ACT-R Dealing with multiple potential consequences

Friday, October 30, 2020 10:07 AM

Conditionals are encoded as productions (duh)
Encoding multiple consequences to a condition involves multiple productions, each with the same LHS
Stochasticity of production selection can happen through the noise associated with the utility/recall function for the production

The big question then is how to encode these multiple productions into chunks?
What if we added a slot to chunks (specifically consequence chunks) that had a "competitive" value of True or False to identify chunks that could be consequences at the cost of other consequences?
i.e.
A
…

consOf	Cond1
Competitive	True

B
…

consOf	Cond1
Competitive	True

C
…

consOf	Cond1
Competitive	False

Would indicate that A _OR_ B can happen, but C always happens?

Weakness: I'm not sure this would work with multiple layers of things (for example (A OR B) AND (C OR D))
Potentially change to show which node it competes with? i.e.
A
…

consOf	Cond1
Competitive	B

B
…

consOf	Cond1
Competitive	A

C
…

consOf	Cond1
Competitive	D

D
…

consOf	Cond1
Competitive	C

Weakness: This might not work if more than two nodes are competing with each other (i.e. A OR B OR C AND D)
Lists aren't an option in ACT-R… would it be reasonable to have a set number of competitive slots or is that not generalizable enough?
Also, how to encode NextCons if you have a branching list of potential next states
Maybe something like (this idea is based off what Mary sent, need clarification on that)

Competitive	True :arg0 A :arg 1 B …

Can args be dynamically sized?

Should L be a Consequence Even? Or should Only edges Be able to be Consequences?

```
K
isa        condition
subj       G
pred       K1
desiredStatePred True
stateOfCondition [True|False]
target     E
parent     A

L
isa        node
role       subject
name       nil
consOf     K
prevCons   nil
nextCons   M
parent     A

M
isa        edge
role       click
source     L
dest       C
state      [True|False]
consOf     K
prevCons   L
nextCons   N
parent     A

N
isa        edge
role       remember
source     L
dest       I
state      [True|False]
consOf     K
prevCons   M
nextCons   nil
parent     A
```

Production pseudocode:

```
p(
    Target: K1
    State: True
=>
    =goal>
        consToDo: L
)

p(
    Goal>
        consToDo: L
=>
    createNodeL (don't like this)
    Goal>
        consToDo:M
)

p(
    Goal>
        consToDo: M
=>
    stateOfM = true
    performClickAction

    Goal> consToDo: N
)

p(
    Goal>
        consToDo: N
=>
    stateOfN = true
    performRememberAction
    Goal> consToDO: nil
)
```

IMPLEMENTING COMPETITIVENESS: Example either click OR remember

```
K
isa        condition
subj       G
pred       K1
desiredStatePred True
stateOfCondition [True|False]
target     E
parent     A

L
isa        node
role       subject
name       nil
consOf     K
prevCons   nil
nextCons   M OR N ?????? HOW
parent     A

M
isa        edge
role       click
source     L
dest       C
state      [True|False]
consOf     K
prevCons   L
nextCons   nil
Competitive N
parent     A

N
isa        edge
role       remember
source     L
dest       I
state      [True|False]
consOf     K
prevCons   M
nextCons   nil
Competitive M
parent     A
```

Production pseudocode:

```
p(
    Target: K1
    State: True
=>
    =goal>
        consToDo: L
)

p(
    Goal>
        consToDo: L
=>
    createNodeL (don't like this)
    Goal>
        consToDo:M
)

p(
    Goal>
        consToDo: L
=>
    createNodeL (don't like this)
    Goal>
        consToDo:N
)

p(
    Goal>
        consToDo: M
=>
    stateOfM = true
    performClickAction

    Goal> consToDo: nil
)

p(
    Goal>
        consToDo: N
=>
    stateOfN = true
    performRememberAction
    Goal> consToDO: nil
)
```