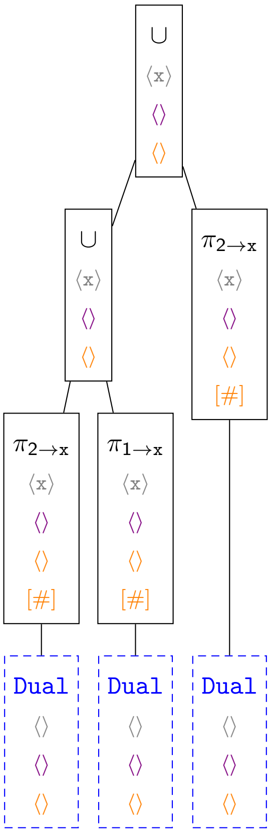
Relational algebra expression for search-based evaluation (Three elements, two unique, not distinct)

$$r = \pi_{2\rightarrow\mathtt{x}}\Big(\mathtt{Dual}\Big) \cup \pi_{1\rightarrow\mathtt{x}}\Big(\mathtt{Dual}\Big) \cup \pi_{2\rightarrow\mathtt{x}}\Big(\mathtt{Dual}\Big)$$

Relational algebra tree for search-based evaluation (Three elements, two unique, not distinct)



Incremental relational algebra tree (Three elements, two unique, not distinct)



A.43 UnwindAcceptance

Progress:

0 of 12

A.43.1 Unwinding a list

Query specification (Unwinding a list)

```
UNWIND [1, 2, 3] AS x
RETURN x
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.43.2 Unwinding a range

Query specification (Unwinding a range)

```
UNWIND range(1, 3) AS x
RETURN x
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.43.3 Unwinding a concatenation of lists

Query specification (Unwinding a concatenation of lists)

```
WITH [1, 2, 3] AS first, [4, 5, 6] AS second
UNWIND (first + second) AS x
RETURN x
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.43.4 Unwinding a collected unwound expression

Query specification (Unwinding a collected unwound expression)

```
UNWIND RANGE(1, 2) AS row
WITH collect(row) AS rows
UNWIND rows AS x
RETURN x
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.43.5 Unwinding a collected expression

Query specification (Unwinding a collected expression)

```
MATCH (row)
WITH collect(row) AS rows
UNWIND rows AS node
RETURN node.id
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.43.6 Double unwinding a list of lists

Query specification (Double unwinding a list of lists)

```
WITH [[1, 2, 3], [4, 5, 6]] AS lol
UNWIND lol AS x
UNWIND x AS y
RETURN y
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.43.7 Unwinding the empty list

Query specification (Unwinding the empty list)

```
UNWIND [] AS empty
RETURN empty
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.43.8 Unwinding null

Query specification (Unwinding null)

```
UNWIND null AS nil
RETURN nil
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.43.9 Unwinding list with duplicates

Query specification (Unwinding list with duplicates)

```
UNWIND [1, 1, 2, 2, 3, 3, 4, 4, 5, 5] AS duplicate
RETURN duplicate
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.43.10 Unwind does not prune context

Query specification (Unwind does not prune context)

```
WITH [1, 2, 3] AS list
UNWIND list AS x
RETURN *
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.43.11 Unwind does not remove variables from scope

Query specification (Unwind does not remove variables from scope)

```
MATCH (a:S)-[:X]->(b1)
WITH a, collect(b1) AS bees
UNWIND bees AS b2
MATCH (a)-[:Y]->(b2)
RETURN a, b2
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.43.12 Multiple unwinds after each other

Query specification (Multiple unwinds after each other)

```
WITH [1, 2] AS xs, [3, 4] AS ys, [5, 6] AS zs
UNWIND xs AS x
UNWIND ys AS y
UNWIND zs AS z
RETURN *
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44 VarLengthAcceptance

Progress:

15 of 29

A.44.1 Handling unbounded variable length match

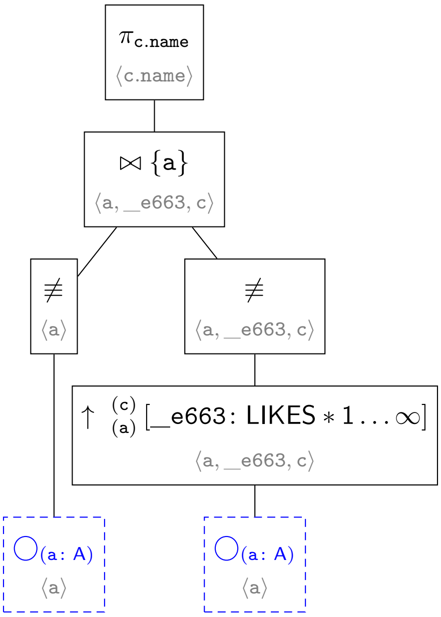
Query specification (Handling unbounded variable length match)

```
MATCH (a:A)
MATCH (a)-[:LIKES*]->(c)
RETURN c.name
```

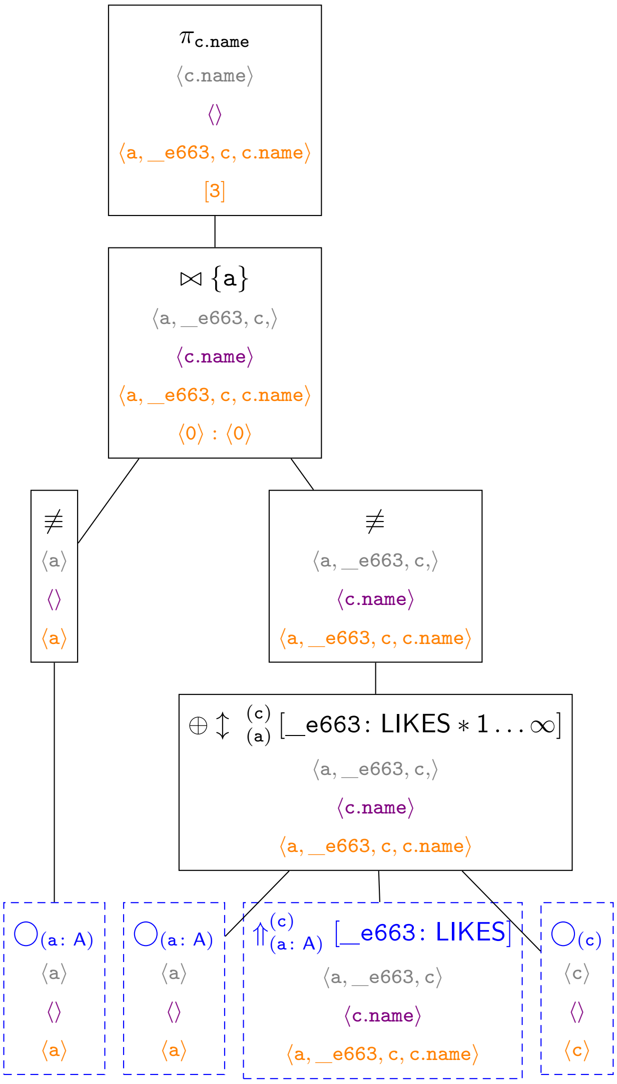
Relational algebra expression for search-based evaluation (Handling unbounded variable length match)

$$r = \pi_{\texttt{c.name}}\Big(\not\equiv \Big(\bigcirc_{(\texttt{a}:\ \mathsf{A})} \Big) \bowtie\!\not\equiv \Big(\uparrow\ {}^{(\texttt{c})}_{(\texttt{a})} [\_\mathsf{e663}\colon \mathsf{LIKES} * 1 \ldots \infty] \Big(\bigcirc_{(\texttt{a}:\ \mathsf{A})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Handling unbounded variable length match)



Incremental relational algebra tree (Handling unbounded variable length match)



A.44.2 Handling explicitly unbounded variable length match

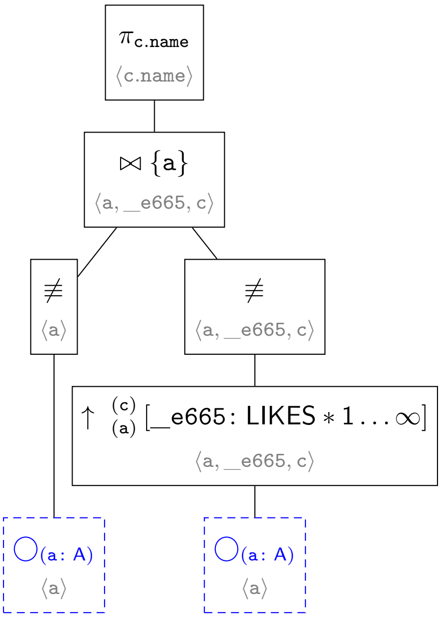
Query specification (Handling explicitly unbounded variable length match)

```
MATCH (a:A)
MATCH (a)-[:LIKES*..]->(c)
RETURN c.name
```

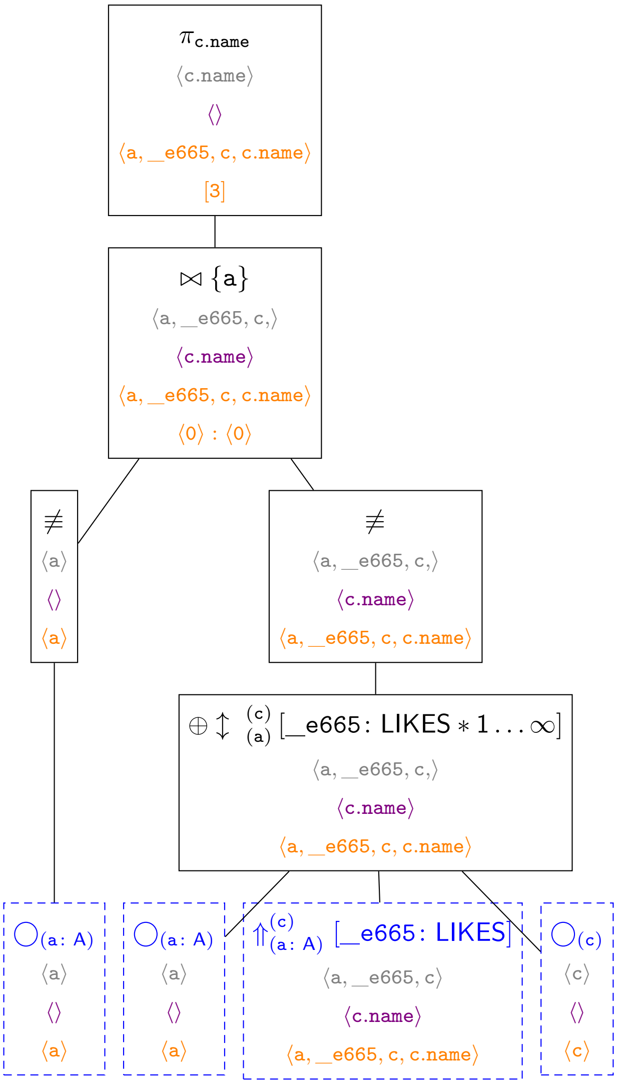
Relational algebra expression for search-based evaluation (Handling explicitly unbounded variable length match)

$$r = \pi_{\texttt{c.name}}\Big(\not\equiv\Big(\bigcirc_{(\mathsf{a}\colon\,\mathsf{A})}\Big)\bowtie\not\equiv\Big(\uparrow\,_{(\mathsf{a})}^{(\mathsf{c})}\,[\_\mathsf{e665}\colon\mathsf{LIKES}*1\ldots\infty]\Big(\bigcirc_{(\mathsf{a}\colon\,\mathsf{A})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Handling explicitly unbounded variable length match)



Incremental relational algebra tree (Handling explicitly unbounded variable length match)



A.44.3 Handling single bounded variable length match 1

Query specification (Handling single bounded variable length match 1)

```
MATCH (a:A)
MATCH (a)-[:LIKES*0]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44.4 Handling single bounded variable length match 2

Query specification (Handling single bounded variable length match 2)

```
MATCH (a:A)
MATCH (a)-[:LIKES*1]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44.5 Handling single bounded variable length match 3

Query specification (Handling single bounded variable length match 3)

```
MATCH (a:A)
MATCH (a)-[:LIKES*2]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44.6 Handling upper and lower bounded variable length match 1

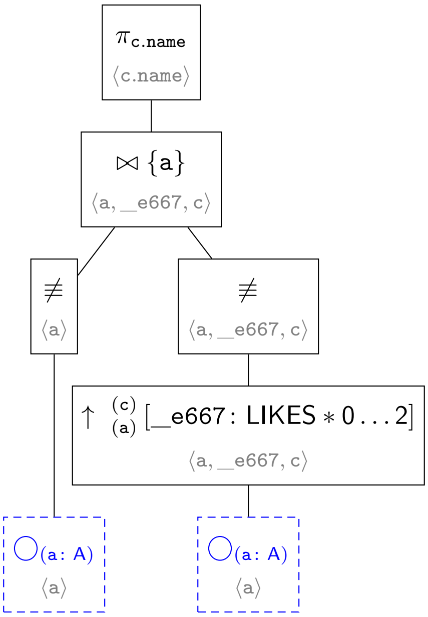
Query specification (Handling upper and lower bounded variable length match 1)

```
MATCH (a:A)
MATCH (a)-[:LIKES*0..2]->(c)
RETURN c.name
```

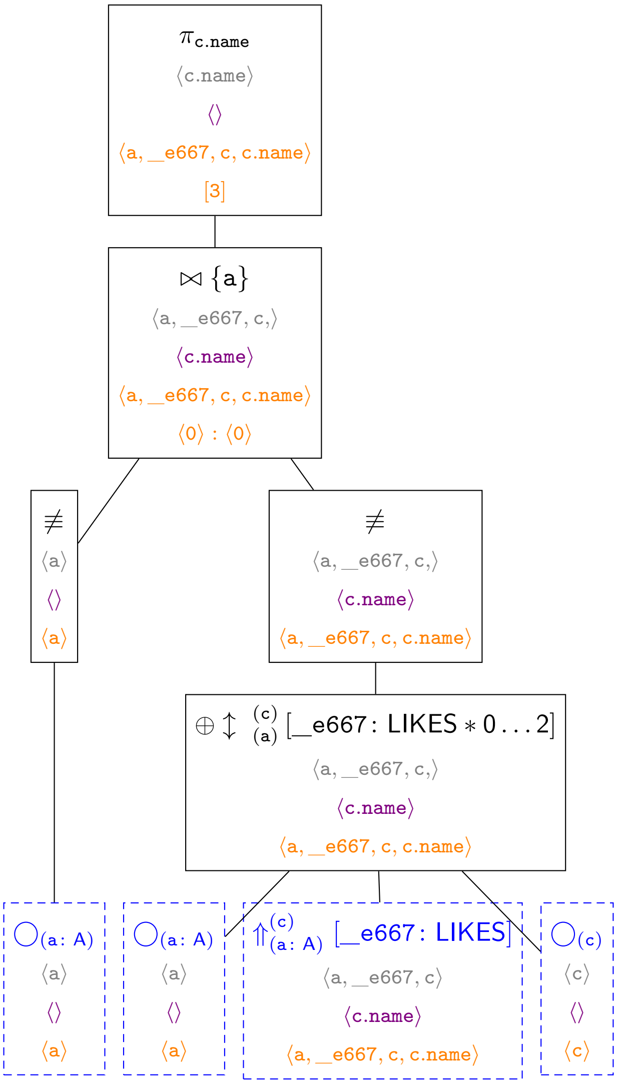
Relational algebra expression for search-based evaluation (Handling upper and lower bounded variable length match 1)

$$r = \pi_{\texttt{c.name}}\Big(\not\equiv\big(\bigcirc_{(\mathsf{a}:\,\mathsf{A})}\big) \bowtie_{\not\equiv} \Big(\uparrow^{(\mathsf{c})}_{(\mathsf{a})} [\_\mathsf{e667}\colon \mathsf{LIKES} * 0 \ldots 2] \big(\bigcirc_{(\mathsf{a}:\,\mathsf{A})}\big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Handling upper and lower bounded variable length match 1)



Incremental relational algebra tree (Handling upper and lower bounded variable length match 1)



A.44.7 Handling upper and lower bounded variable length match 2

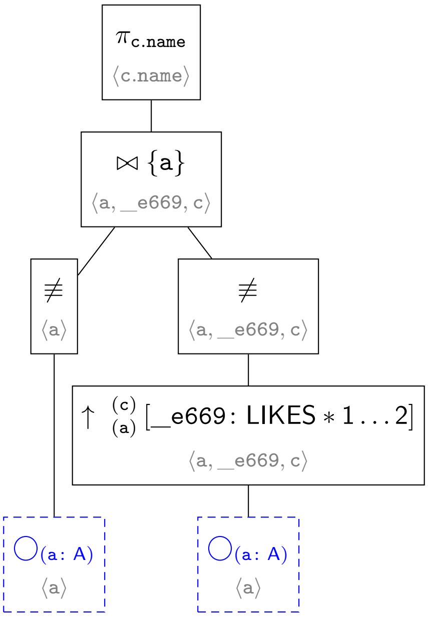
Query specification (Handling upper and lower bounded variable length match 2)

```
MATCH (a:A)
MATCH (a)-[:LIKES*1..2]->(c)
RETURN c.name
```

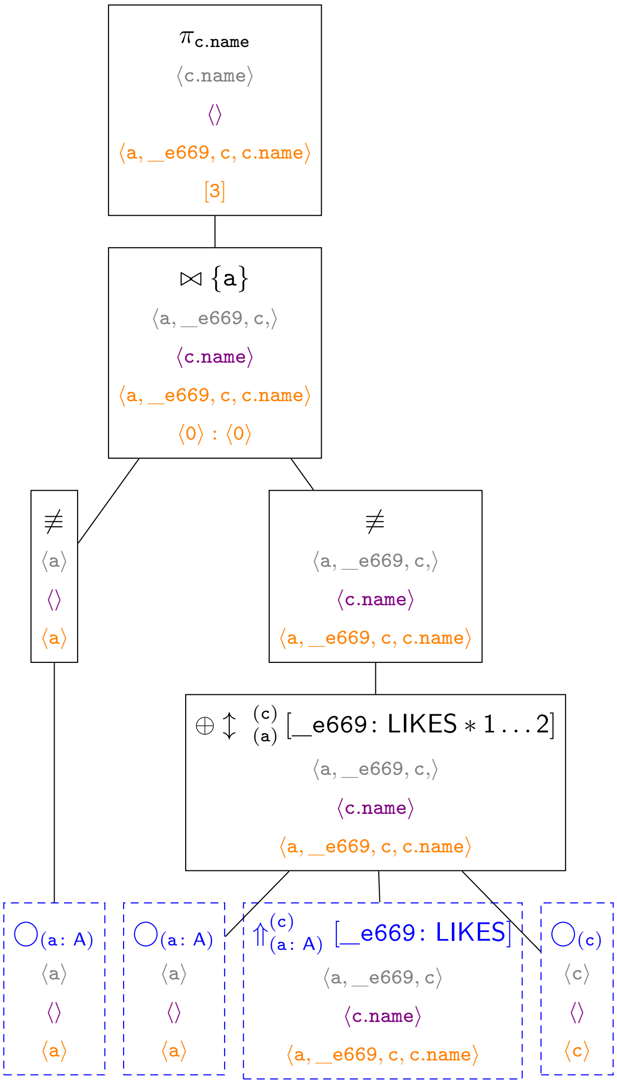
Relational algebra expression for search-based evaluation (Handling upper and lower bounded variable length match 2)

$$r = \pi_{\texttt{c.name}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{a: A})}\Big) \bowtie\!\not\equiv \Big(\uparrow {}^{(\texttt{c})}_{(\texttt{a})} [\_e669\colon \textsf{LIKES} * 1 \ldots 2] \Big(\bigcirc_{(\texttt{a: A})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation **(Handling upper and lower bounded variable length match 2)**



Incremental relational algebra tree (Handling upper and lower bounded variable length match 2)



A.44.8 Handling symmetrically bounded variable length match, bounds are zero

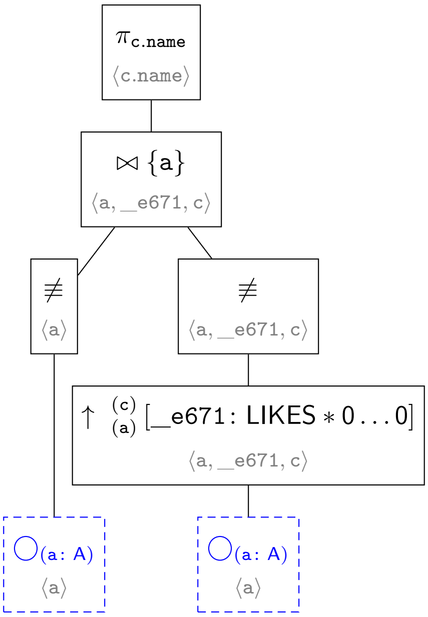
Query specification (Handling symmetrically bounded variable length match, bounds are zero)

```
MATCH (a:A)
MATCH (a)-[:LIKES*0..0]->(c)
RETURN c.name
```

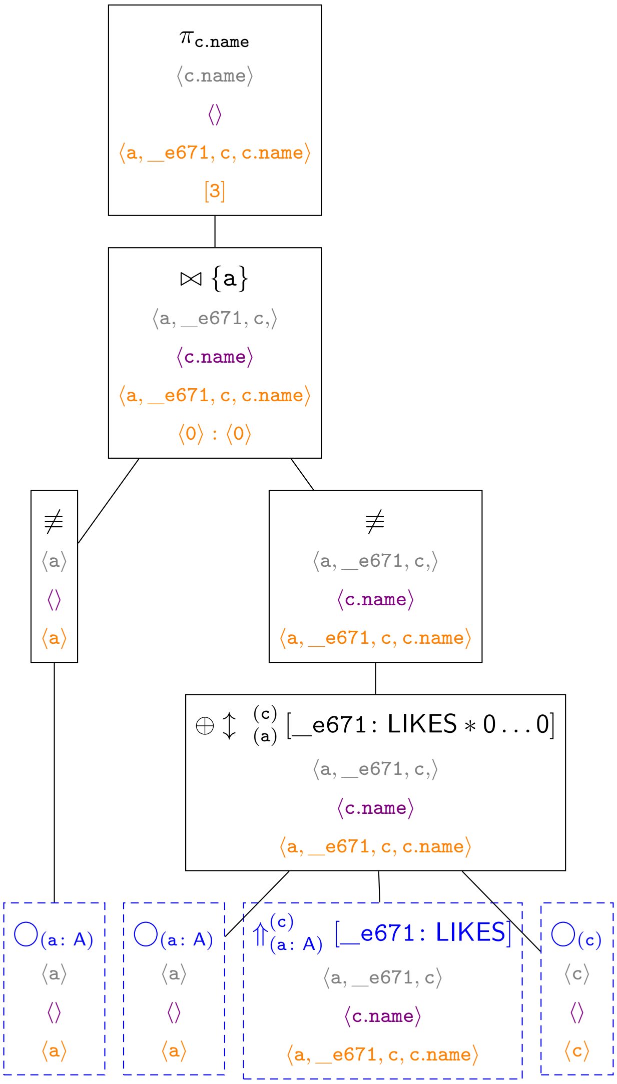
Relational algebra expression for search-based evaluation (Handling symmetrically bounded variable length match, bounds are zero)

$$r = \pi_{\texttt{c.name}}\Big(\not\equiv \Big(\bigcirc_{(\texttt{a: A})} \Big) \bowtie\!\not\equiv \Big(\uparrow {}^{(\texttt{c})}_{(\texttt{a})} [\_\texttt{e671: LIKES} * 0 \ldots 0] \Big(\bigcirc_{(\texttt{a: A})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Handling symmetrically bounded variable length match, bounds are zero)



Incremental relational algebra tree (Handling symmetrically bounded variable length match, bounds are zero)



A.44.9 Handling symmetrically bounded variable length match, bounds are one

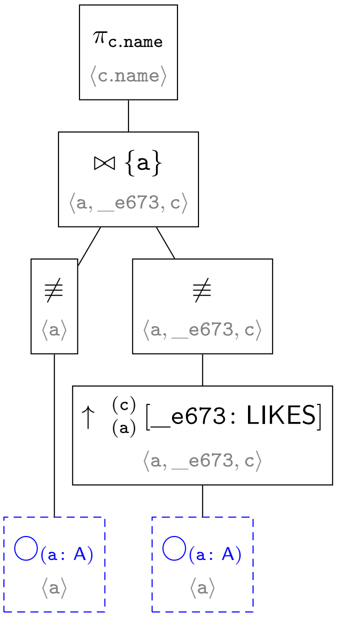
Query specification (Handling symmetrically bounded variable length match, bounds are one)

```
MATCH (a:A)
MATCH (a)-[:LIKES*1..1]->(c)
RETURN c.name
```

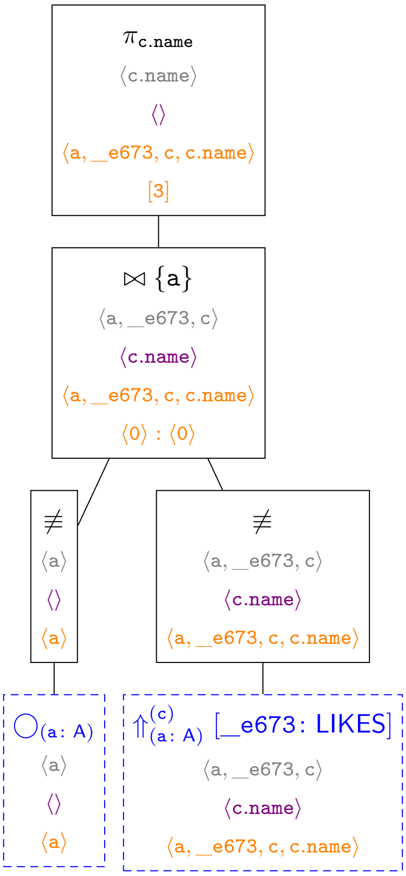
Relational algebra expression for search-based evaluation (Handling symmetrically bounded variable length match, bounds are one)

$$r = \pi_{\texttt{c.name}}\Big(\not\equiv \Big(\bigcirc_{(\mathsf{a}:\,\mathsf{A})} \Big) \bowtie_{\not\equiv} \Big(\uparrow {}^{(\mathsf{c})}_{(\mathsf{a})} [\_\mathsf{e673}\colon \mathsf{LIKES}] \Big(\bigcirc_{(\mathsf{a}:\,\mathsf{A})} \Big) \Big) \Big)$$

Relational algebra tree for search-based evaluation (Handling symmetrically bounded variable length match, bounds are one)



Incremental relational algebra tree (Handling symmetrically bounded variable length match, bounds are one)



A.44.10 Handling symmetrically bounded variable length match, bounds are two

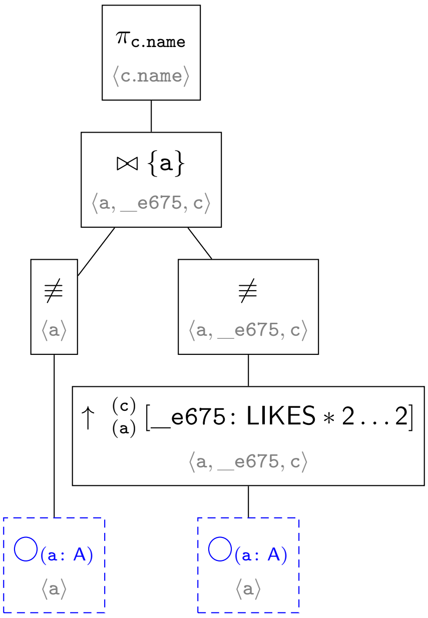
Query specification (Handling symmetrically bounded variable length match, bounds are two)

```
MATCH (a:A)
MATCH (a)-[:LIKES*2..2]->(c)
RETURN c.name
```

Relational algebra expression for search-based evaluation (Handling symmetrically bounded variable length match, bounds are two)

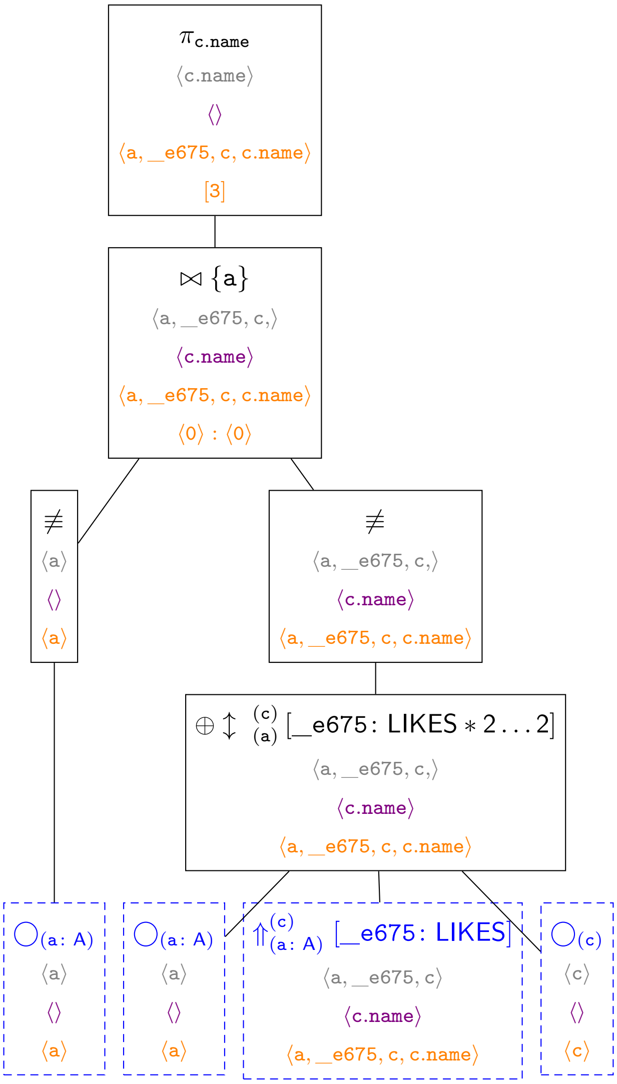
$$r = \pi_{\mathtt{c.name}}\Big(\not\equiv\Big(\bigcirc_{(\mathsf{a:\ A})}\Big) \bowtie\!\not\equiv \Big(\uparrow^{(\mathsf{c})}_{(\mathsf{a})}[\mathsf{\_e675}\colon \mathsf{LIKES} * 2\ldots 2]\Big(\bigcirc_{(\mathsf{a:\ A})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Handling symmetrically bounded variable length match, bounds are two)



$\pi_{\mathtt{c.name}}$
⟨c.name⟩
$\bowtie \{\mathsf{a}\}$
⟨a, _e675, c⟩
$\not\equiv$
⟨a⟩
$\not\equiv$
⟨a, _e675, c⟩
$\uparrow^{(\mathsf{c})}_{(\mathsf{a})}[\mathsf{\_e675}\colon \mathsf{LIKES} * 2\ldots 2]$
⟨a, _e675, c⟩
$\bigcirc_{(\mathsf{a:\ A})}$
⟨a⟩
$\bigcirc_{(\mathsf{a:\ A})}$
⟨a⟩

Incremental relational algebra tree (Handling symmetrically bounded variable length match, bounds are two)



A.44.11 Handling upper and lower bounded variable length match, empty interval 1

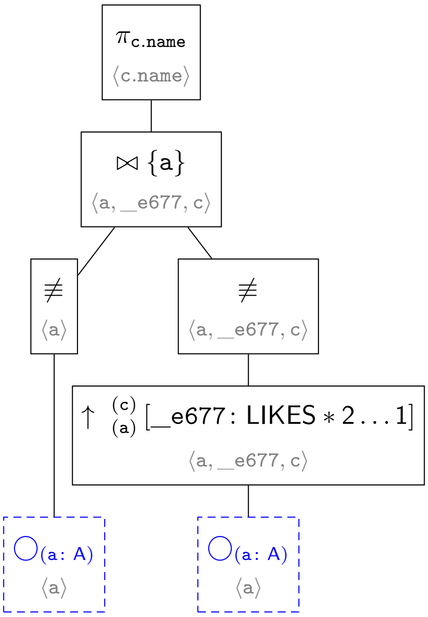
Query specification (Handling upper and lower bounded variable length match, empty interval 1)

```
MATCH (a:A)
MATCH (a)-[:LIKES*2..1]->(c)
RETURN c.name
```

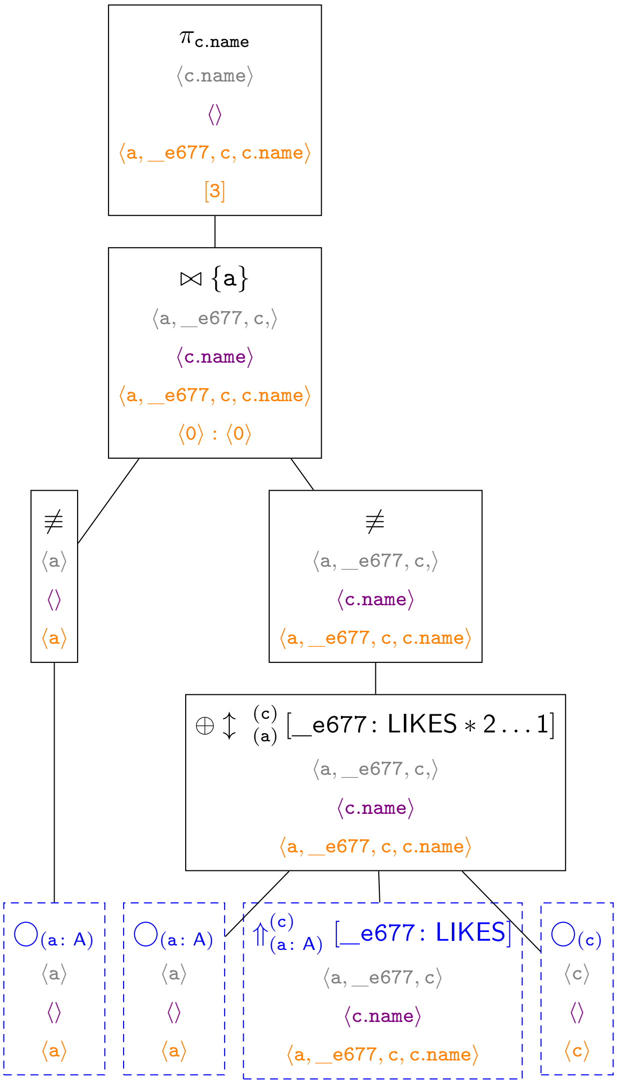
Relational algebra expression for search-based evaluation (Handling upper and lower bounded variable length match, empty interval 1)

$$r = \pi_{\texttt{c.name}}\Big(\not\equiv \Big(\bigcirc_{(\texttt{a: A})} \Big) \bowtie\!\not\equiv \Big(\uparrow {}^{(\texttt{c})}_{(\texttt{a})} [\_\text{e677}\colon \text{LIKES} * 2 \ldots 1] \Big(\bigcirc_{(\texttt{a: A})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation **(Handling upper and lower bounded variable length match, empty interval 1)**



Incremental relational algebra tree (Handling upper and lower bounded variable length match, empty interval 1)



A.44.12 Handling upper and lower bounded variable length match, empty interval 2

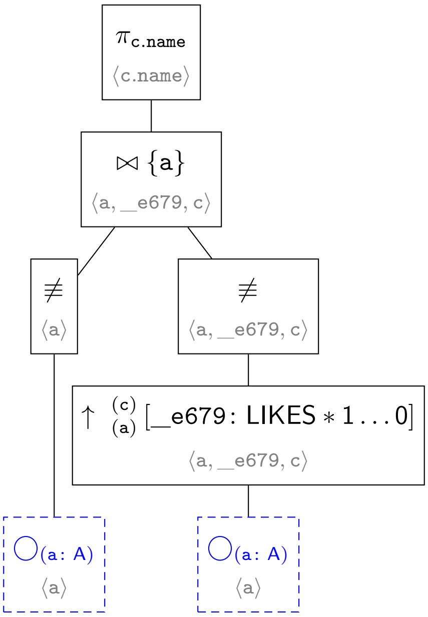
Query specification (Handling upper and lower bounded variable length match, empty interval 2)

```
MATCH (a:A)
MATCH (a)-[:LIKES*1..0]->(c)
RETURN c.name
```

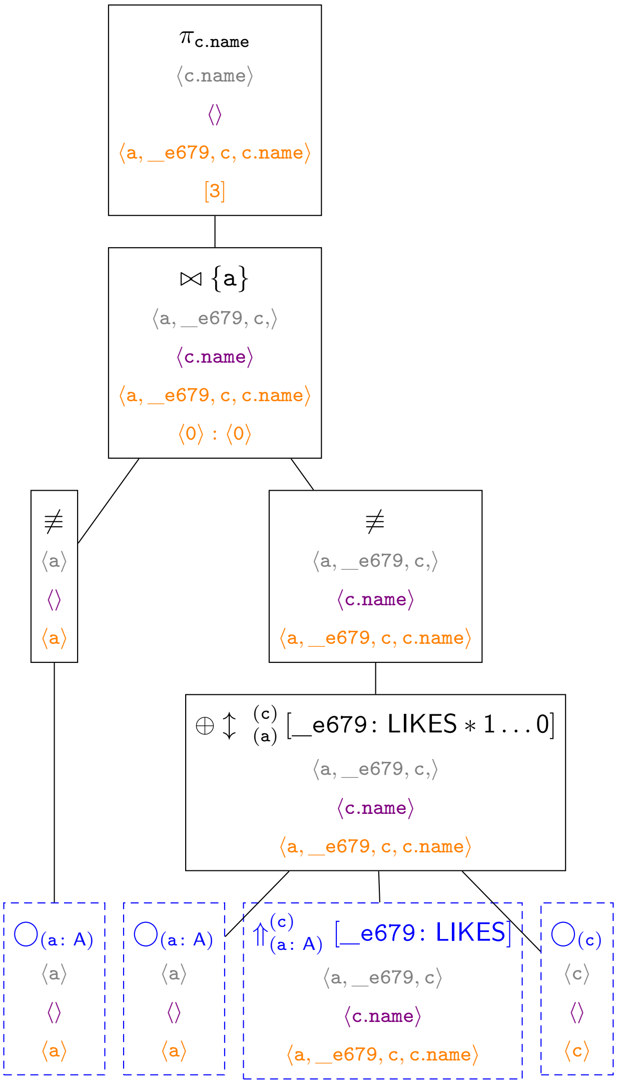
Relational algebra expression for search-based evaluation (Handling upper and lower bounded variable length match, empty interval 2)

$$r = \pi_{\texttt{c.name}}\Big(\not\equiv \Big(\bigcirc_{(\texttt{a: A})} \Big) \bowtie_{\not\equiv} \Big(\uparrow^{(\texttt{c})}_{(\texttt{a})} [\_\text{e679}\colon \text{LIKES} * 1 \ldots 0] \Big(\bigcirc_{(\texttt{a: A})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation **(Handling upper and lower bounded variable length match, empty interval 2)**



Incremental relational algebra tree (Handling upper and lower bounded variable length match, empty interval 2)



A.44.13 Handling upper bounded variable length match, empty interval

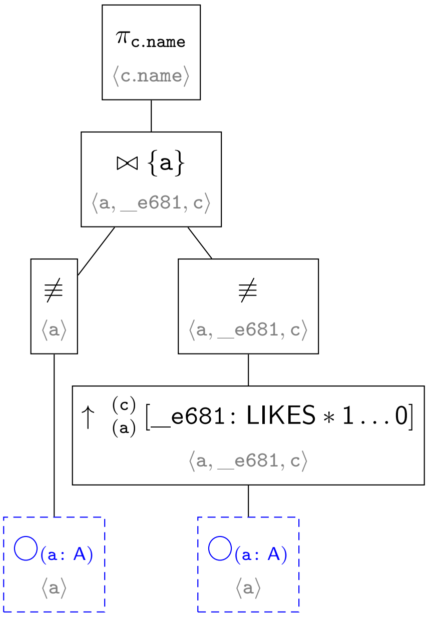
Query specification (Handling upper bounded variable length match, empty interval)

```
MATCH (a:A)
MATCH (a)-[:LIKES*..0]->(c)
RETURN c.name
```

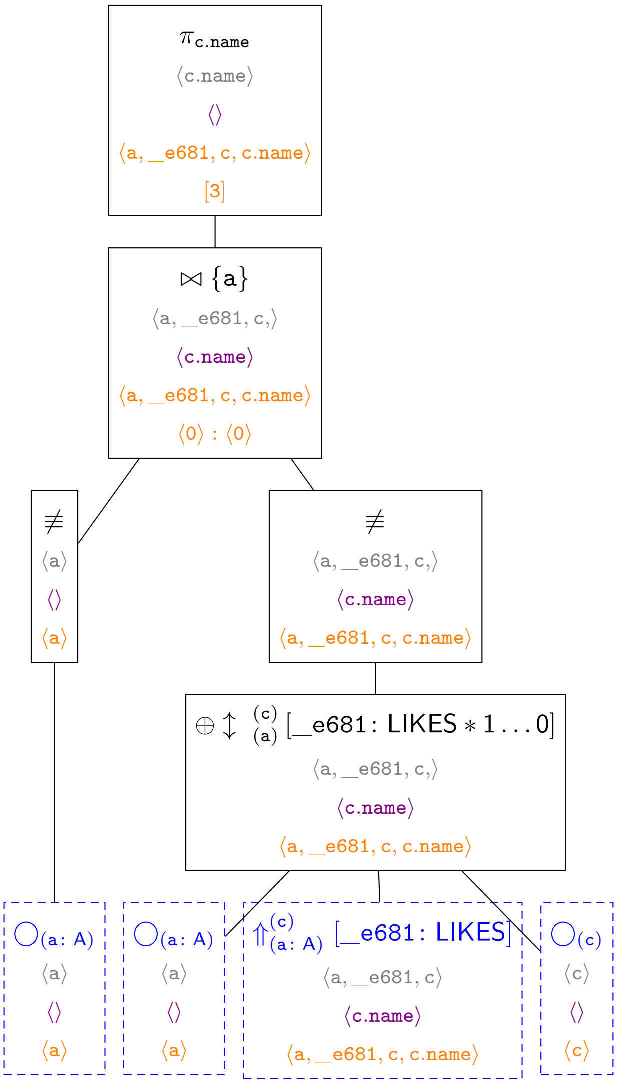
Relational algebra expression for search-based evaluation (Handling upper bounded variable length match, empty interval)

$$r = \pi_{\texttt{c.name}}\Big(\not\equiv \Big(\bigcirc_{(\texttt{a: A})} \Big) \bowtie \not\equiv \Big(\uparrow \, {}^{(\texttt{c})}_{(\texttt{a})} \, [\_\text{e681}\colon \text{LIKES} \ast 1 \ldots 0] \Big(\bigcirc_{(\texttt{a: A})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation **(Handling upper bounded variable length match, empty interval)**



Incremental relational algebra tree (Handling upper bounded variable length match, empty interval)



A.44.14 Handling upper bounded variable length match 1

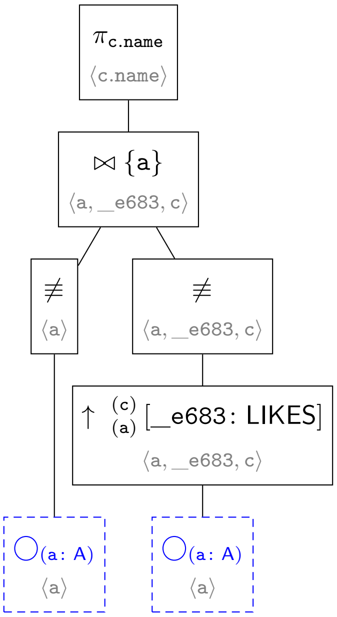
Query specification (Handling upper bounded variable length match 1)

```
MATCH (a:A)
MATCH (a)-[:LIKES*..1]->(c)
RETURN c.name
```

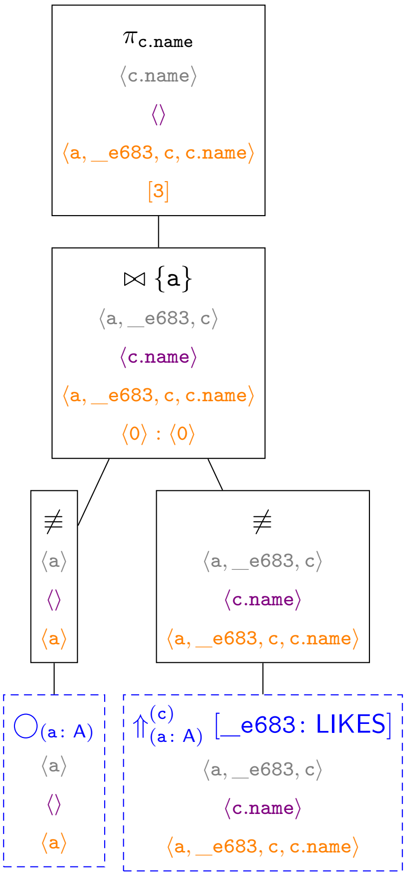
Relational algebra expression for search-based evaluation (Handling upper bounded variable length match 1)

$$r = \pi_{\texttt{c.name}}\Big(\not\equiv \Big(\bigcirc_{(\texttt{a}:\ \texttt{A})} \Big) \bowtie \not\equiv \Big(\uparrow {}^{(\texttt{c})}_{(\texttt{a})} [\_\text{e683}\colon \text{LIKES}] \Big(\bigcirc_{(\texttt{a}:\ \texttt{A})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Handling upper bounded variable length match 1)



Incremental relational algebra tree (Handling upper bounded variable length match 1)



A.44.15 Handling upper bounded variable length match 2

Query specification (Handling upper bounded variable length match 2)

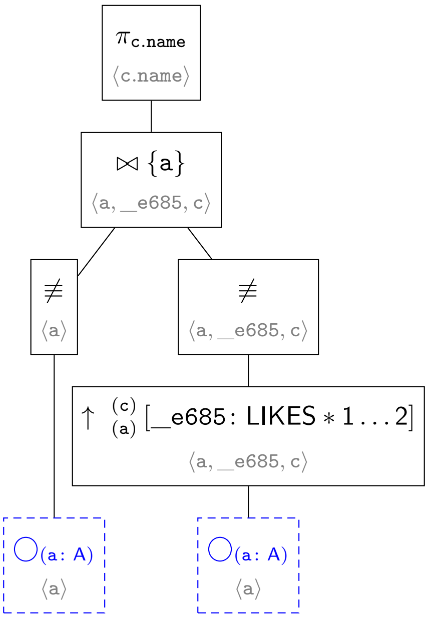
```
MATCH (a:A)
MATCH (a)-[:LIKES*..2]->(c)
```

```
RETURN c.name
```

Relational algebra expression for search-based evaluation **(Handling upper bounded variable length match 2)**

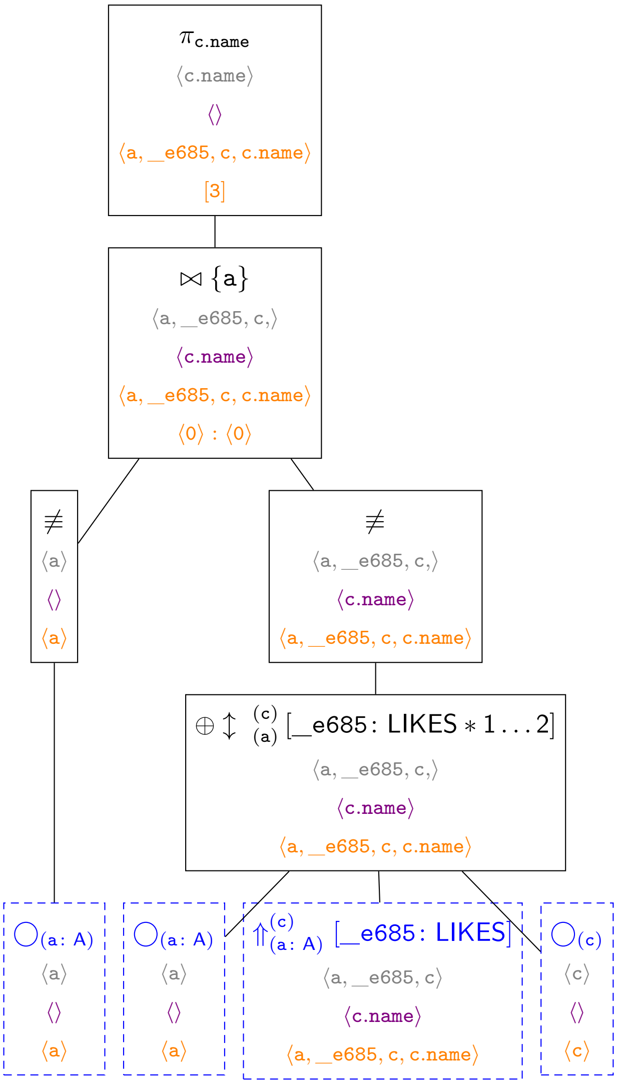
$$r = \pi_{\texttt{c.name}}\left(\not\equiv \left(\bigcirc_{(\texttt{a: A})} \right) \bowtie_{\not\equiv} \left(\uparrow^{(\texttt{c})}_{(\texttt{a})} [\_\text{e685}\colon \text{LIKES} * 1 \ldots 2] \left(\bigcirc_{(\texttt{a: A})} \right) \right) \right)$$

Relational algebra tree for search-based evaluation **(Handling upper bounded variable length match 2)**



$\pi_{\texttt{c.name}}$
⟨c.name⟩
$\bowtie \{\texttt{a}\}$
⟨a, _e685, c⟩
$\not\equiv$
⟨a⟩
$\not\equiv$
⟨a, _e685, c⟩
$\uparrow^{(\texttt{c})}_{(\texttt{a})} [\_\text{e685}\colon \text{LIKES} * 1 \ldots 2]$
⟨a, _e685, c⟩
$\bigcirc_{(\texttt{a: A})}$
⟨a⟩
$\bigcirc_{(\texttt{a: A})}$
⟨a⟩

Incremental relational algebra tree (Handling upper bounded variable length match 2)



A.44.16 Handling lower bounded variable length match 1

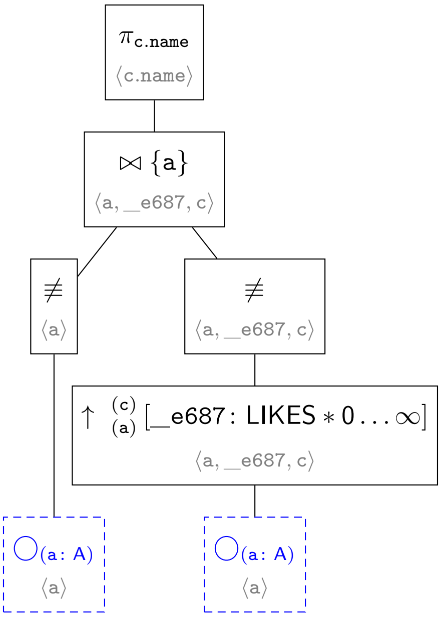
Query specification (Handling lower bounded variable length match 1)

```
MATCH (a:A)
MATCH (a)-[:LIKES*0..]->(c)
RETURN c.name
```

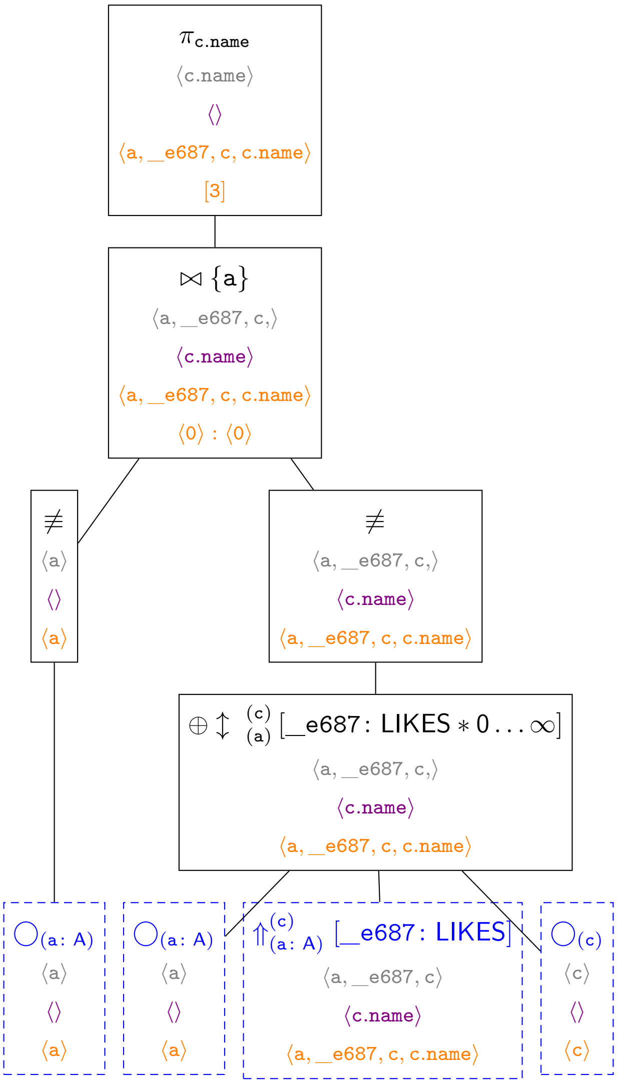
Relational algebra expression for search-based evaluation (Handling lower bounded variable length match 1)

$$r = \pi_{\mathtt{c.name}}\Big(\not\equiv \big(\bigcirc_{(\mathtt{a}:\,\mathsf{A})} \big) \bowtie\!\not\equiv \big(\uparrow {}^{(\mathtt{c})}_{(\mathtt{a})} [\_\mathsf{e687}\colon \mathsf{LIKES} * 0 \ldots \infty] \big(\bigcirc_{(\mathtt{a}:\,\mathsf{A})} \big) \big) \Big)$$

Relational algebra tree for search-based evaluation (Handling lower bounded variable length match 1)



Incremental relational algebra tree (Handling lower bounded variable length match 1)



A.44.17 Handling lower bounded variable length match 2

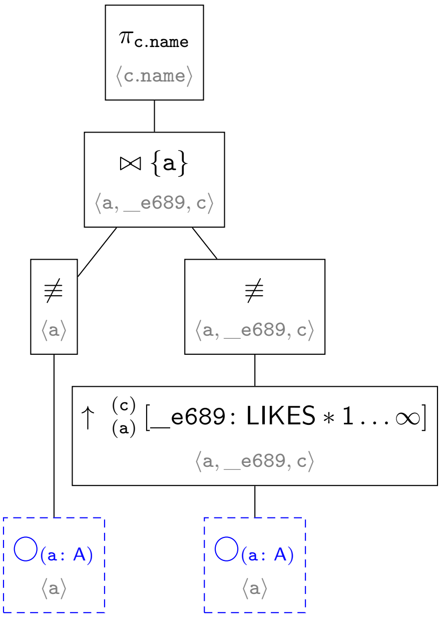
Query specification (Handling lower bounded variable length match 2)

```
MATCH (a:A)
MATCH (a)-[:LIKES*1..]->(c)
RETURN c.name
```

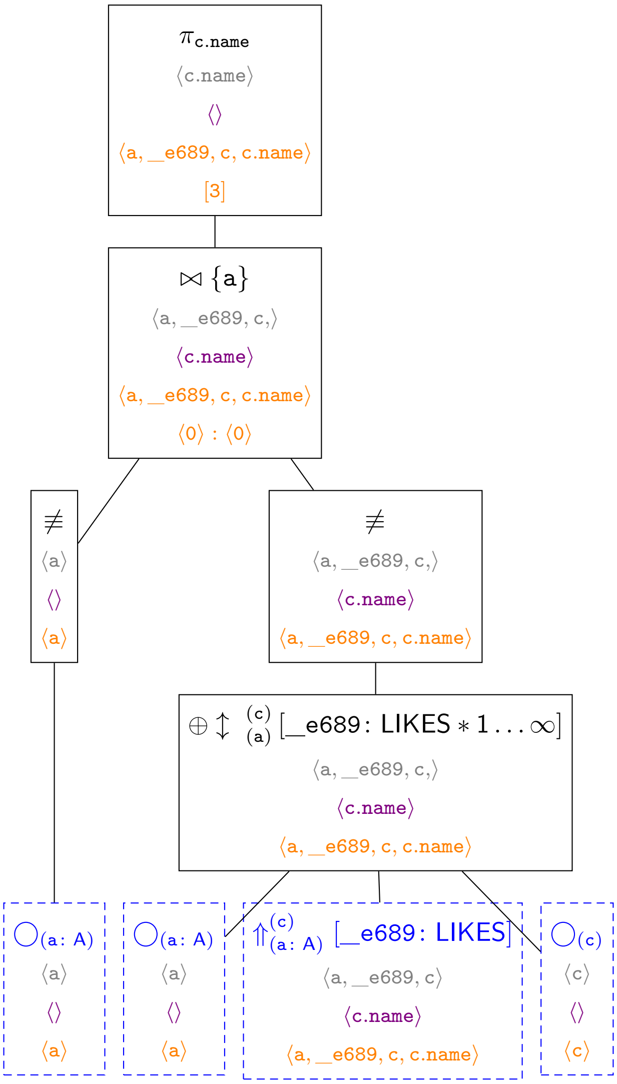
Relational algebra expression for search-based evaluation (Handling lower bounded variable length match 2)

$$r = \pi_{\texttt{c.name}}\left(\not\equiv\left(\bigcirc_{(\mathsf{a}\colon\,\mathsf{A})}\right) \bowtie \not\equiv\left(\uparrow\,{}^{(\mathsf{c})}_{(\mathsf{a})}[\_\mathsf{e689}\colon \mathsf{LIKES} * 1 \ldots \infty]\left(\bigcirc_{(\mathsf{a}\colon\,\mathsf{A})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (Handling lower bounded variable length match 2)


$\pi_{\texttt{c.name}}$
⟨c.name⟩
⋈ {a}
⟨a, _e689, c⟩
≢
⟨a⟩
≢
⟨a, _e689, c⟩
$\uparrow\,{}^{(\mathsf{c})}_{(\mathsf{a})}[\_\mathsf{e689}\colon \mathsf{LIKES} * 1 \ldots \infty]$
⟨a, _e689, c⟩
$\bigcirc_{(\mathsf{a}\colon\,\mathsf{A})}$
⟨a⟩
$\bigcirc_{(\mathsf{a}\colon\,\mathsf{A})}$
⟨a⟩

Incremental relational algebra tree (Handling lower bounded variable length match 2)



A.44.18 Handling lower bounded variable length match 3

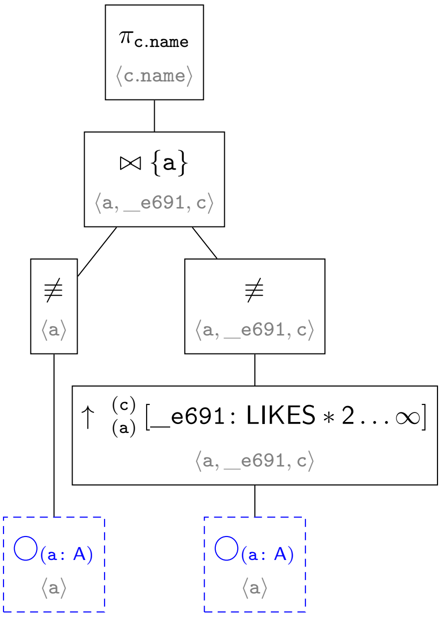
Query specification (Handling lower bounded variable length match 3)

```
MATCH (a:A)
MATCH (a)-[:LIKES*2..]->(c)
RETURN c.name
```

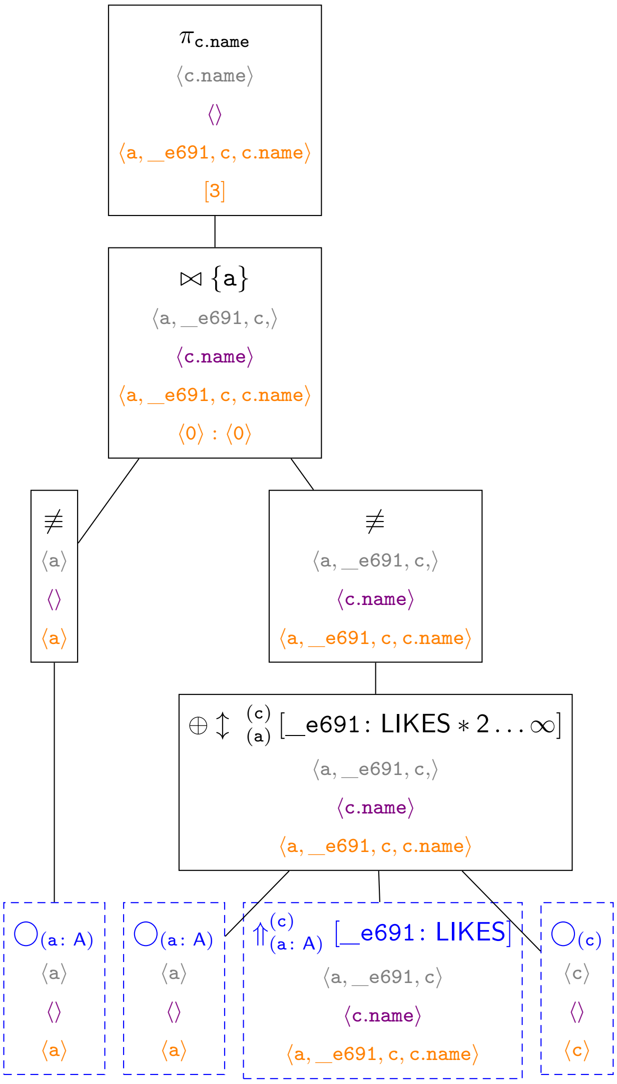
Relational algebra expression for search-based evaluation (Handling lower bounded variable length match 3)

$$r = \pi_{\texttt{c.name}}\Big(\not\equiv \Big(\bigcirc_{(\texttt{a: A})} \Big) \bowtie\not\equiv \Big(\uparrow {}^{(\texttt{c})}_{(\texttt{a})} [\_\text{e691}\colon \text{LIKES} * 2 \ldots \infty] \Big(\bigcirc_{(\texttt{a: A})} \Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Handling lower bounded variable length match 3)



Incremental relational algebra tree (Handling lower bounded variable length match 3)



A.44.19 Handling a variable length relationship and a standard relationship in chain, zero length 1

Query specification **(Handling a variable length relationship and a standard relationship in chain, zero length 1)**

```
MATCH (a:A)
MATCH (a)-[:LIKES*0]->()-[:LIKES]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44.20 Handling a variable length relationship and a standard relationship in chain, zero length 2

Query specification **(Handling a variable length relationship and a standard relationship in chain, zero length 2)**

```
MATCH (a:A)
MATCH (a)-[:LIKES]->()-[:LIKES*0]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44.21 Handling a variable length relationship and a standard relationship in chain, single length 1

Query specification **(Handling a variable length relationship and a standard relationship in chain, single length 1)**

```
MATCH (a:A)
MATCH (a)-[:LIKES*1]->()-[:LIKES]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44.22 Handling a variable length relationship and a standard relationship in chain, single length 2

Query specification **(Handling a variable length relationship and a standard relationship in chain, single length 2)**

```
MATCH (a:A)
MATCH (a)-[:LIKES]->()-[:LIKES*1]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44.23 Handling a variable length relationship and a standard relationship in chain, longer 1

Query specification (Handling a variable length relationship and a standard relationship in chain, longer 1)

```
MATCH (a:A)
MATCH (a)-[:LIKES*2]->()-[:LIKES]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44.24 Handling a variable length relationship and a standard relationship in chain, longer 2

Query specification (Handling a variable length relationship and a standard relationship in chain, longer 2)

```
MATCH (a:A)
MATCH (a)-[:LIKES]->()-[:LIKES*2]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44.25 Handling a variable length relationship and a standard relationship in chain, longer 3

Query specification (Handling a variable length relationship and a standard relationship in chain, longer 3)

```
MATCH (a:A)
MATCH (a)-[:LIKES]->()-[:LIKES*3]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44.26 Handling mixed relationship patterns and directions 1

Query specification (Handling mixed relationship patterns and directions 1)

```
MATCH (a:A)
MATCH (a)<-[:LIKES]-()-[:LIKES*3]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44.27 Handling mixed relationship patterns and directions 2

Query specification **(Handling mixed relationship patterns and directions 2)**

```
MATCH (a:A)
MATCH (a)-[:LIKES]->()<-[:LIKES*3]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44.28 Handling mixed relationship patterns 1

Query specification **(Handling mixed relationship patterns 1)**

```
MATCH (a:A)
MATCH (p)-[:LIKES*1]->()-[:LIKES]->()-[r:LIKES*2]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.44.29 Handling mixed relationship patterns 2

Query specification **(Handling mixed relationship patterns 2)**

```
MATCH (a:A)
MATCH (p)-[:LIKES]->()-[:LIKES*2]->()-[r:LIKES]->(c)
RETURN c.name
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.45 VarLengthAcceptance2

Progress:

0 of 1

A.45.1 Handling relationships that are already bound in variable length paths

Query specification **(Handling relationships that are already bound in variable length paths)**

```
MATCH ()-[r:EDGE]-()
MATCH p = (n)-[*0..1]-()-[r]-()-[*0..1]-(m)
RETURN count(p) AS c
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.46 WhereAcceptance

Progress:

0 of 1

A.46.1 NOT and false

Query specification (NOT and false)

```
MATCH (n)
WHERE NOT(n.name = 'apa' AND false)
RETURN n
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.47 WithAcceptance

Progress:

5 of 17

A.47.1 Passing on pattern nodes

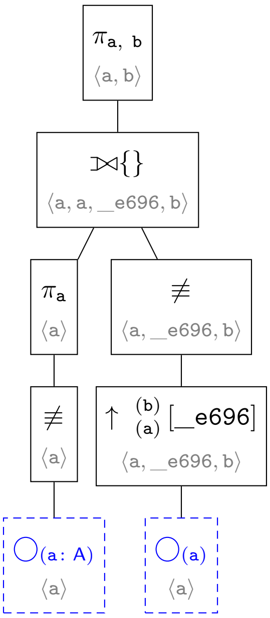
Query specification (Passing on pattern nodes)

```
MATCH (a:A)
WITH a
MATCH (a)-->(b)
RETURN *
```

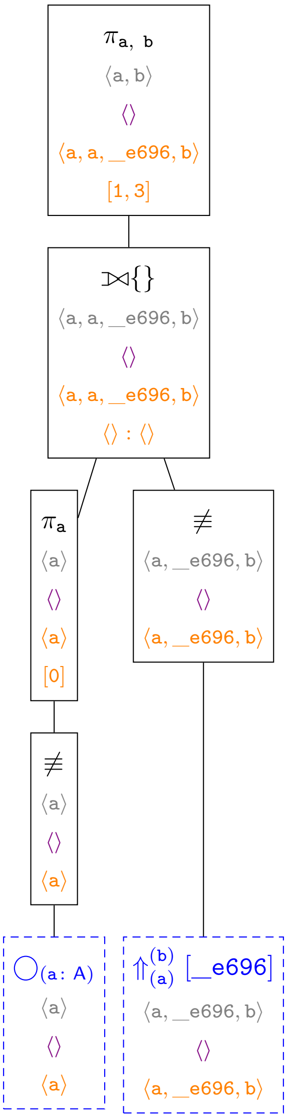
Relational algebra expression for search-based evaluation (Passing on pattern nodes)

$$r = \pi_{\mathsf{a,\ b}}\Big(\pi_{\mathsf{a}}\Big(\not\equiv\Big(\bigcirc_{\mathsf{(a:\ A)}}\Big)\Big)\text{⟕}\not\equiv\Big(\uparrow^{\mathsf{(b)}}_{\mathsf{(a)}}[\_\mathsf{e696}]\Big(\bigcirc_{\mathsf{(a)}}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Passing on pattern nodes)



Incremental relational algebra tree (Passing on pattern nodes)



A.47.2 ORDER BY and LIMIT can be used

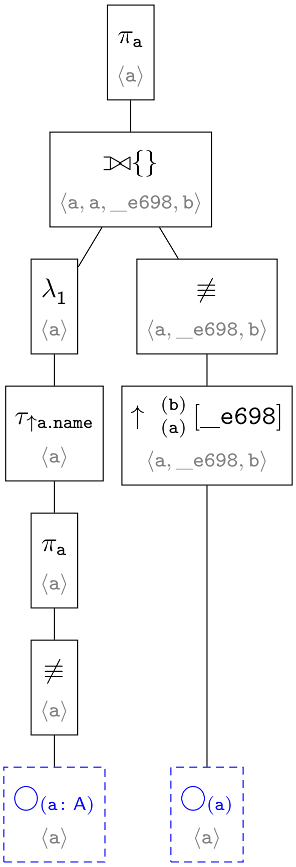
Query specification (ORDER BY and LIMIT can be used)

```
MATCH (a:A)
WITH a
ORDER BY a.name
LIMIT 1
MATCH (a)-->(b)
RETURN a
```

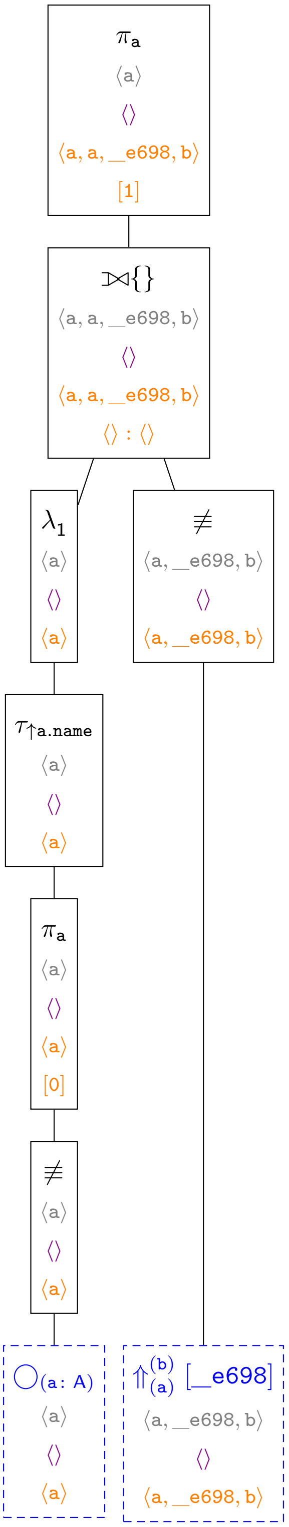
Relational algebra expression for search-based evaluation (ORDER BY and LIMIT can be used)

$$r = \pi_{\mathsf{a}}\Big(\lambda_1\Big(\tau_{\uparrow\texttt{a.name}}\Big(\pi_{\mathsf{a}}\Big(\not\equiv\Big(\bigcirc_{(\mathsf{a}:\ \mathsf{A})}\Big)\Big)\Big)\Big)\bowtie\not\equiv\Big(\uparrow\ ^{(\mathsf{b})}_{(\mathsf{a})}\,[\_\mathsf{e698}]\Big(\bigcirc_{(\mathsf{a})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (ORDER BY and LIMIT can be used)



Incremental relational algebra tree (ORDER BY and LIMIT can be used)



A.47.3 No dependencies between the query parts

Query specification (No dependencies between the query parts)

```
MATCH (a)
WITH a
MATCH (b)
RETURN a, b
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.47.4 Aliasing

Query specification (Aliasing)

```
MATCH (a:Begin)
WITH a.prop AS property
MATCH (b:End)
WHERE property = b.prop
RETURN b
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.47.5 Handle dependencies across WITH

Query specification (Handle dependencies across WITH)

```
MATCH (a:Begin)
WITH a.prop AS property
LIMIT 1
MATCH (b)
WHERE b.id = property
RETURN b
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.47.6 Handle dependencies across WITH with SKIP

Query specification (Handle dependencies across WITH with SKIP)

```
MATCH (a)
WITH a.prop AS property, a.key AS idToUse
ORDER BY property
SKIP 1
MATCH (b)
WHERE b.id = idToUse
RETURN DISTINCT b
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.47.7 WHERE after WITH should filter results

Query specification (WHERE after WITH should filter results)

```
MATCH (a)
WITH a
WHERE a.name = 'B'
RETURN a
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.47.8 WHERE after WITH can filter on top of an aggregation

Query specification (WHERE after WITH can filter on top of an aggregation)

```
MATCH (a)-->()
WITH a, count(*) AS relCount
WHERE relCount > 1
RETURN a
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.47.9 ORDER BY on an aggregating key

Query specification (ORDER BY on an aggregating key)

```
MATCH (a)
WITH a.bar AS bars, count(*) AS relCount
ORDER BY a.bar
RETURN *
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.47.10 ORDER BY a DISTINCT column

Query specification (ORDER BY a DISTINCT column)

```
MATCH (a)
WITH DISTINCT a.bar AS bars
ORDER BY a.bar
RETURN *
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.47.11 WHERE on a DISTINCT column

Query specification (WHERE on a DISTINCT column)

```
MATCH (a)
WITH DISTINCT a.bar AS bars
WHERE a.bar = 'B'
RETURN *
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.47.12 A simple pattern with one bound endpoint

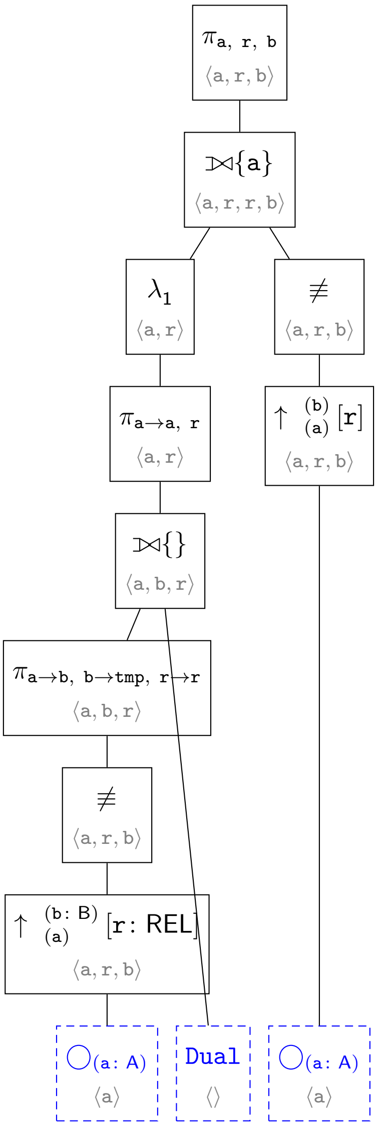
Query specification (A simple pattern with one bound endpoint)

```
MATCH (a:A)-[r:REL]->(b:B)
WITH a AS b, b AS tmp, r AS r
WITH b AS a, r
LIMIT 1
MATCH (a)-[r]->(b)
RETURN a, r, b
```

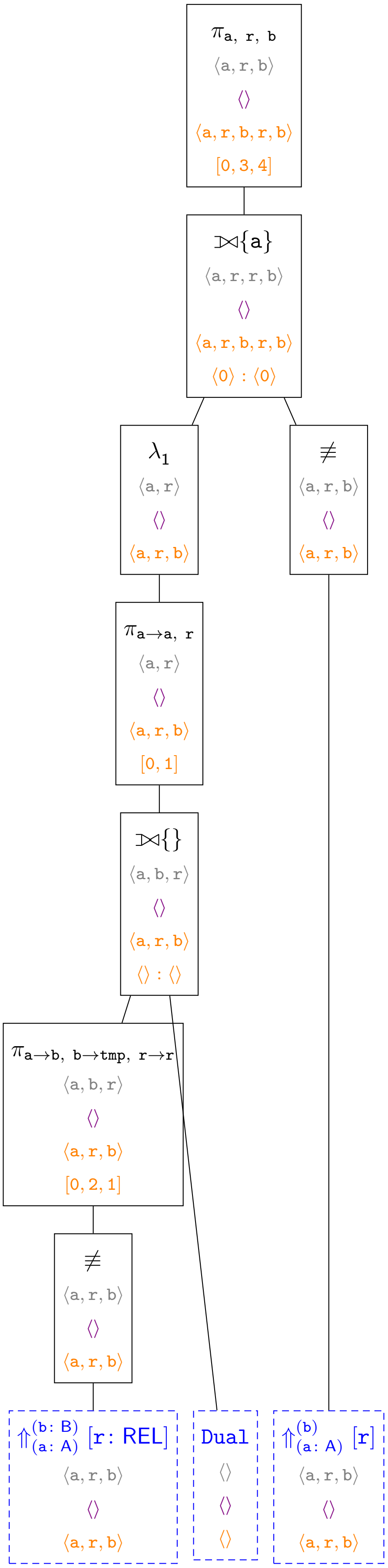
Relational algebra expression for search-based evaluation (A simple pattern with one bound endpoint)

$$r = \pi_{\texttt{a, r, b}}\Big(\lambda_1\Big(\pi_{\texttt{a}\to\texttt{a, r}}\Big(\pi_{\texttt{a}\to\texttt{b, b}\to\texttt{tmp, r}\to\texttt{r}}\Big(\not\equiv\Big(\uparrow\,^{(\mathsf{b:\ B})}_{(\mathsf{a})}\,[\mathsf{r:\ REL}]\Big(\bigcirc_{(\mathsf{a:\ A})}\Big)\Big)\Big)\mathbin{⟕}\mathsf{Dual}\Big)\Big)\mathbin{⟕}\not\equiv\Big(\uparrow\,^{(\mathsf{b})}_{(\mathsf{a})}\,[\mathsf{r}]$$
$$\Big(\bigcirc_{(\mathsf{a:\ A})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (A simple pattern with one bound endpoint)



Incremental relational algebra tree (A simple pattern with one bound endpoint)



A.47.13 Null handling

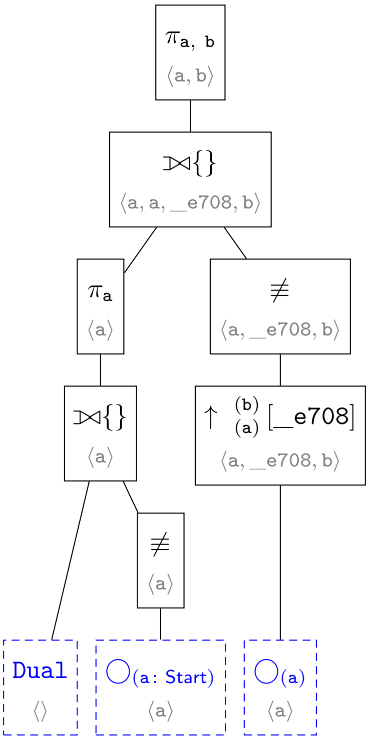
Query specification (Null handling)

```
OPTIONAL MATCH (a:Start)
WITH a
MATCH (a)-->(b)
RETURN *
```

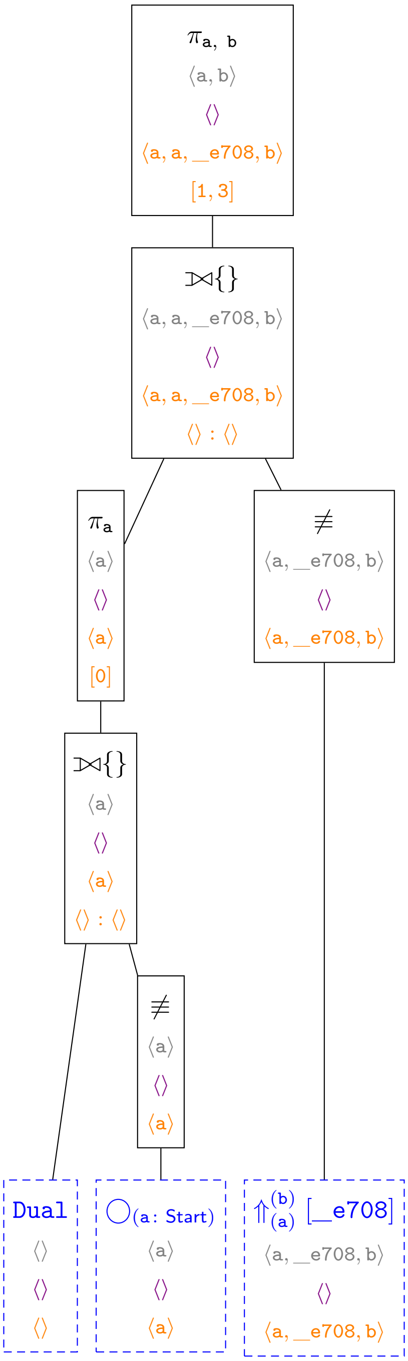
Relational algebra expression for search-based evaluation (Null handling)

$$r = \pi_{\texttt{a, b}}\Big(\pi_{\texttt{a}}\Big(\texttt{Dual} ⟕ \not\equiv \Big(\bigcirc_{(\texttt{a}:\,\mathsf{Start})}\Big)\Big) ⟕ \not\equiv \Big(\uparrow\, {}^{(\texttt{b})}_{(\texttt{a})}\, [\_\mathsf{e708}]\, \Big(\bigcirc_{(\texttt{a})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Null handling)



Incremental relational algebra tree (Null handling)



A.47.14 Nested maps

Query specification (Nested maps)

```
WITH {foo: {bar: 'baz'}} AS nestedMap
RETURN nestedMap.foo.bar
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.47.15 Connected components succeeding WITH

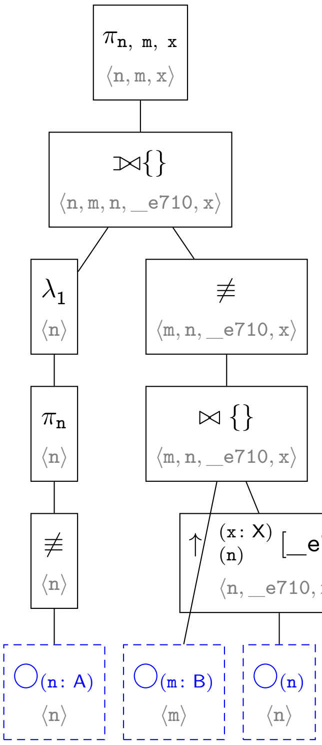
Query specification (Connected components succeeding WITH)

```
MATCH (n:A)
WITH n
LIMIT 1
MATCH (m:B), (n)-->(x:X)
RETURN *
```

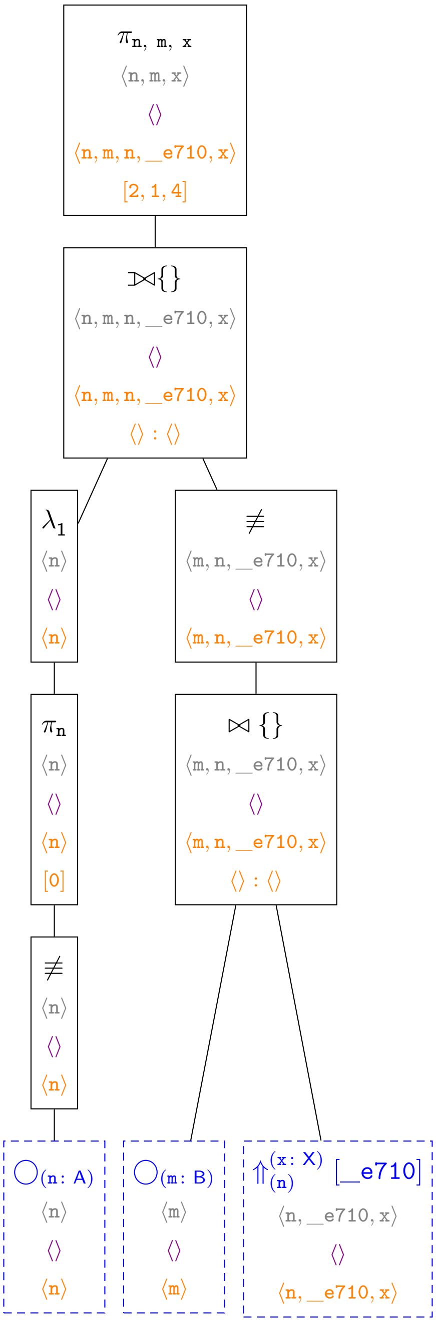
Relational algebra expression for search-based evaluation (Connected components succeeding WITH)

$$r = \pi_{\texttt{n, m, x}}\Big(\lambda_1\Big(\pi_{\texttt{n}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{n}:\ \mathsf{A})}\Big)\Big)\Big) \mathbin{⟕} \not\equiv\Big(\bigcirc_{(\texttt{m}:\ \mathsf{B})} \bowtie \uparrow\,{}^{(\texttt{x}:\ \mathsf{X})}_{(\texttt{n})}\,[\_\mathsf{e710}]\Big(\bigcirc_{(\texttt{n})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (Connected components succeeding WITH)


$\pi_{\texttt{n, m, x}}$
⟨n, m, x⟩
⟕{}
⟨n, m, n, _e710, x⟩
λ_1
⟨n⟩
≢
⟨m, n, _e710, x⟩
$\pi_{\texttt{n}}$
⟨n⟩
⋈ {}
⟨m, n, _e710, x⟩
≢
⟨n⟩
$\uparrow{}^{(\texttt{x}:\ \mathsf{X})}_{(\texttt{n})}[\_\mathsf{e710}]$
⟨n, _e710, x⟩
$\bigcirc_{(\texttt{n}:\ \mathsf{A})}$
⟨n⟩
$\bigcirc_{(\texttt{m}:\ \mathsf{B})}$
⟨m⟩
$\bigcirc_{(\texttt{n})}$
⟨n⟩

Incremental relational algebra tree (Connected components succeeding WITH)



A.47.16 Single WITH using a predicate and aggregation

Query specification (Single WITH using a predicate and aggregation)

```
MATCH (n)
WITH n
WHERE n.prop = 42
RETURN count(*)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

A.47.17 Multiple WITHs using a predicate and aggregation

Query specification **(Multiple WITHs using a predicate and aggregation)**

```
MATCH (david {name: 'David'})--(otherPerson)-->()
WITH otherPerson, count(*) AS foaf
WHERE foaf > 1
WITH otherPerson
WHERE otherPerson.name <> 'NotOther'
RETURN count(*)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

Appendix B

Fraud Detection

Total progress:

0 of 3

B.1 Queries

Progress:

0 of 3

B.1.1 create

Query specification (create)

```
// Create account holders
CREATE (accountHolder1:AccountHolder {
  FirstName: "John",
  LastName: "Doe",
  UniqueId: "JohnDoe" })

CREATE (accountHolder2:AccountHolder {
  FirstName: "Jane",
  LastName: "Appleseed",
  UniqueId: "JaneAppleseed" })

CREATE (accountHolder3:AccountHolder {
  FirstName: "Matt",
  LastName: "Smith",
  UniqueId: "MattSmith" })

// Create Address
CREATE (address1:Address {
  Street: "123 NW 1st Street",
  City: "San Francisco",
  State: "California",
  ZipCode: "94101" })

// Connect 3 account holders to 1 address
CREATE
  (accountHolder1)-[:HAS_ADDRESS]->(address1),
  (accountHolder2)-[:HAS_ADDRESS]->(address1),
  (accountHolder3)-[:HAS_ADDRESS]->(address1)

// Create Phone Number
CREATE (phoneNumber1:PhoneNumber { PhoneNumber: "555-555-5555" })

// Connect 2 account holders to 1 phone number
CREATE
```

```
  (accountHolder1)-[:HAS_PHONENUMBER]->(phoneNumber1),
  (accountHolder2)-[:HAS_PHONENUMBER]->(phoneNumber1)

// Create SSN
CREATE (ssn1:SSN { SSN: "241-23-1234" })

// Connect 2 account holders to 1 SSN
CREATE
  (accountHolder2)-[:HAS_SSN]->(ssn1),
  (accountHolder3)-[:HAS_SSN]->(ssn1)

// Create SSN and connect 1 account holder
CREATE (ssn2:SSN { SSN: "241-23-4567" })<-[:HAS_SSN]-(accountHolder1)

// Create Credit Card and connect 1 account holder
CREATE (creditCard1:CreditCard {
  AccountNumber: "1234567890123456",
  Limit: 5000, Balance: 1442.23,
  ExpirationDate: "01-20",
  SecurityCode: "123" })<-[:HAS_CREDITCARD]-(accountHolder1)

// Create Bank Account and connect 1 account holder
CREATE (bankAccount1:BankAccount {
  AccountNumber: "2345678901234567",
  Balance: 7054.43 })<-[:HAS_BANKACCOUNT]-(accountHolder1)

// Create Credit Card and connect 1 account holder
CREATE (creditCard2:CreditCard {
  AccountNumber: "1234567890123456",
  Limit: 4000, Balance: 2345.56,
  ExpirationDate: "02-20",
  SecurityCode: "456" })<-[:HAS_CREDITCARD]-(accountHolder2)

// Create Bank Account and connect 1 account holder
CREATE (bankAccount2:BankAccount {
  AccountNumber: "3456789012345678",
  Balance: 4231.12 })<-[:HAS_BANKACCOUNT]-(accountHolder2)

// Create Unsecured Loan and connect 1 account holder
CREATE (unsecuredLoan2:UnsecuredLoan {
  AccountNumber: "4567890123456789-0",
  Balance: 9045.53,
  APR: .0541,
  LoanAmount: 12000.00 })<-[:HAS_UNSECUREDLOAN]-(accountHolder2)

// Create Bank Account and connect 1 account holder
CREATE (bankAccount3:BankAccount {
  AccountNumber: "4567890123456789",
  Balance: 12345.45 })<-[:HAS_BANKACCOUNT]-(accountHolder3)

// Create Unsecured Loan and connect 1 account holder
CREATE (unsecuredLoan3:UnsecuredLoan {
  AccountNumber: "5678901234567890-0",
  Balance: 16341.95, APR: .0341,
  LoanAmount: 22000.00 })<-[:HAS_UNSECUREDLOAN]-(accountHolder3)

// Create Phone Number and connect 1 account holder
CREATE (phoneNumber2:PhoneNumber {
  PhoneNumber: "555-555-1234" })<-[:HAS_PHONENUMBER]-(accountHolder3)

RETURN *
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

B.1.2 financial-risk

Query specification (financial-risk)

```
MATCH (accountHolder:AccountHolder)-[]->(contactInformation)
WITH
  contactInformation,
  count(accountHolder) AS RingSize

MATCH
  (contactInformation)<-[]-(accountHolder),
  (accountHolder)-[r:HAS_CREDITCARD|HAS_UNSECUREDLOAN]->(unsecuredAccount)
WITH
  collect(DISTINCT accountHolder.UniqueId) AS AccountHolders,
  contactInformation, RingSize,
  SUM(
    CASE type(r)
      WHEN 'HAS_CREDITCARD' THEN unsecuredAccount.LIMIT
      WHEN 'HAS_UNSECUREDLOAN' THEN unsecuredAccount.Balance
    ELSE 0
  END) AS FinancialRisk
WHERE RingSize > 1

RETURN
  AccountHolders AS FraudRing,
  labels(contactInformation) AS ContactType,
  RingSize,
  round(FinancialRisk) AS FinancialRisk

ORDER BY FinancialRisk DESC
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

B.1.3 shared-contact-information

Query specification (shared-contact-information)

```
MATCH (accountHolder:AccountHolder)-[]->(contactInformation)
WITH
  contactInformation,
  count(accountHolder) AS RingSize

MATCH (contactInformation)<-[]-(accountHolder)
WITH
  collect(accountHolder.UniqueId) AS AccountHolders,
  contactInformation, RingSize
WHERE RingSize > 1

RETURN
  AccountHolders AS FraudRing,
  labels(contactInformation) AS ContactType,
  RingSize
ORDER BY
  RingSize DESC
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

Appendix C

LDBC Social Network Benchmark – Interactive Workload

Total progress:

2 of 9

C.1 Queries

Progress:

2 of 9

C.1.1 query-3

Query specification (query-3)

```
// Q3. Friends within 2 steps that recently traveled to countries X and Y. Find top 20 friends and friends of friends of a given Person who have made a post or a comment in the foreign CountryX and CountryY within a specified period of DurationInDays after a startDate. Sorted results descending by total number of posts.
MATCH
  (person:Person)-[:KNOWS*1..2]-(friend:Person)<-[:HAS_CREATOR]-(messageX),
  (messageX)-[:IS_LOCATED_IN]->(countryX:Country)
WHERE NOT(person = friend) // I think this condition is unnecessary as Cypher will not travel the same edge twice (szarnyasg)
  AND NOT((friend)-[:IS_LOCATED_IN]->()-[:IS_PART_OF]->(countryX))
  AND countryX.name = $countryXName
  AND messageX.creationDate >= $date1
  AND messageX.creationDate < $date2
WITH friend, count(DISTINCT messageX) AS xCount
MATCH (friend)<-[:HAS_CREATOR]-(messageY)-[:IS_LOCATED_IN]->(countryY:Country)
WHERE countryY.name = $countryYName
  AND NOT((friend)-[:IS_LOCATED_IN]->()-[:IS_PART_OF]->(countryY))
  AND messageY.creationDate >= $date1
  AND messageY.creationDate < $date2
WITH
  friend.id AS friendId,
  friend.firstName AS friendFirstName,
  friend.lastName AS friendLastName,
  xCount,
  count(DISTINCT messageY) AS yCount
RETURN
  friendId,
  friendFirstName,
  friendLastName,
  xCount,
  yCount,
  xCount + yCount AS xyCount
ORDER BY
  xyCount DESC,
  friendId ASC
LIMIT 10
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

C.1.2 query-4

Query specification (query-4)

```
// Q4. New Topics. Given a start Person, find the top 10 most popular Tags (by total number of posts with the tag) that are attached to Posts that were created by that Person's friends within a given time interval.
MATCH (person:Person)-[:KNOWS]-(:Person)<-[:HAS_CREATOR]-(post:Post)-[HAS_TAG]->(tag:Tag)
WHERE post.creationDate >= $date1
  AND post.creationDate < $date2
OPTIONAL MATCH (tag)<-[:HAS_TAG]-(oldPost:Post)
WHERE oldPost.creationDate < $date1
WITH tag, post, length(collect(oldPost)) AS oldPostCount
WHERE oldPostCount = 0
RETURN
  tag.name AS tagName,
  length(collect(post)) AS postCount
ORDER BY
  postCount DESC,
  tagName ASC
LIMIT 10
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

C.1.3 query-5

Query specification (query-5)

```
// Q5. New groups. Given a start Person, find the top 20 Forums the friends and friends of friends of that Person joined after a given Date. Sort results descending by the number of Posts in each Forum that were created by any of these Persons.
MATCH (person:Person)-[:KNOWS*1..2]-(friend:Person)<-[membership:HAS_MEMBER]-(forum:Forum)
WHERE membership.joinDate > $date
  AND NOT(person = friend) // I think this condition is unnecessary as Cypher will not travel the same edge twice (szarnyasg)
WITH DISTINCT friend, forum
OPTIONAL MATCH (friend)<-[:HAS_CREATOR]-(post:Post)<-[:CONTAINER_OF]-(forum)
WITH forum, count(post) AS postCount
RETURN
  forum.title AS forumName,
  postCount
ORDER BY
  postCount DESC,
  forum.id ASC
LIMIT 10
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

C.1.4 query-6

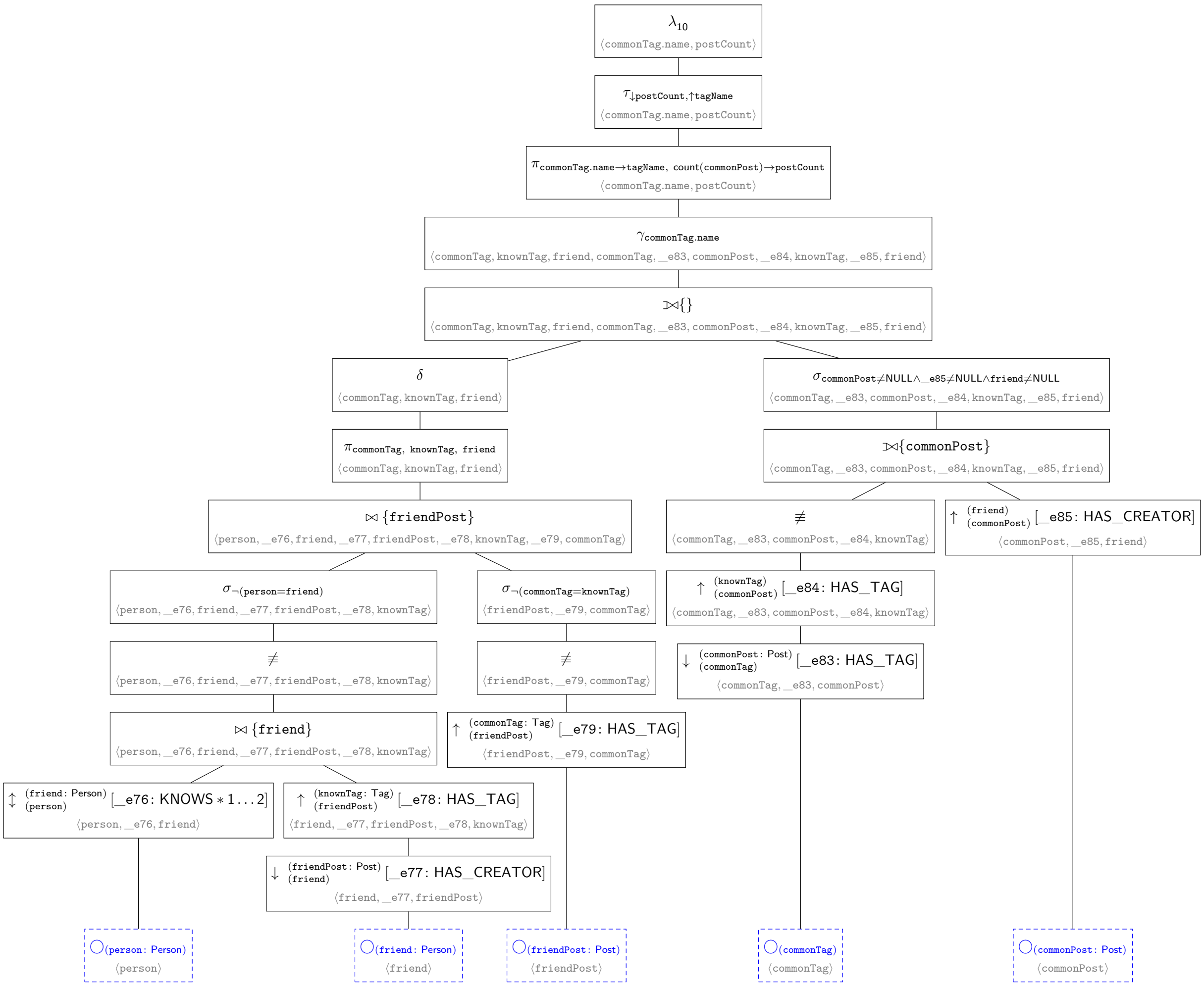
Query specification (query-6)

```
// Q6. Tag co-occurrence. Given a start Person and some Tag, find the other Tags that occur
    together with this Tag on Posts that were created by Person's friends and friends of friends
    . Return top 10 Tags, sorted descending by the count of Posts that were created by these
    Persons, which contain both this Tag and the given Tag.
MATCH
  (person:Person)-[:KNOWS*1..2]-(friend:Person),
  (friend)<-[:HAS_CREATOR]-(friendPost:Post)-[:HAS_TAG]->(knownTag:Tag)
WHERE NOT(person = friend) // I think this condition is unnecessary as Cypher will not travel
    the same edge twice (szarnyasg)
MATCH (friendPost)-[:HAS_TAG]->(commonTag:Tag)
WHERE NOT(commonTag = knownTag)
WITH DISTINCT commonTag, knownTag, friend
MATCH (commonTag)<-[:HAS_TAG]-(commonPost:Post)-[:HAS_TAG]->(knownTag)
WHERE (commonPost)-[:HAS_CREATOR]->(friend)
RETURN
  commonTag.name AS tagName,
  count(commonPost) AS postCount
ORDER BY
  postCount DESC,
  tagName ASC
LIMIT 10
```

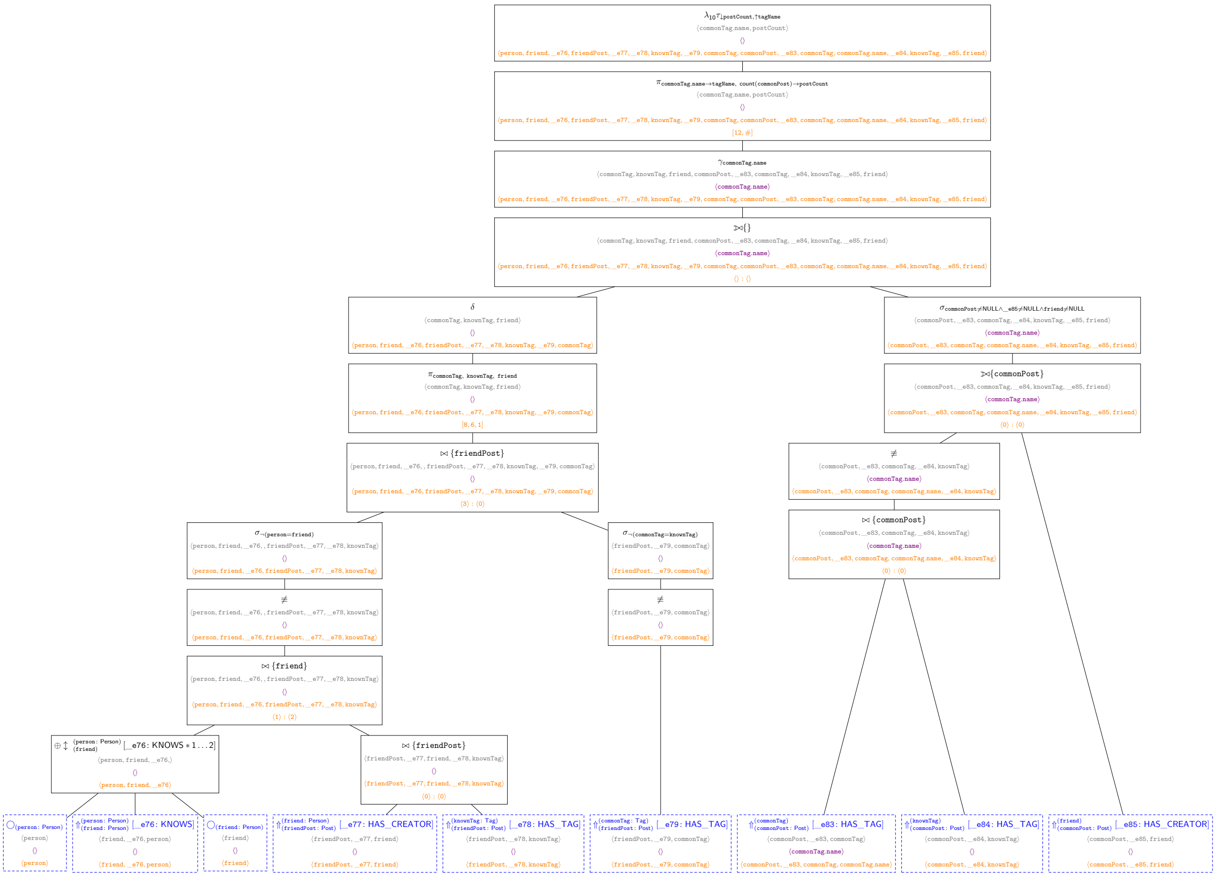
Relational algebra expression for search-based evaluation (query-6)

$$
\begin{aligned}
r = \lambda_{10} &\Bigl(\tau_{\downarrow\texttt{postCount},\uparrow\texttt{tagName}}\Bigl(\pi_{\texttt{commonTag.name}\rightarrow\texttt{tagName},\ \texttt{count(commonPost)}\rightarrow\texttt{postCount}}\Bigl(\gamma_{\texttt{commonTag.name}}\Bigl(\delta \\
&\Bigl(\pi_{\texttt{commonTag},\ \texttt{knownTag},\ \texttt{friend}}\Bigl(\sigma_{\neg(\texttt{person}=\texttt{friend})}\Bigl(\not\equiv\Bigl(\updownarrow\,^{(\texttt{friend}:\ \mathsf{Person})}_{(\texttt{person})}\,[\_\mathsf{e76}\colon \mathsf{KNOWS} * 1 \ldots 2] \\
&\Bigl(\bigcirc_{(\texttt{person}:\ \mathsf{Person})}\Bigr) \bowtie\uparrow\,^{(\texttt{knownTag}:\ \mathsf{Tag})}_{(\texttt{friendPost})}\,[\_\mathsf{e78}\colon \mathsf{HAS\_TAG}] \\
&\Bigl(\downarrow\,^{(\texttt{friendPost}:\ \mathsf{Post})}_{(\texttt{friend})}\,[\_\mathsf{e77}\colon \mathsf{HAS\_CREATOR}]\Bigl(\bigcirc_{(\texttt{friend}:\ \mathsf{Person})}\Bigr)\Bigr)\Bigr)\Bigr) \bowtie \sigma_{\neg(\texttt{commonTag}=\texttt{knownTag})}\Bigl(\not\equiv \\
&\Bigl(\uparrow\,^{(\texttt{commonTag}:\ \mathsf{Tag})}_{(\texttt{friendPost})}\,[\_\mathsf{e79}\colon \mathsf{HAS\_TAG}]\Bigl(\bigcirc_{(\texttt{friendPost}:\ \mathsf{Post})}\Bigr)\Bigr)\Bigr)\Bigr)\Bigr)⟕ \\
&\sigma_{\texttt{commonPost}\neq\mathsf{NULL}\wedge\_\mathsf{e85}\neq\mathsf{NULL}\wedge\texttt{friend}\neq\mathsf{NULL}}\Bigl(\not\equiv\Bigl(\uparrow\,^{(\texttt{knownTag})}_{(\texttt{commonPost})}\,[\_\mathsf{e84}\colon \mathsf{HAS\_TAG}] \\
&\Bigl(\downarrow\,^{(\texttt{commonPost}:\ \mathsf{Post})}_{(\texttt{commonTag})}\,[\_\mathsf{e83}\colon \mathsf{HAS\_TAG}]\Bigl(\bigcirc_{(\texttt{commonTag})}\Bigr)\Bigr)\Bigr)⟕ \\
&\uparrow\,^{(\texttt{friend})}_{(\texttt{commonPost})}\,[\_\mathsf{e85}\colon \mathsf{HAS\_CREATOR}]\Bigl(\bigcirc_{(\texttt{commonPost}:\ \mathsf{Post})}\Bigr)\Bigr)\Bigr)\Bigr)\Bigr)\Bigr)
\end{aligned}
$$

Relational algebra tree for search-based evaluation (query-6)


λ10
⟨commonTag.name, postCount⟩
τ↓postCount,↑tagName
⟨commonTag.name, postCount⟩
πcommonTag.name→tagName, count(commonPost)→postCount
⟨commonTag.name, postCount⟩
γcommonTag.name
⟨commonTag, knownTag, friend, commonTag, _e83, commonPost, _e84, knownTag, _e85, friend⟩
⟕{}
⟨commonTag, knownTag, friend, commonTag, _e83, commonPost, _e84, knownTag, _e85, friend⟩
δ
⟨commonTag, knownTag, friend⟩
πcommonTag, knownTag, friend
⟨commonTag, knownTag, friend⟩
⋈ {friendPost}
⟨person, _e76, friend, _e77, friendPost, _e78, knownTag, _e79, commonTag⟩
σ¬(person=friend)
⟨person, _e76, friend, _e77, friendPost, _e78, knownTag⟩
≢
⟨person, _e76, friend, _e77, friendPost, _e78, knownTag⟩
⋈ {friend}
⟨person, _e76, friend, _e77, friendPost, _e78, knownTag⟩
↕ (friend: Person) (person) [_e76: KNOWS ∗ 1 . . . 2]
⟨person, _e76, friend⟩
↑ (knownTag: Tag) (friendPost) [_e78: HAS_TAG]
⟨friend, _e77, friendPost, _e78, knownTag⟩
↓ (friendPost: Post) (friend) [_e77: HAS_CREATOR]
⟨friend, _e77, friendPost⟩
σ¬(commonTag=knownTag)
⟨friendPost, _e79, commonTag⟩
≢
⟨friendPost, _e79, commonTag⟩
↑ (commonTag: Tag) (friendPost) [_e79: HAS_TAG]
⟨friendPost, _e79, commonTag⟩
σcommonPost≠NULL∧_e85≠NULL∧friend≠NULL
⟨commonTag, _e83, commonPost, _e84, knownTag, _e85, friend⟩
⟕{commonPost}
⟨commonTag, _e83, commonPost, _e84, knownTag, _e85, friend⟩
≢
⟨commonTag, _e83, commonPost, _e84, knownTag⟩
↑ (knownTag) (commonPost) [_e84: HAS_TAG]
⟨commonTag, _e83, commonPost, _e84, knownTag⟩
↓ (commonPost: Post) (commonTag) [_e83: HAS_TAG]
⟨commonTag, _e83, commonPost⟩
↑ (friend) (commonPost) [_e85: HAS_CREATOR]
⟨commonPost, _e85, friend⟩
◯(person: Person)
⟨person⟩
◯(friend: Person)
⟨friend⟩
◯(friendPost: Post)
⟨friendPost⟩
◯(commonTag)
⟨commonTag⟩
◯(commonPost: Post)
⟨commonPost⟩

Incremental relational algebra tree (query-6)


$\lambda_{10}\tau_{\downarrow \texttt{postCount}, \uparrow \texttt{tagName}}$
⟨commonTag.name, postCount⟩
⟨⟩
⟨person, friend, __e76, friendPost, __e77, __e78, knownTag, __e79, commonTag, commonPost, __e83, commonTag, commonTag.name, __e84, knownTag, __e85, friend⟩
$\pi_{\texttt{commonTag.name} \rightarrow \texttt{tagName}, \texttt{count(commonPost)} \rightarrow \texttt{postCount}}$
⟨commonTag.name, postCount⟩
⟨⟩
⟨person, friend, __e76, friendPost, __e77, __e78, knownTag, __e79, commonTag, commonPost, __e83, commonTag, commonTag.name, __e84, knownTag, __e85, friend⟩
[12, #]
$\gamma_{\texttt{commonTag.name}}$
⟨commonTag, knownTag, friend, commonPost, __e83, commonTag, __e84, knownTag, __e85, friend⟩
⟨commonTag.name⟩
⟨person, friend, __e76, friendPost, __e77, __e78, knownTag, __e79, commonTag, commonPost, __e83, commonTag, commonTag.name, __e84, knownTag, __e85, friend⟩
⟕{}
⟨commonTag, knownTag, friend, commonPost, __e83, commonTag, __e84, knownTag, __e85, friend⟩
⟨commonTag.name⟩
⟨person, friend, __e76, friendPost, __e77, __e78, knownTag, __e79, commonTag, commonPost, __e83, commonTag, commonTag.name, __e84, knownTag, __e85, friend⟩
⟨⟩ : ⟨⟩
δ
⟨commonTag, knownTag, friend⟩
⟨⟩
⟨person, friend, __e76, friendPost, __e77, __e78, knownTag, __e79, commonTag⟩
$\pi_{\texttt{commonTag}, \texttt{knownTag}, \texttt{friend}}$
⟨commonTag, knownTag, friend⟩
⟨⟩
⟨person, friend, __e76, friendPost, __e77, __e78, knownTag, __e79, commonTag⟩
[8, 6, 1]
⋈ {friendPost}
⟨person, friend, __e76, , friendPost, __e77, __e78, knownTag, __e79, commonTag⟩
⟨⟩
⟨person, friend, __e76, friendPost, __e77, __e78, knownTag, __e79, commonTag⟩
⟨3⟩ : ⟨0⟩
$\sigma_{\neg(\texttt{person}=\texttt{friend})}$
⟨person, friend, __e76, , friendPost, __e77, __e78, knownTag⟩
⟨⟩
⟨person, friend, __e76, friendPost, __e77, __e78, knownTag⟩
≢
⟨person, friend, __e76, , friendPost, __e77, __e78, knownTag⟩
⟨⟩
⟨person, friend, __e76, friendPost, __e77, __e78, knownTag⟩
⋈ {friend}
⟨person, friend, __e76, , friendPost, __e77, __e78, knownTag⟩
⟨⟩
⟨person, friend, __e76, friendPost, __e77, __e78, knownTag⟩
⟨1⟩ : ⟨2⟩
⊕ ↕ (person: Person) (friend) [_e76: KNOWS ∗ 1 . . . 2]
⟨person, friend, __e76,⟩
⟨⟩
⟨person, friend, __e76⟩
⋈ {friendPost}
⟨friendPost, __e77, friend, __e78, knownTag⟩
⟨⟩
⟨friendPost, __e77, friend, __e78, knownTag⟩
⟨0⟩ : ⟨0⟩
$\sigma_{\neg(\texttt{commonTag}=\texttt{knownTag})}$
⟨friendPost, __e79, commonTag⟩
⟨⟩
⟨friendPost, __e79, commonTag⟩
≢
⟨friendPost, __e79, commonTag⟩
⟨⟩
⟨friendPost, __e79, commonTag⟩
$\sigma_{\texttt{commonPost} \neq \text{NULL} \wedge \texttt{\_\_e85} \neq \text{NULL} \wedge \texttt{friend} \neq \text{NULL}}$
⟨commonPost, __e83, commonTag, __e84, knownTag, __e85, friend⟩
⟨commonTag.name⟩
⟨commonPost, __e83, commonTag, commonTag.name, __e84, knownTag, __e85, friend⟩
⟕{commonPost}
⟨commonPost, __e83, commonTag, __e84, knownTag, __e85, friend⟩
⟨commonTag.name⟩
⟨commonPost, __e83, commonTag, commonTag.name, __e84, knownTag, __e85, friend⟩
⟨0⟩ : ⟨0⟩
≢
⟨commonPost, __e83, commonTag, __e84, knownTag⟩
⟨commonTag.name⟩
⟨commonPost, __e83, commonTag, commonTag.name, __e84, knownTag⟩
⋈ {commonPost}
⟨commonPost, __e83, commonTag, __e84, knownTag⟩
⟨commonTag.name⟩
⟨commonPost, __e83, commonTag, commonTag.name, __e84, knownTag⟩
⟨0⟩ : ⟨0⟩
○(person: Person)
⟨person⟩
⟨⟩
⟨person⟩
⇑(person: Person) (friend: Person) [_e76: KNOWS]
⟨friend, __e76, person⟩
⟨⟩
⟨friend, __e76, person⟩
○(friend: Person)
⟨friend⟩
⟨⟩
⟨friend⟩
⇑(friend: Person) (friendPost: Post) [_e77: HAS_CREATOR]
⟨friendPost, __e77, friend⟩
⟨⟩
⟨friendPost, __e77, friend⟩
⇑(knownTag: Tag) (friendPost: Post) [_e78: HAS_TAG]
⟨friendPost, __e78, knownTag⟩
⟨⟩
⟨friendPost, __e78, knownTag⟩
⇑(commonTag: Tag) (friendPost: Post) [_e79: HAS_TAG]
⟨friendPost, __e79, commonTag⟩
⟨⟩
⟨friendPost, __e79, commonTag⟩
⇑(commonTag) (commonPost: Post) [_e83: HAS_TAG]
⟨commonPost, __e83, commonTag⟩
⟨commonTag.name⟩
⟨commonPost, __e83, commonTag, commonTag.name⟩
⇑(knownTag) (commonPost: Post) [_e84: HAS_TAG]
⟨commonPost, __e84, knownTag⟩
⟨⟩
⟨commonPost, __e84, knownTag⟩
⇑(friend) (commonPost: Post) [_e85: HAS_CREATOR]
⟨commonPost, __e85, friend⟩
⟨⟩
⟨commonPost, __e85, friend⟩

C.1.5 query-7

Query specification (query-7)

```
// Q7. Recent likes. For the specified Person get the most recent likes of any of the person's posts, and the latency between the corresponding post and the like. Flag Likes from outside the direct connections. Return top 20 Likes, ordered descending by creation date of the like.
MATCH (person:Person)<-[:HAS_CREATOR]-(message)<-[like:LIKES]-(liker:Person)
WITH liker, message, like.creationDate AS likeTime, person
ORDER BY
  likeTime DESC,
  message.id ASC
WITH liker, head(collect({msg: message, likeTime: likeTime})) AS latestLike, person
RETURN
  liker.id AS personId,
  liker.firstName AS personFirstName,
  liker.lastName AS personLastName,
  latestLike.likeTime AS likeTime,
  NOT((liker)-[:KNOWS]-(person)) AS isNew,
  latestLike.msg.id AS messageId,
  latestLike.msg.content AS messageContent,
  latestLike.likeTime - latestLike.msg.creationDate AS latencyAsMilli
ORDER BY
  likeTime DESC,
  personId ASC
LIMIT 10
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

C.1.6 query-8

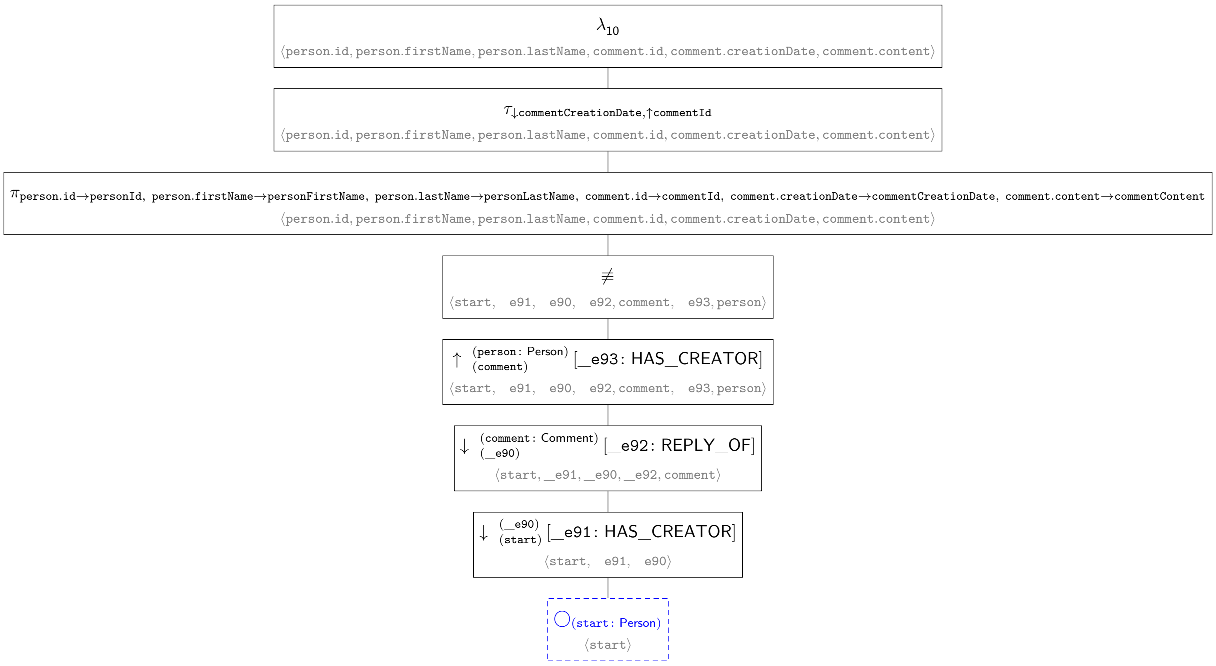
Query specification (query-8)

```
// Q8. Most recent replies. This query retrieves the 20 most recent reply comments to all the posts and comments of Person, ordered descending by creation date.
MATCH (start:Person)<-[:HAS_CREATOR]-()<-[:REPLY_OF]-(comment:Comment)-[:HAS_CREATOR]->(person:Person)
RETURN
  person.id AS personId,
  person.firstName AS personFirstName,
  person.lastName AS personLastName,
  comment.id AS commentId,
  comment.creationDate AS commentCreationDate,
  comment.content AS commentContent
ORDER BY
  commentCreationDate DESC,
  commentId ASC
LIMIT 10
```

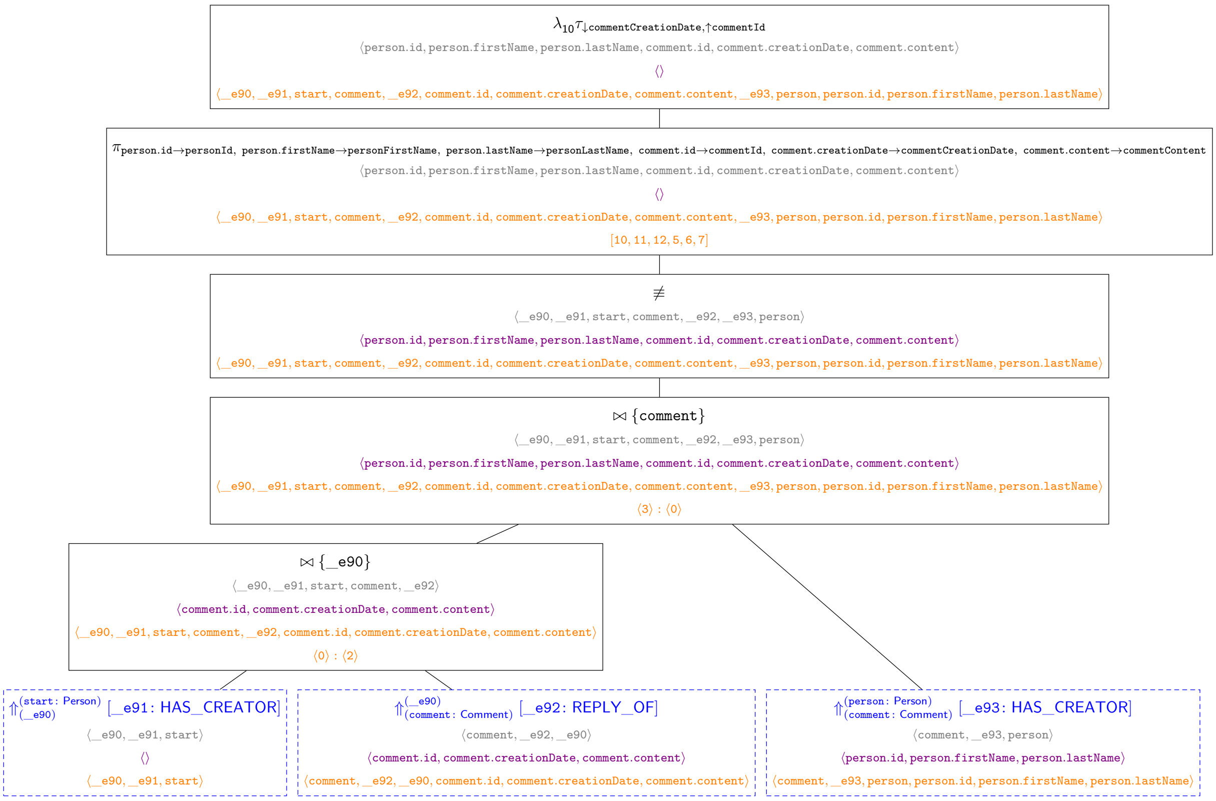
Relational algebra expression for search-based evaluation (query-8)

$$r = \lambda_{10}\Big(\tau_{\downarrow\texttt{commentCreationDate},\uparrow\texttt{commentId}}$$
$$\Big(\pi_{\texttt{person.id}\to\texttt{personId},\ \texttt{person.firstName}\to\texttt{personFirstName},\ \texttt{person.lastName}\to\texttt{personLastName},\ \texttt{comment.id}\to\texttt{commentId},\ \texttt{comment.creationDate}-}$$
$$\Big(\not\equiv\Big(\uparrow\,{\substack{(\texttt{person}:\,\mathsf{Person})\\(\texttt{comment})}}\,[\_\mathsf{e93}:\,\mathsf{HAS\_CREATOR}]\Big(\downarrow\,{\substack{(\texttt{comment}:\,\mathsf{Comment})\\(\_\mathsf{e90})}}\,[\_\mathsf{e92}:\,\mathsf{REPLY\_OF}]$$
$$\Big(\downarrow\,{\substack{(\_\mathsf{e90})\\(\texttt{start})}}\,[\_\mathsf{e91}:\,\mathsf{HAS\_CREATOR}]\Big(\bigcirc_{(\texttt{start}:\,\mathsf{Person})}\Big)\Big)\Big)\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (query-8)


λ10
⟨person.id, person.firstName, person.lastName, comment.id, comment.creationDate, comment.content⟩
τ↓commentCreationDate,↑commentId
⟨person.id, person.firstName, person.lastName, comment.id, comment.creationDate, comment.content⟩
πperson.id→personId, person.firstName→personFirstName, person.lastName→personLastName, comment.id→commentId, comment.creationDate→commentCreationDate, comment.content→commentContent
⟨person.id, person.firstName, person.lastName, comment.id, comment.creationDate, comment.content⟩
≢
⟨start, _e91, _e90, _e92, comment, _e93, person⟩
↑ (person: Person) (comment) [_e93: HAS_CREATOR]
⟨start, _e91, _e90, _e92, comment, _e93, person⟩
↓ (comment: Comment) (_e90) [_e92: REPLY_OF]
⟨start, _e91, _e90, _e92, comment⟩
↓ (_e90) (start) [_e91: HAS_CREATOR]
⟨start, _e91, _e90⟩
○(start: Person)
⟨start⟩

Incremental relational algebra tree (query-8)


$\lambda_{10}\tau_{\downarrow\texttt{commentCreationDate},\uparrow\texttt{commentId}}$
⟨person.id, person.firstName, person.lastName, comment.id, comment.creationDate, comment.content⟩
⟨⟩
⟨_e90, _e91, start, comment, _e92, comment.id, comment.creationDate, comment.content, _e93, person, person.id, person.firstName, person.lastName⟩
$\pi_{\texttt{person.id}\to\texttt{personId},\ \texttt{person.firstName}\to\texttt{personFirstName},\ \texttt{person.lastName}\to\texttt{personLastName},\ \texttt{comment.id}\to\texttt{commentId},\ \texttt{comment.creationDate}\to\texttt{commentCreationDate},\ \texttt{comment.content}\to\texttt{commentContent}}$
⟨person.id, person.firstName, person.lastName, comment.id, comment.creationDate, comment.content⟩
⟨⟩
⟨_e90, _e91, start, comment, _e92, comment.id, comment.creationDate, comment.content, _e93, person, person.id, person.firstName, person.lastName⟩
[10, 11, 12, 5, 6, 7]
$\not\equiv$
⟨_e90, _e91, start, comment, _e92, _e93, person⟩
⟨person.id, person.firstName, person.lastName, comment.id, comment.creationDate, comment.content⟩
⟨_e90, _e91, start, comment, _e92, comment.id, comment.creationDate, comment.content, _e93, person, person.id, person.firstName, person.lastName⟩
⋈ {comment}
⟨_e90, _e91, start, comment, _e92, _e93, person⟩
⟨person.id, person.firstName, person.lastName, comment.id, comment.creationDate, comment.content⟩
⟨_e90, _e91, start, comment, _e92, comment.id, comment.creationDate, comment.content, _e93, person, person.id, person.firstName, person.lastName⟩
⟨3⟩ : ⟨0⟩
⋈ {_e90}
⟨_e90, _e91, start, comment, _e92⟩
⟨comment.id, comment.creationDate, comment.content⟩
⟨_e90, _e91, start, comment, _e92, comment.id, comment.creationDate, comment.content⟩
⟨0⟩ : ⟨2⟩
$\Uparrow^{(\texttt{start}:\ \text{Person})}_{(\_\text{e90})}$ [_e91: HAS_CREATOR]
⟨_e90, _e91, start⟩
⟨⟩
⟨_e90, _e91, start⟩
$\Uparrow^{(\_\text{e90})}_{(\texttt{comment}:\ \text{Comment})}$ [_e92: REPLY_OF]
⟨comment, _e92, _e90⟩
⟨comment.id, comment.creationDate, comment.content⟩
⟨comment, _e92, _e90, comment.id, comment.creationDate, comment.content⟩
$\Uparrow^{(\texttt{person}:\ \text{Person})}_{(\texttt{comment}:\ \text{Comment})}$ [_e93: HAS_CREATOR]
⟨comment, _e93, person⟩
⟨person.id, person.firstName, person.lastName⟩
⟨comment, _e93, person, person.id, person.firstName, person.lastName⟩

C.1.7 query-10

Query specification (query-10)

```
// Q10. Friend recommendation. Find top 10 friends of a friend who posts much about the interests of Person and little about not interesting topics for the user. The search is restricted by the candidate's horoscopeSign. Returns friends for whom the difference between the total number of their posts about the interests of the specified user and the total number of their posts about topics that are not interests of the user, is as large as possible. Sort the result descending by this difference.
MATCH (person:Person)-[:KNOWS*2..2]-(friend:Person)-[:IS_LOCATED_IN]->(city:City)
WHERE ((friend.birthday_month = $month AND friend.birthday_day >= 21)
   OR (friend.birthday_month = ($month + 1) % 12 AND friend.birthday_day < 22))
  AND NOT(friend = person) // I think this condition is unnecessary as Cypher will not travel the same edge twice (szarnyasg)
  AND NOT((friend)-[:KNOWS]-(person))
WITH DISTINCT friend, city, person
OPTIONAL MATCH (friend)<-[:HAS_CREATOR]-(post:Post)
WITH
  friend,
  city,
  collect(post) AS posts,
  person
WITH
  friend,
  city,
  length(posts) AS postCount,
  length([p IN posts WHERE (p)-[:HAS_TAG]->(:Tag)<-[:HAS_INTEREST]-(person)]) AS commonPostCount
RETURN
  friend.id AS personId,
  friend.firstName AS personFirstName,
  friend.lastName AS personLastName,
  friend.gender AS personGender,
  city.name AS personCityName,
  commonPostCount - (postCount - commonPostCount) AS commonInterestScore
ORDER BY
  commonInterestScore DESC,
  personId ASC
LIMIT 10
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

C.1.8 query-11

Query specification (query-11)

```
// Q11. Job referral. Find top 10 friends of the specified Person, or a friend of her friend (excluding the specified person), who has long worked in a company in a specified Country. Sort ascending by start date, and then ascending by person identifier.
MATCH (person:Person)-[:KNOWS*1..2]-(friend:Person)
WHERE NOT(person = friend) // I think this condition is unnecessary as Cypher will not travel the same edge twice (szarnyasg)
WITH DISTINCT friend
MATCH (friend)-[worksAt:WORKS_AT]->(company:Company)-[:IS_LOCATED_IN]->(:Country)
WHERE worksAt.workFrom < $date
RETURN
  friend.id AS friendId,
  friend.firstName AS friendFirstName,
  friend.lastName AS friendLastName,
  worksAt.workFrom AS workFromYear,
  company.name AS companyName
ORDER BY
  workFromYear ASC,
  friendId ASC,
  companyName DESC
LIMIT 10
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

C.1.9 query-12

Query specification (query-12)

```
// Q12. Expert Search. Find friends of a Person who have replied the most to posts with a tag in a given TagCategory. Return top 20 persons, sorted descending by number of replies.
MATCH (:Person)-[:KNOWS]-(friend:Person)
OPTIONAL MATCH
  (friend)<-[:HAS_CREATOR]-(comment:Comment)-[:REPLY_OF]->(:Post)-[:HAS_TAG]->(tag:Tag),
  (tag)-[:HAS_TYPE]->(tagClass:TagClass)-[:IS_SUBCLASS_OF*0..]->(baseTagClass:TagClass)
WHERE tagClass.name = $class
   OR baseTagClass.name = $class
RETURN
  friend.id AS friendId,
  friend.firstName AS friendFirstName,
  friend.lastName AS friendLastName,
  collect(DISTINCT tag.name) AS tagNames,
  count(DISTINCT comment) AS count
ORDER BY
  count DESC,
  friendId ASC
LIMIT 10
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

Appendix D

LDBC Social Network Benchmark – Business Intelligence Workload

Total progress:

0 of 0

D.1 Queries

Progress:

0 of 0

Appendix E

Movie Database

Total progress:

E.1 Queries

Progress:

E.1.1 query-1

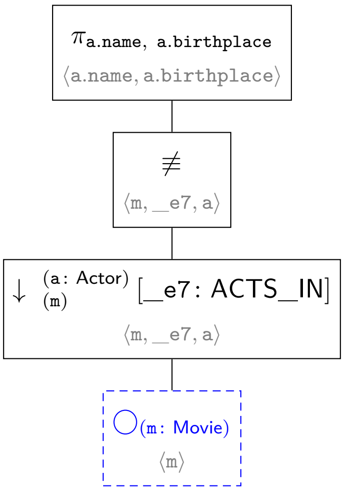
Query specification (query-1)

```
MATCH (m:Movie {title: 'Forrest Gump'})<-[:ACTS_IN]-(a:Actor)
RETURN a.name, a.birthplace
```

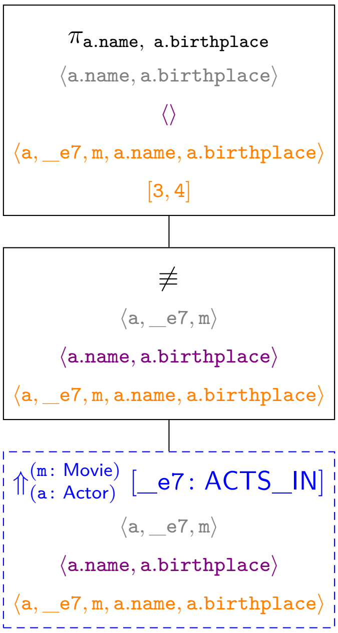
Relational algebra expression for search-based evaluation (query-1)

$$r = \pi_{\texttt{a.name, a.birthplace}}\left(\not\equiv\left(\downarrow^{(\texttt{a}:\,\textsf{Actor})}_{(\texttt{m})}[\_\textsf{e7}:\textsf{ACTS\_IN}]\left(\bigcirc_{(\texttt{m}:\,\textsf{Movie})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (query-1)


$\pi_{\texttt{a.name, a.birthplace}}$
⟨a.name, a.birthplace⟩
$\not\equiv$
⟨m, _e7, a⟩
$\downarrow^{(\texttt{a}:\,\textsf{Actor})}_{(\texttt{m})}[\_\textsf{e7}:\textsf{ACTS\_IN}]$
⟨m, _e7, a⟩
$\bigcirc_{(\texttt{m}:\,\textsf{Movie})}$
⟨m⟩

Incremental relational algebra tree (query-1)


π a.name, a.birthplace
⟨a.name, a.birthplace⟩
⟨⟩
⟨a, _e7, m, a.name, a.birthplace⟩
[3, 4]
≢
⟨a, _e7, m⟩
⟨a.name, a.birthplace⟩
⟨a, _e7, m, a.name, a.birthplace⟩
⇑(m: Movie) (a: Actor) [_e7: ACTS_IN]
⟨a, _e7, m⟩
⟨a.name, a.birthplace⟩
⟨a, _e7, m, a.name, a.birthplace⟩

E.1.2 query-2

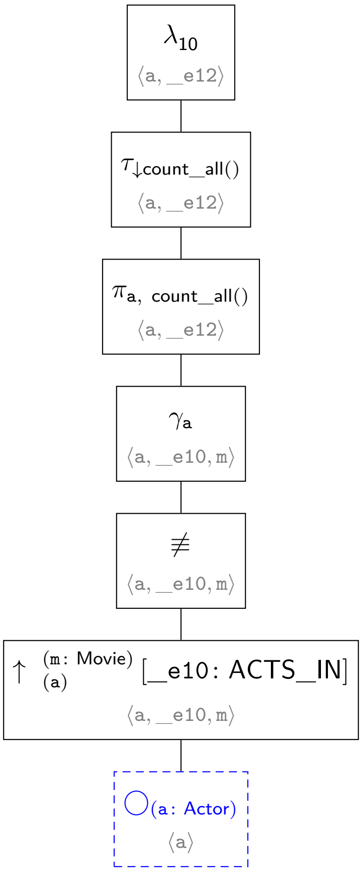
Query specification (query-2)

```
MATCH (a:Actor)-[:ACTS_IN]->(m:Movie)
RETURN a, count(*)
ORDER BY count(*) DESC LIMIT 10;
```

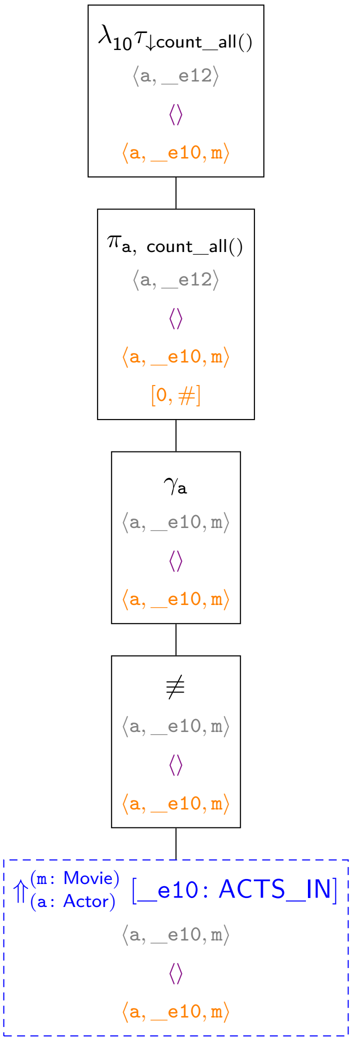
Relational algebra expression for search-based evaluation (query-2)

$$r = \lambda_{10}\left(\tau_{\downarrow \mathsf{count\_all()}}\left(\pi_{\mathsf{a},\ \mathsf{count\_all()}}\left(\gamma_{\mathsf{a}}\left(\not\equiv\left(\uparrow^{(\mathtt{m}:\ \mathsf{Movie})}_{(\mathtt{a})}[\_\mathsf{e10}:\ \mathsf{ACTS\_IN}]\left(\bigcirc_{(\mathsf{a}:\ \mathsf{Actor})}\right)\right)\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (query-2)


λ_{10}
$\langle$a, __e12$\rangle$
$\tau_{\downarrow \text{count\_all()}}$
$\langle$a, __e12$\rangle$
$\pi_{\text{a, count\_all()}}$
$\langle$a, __e12$\rangle$
γ_{a}
$\langle$a, __e10, m$\rangle$
$\not\equiv$
$\langle$a, __e10, m$\rangle$
$\uparrow^{(\text{m: Movie})}_{(\text{a})}$ [_e10: ACTS_IN]
$\langle$a, __e10, m$\rangle$
$\bigcirc_{(\text{a: Actor})}$
$\langle$a$\rangle$

Incremental relational algebra tree (query-2)



E.1.3 query-3

Query specification (query-3)

```
MATCH (a:Actor)-[:ACTS_IN]->(m:Movie)
WITH a, count(m) AS movie_count
WHERE movie_count < 3
RETURN a, movie_count
ORDER BY movie_count DESC LIMIT 5;
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

E.1.4 query-4

Query specification (query-4)

```
MATCH (a:Actor)-[:ACTS_IN]->(m:Movie)
WITH a, collect(m.title) AS movies
WHERE length(movies) >= 20
RETURN a, movies
ORDER BY length(movies) DESC LIMIT 10
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

E.1.5 query-5

Query specification (query-5)

```
MATCH (a:Actor)-[:ACTS_IN]->(m:Movie)
WITH a, count(m) AS acted
WHERE acted >= 10
WITH a, acted
MATCH (a:Director)-[:DIRECTED]->(m:Movie)
WITH a, acted, collect(m.title) AS directed
WHERE length(directed) >= 2
RETURN a.name, acted, directed
ORDER BY length(directed) DESC, acted DESC
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

E.1.6 query-6

Query specification (query-6)

```
MATCH (a:Actor:Director)-[:ACTS_IN]->(m:Movie)
WITH a, count(1) AS acted
WHERE acted >= 10
WITH a, acted
MATCH (a:Actor:Director)-[:DIRECTED]->(m:Movie)
WITH a, acted, collect(m.title) AS directed
WHERE length(directed) >= 2
RETURN a.name, acted, directed
ORDER BY length(directed) DESC, acted DESC
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

E.1.7 query-7

Query specification (query-7)

```
MATCH (a:Director)-[:ACTS_IN]->(m)
WITH a, count(m) AS acted
WHERE acted >= 10
WITH a, acted
MATCH (a)-[:DIRECTED]->(m)
WITH a, acted, collect(m.title) AS directed
WHERE length(directed) >= 2
RETURN a.name, acted, directed
ORDER BY length(directed) DESC, acted DESC
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

E.1.8 query-8

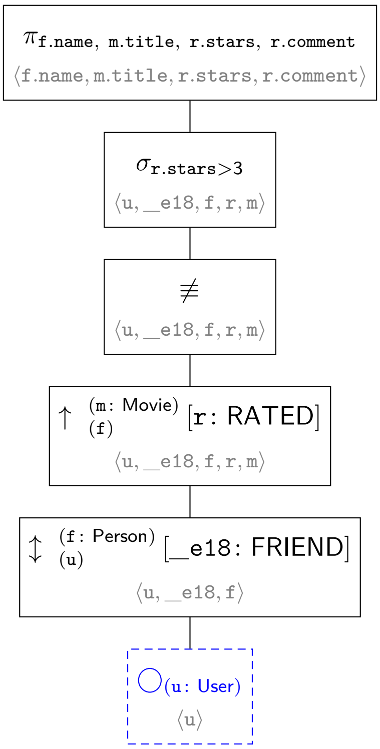
Query specification (query-8)

```
MATCH (u:User {login: 'Michael'})-[:FRIEND]-(f:Person)-[r:RATED]->(m:Movie)
WHERE r.stars > 3
RETURN f.name, m.title, r.stars, r.comment
```

Relational algebra expression for search-based evaluation (query-8)

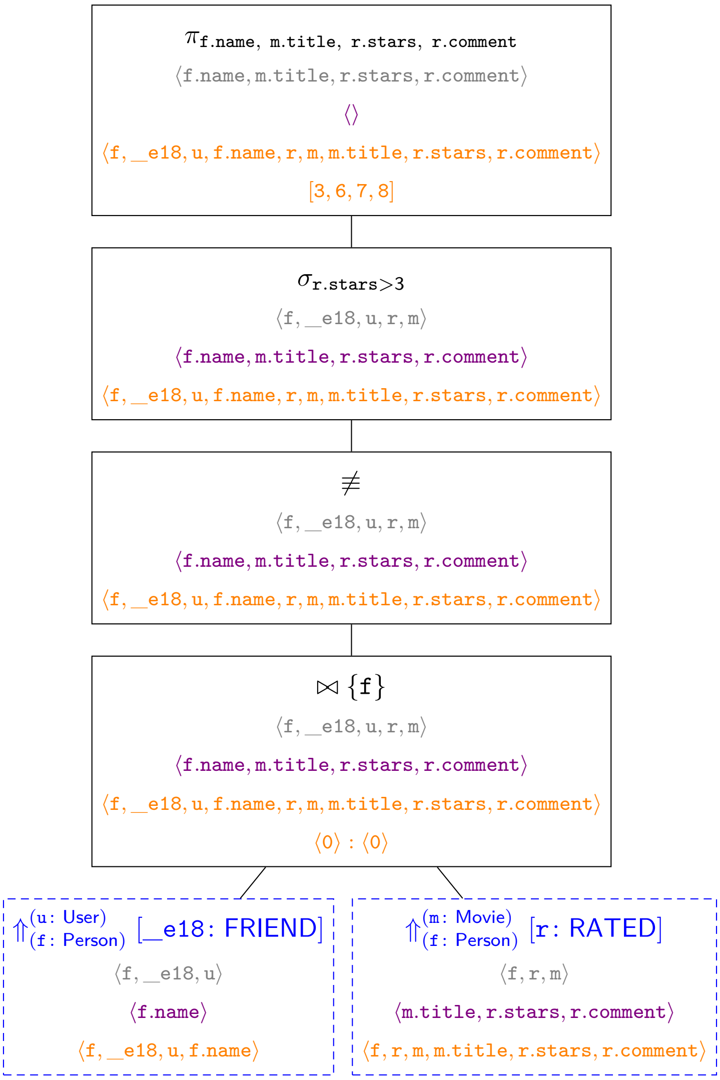
$$r = \pi_{\texttt{f.name},\ \texttt{m.title},\ \texttt{r.stars},\ \texttt{r.comment}}\left(\sigma_{\texttt{r.stars}>3}\left(\not\equiv\left(\uparrow^{(\texttt{m}\colon \textsf{Movie})}_{(\texttt{f})}[\texttt{r}\colon \textsf{RATED}]\left(\updownarrow^{(\texttt{f}\colon \textsf{Person})}_{(\texttt{u})}[\textsf{\_e18}\colon \textsf{FRIEND}]\left(\bigcirc_{(\texttt{u}\colon \textsf{User})}\right)\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (query-8)



$\pi_{\texttt{f.name},\ \texttt{m.title},\ \texttt{r.stars},\ \texttt{r.comment}}$
$\langle \texttt{f.name}, \texttt{m.title}, \texttt{r.stars}, \texttt{r.comment} \rangle$
$\sigma_{\texttt{r.stars}>3}$
$\langle \texttt{u}, \texttt{\_e18}, \texttt{f}, \texttt{r}, \texttt{m} \rangle$
$\not\equiv$
$\langle \texttt{u}, \texttt{\_e18}, \texttt{f}, \texttt{r}, \texttt{m} \rangle$
$\uparrow^{(\texttt{m}\colon \textsf{Movie})}_{(\texttt{f})}[\texttt{r}\colon \textsf{RATED}]$
$\langle \texttt{u}, \texttt{\_e18}, \texttt{f}, \texttt{r}, \texttt{m} \rangle$
$\updownarrow^{(\texttt{f}\colon \textsf{Person})}_{(\texttt{u})}[\textsf{\_e18}\colon \textsf{FRIEND}]$
$\langle \texttt{u}, \texttt{\_e18}, \texttt{f} \rangle$
$\bigcirc_{(\texttt{u}\colon \textsf{User})}$
$\langle \texttt{u} \rangle$

Incremental relational algebra tree (query-8)


π f.name, m.title, r.stars, r.comment
⟨f.name, m.title, r.stars, r.comment⟩
⟨⟩
⟨f, _e18, u, f.name, r, m, m.title, r.stars, r.comment⟩
[3, 6, 7, 8]
σ r.stars>3
⟨f, _e18, u, r, m⟩
⟨f.name, m.title, r.stars, r.comment⟩
⟨f, _e18, u, f.name, r, m, m.title, r.stars, r.comment⟩
≢
⟨f, _e18, u, r, m⟩
⟨f.name, m.title, r.stars, r.comment⟩
⟨f, _e18, u, f.name, r, m, m.title, r.stars, r.comment⟩
⋈ {f}
⟨f, _e18, u, r, m⟩
⟨f.name, m.title, r.stars, r.comment⟩
⟨f, _e18, u, f.name, r, m, m.title, r.stars, r.comment⟩
⟨0⟩ : ⟨0⟩
⇑ (u: User) (f: Person) [_e18: FRIEND]
⟨f, _e18, u⟩
⟨f.name⟩
⟨f, _e18, u, f.name⟩
⇑ (m: Movie) (f: Person) [r: RATED]
⟨f, r, m⟩
⟨m.title, r.stars, r.comment⟩
⟨f, r, m, m.title, r.stars, r.comment⟩

E.1.9 query-9

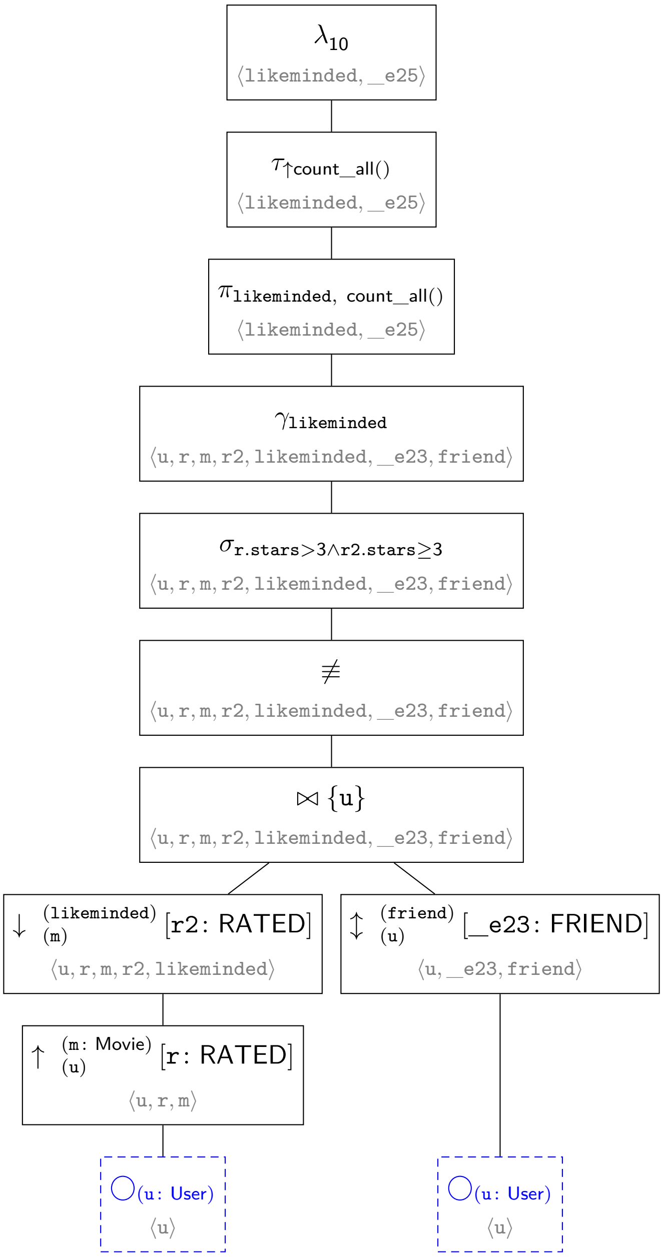
Query specification (query-9)

```
MATCH (u:User)-[r:RATED]->(m:Movie)<-[r2:RATED]-(likeminded),
(u)-[:FRIEND]-(friend)
WHERE r.stars > 3 AND r2.stars >= 3
RETURN likeminded, count(*)
ORDER BY count(*) desc LIMIT 10
```

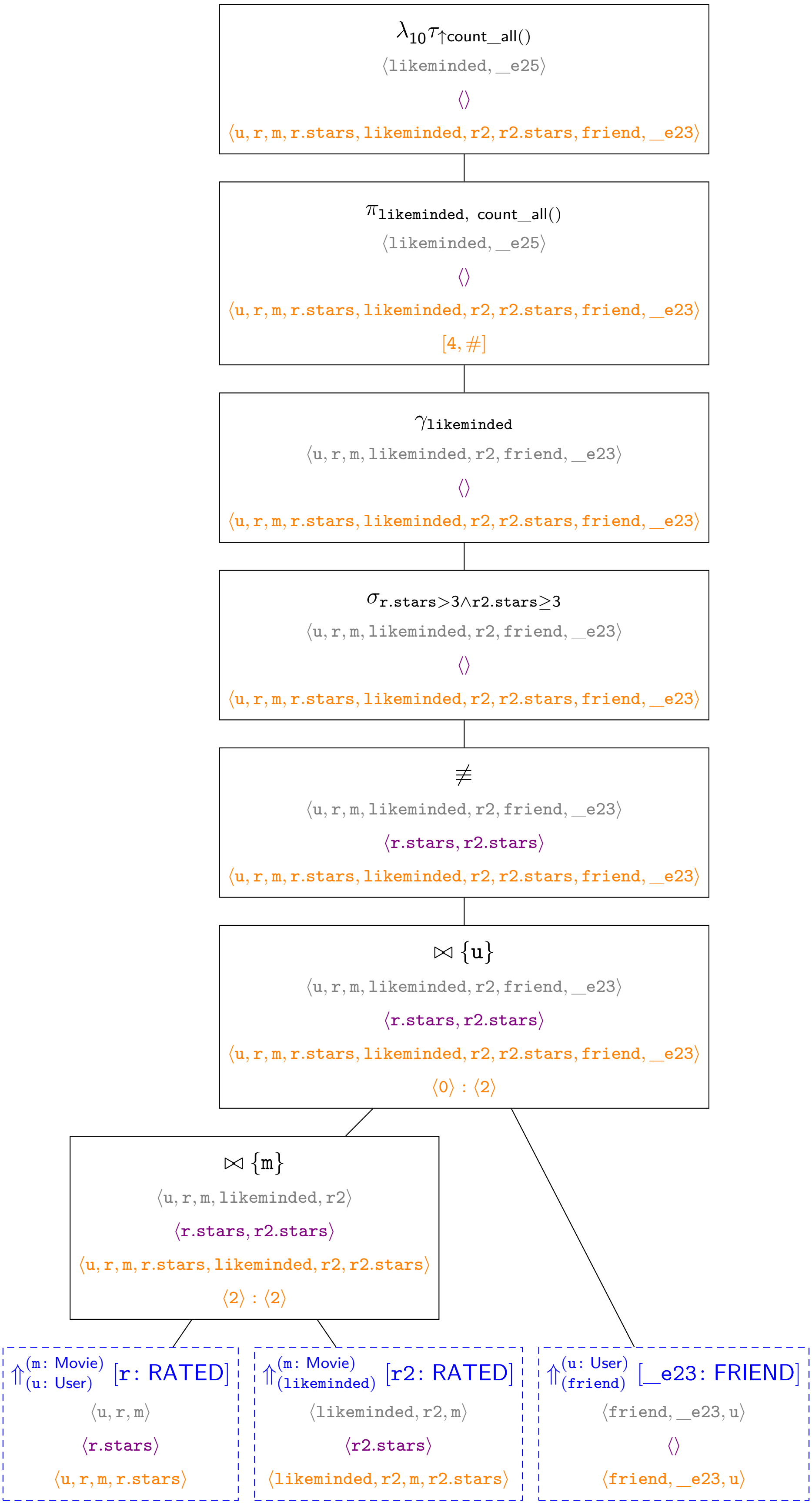
Relational algebra expression for search-based evaluation (query-9)

$$r = \lambda_{10}\Big(\tau_{\uparrow\mathsf{count\_all()}}\Big(\pi_{\mathtt{likeminded},\ \mathsf{count\_all()}}\Big(\gamma_{\mathtt{likeminded}}\Big(\sigma_{\mathtt{r.stars}>3\wedge\mathtt{r2.stars}\geq 3}\Big(\not\equiv \Big(\downarrow^{(\mathtt{likeminded})}_{(\mathtt{m})}[\mathtt{r2}\colon \mathsf{RATED}]\Big(\uparrow^{(\mathtt{m}\colon\ \mathsf{Movie})}_{(\mathtt{u})}[\mathtt{r}\colon \mathsf{RATED}]\Big(\bigcirc_{(\mathtt{u}\colon\ \mathsf{User})}\Big)\Big)\bowtie\!\updownarrow^{(\mathtt{friend})}_{(\mathtt{u})}[\_\mathsf{e23}\colon \mathsf{FRIEND}]\Big(\bigcirc_{(\mathtt{u}\colon\ \mathsf{User})}\Big)\Big)\Big)\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (query-9)


λ_{10}
⟨likeminded, _e25⟩
$\tau_{\uparrow \text{count\_all()}}$
⟨likeminded, _e25⟩
$\pi_{\text{likeminded, count\_all()}}$
⟨likeminded, _e25⟩
$\gamma_{\text{likeminded}}$
⟨u, r, m, r2, likeminded, _e23, friend⟩
$\sigma_{\text{r.stars}>3 \wedge \text{r2.stars} \geq 3}$
⟨u, r, m, r2, likeminded, _e23, friend⟩
$\not\equiv$
⟨u, r, m, r2, likeminded, _e23, friend⟩
$\bowtie \{u\}$
⟨u, r, m, r2, likeminded, _e23, friend⟩
$\downarrow$ (likeminded) (m) [r2: RATED]
⟨u, r, m, r2, likeminded⟩
$\updownarrow$ (friend) (u) [_e23: FRIEND]
⟨u, _e23, friend⟩
$\uparrow$ (m: Movie) (u) [r: RATED]
⟨u, r, m⟩
○(u: User)
⟨u⟩
○(u: User)
⟨u⟩

Incremental relational algebra tree (query-9)


λ10 τ↑count_all()
⟨likeminded, _e25⟩
⟨⟩
⟨u, r, m, r.stars, likeminded, r2, r2.stars, friend, _e23⟩
π likeminded, count_all()
⟨likeminded, _e25⟩
⟨⟩
⟨u, r, m, r.stars, likeminded, r2, r2.stars, friend, _e23⟩
[4, #]
γ likeminded
⟨u, r, m, likeminded, r2, friend, _e23⟩
⟨⟩
⟨u, r, m, r.stars, likeminded, r2, r2.stars, friend, _e23⟩
σ r.stars>3∧r2.stars≥3
⟨u, r, m, likeminded, r2, friend, _e23⟩
⟨⟩
⟨u, r, m, r.stars, likeminded, r2, r2.stars, friend, _e23⟩
≢
⟨u, r, m, likeminded, r2, friend, _e23⟩
⟨r.stars, r2.stars⟩
⟨u, r, m, r.stars, likeminded, r2, r2.stars, friend, _e23⟩
⋈ {u}
⟨u, r, m, likeminded, r2, friend, _e23⟩
⟨r.stars, r2.stars⟩
⟨u, r, m, r.stars, likeminded, r2, r2.stars, friend, _e23⟩
⟨0⟩ : ⟨2⟩
⋈ {m}
⟨u, r, m, likeminded, r2⟩
⟨r.stars, r2.stars⟩
⟨u, r, m, r.stars, likeminded, r2, r2.stars⟩
⟨2⟩ : ⟨2⟩
⇑ (m: Movie) (u: User) [r: RATED]
⟨u, r, m⟩
⟨r.stars⟩
⟨u, r, m, r.stars⟩
⇑ (m: Movie) (likeminded) [r2: RATED]
⟨likeminded, r2, m⟩
⟨r2.stars⟩
⟨likeminded, r2, m, r2.stars⟩
⇑ (u: User) (friend) [_e23: FRIEND]
⟨friend, _e23, u⟩
⟨⟩
⟨friend, _e23, u⟩

E.1.10 query-10

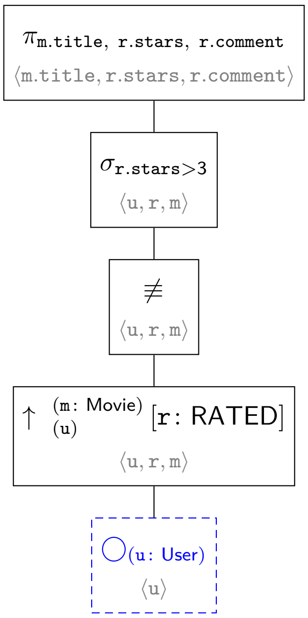
Query specification (query-10)

```
MATCH (u:User {login: 'Michael'})-[r:RATED]->(m:Movie)
WHERE r.stars > 3
RETURN m.title, r.stars, r.comment
```

Relational algebra expression for search-based evaluation (query-10)

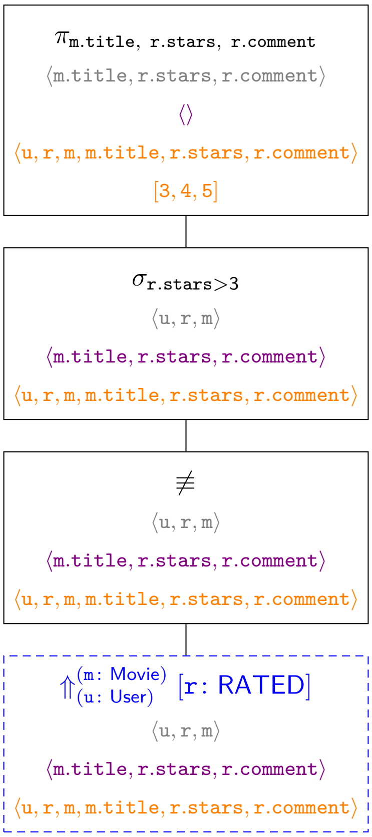
$$r = \pi_{\texttt{m.title},\ \texttt{r.stars},\ \texttt{r.comment}}\left(\sigma_{\texttt{r.stars}>3}\left(\not\equiv\left(\uparrow^{(\texttt{m}:\ \mathsf{Movie})}_{(\texttt{u})}[\texttt{r}:\ \mathsf{RATED}]\left(\bigcirc_{(\texttt{u}:\ \mathsf{User})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (query-10)



$\pi_{\texttt{m.title},\ \texttt{r.stars},\ \texttt{r.comment}}$
$\langle \texttt{m.title}, \texttt{r.stars}, \texttt{r.comment} \rangle$
$\sigma_{\texttt{r.stars}>3}$
$\langle \texttt{u}, \texttt{r}, \texttt{m} \rangle$
$\not\equiv$
$\langle \texttt{u}, \texttt{r}, \texttt{m} \rangle$
$\uparrow^{(\texttt{m}:\ \mathsf{Movie})}_{(\texttt{u})}[\texttt{r}:\ \mathsf{RATED}]$
$\langle \texttt{u}, \texttt{r}, \texttt{m} \rangle$
$\bigcirc_{(\texttt{u}:\ \mathsf{User})}$
$\langle \texttt{u} \rangle$

Incremental relational algebra tree (query-10)


$\pi_{\texttt{m.title},\ \texttt{r.stars},\ \texttt{r.comment}}$
$\langle \texttt{m.title}, \texttt{r.stars}, \texttt{r.comment} \rangle$
$\langle \rangle$
$\langle \texttt{u}, \texttt{r}, \texttt{m}, \texttt{m.title}, \texttt{r.stars}, \texttt{r.comment} \rangle$
$[3, 4, 5]$
$\sigma_{\texttt{r.stars}>3}$
$\langle \texttt{u}, \texttt{r}, \texttt{m} \rangle$
$\langle \texttt{m.title}, \texttt{r.stars}, \texttt{r.comment} \rangle$
$\langle \texttt{u}, \texttt{r}, \texttt{m}, \texttt{m.title}, \texttt{r.stars}, \texttt{r.comment} \rangle$
$\not\equiv$
$\langle \texttt{u}, \texttt{r}, \texttt{m} \rangle$
$\langle \texttt{m.title}, \texttt{r.stars}, \texttt{r.comment} \rangle$
$\langle \texttt{u}, \texttt{r}, \texttt{m}, \texttt{m.title}, \texttt{r.stars}, \texttt{r.comment} \rangle$
$\Uparrow^{(\texttt{m}:\ \text{Movie})}_{(\texttt{u}:\ \text{User})} [\texttt{r}: \text{RATED}]$
$\langle \texttt{u}, \texttt{r}, \texttt{m} \rangle$
$\langle \texttt{m.title}, \texttt{r.stars}, \texttt{r.comment} \rangle$
$\langle \texttt{u}, \texttt{r}, \texttt{m}, \texttt{m.title}, \texttt{r.stars}, \texttt{r.comment} \rangle$

E.1.11 query-11

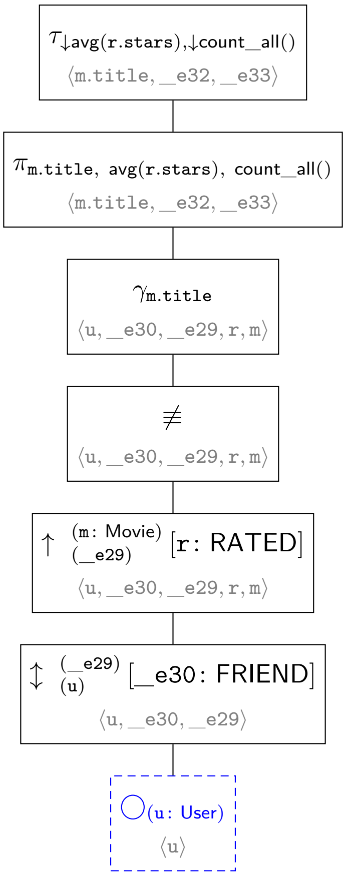
Query specification (query-11)

```
MATCH (u:User {login: 'Michael'})-[:FRIEND]-()-[r:RATED]->(m:Movie)
RETURN m.title, avg(r.stars), count(*)
ORDER BY AVG(r.stars) DESC, count(*) DESC
```

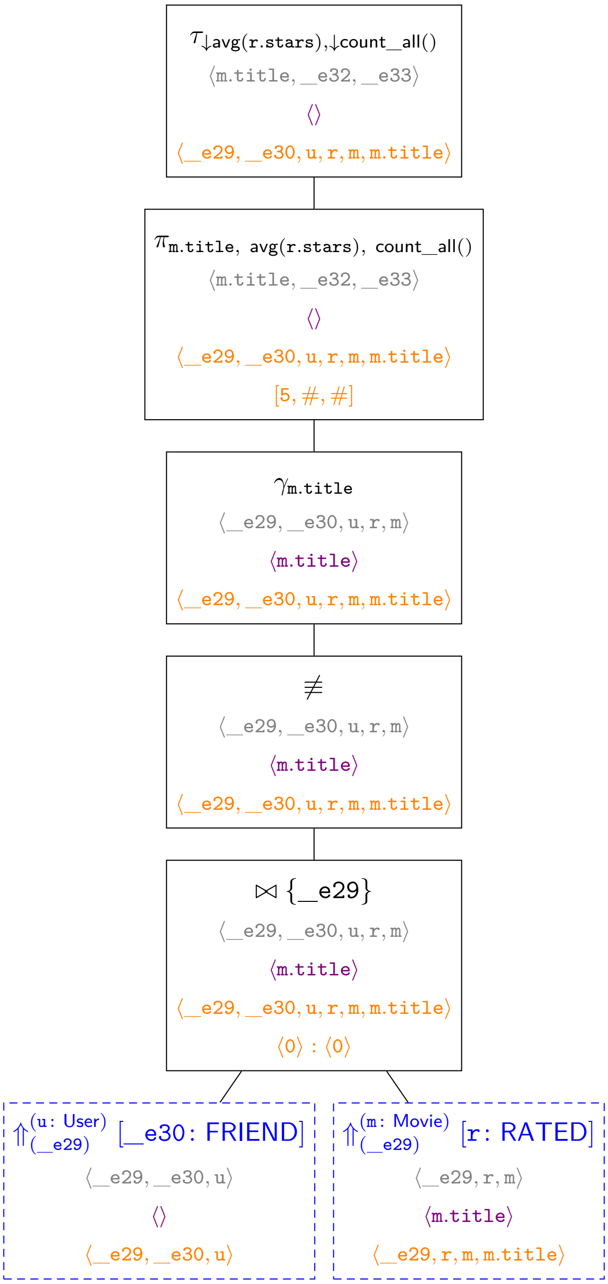
Relational algebra expression for search-based evaluation (query-11)

$$r = \tau_{\downarrow \text{avg}(\texttt{r.stars}), \downarrow \text{count\_all}()} \left(\pi_{\texttt{m.title},\ \text{avg}(\texttt{r.stars}),\ \text{count\_all}()} \left(\gamma_{\texttt{m.title}} \left(\not\equiv \left(\uparrow^{(\texttt{m}:\ \text{Movie})}_{(\text{\_e29})} [\texttt{r}: \text{RATED}] \left(\updownarrow^{(\text{\_e29})}_{(\texttt{u})} [\text{\_e30}: \text{FRIEND}] \left(\bigcirc_{(\texttt{u}:\ \text{User})} \right) \right) \right) \right) \right) \right)$$

Relational algebra tree for search-based evaluation (query-11)



Incremental relational algebra tree (query-11)


τ↓avg(r.stars),↓count_all()
⟨m.title, _e32, _e33⟩
⟨⟩
⟨_e29, _e30, u, r, m, m.title⟩
πm.title, avg(r.stars), count_all()
⟨m.title, _e32, _e33⟩
⟨⟩
⟨_e29, _e30, u, r, m, m.title⟩
[5, #, #]
γm.title
⟨_e29, _e30, u, r, m⟩
⟨m.title⟩
⟨_e29, _e30, u, r, m, m.title⟩
≢
⟨_e29, _e30, u, r, m⟩
⟨m.title⟩
⟨_e29, _e30, u, r, m, m.title⟩
⋈ {_e29}
⟨_e29, _e30, u, r, m⟩
⟨m.title⟩
⟨_e29, _e30, u, r, m, m.title⟩
⟨0⟩ : ⟨0⟩
⇑(u: User)(_e29) [_e30: FRIEND]
⟨_e29, _e30, u⟩
⟨⟩
⟨_e29, _e30, u⟩
⇑(m: Movie)(_e29) [r: RATED]
⟨_e29, r, m⟩
⟨m.title⟩
⟨_e29, r, m, m.title⟩

E.1.12 query-12

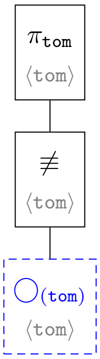
Query specification (query-12)

```
MATCH (tom {name: "Tom Hanks"})
RETURN tom
```

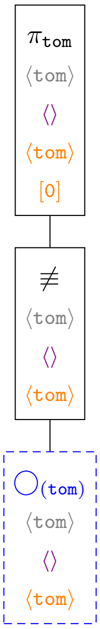
Relational algebra expression for search-based evaluation (query-12)

$$r = \pi_{\texttt{tom}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{tom})}\Big)\Big)$$

Relational algebra tree for search-based evaluation (query-12)



Incremental relational algebra tree (query-12)



E.1.13 query-13

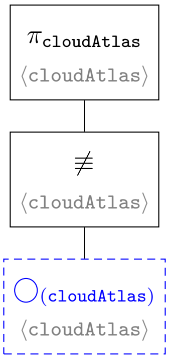
Query specification (query-13)

```
MATCH (cloudAtlas {title: "Cloud Atlas"})
RETURN cloudAtlas
```

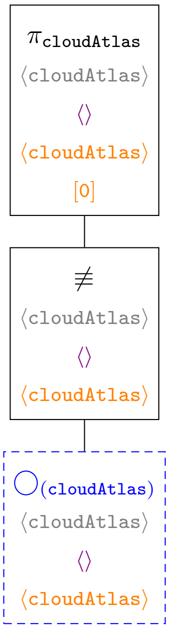
Relational algebra expression for search-based evaluation (query-13)

$$r = \pi_{\texttt{cloudAtlas}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{cloudAtlas})}\Big)\Big)$$

Relational algebra tree for search-based evaluation (query-13)



Incremental relational algebra tree (query-13)



E.1.14 query-14

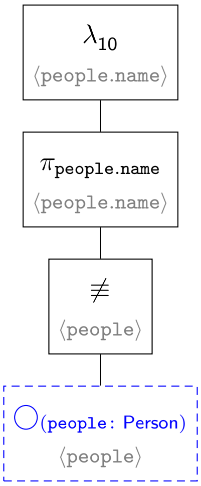
Query specification (query-14)

```
MATCH (people:Person)
RETURN people.name
LIMIT 10
```

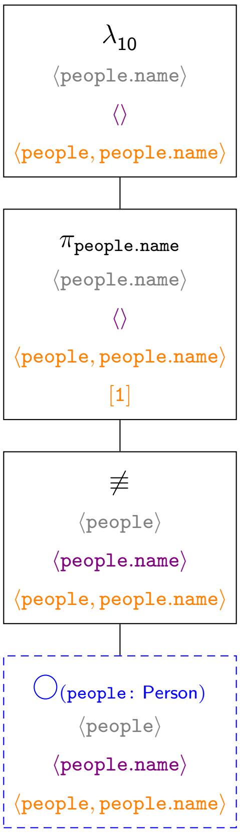
Relational algebra expression for search-based evaluation (query-14)

$$r = \lambda_{10}\Big(\pi_{\texttt{people.name}}\Big(\not\equiv\Big(\bigcirc_{(\texttt{people}:\,\mathsf{Person})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (query-14)



Incremental relational algebra tree (query-14)



E.1.15 query-15

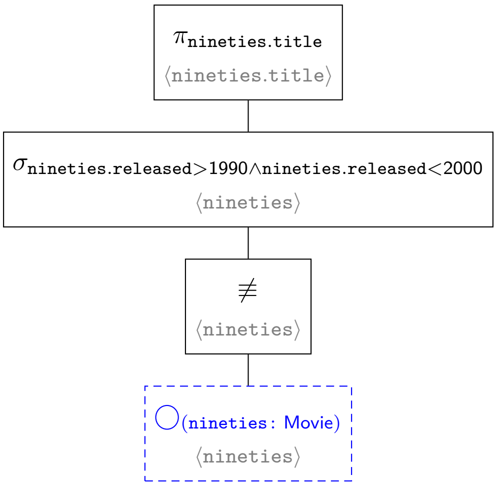
Query specification (query-15)

```
MATCH (nineties:Movie)
WHERE nineties.released > 1990 AND nineties.released < 2000
RETURN nineties.title
```

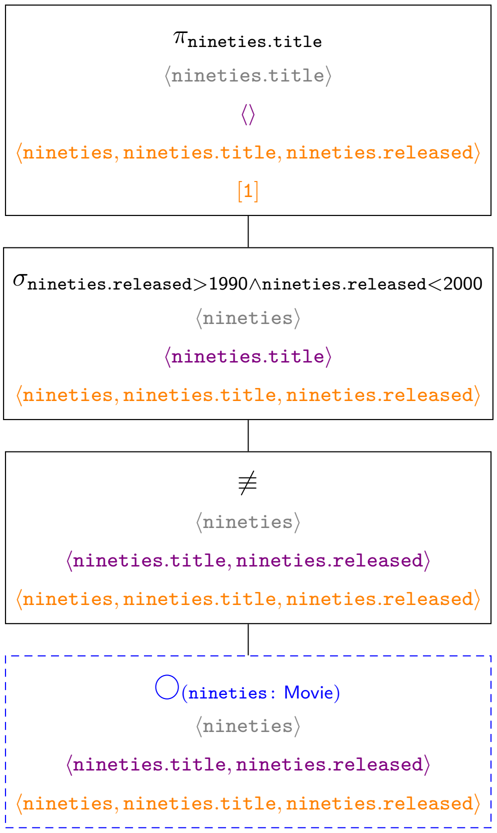
Relational algebra expression for search-based evaluation (query-15)

$$r = \pi_{\texttt{nineties.title}}\left(\sigma_{\texttt{nineties.released}>1990\wedge\texttt{nineties.released}<2000}\left(\not\equiv\left(\bigcirc_{(\texttt{nineties}:\ \textsf{Movie})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (query-15)


$\pi_{\texttt{nineties.title}}$
⟨nineties.title⟩
$\sigma_{\texttt{nineties.released}>1990\wedge\texttt{nineties.released}<2000}$
⟨nineties⟩
$\not\equiv$
⟨nineties⟩
$\bigcirc_{(\texttt{nineties}:\,\textsf{Movie})}$
⟨nineties⟩

Incremental relational algebra tree (query-15)


$\pi_{\texttt{nineties.title}}$
⟨nineties.title⟩
⟨⟩
⟨nineties, nineties.title, nineties.released⟩
[1]
$\sigma_{\texttt{nineties.released}>1990\wedge\texttt{nineties.released}<2000}$
⟨nineties⟩
⟨nineties.title⟩
⟨nineties, nineties.title, nineties.released⟩
$\not\equiv$
⟨nineties⟩
⟨nineties.title, nineties.released⟩
⟨nineties, nineties.title, nineties.released⟩
$\bigcirc_{(\texttt{nineties}:\,\textsf{Movie})}$
⟨nineties⟩
⟨nineties.title, nineties.released⟩
⟨nineties, nineties.title, nineties.released⟩

E.1.16 query-16

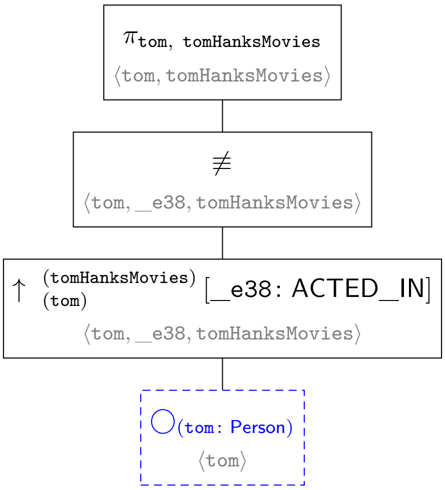
Query specification (query-16)

```
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(tomHanksMovies)
RETURN tom, tomHanksMovies
```

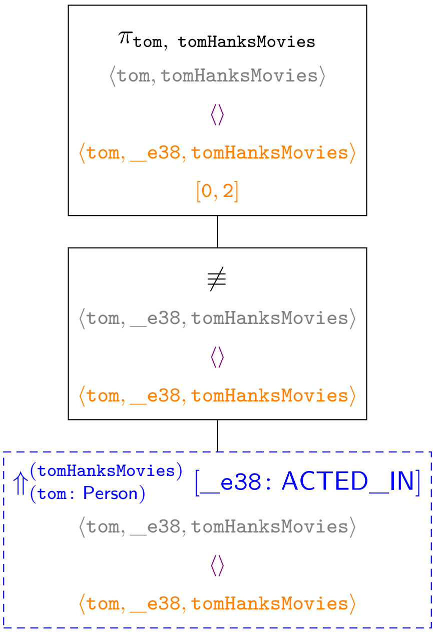
Relational algebra expression for search-based evaluation (query-16)

$$r = \pi_{\texttt{tom, tomHanksMovies}}\left(\not\equiv\left(\uparrow^{(\texttt{tomHanksMovies})}_{(\texttt{tom})}[\_e38\colon \textsf{ACTED\_IN}]\left(\bigcirc_{(\texttt{tom}:\,\textsf{Person})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (query-16)


π tom, tomHanksMovies
⟨tom, tomHanksMovies⟩
≢
⟨tom, _e38, tomHanksMovies⟩
↑ (tomHanksMovies) (tom) [_e38: ACTED_IN]
⟨tom, _e38, tomHanksMovies⟩
○(tom: Person)
⟨tom⟩

Incremental relational algebra tree (query-16)


π tom, tomHanksMovies
⟨tom, tomHanksMovies⟩
⟨⟩
⟨tom, _e38, tomHanksMovies⟩
[0, 2]
≢
⟨tom, _e38, tomHanksMovies⟩
⟨⟩
⟨tom, _e38, tomHanksMovies⟩
⇑ (tomHanksMovies) (tom: Person) [_e38: ACTED_IN]
⟨tom, _e38, tomHanksMovies⟩
⟨⟩
⟨tom, _e38, tomHanksMovies⟩

E.1.17 query-17

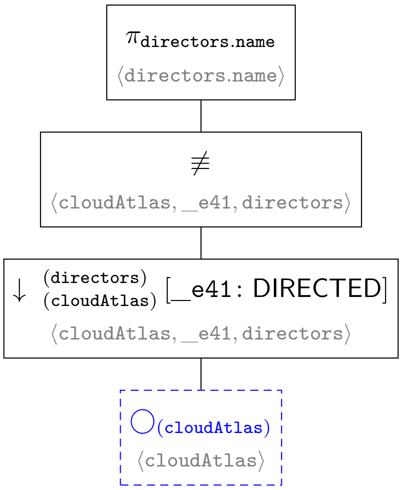
Query specification (query-17)

```
MATCH (cloudAtlas {title: "Cloud Atlas"})<-[:DIRECTED]-(directors)
RETURN directors.name
```

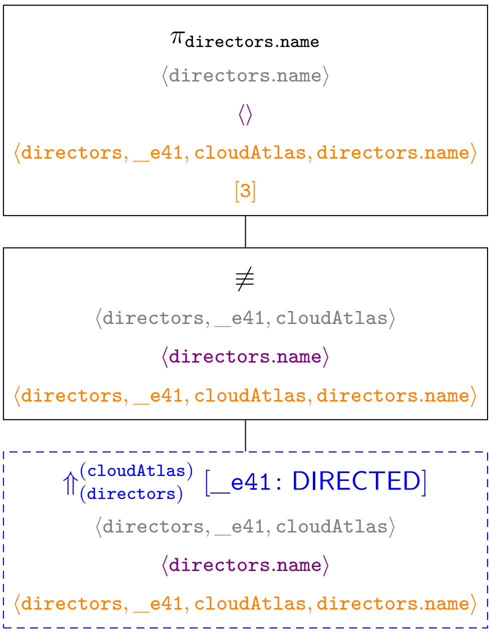
Relational algebra expression for search-based evaluation (query-17)

$$r = \pi_{\texttt{directors.name}}\left(\not\equiv\left(\downarrow^{\texttt{(directors)}}_{\texttt{(cloudAtlas)}}[\_\mathsf{e41}\colon \mathsf{DIRECTED}]\left(\bigcirc_{\texttt{(cloudAtlas)}}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (query-17)


π directors.name
⟨directors.name⟩
≢
⟨cloudAtlas, _e41, directors⟩
↓ (directors) (cloudAtlas) [_e41 : DIRECTED]
⟨cloudAtlas, _e41, directors⟩
○ (cloudAtlas)
⟨cloudAtlas⟩

Incremental relational algebra tree (query-17)


π directors.name
⟨directors.name⟩
⟨⟩
⟨directors, _e41, cloudAtlas, directors.name⟩
[3]
≢
⟨directors, _e41, cloudAtlas⟩
⟨directors.name⟩
⟨directors, _e41, cloudAtlas, directors.name⟩
⇑ (cloudAtlas) (directors) [_e41 : DIRECTED]
⟨directors, _e41, cloudAtlas⟩
⟨directors.name⟩
⟨directors, _e41, cloudAtlas, directors.name⟩

E.1.18 query-18

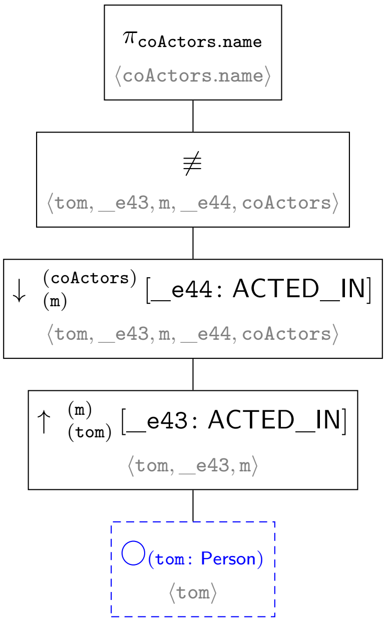
Query specification (query-18)

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors)
RETURN coActors.name
```

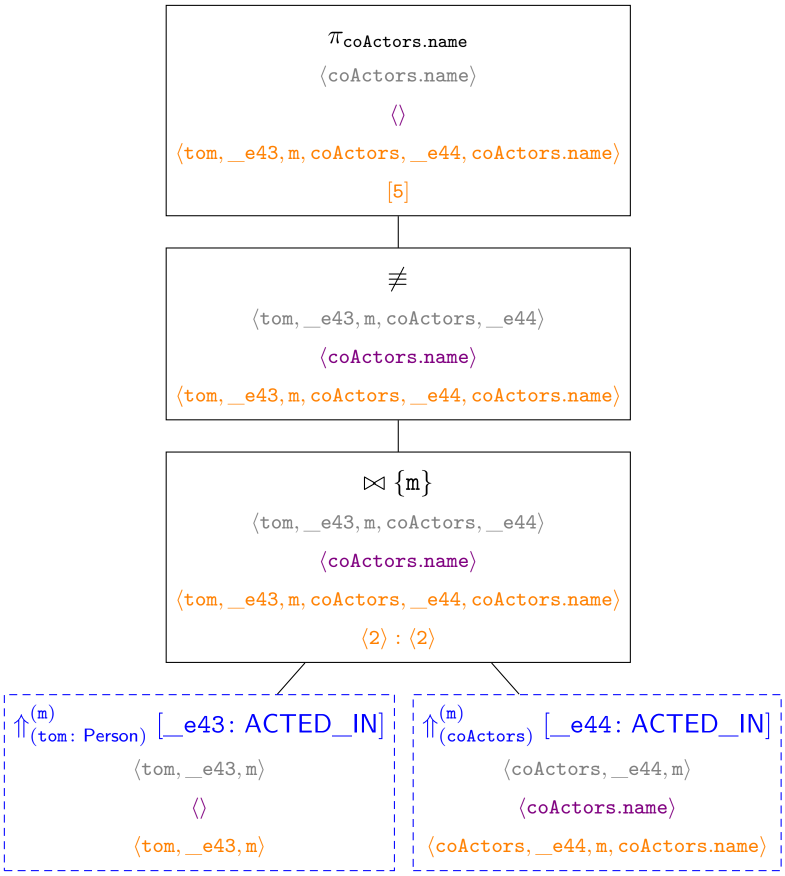
Relational algebra expression for search-based evaluation (query-18)

$$r = \pi_{\texttt{coActors.name}}\Big(\not\equiv\Big(\downarrow^{(\texttt{coActors})}_{(\texttt{m})}[\_e44\colon \mathsf{ACTED\_IN}]\Big(\uparrow^{(\texttt{m})}_{(\texttt{tom})}[\_e43\colon \mathsf{ACTED\_IN}]\Big(\bigcirc_{(\texttt{tom}\colon\ \mathsf{Person})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (query-18)


π coActors.name
⟨coActors.name⟩
≢
⟨tom, _e43, m, _e44, coActors⟩
↓ (coActors) (m) [_e44: ACTED_IN]
⟨tom, _e43, m, _e44, coActors⟩
↑ (m) (tom) [_e43: ACTED_IN]
⟨tom, _e43, m⟩
○(tom: Person)
⟨tom⟩

Incremental relational algebra tree (query-18)


π coActors.name
⟨coActors.name⟩
⟨⟩
⟨tom, _e43, m, coActors, _e44, coActors.name⟩
[5]
≢
⟨tom, _e43, m, coActors, _e44⟩
⟨coActors.name⟩
⟨tom, _e43, m, coActors, _e44, coActors.name⟩
⋈ {m}
⟨tom, _e43, m, coActors, _e44⟩
⟨coActors.name⟩
⟨tom, _e43, m, coActors, _e44, coActors.name⟩
⟨2⟩ : ⟨2⟩
⇑ (m) (tom: Person) [_e43: ACTED_IN]
⟨tom, _e43, m⟩
⟨⟩
⟨tom, _e43, m⟩
⇑ (m) (coActors) [_e44: ACTED_IN]
⟨coActors, _e44, m⟩
⟨coActors.name⟩
⟨coActors, _e44, m, coActors.name⟩

E.1.19 query-19

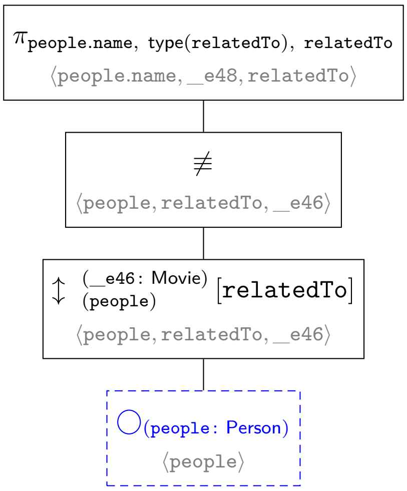
Query specification (query-19)

```
MATCH (people:Person)-[relatedTo]-(:Movie {title: "Cloud Atlas"})
RETURN people.name, Type(relatedTo), relatedTo
```

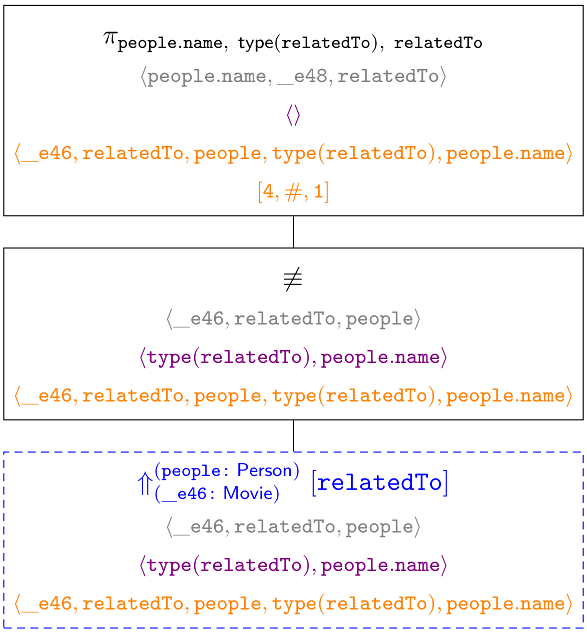
Relational algebra expression for search-based evaluation (query-19)

$$r = \pi_{\texttt{people.name},\ \texttt{type(relatedTo)},\ \texttt{relatedTo}}\left(\not\equiv\left(\updownarrow^{(\_\mathsf{e46}:\,\mathsf{Movie})}_{(\texttt{people})}[\texttt{relatedTo}]\left(\bigcirc_{(\mathsf{people}:\,\mathsf{Person})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (query-19)


$\pi_{\texttt{people.name},\ \texttt{type(relatedTo)},\ \texttt{relatedTo}}$
$\langle \texttt{people.name}, \_\texttt{e48}, \texttt{relatedTo} \rangle$
$\not\equiv$
$\langle \texttt{people}, \texttt{relatedTo}, \_\texttt{e46} \rangle$
$\updownarrow^{(\_\mathsf{e46}:\,\mathsf{Movie})}_{(\texttt{people})}[\texttt{relatedTo}]$
$\langle \texttt{people}, \texttt{relatedTo}, \_\texttt{e46} \rangle$
$\bigcirc_{(\mathsf{people}:\,\mathsf{Person})}$
$\langle \texttt{people} \rangle$

Incremental relational algebra tree (query-19)


$\pi_{\texttt{people.name},\ \texttt{type(relatedTo)},\ \texttt{relatedTo}}$
$\langle \texttt{people.name}, \_\texttt{e48}, \texttt{relatedTo} \rangle$
$\langle \rangle$
$\langle \_\texttt{e46}, \texttt{relatedTo}, \texttt{people}, \texttt{type(relatedTo)}, \texttt{people.name} \rangle$
$[4, \#, 1]$
$\not\equiv$
$\langle \_\texttt{e46}, \texttt{relatedTo}, \texttt{people} \rangle$
$\langle \texttt{type(relatedTo)}, \texttt{people.name} \rangle$
$\langle \_\texttt{e46}, \texttt{relatedTo}, \texttt{people}, \texttt{type(relatedTo)}, \texttt{people.name} \rangle$
$\Uparrow^{(\mathsf{people}:\,\mathsf{Person})}_{(\_\mathsf{e46}:\,\mathsf{Movie})}[\texttt{relatedTo}]$
$\langle \_\texttt{e46}, \texttt{relatedTo}, \texttt{people} \rangle$
$\langle \texttt{type(relatedTo)}, \texttt{people.name} \rangle$
$\langle \_\texttt{e46}, \texttt{relatedTo}, \texttt{people}, \texttt{type(relatedTo)}, \texttt{people.name} \rangle$

E.1.20 query-20

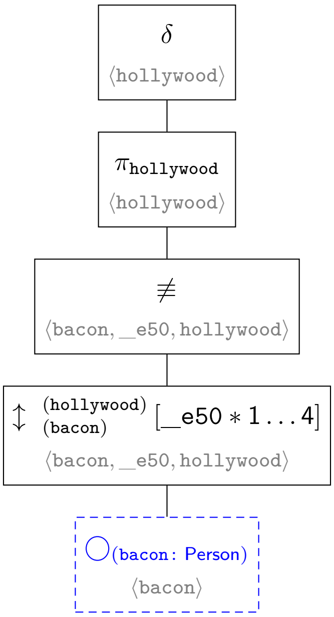
Query specification (query-20)

```
MATCH (bacon:Person {name:"Kevin Bacon"})-[*1..4]-(hollywood)
RETURN DISTINCT hollywood
```

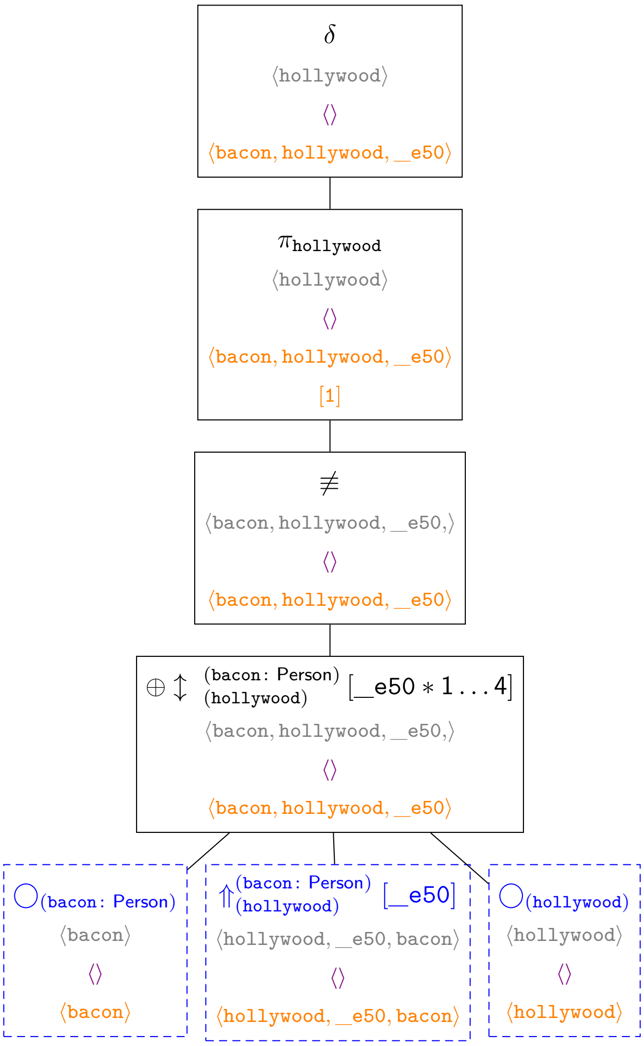
Relational algebra expression for search-based evaluation (query-20)

$$r = \delta\left(\pi_{\texttt{hollywood}}\left(\not\equiv\left(\updownarrow^{(\texttt{hollywood})}_{(\texttt{bacon})}[\_\mathsf{e50} * 1 \ldots 4]\left(\bigcirc_{(\mathsf{bacon}:\,\mathsf{Person})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (query-20)



Incremental relational algebra tree (query-20)



E.1.21 query-21

Query specification (query-21)

```
MATCH p=shortestPath(
(bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"})
)
RETURN p
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

E.1.22 query-22

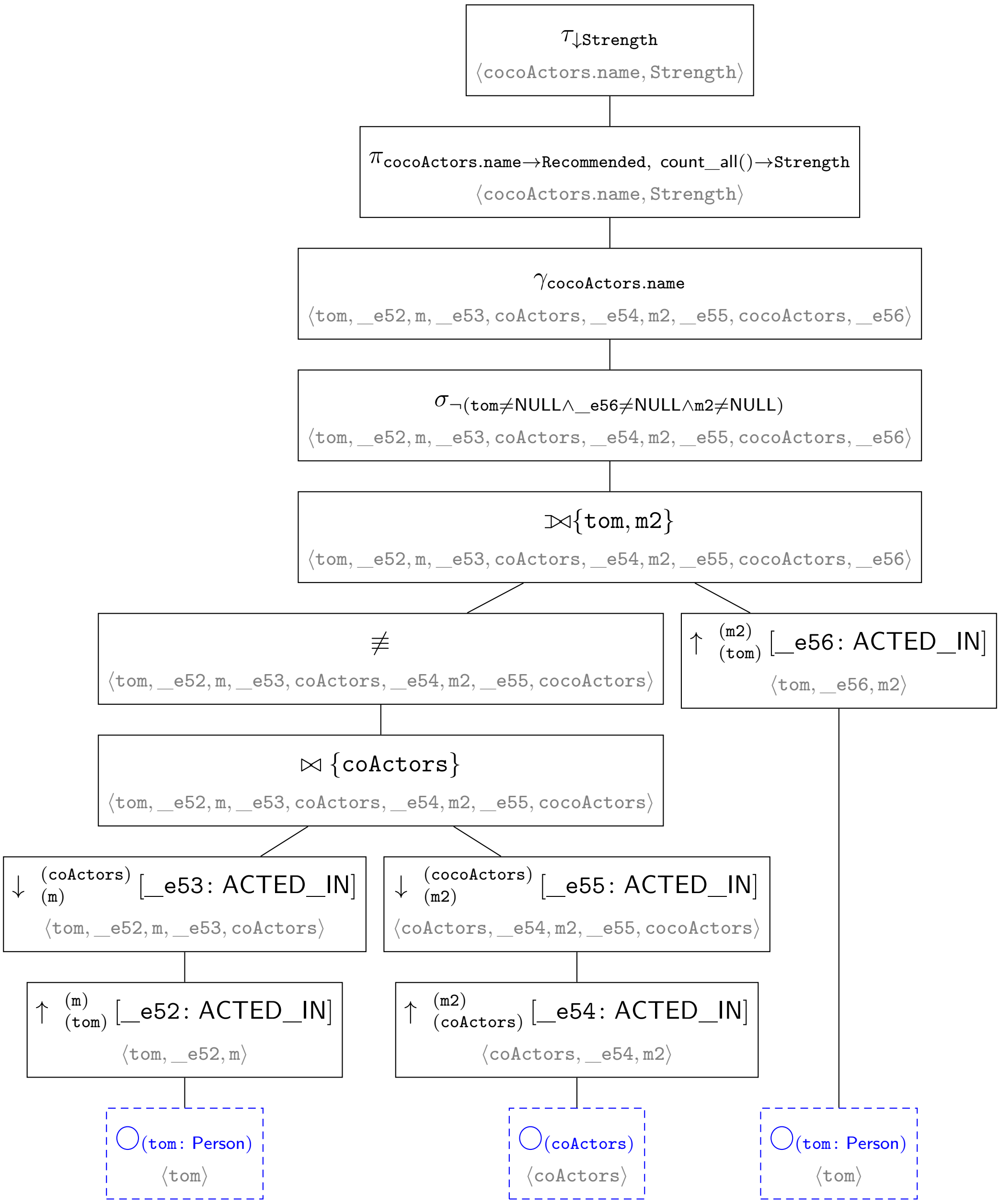
Query specification (query-22)

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors),
(coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cocoActors)
WHERE NOT (tom)-[:ACTED_IN]->(m2)
RETURN cocoActors.name AS Recommended, count(*) AS Strength ORDER BY Strength DESC
```

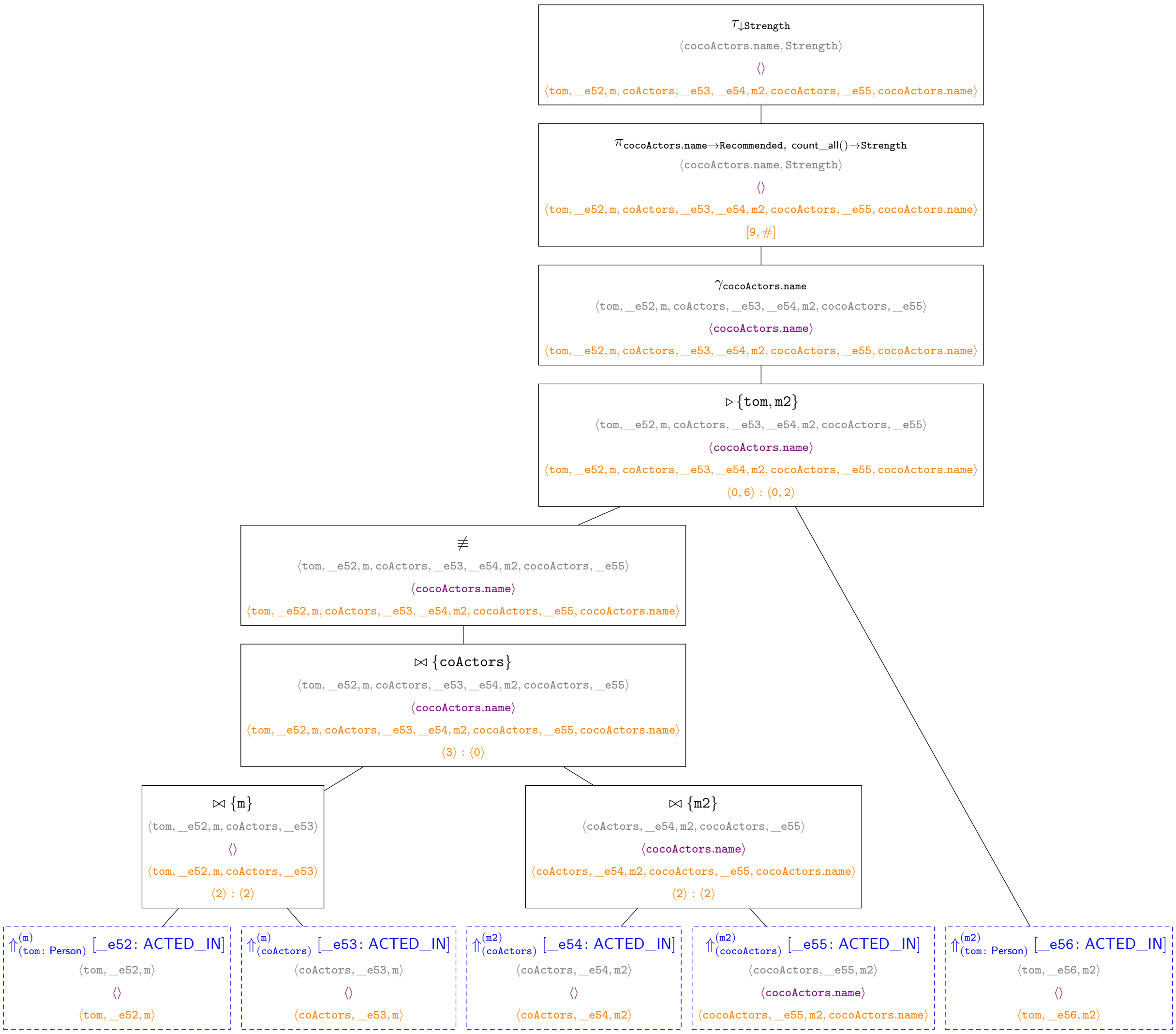
Relational algebra expression for search-based evaluation (query-22)

$$
\begin{aligned}
r = \tau_{\downarrow\texttt{Strength}}\Big(&\pi_{\texttt{cocoActors.name}\rightarrow\texttt{Recommended},\ \mathsf{count\_all}()\rightarrow\texttt{Strength}}\Big(\gamma_{\texttt{cocoActors.name}}\\
&\Big(\sigma_{\neg(\texttt{tom}\neq\mathsf{NULL}\wedge\mathsf{\_e56}\neq\mathsf{NULL}\wedge\texttt{m2}\neq\mathsf{NULL})}\Big(\not\equiv\Big(\downarrow\,\substack{\texttt{(coActors)}\\\texttt{(m)}}\,[\mathsf{\_e53\colon ACTED\_IN}]\\
&\Big(\uparrow\,\substack{\texttt{(m)}\\\texttt{(tom)}}\,[\mathsf{\_e52\colon ACTED\_IN}]\Big(\bigcirc_{(\texttt{tom}\colon\,\mathsf{Person})}\Big)\Big)\bowtie\downarrow\,\substack{\texttt{(cocoActors)}\\\texttt{(m2)}}\,[\mathsf{\_e55\colon ACTED\_IN}]\\
&\Big(\uparrow\,\substack{\texttt{(m2)}\\\texttt{(coActors)}}\,[\mathsf{\_e54\colon ACTED\_IN}]\Big(\bigcirc_{(\texttt{coActors})}\Big)\Big)\Big)⟕\uparrow\,\substack{\texttt{(m2)}\\\texttt{(tom)}}\,[\mathsf{\_e56\colon ACTED\_IN}]\\
&\Big(\bigcirc_{(\texttt{tom}\colon\,\mathsf{Person})}\Big)\Big)\Big)\Big)\Big)
\end{aligned}
$$

Relational algebra tree for search-based evaluation (query-22)


τ↓Strength
⟨cocoActors.name, Strength⟩
πcocoActors.name→Recommended, count_all()→Strength
⟨cocoActors.name, Strength⟩
γcocoActors.name
⟨tom, _e52, m, _e53, coActors, _e54, m2, _e55, cocoActors, _e56⟩
σ¬(tom≠NULL∧_e56≠NULL∧m2≠NULL)
⟨tom, _e52, m, _e53, coActors, _e54, m2, _e55, cocoActors, _e56⟩
⟕{tom, m2}
⟨tom, _e52, m, _e53, coActors, _e54, m2, _e55, cocoActors, _e56⟩
≢
⟨tom, _e52, m, _e53, coActors, _e54, m2, _e55, cocoActors⟩
↑ (m2) (tom) [_e56: ACTED_IN]
⟨tom, _e56, m2⟩
⋈ {coActors}
⟨tom, _e52, m, _e53, coActors, _e54, m2, _e55, cocoActors⟩
↓ (coActors) (m) [_e53: ACTED_IN]
⟨tom, _e52, m, _e53, coActors⟩
↓ (cocoActors) (m2) [_e55: ACTED_IN]
⟨coActors, _e54, m2, _e55, cocoActors⟩
↑ (m) (tom) [_e52: ACTED_IN]
⟨tom, _e52, m⟩
↑ (m2) (coActors) [_e54: ACTED_IN]
⟨coActors, _e54, m2⟩
○(tom: Person)
⟨tom⟩
○(coActors)
⟨coActors⟩
○(tom: Person)
⟨tom⟩

Incremental relational algebra tree (query-22)


τ↓Strength
⟨cocoActors.name, Strength⟩
⟨⟩
⟨tom, _e52, m, coActors, _e53, _e54, m2, cocoActors, _e55, cocoActors.name⟩
π cocoActors.name→Recommended, count_all()→Strength
⟨cocoActors.name, Strength⟩
⟨⟩
⟨tom, _e52, m, coActors, _e53, _e54, m2, cocoActors, _e55, cocoActors.name⟩
[9, #]
γ cocoActors.name
⟨tom, _e52, m, coActors, _e53, _e54, m2, cocoActors, _e55⟩
⟨cocoActors.name⟩
⟨tom, _e52, m, coActors, _e53, _e54, m2, cocoActors, _e55, cocoActors.name⟩
▷ {tom, m2}
⟨tom, _e52, m, coActors, _e53, _e54, m2, cocoActors, _e55⟩
⟨cocoActors.name⟩
⟨tom, _e52, m, coActors, _e53, _e54, m2, cocoActors, _e55, cocoActors.name⟩
⟨0, 6⟩ : ⟨0, 2⟩
≢
⟨tom, _e52, m, coActors, _e53, _e54, m2, cocoActors, _e55⟩
⟨cocoActors.name⟩
⟨tom, _e52, m, coActors, _e53, _e54, m2, cocoActors, _e55, cocoActors.name⟩
⋈ {coActors}
⟨tom, _e52, m, coActors, _e53, _e54, m2, cocoActors, _e55⟩
⟨cocoActors.name⟩
⟨tom, _e52, m, coActors, _e53, _e54, m2, cocoActors, _e55, cocoActors.name⟩
⟨3⟩ : ⟨0⟩
⋈ {m}
⟨tom, _e52, m, coActors, _e53⟩
⟨⟩
⟨tom, _e52, m, coActors, _e53⟩
⟨2⟩ : ⟨2⟩
⋈ {m2}
⟨coActors, _e54, m2, cocoActors, _e55⟩
⟨cocoActors.name⟩
⟨coActors, _e54, m2, cocoActors, _e55, cocoActors.name⟩
⟨2⟩ : ⟨2⟩
⇑(m)(tom: Person) [_e52: ACTED_IN]
⟨tom, _e52, m⟩
⟨⟩
⟨tom, _e52, m⟩
⇑(m)(coActors) [_e53: ACTED_IN]
⟨coActors, _e53, m⟩
⟨⟩
⟨coActors, _e53, m⟩
⇑(m2)(coActors) [_e54: ACTED_IN]
⟨coActors, _e54, m2⟩
⟨⟩
⟨coActors, _e54, m2⟩
⇑(m2)(cocoActors) [_e55: ACTED_IN]
⟨cocoActors, _e55, m2⟩
⟨cocoActors.name⟩
⟨cocoActors, _e55, m2, cocoActors.name⟩
⇑(m2)(tom: Person) [_e56: ACTED_IN]
⟨tom, _e56, m2⟩
⟨⟩
⟨tom, _e56, m2⟩

E.1.23 query-23

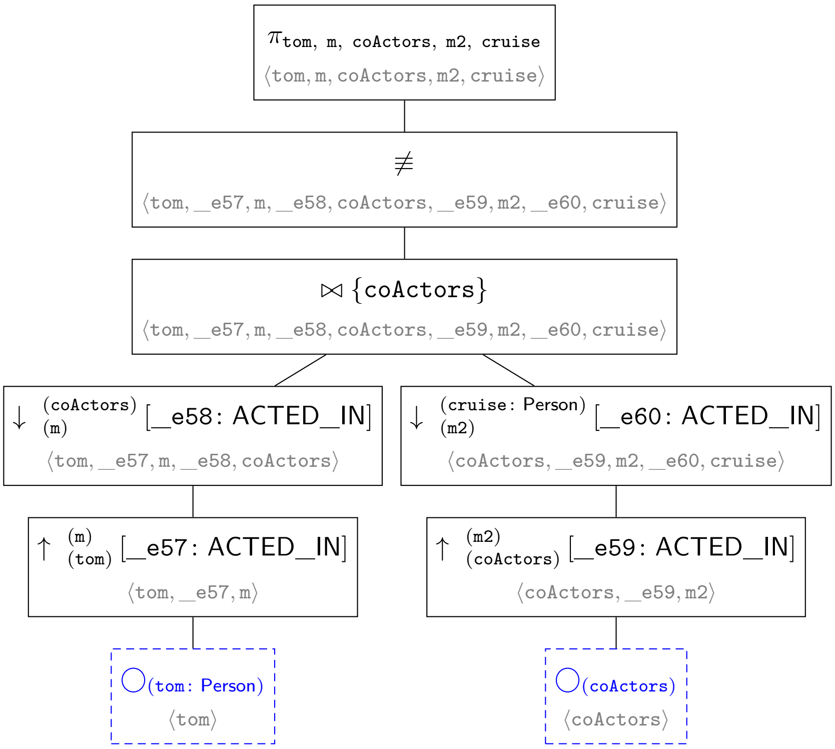
Query specification (query-23)

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors),
(coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cruise:Person {name:"Tom Cruise"})
RETURN tom, m, coActors, m2, cruise
```

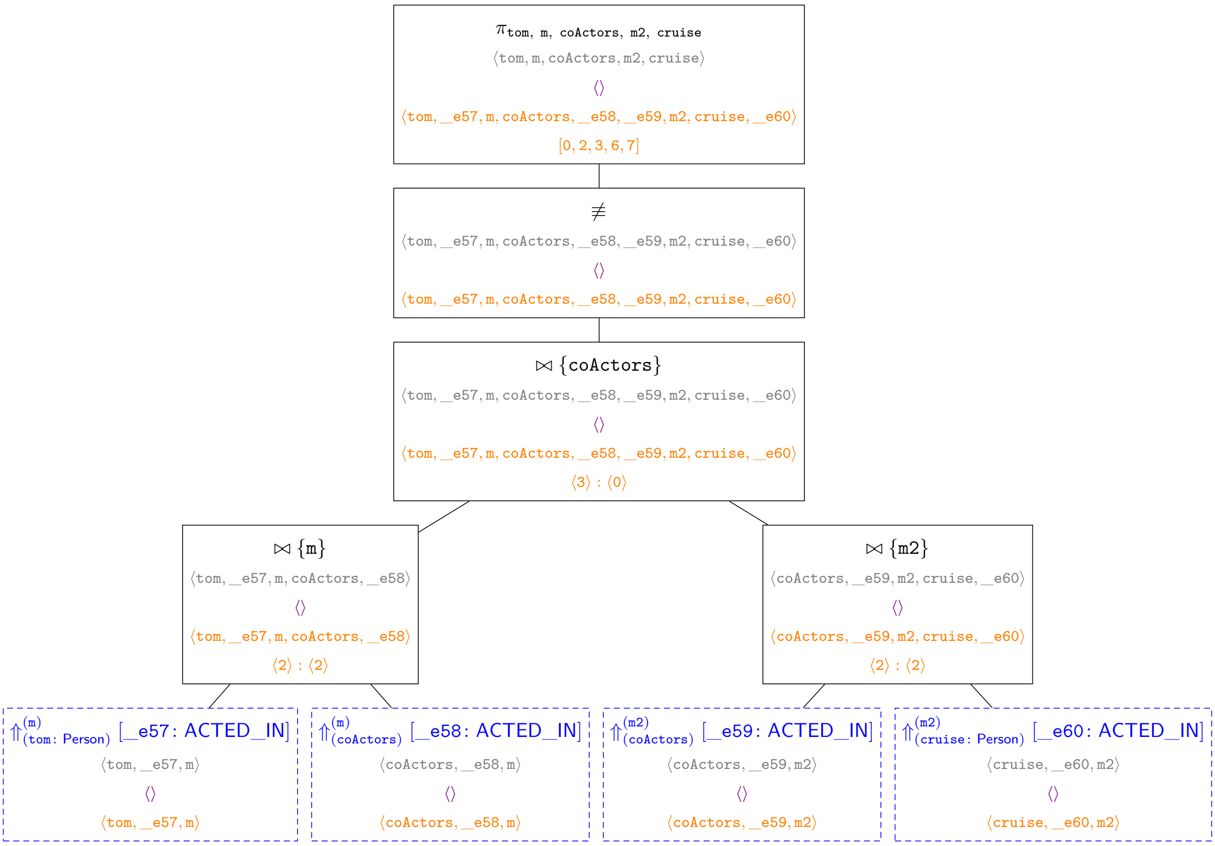
Relational algebra expression for search-based evaluation (query-23)

$$r = \pi_{\texttt{tom, m, coActors, m2, cruise}} \left(\not\equiv \left(\downarrow^{(\texttt{coActors})}_{(\texttt{m})} [\_\mathsf{e58}: \mathsf{ACTED\_IN}] \left(\uparrow^{(\texttt{m})}_{(\texttt{tom})} [\_\mathsf{e57}: \mathsf{ACTED\_IN}] \left(\bigcirc_{(\texttt{tom}:\ \mathsf{Person})} \right) \right) \bowtie\!\downarrow^{(\texttt{cruise}:\ \mathsf{Person})}_{(\texttt{m2})} [\_\mathsf{e60}: \mathsf{ACTED\_IN}] \left(\uparrow^{(\texttt{m2})}_{(\texttt{coActors})} [\_\mathsf{e59}: \mathsf{ACTED\_IN}] \left(\bigcirc_{(\texttt{coActors})} \right) \right) \right) \right)$$

Relational algebra tree for search-based evaluation (query-23)


π tom, m, coActors, m2, cruise
⟨tom, m, coActors, m2, cruise⟩
≠
⟨tom, _e57, m, _e58, coActors, _e59, m2, _e60, cruise⟩
⋈ {coActors}
⟨tom, _e57, m, _e58, coActors, _e59, m2, _e60, cruise⟩
↓ (coActors) (m) [_e58: ACTED_IN]
⟨tom, _e57, m, _e58, coActors⟩
↓ (cruise: Person) (m2) [_e60: ACTED_IN]
⟨coActors, _e59, m2, _e60, cruise⟩
↑ (m) (tom) [_e57: ACTED_IN]
⟨tom, _e57, m⟩
↑ (m2) (coActors) [_e59: ACTED_IN]
⟨coActors, _e59, m2⟩
○(tom: Person)
⟨tom⟩
○(coActors)
⟨coActors⟩

Incremental relational algebra tree (query-23)


π tom, m, coActors, m2, cruise
⟨tom, m, coActors, m2, cruise⟩
⟨⟩
⟨tom, _e57, m, coActors, _e58, _e59, m2, cruise, _e60⟩
[0, 2, 3, 6, 7]
≠
⟨tom, _e57, m, coActors, _e58, _e59, m2, cruise, _e60⟩
⟨⟩
⟨tom, _e57, m, coActors, _e58, _e59, m2, cruise, _e60⟩
⋈ {coActors}
⟨tom, _e57, m, coActors, _e58, _e59, m2, cruise, _e60⟩
⟨⟩
⟨tom, _e57, m, coActors, _e58, _e59, m2, cruise, _e60⟩
⟨3⟩ : ⟨0⟩
⋈ {m}
⟨tom, _e57, m, coActors, _e58⟩
⟨⟩
⟨tom, _e57, m, coActors, _e58⟩
⟨2⟩ : ⟨2⟩
⋈ {m2}
⟨coActors, _e59, m2, cruise, _e60⟩
⟨⟩
⟨coActors, _e59, m2, cruise, _e60⟩
⟨2⟩ : ⟨2⟩
⇑ (m) (tom: Person) [_e57: ACTED_IN]
⟨tom, _e57, m⟩
⟨⟩
⟨tom, _e57, m⟩
⇑ (m) (coActors) [_e58: ACTED_IN]
⟨coActors, _e58, m⟩
⟨⟩
⟨coActors, _e58, m⟩
⇑ (m2) (coActors) [_e59: ACTED_IN]
⟨coActors, _e59, m2⟩
⟨⟩
⟨coActors, _e59, m2⟩
⇑ (m2) (cruise: Person) [_e60: ACTED_IN]
⟨cruise, _e60, m2⟩
⟨⟩
⟨cruise, _e60, m2⟩

Appendix F

Static Analysis for Java

Total progress:

0 of 0

F.1 Queries

Progress:

0 of 0

Appendix G

Static Analysis for JavaScript

Total progress:

1 of 28

G.1 Queries

Progress:

1 of 28

G.1.1 BlockStatement

Query specification (BlockStatement)

```
MATCH
  (bs:BlockStatement)-[:block]->(b:Block)-[:statements]->(list:List),

  (bs)    -[:`_end`]->  (bsE:End),
  (list)  -[:`_end`]->  (listE:End)

MERGE
  (bs)    -[:`_normal`]-> (list)    -[:`_end`]->
  (listE) -[:`_normal`]-> (bsE)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.2 Boolean

Query specification (Boolean)

```
MATCH
  (lit:LiteralBooleanExpression),
  (ts:TypeSystem)-[:`_instance`]->(tag:Tag:`Boolean`)

MERGE
  (lit)-[:`_type`]->(tag)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.3 CallExpressionNoParam

Query specification (CallExpressionNoParam)

```
MATCH
  (call:CallExpression)-[:callee]->(:IdentifierExpression)
  <-[:node]-(:Reference)<-[:references]-(:Variable)
  -[:declarations]->(:Declaration)-[:node]->(:BindingIdentifier)
  <-[:name]-(fd:FunctionDeclaration),

  (call)    -[:`_end`]->  (callE:End),
  (fd)      -[:`_end`]->  (fdE:End)

WHERE
  NOT (call)-[:arguments]->()

MERGE
  (call)    -[:`_normal`]-> (fd)      -[:`_end`]->
  (fdE)     -[:`_normal`]-> (callE)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.4 CallExpressionParam

Query specification (CallExpressionParam)

```
MATCH
  (arguments:List) <-[:arguments]- (callExp:CallExpression) -[:callee]->
  (:IdentifierExpression) <-[:`node`]- (:Reference) <-[:references]-
  (:Variable) -[:declarations]-> (:Declaration) -[:`node`]->
  (:BindingIdentifier) <-[:name]- (fd:FunctionDeclaration) -[:params]->
  (:FormalParameters) -[:items]-> (params:List),

  (callExp)     -[:`_end`]->  (callExpE:`End`),
  (fd)          -[:`_end`]->  (fdE:`End`),
  (arguments)   -[:`_end`]->  (argumentsE:`End`)

MERGE
  (callExp)     -[:`_normal`]-> (arguments) -[:`_end`]->
  (argumentsE)  -[:`_normal`]-> (fd)        -[:`_end`]->
  (fdE)         -[:`_normal`]-> (callExpE)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.5 ExpressionStatement

Query specification (ExpressionStatement)

```
MATCH
  (es:ExpressionStatement)-[:expression]->(exp:Expression),

  (es)  -[:`_end`]->  (esE:End),
  (exp) -[:`_end`]->  (expE:End)

MERGE
  (es)    -[:`_normal`]-> (exp)   -[:`_end`]->
  (expE)  -[:`_normal`]-> (esE)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.6 FunctionDeclaration

Query specification (FunctionDeclaration)

```
MATCH
  (fd:FunctionDeclaration)-[:body]->(b:FunctionBody)-[:statements]->(list:List),

  (fd)    -[:`_end`]->  (fdE:End),
  (list)  -[:`_end`]->  (listE:End)

MERGE
  (fd)    -[:`_normal`]-> (list)    -[:`_end`]->
  (listE) -[:`_normal`]-> (fdE)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.7 IfStatementAlternate

Query specification (IfStatementAlternate)

```
MATCH
  (test:Node)<-[:test]-(if:IfStatement)-[:consequent]->(consequent:Node),
  (if)-[:alternate]->(alternate:Statement),

  (alternate)   -[:`_end`]->    (alternateE:End),

  (if)          -[:`_end`]->  (ifE:End),
  (test)        -[:`_end`]->  (testE:End),
  (consequent)  -[:`_end`]->  (consequentE:End)

MERGE
  (if)          -[:`_normal`]-> (test)        -[:`_end`]->
  (testE)       -[:`_true`]->   (consequent)  -[:`_end`]->
  (consequentE) -[:`_normal`]-> (ifE)

MERGE
  (testE)       -[:`_false`]->  (alternate)   -[:`_end`]->
  (alternateE)  -[:`_normal`]-> (ifE)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.8 IfStatementNoAlternate

Query specification (IfStatementNoAlternate)

```
MATCH
  (test:Node)<-[:test]-(if:IfStatement)-[:consequent]->(consequent:Node),

  (if)          -[:`_end`]->  (ifE:End),
  (test)        -[:`_end`]->  (testE:End),
  (consequent)  -[:`_end`]->  (consequentE:End)

WHERE
  NOT (if)-[:alternate]->(:Statement)

MERGE
  (if)          -[:`_normal`]-> (test)       -[:`_end`]->
  (testE)       -[:`_true`]->   (consequent) -[:`_end`]->
  (consequentE) -[:`_normal`]-> (ifE)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.9 Infinity

Query specification (Infinity)

```
MATCH
  (lit:LiteralInfinityExpression),
  (ts:TypeSystem)-[:`_instance`]->(tag:Tag:`Infinity`)

MERGE
  (lit)-[:`_type`]->(tag)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.10 ListNoItem

Query specification (ListNoItem)

```
MATCH
  (l:List),

  (l)     -[:`_end`]->  (lE:End)

WHERE
  NOT (l)-[:`0`]->()

MERGE
  (l)     -[:`_normal`]-> (lE)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.11 ListWithItem

Query specification (ListWithItem)

```
MATCH
  (l:List)-[:`0`]->(first),
  (l)-[:last]->(last),

  (l)     -[:`_end`]->  (lE:End),
  (last)  -[:`_end`]->  (lastE:End)

MERGE (l)     -[:`_normal`]-> (first)
MERGE (lastE) -[:`_normal`]-> (lE)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.12 LiteralX

Query specification (LiteralX)

```
MATCH (le:Literal)  -[:`_end`]->  (leE:End)
MERGE (le)    -[:`_normal`]-> (leE)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.13 LogicalOr

Query specification (LogicalOr)

```
MATCH
  (ltag:Tag)<-[:`_type`]-(left:Expression)
  <-[:left]-(exp:BinaryExpression)-[:right]->
  (right:Expression)-[:`_type`]->(rtag:Tag),

  (ts:TypeSystem)-[:`_instance`]->(btag:Tag:`Boolean`)

WHERE
  exp.operator = 'LogicalOr'

MERGE (exp)-[:`_type`]->(rtag)
MERGE (exp)-[:`_type`]->(ltag)
MERGE (exp)-[:`_type`]->(btag)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.14 Null

Query specification (Null)

```
MATCH
  (lit:LiteralNullExpression),
  (ts:TypeSystem)-[:`_instance`]->(tag:Tag:`Null`)

MERGE
  (lit)-[:`_type`]->(tag)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.15 Numeric

Query specification (Numeric)

```
MATCH
  (lit:LiteralNumericExpression),
  (ts:TypeSystem)-[:`_instance`]->(tag:Tag:`Number`)

MERGE
  (lit)-[:`_type`]->(tag)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.16 Read

Query specification (Read)

```
MATCH
  (v:Variable)-[:references]->(r:Reference)-[:node]->(ide:IdentifierExpression),
  (v)-[:`_type`]->(tag:Tag)

MERGE
  (ide)-[:`_type`]->(tag)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.17 RegExp

Query specification (RegExp)

```
MATCH
  (lit:LiteralRegExpExpression),
  (ts:TypeSystem)-[:`_instance`]->(tag:Tag:`RegExp`)

MERGE
  (lit)-[:`_type`]->(tag)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.18 String

Query specification (String)

```
MATCH
  (lit:LiteralStringExpression),
  (ts:TypeSystem)-[:`_instance`]->(tag:Tag:`String`)

MERGE
  (lit)-[:`_type`]->(tag)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.19 TypeSystem

Query specification (TypeSystem)

```
MERGE (ts:TypeSystem)

MERGE (ts)-[:`_instance`]->(:Tag:`Undefined`)
MERGE (ts)-[:`_instance`]->(:Tag:`Null`)
MERGE (ts)-[:`_instance`]->(:Tag:`Boolean`)
MERGE (ts)-[:`_instance`]->(:Tag:`Number`)
MERGE (ts)-[:`_instance`]->(:Tag:`String:`)
MERGE (ts)-[:`_instance`]->(:Tag:`Symbol`)
MERGE (ts)-[:`_instance`]->(:Tag:`Object`)
MERGE (ts)-[:`_instance`]->(:Tag:`Function`)
MERGE (ts)-[:`_instance`]->(:Tag:`Error`)

MERGE (ts)-[:`_instance`]->(:Tag:`Math`)
MERGE (ts)-[:`_instance`]->(:Tag:`Date`)
MERGE (ts)-[:`_instance`]->(:Tag:`RegExp`)
MERGE (ts)-[:`_instance`]->(:Tag:`Array`)
MERGE (ts)-[:`_instance`]->(:Tag:`Map`)
MERGE (ts)-[:`_instance`]->(:Tag:`Set`)
MERGE (ts)-[:`_instance`]->(:Tag:`JSON`)
MERGE (ts)-[:`_instance`]->(:Tag:`ArrayBuffer`)
MERGE (ts)-[:`_instance`]->(:Tag:`DataView`)
MERGE (ts)-[:`_instance`]->(:Tag:`Promise`)
MERGE (ts)-[:`_instance`]->(:Tag:`Proxy`)
MERGE (ts)-[:`_instance`]->(:Tag:`Reflect`)


MERGE (ts)-[:`_instance`]->(:Tag:`Infinity`)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.20 VariableDeclarationStatement

Query specification (VariableDeclarationStatement)

```
MATCH
  (vds:VariableDeclarationStatement)-[:declaration]->(vdion:VariableDeclaration)
  -[:declarators]->(vdor:VariableDeclarator)-[:init]->(exp:Expression),

  (vds)     -[:`_end`]->  (vdsE:End),
  (vdion)   -[:`_end`]->  (vdionE:End),
  (exp)     -[:`_end`]->  (expE:End)

MERGE (vdion)   -[:`_normal`]-> (vdionE)

MERGE
  (vds)     -[:`_normal`]-> (exp)   -[:`_end`]->
  (expE)    -[:`_normal`]-> (vdion) -[:`_end`]->
  (vdionE)  -[:`_normal`]-> (vdsE)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.21 VariableDeclarator

Query specification (VariableDeclarator)

```
MATCH
  (bi:BindingIdentifier)<-[:binding]-
    (vd:VariableDeclarator)
      -[:init]->(exp:Expression),

  (exp)-[:`_type`]->(tag:Tag)

MERGE
  (bi)-[:`_type`]->(tag)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.22 Write

Query specification (Write)

```
MATCH
  (v:Variable)-[:references]->(r:Reference)-[:node]->(bid:BindingIdentifier),
  (bid)-[:`_type`]->(tag:Tag)

MERGE
  (v)-[:`_type`]->(tag)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.23 generatecalls

Query specification (generatecalls)

```
MATCH
    // Match called FunctionDeclarations for every CallExpression
    (call:CallExpression)-[:callee]->(:IdentifierExpression)
    <-[:node]-(:Reference)<-[:references]-(:Variable)
    -[:declarations]->(:Declaration)-[:node]->(:BindingIdentifier)
    <-[:name]-(fd:FunctionDeclaration)
MATCH
    // List every call from a function body
    (fun:FunctionDeclaration), (call:CallExpression),
    p = shortestPath((fun)-[*]->(call))

MERGE
    // Create a calls relationship between the caller
    // FunctionDeclaration and the called FunctionDeclaration
    (fun)-[:calls]->(fd)
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.24 getlastcommithash

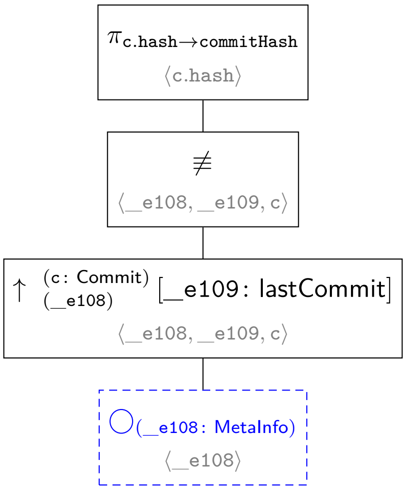
Query specification (getlastcommithash)

```
MATCH (:MetaInfo)-[:lastCommit]->(c:Commit)
RETURN c.hash as commitHash
```

Relational algebra expression for search-based evaluation (getlastcommithash)

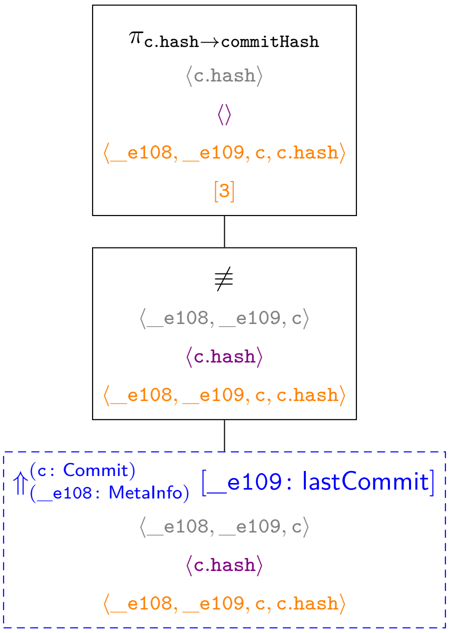
$$r = \pi_{\texttt{c.hash}\rightarrow\texttt{commitHash}}\left(\not\equiv\left(\uparrow^{(\mathsf{c}:\ \mathsf{Commit})}_{(\_\mathsf{e108})}[\_\mathsf{e109}\colon \mathsf{lastCommit}]\left(\bigcirc_{(\_\mathsf{e108}:\ \mathsf{MetaInfo})}\right)\right)\right)$$

Relational algebra tree for search-based evaluation (getlastcommithash)



$\pi_{\texttt{c.hash}\rightarrow\texttt{commitHash}}$
⟨c.hash⟩
$\not\equiv$
⟨_e108, _e109, c⟩
$\uparrow^{(\mathsf{c}:\ \mathsf{Commit})}_{(\_\mathsf{e108})}[\_\mathsf{e109}\colon \mathsf{lastCommit}]$
⟨_e108, _e109, c⟩
$\bigcirc_{(\_\mathsf{e108}:\ \mathsf{MetaInfo})}$
⟨_e108⟩

Incremental relational algebra tree (getlastcommithash)



$\pi_{\texttt{c.hash}\rightarrow\texttt{commitHash}}$
⟨c.hash⟩
⟨⟩
⟨_e108, _e109, c, c.hash⟩
[3]
$\not\equiv$
⟨_e108, _e109, c⟩
⟨c.hash⟩
⟨_e108, _e109, c, c.hash⟩
$\Uparrow^{(\mathsf{c}:\ \mathsf{Commit})}_{(\_\mathsf{e108}:\ \mathsf{MetaInfo})}[\_\mathsf{e109}\colon \mathsf{lastCommit}]$
⟨_e108, _e109, c⟩
⟨c.hash⟩
⟨_e108, _e109, c, c.hash⟩

G.1.25 removefile

Query specification (removefile)

```
MATCH
    ( cu:CompilationUnit
        {
            path: {path}
        }
    )-[:contains]-(el)
WHERE
  // iff the provided sessionid parameter is NULL, then delete the fix graph of
  // the CompilationUnit; else delete the temporal one with the given sessionid
     ( {sessionid} IS NULL AND NOT exists(cu.sessionid) )
  OR ( cu.sessionid = {sessionid} )
DETACH DELETE
    cu, el
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.26 setcommithash

Query specification (setcommithash)

```
MERGE (:MetaInfo)-[:lastCommit]->(c:Commit)
SET c.hash = {commithash}
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.27 typing

Query specification (typing)

```
/**
 * Type inferencing
 *
 * Repeat the following steps until either no change occurs in the database or a
 * given repeat limit is reached.
 *
 *  1. Find the literal values and assign type tags to them.
 *  2. Propagate the type information to the Variables in the given Scope for
 *     every VariableDeclaration with initial value. Handle the corner cases.
 *  3. Handle built-in Unary and Binary Expressions.
 *  4. Propagate type information into function calls taking care of in-function
 *     type differentiation based on input types.
 */

/**
 *  1. Find the literal values and assign type tags to them.
 *     - LiteralNumericExpression -> NumberTag
 *     - LiteralStringExpression  -> StringTag
 *     - LiteralBooleanExpression -> BooleanTag
 *     - LiteralNullExpression    -> NullTag
 */

// LiteralNumericExpression -> NumberTag
```

```
MATCH (exp:LiteralNumericExpression)
MERGE (exp)-[:type]->(tag:Tag:NumberTag)
SET tag.session = exp.session
;

// LiteralStringExpression  -> StringTag
MATCH (exp:LiteralStringExpression)
MERGE (exp)-[:type]->(tag:Tag:StringTag)
SET tag.session = exp.session
;

// LiteralBooleanExpression -> BooleanTag
MATCH (exp:LiteralBooleanExpression)
MERGE (exp)-[:type]->(tag:Tag:BooleanTag)
SET tag.session = exp.session
;

// LiteralNullExpression    -> NullTag
MATCH (exp:LiteralNullExpression)
MERGE (exp)-[:type]->(tag:Tag:NullTag)
SET tag.session = exp.session
;

/**
 *  2. Propagate the type information to Variables from their VariableDeclaration.
 */

MATCH
    //(vds:VariableDeclarationStatement)-[:declaration]->
    //(vdion:VariableDeclaration)-[declarators]->(vdor)
    (var:Variable)-[:declarations]->
    (dec:Declaration)-[:node]->
    (bi:BindingIdentifier)<-[:binding]-
    (vdor:VariableDeclarator)-[:init]->
    (exp:Expression)-[:type]->
    (type:Tag)
//MERGE
//    (var)-[:type]->(tag:Tag)
//CALL apoc.create.addLabels(tag, labels(type))
WHERE
    NOT (var)-[:type]-(:Tag)-[:from]->(type)
CALL
    apoc.refactor.cloneNodes([type]) YIELD input, output as tag, error
MERGE
    (var)-[:type]->(tag)-[:from]->(type)
//MERGE
//    (tag)-[:from]->(type)
MERGE
    (tag)-[:through]->(vdor)
;


/**
 * Find the VariableDeclarators without initial value and mark them as
 * undefined.
 */

 MATCH
     (var:Variable)-[:declarations]->
     (dec:Declaration)-[:node]->
     (bi:BindingIdentifier)<-[:binding]-
     (vdor:VariableDeclarator)
 WHERE
```

```
     NOT (vdor)-[:init]->(:Expression)
 MERGE
     (var)-[:type]->(tag:Tag:UndefinedTag)
 SET
     tag.session = var.session
 MERGE
     (tag)-[:through]->(vdor)
 ;

/**
 * Find ArrayExpressions and tag them.
 */
 // TODO

/**
 * Find ObjectExpressions and tag them.
 */
 // TODO


/**
 *  3. Handle built-in Unary and Binary Expressions.
 */

/**
 * Propagate the Variable type information to the appropriate
 * IdentifierExpression.
 */

MATCH
    (type:Tag)<-[:type]-
    (var:Variable)-[:references]->
    (ref:Reference)-[:node]->
    (exp:Expression)
WHERE
    NOT (exp)-[:type]-(:Tag)-[:from]->(type)
CALL
    apoc.refactor.cloneNodes([type]) YIELD input, output as tag, error
MERGE
    (exp)-[:type]->(tag)-[:from]->(type)
;

/**
 * PLUS binary operator.
 */
MATCH
    (ltag:Tag)<-[:type]-(left:Expression)
    <-[:left]-(exp:BinaryExpression)-[:right]->
    (right:Expression)-[:type]->(rtag:Tag)
WHERE
        exp.operator = 'Plus'
    AND ltag :NumberTag
    AND rtag :NumberTag
MERGE
    (exp)-[:type]->(tag:Tag:NumberTag)
MERGE
    (ltag)<-[:from]-(tag)-[:from]->(rtag)
SET
    tag.session = ltag.session
;
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

G.1.28 unusedfunctions

Query specification (unusedfunctions)

```
/**
 * Get not used FunctionDeclarations
 */
MATCH
    // Find the exported FunctionDeclaration that may be an entrance point
    p = (main)-[:items]->(:ExportDeclaration)-[:declaration]->(fd:FunctionDeclaration)

MATCH
    // Find every FunctionDeclaration that should be available through the
    // entrance points
    q = (dead:FunctionDeclaration)-[:location]->(span:SourceSpan),
        (start:SourceLocation)<-[:start]-(span)-[:end]->(end:SourceLocation)

WHERE
    // List the ones that are not available (Kleene closure) from the
    // entrance nodes (thus are not the entrance nodes "<>").
        ( NOT (fd)-[:calls*]->(dead) )
    AND ( dead <> fd )
    AND ( main:Script OR main:Module )
    AND ( ALL (
                x in (nodes(p) + nodes(q))
                WHERE NOT exists(x.session) OR x.session = {sessionid}
          ) )

RETURN DISTINCT
    ID(dead) as id, start.line, start.column, end.line, end.column, dead.session
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

Appendix H

Train Benchmark

Total progress:

9 of 10

H.1 Queries

Progress:

9 of 10

H.1.1 ActiveRoute

Query specification (ActiveRoute)

```
MATCH (inactiveRoute:Route)-[:follows]->(swP:SwitchPosition)-[:target]->(sw:Switch)
WHERE swP.position <> sw.currentPosition
WITH collect(inactiveRoute) AS inactiveRoutes

MATCH (activeRoute:Route)
WHERE NOT activeRoute IN inactiveRoutes

RETURN activeRoute
```

Cannot parse query

Cannot parse query. This is probably a limitation in our current parser and not an error in the query specification.

H.1.2 ConnectedSegments

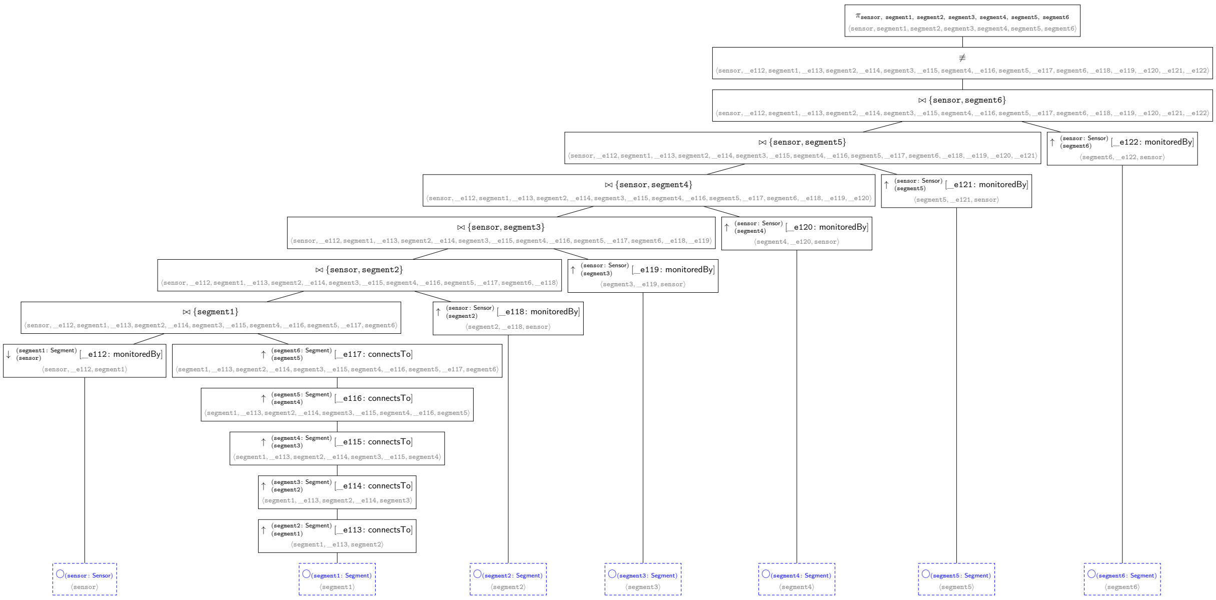
Query specification (ConnectedSegments)

```
MATCH
(sensor:Sensor)<-[:monitoredBy]-(segment1:Segment),
(segment1:Segment)-[:connectsTo]->
(segment2:Segment)-[:connectsTo]->
(segment3:Segment)-[:connectsTo]->
(segment4:Segment)-[:connectsTo]->
(segment5:Segment)-[:connectsTo]->(segment6:Segment),
(segment2:Segment)-[:monitoredBy]->(sensor:Sensor),
(segment3:Segment)-[:monitoredBy]->(sensor:Sensor),
(segment4:Segment)-[:monitoredBy]->(sensor:Sensor),
(segment5:Segment)-[:monitoredBy]->(sensor:Sensor),
(segment6:Segment)-[:monitoredBy]->(sensor:Sensor)
RETURN sensor, segment1, segment2, segment3, segment4, segment5, segment6
```

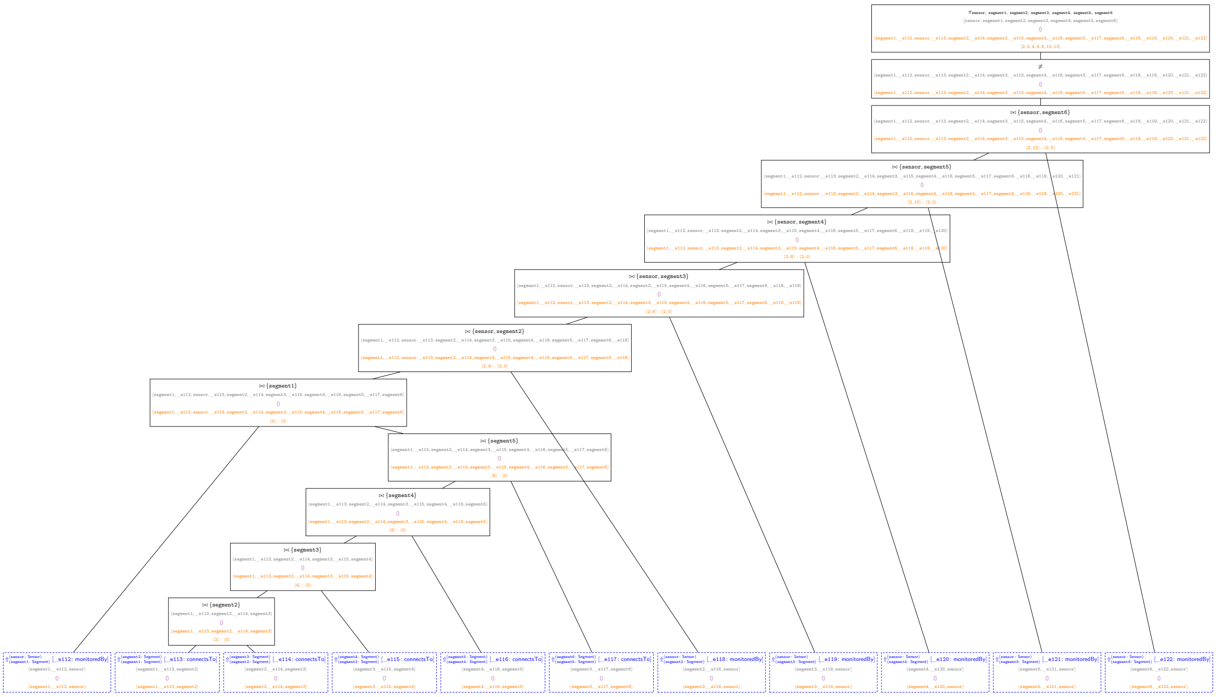
Relational algebra expression for search-based evaluation (ConnectedSegments)

$$
\begin{aligned}
r = \pi_{\texttt{sensor, segment1, segment2, segment3, segment4, segment5, segment6}} \Bigg(& \not\equiv \\
& \Bigg(\downarrow^{(\texttt{segment1}:\,\mathsf{Segment})}_{(\texttt{sensor})} [\mathsf{\_e112}:\,\mathsf{monitoredBy}] \\
& \Bigg(\bigcirc_{(\texttt{sensor}:\,\mathsf{Sensor})} \Bigg) \bowtie \\
& \uparrow^{(\texttt{segment6}:\,\mathsf{Segment})}_{(\texttt{segment5})} [\mathsf{\_e117}:\,\mathsf{connectsTo}] \\
& \Bigg(\uparrow^{(\texttt{segment5}:\,\mathsf{Segment})}_{(\texttt{segment4})} [\mathsf{\_e116}:\,\mathsf{connectsTo}] \\
& \Bigg(\uparrow^{(\texttt{segment4}:\,\mathsf{Segment})}_{(\texttt{segment3})} [\mathsf{\_e115}:\,\mathsf{connectsTo}] \\
& \Bigg(\uparrow^{(\texttt{segment3}:\,\mathsf{Segment})}_{(\texttt{segment2})} [\mathsf{\_e114}:\,\mathsf{connectsTo}] \\
& \Bigg(\uparrow^{(\texttt{segment2}:\,\mathsf{Segment})}_{(\texttt{segment1})} [\mathsf{\_e113}:\,\mathsf{connectsTo}] \\
& \Bigg(\bigcirc_{(\texttt{segment1}:\,\mathsf{Segment})} \Bigg)\Bigg)\Bigg)\Bigg)\Bigg) \bowtie \\
& \uparrow^{(\texttt{sensor}:\,\mathsf{Sensor})}_{(\texttt{segment2})} [\mathsf{\_e118}:\,\mathsf{monitoredBy}] \\
& \Bigg(\bigcirc_{(\texttt{segment2}:\,\mathsf{Segment})} \Bigg) \bowtie \\
& \uparrow^{(\texttt{sensor}:\,\mathsf{Sensor})}_{(\texttt{segment3})} [\mathsf{\_e119}:\,\mathsf{monitoredBy}] \\
& \Bigg(\bigcirc_{(\texttt{segment3}:\,\mathsf{Segment})} \Bigg) \bowtie \\
& \uparrow^{(\texttt{sensor}:\,\mathsf{Sensor})}_{(\texttt{segment4})} [\mathsf{\_e120}:\,\mathsf{monitoredBy}] \\
& \Bigg(\bigcirc_{(\texttt{segment4}:\,\mathsf{Segment})} \Bigg) \bowtie \\
& \uparrow^{(\texttt{sensor}:\,\mathsf{Sensor})}_{(\texttt{segment5})} [\mathsf{\_e121}:\,\mathsf{monitoredBy}] \\
& \Bigg(\bigcirc_{(\texttt{segment5}:\,\mathsf{Segment})} \Bigg) \bowtie \\
& \uparrow^{(\texttt{sensor}:\,\mathsf{Sensor})}_{(\texttt{segment6})} [\mathsf{\_e122}:\,\mathsf{monitoredBy}] \\
& \Bigg(\bigcirc_{(\texttt{segment6}:\,\mathsf{Segment})} \Bigg)\Bigg)\Bigg)
\end{aligned}
$$

Relational algebra tree for search-based evaluation (ConnectedSegments)


π sensor, segment1, segment2, segment3, segment4, segment5, segment6
⟨sensor, segment1, segment2, segment3, segment4, segment5, segment6⟩
≢
⟨sensor, _e112, segment1, _e113, segment2, _e114, segment3, _e115, segment4, _e116, segment5, _e117, segment6, _e118, _e119, _e120, _e121, _e122⟩
⋈ {sensor, segment6}
⟨sensor, _e112, segment1, _e113, segment2, _e114, segment3, _e115, segment4, _e116, segment5, _e117, segment6, _e118, _e119, _e120, _e121, _e122⟩
⋈ {sensor, segment5}
⟨sensor, _e112, segment1, _e113, segment2, _e114, segment3, _e115, segment4, _e116, segment5, _e117, segment6, _e118, _e119, _e120, _e121⟩
↑ (sensor: Sensor) (segment6) [_e122: monitoredBy]
⟨segment6, _e122, sensor⟩
⋈ {sensor, segment4}
⟨sensor, _e112, segment1, _e113, segment2, _e114, segment3, _e115, segment4, _e116, segment5, _e117, segment6, _e118, _e119, _e120⟩
↑ (sensor: Sensor) (segment5) [_e121: monitoredBy]
⟨segment5, _e121, sensor⟩
⋈ {sensor, segment3}
⟨sensor, _e112, segment1, _e113, segment2, _e114, segment3, _e115, segment4, _e116, segment5, _e117, segment6, _e118, _e119⟩
↑ (sensor: Sensor) (segment4) [_e120: monitoredBy]
⟨segment4, _e120, sensor⟩
⋈ {sensor, segment2}
⟨sensor, _e112, segment1, _e113, segment2, _e114, segment3, _e115, segment4, _e116, segment5, _e117, segment6, _e118⟩
↑ (sensor: Sensor) (segment3) [_e119: monitoredBy]
⟨segment3, _e119, sensor⟩
⋈ {segment1}
⟨sensor, _e112, segment1, _e113, segment2, _e114, segment3, _e115, segment4, _e116, segment5, _e117, segment6⟩
↑ (sensor: Sensor) (segment2) [_e118: monitoredBy]
⟨segment2, _e118, sensor⟩
↓ (segment1: Segment) (sensor) [_e112: monitoredBy]
⟨sensor, _e112, segment1⟩
↑ (segment6: Segment) (segment5) [_e117: connectsTo]
⟨segment1, _e113, segment2, _e114, segment3, _e115, segment4, _e116, segment5, _e117, segment6⟩
↑ (segment5: Segment) (segment4) [_e116: connectsTo]
⟨segment1, _e113, segment2, _e114, segment3, _e115, segment4, _e116, segment5⟩
↑ (segment4: Segment) (segment3) [_e115: connectsTo]
⟨segment1, _e113, segment2, _e114, segment3, _e115, segment4⟩
↑ (segment3: Segment) (segment2) [_e114: connectsTo]
⟨segment1, _e113, segment2, _e114, segment3⟩
↑ (segment2: Segment) (segment1) [_e113: connectsTo]
⟨segment1, _e113, segment2⟩
○(sensor: Sensor)
⟨sensor⟩
○(segment1: Segment)
⟨segment1⟩
○(segment2: Segment)
⟨segment2⟩
○(segment3: Segment)
⟨segment3⟩
○(segment4: Segment)
⟨segment4⟩
○(segment5: Segment)
⟨segment5⟩
○(segment6: Segment)
⟨segment6⟩

Incremental relational algebra tree (ConnectedSegments)



H.1.3 PosLength-simple-return

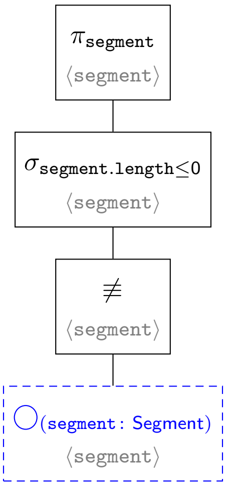
Query specification (PosLength-simple-return)

```
MATCH (segment:Segment)
WHERE segment.length <= 0
RETURN segment
```

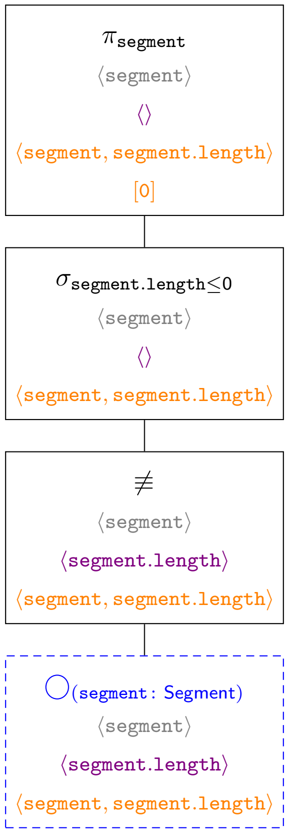
Relational algebra expression for search-based evaluation (PosLength-simple-return)

$$r = \pi_{\mathtt{segment}}\Big(\sigma_{\mathtt{segment.length}\leq 0}\Big(\not\equiv\Big(\bigcirc_{(\mathtt{segment}:\ \mathsf{Segment})}\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (PosLength-simple-return)



Incremental relational algebra tree (PosLength-simple-return)


$\pi_{\texttt{segment}}$
⟨segment⟩
⟨⟩
⟨segment, segment.length⟩
[0]
$\sigma_{\texttt{segment.length}\leq 0}$
⟨segment⟩
⟨⟩
⟨segment, segment.length⟩
$\not\equiv$
⟨segment⟩
⟨segment.length⟩
⟨segment, segment.length⟩
$\bigcirc_{(\texttt{segment}:\ \mathsf{Segment})}$
⟨segment⟩
⟨segment.length⟩
⟨segment, segment.length⟩

H.1.4 PosLength

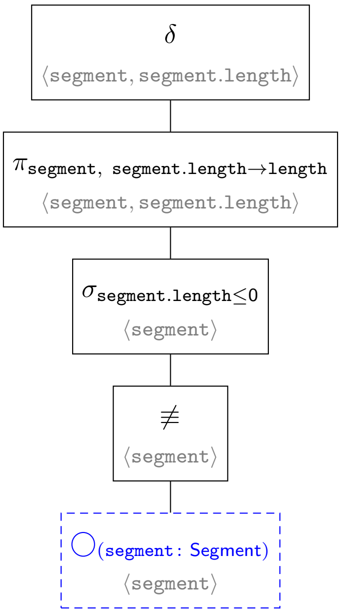
Query specification (PosLength)

```
MATCH (segment:Segment)
WHERE segment.length <= 0
RETURN DISTINCT segment, segment.length AS length
```

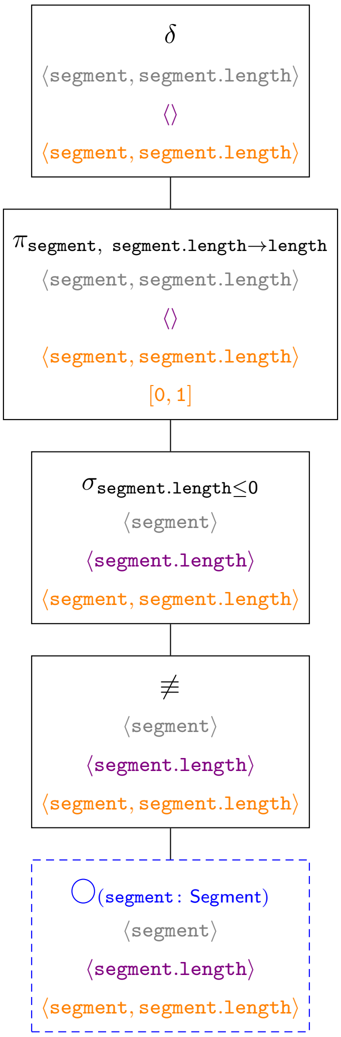
Relational algebra expression for search-based evaluation (PosLength)

$$r = \delta\left(\pi_{\texttt{segment},\ \texttt{segment.length}\rightarrow\texttt{length}}\left(\sigma_{\texttt{segment.length}\leq 0}\left(\not\equiv\left(\bigcirc_{(\texttt{segment}:\ \mathsf{Segment})}\right)\right)\right)\right)$$

Relational algebra tree for search-based evaluation (PosLength)


δ
⟨segment, segment.length⟩
π segment, segment.length→length
⟨segment, segment.length⟩
σ segment.length≤0
⟨segment⟩
≢
⟨segment⟩
◯(segment : Segment)
⟨segment⟩

Incremental relational algebra tree (PosLength)


δ
⟨segment, segment.length⟩
⟨⟩
⟨segment, segment.length⟩
π segment, segment.length→length
⟨segment, segment.length⟩
⟨⟩
⟨segment, segment.length⟩
[0, 1]
σ segment.length≤0
⟨segment⟩
⟨segment.length⟩
⟨segment, segment.length⟩
≢
⟨segment⟩
⟨segment.length⟩
⟨segment, segment.length⟩
◯(segment : Segment)
⟨segment⟩
⟨segment.length⟩
⟨segment, segment.length⟩

H.1.5 RouteSensor

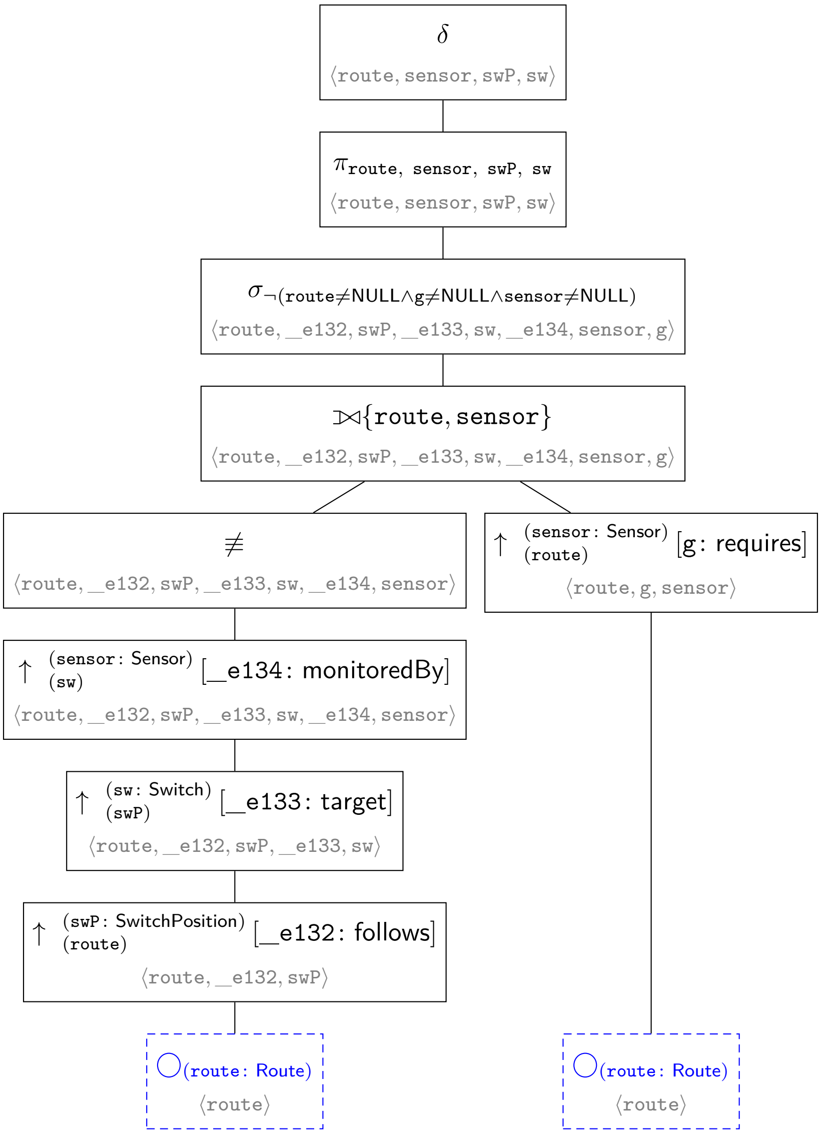
Query specification (RouteSensor)

```
MATCH (route:Route)-[:follows]->(swP:SwitchPosition)-[:target]->(sw:Switch)-[:monitoredBy]->(
    sensor:Sensor)
WHERE NOT ((route)-[g:requires]->(sensor))
RETURN DISTINCT route, sensor, swP, sw
```

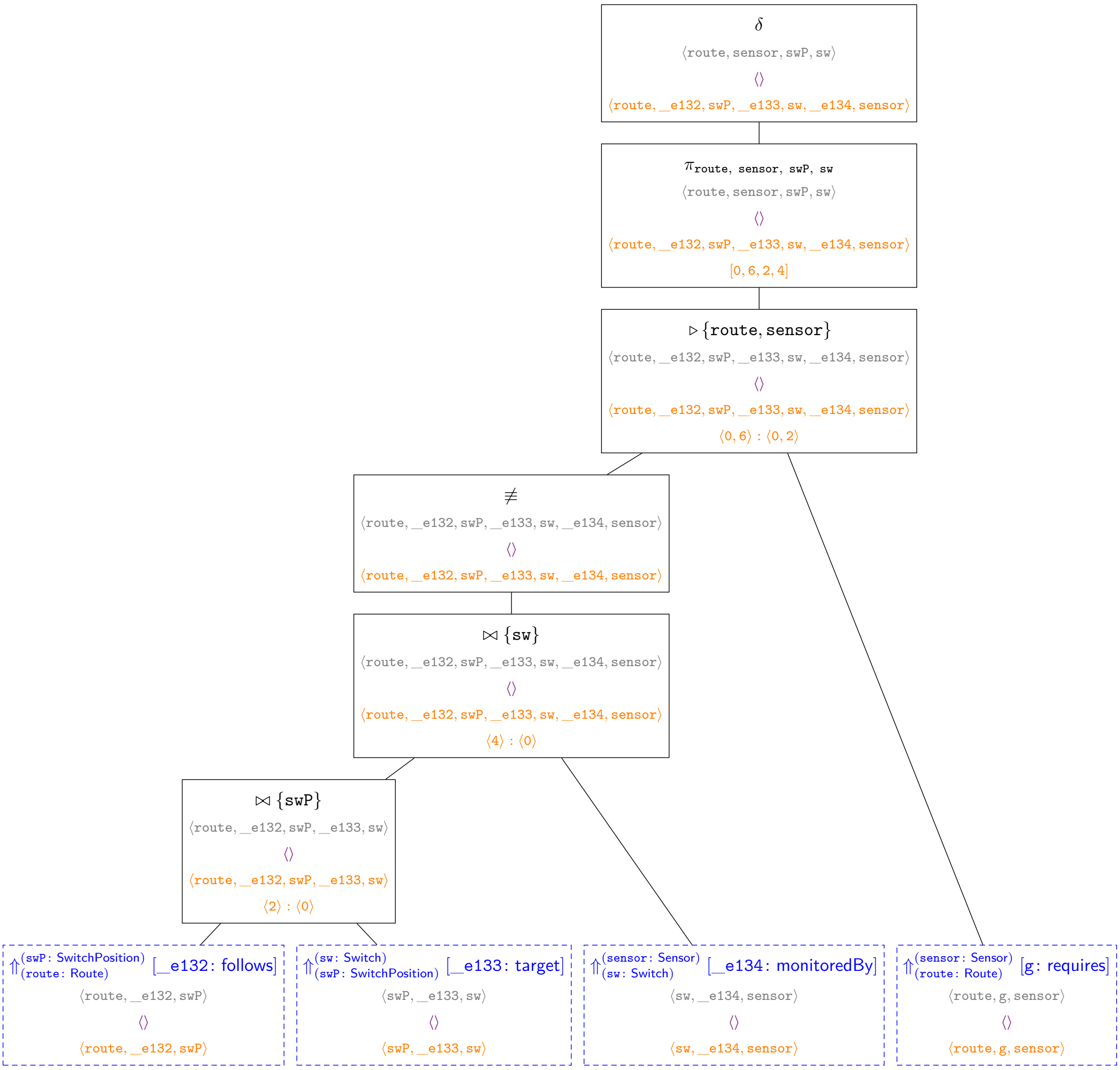
Relational algebra expression for search-based evaluation (RouteSensor)

$$
\begin{aligned}
r = \delta\Bigg(&\pi_{\texttt{route, sensor, swP, sw}}\Bigg(\sigma_{\neg(\texttt{route}\neq\mathsf{NULL}\wedge\texttt{g}\neq\mathsf{NULL}\wedge\texttt{sensor}\neq\mathsf{NULL})}\Bigg(\not\equiv\\
&\Bigg(\uparrow^{(\texttt{sensor}:\ \mathsf{Sensor})}_{(\texttt{sw})}[\_\mathsf{e134}:\ \mathsf{monitoredBy}]\Bigg(\uparrow^{(\texttt{sw}:\ \mathsf{Switch})}_{(\texttt{swP})}[\_\mathsf{e133}:\ \mathsf{target}]\\
&\Bigg(\uparrow^{(\texttt{swP}:\ \mathsf{SwitchPosition})}_{(\texttt{route})}[\_\mathsf{e132}:\ \mathsf{follows}]\Bigg(\bigcirc_{(\texttt{route}:\ \mathsf{Route})}\Bigg)\Bigg)\Bigg)\Bigg)\rhd\uparrow^{(\texttt{sensor}:\ \mathsf{Sensor})}_{(\texttt{route})}[\mathsf{g}:\ \mathsf{requires}]\\
&\Bigg(\bigcirc_{(\texttt{route}:\ \mathsf{Route})}\Bigg)\Bigg)\Bigg)\Bigg)
\end{aligned}
$$

Relational algebra tree for search-based evaluation (RouteSensor)


δ
⟨route, sensor, swP, sw⟩
π route, sensor, swP, sw
⟨route, sensor, swP, sw⟩
σ¬(route≠NULL∧g≠NULL∧sensor≠NULL)
⟨route, _e132, swP, _e133, sw, _e134, sensor, g⟩
▷{route, sensor}
⟨route, _e132, swP, _e133, sw, _e134, sensor, g⟩
≢
⟨route, _e132, swP, _e133, sw, _e134, sensor⟩
↑ (sensor: Sensor) (route) [g: requires]
⟨route, g, sensor⟩
↑ (sensor: Sensor) (sw) [_e134: monitoredBy]
⟨route, _e132, swP, _e133, sw, _e134, sensor⟩
↑ (sw: Switch) (swP) [_e133: target]
⟨route, _e132, swP, _e133, sw⟩
↑ (swP: SwitchPosition) (route) [_e132: follows]
⟨route, _e132, swP⟩
○(route: Route)
⟨route⟩
○(route: Route)
⟨route⟩

Incremental relational algebra tree (RouteSensor)


δ
⟨route, sensor, swP, sw⟩
⟨⟩
⟨route, _e132, swP, _e133, sw, _e134, sensor⟩
π route, sensor, swP, sw
⟨route, sensor, swP, sw⟩
⟨⟩
⟨route, _e132, swP, _e133, sw, _e134, sensor⟩
[0, 6, 2, 4]
▷ {route, sensor}
⟨route, _e132, swP, _e133, sw, _e134, sensor⟩
⟨⟩
⟨route, _e132, swP, _e133, sw, _e134, sensor⟩
⟨0, 6⟩ : ⟨0, 2⟩
≢
⟨route, _e132, swP, _e133, sw, _e134, sensor⟩
⟨⟩
⟨route, _e132, swP, _e133, sw, _e134, sensor⟩
⋈ {sw}
⟨route, _e132, swP, _e133, sw, _e134, sensor⟩
⟨⟩
⟨route, _e132, swP, _e133, sw, _e134, sensor⟩
⟨4⟩ : ⟨0⟩
⋈ {swP}
⟨route, _e132, swP, _e133, sw⟩
⟨⟩
⟨route, _e132, swP, _e133, sw⟩
⟨2⟩ : ⟨0⟩
⇑(swP: SwitchPosition) (route: Route) [_e132: follows]
⟨route, _e132, swP⟩
⟨⟩
⟨route, _e132, swP⟩
⇑(sw: Switch) (swP: SwitchPosition) [_e133: target]
⟨swP, _e133, sw⟩
⟨⟩
⟨swP, _e133, sw⟩
⇑(sensor: Sensor) (sw: Switch) [_e134: monitoredBy]
⟨sw, _e134, sensor⟩
⟨⟩
⟨sw, _e134, sensor⟩
⇑(sensor: Sensor) (route: Route) [g: requires]
⟨route, g, sensor⟩
⟨⟩
⟨route, g, sensor⟩

H.1.6 RouteSensorPositive

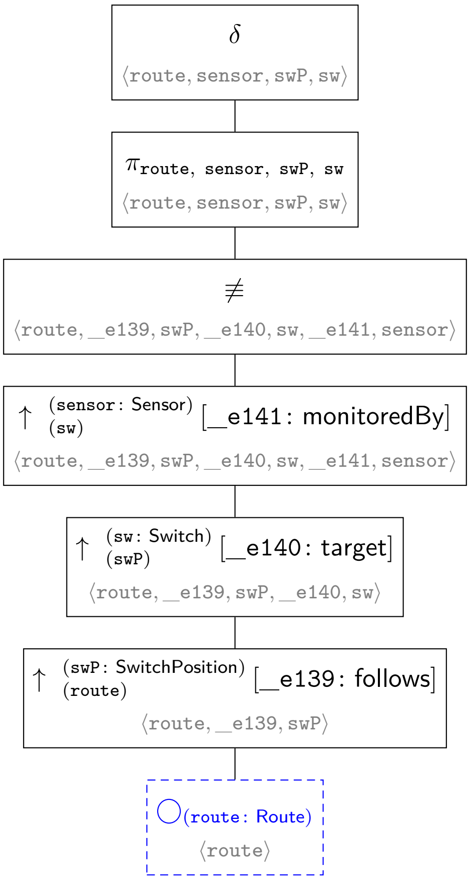
Query specification (RouteSensorPositive)

```
MATCH (route:Route)-[:follows]->(swP:SwitchPosition)-[:target]->(sw:Switch)-[:monitoredBy]->(
    sensor:Sensor)
RETURN DISTINCT route, sensor, swP, sw
```

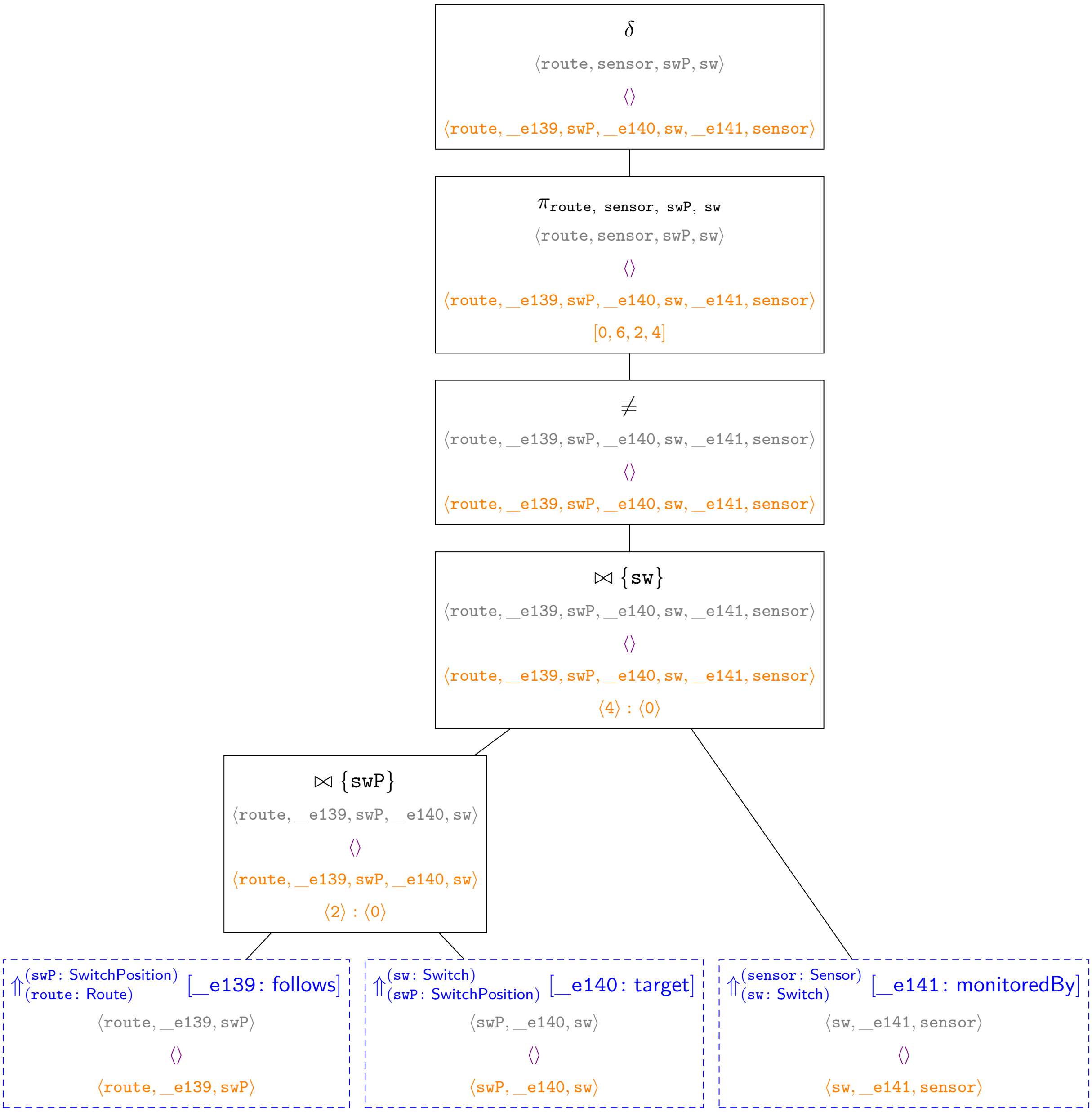
Relational algebra expression for search-based evaluation (RouteSensorPositive)

$$r = \delta\Big(\pi_{\texttt{route, sensor, swP, sw}}\Big(\not\equiv\Big(\uparrow^{(\texttt{sensor}\colon\textsf{Sensor})}_{(\texttt{sw})}[\_\textsf{e141}\colon\textsf{monitoredBy}]\Big(\uparrow^{(\texttt{sw}\colon\textsf{Switch})}_{(\texttt{swP})}[\_\textsf{e140}\colon\textsf{target}]\Big(\uparrow^{(\texttt{swP}\colon\textsf{SwitchPosition})}_{(\texttt{route})}[\_\textsf{e139}\colon\textsf{follows}]\Big(\bigcirc_{(\texttt{route}\colon\textsf{Route})}\Big)\Big)\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (RouteSensorPositive)


δ
⟨route, sensor, swP, sw⟩
π route, sensor, swP, sw
⟨route, sensor, swP, sw⟩
≢
⟨route, _e139, swP, _e140, sw, _e141, sensor⟩
↑ (sensor: Sensor) (sw) [_e141: monitoredBy]
⟨route, _e139, swP, _e140, sw, _e141, sensor⟩
↑ (sw: Switch) (swP) [_e140: target]
⟨route, _e139, swP, _e140, sw⟩
↑ (swP: SwitchPosition) (route) [_e139: follows]
⟨route, _e139, swP⟩
○(route: Route)
⟨route⟩

Incremental relational algebra tree (RouteSensorPositive)


δ
⟨route, sensor, swP, sw⟩
⟨⟩
⟨route, _e139, swP, _e140, sw, _e141, sensor⟩
π route, sensor, swP, sw
⟨route, sensor, swP, sw⟩
⟨⟩
⟨route, _e139, swP, _e140, sw, _e141, sensor⟩
[0, 6, 2, 4]
≠
⟨route, _e139, swP, _e140, sw, _e141, sensor⟩
⟨⟩
⟨route, _e139, swP, _e140, sw, _e141, sensor⟩
⋈ {sw}
⟨route, _e139, swP, _e140, sw, _e141, sensor⟩
⟨⟩
⟨route, _e139, swP, _e140, sw, _e141, sensor⟩
⟨4⟩ : ⟨0⟩
⋈ {swP}
⟨route, _e139, swP, _e140, sw⟩
⟨⟩
⟨route, _e139, swP, _e140, sw⟩
⟨2⟩ : ⟨0⟩
⇑(swP: SwitchPosition)(route: Route) [_e139: follows]
⟨route, _e139, swP⟩
⟨⟩
⟨route, _e139, swP⟩
⇑(sw: Switch)(swP: SwitchPosition) [_e140: target]
⟨swP, _e140, sw⟩
⟨⟩
⟨swP, _e140, sw⟩
⇑(sensor: Sensor)(sw: Switch) [_e141: monitoredBy]
⟨sw, _e141, sensor⟩
⟨⟩
⟨sw, _e141, sensor⟩

H.1.7 SemaphoreNeighbor

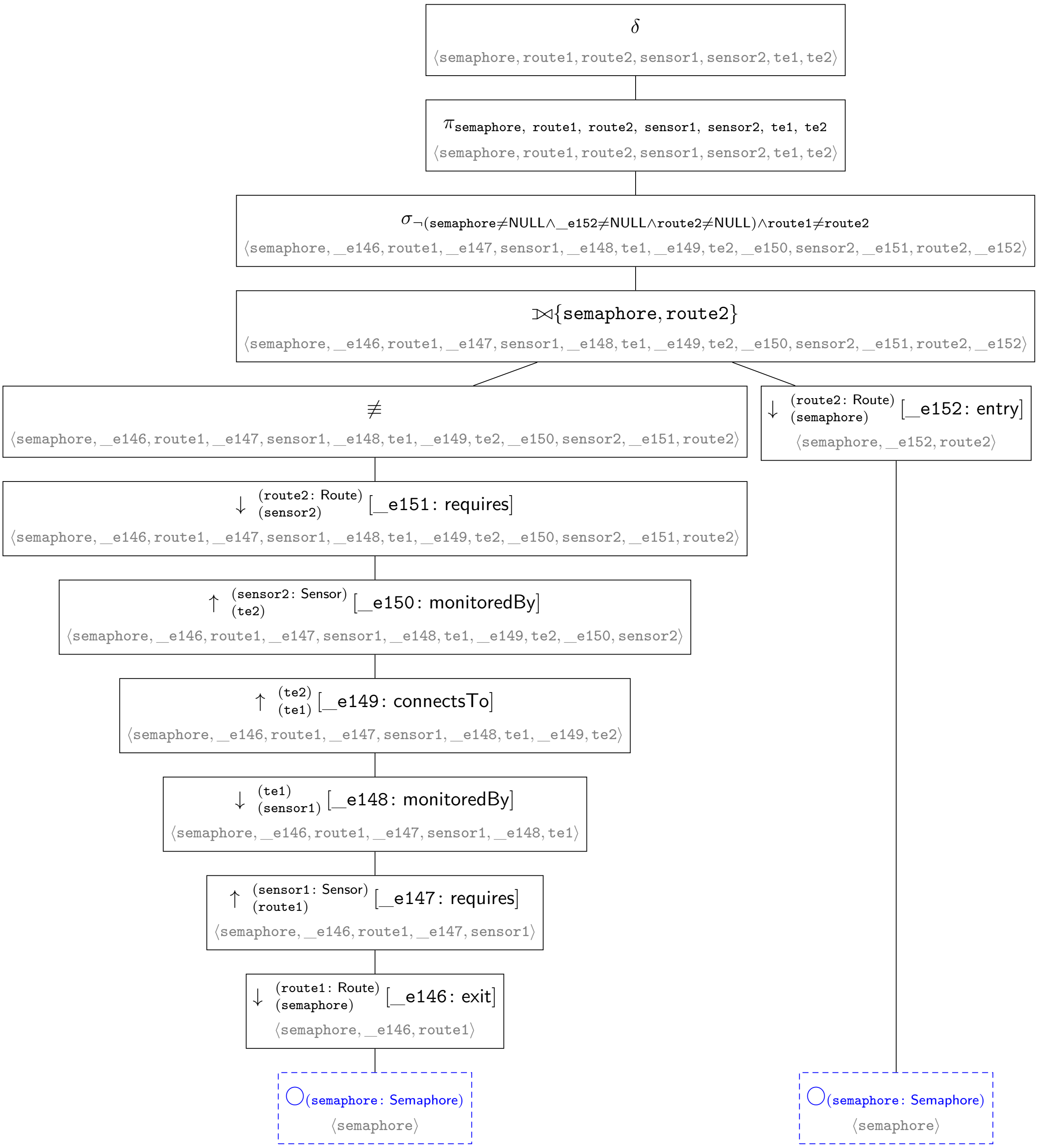
Query specification (SemaphoreNeighbor)

```
MATCH (semaphore:Semaphore)<-[:exit]-(route1:Route)-[:requires]->(sensor1:Sensor)<-[:monitoredBy
    ]-(te1)-[:connectsTo]->(te2)-[:monitoredBy]->(sensor2:Sensor)<-[:requires]-(route2:Route)
WHERE NOT ((semaphore)<-[:entry]-(route2))
      AND route1 != route2
RETURN DISTINCT semaphore, route1, route2, sensor1, sensor2, te1, te2
```

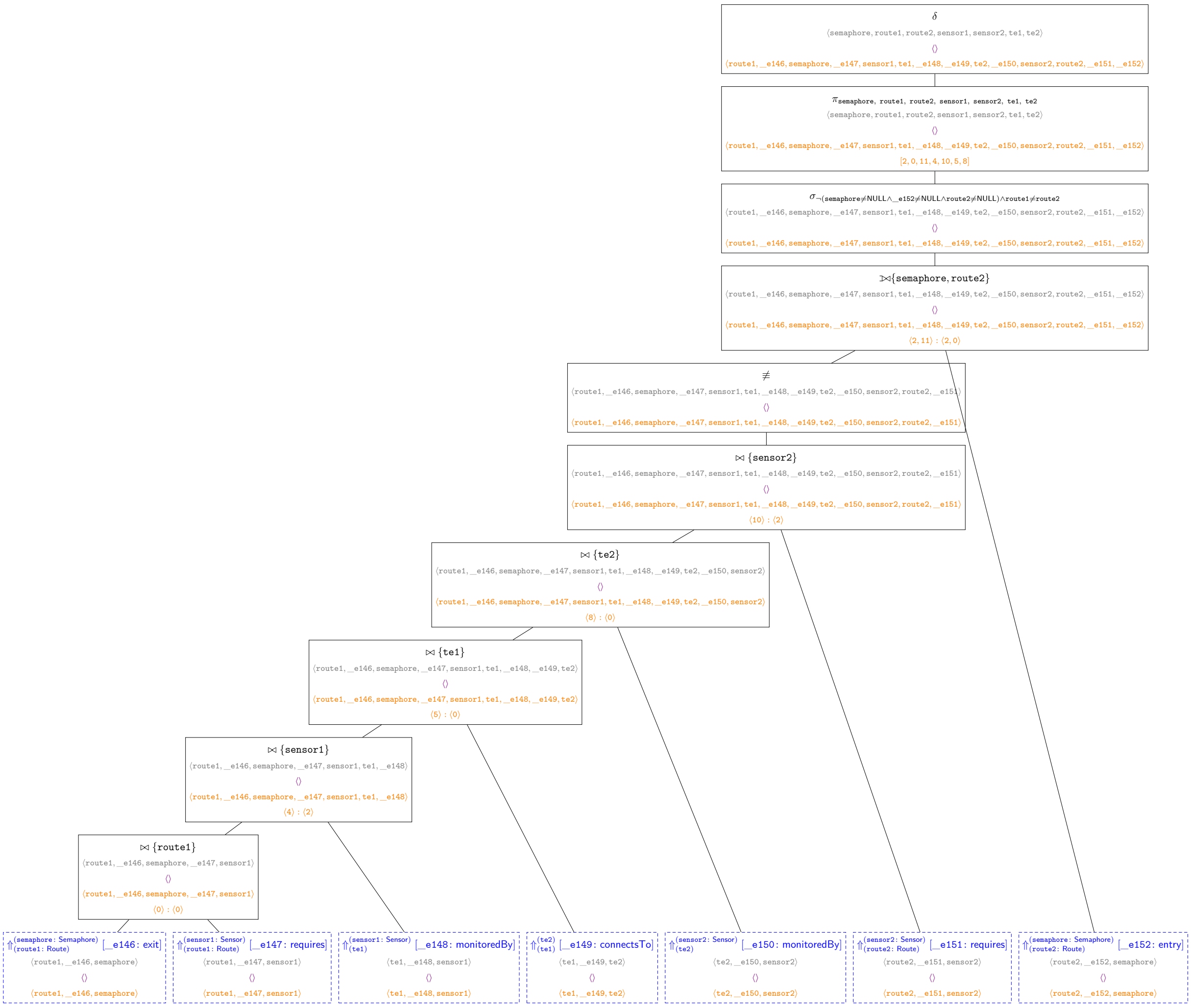
Relational algebra expression for search-based evaluation (SemaphoreNeighbor)

$$
\begin{aligned}
r = \delta\Big(&\pi_{\texttt{semaphore, route1, route2, sensor1, sensor2, te1, te2}} \\
&\Big(\sigma_{\neg(\texttt{semaphore}\neq\mathsf{NULL}\wedge\texttt{\_e152}\neq\mathsf{NULL}\wedge\texttt{route2}\neq\mathsf{NULL})\wedge\texttt{route1}\neq\texttt{route2}}\Big(\not\equiv\Big(\downarrow\begin{smallmatrix}(\texttt{route2}:\ \mathsf{Route})\\(\texttt{sensor2})\end{smallmatrix}\ [\mathsf{\_e151}:\ \mathsf{requires}] \\
&\Big(\uparrow\begin{smallmatrix}(\texttt{sensor2}:\ \mathsf{Sensor})\\(\texttt{te2})\end{smallmatrix}\ [\mathsf{\_e150}:\ \mathsf{monitoredBy}]\Big(\uparrow\begin{smallmatrix}(\texttt{te2})\\(\texttt{te1})\end{smallmatrix}\ [\mathsf{\_e149}:\ \mathsf{connectsTo}] \\
&\Big(\downarrow\begin{smallmatrix}(\texttt{te1})\\(\texttt{sensor1})\end{smallmatrix}\ [\mathsf{\_e148}:\ \mathsf{monitoredBy}]\Big(\uparrow\begin{smallmatrix}(\texttt{sensor1}:\ \mathsf{Sensor})\\(\texttt{route1})\end{smallmatrix}\ [\mathsf{\_e147}:\ \mathsf{requires}] \\
&\Big(\downarrow\begin{smallmatrix}(\texttt{route1}:\ \mathsf{Route})\\(\texttt{semaphore})\end{smallmatrix}\ [\mathsf{\_e146}:\ \mathsf{exit}]\Big(\bigcirc_{(\texttt{semaphore}:\ \mathsf{Semaphore})}\Big)\Big)\Big)\Big)\Big)\Big)\Big)\text{⟕}\downarrow\begin{smallmatrix}(\texttt{route2}:\ \mathsf{Route})\\(\texttt{semaphore})\end{smallmatrix}\ [\mathsf{\_e152}:\ \mathsf{entry}] \\
&\Big(\bigcirc_{(\texttt{semaphore}:\ \mathsf{Semaphore})}\Big)\Big)\Big)\Big)
\end{aligned}
$$

Relational algebra tree for search-based evaluation (SemaphoreNeighbor)


δ
⟨semaphore, route1, route2, sensor1, sensor2, te1, te2⟩
π semaphore, route1, route2, sensor1, sensor2, te1, te2
⟨semaphore, route1, route2, sensor1, sensor2, te1, te2⟩
σ ¬(semaphore≠NULL∧_e152≠NULL∧route2≠NULL)∧route1≠route2
⟨semaphore, _e146, route1, _e147, sensor1, _e148, te1, _e149, te2, _e150, sensor2, _e151, route2, _e152⟩
⟕{semaphore, route2}
⟨semaphore, _e146, route1, _e147, sensor1, _e148, te1, _e149, te2, _e150, sensor2, _e151, route2, _e152⟩
≢
⟨semaphore, _e146, route1, _e147, sensor1, _e148, te1, _e149, te2, _e150, sensor2, _e151, route2⟩
↓ (route2: Route) (semaphore) [_e152: entry]
⟨semaphore, _e152, route2⟩
↓ (route2: Route) (sensor2) [_e151: requires]
⟨semaphore, _e146, route1, _e147, sensor1, _e148, te1, _e149, te2, _e150, sensor2, _e151, route2⟩
↑ (sensor2: Sensor) (te2) [_e150: monitoredBy]
⟨semaphore, _e146, route1, _e147, sensor1, _e148, te1, _e149, te2, _e150, sensor2⟩
↑ (te2) (te1) [_e149: connectsTo]
⟨semaphore, _e146, route1, _e147, sensor1, _e148, te1, _e149, te2⟩
↓ (te1) (sensor1) [_e148: monitoredBy]
⟨semaphore, _e146, route1, _e147, sensor1, _e148, te1⟩
↑ (sensor1: Sensor) (route1) [_e147: requires]
⟨semaphore, _e146, route1, _e147, sensor1⟩
↓ (route1: Route) (semaphore) [_e146: exit]
⟨semaphore, _e146, route1⟩
○(semaphore: Semaphore)
⟨semaphore⟩
○(semaphore: Semaphore)
⟨semaphore⟩

Incremental relational algebra tree (SemaphoreNeighbor)


δ
⟨semaphore, route1, route2, sensor1, sensor2, te1, te2⟩
⟨⟩
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2, _e150, sensor2, route2, _e151, _e152⟩
π semaphore, route1, route2, sensor1, sensor2, te1, te2
⟨semaphore, route1, route2, sensor1, sensor2, te1, te2⟩
⟨⟩
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2, _e150, sensor2, route2, _e151, _e152⟩
[2, 0, 11, 4, 10, 5, 8]
σ ¬(semaphore≠NULL∧_e152≠NULL∧route2≠NULL)∧route1≠route2
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2, _e150, sensor2, route2, _e151, _e152⟩
⟨⟩
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2, _e150, sensor2, route2, _e151, _e152⟩
⟕{semaphore, route2}
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2, _e150, sensor2, route2, _e151, _e152⟩
⟨⟩
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2, _e150, sensor2, route2, _e151, _e152⟩
⟨2, 11⟩ : ⟨2, 0⟩
≢
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2, _e150, sensor2, route2, _e151⟩
⟨⟩
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2, _e150, sensor2, route2, _e151⟩
⋈ {sensor2}
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2, _e150, sensor2, route2, _e151⟩
⟨⟩
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2, _e150, sensor2, route2, _e151⟩
⟨10⟩ : ⟨2⟩
⋈ {te2}
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2, _e150, sensor2⟩
⟨⟩
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2, _e150, sensor2⟩
⟨8⟩ : ⟨0⟩
⋈ {te1}
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2⟩
⟨⟩
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148, _e149, te2⟩
⟨5⟩ : ⟨0⟩
⋈ {sensor1}
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148⟩
⟨⟩
⟨route1, _e146, semaphore, _e147, sensor1, te1, _e148⟩
⟨4⟩ : ⟨2⟩
⋈ {route1}
⟨route1, _e146, semaphore, _e147, sensor1⟩
⟨⟩
⟨route1, _e146, semaphore, _e147, sensor1⟩
⟨0⟩ : ⟨0⟩
⇑(semaphore: Semaphore) (route1: Route) [_e146: exit]
⟨route1, _e146, semaphore⟩
⟨⟩
⟨route1, _e146, semaphore⟩
⇑(sensor1: Sensor) (route1: Route) [_e147: requires]
⟨route1, _e147, sensor1⟩
⟨⟩
⟨route1, _e147, sensor1⟩
⇑(sensor1: Sensor) (te1) [_e148: monitoredBy]
⟨te1, _e148, sensor1⟩
⟨⟩
⟨te1, _e148, sensor1⟩
⇑(te2) (te1) [_e149: connectsTo]
⟨te1, _e149, te2⟩
⟨⟩
⟨te1, _e149, te2⟩
⇑(sensor2: Sensor) (te2) [_e150: monitoredBy]
⟨te2, _e150, sensor2⟩
⟨⟩
⟨te2, _e150, sensor2⟩
⇑(sensor2: Sensor) (route2: Route) [_e151: requires]
⟨route2, _e151, sensor2⟩
⟨⟩
⟨route2, _e151, sensor2⟩
⇑(semaphore: Semaphore) (route2: Route) [_e152: entry]
⟨route2, _e152, semaphore⟩
⟨⟩
⟨route2, _e152, semaphore⟩

H.1.8 SwitchMonitored

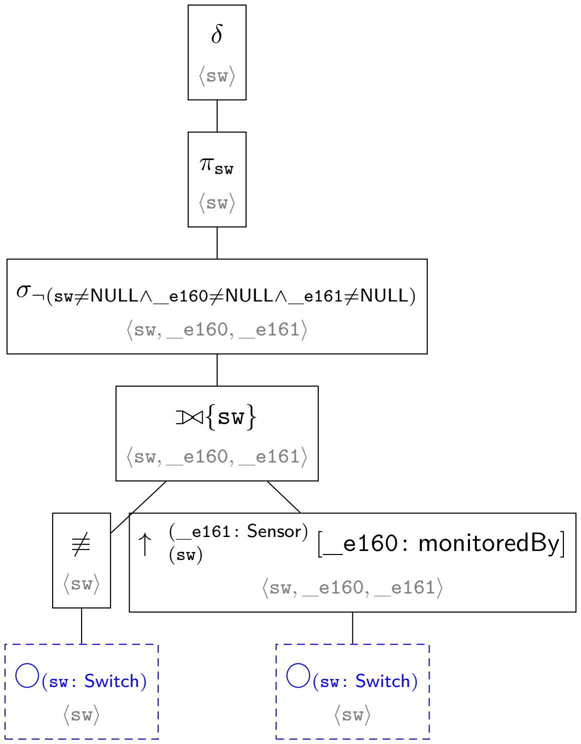
Query specification (SwitchMonitored)

```
MATCH (sw:Switch)
WHERE NOT ((sw)-[:monitoredBy]->(:Sensor))
RETURN DISTINCT sw
```

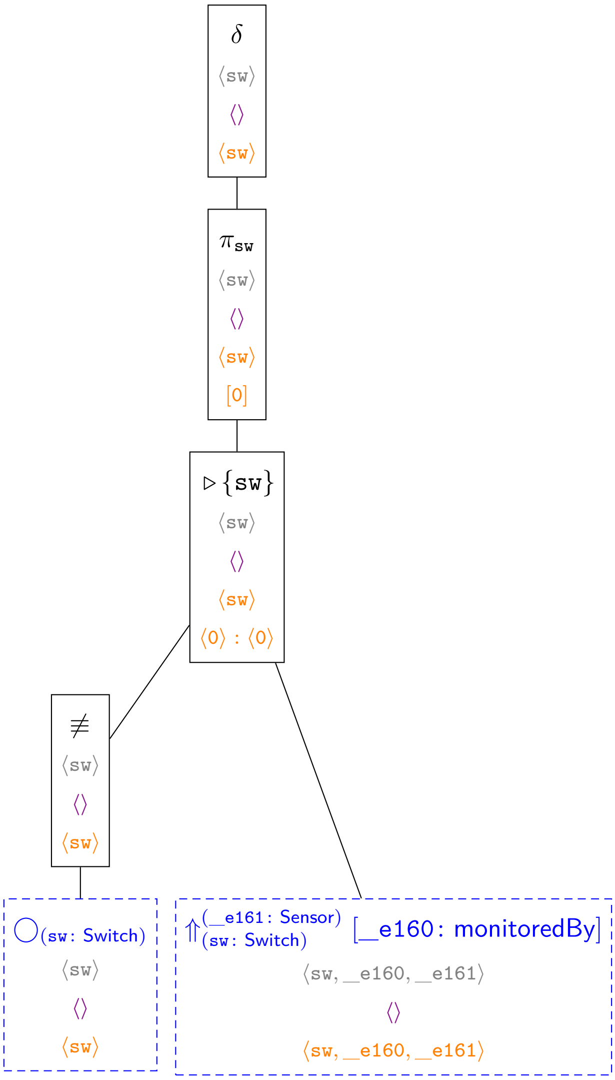
Relational algebra expression for search-based evaluation (SwitchMonitored)

$$r = \delta\Big(\pi_{\texttt{sw}}\Big(\sigma_{\neg(\texttt{sw}\neq\mathsf{NULL}\wedge\_\mathsf{e160}\neq\mathsf{NULL}\wedge\_\mathsf{e161}\neq\mathsf{NULL})}\Big(\not\equiv\Big(\bigcirc_{(\texttt{sw}:\,\mathsf{Switch})}\Big)⟕ \\ \uparrow^{(\_\mathsf{e161}:\,\mathsf{Sensor})}_{(\texttt{sw})}[\_\mathsf{e160}:\,\mathsf{monitoredBy}]\Big(\bigcirc_{(\texttt{sw}:\,\mathsf{Switch})}\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (SwitchMonitored)



Incremental relational algebra tree (SwitchMonitored)



H.1.9 SwitchSet-simple-return

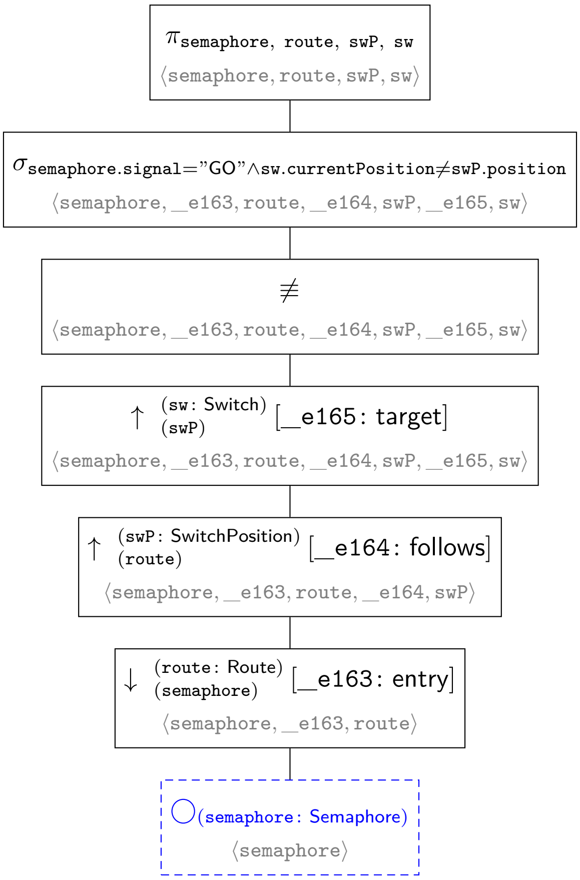
Query specification (SwitchSet-simple-return)

```
MATCH (semaphore:Semaphore)<-[:entry]-(route:Route)-[:follows]->(swP:SwitchPosition)-[:target]->(sw:Switch)
WHERE semaphore.signal = "GO"
  AND sw.currentPosition != swP.position
RETURN semaphore, route, swP, sw
```

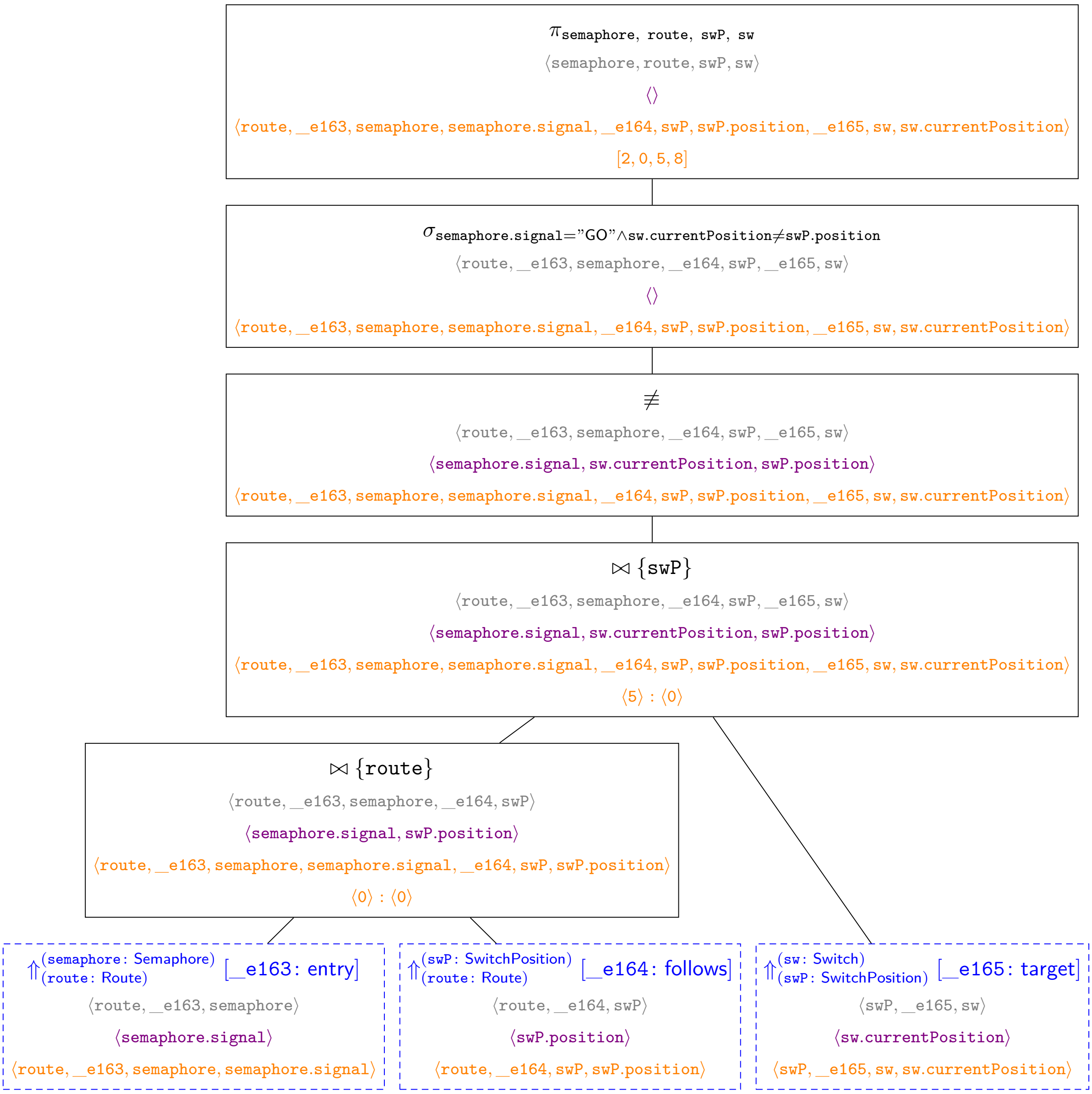
Relational algebra expression for search-based evaluation (SwitchSet-simple-return)

$$r = \pi_{\mathtt{semaphore,\ route,\ swP,\ sw}}\Big(\sigma_{\mathtt{semaphore.signal}=\text{"GO"}\wedge\mathtt{sw.currentPosition}\neq\mathtt{swP.position}}\Big(\not\equiv \\ \Big(\uparrow^{(\mathtt{sw}:\ \mathsf{Switch})}_{(\mathtt{swP})}[\_\mathsf{e165}:\ \mathsf{target}]\Big(\uparrow^{(\mathtt{swP}:\ \mathsf{SwitchPosition})}_{(\mathtt{route})}[\_\mathsf{e164}:\ \mathsf{follows}] \\ \Big(\downarrow^{(\mathtt{route}:\ \mathsf{Route})}_{(\mathtt{semaphore})}[\_\mathsf{e163}:\ \mathsf{entry}]\Big(\bigcirc_{(\mathtt{semaphore}:\ \mathsf{Semaphore})}\Big)\Big)\Big)\Big)\Big)\Big)$$

Relational algebra tree for search-based evaluation (SwitchSet-simple-return)


$\pi_{\texttt{semaphore, route, swP, sw}}$
⟨semaphore, route, swP, sw⟩
$\sigma_{\texttt{semaphore.signal}=\text{"GO"} \wedge \texttt{sw.currentPosition} \neq \texttt{swP.position}}$
⟨semaphore, _e163, route, _e164, swP, _e165, sw⟩
$\not\equiv$
⟨semaphore, _e163, route, _e164, swP, _e165, sw⟩
↑ (sw : Switch) (swP) [_e165 : target]
⟨semaphore, _e163, route, _e164, swP, _e165, sw⟩
↑ (swP : SwitchPosition) (route) [_e164 : follows]
⟨semaphore, _e163, route, _e164, swP⟩
↓ (route : Route) (semaphore) [_e163 : entry]
⟨semaphore, _e163, route⟩
○(semaphore : Semaphore)
⟨semaphore⟩

Incremental relational algebra tree (SwitchSet-simple-return)


π semaphore, route, swP, sw
⟨semaphore, route, swP, sw⟩
⟨⟩
⟨route, _e163, semaphore, semaphore.signal, _e164, swP, swP.position, _e165, sw, sw.currentPosition⟩
[2, 0, 5, 8]
σ semaphore.signal="GO"∧sw.currentPosition≠swP.position
⟨route, _e163, semaphore, _e164, swP, _e165, sw⟩
⟨⟩
⟨route, _e163, semaphore, semaphore.signal, _e164, swP, swP.position, _e165, sw, sw.currentPosition⟩
≢
⟨route, _e163, semaphore, _e164, swP, _e165, sw⟩
⟨semaphore.signal, sw.currentPosition, swP.position⟩
⟨route, _e163, semaphore, semaphore.signal, _e164, swP, swP.position, _e165, sw, sw.currentPosition⟩
⋈ {swP}
⟨route, _e163, semaphore, _e164, swP, _e165, sw⟩
⟨semaphore.signal, sw.currentPosition, swP.position⟩
⟨route, _e163, semaphore, semaphore.signal, _e164, swP, swP.position, _e165, sw, sw.currentPosition⟩
⟨5⟩ : ⟨0⟩
⋈ {route}
⟨route, _e163, semaphore, _e164, swP⟩
⟨semaphore.signal, swP.position⟩
⟨route, _e163, semaphore, semaphore.signal, _e164, swP, swP.position⟩
⟨0⟩ : ⟨0⟩
⇑(semaphore: Semaphore) (route: Route) [_e163: entry]
⟨route, _e163, semaphore⟩
⟨semaphore.signal⟩
⟨route, _e163, semaphore, semaphore.signal⟩
⇑(swP: SwitchPosition) (route: Route) [_e164: follows]
⟨route, _e164, swP⟩
⟨swP.position⟩
⟨route, _e164, swP, swP.position⟩
⇑(sw: Switch) (swP: SwitchPosition) [_e165: target]
⟨swP, _e165, sw⟩
⟨sw.currentPosition⟩
⟨swP, _e165, sw, sw.currentPosition⟩

H.1.10 SwitchSet

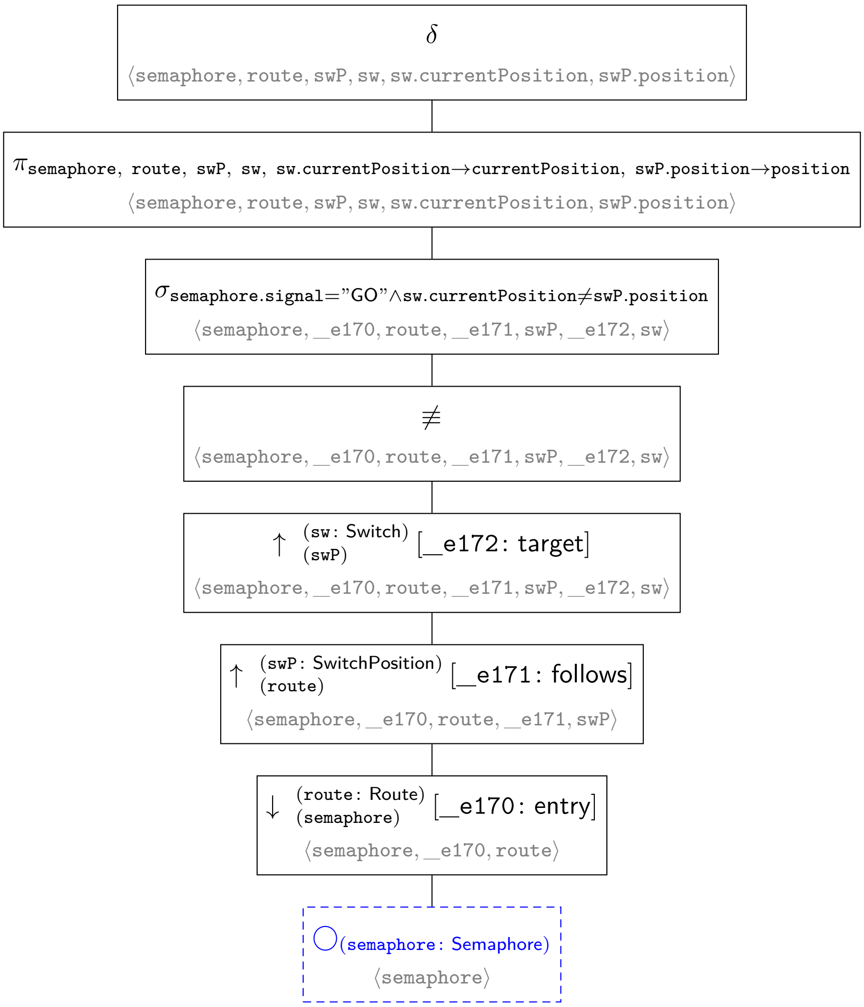
Query specification (SwitchSet)

```
MATCH (semaphore:Semaphore)<-[:entry]-(route:Route)-[:follows]->(swP:SwitchPosition)-[:target
    ]->(sw:Switch)
WHERE semaphore.signal = "GO"
  AND sw.currentPosition != swP.position
RETURN DISTINCT semaphore, route, swP, sw, sw.currentPosition AS currentPosition, swP.position
    AS position
```

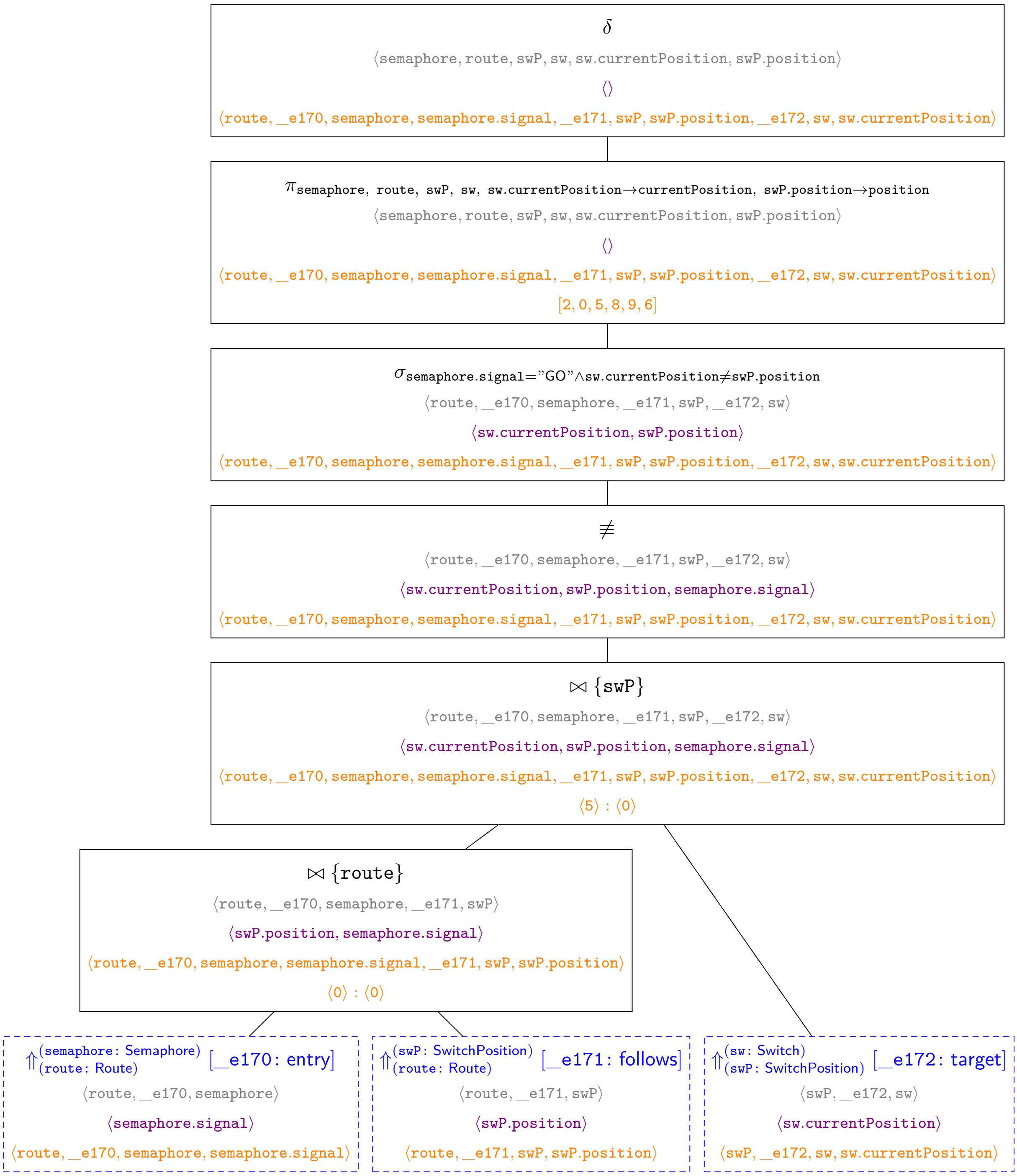
Relational algebra expression for search-based evaluation (SwitchSet)

$$
\begin{aligned}
r = \delta\Big(&\pi_{\texttt{semaphore, route, swP, sw, sw.currentPosition}\rightarrow\texttt{currentPosition, swP.position}\rightarrow\texttt{position}}\\
&\Big(\sigma_{\texttt{semaphore.signal}=\text{"GO"}\wedge\texttt{sw.currentPosition}\neq\texttt{swP.position}}\Big(\not\equiv\Big(\uparrow\begin{array}{l}(\texttt{sw}\colon \mathsf{Switch})\\(\texttt{swP})\end{array}[\_\mathsf{e172}\colon \mathsf{target}]\\
&\Big(\uparrow\begin{array}{l}(\texttt{swP}\colon \mathsf{SwitchPosition})\\(\texttt{route})\end{array}[\_\mathsf{e171}\colon \mathsf{follows}]\Big(\downarrow\begin{array}{l}(\texttt{route}\colon \mathsf{Route})\\(\texttt{semaphore})\end{array}[\_\mathsf{e170}\colon \mathsf{entry}]\Big(\bigcirc_{(\texttt{semaphore}\colon \mathsf{Semaphore})}\Big)\Big)\Big)\Big)\Big)\\
&\Big)\Big)
\end{aligned}
$$

Relational algebra tree for search-based evaluation (SwitchSet)


δ
⟨semaphore, route, swP, sw, sw.currentPosition, swP.position⟩
π semaphore, route, swP, sw, sw.currentPosition→currentPosition, swP.position→position
⟨semaphore, route, swP, sw, sw.currentPosition, swP.position⟩
σ semaphore.signal="GO"∧sw.currentPosition≠swP.position
⟨semaphore, _e170, route, _e171, swP, _e172, sw⟩
≢
⟨semaphore, _e170, route, _e171, swP, _e172, sw⟩
↑ (sw: Switch) (swP) [_e172: target]
⟨semaphore, _e170, route, _e171, swP, _e172, sw⟩
↑ (swP: SwitchPosition) (route) [_e171: follows]
⟨semaphore, _e170, route, _e171, swP⟩
↓ (route: Route) (semaphore) [_e170: entry]
⟨semaphore, _e170, route⟩
○(semaphore: Semaphore)
⟨semaphore⟩

Incremental relational algebra tree (SwitchSet)


δ
⟨semaphore, route, swP, sw, sw.currentPosition, swP.position⟩
⟨⟩
⟨route, _e170, semaphore, semaphore.signal, _e171, swP, swP.position, _e172, sw, sw.currentPosition⟩
π semaphore, route, swP, sw, sw.currentPosition→currentPosition, swP.position→position
⟨semaphore, route, swP, sw, sw.currentPosition, swP.position⟩
⟨⟩
⟨route, _e170, semaphore, semaphore.signal, _e171, swP, swP.position, _e172, sw, sw.currentPosition⟩
[2, 0, 5, 8, 9, 6]
σ semaphore.signal="GO"∧sw.currentPosition≠swP.position
⟨route, _e170, semaphore, _e171, swP, _e172, sw⟩
⟨sw.currentPosition, swP.position⟩
⟨route, _e170, semaphore, semaphore.signal, _e171, swP, swP.position, _e172, sw, sw.currentPosition⟩
⋕
⟨route, _e170, semaphore, _e171, swP, _e172, sw⟩
⟨sw.currentPosition, swP.position, semaphore.signal⟩
⟨route, _e170, semaphore, semaphore.signal, _e171, swP, swP.position, _e172, sw, sw.currentPosition⟩
⋈ {swP}
⟨route, _e170, semaphore, _e171, swP, _e172, sw⟩
⟨sw.currentPosition, swP.position, semaphore.signal⟩
⟨route, _e170, semaphore, semaphore.signal, _e171, swP, swP.position, _e172, sw, sw.currentPosition⟩
⟨5⟩ : ⟨0⟩
⋈ {route}
⟨route, _e170, semaphore, _e171, swP⟩
⟨swP.position, semaphore.signal⟩
⟨route, _e170, semaphore, semaphore.signal, _e171, swP, swP.position⟩
⟨0⟩ : ⟨0⟩
⇑ (semaphore: Semaphore) (route: Route) [_e170: entry]
⟨route, _e170, semaphore⟩
⟨semaphore.signal⟩
⟨route, _e170, semaphore, semaphore.signal⟩
⇑ (swP: SwitchPosition) (route: Route) [_e171: follows]
⟨route, _e171, swP⟩
⟨swP.position⟩
⟨route, _e171, swP, swP.position⟩
⇑ (sw: Switch) (swP: SwitchPosition) [_e172: target]
⟨swP, _e172, sw⟩
⟨sw.currentPosition⟩
⟨swP, _e172, sw, sw.currentPosition⟩