The Phase System

Plan Representation in Soar

Sean Lisse


Soar Technology

Challenge

- Create an in-Soar-Agent formalism for a set of tasks - easy to:
  - Program
  - Understand
  - Debug
- Allow those tasks to be modified on-the-fly by an agent

Real-World Task Example

- Agent is given two concurrent tasks to accomplish:
  - Driving a car
  - Talking on a cell-phone

Solving example via Goal Stack

- In top-state, when the way is clear
  - start driving the car
  - When the car is going
    - dial the phone
  - When the phone is dialed
    - talk on the phone
- When you reach your destination, stop driving
  - (and talking… oops!)

The Phase System

- Roots in engineering, not academia
- Similar to “sketchy planners”
  - High-level actions laid out on top-state
  - Low-level actions implemented as operators
- Stores plan info on the top state
- Consists of three primary mechanisms
  - *Phases*
  - *Conditions*
  - *Tasks*

Phases (Plans)

- *Ex: "Drive the car", "Talk on the phone*
- Represent a logical or high-level step of a plan
- May be *active, inactive, complete, or aborted*
  - Active/inactive: Transient (I-support)
  - Complete/Aborted: Permanent (O-support)
- If *complete* or *aborted,* are by definition *inactive.*
- May contain
  - *Subphases*
  - *Conditions*
  - *Tasks*

Conditions (Preconditions/Constraints)

- *Ex: "The way forward is clear", "The phone has battery left"*
- Predicates that control phase activation
- Represent restrictions on when a phase becomes active
- Can have parameters, which vary by condition type
- May be
  - One of 4 'classes'
  - One of any number of 'types'

Conditions: ‘Classes’

- Precondition
  - *Ex: (Driving) I have the keys in my pocket*
  - Must **all** be satisfied in order for a phase to become active
- Invariant
  - *Ex: (Phone) The phone has battery left*
  - Must **all** be satisfied for a phase to become and remain active
- Postcondition
  - *Ex: (Driving) I am at my destination*
  - When **all** satisfied, mark the phase complete
- Abort-Condition
  - *Ex: (Phone) The phone has gone dead*
  - When **any** satisfied, mark the phase aborted

Conditions: 'Types'

- Conditions are characterized by their ^type attribute – Some possible condition types:
  - Time based
    - *Ex: It's after 4 PM*
  - Sequence based
    - *Ex: I completed starting the car*
  - Stimulus based
    - *Ex: The phone has gone dead*

Tasks (Actions/Operator Proposal Triggers)

- *Ex: Dial the phone number*
- Represent basic activities an agent can perform in the world
- Characterized by
  - ^*type* attribute
  - Parameters
- With parent phases, are “operator triggers”
- Can be marked complete by triggered operator

Implementation: <dial-phone> about to activate

- ^phase <dial-phone>
  - ^precondition — ^active *yes*
    - ^type dial-tone
    - ^which-device <phone>
  - ^invariant — ^active *yes*
    - ^type power-on
    - ^which-device <phone>
  - ^postcondition — ^active *no*
    - ^type task-complete
  - ^task <dial-phone-task> — ^completed *no*
    - ^type phone-dialing
    - ^phone-number (ZZZ) XXX-YYYY

Nuggets

- Allows independent tasks to remain independent
  - Gives a specific place to store relevant information re: progress of an action
- Flexible structure
  - Reuse conditions and tasks in myriad combinations
  - Allows creation of task and condition libraries
- Debugging is easy
  - Look for the ^active flags
- I-support for activation
  - Supports automatic rollback and propagation
- Introspection
  - Explicit goals and constraints allow agents to create plans on the fly and reason over them
- Fits within current Soar Architecture

Coal

- More complicated than goal-stack-only Soar
- Where to draw the dividing line between phase system and operators?
- Only exit conditions for phases/tasks are “aborted” and “completed”
- Conditions, tasks, and phases are not always cleanly separable
- Chunking?

Future Directions + Possibilities

- More automated plan generation
  - Agents that explicitly work toward fulfilling or avoiding conditions
- Better error reporting by agents
- High-Level Editor/IDE for composing plans?
- Research performance effects of the design
- Work in Robustness and error recovery
- Incorporation of advanced concepts from planning literature

Acknowledgements

- The GDRS project, for funding development of UV-Soar and with it this Phase System
- Dr. Paul Nielsen, for his help in threshing out the ideas for the system and identifying weaknesses in earlier approaches

End of Presentation

Implementation – ^Phases

- ^phases
  - ^phase <drive-car> *(active)*
  - ^phase <talk-on-phone> *(active)*
    - ^subphase <dial-phone> *(active)*
    - ^subphase <talk-to-mom> *(inactive)*
  - ^current-phase <drive-car>
  - ^current-phase <talk-on-phone>
  - ^current-phase <dial-phone>

Implementation: Inside <talk-on-phone>

- ^phase <talk-on-phone>
  - ^name |Talk on the Phone|
  - ^abort-condition
    - ^type out-of-batteries
    - ^which-device <phone>
  - ^invariant
    - ^type in-possession
    - ^which-device <phone>
  - ^postcondition
    - ^type subphases-complete
  - ^subphase <dial-phone>
  - ^subphase <talk-to-mom>

Implementation: Inside <dial-phone>

- ^phase <dial-phone>
  - ^name |Dial the Phone|
  - ^precondition
    - ^type dial-tone
    - ^which-device <phone>
  - ^invariant
    - ^type power-on
    - ^which-device <phone>
  - ^postcondition
    - ^type task-complete
  - ^task <dial-phone-task>
    - ^type phone-dialing
    - ^phone-number (ZZZ) XXX-YYYY

Implementation: 'Pull-Up Recursion'

- Conditions of top-level and *potential* phases constantly truth-maintained
- When its conditions warrant, a potential or top-level phase is added as a current phase
- When a phase is current, its subphases become potential phases