Radical Randy Revisited

Top-State Goal Trees in a Real Application

Randolph M. Jones

Soar Technology/Colby College

Soar Workshop 2003
June 25, 2003

Soar Technology
Thinking *inside* the box.

Background

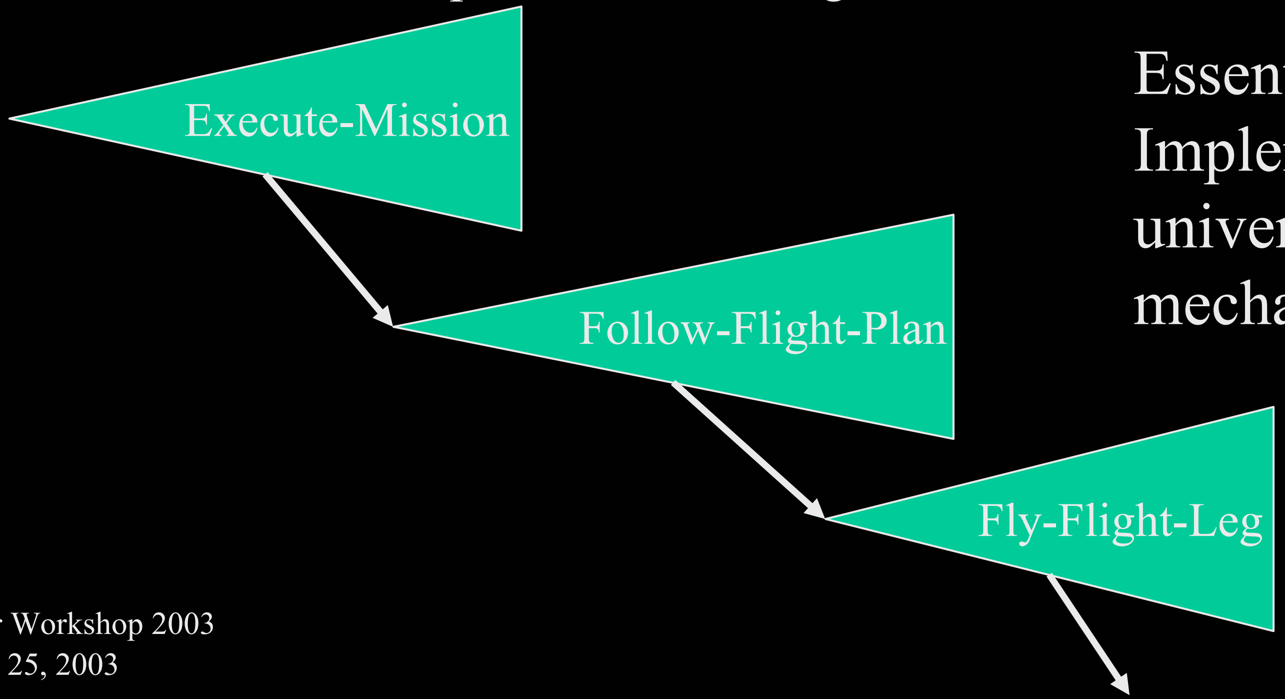
- Eight years ago or so, we used to have discussions about different ways to represent goals in Soar
  - And what the role is (or ought to be) of operators in Soar
- One proposed approach was "Goal Trees" (or "The Radical Randy Approach)
  - But only a shallow research investigation
- Now this approach has been used in a "real" agent system

Application Area

- Intelligent agent to serve as an automated wingman for Army Rotary-Wing Aircraft missions
  - Soar 8.3
  - Hooked up to MÄK's VR-Forces simulator
  - Using gSKI
  - Writing behaviors from scratch, but relying on lots of "conceptual reuse"
    - From TacAir-Soar and RWA-Soar
      - Both written in Soar 7
      - Alternative to trying to re-engineer code for a new application *and* a new architecture

Review of “The Michigan Approach”

- The most common way to represent task goals in Soar:
  - Select an operator
  - If the operator represents a “high-level” action (i.e., it takes time to achieve), it cannot immediately execute, so an impasse generates a subgoal
  - Select an operator in the subgoal, etc.



Essentially a goal stack Implemented by Soar’s universal subgoaling mechanism

Functional Concerns

- What is the best way to represent trees of goals?
- What happens if I want an operator relevant to a "high" goal to get selected without destroying the rest of the goal stack?

Philosophical Concerns

- Why should my agent still need to generate operator no-change impasses even after it has learned?
- Is an operator an atomic action or not?

Psychological Concerns

- Who ever came up with the idea that an accurate model of the mind would have a goal stack in it?
  - If there were ever any doubts, Altmann and Trafton have made a good case to put them to rest
- There's no reason to think that "active goals" should be represented any differently from other active working-memory elements

Laziness Concerns

- In Soar 8, I need enough information on the top state to allow myself to regenerate a goal stack any time it might get interrupted
  - The goal stack is redundant
  - Why not just use the top-state structure and not bother to duplicate it in a goal hierarchy?

A Simple Example

```
S1 ^goal G1
S1 ^goal G2
S1 ^goal G3
G1 ^name execute mission
G1 ^subgoal G2
G1 ^subgoal G3
G2 ^name follow-flight-plan
G3 ^name follow-leader
```

Code Examples

```
sp {top-state*elaborate*goal*subgoal
   (state <s> ^name top-state
              ^goal.subgoal <sg>)
-->
   (<s> ^goal <sg>)
}

sp {execute-mission*subgoal*follow-flight-plan
   (state <s> ^goal <g>
              ^flight-plan.active *yes*)
   (<g> ^name execute-mission)
-->
   (<g> ^subgoal <sg>)
   (<sg> ^name follow-flight-plan)
}
```

What Happens To Operators?

- Operators intentionally only remain selected for a single decision
  - It is still appropriate to learn operator implementations in some cases
  - But you will not (and should not) get operator no-change impasses after learning
- Operators for independent goals interleave at the top state
  - (If you want them to)
  - There are also opportunities to engineer operator tie or conflict impasses based on contention for resources

What About Those Psychological Concerns?

- This scheme works in part because the truth maintenance system automatically maintains (and cleans up) goal relationships
  - This is still not psychologically plausible
  - But the method puts goals and other working-memory elements on an equal footing
    - Any architectural changes that address the psychological validity of working memory will also affect goals

Mineralogical Assessment

- Coal:
  - A rigorously objective evaluation (or comparison to alternatives) has not been performed
- Gold:
  - Based on a rigorously subjective evaluation in a small but “real” system, the method works extremely well
    - Particularly aids software engineering

- Coal:
  - An operator cannot learn to send multiple simultaneous output commands
- Gold:
  - An operator cannot learn to send multiple simultaneous output commands