**COMP 3002 Winter 2021 Assignment #10**

**(Building Tables – Last Assignment)**

**Due: Tuesday Midnight March 29**

From the previous assignment

Implement a table builder using instance variables such as

readaheadStates

readbackStates

reduceStates

semanticStates

acceptState

grammar

left right down up

1. For now, make the table builder invoke the grammar builder from the previous assignment in order to create a grammar. Use one of the toy grammars.

2. Extend the grammar by hand-building the augmented production for that grammar.

3. Create two relations for the entire augmented grammar:

a) right (triple has right part state, transition name, right part state)

b) down (triple has right part state, nonterminal, right part state)

**For this assignment, build readahead, readback, reduce states and semantic actions states using the building FSM notes, building left from right, and building left from right notes in notes #12.5 along with the process for processing semantic action.**

**Note: the slides are more detailed but a repeat of the process is given below.**

**BUILD READAHEAD STATES**

While building readahead states, we will be building left from right and building up from down.

1. Start with an initial readahead state built from the initial item of the augmented production and add it to a working collection.

2. Use a loop such as

workingCollection do: [:currentState:

Process the currentState.

3. From the currentState items, compute “items down\*” and while doing that, compute corresponding

ups (the reverse of the downs with pairs for everything, the second part of the pair is a readahead state)

. Then use the relation operation

from: “items down\*” do: [:transitionName :successorRelation | …]

to compute successor items.

Let items = successorRelation allTos.

Let goto = “if items match existing readheadState”  
 then existingState   
 else new readahead state with items and add the state to states

Add transistion from currentState under transitionName to goto.

Build left from the information in successorRelation.

**BUILD SEMANTIC STATES**

See notes since it’s fairly complex.

**BUILD LOOKAHEADS TO READBACK STATES**

4. Once the readhead states are done, loop through readahead states and partition final items by

their NONTERMINAL left parts.

Given final items F associated with nonterminal A, create a look transition from the readahead state

under look transitions Follow (A) to an initial readback state with items consisting of F

Compute successor states using the same algorithm as above; i.e.

5. Start with an initial readback state from above

6. Use a loop such as

workingCollection do: [:currentState:

Process the currentState.

7. From the currentState items, compute “items invisibleLeft\*” of the existing items.

Then use the relation operation

from: “items invisibleLeft\*” do: [:transitionPair :successorRelation | …]

to compute successor items.

Let items = successorRelation allTos.

Let goto = “if items match existing readbackState”  
 then existingState   
 else new readback state with items and add the state to states

Add transistion from currentState under transitionPair to goto.

BUILD LOOKBACK TO REDUCE STATES

8. See notes for the lookbacks since it’s fairly complex.