Achieving parsimony between NGS and the Michigan approach 2012 Update

June 21, 2012

Randolph M. Jones, PhD
Bob Marinier, PhD

SoarTech

Modeling human reasoning.
Enhancing human performance.

Introduction

- The Michigan approach and NGS are often presented as opposing approaches to goal management
- There are some historical reasons for that, BUT:
  - There is no current, essential incompatibility
  - With some minor modification, they can be mutually beneficial

Michigan Approach

- Tasks are represented in a goal hierarchy
  - Operators that persist for more than one decision become goals
- The architecture allows a single decomposition stack to exist at once (the state stack)
- The architecture commits to operators (and thus decompositions) until the operator is no longer relevant or another operator is preferred
  - If operators are indifferent, Soar will commit to one, not flip-flop between them
  - Because operators become goals, this also means Soar might not interleave between “mutually indifferent goals”

Michigan Approach in Practice

- Interleaving tasks can be difficult, and there is no standardized way to manage it
- Interleaving comes at the cost of "losing the stack"
  - To be able to "pick up where we left off", superstate structures must be maintained if want to interrupt an ongoing task to do another task
- Long term "stack regeneration knowledge" implicitly represents information associated with long-term goals, but representational approaches are generally ad hoc

SOARTECH

NGS Approach

- Tasks are represented in a goal hierarchy that is maintained on the top state, rather than as operators
- Multiple goals may be active at once; goals may be i-supported or o-supported, depending on the application
- Can create complex goal-subgoal relationships, usually in a tree or a forest or a directed acyclic graph
- Operator proposals typically test for the presence of a top-state goal structure
  - Syntactically very similar to testing for an operator in the state stack
- Task interleaving schemes are easy to implement using goal priorities and/or operator preferences

NGS Approach in Practice

- NGS has so far not been used much in learning systems, and is generally designed to avoid the use of one type of operator no-change impasse
- NGS may introduce some difficulties in Soar-style impasse-driven learning
  - In particular, it can be harder to detect “no-change” types of impasses
- Some task interleaving schemes can make debugging and “threading” issues painful
  - But this may be more a property of task interleaving than of NGS
- May have to do a little extra work when you *don’t* want to interleave

SOARTECH

Combining the Approaches

- We desire a common Soar programming style (with supporting code) that mixes the UM and NGS approaches, combining their strengths
  - Task interleaving is easier and less error prone for tasks that benefit from it
  - Goal hierarchy management/rebuilding is done in a more uniform and reusable way
  - We take full advantage of Soar's impasse-driven learning mechanisms, for models that benefit from them
  - Soar coding styles and development tools naturally support the mixed approach
- Which direction should we go?
  - Should we build a library that automatically generates NGS structures from Michigan-style goal stacks?
  - Or should we build a library that automatically generates Michigan-style goal stacks from NGS structures?

NeoNGS Design Requirements

- A goal structure should persist as long as it is relevant, even when the impasse associated with the goal is (temporarily) missing from the stack
- The solution should work with all varieties of learning in Soar
- Goal structures can be either I-supported or O-supported, depending on developer/application preference
- The support received by a goal structure should never be a "surprise"
- Goal-implementation patterns should be easy to use, should foster reuse, and should not require major changes to programming style
- A particular model should easily be able to use NeoNGS for none, some, or all of its goal representations
- NeoNGS should make it easier to conceptualize and implement models that
  - Use goal hierarchies
  - Have to interleave attention between goal hierarches
- Design choice: Generate Michigan-style stacks from NGS structures
  - Primarily because of "support" requirements…generating NGS structures automatically involves "returning results" in Soar
  - Means that programmers will be writing "goal creation rules" instead of "operator proposal rules"

SOARTECH

NGS in a Nutshell

```
sp "elaborate*goal-set
 (state <s> ^superstate nil)
-->
(<s> ^goals <g>) "

sp "elaborate*goal*subgoal
(state <s> ^superstate nil
           ^goals <gs>)
(<gs> goal.subgoal <sgoal>)
-->
(<gs> ^goal <sgoal>) "
```

```
sp "elaborate*supergoal
(state <s> ^superstate nil
         ^goals.goal <goal>)
(<goal> ^subgoal <sgoal>)
-->
(<sgoal> ^supergoal <goal>) "
```

NeoNGS in a Nutshell

```
sp "elaborate*goal-set*substates
(state <s> ^superstate.goals
<goals>)
-->
(<s> ^goals <goals>) "

sp "propose*pursue-goal
(state <s> ^superstate nil
           ^goals.goal <goal>)
(<goal> ^name <name>
       -^supergoal)
-->
(<s> ^operator <o> +)
(<o> ^name <name>
     ^goal <goal>) "
```

```
sp "propose*pursue-subgoal
(state <s> ^goal <g>
           ^name <name>)
(<g> ^name <name>
     ^subgoal <sgoal>)
(<sgoal> ^name <sname>)
-->
(<s> ^operator <o> +)
(<o> ^name <sname>
     ^goal <sgoal>) "
```

NeoNGS Experiments

- Receive message to count from 1 to 10
- In the middle of counting, receive message to count from 100 to 103
- Interrupt original counting task, complete higher priority task, resume original task
- Works with chunking
- Working on experimental models for blocks world, water jugs, and robot simulator

Experiment Trace

```
 1: O: O1 (achieve-handle-message)
    2: ==>S: S4 (operator no-change)
    3:    O: O3 (achieve-count)
    4:    ==>S: S6 (operator no-change)
    5:       O: O4 (init-count)
current count = 1
    6:       O: O5 (count-1)
current count = 2
    7:       O: O6 (count-1)
current count = 3
    8:       O: O7 (count-1)
current count = 4
    9:       O: O8 (count-1)
current count = 5
   10: O: O10 (achieve-handle-message)
   11: ==>S: S9 (operator no-change)
   12:    O: O12 (achieve-count)
   13:    ==>S: S11 (operator no-change)
   14:       O: O13 (init-count)
current count = 101
   15:       O: O14 (count-1)
current count = 102
   16:       O: O15 (count-1)
current count = 103
   17: O: O1 (achieve-handle-message)
   18: ==>S: S13 (operator no-change)
   19:    O: O16 (achieve-count)
   20:    ==>S: S15 (operator no-change)
   21:       O: O17 (count-1)
current count = 6
   22:       O: O18 (count-1)
current count = 7
   23:       O: O19 (count-1)
current count = 8
   24:       O: O20 (count-1)
current count = 9
   25:       O: O21 (count-1)
current count = 10
```

Experiment Trace After Chunking

```
 1: O: O1 (achieve-handle-message)
current count = 1
current count = 2
current count = 3
current count = 4
current count = 5
    2: O: O3 (achieve-handle-message)
current count = 101
current count = 102
current count = 103
    3: O: O1 (achieve-handle-message)
current count = 6
current count = 7
current count = 8
current count = 9
current count = 10
```

Using NeoNGS to support the Michigan approach

- In the Michigan approach, handling interruptions and interleaving while maintaining the decomposition relationship relies on ad-hoc structures to store intermediate information
- NeoNGS goals can be those structures, standardizing how agents are designed to deal with these issues
- Standardization will make it easier for people to create, understand, and maintain Soar agents
  - Especially complex agents that implement task interleaving

Summary

- Nuggets
  - Significant step toward resovling/integrating NGS and UM approachess
  - UM-style behavior before chunking, NGS-style behavior after chunking
  - Better understanding of the roles of interleaving and commitments in representational choices
- Coal
  - Still not a robust package of reusable code
  - Would be nice if we can resolve automated building of goal structures from operators
  - Can we ensure the Soar development tools support the integrated approach?

BACKUP

Example “Goal Proposal Rule”

```
sp "create-subgoal*achieve-count
(state <s> ^superstate nil
           ^goals <goals>)
(<goals> ^goal <g>)
(<g> ^name achieve-handle-message
     ^message <msg>)
(<msg> ^task count
       ^params <par>)
(<par> ^start-num <start>
       ^end-num <end>)
-->
(<g> ^subgoal <sg>)
(<sg> ^name achieve-count
     ^start-num <start>
     ^end-num <end>) "
```

Example Chunk From Experiment

```
sp {chunk-7*d15*opnochange*1
    :chunk
    (state <s1> ^operator <o1>)
    (<o1> ^goal <g1>)
    (<g1> ^subgoal <s2>)
    (<s2> ^cur-num 101
          ^end-num 103
          ^name achieve-count)
    -->
    (<s2> ^cur-num 101 - ^cur-num 102 +)
}
```