Human-like artificial creatures

9. Soar

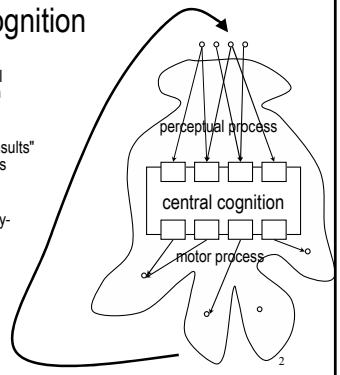
Cyril Brom
Faculty of Mathematics and Physics
Charles University in Prague
brom@ksvi.mff.cuni.cz

Soar – an architecture for human cognition

1. External stimuli evoke internal symbols (representation of an environment)
   - perceptual process
2. Central cognitive system "consults" the representation and creates motor symbols
   - if-then rules
3. Internal symbols cause a body-change (i.e. change in the environment)
   - motor process

- Other stuff...
  - priorioception, encoding productions...

[Newell, 1990]



Soar – central cognition

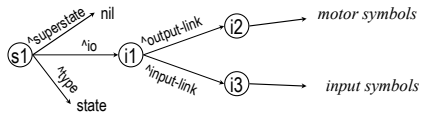
1. Input phase
   - inputs are stored in the working memory
2. Proposing phase
   - various if-then rules propose what to do next
3. Decision phase
   - just one proposal is chosen
4. Applying phase
   - new internal symbols or motor symbols are created by an if-then rule
5. Output phase
   - motor symbols are interpreted and the state of the environment is changed

Central cognitive system

- the architecture involves
  - a programming language
  - a working memory (symbols - "variables")
  - a long-term memory (if-then rules)
  - a learning

Working memory

- Contains all of a Soar-entity's information about its world and its internal reasoning.
- All information is organized as **states**
  - the simplest applications need only one state
- Working memory is a graph structure with nodes, connected by links.
  - identifiers, constants
- if-then rules match the elements of the working memory (and nothing else)



Proposing phase

- All rules firing in this phase create non-persistent structures in the working memory.
- When the condition change, the working memory element is retracted.
- An ordinary rule proposes an **operator**.
  - "Auxiliary" rules are possible (e.g. elaboration rules)

a bounded variable

```
sp {propose*move-north
    (state <s> ^io.input-link.eater <e>)
    (<e> ^x <x> ^y <y>)
-->
    (<s> ^operator <o> +)
    (<o> ^name move-north
         ^actions.move-north
    )}
```

condition part

action part

Decision phase

- The decision procedure decides in virtue of operators' preferences
  - acceptable: a+ (only acceptable proposals will be considered by the decision procedure)
  - indifferent: a = b
  - better: a > b
  - worse: a < b
  - the best: a > all
  - the worst: a < all
  - reject: a-

⇨ eaters

- What happens if the procedure can not decide?
  - an impass!

Applying phase

- Rules firing in this phase create persistent structures in the working memory.
  - application rules, recording/deleting rules, ...
- An ordinary generic application rule just copies the action of selected operator to the output-link
  - the action of each operator instance can be copied only once (status completed)

```
sp {apply*operator*create-action-command
   (state <s> ^operator.actions <att> <value>)
   (          ^io.output-link <ol>)
-->
   (<ol> ^<att> <value>)
   }
```


Impasses

- Situations that prevent Soar from "moving forward"
- Four types:
  - operator no-change: an operator is proposed, but there is no apply-rule that would match with the operator in the current situation
  - state no-change: no operator is proposed
  - operator tie: multiple operators are proposed, but insufficient preferences to select between them
  - operator conflict: multiple operators are proposed, and the preferences conflict
- How does Soar solve an impass?
  - a new state is automatically created

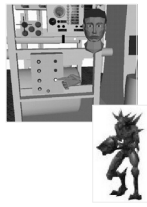
Substates and chunking

- Substates' purpose is to provide a context for selecting and applying an operator to resolve an impasse.
- A context stack is maintained
- How to resolve an impass?
  Soar allows for combination of various kinds of solving an impass
  - an additional rule can apply
    - e.g. behavioural decomposition ⇨ soartank
  - rules are evaluated by preferences – a selection problem and an evaluation problem
    - e.g. look ahead-planning, iterative deepening
- The impass resolution can be remembered: **chunking learning**
  - a new rule is recorded into the long-term memory
  - a backtracing algorithm
  - an overgeneralization problem

```
sp {wander*propose*move
   (state <s> ^superstate.operator.name wander
   (          ^io.output-link.blocked forward no)
-->
   (<s> ^operator <o> + =)
   (<o> ^name move
        ^actions.move.direction forward
   )}
```


Applications

- Steve
  - an educational agent
  - military scenarios
- Soarbot
  - anticipation
  - 100 operators (20 of them with substates), 715 rules altogether
- bots for military simulations
- TacAir Soar
  - agents for flight simulations
  - 7500 rules
- cognitive research
- ...



Evaluation

PLUS

- Powerful programming vehicle
- Rules-matching algorithm based on RETE
- Learning as an architectural build-in
- Universal
  - combinations of reactive and planning techniques allowed
- 20 years of history
  - tutorials, documentations
  - it works without bugs!
  - current release: 8.6.0
- A formal theory underlies it

CONS

- Low-level
  - hard for non AI experts
  - "programming in Soar resembles brain surgery"
- Fuzzy-rules not allowed
- Perception process / motor process not addressed
  - connecting Soar to an own world is not straightforward

References

1. Newell Allen. Unified Theories of Cognition. Harvard University Press, USA (1992)
2. Soar homepage (university): http://sitemaker.umich.edu/soar
   (tutorials, introductions, documentations)
3. Soar technology (industry): http://www.soartech.com/company.php