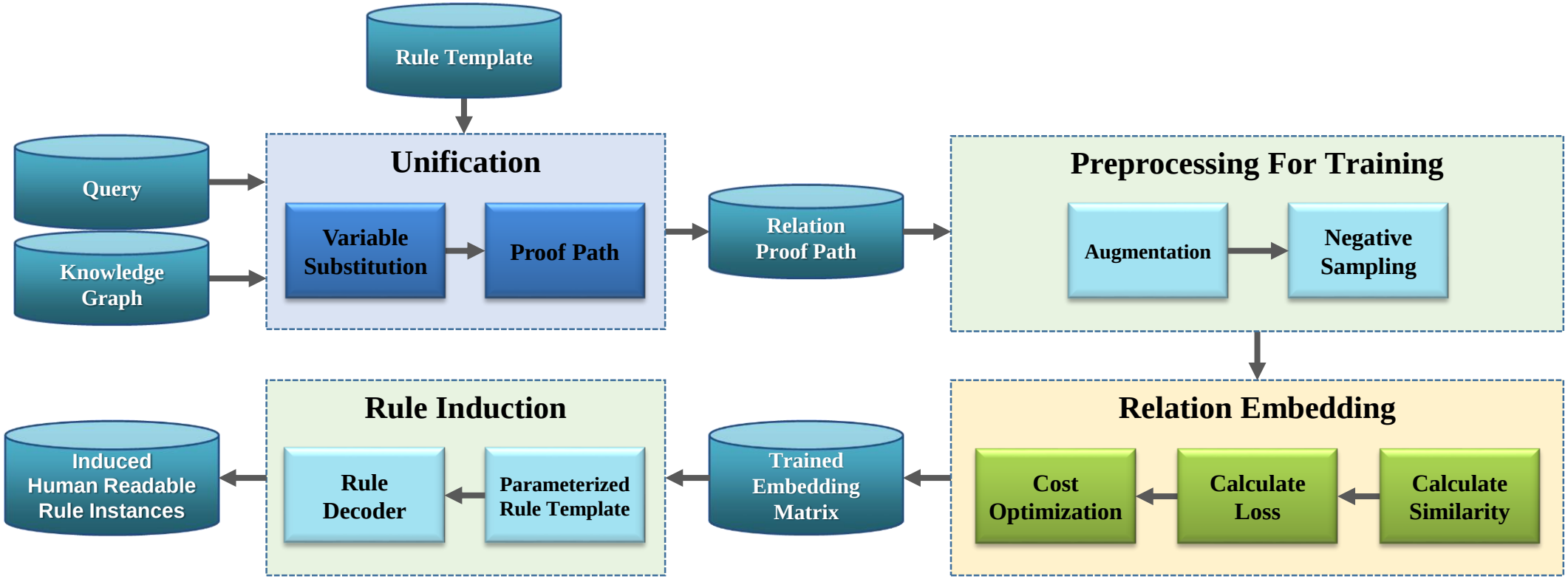
뉴로 심볼릭 기반 릴레이션 유사도 학습 Document

뉴로 심볼릭 기반 지식 완성 개념도



Unification Overview

- **Unification()**
  - Unification을 재귀적으로 호출하여 주어진 Query과 Rule template으로부터 Rule component substitution을 생성
  - Proof path completion을 호출하여 Proof path에 대한 Relation group을 return

Rule Schema

Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)
Rule 1 : #1(X, Y) :- #2(X, Z), #3(Z, W), #4(W, Y)

[[(#1, X, Y), (#2, X, Z), (#3, Z, Y), 2]
[(#1, X, Y), (#2, X, Z), (#3, Z, W), (#4, W, Y), 2]]

Rule Template

r(h, t)

Knowledge Graph

Query : [nationality, BART, USA]

Unification()

Rule Component Substitution(Dict)

Key(Tuple)	Value(DataFrame)		
(#1, X, Y)	**#1**	**X**	**Y**
	nationality	BART	USA
(#2, X, Z)	**#2**	**X**	**Z**
	placeOfBirth	BART	NEWYORK
	hasFather	BART	HOMMER
(#3, Z, Y)	**#3**	**Z**	**Y**
	locatedIn	NEWYORK	USA
	nationality	HOMMER	USA

Proof Path Completion()

Relation Group

[[(#1, nationality), (#2, placeOfBirth), (#3, locatedIn)],
[(#1, nationality), (#2, hasFather), (#3, nationality)]]

Unification Case

- **Unification의 3가지 Case**
  - Rule Template의 depth에 따라 Unification수행

```python
def unification(goal, rule, KG, KG_index, rule_num, depth = 0,
                variable_substitution={}, rComp_substitution={},rule_structure=None):

    if depth == 0 :
        rComp_substitution[rule[depth]] = pd.DataFrame(goal,
                                                       index=[rule[depth][0],rule[depth][1],rule[depth][2]]).transpose()
        # subject variable binding
        if rule[depth][1] not in variable_substitution.keys():
            variable_substitution[rule[depth][1]] = [goal[1]]
        # object variable binding
        if rule[depth][2] not in variable_substitution.keys():
            variable_substitution[rule[depth][2]] = [goal[2]]
        depth += 1

    if depth == len(rule)-1:

        return proof_path_completion(rComp_substitution, rule)

    else :
        common_variable = []
        current_body = rule_structure[rule_structure['rule_number'] == rule_num].iloc[0,depth]
        current_variable = list(current_body.keys())
        common_variable = [variable for variable in current_variable if variable in variable_substitution]
        cVar_position = list(map(current_body.get, common_variable))
        unified_cVar= list(map(variable_substitution.get, common_variable))

        if len(common_variable) == 1:
            cVar_index = set(itertools.chain.from_iterable(
                             list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))

            sub_goal = KG.loc[cVar_index]
            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
            rComp_substitution[rule[depth]] = sub_goal

            # subject variable binding
            if rule[depth][1] not in variable_substitution.keys():
                variable_substitution[rule[depth][1]] = list(set(sub_goal[rule[depth][1]].values))
            # object variable binding
            if rule[depth][2] not in variable_substitution.keys():
                variable_substitution[rule[depth][2]] = list(set(sub_goal[rule[depth][2]].values))

        else :
            subj_cVar_index = set(itertools.chain.from_iterable(
                                  list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
            obj_cVar_index = set(itertools.chain.from_iterable(
                                  list(map(lambda x : KG_index.get(x).get(cVar_position[1]), unified_cVar[1]))))
            sub_goal = KG.loc[subj_cVar_index &obj_cVar_index]
            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
            rComp_substitution[rule[depth]] = sub_goal
        depth += 1
        return unification(goal, rule, KG, KG_index, rule_num, depth,
                           variable_substitution, rComp_substitution,rule_structure)
```

depth : 0

- **Case-1 :** Head Unification [(#1,X, Y) , (#2,X, Z) , (#3,Z,Y) , 2]

Rule Component Substitution			
Key(Tuple)	Value(DataFrame)		
(#1, X, Y)	#1	X	Y
	nationality	BART	USA

- **Case-2 :** Unification End
  Call proof path completion()

depth : 1 depth : 2

- **Case-3 :** Body Unification [(#1,X, Y) , (#2,X, Z) , (#3,Z,Y) , 2]

Rule Component Substitution			
Key(Tuple)	Value(DataFrame)		
(#2, X, Z)	#2	X	Z
	placeOfBirth	BART	NEWYORK
	hasFather	BART	HOMMER
(#3, Z, Y)	#3	Z	Y
	locatedIn	NEWYORK	USA
	nationality	HOMMER	USA

Unification Case1 (depth = 0)

- **Head Unification**
  - Rule의 Head와 Unification을 수행하여 Head에 대한 Component를 생성하고, Variable을 Binding
  - Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```
4    if depth == 0 :
5        rComp_substitution[rule[depth]] = pd.DataFrame(goal,
6                                                       index=[rule[depth][0],rule[depth][1],rule[depth][2]]).transpose()
7        # subject variable binding
8        if rule[depth][1] not in variable_substitution.keys():
9            variable_substitution[rule[depth][1]] = [goal[1]]
10       # object variable binding
11       if rule[depth][2] not in variable_substitution.keys():
12           variable_substitution[rule[depth][2]] = [goal[2]]
13       depth += 1
```

goal : [nationality, BART, USA]

rule : [(#1,X, Y) , (#2,X, Z) , (#3,Z,Y) , 2]

depth = 0

Unification()
depth = 0

Rule Component Substitution(Dict)			
Key(Tuple)	Value(DataFrame)		
(#1, X, Y)	#1	X	Y
	nationality	BART	USA

Variable Substitution(Dict)	
Key(Str)	Value(list)
X	[BART]
Y	[USA]

Unification Case3 (depth = 1)

- **Body Unification**
  - Rule structure : Rule template에서 각 Rule의 Component가 갖는 변수 및 위치정보를 저장한 DataFrame

```
    else :
        common_variable = []
        current_body = rule_structure[rule_structure['rule_number'] == rule_num].iloc[0,depth]
        current_variable = list(current_body.keys())
        common_variable = [variable for variable in current_variable if variable in variable_substitution]
        cVar_position = list(map(current_body.get, common_variable))
        unified_cVar= list(map(variable_substitution.get, common_variable))

        if len(common_variable) == 1:
            cVar_index = set(itertools.chain.from_iterable(
                            list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))

            sub_goal = KG.loc[cVar_index]
            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
            rComp_substitution[rule[depth]] = sub_goal

            # subject variable binding
            if rule[depth][1] not in variable_substitution.keys():
                variable_substitution[rule[depth][1]] = list(set(sub_goal[rule[depth][1]].values))
            # object variable binding
            if rule[depth][2] not in variable_substitution.keys():
                variable_substitution[rule[depth][2]] = list(set(sub_goal[rule[depth][2]].values))

        else :
            subj_cVar_index = set(itertools.chain.from_iterable(
                                list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
            obj_cVar_index = set(itertools.chain.from_iterable(
                                list(map(lambda x : KG_index.get(x).get(cVar_position[1]), unified_cVar[1]))))
            sub_goal = KG.loc[subj_cVar_index &obj_cVar_index]
            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
            rComp_substitution[rule[depth]] = sub_goal
        depth += 1
        return unification(goal, rule, KG, KG_index, rule_num, depth,
                           variable_substitution, rComp_substitution,rule_structure)
```

Rule Templates

$[\ [(\#1, X, Y), (\#2, X, Z), (\#3, Z, Y), 2]$ → Index 0

$[(\#1, X, Y), (\#2, X, Z), (\#3, Z, W), (\#4, W, Y), 2]\ ]$ → Index 1

Generate Rule Structure

Rule Structure(DataFrame)

0	1	2	3	rule number
{ X : subj, Y : obj }	{ X : subj , Z : obj }	{ Z : subj, Y : obj }	None	0
{ X : subj, Y : obj }	{ X : subj , Z : obj }	{ Z : subj, W : obj }	{ W : subj, Y : obj }	1

Unification Case3 (depth = 1)

- **Body Unification**
  - Rule structure를 참조하여 첫 번째 Body component의 변수에 대한 정보 저장
  - Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```
19    else :
20        common_variable = []
21        current_body = rule_structure[rule_structure['rule_number'] == rule_num].iloc[0,depth]
22        current_variable = list(current_body.keys())
23        common_variable = [variable for variable in current_variable if variable in variable_substitution]
24        cVar_position = list(map(current_body.get, common_variable))
25        unified_cVar= list(map(variable_substitution.get, common_variable))
26
27        if len(common_variable) == 1:
28            cVar_index = set(itertools.chain.from_iterable(
29                            list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
30
31            sub_goal = KG.loc[cVar_index]
32            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
33            rComp_substitution[rule[depth]] = sub_goal
34
35            # subject variable binding
36            if rule[depth][1] not in variable_substitution.keys():
37                variable_substitution[rule[depth][1]] = list(set(sub_goal[rule[depth][1]].values))
38            # object variable binding
39            if rule[depth][2] not in variable_substitution.keys():
40                variable_substitution[rule[depth][2]] = list(set(sub_goal[rule[depth][2]].values))
41
42        else :
43            subj_cVar_index = set(itertools.chain.from_iterable(
44                                list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
45            obj_cVar_index = set(itertools.chain.from_iterable(
46                                list(map(lambda x : KG_index.get(x).get(cVar_position[1]), unified_cVar[1]))))
47            sub_goal = KG.loc[subj_cVar_index &obj_cVar_index]
48            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
49            rComp_substitution[rule[depth]] = sub_goal
50        depth += 1
51        return unification(goal, rule, KG, KG_index, rule_num, depth,
52                           variable_substitution, rComp_substitution,rule_structure)
```

Rule Templates

depth = 1

[[(#1,X, Y), (#2,X, Z), (#3,Z, Y), 2] → Index 0 = rule _num

[(#1,X, Y), (#2,X, Z), (#3,Z, W),(#4,W, Y), 2]]

Rule Structure(DataFrame)				
0	1	2	3	rule number
{ X : subj, Y : obj }	{ X : subj , Z : obj }	{ Z : subj, Y : obj }	None	0
{ X : subj, Y : obj }	{ X : subj , Z : obj }	{ Z : subj, W : obj }	{ W : subj, Y : obj }	1

current _body = { X : subj , Z : obj }

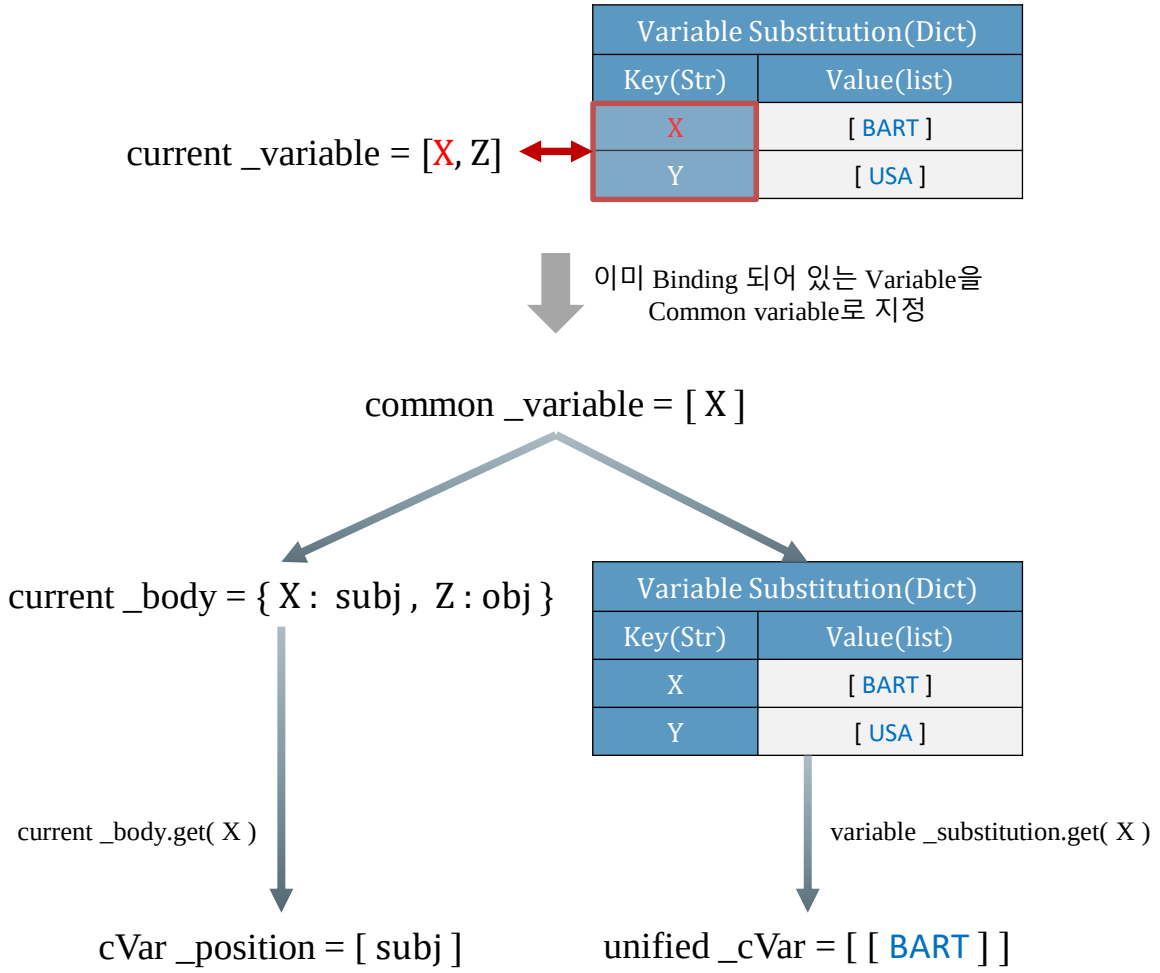
current _body.keys()

current _variable = [X, Z]

Unification Case3 (depth = 1)

- **Body Unification**
  - Variable substitution을 참조하여 Common variable의 위치정보와 Unify된 엔티티를 저장
  - Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```
19    else :
20        common_variable = []
21        current_body = rule_structure[rule_structure['rule_number'] == rule_num].iloc[0,depth]
22        current_variable = list(current_body.keys())
23        common_variable = [variable for variable in current_variable if variable in variable_substitution]
24        cVar_position = list(map(current_body.get, common_variable))
25        unified_cVar= list(map(variable_substitution.get, common_variable))
26
27        if len(common_variable) == 1:
28            cVar_index = set(itertools.chain.from_iterable(
29                             list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
30
31            sub_goal = KG.loc[cVar_index]
32            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
33            rComp_substitution[rule[depth]] = sub_goal
34
35            # subject variable binding
36            if rule[depth][1] not in variable_substitution.keys():
37                variable_substitution[rule[depth][1]] = list(set(sub_goal[rule[depth][1]].values))
38            # object variable binding
39            if rule[depth][2] not in variable_substitution.keys():
40                variable_substitution[rule[depth][2]] = list(set(sub_goal[rule[depth][2]].values))
41
42        else :
43            subj_cVar_index = set(itertools.chain.from_iterable(
44                                  list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
45            obj_cVar_index = set(itertools.chain.from_iterable(
46                                  list(map(lambda x : KG_index.get(x).get(cVar_position[1]), unified_cVar[1]))))
47            sub_goal = KG.loc[subj_cVar_index &obj_cVar_index]
48            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
49            rComp_substitution[rule[depth]] = sub_goal
50        depth += 1
51        return unification(goal, rule, KG, KG_index, rule_num, depth,
52                           variable_substitution, rComp_substitution,rule_structure)
```



Unification Case3 (depth = 1)

- **Body Unification**
  - KG index : KG의 각 엔티티를 Subject, Object로 갖는 트리플의 Index를 맵핑한 Dictionary

```
19    else :
20        common_variable = []
21        current_body = rule_structure[rule_structure['rule_number'] == rule_num].iloc[0,depth]
22        current_variable = list(current_body.keys())
23        common_variable = [variable for variable in current_variable if variable in variable_substitution]
24        cVar_position = list(map(current_body.get, common_variable))
25        unified_cVar= list(map(variable_substitution.get, common_variable))
26
27        if len(common_variable) == 1:
28            cVar_index = set(itertools.chain.from_iterable(
29                             list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
30
31            sub_goal = KG.loc[cVar_index]
32            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
33            rComp_substitution[rule[depth]] = sub_goal
34
35            # subject variable binding
36            if rule[depth][1] not in variable_substitution.keys():
37                variable_substitution[rule[depth][1]] = list(set(sub_goal[rule[depth][1]].values))
38            # object variable binding
39            if rule[depth][2] not in variable_substitution.keys():
40                variable_substitution[rule[depth][2]] = list(set(sub_goal[rule[depth][2]].values))
41
42        else :
43            subj_cVar_index = set(itertools.chain.from_iterable(
44                                 list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
45            obj_cVar_index = set(itertools.chain.from_iterable(
46                                 list(map(lambda x : KG_index.get(x).get(cVar_position[1]), unified_cVar[1]))))
47            sub_goal = KG.loc[subj_cVar_index &obj_cVar_index]
48            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
49            rComp_substitution[rule[depth]] = sub_goal
50        depth += 1
51        return unification(goal, rule, KG, KG_index, rule_num, depth,
52                           variable_substitution, rComp_substitution,rule_structure)
```

Knowledge Graph(DataFrame)			
Index	**pred**	**subj**	**obj**
0	nationality	BART	USA
1	placeOfBirth	BART	NEWYORK
2	hasFather	BART	HOMMER
3	placeOfBirth	HOMMER	NEWYORK
4	nationality	HOMMER	USA
5	locatedIn	NEWYORK	USA

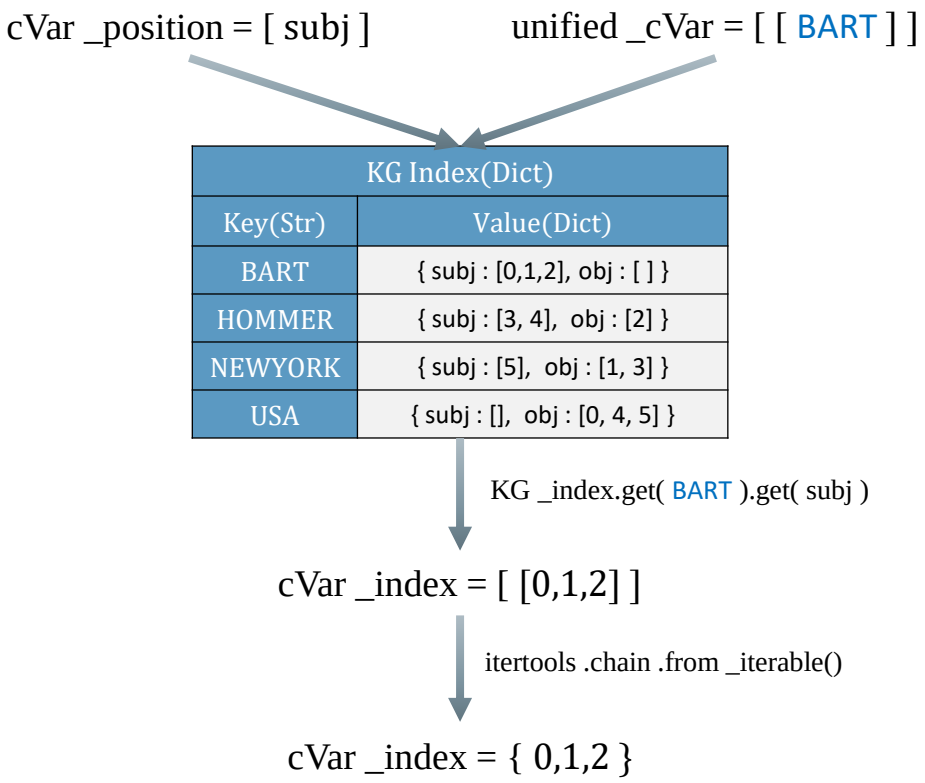
Generate KG Index

KG Index(Dict)	
Key(Str)	Value(Dict)
BART	{ subj : [0,1,2], obj : [ ] }
HOMMER	{ subj : [3, 4], obj : [2] }
NEWYORK	{ subj : [5], obj : [1, 3] }
USA	{ subj : [], obj : [0, 4, 5] }

Unification Case3 (depth = 1)

- **Body Unification**
  - Common variable의 위치정보와 엔티티 정보를 기반으로 KG index를 참조하여 Unify될 트리플의 Index정보 저장
  - Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```
19    else :
20        common_variable = []
21        current_body = rule_structure[rule_structure['rule_number'] == rule_num].iloc[0,depth]
22        current_variable = list(current_body.keys())
23        common_variable = [variable for variable in current_variable if variable in variable_substitution]
24        cVar_position = list(map(current_body.get, common_variable))
25        unified_cVar= list(map(variable_substitution.get, common_variable))
26
27        if len(common_variable) == 1:
28            cVar_index = set(itertools.chain.from_iterable(
29                             list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
30
31            sub_goal = KG.loc[cVar_index]
32            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
33            rComp_substitution[rule[depth]] = sub_goal
34
35            # subject variable binding
36            if rule[depth][1] not in variable_substitution.keys():
37                variable_substitution[rule[depth][1]] = list(set(sub_goal[rule[depth][1]].values))
38            # object variable binding
39            if rule[depth][2] not in variable_substitution.keys():
40                variable_substitution[rule[depth][2]] = list(set(sub_goal[rule[depth][2]].values))
41
42        else :
43            subj_cVar_index = set(itertools.chain.from_iterable(
44                                 list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
45            obj_cVar_index = set(itertools.chain.from_iterable(
46                                 list(map(lambda x : KG_index.get(x).get(cVar_position[1]), unified_cVar[1]))))
47            sub_goal = KG.loc[subj_cVar_index &obj_cVar_index]
48            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
49            rComp_substitution[rule[depth]] = sub_goal
50        depth += 1
51        return unification(goal, rule, KG, KG_index, rule_num, depth,
52                           variable_substitution, rComp_substitution,rule_structure)
```



Unification Case3 (depth = 1)

- **Body Unification**
  - Index정보와 KG로부터 Unify될 트리플을 가져와 해당 Component의 Substitution생성
  - Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```
19    else :
20        common_variable = []
21        current_body = rule_structure[rule_structure['rule_number'] == rule_num].iloc[0,depth]
22        current_variable = list(current_body.keys())
23        common_variable = [variable for variable in current_variable if variable in variable_substitution]
24        cVar_position = list(map(current_body.get, common_variable))
25        unified_cVar= list(map(variable_substitution.get, common_variable))
26
27        if len(common_variable) == 1:
28            cVar_index = set(itertools.chain.from_iterable(
29                             list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
30
31            sub_goal = KG.loc[cVar_index]
32            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
33            rComp_substitution[rule[depth]] = sub_goal
34
35            # subject variable binding
36            if rule[depth][1] not in variable_substitution.keys():
37                variable_substitution[rule[depth][1]] = list(set(sub_goal[rule[depth][1]].values))
38            # object variable binding
39            if rule[depth][2] not in variable_substitution.keys():
40                variable_substitution[rule[depth][2]] = list(set(sub_goal[rule[depth][2]].values))
41
42        else :
43            subj_cVar_index = set(itertools.chain.from_iterable(
44                                 list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
45            obj_cVar_index = set(itertools.chain.from_iterable(
46                                 list(map(lambda x : KG_index.get(x).get(cVar_position[1]), unified_cVar[1]))))
47            sub_goal = KG.loc[subj_cVar_index &obj_cVar_index]
48            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
49            rComp_substitution[rule[depth]] = sub_goal
50        depth += 1
51        return unification(goal, rule, KG, KG_index, rule_num, depth,
52                           variable_substitution, rComp_substitution,rule_structure)
```

cVar _index = { 0,1,2 }

Knowledge Graph(DataFrame)

Index	pred	subj	obj
0	nationality	BART	USA
1	placeOfBirth	BART	NEWYORK
2	hasFather	BART	HOMMER
3	placeOfBirth	HOMMER	NEWYORK
4	nationality	HOMMER	USA
5	locatedIn	NEWYORK	USA

depth = 1

rule : [(#1,X, Y) , (#2,X, Z) , (#3,Z, Y) , 2]

Rule Component Substitution(Dict)

Key(Tuple)	Value(DataFrame)		
(#1, X, Y)	#1	X	Y
	nationality	BART	USA
(#2, X, Z)	#2	X	Z
	nationality	BART	USA
	placeOfBirth	BART	NEWYORK
	hasFather	BART	HOMMER

Unification Case3 (depth = 1)

- **Body Unification**
  - 해당 Rule component의 Substitution으로부터 Variable에 대한 Binding수행
  - Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```
19    else :
20        common_variable = []
21        current_body = rule_structure[rule_structure['rule_number'] == rule_num].iloc[0,depth]
22        current_variable = list(current_body.keys())
23        common_variable = [variable for variable in current_variable if variable in variable_substitution]
24        cVar_position = list(map(current_body.get, common_variable))
25        unified_cVar= list(map(variable_substitution.get, common_variable))
26
27        if len(common_variable) == 1:
28            cVar_index = set(itertools.chain.from_iterable(
29                             list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
30
31            sub_goal = KG.loc[cVar_index]
32            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
33            rComp_substitution[rule[depth]] = sub_goal
34
35            # subject variable binding
36            if rule[depth][1] not in variable_substitution.keys():
37                variable_substitution[rule[depth][1]] = list(set(sub_goal[rule[depth][1]].values))
38            # object variable binding
39            if rule[depth][2] not in variable_substitution.keys():
40                variable_substitution[rule[depth][2]] = list(set(sub_goal[rule[depth][2]].values))
41
42        else :
43            subj_cVar_index = set(itertools.chain.from_iterable(
44                                  list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
45            obj_cVar_index = set(itertools.chain.from_iterable(
46                                  list(map(lambda x : KG_index.get(x).get(cVar_position[1]), unified_cVar[1]))))
47            sub_goal = KG.loc[subj_cVar_index &obj_cVar_index]
48            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
49            rComp_substitution[rule[depth]] = sub_goal
50        depth += 1
51        return unification(goal, rule, KG, KG_index, rule_num, depth,
52                           variable_substitution, rComp_substitution,rule_structure)
```

depth = 1

rule : [(#1,X, Y) , (#2,X, Z) , (#3,Z, Y) , 2]

Variable Substitution(Dict)	
Key(Str)	Value(list)
X	[BART]
Y	[USA]

Binding 되어 있지 않은 Variable에 대해 Sub goal로부터 Binding

Sub Goal(DataFrame)		
#2	X	Z
nationality	BART	USA
placeOfBirth	BART	NEWYORK
hasFather	BART	HOMMER

Variable Substitution(Dict)	
Key(Str)	Value(list)
X	[BART]
Y	[USA]
Z	[USA, NEWYORK, HOMMER]

Unification Case3 (depth = 2)

- **Body Unification**
  - Rule structure를 참조하여 두 번째 Body component의 변수에 대한 정보 저장
  - Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```
19      else :
20          common_variable = []
21          current_body = rule_structure[rule_structure['rule_number'] == rule_num].iloc[0,depth]
22          current_variable = list(current_body.keys())
23          common_variable = [variable for variable in current_variable if variable in variable_substitution]
24          cVar_position = list(map(current_body.get, common_variable))
25          unified_cVar= list(map(variable_substitution.get, common_variable))
26
27          if len(common_variable) == 1:
28              cVar_index = set(itertools.chain.from_iterable(
29                              list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
30
31              sub_goal = KG.loc[cVar_index]
32              sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
33              rComp_substitution[rule[depth]] = sub_goal
34
35              # subject variable binding
36              if rule[depth][1] not in variable_substitution.keys():
37                  variable_substitution[rule[depth][1]] = list(set(sub_goal[rule[depth][1]].values))
38              # object variable binding
39              if rule[depth][2] not in variable_substitution.keys():
40                  variable_substitution[rule[depth][2]] = list(set(sub_goal[rule[depth][2]].values))
41
42          else :
43              subj_cVar_index = set(itertools.chain.from_iterable(
44                                  list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
45              obj_cVar_index = set(itertools.chain.from_iterable(
46                                  list(map(lambda x : KG_index.get(x).get(cVar_position[1]), unified_cVar[1]))))
47              sub_goal = KG.loc[subj_cVar_index &obj_cVar_index]
48              sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
49              rComp_substitution[rule[depth]] = sub_goal
50          depth += 1
51          return unification(goal, rule, KG, KG_index, rule_num, depth,
52                             variable_substitution, rComp_substitution,rule_structure)
```

Rule Templates

depth = 2

[[(#1,X, Y), (#2,X, Z), (#3,Z, Y) , 2] → Index 0 = rule _num

[(#1,X, Y), (#2,X, Z), (#3,Z, W),(#4,W, Y), 2]]

Rule Structure(DataFrame)				
0	1	2	3	rule number
{ X : subj, Y : obj }	{ X : subj , Z : obj }	{ Z : subj, Y : obj }	None	**0**
{ X : subj, Y : obj }	{ X : subj , Z : obj }	{ Z : subj, W : obj }	{ W : subj, Y : obj }	**1**

current _body = { Z : subj , Y : obj }

current _body.keys()

current _variable = [Z, Y]

Unification Case3 (depth = 2)

- **Body Unification**
  - Variable substitution을 참조하여 Common variable의 위치정보와 Unify된 엔티티를 저장
  - Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```
19    else :
20        common_variable = []
21        current_body = rule_structure[rule_structure['rule_number'] == rule_num].iloc[0,depth]
22        current_variable = list(current_body.keys())
23        common_variable = [variable for variable in current_variable if variable in variable_substitution]
24        cVar_position = list(map(current_body.get, common_variable))
25        unified_cVar= list(map(variable_substitution.get, common_variable))
26
27        if len(common_variable) == 1:
28            cVar_index = set(itertools.chain.from_iterable(
29                             list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
30
31            sub_goal = KG.loc[cVar_index]
32            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
33            rComp_substitution[rule[depth]] = sub_goal
34
35            # subject variable binding
36            if rule[depth][1] not in variable_substitution.keys():
37                variable_substitution[rule[depth][1]] = list(set(sub_goal[rule[depth][1]].values))
38            # object variable binding
39            if rule[depth][2] not in variable_substitution.keys():
40                variable_substitution[rule[depth][2]] = list(set(sub_goal[rule[depth][2]].values))
41
42        else :
43            subj_cVar_index = set(itertools.chain.from_iterable(
44                                  list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
45            obj_cVar_index = set(itertools.chain.from_iterable(
46                                  list(map(lambda x : KG_index.get(x).get(cVar_position[1]), unified_cVar[1]))))
47            sub_goal = KG.loc[subj_cVar_index &obj_cVar_index]
48            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
49            rComp_substitution[rule[depth]] = sub_goal
50        depth += 1
51        return unification(goal, rule, KG, KG_index, rule_num, depth,
52                           variable_substitution, rComp_substitution,rule_structure)
```

current _variable = [Z, Y]

Variable Substitution(Dict)	
Key(Str)	Value(list)
X	[BART]
Y	[USA]
Z	[USA, NEWYORK, HOMMER]

이미 Binding 되어 있는 Variable을 Common variable로 지정

common _variable = [Z, Y]

current _body = { Z : subj , Y : obj }

current _body.get(Z)
current _body.get(Y)

cVar _position = [subj, obj]

Variable Substitution(Dict)	
Key(Str)	Value(list)
X	[BART]
Y	[USA]
Z	[USA, NEWYORK, HOMMER]

variable _substitution.get(Z)
variable _substitution.get(Y)

unified _cVar = [[USA, NEWYORK, HOMMER]
[USA]]

Unification Case3 (depth = 2)

- **Body Unification**
  - Common cariable의 위치정보와 엔티티 정보를 기반으로 KG Index를 참조하여 Unify될 트리플의 Index정보 저장
  - Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```
19    else :
20        common_variable = []
21        current_body = rule_structure[rule_structure['rule_number'] == rule_num].iloc[0,depth]
22        current_variable = list(current_body.keys())
23        common_variable = [variable for variable in current_variable if variable in variable_substitution]
24        cVar_position = list(map(current_body.get, common_variable))
25        unified_cVar= list(map(variable_substitution.get, common_variable))
26
27        if len(common_variable) == 1:
28            cVar_index = set(itertools.chain.from_iterable(
29                             list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
30
31            sub_goal = KG.loc[cVar_index]
32            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
33            rComp_substitution[rule[depth]] = sub_goal
34
35            # subject variable binding
36            if rule[depth][1] not in variable_substitution.keys():
37                variable_substitution[rule[depth][1]] = list(set(sub_goal[rule[depth][1]].values))
38            # object variable binding
39            if rule[depth][2] not in variable_substitution.keys():
40                variable_substitution[rule[depth][2]] = list(set(sub_goal[rule[depth][2]].values))
41
42        else :
43            subj_cVar_index = set(itertools.chain.from_iterable(
44                                 list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
45            obj_cVar_index = set(itertools.chain.from_iterable(
46                                 list(map(lambda x : KG_index.get(x).get(cVar_position[1]), unified_cVar[1]))))
47            sub_goal = KG.loc[subj_cVar_index &obj_cVar_index]
48            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
49            rComp_substitution[rule[depth]] = sub_goal
50        depth += 1
51        return unification(goal, rule, KG, KG_index, rule_num, depth,
52                           variable_substitution, rComp_substitution,rule_structure)
```

cVar _position = [subj, obj]

unified _cVar = [[USA, NEWYORK, HOMMER]
[USA]]

KG Index(Dict)	
Key(Str)	Value(Dict)
BART	{ subj : [0,1,2], obj : [ ] }
HOMMER	{ subj : [3, 4], obj : [2] }
NEWYORK	{ subj : [5], obj : [1, 3] }
USA	{ subj : [], obj : [0, 4, 5] }

KG _index.get(USA, NEWYORK, HOMMER).get(subj)
KG _index.get(USA).get(obj)

subj _cVar _index = [[3,4],[5]]
obj _cVar _index = [[0,4,5]]

itertools .chain .from _iterable()

subj _cVar _index = { 3,4,5 }
obj _cVar _index = { 0,4,5 }

Unification Case3 (depth = 2)

- **Body Unification**
  - Index정보와 KG로부터 Unify될 트리플을 가져와 해당 Component의 Substitution생성
  - Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```
19    else :
20        common_variable = []
21        current_body = rule_structure[rule_structure['rule_number'] == rule_num].iloc[0,depth]
22        current_variable = list(current_body.keys())
23        common_variable = [variable for variable in current_variable if variable in variable_substitution]
24        cVar_position = list(map(current_body.get, common_variable))
25        unified_cVar= list(map(variable_substitution.get, common_variable))
26
27        if len(common_variable) == 1:
28            cVar_index = set(itertools.chain.from_iterable(
29                             list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
30
31            sub_goal = KG.loc[cVar_index]
32            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
33            rComp_substitution[rule[depth]] = sub_goal
34
35            # subject variable binding
36            if rule[depth][1] not in variable_substitution.keys():
37                variable_substitution[rule[depth][1]] = list(set(sub_goal[rule[depth][1]].values))
38            # object variable binding
39            if rule[depth][2] not in variable_substitution.keys():
40                variable_substitution[rule[depth][2]] = list(set(sub_goal[rule[depth][2]].values))
41
42        else :
43            subj_cVar_index = set(itertools.chain.from_iterable(
44                                  list(map(lambda x : KG_index.get(x).get(cVar_position[0]), unified_cVar[0]))))
45            obj_cVar_index = set(itertools.chain.from_iterable(
46                                  list(map(lambda x : KG_index.get(x).get(cVar_position[1]), unified_cVar[1]))))
47            sub_goal = KG.loc[subj_cVar_index &obj_cVar_index]
48            sub_goal.columns = [rule[depth][0], rule[depth][1], rule[depth][2]]
49            rComp_substitution[rule[depth]] = sub_goal
50        depth += 1
51        return unification(goal, rule, KG, KG_index, rule_num, depth,
52                           variable_substitution, rComp_substitution,rule_structure)
```

[USA, NEWYORK, HOMMER]

subj _cVar _index = { 3,4,5 }

&

obj _cVar _index = { 0,4,5 }

[USA]

{ 4,5 }

Knowledge Graph(DataFrame)

Index	pred	subj	obj
0	nationality	BART	USA
1	placeOfBirth	BART	NEWYORK
2	hasFather	BART	HOMMER
3	placeOfBirth	HOMMER	NEWYORK
4	nationality	HOMMER	USA
5	locatedIn	NEWYORK	USA

depth = 2

rule : [(#1,X, Y) , (#2,X, Z) , (#3,Z, Y) , 2]

Rule Component Substitution(Dict)

Key(Tuple)	Value(DataFrame)		
(#1, X, Y)	**#1**	**X**	**Y**
	nationality	BART	USA
(#2, X, Z)	**#2**	**X**	**Z**
	nationality	BART	USA
	placeOfBirth	BART	NEWYORK
	hasFather	BART	HOMMER
(#3, Z, Y)	**#3**	**Z**	**Y**
	locatedIn	NEWYORK	USA
	nationality	HOMMER	USA

Unification Case2 (depth = 3)

- **Call Proof Path Completion()**
  - 생성된 Rule component substitution과 Rule을 입력으로 Relation group을 생성
  - Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```
15      if depth == len(rule)-1:
16          return proof_path_completion(rComp_substitution, rule)
```

Rule Component Substitution(Dict)			
Key(Tuple)	Value(DataFrame)		
(#1, X, Y)	**#1**	**X**	**Y**
	nationality	BART	USA
(#2, X, Z)	**#2**	**X**	**Z**
	nationality	BART	USA
	placeOfBirth	BART	NEWYORK
	hasFather	BART	HOMMER
(#3, Z, Y)	**#3**	**Z**	**Y**
	locatedIn	NEWYORK	USA
	nationality	HOMMER	USA

Rule : [(#1,X, Y) , (#2,X, Z) , (#3,Z,Y) , 2]

Proof Path Completion()

Relation Group

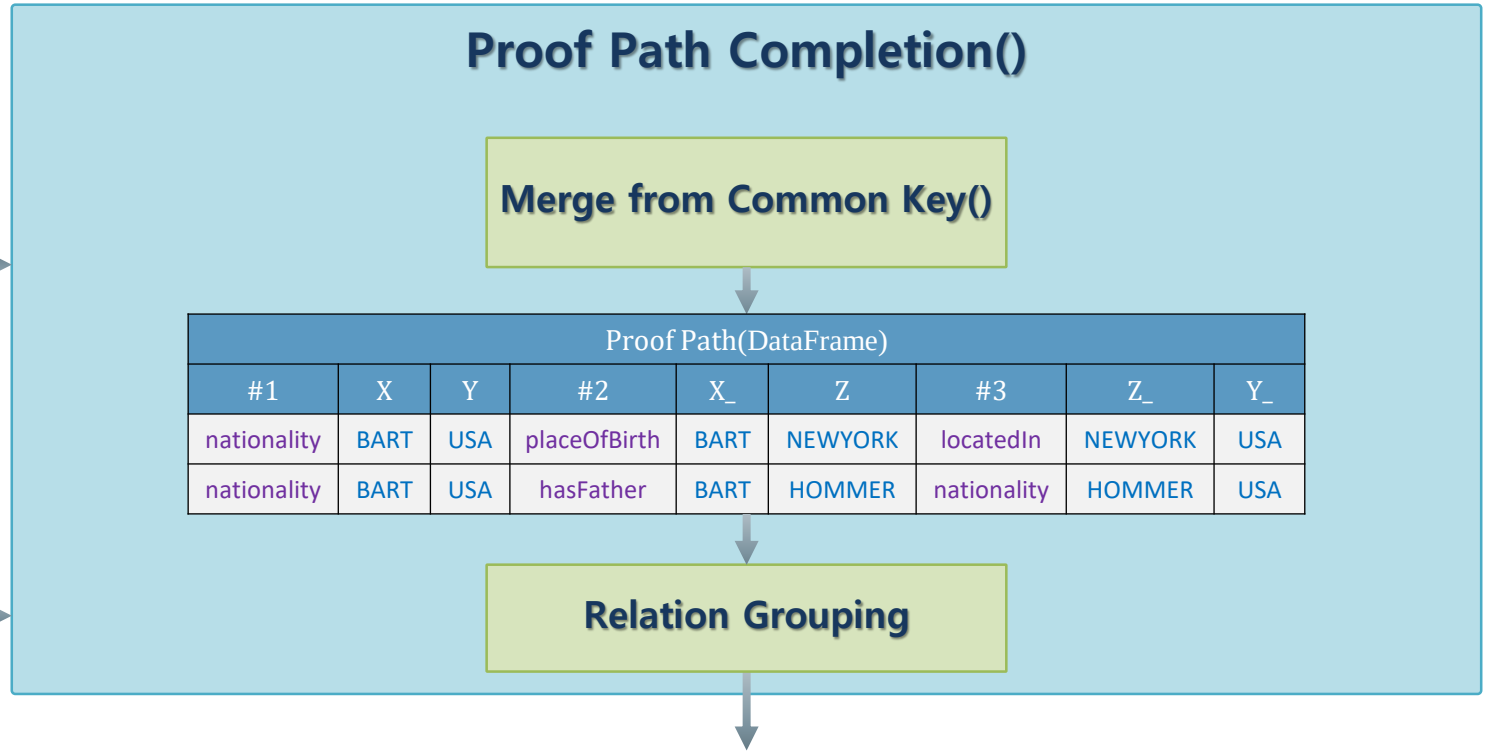
[[(#1, nationality), (#2, placeOfBirth), (#3, locatedIn)],
[(#1, nationality), (#2, hasFather), (#3, nationality)]]

Proof Path Completion Overview

- **Proof Path Completion()**
  - 인접한 두 Rule component간의 Common variable을 Key로 Join하여 Proof path를 생성
  - 생성된 Proof path에서 KG relation과 Rule relation을 Grouping하여 Relation group을 생성

Rule Component Substitution(Dict)			
Key(Tuple)	Value(DataFrame)		
(#1, X, Y)	**#1**	**X**	**Y**
	nationality	BART	USA
(#2, X, Z)	**#2**	**X**	**Z**
	nationality	BART	USA
	placeOfBirth	BART	NEWYORK
	hasFather	BART	HOMMER
(#3, Z, Y)	**#3**	**Z**	**Y**
	locatedIn	NEWYORK	USA
	nationality	HOMMER	USA

Rule : [(#1,X, Y) , (#2,X, Z) , (#3,Z,Y) , 2]

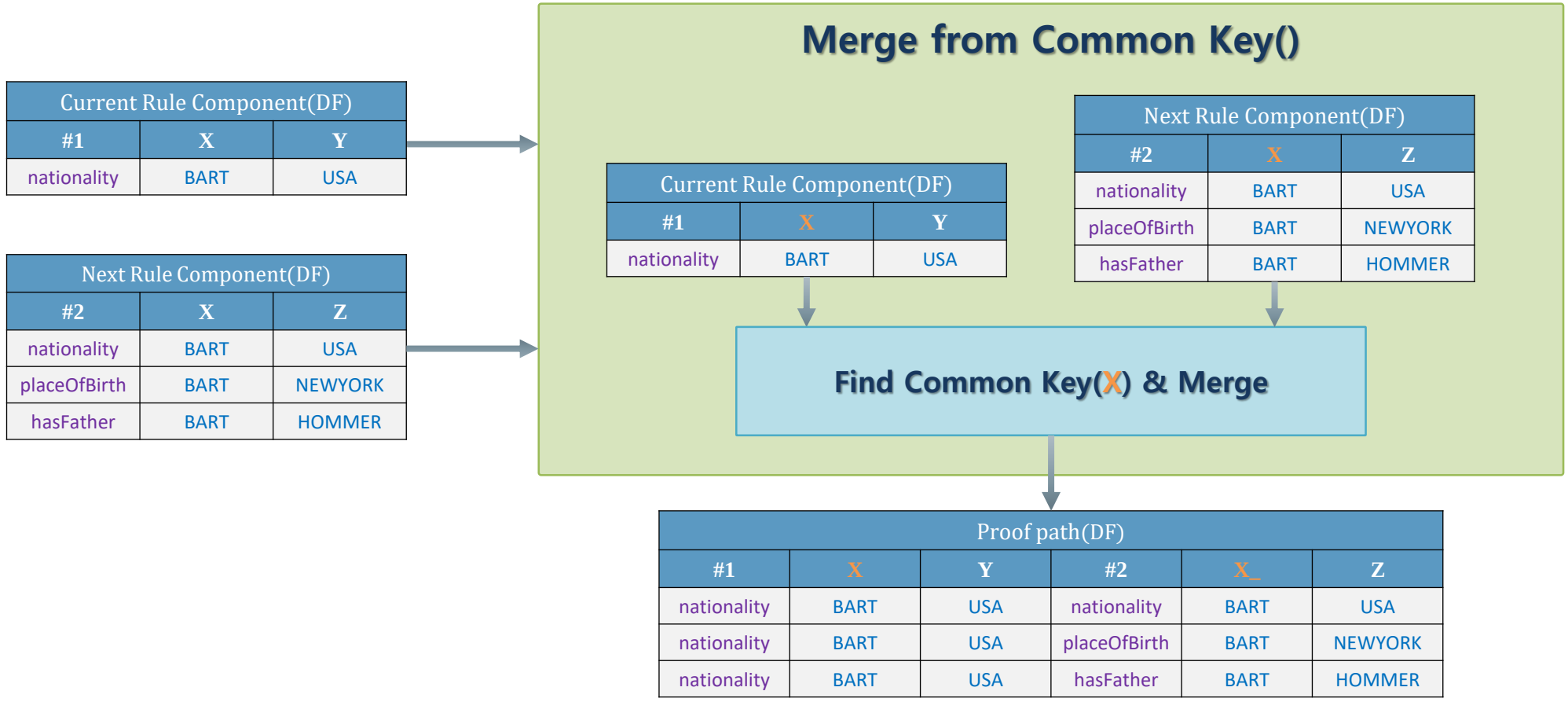


Relation Group

[[(#1, nationality), (#2, placeOfBirth), (#3, locatedIn)],
[(#1, nationality), (#2, hasFather), (#3, nationality)]]

Merge from Common Key Overview

- **Merge from Common Key()**
  - 두 Rule component간의 Common variable을 찾아 Join하여 Proof path 생성
  - Given Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)



Current Rule Component(DF)

#1	X	Y
nationality	BART	USA

Next Rule Component(DF)

#2	X	Z
nationality	BART	USA
placeOfBirth	BART	NEWYORK
hasFather	BART	HOMMER

Current Rule Component(DF)

#1	X	Y
nationality	BART	USA

Next Rule Component(DF)

#2	X	Z
nationality	BART	USA
placeOfBirth	BART	NEWYORK
hasFather	BART	HOMMER

Proof path(DF)

#1	X	Y	#2	X_	Z
nationality	BART	USA	nationality	BART	USA
nationality	BART	USA	placeOfBirth	BART	NEWYORK
nationality	BART	USA	hasFather	BART	HOMMER

Proof Path Completion (depth = 0)

- **Load DataFrame**
  - Rule component substitution으로 부터 Join될 Data를 Load
  - Given Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```python
def proof_path_completion(rComp_substitution, rule):

    for depth in range(len(rule)-2):
        if depth == 0:
            #Load dataframes
            current_rComp = rComp_substitution[rule[depth]]
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

        else:
            #Load dataframes
            current_rComp = partial_proof_path
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

    #remove duplicate  Adjacent triple
    proof_path = partial_proof_path
    proof_path = check_duplicate(rule, proof_path)

    #stackoverflow How to get that result without for loop (python)
    relation_group = proof_path[[r[0] for r in rule[:-1]]].drop_duplicates()#
           .apply(lambda x: list(zip([r[0] for r in rule[:-1]], x)), axis=1).values.tolist()

    return relation_group
```

depth = 0 depth = 1

rule : [(#1,X, Y) , (#2,X, Z) , (#3,Z, Y) , 2]

Rule Component Substitution(Dict)			
Key(Tuple)	Value(DataFrame)		
(#1, X, Y)	**#1**	**X**	**Y**
	nationality	BART	USA
(#2, X, Z)	**#2**	**X**	**Z**
	nationality	BART	USA
	placeOfBirth	BART	NEWYORK
	hasFather	BART	HOMMER
(#3, Z, Y)	**#3**	**Z**	**Y**
	locatedIn	NEWYORK	USA
	nationality	HOMMER	USA

Load DataFrame

Current Rule Component(DF)		
#1	**X**	**Y**
nationality	BART	USA

Next Rule Component(DF)		
#2	**X**	**Z**
nationality	BART	USA
placeOfBirth	BART	NEWYORK
hasFather	BART	HOMMER

Proof Path Completion (depth = 0)

- **Merge from Common Key()**
  - Load된 Data를 Join하여 Partial proof path생성
  - Given Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```python
def proof_path_completion(rComp_substitution, rule):

    for depth in range(len(rule)-2):
        if depth == 0:
            #Load dataframes
            current_rComp = rComp_substitution[rule[depth]]
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

        else:
            #Load dataframes
            current_rComp = partial_proof_path
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

    #remove duplicate  Adjacent triple
    proof_path = partial_proof_path
    proof_path = check_duplicate(rule, proof_path)

    #stackoverflow How to get that result without for loop (python)
    relation_group = proof_path[[r[0] for r in rule[:-1]]].drop_duplicates()#
          .apply(lambda x: list(zip([r[0] for r in rule[:-1]], x)), axis=1).values.tolist()

    return relation_group
```

```python
#stackoverflow How to insert a dropped join key column from Dataframe join in order
def merge_from_common_key(current_rComp, next_rComp):

    key = current_rComp.columns.intersection(next_rComp.columns)
    new_key = key + '_'
    d = dict(zip(key, new_key))

    proof_path = pd.merge(current_rComp, next_rComp.rename(columns=d), how='inner',
                          left_on=key.tolist(), right_on=new_key.tolist())

    return proof_path
```

Current Rule Component(DF)		
#1	X	Y
nationality	BART	USA

Next Rule Component(DF)		
#2	X	Z
nationality	BART	USA
placeOfBirth	BART	NEWYORK
hasFather	BART	HOMMER

intersection()

key = Index ([X])

key + '_'

new _ key = Index ([X_])

dict(zip(key, new _ key))

key _ dict = { X : X_ }

Proof Path Completion (depth = 0)

- **Merge from Common Key()**
  - Load된 Data를 Join하여 Partial proof path생성
  - Given Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```python
def proof_path_completion(rComp_substitution, rule):

    for depth in range(len(rule)-2):
        if depth == 0:
            #Load dataframes
            current_rComp = rComp_substitution[rule[depth]]
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

        else:
            #Load dataframes
            current_rComp = partial_proof_path
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

    #remove duplicate  Adjacent triple
    proof_path = partial_proof_path
    proof_path = check_duplicate(rule, proof_path)

    #stackoverflow How to get that result without for loop (python)
    relation_group = proof_path[[r[0] for r in rule[:-1]]].drop_duplicates()#
          .apply(lambda x: list(zip([r[0] for r in rule[:-1]], x)), axis=1).values.tolist()

    return relation_group
```

```python
#stackoverflow How to insert a dropped join key column from Dataframe join in order
def merge_from_common_key(current_rComp, next_rComp):

    key = current_rComp.columns.intersection(next_rComp.columns)
    new_key = key + '_'
    d = dict(zip(key, new_key))

    proof_path = pd.merge(current_rComp, next_rComp.rename(columns=d), how='inner',
                          left_on=key.tolist(), right_on=new_key.tolist())

    return proof_path
```

key = Index ([X])

new _ key = Index ([X_])

Current Rule Component(DF)		
#1	X	Y
nationality	BART	USA

Next Rule Component(DF)		
#2	X	Z
nationality	BART	USA
placeOfBirth	BART	NEWYORK
hasFather	BART	HOMMER

key _ dict = { X : X_ }

Next Rule Component(DF)		
#2	X_	Z
nationality	BART	USA
placeOfBirth	BART	NEWYORK
hasFather	BART	HOMMER

Merge(left_on[X], right_on[X_])

Partial Proof path(DF)					
#1	X	Y	#2	X_	Z
nationality	BART	USA	nationality	BART	USA
nationality	BART	USA	placeOfBirth	BART	NEWYORK
nationality	BART	USA	hasFather	BART	HOMMER

Proof Path Completion (depth = 1)

- **Load DataFrame & Merge from Common Key()**
  - Partial proof path와 다음 Rule body를 Load하여 Join수행
  - Given Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```python
def proof_path_completion(rComp_substitution, rule):

    for depth in range(len(rule)-2):
        if depth == 0:
            #Load dataframes
            current_rComp = rComp_substitution[rule[depth]]
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

        else:
            #Load dataframes
            current_rComp = partial_proof_path
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

    #remove duplicate  Adjacent triple
    proof_path = partial_proof_path
    proof_path = check_duplicate(rule, proof_path)

    #stackoverflow How to get that result without for loop (python)
    relation_group = proof_path[[r[0] for r in rule[:-1]]].drop_duplicates()#
          .apply(lambda x: list(zip([r[0] for r in rule[:-1]], x)), axis=1).values.tolist()

    return relation_group
```

depth = 2

rule : [(#1,X, Y) , (#2,X, Z) , (#3,Z, Y) , 2]

Rule Component Substitution(Dict)

Key(Tuple)	Value(DataFrame)		
(#1, X, Y)	#1	X	Y
	nationality	BART	USA
(#2, X, Z)	#2	X	Z
	nationality	BART	USA
	placeOfBirth	BART	NEWYORK
	hasFather	BART	HOMMER
(#3, Z, Y)	#3	Z	Y
	locatedIn	NEWYORK	USA
	nationality	HOMMER	USA

Current Rule Component (Partial Proof path)

#1	X	Y	#2	X_	Z
nationality	BART	USA	nationality	BART	USA
nationality	BART	USA	placeOfBirth	BART	NEWYORK
nationality	BART	USA	hasFather	BART	HOMMER

Next Rule Component

#3	Z_	Y_
locatedIn	NEWYORK	USA
nationality	HOMMER	USA

Merge(left_on[Z , Y], right_on[Z_ , Y_])

Partial Proof path(DataFrame)

#1	X	Y	#2	X_	Z	#3	Z_	Y_
nationality	BART	USA	placeOfBirth	BART	NEWYORK	locatedIn	NEWYORK	USA
nationality	BART	USA	hasFather	BART	HOMMER	nationality	HOMMER	USA

Proof Path Completion

- **Check duplicate()**
  - 중복된 트리플을 포함하는 Proof path 제거
  - Given Rule 0 : #1(X, Y) :- #2(X, Z), #3(W, Z), #4(W, Y)

```python
def proof_path_completion(rComp_substitution, rule):

    for depth in range(len(rule)-2):
        if depth == 0:
            #Load dataframes
            current_rComp = rComp_substitution[rule[depth]]
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

        else:
            #Load dataframes
            current_rComp = partial_proof_path
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

    #remove duplicate  Adjacent triple
    proof_path = partial_proof_path
    proof_path = check_duplicate(rule, proof_path)

    #stackoverflow How to get that result without for loop (python)
    relation_group = proof_path[[r[0] for r in rule[:-1]]].drop_duplicates()#
          .apply(lambda x: list(zip([r[0] for r in rule[:-1]], x)), axis=1).values.tolist()

    return relation_group
```

```python
#stackoverflow Check multi-value duplication in pandas
def check_duplicate(proof_path):

    triple_size = 3
    column_len = len(proof_path.columns)
    num_unique_triples = int(column_len/triple_size)

    proof_path['n_unique_triples'] = \
        proof_path.apply(lambda row: len(set([tuple(row[ i*triple_size : (i+1)*triple_size ])
                                              for i in range(num_unique_triples)])), axis=1)

    proof_path = proof_path[proof_path.n_unique_triples == num_unique_triples]

    return proof_path
```

Proof path(DataFrame)											
#1	X	Y	#2	X_	Z	#3	W	Z_	#4	W_	Y_
p1	A	B	p2	A	C	p2	A	C	p3	A	B
p1	A	B	p2	A	C	p3	A	C	p1	A	B
p1	A	B	p2	A	D	p3	E	D	p4	E	B

$[\ (p1, A, B),\ (p2, A, C),\ (p2, A, C),\ (p3, A, B)\ ]$
$[\ (p1, A, B),\ (p2, A, C),\ (p3, A, C),\ (p1, A, B)\ ]$
$[\ (p1, A, B),\ (p2, A, D),\ (p3, E, D),\ (p4, E, B)\ ]$

set()

$\{\ (p1, A, B),\ (p2, A, C),\ (p3, A, B)\ \}$
$\{\ (p1, A, B),\ (p2, A, C),\ (p3, A, C)\ \}$
$\{\ (p1, A, B),\ (p2, A, D),\ (p3, E, D),\ (p4, E, B)\ \}$

len() → 3, 3, 4

Proof Path Completion

- **Check duplicate()**
  - 중복된 트리플을 포함하는 Proof path 제거
  - Given Rule 0 : #1(X, Y) :- #2(X, Z), #3(W, Z), #4(W, Y)

```python
def proof_path_completion(rComp_substitution, rule):

    for depth in range(len(rule)-2):
        if depth == 0:
            #Load dataframes
            current_rComp = rComp_substitution[rule[depth]]
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

        else:
            #Load dataframes
            current_rComp = partial_proof_path
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

    #remove duplicate  Adjacent triple
    proof_path = partial_proof_path
    proof_path = check_duplicate(rule, proof_path)

    #stackoverflow How to get that result without for loop (python)
    relation_group = proof_path[[r[0] for r in rule[:-1]]].drop_duplicates()#
          .apply(lambda x: list(zip([r[0] for r in rule[:-1]], x)), axis=1).values.tolist()

    return relation_group
```

```python
#stackoverflow Check multi-value duplication in pandas
def check_duplicate(proof_path):

    triple_size = 3
    column_len = len(proof_path.columns)
    num_unique_triples = int(column_len/triple_size)

    proof_path['n_unique_triples'] = \
        proof_path.apply(lambda row: len(set([tuple(row[ i*triple_size : (i+1)*triple_size ])
                                              for i in range(num_unique_triples)])), axis=1)

    proof_path = proof_path[proof_path.n_unique_triples == num_unique_triples]

    return proof_path
```

Proof path(DataFrame)

#1	X	Y	#2	X_	Z	#3	W	Z_	#4	W_	Y_
p1	A	B	p2	A	C	p2	A	C	p3	A	B
p1	A	B	p2	A	C	p3	A	C	p1	A	B
p1	A	B	p2	A	D	p3	E	D	p4	E	B

3
3
4

Proof path(DataFrame)

#1	X	Y	#2	X_	Z	#3	W	Z_	#4	W_	Y_	#unique
p1	A	B	p2	A	C	p2	A	C	p3	A	B	3
p1	A	B	p2	A	C	p3	A	C	p1	A	B	3
p1	A	B	p2	A	D	p3	E	D	p4	E	B	4

proof_path[proof_path.n_unique_triples == num_unique_triples]

Proof path(DataFrame)

#1	X	Y	#2	X_	Z	#3	W	Z_	#4	W_	Y_	#unique
p1	A	B	p2	A	D	p3	E	D	p4	E	B	4

Proof Path Completion

- **Relation Grouping**
  - 생성된 Proof path에서 Relation정보만을 취하여 Relation proof path생성
  - Given Rule 0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)

```python
def proof_path_completion(rComp_substitution, rule):

    for depth in range(len(rule)-2):
        if depth == 0:
            #Load dataframes
            current_rComp = rComp_substitution[rule[depth]]
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

        else:
            #Load dataframes
            current_rComp = partial_proof_path
            next_rComp = rComp_substitution[rule[depth+1]]
            #merge from common key
            partial_proof_path = merge_from_common_key(current_rComp, next_rComp)

    #remove duplicate  Adjacent triple
    proof_path = partial_proof_path
    proof_path = check_duplicate(rule, proof_path)

    #stackoverflow How to get that result without for loop (python)
    relation_group = proof_path[[r[0] for r in rule[:-1]]].drop_duplicates()#
          .apply(lambda x: list(zip([r[0] for r in rule[:-1]], x)), axis=1).values.tolist()

    return relation_group
```

Rule : [(#1,X, Y) , (#2,X, Z) , (#3,Z, Y) , 2]

Proof path(DataFrame)

#1	X	Y	#2	X_	Z	#3	Z_	Y_
nationality	BART	USA	placeOfBirth	BART	NEWYORK	locatedIn	NEWYORK	USA
nationality	BART	USA	hasFather	BART	HOMMER	nationality	HOMMER	USA

proof_path[[r[0] for r in rule[:-1]]].drop_duplicates()

Relation Group(DF)

#1	#2	#3
nationality	placeOfBirth	locatedIn
nationality	hasFather	nationality

[nationality, placeOfBirth, locatedIn]

.apply(lambda x: list(zip([r[0] for r in rule[:-1]], x)), axis=1)

[#1, #2, #3]

Relation Group(Serise)

0 [(#1, nationality), (#2, placeOfBirth), (#3, locatedIn)]
1 [(#1, nationality), (#2, hasFather), (#3, nationality)]

.values.tolist()

Relation Group(List)

[[(#1, nationality), (#2, placeOfBirth), (#3, locatedIn)],
  [(#1, nationality), (#2, hasFather), (#3, nationality)]]

Relation Path 생성

- **n개의 Query와 m개의 Rule Template을 이용하여 Relation Path 생성**
  - Query Q : { nationality(BART, USA), … hasFather(BART, HOMMER) }
  - Rule Template RT : { #1(X, Y) :- #2(X, Z), #3(Z, Y), …, #1(X, Y) :- #2(X, Y) }의 형태임

Relation Path	Query \ Rule Template	RT_0 #1(X, Y) :- #2(X, Z), #3(Z, Y)	…	RT_m #1(X, Y) :- #2(X, Y)
	Q_0	Encoded Augmented Unified Relation path by Q_0 and RT_0	…	Encoded Augmented Unified Relation path by Q_0 and RT_m
	…	…	…	…
	Q_n	Encoded Augmented Unified Relation path by Q_n and RT_0	…	Encoded Augmented Unified Relation path by Q_n and RT_m

- **예 : Encoded Augmented Unified Relation path by Q_0 and RT_0**
  - Q_0 : nationality(BART, USA)
  - RT_0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)
  - Neg per Pos = 1
  - Augment Number = 2

	Augment 0	Augment 1
Positive	$[[id_{nationality}, id_{placeOfBirth}, id_{locatedIn}],$ $[id_{nationality}, id_{hasFather}, id_{nationality}],$	$[id_{nationality}, id_{placeOfBirth}, id_{locatedIn}]],$ $[id_{nationality}, id_{hasFather}, id_{nationality}]],$
Negative	$[[id_{locatedIn}, id_{nationality}, id_{hasFather}],$ $[[id_{placeOfBirth}, id_{locatedIn}, id_{hasFather}],$	$[id_{locatedIn}, id_{nationality}, id_{hasFather}]],$ $[id_{placeOfBirth}, id_{locatedIn}, id_{hasFather}]]$

Rule Template Path 생성

- **n개의 Query와 m개의 Rule Template을 이용하여 Relation Path 생성**
  - Query Q : { nationality(BART, USA), … hasFather(BART, HOMMER) }
  - Rule Template RT : { #1(X, Y) :- #2(X, Z), #3(Z, Y), …, #1(X, Y) :- #2(X, Y) }의 형태임

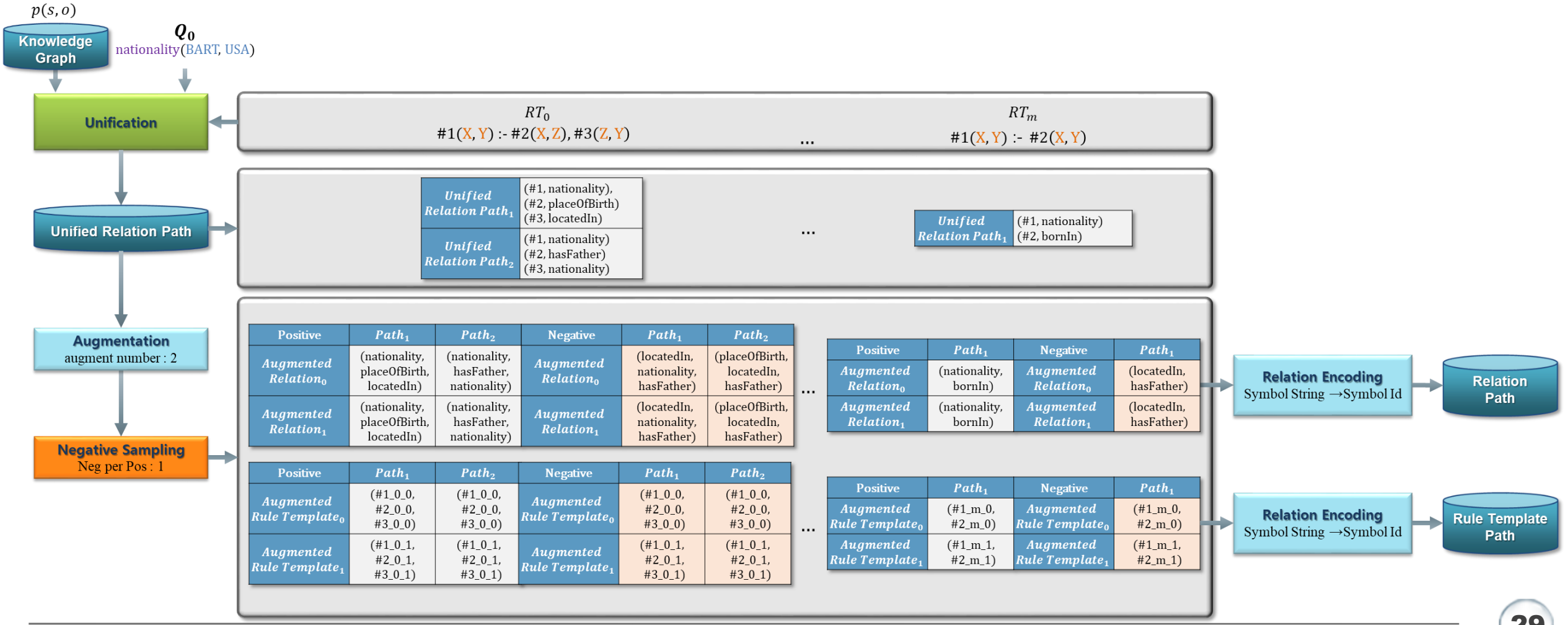
Rule Template Path	Query \ Rule Template	RT_0 #1(X, Y) :- #2(X, Z), #3(Z, Y)	…	RT_m #1(X, Y) :- #2(X, Y)
	Q_0	Encoded Augmented Unified Rule Template path by Q_0 and RT_0	…	Encoded Augmented Unified Rule Template path by Q_0 and RT_m
	…	…	…	…
	Q_n	Encoded Augmented Unified Rule Template path by Q_n and RT_0	…	Encoded Augmented Unified Rule Template path by Q_n and RT_m

- **예 : Encoded Augmented Unified Rule Template path by Q_0 and RT_0**
  - Q_0 : nationality(BART, USA)
  - RT_0 : #1(X, Y) :- #2(X, Z), #3(Z, Y)
  - Neg per Pos = 1
  - Augment Number = 2

	Augment 0	Augment 1
Positive	[[$id_{\#1\_0\_0}$, $id_{\#2\_0\_0}$, $id_{\#3\_0\_0}$],	[$id_{\#1\_0\_1}$, $id_{\#2\_0\_1}$, $id_{\#3\_0\_1}$]],
	[[$id_{\#1\_0\_0}$, $id_{\#2\_0\_0}$, $id_{\#3\_0\_0}$],	[$id_{\#1\_0\_1}$, $id_{\#2\_0\_1}$, $id_{\#3\_0\_1}$]],
Negative	[[$id_{\#1\_0\_0}$, $id_{\#2\_0\_0}$, $id_{\#3\_0\_0}$],	[$id_{\#1\_0\_1}$, $id_{\#2\_0\_1}$, $id_{\#3\_0\_1}$]],
	[[$id_{\#1\_0\_0}$, $id_{\#2\_0\_0}$, $id_{\#3\_0\_0}$],	[$id_{\#1\_0\_1}$, $id_{\#2\_0\_1}$, $id_{\#3\_0\_1}$]]]

Neuro Symbolic Unification (1개의 Query, m개의 Rule Template)

- **1개의 Query와 m개의 Rule Template을 이용하여 Relation Path및 Rule Template Path 생성과정**
  - 하나의 Query로부터 생성되는 여러 개의 Relation Path를 학습하기 위한 Augmentation수행
  - 보다 효과적인 유사도 학습을 위해 Negative Sampling을 통해 Negative Path생성



Unified Relation Path (1개의 Query, 1개의 Rule Template)

- **Unified Relation Path 생성을 위한 Unification 과정**

Knowledge Graph(DataFrame)			
Index	**pred**	**subj**	**obj**
0	nationality	BART	USA
1	placeOfBirth	BART	NEWYORK
2	hasFather	BART	HOMMER
3	placeOfBirth	HOMMER	NEWYORK
4	nationality	HOMMER	USA
5	locatedIn	NEWYORK	USA

Query(DataFrame)			
Index	**pred**	**subj**	**obj**
0	nationality	BART	USA
1	hasFather	BART	HOMMER
...	...	...	...

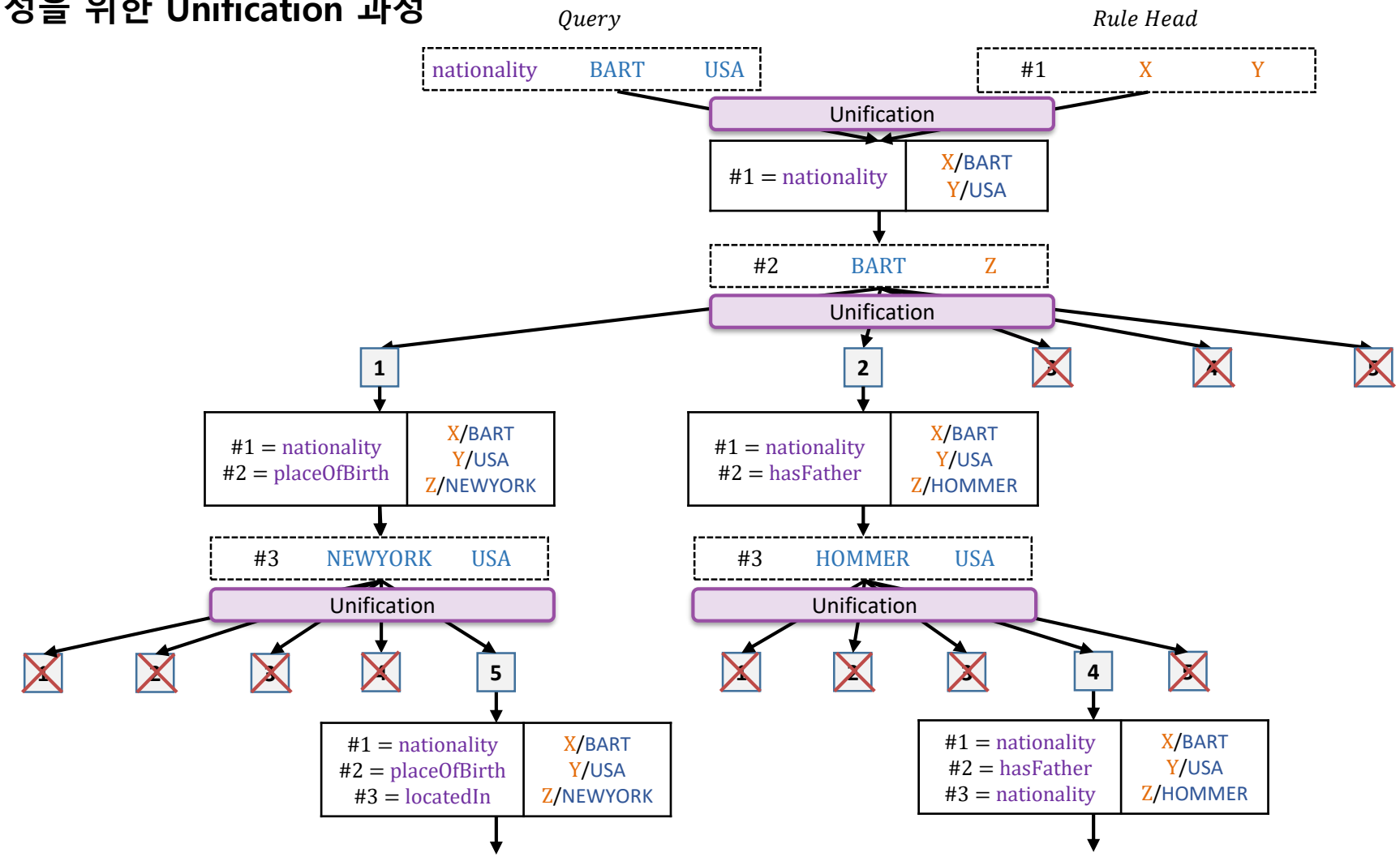
Rule Template

$\#1(X, Y) :- \#2(X, Z), \#3(Z, Y).$ 2

$\#1(X, Y) :- \#2(Z, X), \#3(Z, Y).$ 2

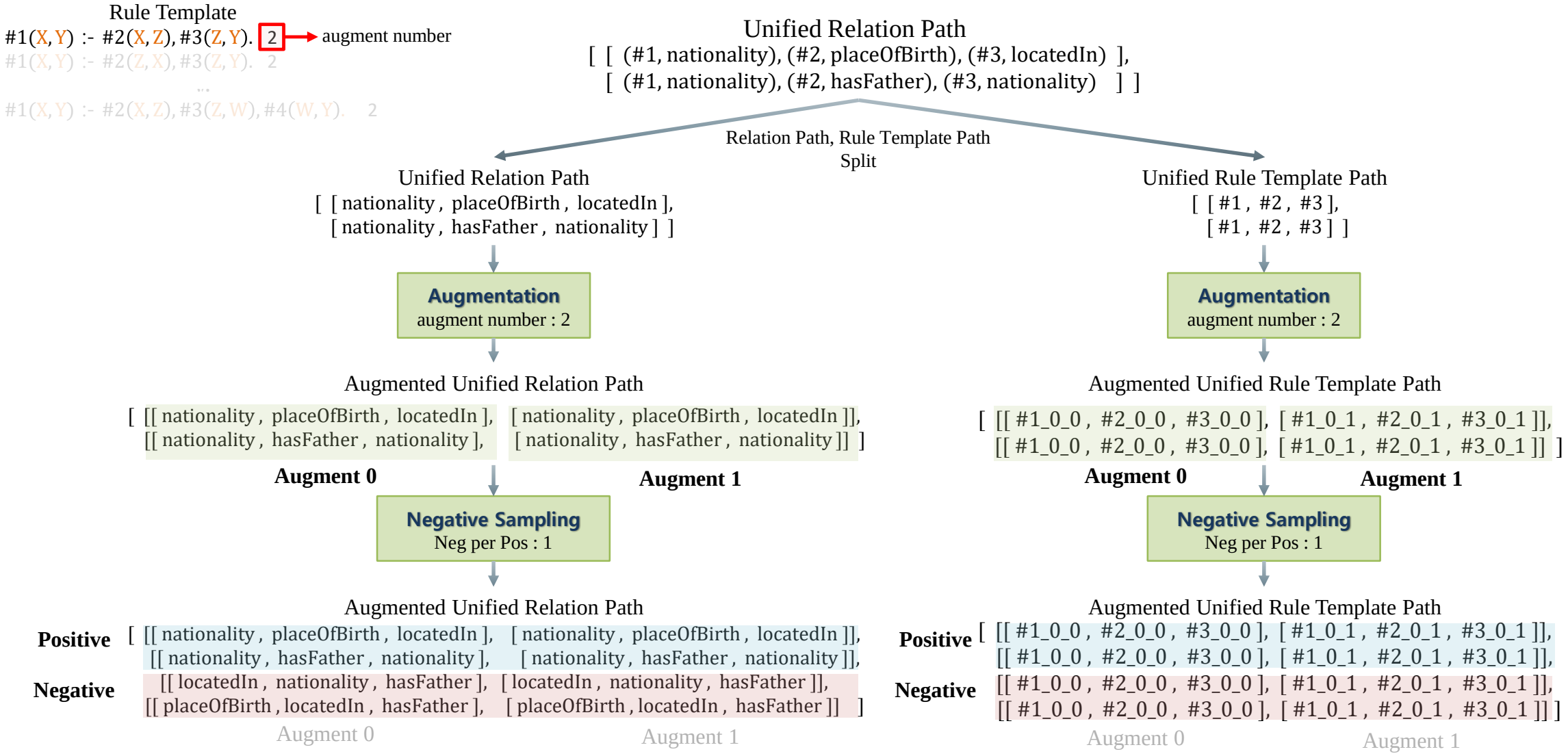
...

$\#1(X, Y) :- \#2(X, Z), \#3(Z, W), \#4(W, Y).$ 2



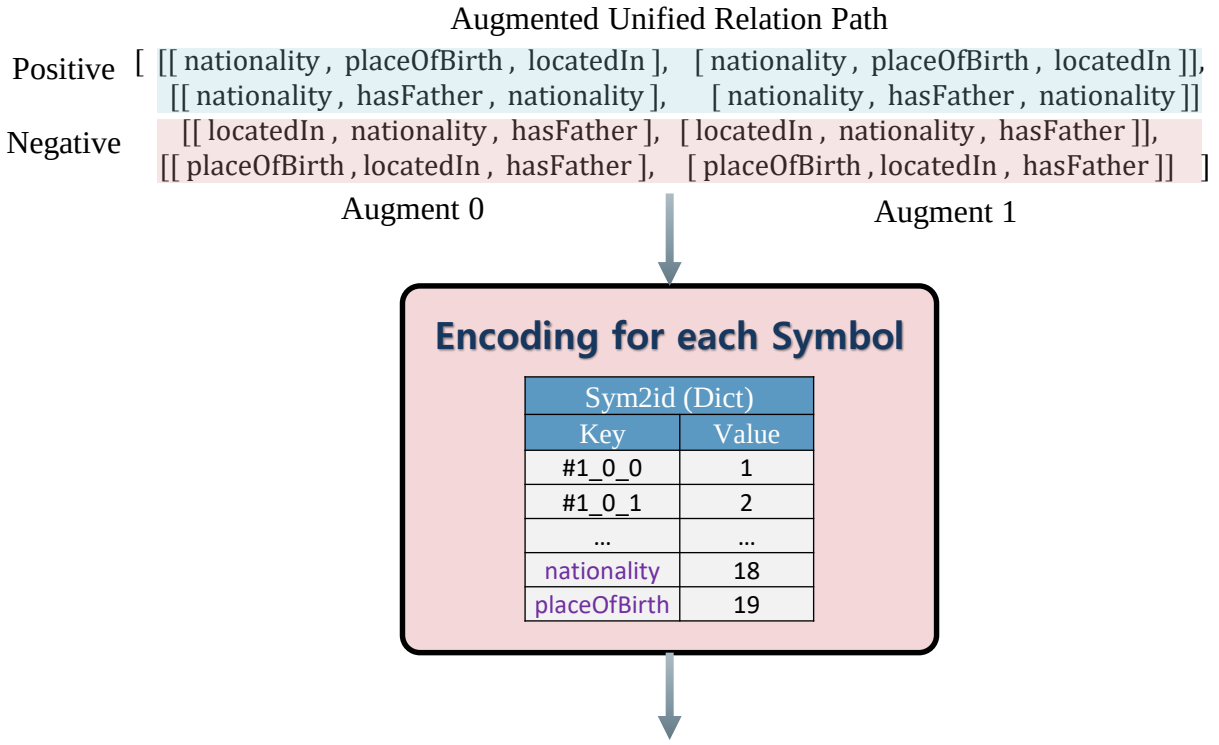
Augmented Unified Relation Path (1개의 Query, 1개의 Rule Template)

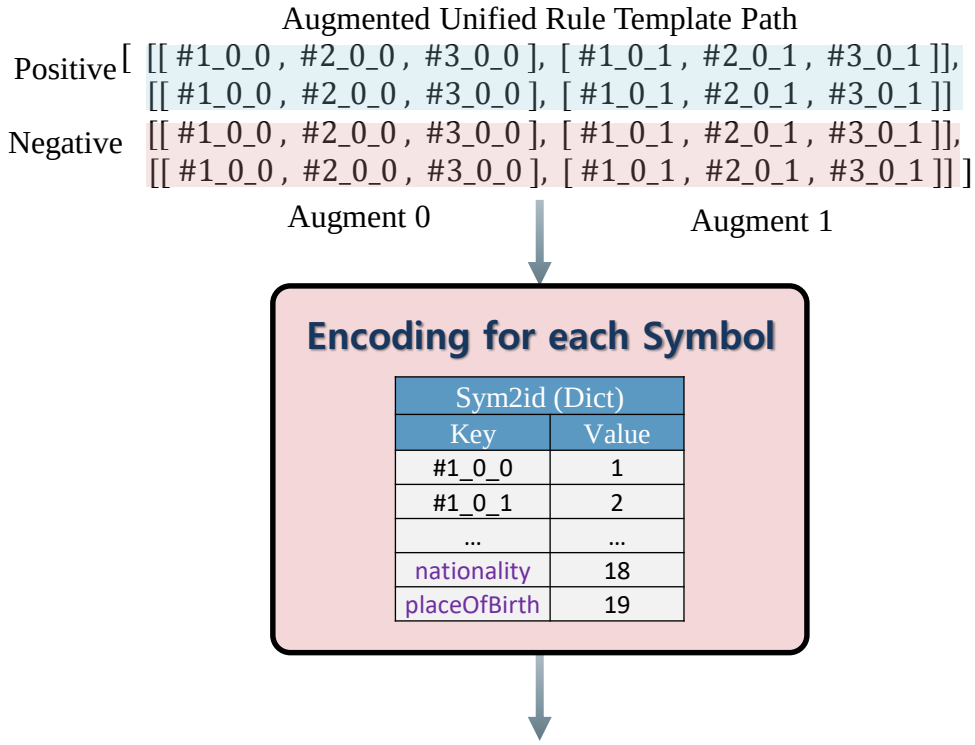
- **Unified Relation Path의 Augmentation 및 Negative Sampling 과정**



Encoded Augmented Unified Relation Path (1개의 Query, 1개의 Rule Template)

- **Augmented Unified Relation Path의 Encoding 과정**



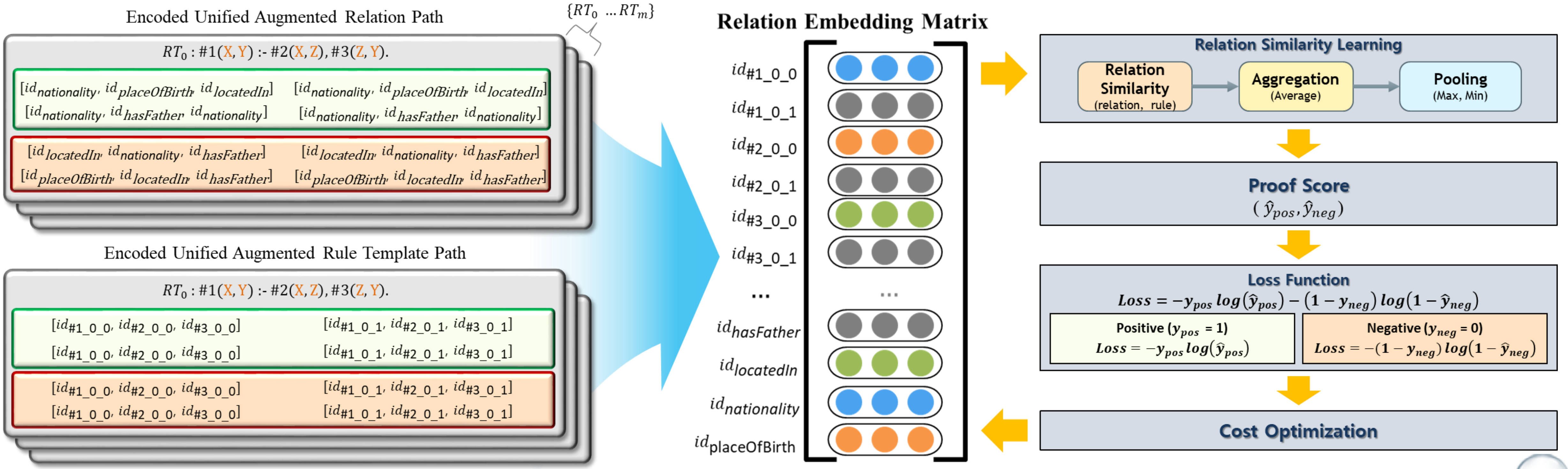


Relation Similarity Learning (1개의 Query, m개의 Rule Template)

- **주어진 Query에 대한 Relation Similarity 학습 방법**
  - 주어진 Query의 m개의 Rule Template으로 부터 1개의 Positive Proof Score, 1개의 Negative Proof Score를 산출
  - Pooling을 통해 선택된 하나의 Negative Path, Positive Path의 Relation에 대한 Embedding vector update

Query Q_0
nationality(BART, USA)

Rule Template
RT_0 : #1(X, Y) :- #2(X, Z), #3(Z, Y). 2
...
RT_m: #1(X, Y) :- #2(X, Y) 2



Relation Similarity Learning (1개의 Query, m개의 Rule Template)

- 주어진 Query에 대한 Proof Score 산출 방법

Query Q_0
nationality(BART, USA)

Rule Template

RT_0 : #1(X, Y) :- #2(X, Z), #3(Z, Y). 2

...

RT_m: #1(X, Y) :- #2(X, Y) 2

Positive	$Path_1$ To RT_0
Augmented $Relation_0$	$(id_{nationality}, id_{placeOfBirth}, id_{locatedIn})$

Relation Embedding : re

$re^0_{1,1}$ $re^0_{1,2}$ $re^0_{1,3}$ $re^{\#augment}_{\#path,\#relation}$

Positive	$Path_1$ To RT_0
Augmented Rule $Template_0$	$(id_{\#1\_0\_0}, id_{\#2\_0\_0}, id_{\#3\_0\_0})$

Rule Template Embedding : rte

$rte^0_{1,1}$ $rte^0_{1,2}$ $rte^0_{1,3}$ $rte^{\#augment}_{\#path,\#relation}$

STEP1 : Relation Embedding
Symbol Id → Symbol Embedding
$id_{nationality}$ → $re_{nationality}$

STEP2 : Relation Similarity
$sim^a_{p,r} = \mathrm{sim}(re^a_{p,r}, rte^a_{p,r})$

STEP3 : Aggregation (Average)
$avgsim^a_p = \underset{r}{avg}\,(sim^a_{p,r})$

STEP4 : Max Pooling (Augment)
$avgsim_p = \max_a (avgsim^a_p)$

STEP5 : Min Pooling (Path)
$avgsim = \min_p (avgsim_p)$

$\{RT_0, \dots, RT_m\}$

RT_0 : #1(X, Y) :- #2(X, Z), #3(Z, Y).

RT_0 : #1(X, Y) :- #2(X, Z), #3(Z, Y).

RT_0 : #1(X, Y) :- #2(X, Z), #3(Z, Y).

RT_0 : #1(X, Y) :- #2(X, Z), #3(Z, Y).

$\{RT_0, \dots, RT_m\}$

$\min(RT_0, \dots, RT_m)$

Proof Score

Relation Similarity Learning STEP1 : Relation Embedding

- **Augmented Rule Template과 KG Relation에 대한 Embedding Matrix생성**

Rule Template

RT_0 : #1(X, Y) :- #2(X, Z), #3(Z, Y). 2

...

RT_m: #1(X, Y) :- #2(X, Y) 2

augmentation

Augmented Rule Template

#1_0_0(X, Y) :- #2_0_0(X, Z), #3_0_0(Y, Z).
#1_0_1(X, Y) :- #2_0_1(X, Z), #3_0_1(Y, Z).

...

#1_m_0(X, Y) :- #2_m_0(X, Y).
#1_m_1(X, Y) :- #2_m_1(X, Y).

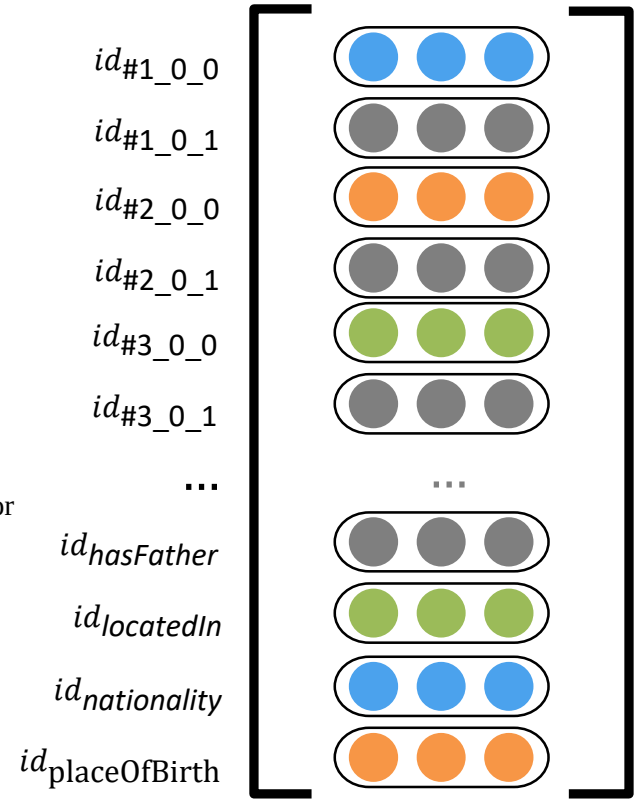
predicate num _ rule num _ augment num

Knowledge Graph(DataFrame)		
pred	**subj**	**obj**
nationality	BART	USA
placeOfBirth	BART	NEWYORK
hasFather	BART	HOMMER
placeOfBirth	HOMMER	NEWYORK
nationality	HOMMER	USA
locatedIn	NEWYORK	USA

Sym2id (Dict)	
Key(Relation)	Value(id)
#1_0_0	0
#1_0_1	1
#2_0_0	2
#2_0_1	3
#3_0_0	4
#3_0_1	5
...	...
hasFather	16
locatedIn	17
nationality	18
placeOfBirth	19

Each Relation Gets a
(1 × Embedding Size)
Random Initialized vector

Relation Embedding Matrix
(Vocab Size × Embedding Size)



Relation Similarity Learning STEP1 : Relation Embedding

- 주어진 Query와 Rule Template에 대한 Encoded Path를 Embedding Vector로 변환

Query Q_0
nationality(BART, USA)

Rule Template
RT_0 : #1(X, Y) :- #2(X, Z), #3(Z, Y). 2

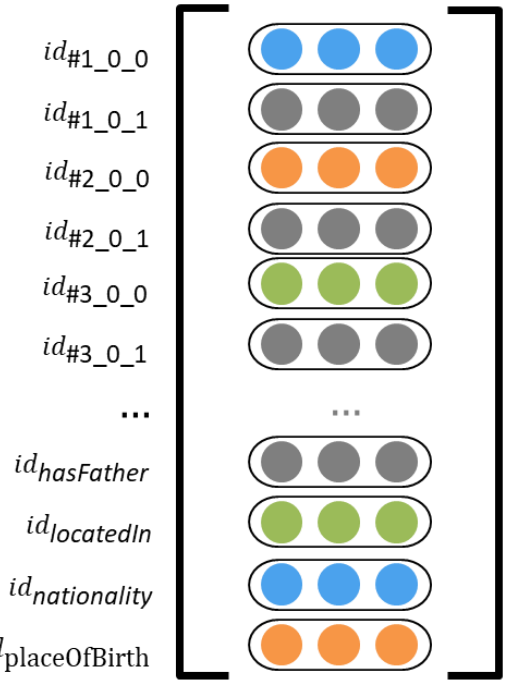
Relation ID : $rid^{\#augment}_{\#path,\ \#relation}$

Encoded Augmented Unified Relation Path		
	$Augment_0$	$Augment_1$
$Path_1$	$(rid^0_{1,1}, rid^0_{1,2}, rid^0_{1,3})$	$(rid^1_{1,1}, rid^1_{1,2}, rid^1_{1,3})$
$Path_2$	$(rid^0_{2,1}, rid^0_{2,2}, rid^0_{2,3})$	$(rid^1_{2,1}, rid^1_{2,2}, rid^1_{2,3})$
$Path_1$	$(rid^0_{1,1}, rid^0_{1,2}, rid^0_{1,3})$	$(rid^1_{1,1}, rid^1_{1,2}, rid^1_{1,3})$
$Path_2$	$(rid^0_{2,1}, rid^0_{2,2}, rid^0_{2,3})$	$(rid^1_{2,1}, rid^1_{2,2}, rid^1_{2,3})$

Rule Template ID : $rtid^{\#augment}_{\#path,\ \#relation}$

Encoded Augmented Unified Rule Template Path		
	$Augment_0$	$Augment_1$
$Path_1$	$(rtid^0_{1,1}, rtid^0_{1,2}, rtid^0_{1,3})$	$(rtid^1_{1,1}, rtid^1_{1,2}, rtid^1_{1,3})$
$Path_2$	$(rtid^0_{2,1}, rtid^0_{2,2}, rtid^0_{2,3})$	$(rtid^1_{2,1}, rtid^1_{2,2}, rtid^1_{2,3})$
$Path_1$	$(rtid^0_{1,1}, rtid^0_{1,2}, rtid^0_{1,3})$	$(rtid^1_{1,1}, rtid^1_{1,2}, rtid^1_{1,3})$
$Path_2$	$(rtid^0_{2,1}, rtid^0_{2,2}, rtid^0_{2,3})$	$(rtid^1_{2,1}, rtid^1_{2,2}, rtid^1_{2,3})$

Relation Embedding Matrix
(Vocab Size × Embedding Size)



Relation Embedding : $re^{\#augment}_{\#path,\ \#relation}$

Embedded Augmented Unified Relation Path		
	$Augment_0$	$Augment_1$
$Path_1$	$(re^0_{1,1}, re^0_{1,2}, re^0_{1,3})$	$(re^1_{1,1}, re^1_{1,2}, re^1_{1,3})$
$Path_2$	$(re^0_{2,1}, re^0_{2,2}, re^0_{2,3})$	$(re^1_{2,1}, re^1_{2,2}, re^1_{2,3})$
$Path_1$	$(re^0_{1,1}, re^0_{1,2}, re^0_{1,3})$	$(re^1_{1,1}, re^1_{1,2}, re^1_{1,3})$
$Path_2$	$(re^0_{2,1}, re^0_{2,2}, re^0_{2,3})$	$(re^1_{2,1}, re^1_{2,2}, re^1_{2,3})$

Rule Template Embedding : $rte^{\#augment}_{\#path,\ \#relation}$

Embedded Augmented Unified Rule Template Path		
	$Augment_0$	$Augment_1$
$Path_1$	$(rte^0_{1,1}, rte^0_{1,2}, rte^0_{1,3})$	$(rte^1_{1,1}, rte^1_{1,2}, rte^1_{1,3})$
$Path_2$	$(rte^0_{2,1}, rte^0_{2,2}, rte^0_{2,3})$	$(rte^1_{2,1}, rte^1_{2,2}, rte^1_{2,3})$
$Path_1$	$(rte^0_{1,1}, rte^0_{1,2}, rte^0_{1,3})$	$(rte^1_{1,1}, rte^1_{1,2}, rte^1_{1,3})$
$Path_2$	$(rte^0_{2,1}, rte^0_{2,2}, rte^0_{2,3})$	$(rte^1_{2,1}, rte^1_{2,2}, rte^1_{2,3})$

Example: $(rid^0_{1,1}, rid^0_{1,2}, rid^0_{1,3}) = (id_{nationality}, id_{placeOfBirth}, id_{locatedIn})$
$(rtid^0_{1,1}, rtid^0_{1,2}, rtid^0_{1,3}) = (id_{\#1\_0\_0}, id_{\#2\_0\_0}, id_{\#3\_0\_0})$

Example: $(re^0_{1,1}, re^0_{1,2}, re^0_{1,3}) = (E_{nationality}, E_{placeOfBirth}, E_{locatedIn})$
$(rte^0_{1,1}, rte^0_{1,2}, rte^0_{1,3}) = (E_{\#1\_0\_0}, E_{\#2\_0\_0}, E_{\#3\_0\_0})$

Relation Similarity Learning STEP2 : Relation Similarity

- **Relation Path와 Rule Template path의 각 Symbol이 갖는 Embedding Vector간 유사도 계산**

Relation Embedding : $re^{\#augment}_{\#path,\ \#relation}$

Embedded Augmented Unified Relation Path		
	$Augment_0$	$Augment_1$
$Path_1$	$(re^0_{1,1}, re^0_{1,2}, re^0_{1,3})$	$(re^1_{1,1}, re^1_{1,2}, re^1_{1,3})$
$Path_2$	$(re^0_{2,1}, re^0_{2,2}, re^0_{2,3})$	$(re^1_{2,1}, re^1_{2,2}, re^1_{2,3})$
$Path_1$	$(re^0_{1,1}, re^0_{1,2}, re^0_{1,3})$	$(re^1_{1,1}, re^1_{1,2}, re^1_{1,3})$
$Path_2$	$(re^0_{2,1}, re^0_{2,2}, re^0_{2,3})$	$(re^1_{2,1}, re^1_{2,2}, re^1_{2,3})$

Rule Template Embedding : $rte^{\#augment}_{\#path,\ \#relation}$

Embedded Augmented Unified Rule Template Path		
	$Augment_0$	$Augment_1$
$Path_1$	$(rte^0_{1,1}, rte^0_{1,2}, rte^0_{1,3})$	$(rte^1_{1,1}, rte^1_{1,2}, rte^1_{1,3})$
$Path_2$	$(rte^0_{2,1}, rte^0_{2,2}, rte^0_{2,3})$	$(rte^1_{2,1}, rte^1_{2,2}, rte^1_{2,3})$
$Path_1$	$(rte^0_{1,1}, rte^0_{1,2}, rte^0_{1,3})$	$(rte^1_{1,1}, rte^1_{1,2}, rte^1_{1,3})$
$Path_2$	$(rte^0_{2,1}, rte^0_{2,2}, rte^0_{2,3})$	$(rte^1_{2,1}, rte^1_{2,2}, rte^1_{2,3})$

Example: $(re^0_{1,1}, re^0_{1,2}, re^0_{1,3}) = (E_{nationality}, E_{placeOfBirth}, E_{locatedIn})$
$(rte^0_{1,1}, rte^0_{1,2}, rte^0_{1,3}) = (E_{\#1\_0\_0}, E_{\#2\_0\_0}, E_{\#3\_0\_0})$

Relation Similarity
$sim^a_{p,r} = \mathrm{sim}(re^a_{p,r}, rte^a_{p,r})$

Relation Similarity : $sim^{\#augment}_{\#path,\ \#relation}$

Relation Similarity		
	$Augment_0$	$Augment_1$
$Path_1$	$\{sim^0_{1,1}, sim^0_{1,2}, sim^0_{1,3}\}$	$\{sim^1_{1,1}, sim^1_{1,2}, sim^1_{1,3}\}$
$Path_2$	$\{sim^0_{2,1}, sim^0_{2,2}, sim^0_{2,3}\}$	$\{sim^1_{2,1}, sim^1_{2,2}, sim^1_{2,3}\}$
$Path_1$	$\{sim^0_{1,1}, sim^0_{1,2}, sim^0_{1,3}\}$	$\{sim^1_{1,1}, sim^1_{1,2}, sim^1_{1,3}\}$
$Path_2$	$\{sim^0_{2,1}, sim^0_{2,2}, sim^0_{2,3}\}$	$\{sim^1_{2,1}, sim^1_{2,2}, sim^1_{2,3}\}$

Example: $\{sim^0_{1,1}, sim^0_{1,2}, sim^0_{1,3}\} = \{sim(E_{nationality}, E_{\#1\_0\_0}),$
$sim(E_{placeOfBirth}, E_{\#2\_0\_0}),$
$sim(E_{locatedIn}, E_{\#3\_0\_0})\}$

Relation Similarity Learning STEP3 : Aggregation

- 각 Path의 Augment마다 구해진 Similarity의 평균 계산

Relation Similarity : $sim^{\#augment}_{\#path,\ \#relation}$

Relation Similarity		
	$Augment_0$	$Augment_1$
$Path_1$	$\{sim^0_{1,1}, sim^0_{1,2}, sim^0_{1,3}\}$	$\{sim^1_{1,1}, sim^1_{1,2}, sim^1_{1,3}\}$
$Path_2$	$\{sim^0_{2,1}, sim^0_{2,2}, sim^0_{2,3}\}$	$\{sim^1_{2,1}, sim^1_{2,2}, sim^1_{2,3}\}$
$Path_1$	$\{sim^0_{1,1}, sim^0_{1,2}, sim^0_{1,3}\}$	$\{sim^1_{1,1}, sim^1_{1,2}, sim^1_{1,3}\}$
$Path_2$	$\{sim^0_{2,1}, sim^0_{2,2}, sim^0_{2,3}\}$	$\{sim^1_{2,1}, sim^1_{2,2}, sim^1_{2,3}\}$

Aggregation (Average)

$$avgsim^a_p = \underset{r}{avg}\ (sim^a_{p,r})$$

Average Relation Similarity : $avgsim^{\#augment}_{\#path}$

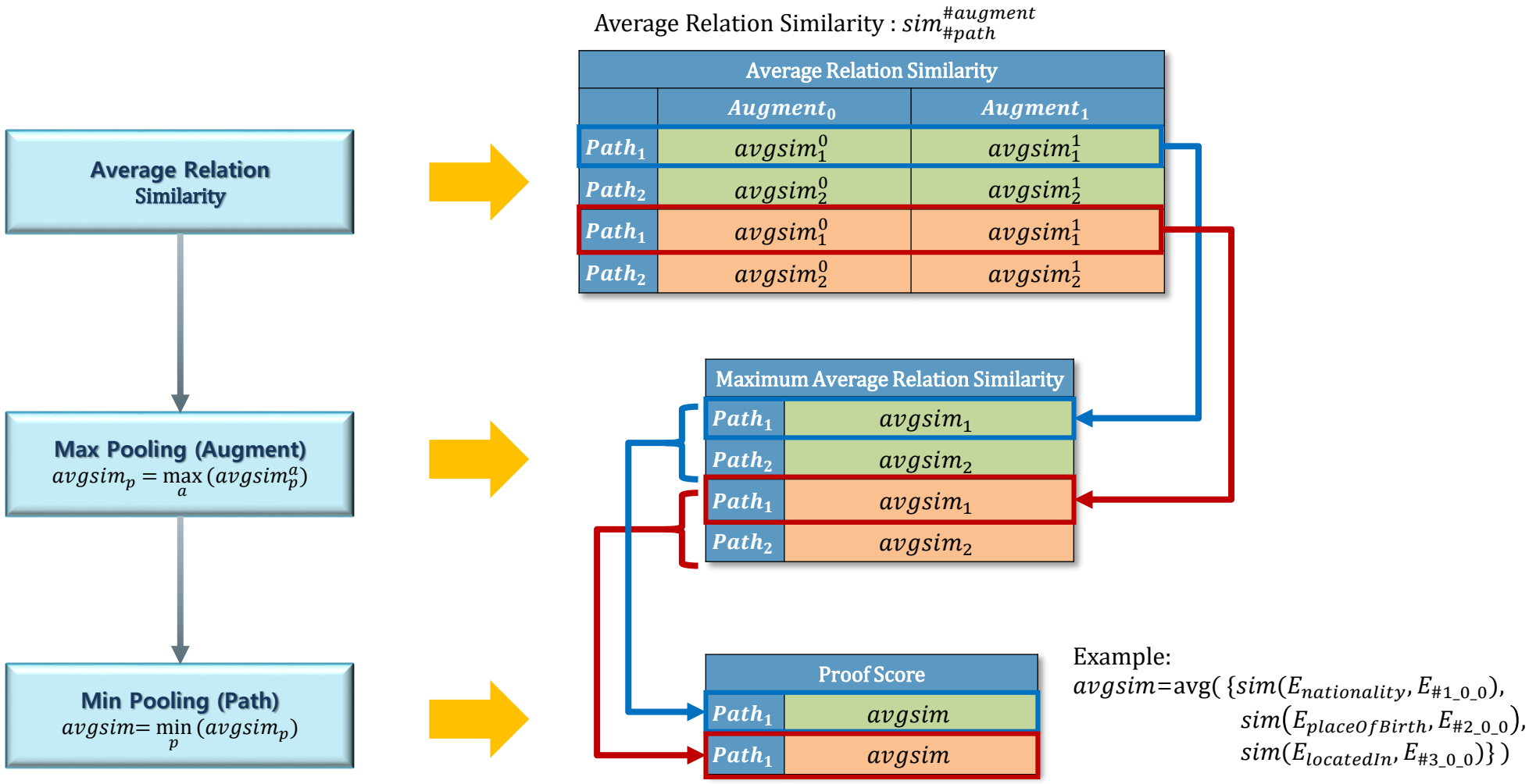
Average Relation Similarity		
	$Augment_0$	$Augment_1$
$Path_1$	$avgsim^0_1$	$avgsim^1_1$
$Path_2$	$avgsim^0_2$	$avgsim^1_2$
$Path_1$	$avgsim^0_1$	$avgsim^1_1$
$Path_2$	$avgsim^0_2$	$avgsim^1_2$

Example: $\{sim^0_{1,1}, sim^0_{1,2}, sim^0_{1,3}\}=\{sim(E_{nationality}, E_{\#1\_0\_0}),$
$sim(E_{placeOfBirth}, E_{\#2\_0\_0}),$
$sim(E_{locatedIn}, E_{\#3\_0\_0})\}$

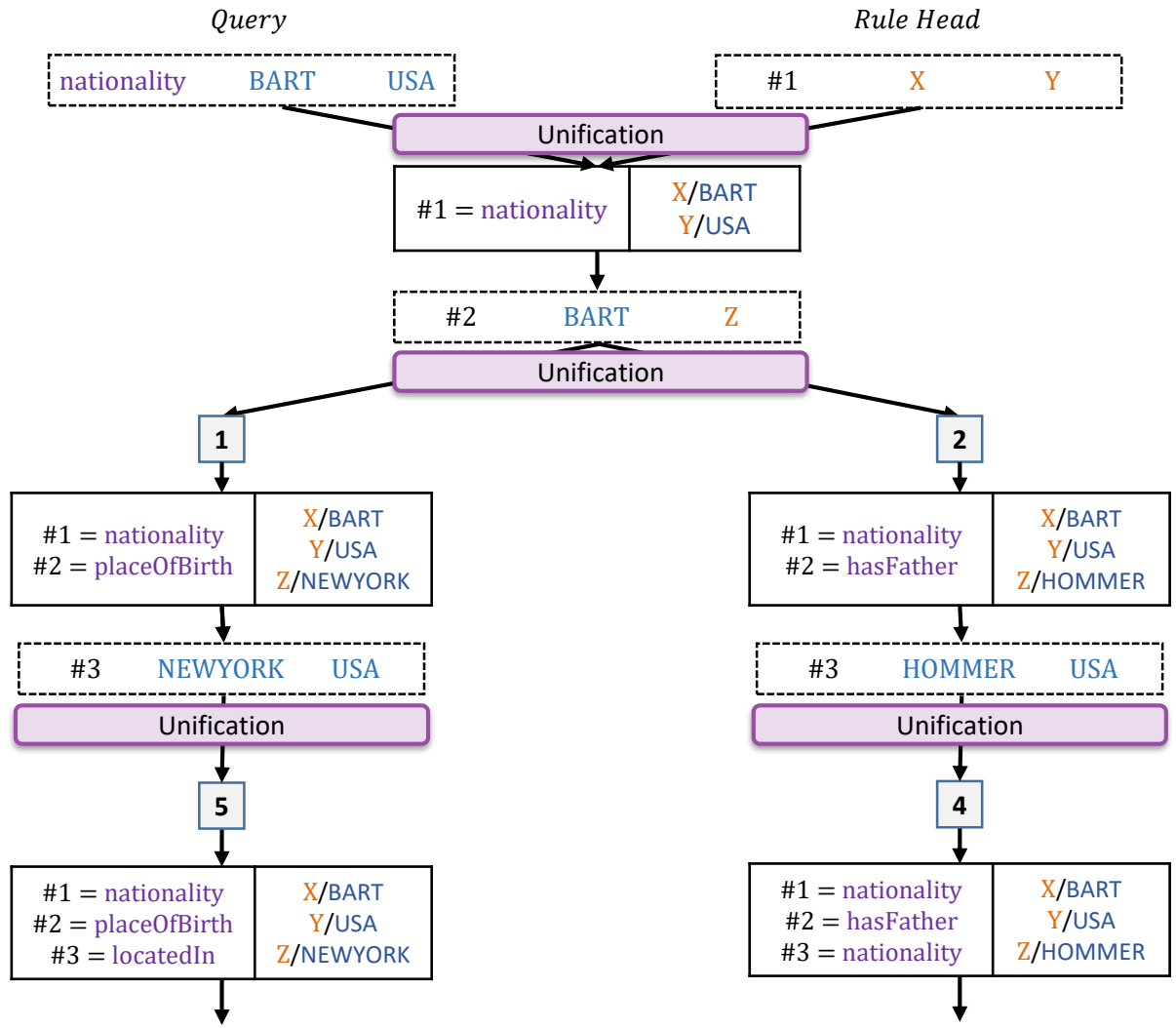
Example: $avgsim^0_1$=avg($\{sim(E_{nationality}, E_{\#1\_0\_0}),$
$sim(E_{placeOfBirth}, E_{\#2\_0\_0}),$
$sim(E_{locatedIn}, E_{\#3\_0\_0})\}$)

Relation Similarity Learning STEP4&5 : Pooling

- Max, Min Pooling을 통한 Proof Score 산출 과정



Cost Optimization



Unified Relation Path 1

[(#1, nationality), (#2, placeOfBirth), (#3, locatedIn)]

Unified Relation Path 2

[(#1, nationality), (#2, hasFather), (#3, nationality)]

Knowledge Graph(DataFrame)			
Index	**pred**	**subj**	**obj**
0	nationality	BART	USA
1	placeOfBirth	BART	NEWYORK
2	hasFather	BART	HOMMER
3	placeOfBirth	HOMMER	NEWYORK
4	nationality	HOMMER	USA
5	locatedIn	NEWYORK	USA

Query $\boldsymbol{Q_0}$

nationality(BART, USA)

Rule Template

$\boldsymbol{RT_0}$: #1(X, Y) :- #2(X, Z), #3(Z, Y).

- **Unified Relation Path Given Query Q & Rule Template RT**
  - $\Pi^{Q,RT} = \cup_{p=1}^{n} \pi_p$
    - $\pi_1 = \{(\#1, nationality), (\#2, placeOfBirth), (\#3, locatedIn)\}$
    - $\pi_2 = \{(\#1, nationality), (\#2, hasFather), (\#3, nationality)\}$
- **Cost Function of Π**
  - $Loss = -y\, log(f(\Pi^{Q,RT})) - (1-y)\, log(1 - f(\tilde{\Pi}^{Q,RT}))$
  - $f(\Pi^{Q,RT}) = \min_{p}(\max_{a}(\cup_{a,p}\, avgsim_p^a))$ $\quad a$: #augment , p : #path
  - $avgsim_1^0 = avg(\pi_1^0) = avg\begin{pmatrix} sim(E_{\#1\_0\_0}, E_{nationality}), \\ sim(E_{\#2\_0\_0}, E_{placeOfBirth}), \\ sim(E_{\#3\_0\_0}, E_{locatedIn}) \end{pmatrix}$

Rule Induction

Rule Template

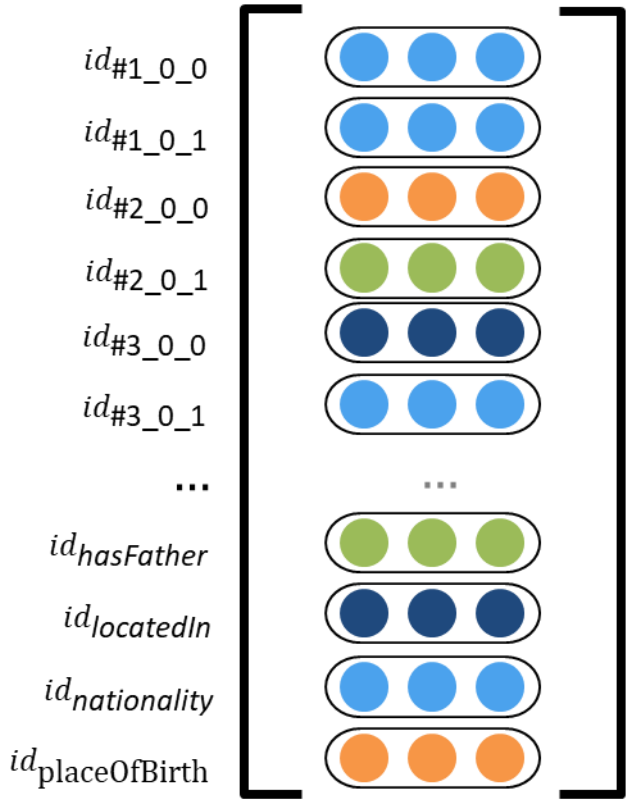
RT_0 : #1(X, Y) :- #2(X, Z), #3(Z, Y).

augmentation

Augmented Rule Template

#1_0_0(X, Y) :- #2_0_0(X, Z), #3_0_0(Z, Y).
#1_0_1(X, Y) :- #2_0_1(X, Z), #3_0_1(Z, Y).

Relation Embedding Matrix
(Vocab Size × Embedding Size)

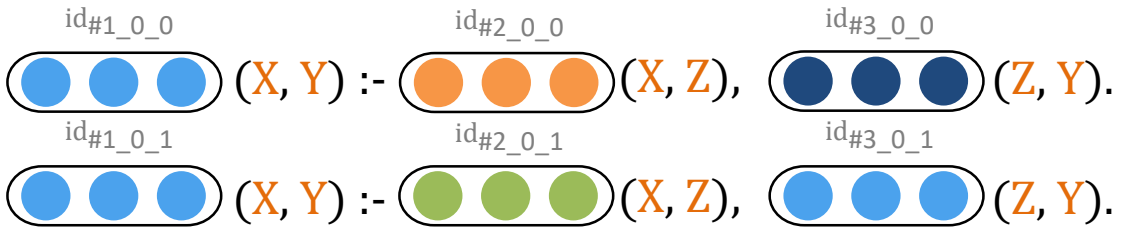


Embedding Look Up

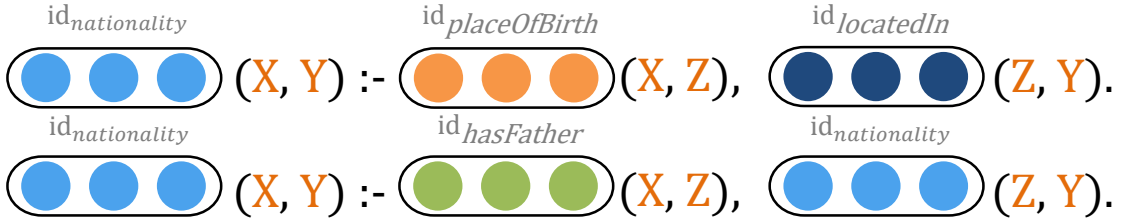
Select Most Similar Relation Vector

Decoding

Parameterized Augmented Rule Template



Parameterized Rule Instance



Rule Instance

nationality(X, Y) :- placeOfBirth(X, Z), locatedIn(Z, Y).
nationality(X, Y) :- hasFather (X, Z), nationality(Z, Y).

END