Appendix 1 : the JDR algorithm

Hanyang Cao[1], Jean-Rémy Falleri[1], Xavier Blanc[1], Li Zhang[2]

[1] University of Bordeaux, LaBRI, UMR 5800
F-33400, Talence, France
[2] Beihang University

1 Algorithms

We assume that nodes have a `kind` attribute which has a value in the {literal, object, array} set. We also assume that nodes have an `hash` attribute that corresponds to the hash of the subtree rooted at this object for object nodes, the hash of the value for literal nodes, and the hash of the sequence for array nodes. They also have a `props` attribute containing the set of properties owned by the object. Object nodes and array nodes have a `path` attribute that contains their JSON path in the JSON Literal nodes have `type` attribute which has a value in the {boolean, number, string} set, and a `value` attribute that contains the literal's value.

Algorithm 1 The main algorithm

```
commons = hashmap()
additions = hashmap()
deletions = hashmap()
actions = list()
for all key ∈ additions.keys do
    delNodes = deletions[key]
    for all addNode ∈ additions[key] do
        if delNodes.size() > 0 then
            delNode = delNodes.pop()
            actions.add(move(delNode.path, addNode.path))
        else
            node = findUnchangedNodeByHash(old, new, addNode)
            if node ≠ NULL then
                actions.add(copy(node.path, addNode.path))
            else
                actions.add(addition(addNode.path, addNode.json))
            end if
        end if
    end for
end for
```

Algorithm 2 The compare algorithm

```
function COMPARE(old, new)
    if old.type ≠ new.type then
        deletions[old.hash].add(old)
        additions[new.hash].add(new)
    else if old.type == literal then
        if old.value ≠ new.value then
            actions.add(replace(old.path, new.value))
        end if
    else if old.type == object then
        for all prop ∈ old.props ∩ new.props do
            compare(old[prop], new[prop])
        end for
        for all prop ∈ old.props \ new.props do
            deletions[old[prop].hash].add(old[prop])
        end for
        for all prop ∈ new.props \ old.props do
            additions[new[prop].hash].add(new[prop])
        end for
    else if old.type == array then
        compareArray(old, new)
    end if
end function
```

Algorithm 3 The compareArray algorithm

```
function compareArray(old, new)
    oldPos = hashmap()
    newPos = hashmap()
    tmpPatch = list()
    for i = 0 → i < old.size do
        oldPos[old[i].hash].add(i)
    end for
    for i = 0 → i < new.size do
        newPos[new[i].hash].add(i)
    end for
    for all key ∈ oldPos.keys ∩ newPos.keys do
        for i = 0 → i < max(oldPos[key], newPos[key]) do
            if i ≥ oldPos[key].size then
                tmpPatch.add(arrayAddition(newPos[key][i], new[newPos[key][i]]))
            else if i ≥ newPos[key].size then
                tmpPatch.add(arrayDeletion(oldPos[key][i]))
            else
                tmpPatch.add(arrayMove(oldPos[key][i], newPos[key][i]))
            end if
        end for
    end for
    for all key ∈ oldPos.keys \ newPos.keys do
        for all i ∈ oldPos[key] do
            tmpPatch.add(arrayDeletion(i))
        end for
    end for
    for all key ∈ newPos.keys \ oldPos.keys do
        for all i ∈ newPos[key] do
            tmpPatch.add(arrayAddition(i, new[i])
        end for
    end for
    sort(tmpPatch)
    for i = 0 → i ≥ tmpPatch.size do
        action = tmpPath[i]
        if action.type == move then
            if action.from == action.to then
                break
            end if
            isForward = action.from < action.to
            for j = i + 1 → i ≥ tmpPatch.size do
                if isForward == true then
                    bound = action.to
                    impact = −1
                else
                    bound = action.from
                    impact = 1
                end if
                changeIndex(tmpPatch[j], impact, bound)
            end for
        else if action.type == deletion then
            for j = i + 1 → i ≥ tmpPatch.size do
                changeIndex(tmpPatch[j], −1, tmpPatch.size)
            end for
        else
            for j = i + 1 → i ≥ tmpPatch.size do
                changeIndex(tmpPatch[j], +1, tmpPatch.size)
            end for
        end if
    end for
end function
```

Algorithm 4 The changeIndex algorithm

```
function CHANGEINDEX(action, impact, bound)
    if action.type == arrayDeletion then
        if action.at < bound then // FIXME
            action.at = action.at + impact
        end if
    else if action.type == arrayMove then
        if action.from < bound then
            action.from = action.from + impact
        end if
    end if
end function
```