Thesis Algorithms

Shyama Wilson

September 2022

1 Modularity

2 Completeness

Algorithm 1 Maximum depth of an ontology

```
 1: Data: Root Node: OWLThing in OWL ontology
 2: Result: Number of classes in the longest path/branch
 3: visited ←NULL
 4: depth ←0
 5: maxDepth ←0
 6: circularityClass ←NULL
 7: MAXDEPTH(root);
 8:
 9: function MAXDEPTH(root)
10:     classList ←get adjacent connected sub classes of the root
    node:OWLThing
11:     if classList is EMPTY then
12:         Message: Empty Class List
13:     else
14:         for each class of classList do
15:             DFS(class);
16:             if $maxDepth \leq depth$ then
17:                 $maxDepth \leftarrow depth$
18:             end if
19:         end for
20:     end if
21:     return maxDepth
22: end function
23:
24: function DFS(class)
25:     if visited NOT contain the class then
26:         add class to the visited list
27:         depth++
28:         classList ←get adjacent connected sub classes of the current class
29:         for each class of classList do
30:             DFS(class);
31:         end for
32:     else
33:         Message: No new subclasses in the List and CIRCULARITY is
    delected
34:         circularityClass.add (class);
35:     end if
36: end function
```

Algorithm 2 Maximum breadth of an ontology

```
Data: Root Node: OWLThing in OWL ontology
Result: Number of classes in the widest path/branch
$visited$ ←NULL
$breadth$ ←0
$maxBreadth$ ←0
$subBrachClasses$ ←NULL
$circularityClass$ ←NULL
MAXBREADTH(root);

function MAXBREADTH(current class)
    if $visited$ NOT contain the current class then
        $classList$ ←get adjacent connected sub classes of the current class
        if $classList$ is EMPTY then
            Message: Empty Class List
        else
            $breadth$ ←the number of classes in the classList
            if $maxBreadth \leq breadth$ then
                $maxBreadth \leftarrow breadth$
            end if
            $subBrachClasses.add(classList)$
            for each class of $subBrachClasses$ do
                MAXBREADTH(class);
            end for
        end if
    else
        No new subclasses in the List and CIRCULARITY is delected
        circularityClass.add (class);
    end if
    return $maxBreadth$
end function
```

Algorithm 3 Relationship richness

```
Data: O: OWL ontology, r: reasoner
Result: Relationship Richness as a percentage
$P$ ←0
$H$ ←0
$relationshipRichness$ ←0
RR(ontology);

function RR(O)
    $P$ ←r.getNonTaxonomic relations(O).size();
    $H$ ←r.getTaxonomic relations(O).size();
    if P + H is NOT 0 then
        $relationshipRichness \leftarrow \frac{P}{(P+H)} \times 100\%$
    end if
    return $relationshipRichness$
end function
```

Algorithm 4 Concept richness

```
Data: O: OWL ontology, r: reasoner
Result: Concept richness as a percentage
CR(ontology);
$C$ ←0
$C^{‘}$ ←0
$conceptRichness$ ←0
$classList$ ←NULL
$usedClassList$ ←NULL

function CR(O)
    $classList$ ←getDeclaredClasses(O);
    $C$ ←classList.size();
    for each class in $classList$ do
        if $r.getIndividuals(class)$ is NOT EMPTY then
            $usedClassList.add(class)$
        end if
    end for
    $C^{‘}$ ←usedClassList.size();
    if C is NOT 0 then
        conceptRichness $\leftarrow \frac{C^{‘}}{C} \times 100\%$
    end if
    return $conceptRichness$
end function
```

Algorithm 5 Object property usage

```
Data: O: OWL ontology, r: reasoner
Result: Object property usage as a percentage
OPU(ontology);
$declaredRelations$ ←0
$relationshipUsage$ ←0
$relationshipList$ ←NULL
$linkedRelationList$ ←NULL
$isolatedRelationshipList$ ←NULL

function OPU(O)
    $relationshipList$ ←o.getDeclaredObjectProperties();
    $declaredRelations$ ←relationshipList.size();
    for each relation in $relationshipList$ do
        $relationDomain$ ←o.getDomain(objectProperty);
        $relationRange$ ←o.getRange(objectProperty);
        if $relationDomainandrelationRange$ is NOT EMPTY then
            linkedRelationList.add(objectProperty);
        else if inferred domains and ranges are exists for relation then
            linkedRelationList.add(objectProperty);
        else
            isolatedRelationshipList.add(objectProperty)
        end if
    end for
    if declaredRelations is NOT 0 then
        relationshipUsage ← $\frac{linkedRelationList.size()}{declaredRelations} \times 100\%$
    end if
    return $relationshipUsage$
end function
```

Algorithm 6 Data property usage

```
Data: O: OWL ontology, r: reasoner
Result: Data property usage as a percentage
DPU(ontology);
$declaredAttributes$ ←0
$attributeUsage$ ←0
$attributeList$ ←NULL
$linkedAttributesList$ ←NULL
$isolatedAttributeList$ ←NULL

function DPU(O)
    $attributeList$ ←o.getDeclaredDataProperties();
    $declaredAttributes$ ←attributeList.size();
    for each attributes in $attributeList$ do
        $attributeDomain$ ←r.getDomain(dataProperty);
        $attributeRange$ ←r.getRange(dataProperty);
        if $attributeDomainandattributeRange$ is NOT EMPTY then
            linkedAttributesList.add(dataProperty);
        else
            isolatedAttributeList.add(dataProperty)
        end if
    end for
    if declaredAttributes is NOT 0 then
        attributeUsage ← $\frac{linkedAttributesList.size()}{declaredAttributes} \times 100\%$
    end if
    return $attributeUsage$
end function
```

Algorithm 7 Instance usage

```
Data: O: OWL ontology, r: reasoner
Result: Instance usage as a percentage
IU(ontology);
$declaredInstance$ ←0
$instanceUsage$ ←0
$instanceList$ ←NULL
$instanceHasTypeList$ ←NULL
$isolatedInstanceList$ ←NULL

function IU(O)
    $instanceList$ ←o.getDeclaredIndividuals();
    $declaredInstance$ ←instanceList.size();
    for each instance in $instanceList$ do
        $instanceHasType$ ←r.getType(instance);
        if $instanceHasType$ is NOT NULL then
            instanceHasTypeList.add(instance);
        else
            isolatedInstanceList.add(instance)
        end if
    end for
    if declaredInstance is NOT 0 then
        instanceUsage ← $\frac{instanceHasTypeList.size()}{declaredInstance} \times 100\%$
    end if
    return $instanceUsage$
end function
```