Thoughts on the future of HLSR

Bob Marinier, SoarTech

29th Soar Workshop

June 2009

What is HLSR?

- High-Level Symbolic Representation language for cognitive architectures
- An attempt to reduce the cost of developing behavior models by:
  - Reducing the amount of code one needs to write
  - Making code easier to understand and maintain
- A compiler (HLSR language->Soar/ACT-R)
  - Each HLSR language construct has a microtheory that dictates how it is realized on a particular architecture

Primary HLSR constructs

- Relations (type definitions)
  - Can instantiate as facts
  - Can retrieve facts of particular types
- Transforms (sequences)
  - Explicit support for atomic sequences
- Activation tables (decision logic)
  - Puts related logic together in one place

Relations: Example

```
relation Disk(name isa string, size isa integer)
relation Peg(name isa string)
relation NextSmallestDiskOnPeg(current isa Disk, next isa Disk, peg isa Peg)
(
    SmallerThan(next, current)
    DiskIsOnPeg(next, peg)
    forall d isa Disk
      if (SmallerThan(d, current))
      then (!SmallerThan(next, d) or !DiskIsOnPeg(d, peg))
)
```

- Current
  - Strongly typed
  - Can be "i-supported" or "o-supported"
  - No extra conditions necessary to connect to state
- Future Work
  - Mutability (e.g., null)
  - Enumerations
  - Type hierarchies

Transforms: Example

```
transform UpdateMapCell(mapCell isa MapCell, contents isa string) (
    consider-if(MapCellOutdated(mapCell, contents))
    body(
        new<MapCell>(mapCell.x, mapCell.y, contents)
        reconsider(mapCell)
    )
)
```

- Current
  - Atomic sequences
  - Can be invoked explicitly (like a function call) or automatically (consider-if)
- Future Work
  - Branching
  - Unordered actions (esp. for output)
  - Polymorphism

Soar: Covering a space of conditions for possible actions

- Propose an operator for each action
- Each operator proposal contains some combination of conditions
- Seeing what parts of space are covered is hard
  - Conditions are spread across separate proposals
  - Proposals often spread out all over multiple files (VisualSoar encourages this, SoarIDE doesn't help prevent it)
  - Ex: TankSoar simple-bot selects the top-level goal using 7 proposals across 4 files

HLSR: Covering a space of conditions for possible actions

- Insight: condition combinations are like a *truth table*, which can be compactly represented
- HLSR embodies this in an *activation table*
  - Cross-cutting logic (aspect-oriented programming)

Activation tables: Example

```
# This is (almost) the exact logic from TankSoar simple-bot
activation-table SelectGoal (
   conditions (
      1: MissilesEnergyLow()
      2: sensed(Incoming(true, *))
      3: InRadarContact()
      4: sensed(Sound("silent"))
   )
   actions (
      [F*T*]: (new-goal<OutOfRadarContact>()) # Attack
      [F*FF]: (new-goal<InRadarContact>())    # Chase
      [*FFT]: (new-goal<InContact>())         # Wander
      [T**F]: (new-goal<OutOfContact>()) # Retreat
      [T*T*]: (new-goal<OutOfContact>()) # Retreat
      [TT**]: (new-goal<OutOfContact>()) # Retreat
      [*TFT]: (new-goal<OutOfContact>()) # Retreat
   )
)
```

Activation tables

- Current
  - Related logic grouped together in one place
  - Easier to see coverage
- Future
  - IDE support for coverage
  - Support for context conditions

Learning in HLSR

- Learning is not currently implemented in HLSR, but there are at least two ways it could be supported
  - Learning at the HLSR level
    - E.g., RL-like mechanism for tuning which action to execute when there are multiple options
  - Learning at the microtheory level
    - HLSR compiles to generic constructs; architectural learning mechanisms can improve those "behind the scenes"

Some other things to note (more nuggets and coal)

- Current
  - Supports OR logic
  - In principle, can support multiple microtheories
- Future
  - Improve goal semantics
  - Improve generated code (less verbose/more efficient in terms of operators)
  - Stability improvements (develop larger, more complex agents)
    - Note: support for more complex constructs undermines stability