Achieving parsimony between NGS and the Michigan approach

SOARTECH

Modeling human reasoning.
Enhancing human performance.

June 16, 2011

Randolph M. Jones, PhD
Bob Marinier, PhD

SOARTECH

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
  - Joint operators combine multiple tasks, but this can lead to a **combinatorial number of operators**
  - Floating operators can insert tasks into an existing hierarchy, but this **doesn't maintain the decomposition relationship**
    - E.g., if need to handle a message while performing a flight maneuver, the message handling can be inserted at the bottom of the stack, but message handling is not a natural part of the flight maneuver
- 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
    - This applies both to deliberate interleaving and GDS interruptions
    - E.g., if need to handle a message while performing a flight maneuver, can save the current state of the flight maneuver on the top state, handle the message by replacing the stack with a new decomposition, and then recreate the original stack from the saved state
- 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
- A library of rules and macros is used to maintain goal structures on the top state
- Multiple goals may be active at once; goals may be i-supported or o-supported, depending on the application
  - NGS goals capture a common set of conditions that many operators may need to test for
  - Essentially uses a structure instead of just a state name or other simple symbol
- 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
- By default, proposed operators are indifferent, so Soar naturally interleaves work across multiple goals; preferences impose goal priorities
- Supports representation of long-term goals, even if no immediate progress can be made

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
  - An operator no-change can occur either because the agent does not know how to proceed in applying an operator or because the agent needs to wait for the external environment to change in some way
  - NGS was originally conceived as an alternative method for representing "long-duration" goals
  - Thus, impasses using NGS are supposed to represent "real" knowledge deficits, rather than merely waiting for the world to change
- NGS may introduce some difficulties in Soar-style impasse-driven learning
  - It can be difficult to identify a lack of knowledge for a particular goal, as SNC impasses won't arise as long as at least one goal is making progress
  - Even if you want to use impasses, the current NGS package **doesn't support substates** (it may not be difficult to allow this)
- Debugging can be painful, as all goals and subgoals are mixed together in WM
  - Part of the idea of NGS, though, is to impose some discipline on the organization of these top-state structures
- Traditional threading issues can arise if multiple goals are modifying the same data structures

```
sp {elaborate*task*robot
   [match-root-goal <g> <s>]
-->
   [create-subgoal <sg> robot <g>]}

sp {robot*propose*patrol
   [match-goal <g> robot <ts>]
   (<ts> ^current-mission.name patrol)
-->
   [create-subgoal <sg> patrol <g>]}


sp {circuit-patrol*propose*goto-next-area
   [match-goal <g> patrol <ts>]
   (<g> -^achieved-area true)
   (<ts> ^current-mission.next-area.id <id>
         ^areas.area <area>)
   (<area> ^id <id>)
-->
   [create-subgoal <sg> goto-next-area <g>]
   (<sg> ^type go-to-area
         ^area <area>)}


sp {go-to-area*propose*go-to-next-area*plan
   [match-goal <g> <gn> <ts>]
   (<g> ^type go-to-area)
   (<ts> ^current-location.next <next-wp>)
-->
   [propose-operator <op> go-to-next-area
         <g> <ts>]
   (<op> ^next-waypoint <next-wp>)}
```

```
sp {elaborate*task*robot
   (state <s> ^superstate nil)
-->
   (<s> ^name robot)}

sp {robot*propose*patrol
   (state <s> ^name robot
              ^current-mission.name patrol)
-->
   (<s> ^operator <op> + =)
   (<op> ^name patrol)}

sp {circuit-patrol*propose*goto-next-area
   (state <s> ^name patrol
             -^achieved-area true
              ^top-state <ts>)
   (<ts> ^current-mission.next-area.id <id>
         ^areas.area <area>)
   (<area> ^id <id>)
-->
   (<s> ^operator <op> + =, >)
   (<op> ^name goto-next-area
         ^type go-to-area
         ^area <area>)}

sp {go-to-area*propose*go-to-next-area*plan
   (state <s> ^type go-to-area
              ^top-state.current-
  location.next <next-wp>)
-->
   (<s> ^operator <op> +)
   (<op> ^name go-to-next-area
         ^next-waypoint <next-wp>)}
```

NeoNGS: Extending NGS to Substates

- NGS will currently only create structures on the top state
- Extending NGS to create independent goal structures in substates should be easy
- Substates could be linked to superstate goals, providing a standard structure in which to store information and get additional context
  - In ONC impasses, Michigan approach typically just names the substate after the superoperator; additional context is provided ad-hoc

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
  - Existing NGS could work, too, so long as you only need to store information on the top state; in general, though, any superstate needs to be supported
- Standardization will make it easier for people to create, understand, and maintain Soar agents

Using Impasses to Enhance NGS-Based Systems

- All of Soar's impasse types, including operator no-changes have always been able to work, in theory, with NGS systems
- State no-changes can produce new goal creation and elaboration knowledge
- Operator no-changes still reflect a lack of knowledge in proceeding with a deliberate action
- Ties represent a lack of prioritization knowledge in attending to a single goal or interleaving across goals
- In practice, we have not built systems that do these things, and NeoNGS should make sure there are no surprises

SOARTECH

Way Forward

- Clean up the existing NGS
  - Make sure it includes whatever features people want from their customized versions
  - Possibly remove or factor out little-used features, to reduce overhead and cruft
- Create NeoNGS
  - Remove unnecessary "superstate nil" tests from various rules/macros
  - Fix whatever problems this causes
  - Add new rules/macros to add new supporting structures to substates, like links to NGS goals in superstates
- Convert one or more Michigan approach agents to use NeoNGS
  - E.g., water-jug-look-ahead: keep the task hierarchy, but add creation of appropriate NGS goals at each level
  - Ensure that impasses and chunking still work
  - Perform some kind of evaluation: performance tests, number of rules, etc.
- Open source NeoNGS + Tcl code necessary to make macros work