Meta Programming and Soar

Jacob Crossman
Soar Technology, Inc.
jcrossman@soartech.com

Basic Form of Beliefs/Goals

```
  (state <s>   ^operator <o>
               ^game-state <gs>)
   (<o>        ^name evaluate-action
               ^action     <act-root>
               ^tmp-state <new-ws>)
   (<act-root> ^processed *yes*
               ^action <action>
               ^annotations <as>)
    …
    $lhstest
-->
    (<as>    ^applied-belief <ab>)
    (<ab>      ^id           $name
               …)
    $rhsupdate
```

- Four Parts:
  - Plumbing interface – same for all related beliefs
  - High-level Test – what the agent really cares about
  - Output interface – where results must go
  - Real Output – meaningful change to working memory

Issues with Beliefs/Goals

- Duplication: 90-95% of most beliefs and goals the same
- Typing/Cut & Paste errors
- Heavy coupling to infrastructure – I know where you keep your game state
- Resistant to change/modifications – Can I please just change this one structure?
- Human readability – What exactly does that belief mean?

CS/SE Solutions

- Computer Science and Software Engineering have useful solutions for most of these problems
  - Modules
  - Encapsulation
  - Interfaces
  - Indirection/Proxies
  - Generic Programming – newest and most promising for Soar

Tools Available

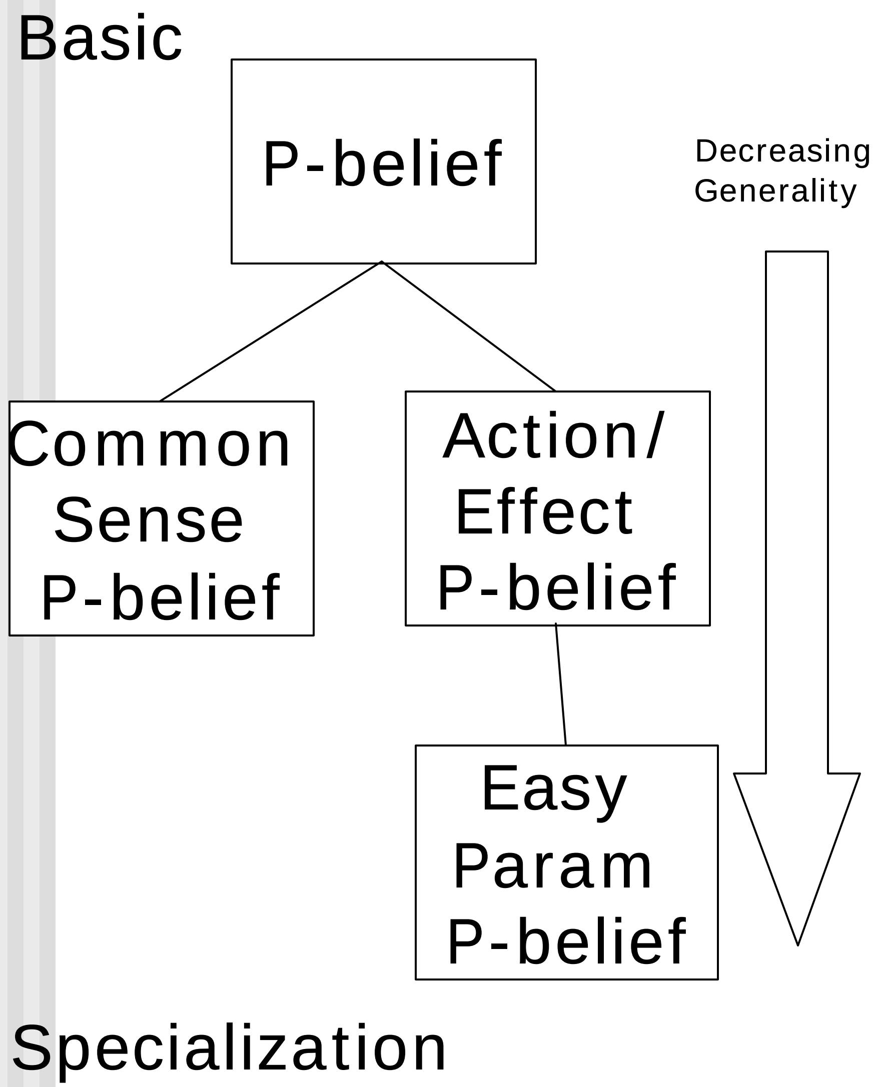
- Glenn (but he occasionally goes home)
- Soar
  - Provides the problem space at runtime
  - Not much at rule definition/load time
- TCL
  - Tied to Soar right now
  - Very good string processor – useful for meta programming
  - Constants
  - Procedures

TCL Templates

```
Proc gen-belief { action country ... } {
  (state <s>  ^operator <o>
              ^game-state <gs>)
  (<o>        ^name evaluate-action
              ^action    <act-root>
              ^tmp-state <new-ws>)
  (<act-root> ^processed *yes*
              ^action $action
              ^annotations <as>)
   …
   $lhstest
-->
   (<as>    ^applied-belief <ab>)
   (<ab>    ^id            $name
            …)
   $rhsupdate
}
```

- TCL procedure containing plumbing specific code
- Parameters for common variations
- LHS and RHS production “fragments” for customization
- Side Note: you can (and should) use defined constants

Building Abstraction



- You can compose them!
- Lets you build up abstraction
- Is:
  - Procedure call
  - String replacement
- Similar to:
  - Template Instantiation
  - Aggregation

Some Stats

- Number of templates: approx. 40
- Number of productions per template: 1-8 (average about 2)
- Current instantiations per agent: about 100-150
- Number of agents: 10
- Approximate number of rules per agent 150-300 (beliefs/goals only)
- Total rules (all agents) on the order of 2000

What do I get? (Nuggets)

- Much less typing – fewer errors
- Soar-lover code → Simpler template instantiations
- You can quickly and *safely* change:
  - Infrastructure
  - Techniques and strategies (single production v.multi-production, operator v. elaborations)
- Helps manage complexity

What’s the catch? (Coal)

- Right now, must use TCL
  - Slow
  - Not type safe
  - Not really generic programming! More like macros.
  - TCL doesn’t really understand Soar
  - Many people don’t know (or want to know) TCL
- Can cause rule explosion (if you care about lots of rules)