Appendix A Algorithm Definition

Function `ProcessConflict`(*Rule* r_1, *Rule* r_2, *Meta-policy* MP)**:**
- $boolean_1$=$MP.strongerRule$.`matchRule`(r_1)
- $boolean_2$=$MP.weakerRule$.`matchRule`(r_2)
- **return** $boolean_1$&$boolean_2$

Algorithm 1: Function Process Conflict

Function `Compare`(*Rule* r_1, *Rule* r_2, *Meta-policy* MP)**:**
- **if** `ProcessConflict`(r_1,r_2,MP) **then**
  - **return** 1
- **else if** `ProcessConflict`(r_2,r_1,MP) **then**
  - **return** 2
- **return** 0

Algorithm 2: Function Compare

Function `IsOpposite`(*Deontic* d_1, *Deontic* d_2)**:**
- $boolean = False$
- **if** *(*(d_1=**O**& d_2=**P**) ***Or*** (d_2=**O**& d_1=**P**)
- ***Or*** (d_1=**A**& d_2=**P**) ***Or*** (d_2=**A**& d_1=**P**)
- ***Or*** (d_1=**O**& d_2=**D**) ***Or*** (d_2=**O**& d_1=**D**)*)* **then**
  - $boolean = True$
- **return** *boolean*

Algorithm 3: Function IsOpposite

```
Function Minus(Condition cond_1, Condition cond_2):
    for triple in cond_1 do
        if triple not in cond_2 then
            cond_diff.add(triple)
    return cond_diff
```

Algorithm 4: Function Minus

```
Function GetMappings(Condition cond, Knowledge Base K):
    query = "SELECT * WHERE { " +cond+ "}"
    result = ExecuteQuery(query, K)
    Ω= []
    for row in result do
        μ = {}
        for var in row.keys() do
            key= var
            value= row.get(key)
            μ.add(key,var)
    Ω.insert(μ)
return Ω
```

Algorithm 5: Function GetMappings

```
Function GetMapping(Condition cond, Knowledge Base K, IRI iri):
    query = "SELECT * WHERE {" + cond + "FILTER" +(?x = iri )+ "}"
    result = ExecuteQuery(query, K)
    μ = {}
    for var in result.keys() do
        key= var
        value= result.get(key)
        μ.add(key,var)
    return μ
```

Algorithm 6: Function GetMapping

Function `Exists(`*Action a, Knowledge Base K*`):`
boolean = ExecuteQuery("ASK WHERE {" +a+ "}", K)
return boolean

Algorithm 7: Function Exists