New SVS Overview

Joseph Xu

Soar Workshop 31

June 2011

Outline

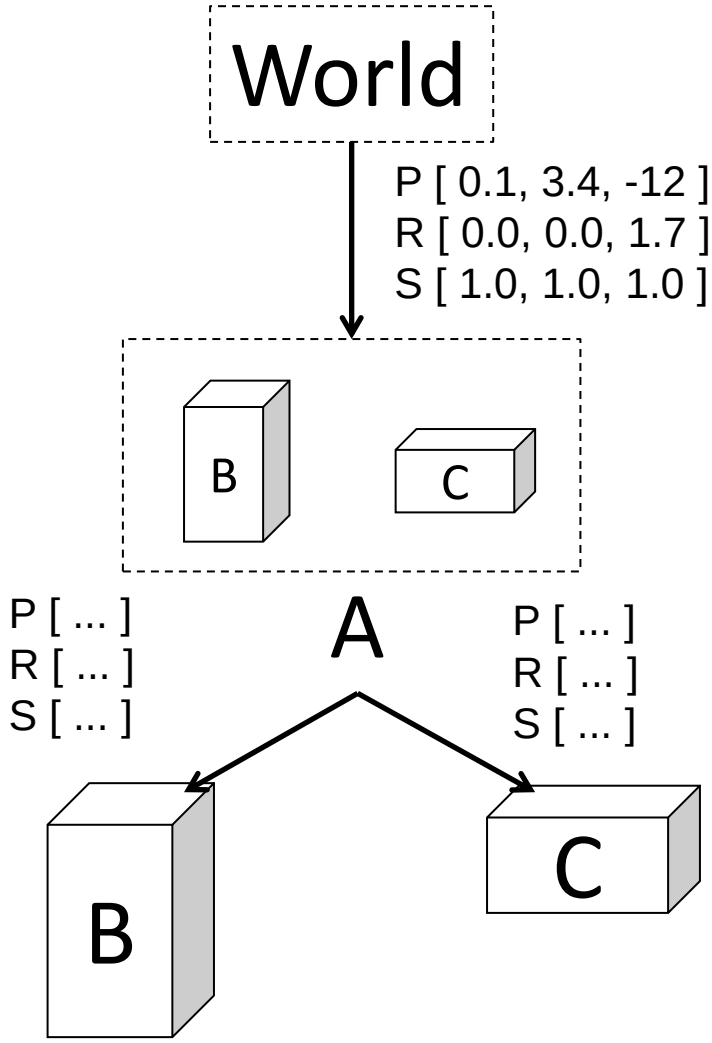
- Functionality in old SVS
- Addition of continuous control
- New implementation
- Conclusions

The Gist of SVS Past

- Hypothesis: Important properties of continuous environments cannot be completely captured with purely symbolic representations
- Use continuous representation to supplement symbolic reasoning
  - Continuous spatial scene graph
  - Symbolic Soar extracts information by asking questions in the form of spatial predicates
  - Symbolic Soar modifies scene graph using imagery commands

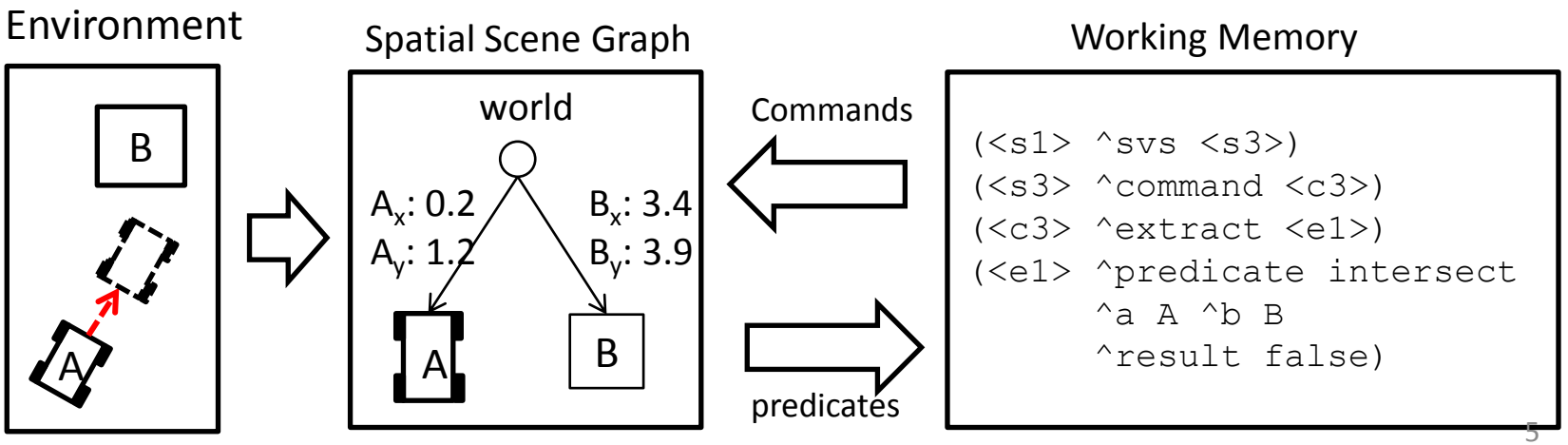
Spatial Scene Graph

- Set of discrete objects
- Hierarchy of “part-of” relationships
- Each object is transformed relative to parent in terms of position, rotation, and scaling
- Each leaf object has a concrete geometric form



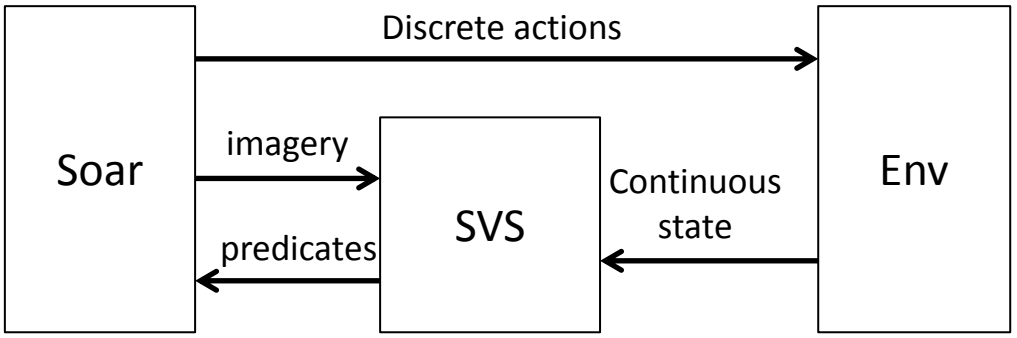
SVS Commands

- Predicate extraction
  - Ask a question about the current state
    - Is car A intersecting box B?
- Imagery
  - Imagine a change to the state for the purpose of reasoning about it
    - If A is to the right of B, is it to the left of C?
- SVS is only a mechanism, it's not smart
  - Needs knowledge about which predicates to extract and imagery operations to perform



Applications

- Applied to domains where spatial properties of state are important
- Controlled with given discrete sets of actions
- One step predictions using ad-hoc model learning methods



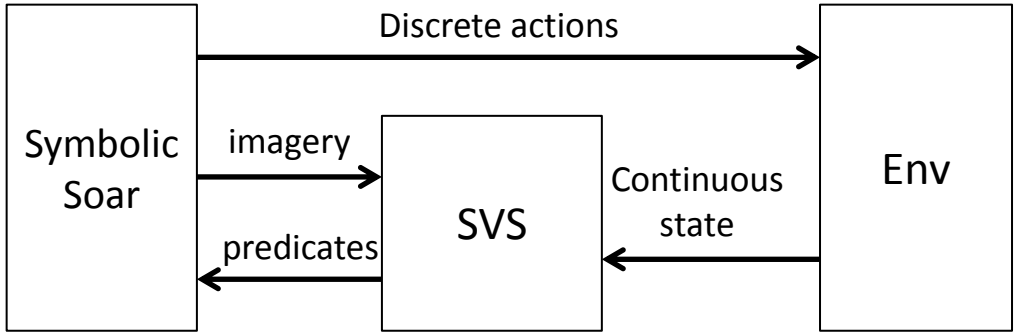
Continuous Control

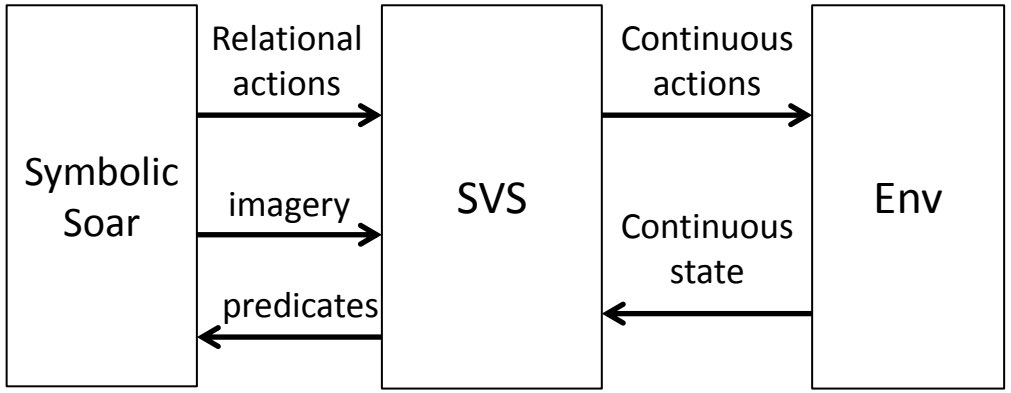
- Many real world environments expect continuous control signals from agent
  - Example: robot domain expects left and right motor voltages
- Traditional approach is to hand-code middleware to translate set of discrete actions into continuous output
  - Action discretization is a priori, leading to non-adaptive and non-optimal behavior
  - Not part of the cognitive architecture theory
  - Need new middleware for every new environment

My Motivation

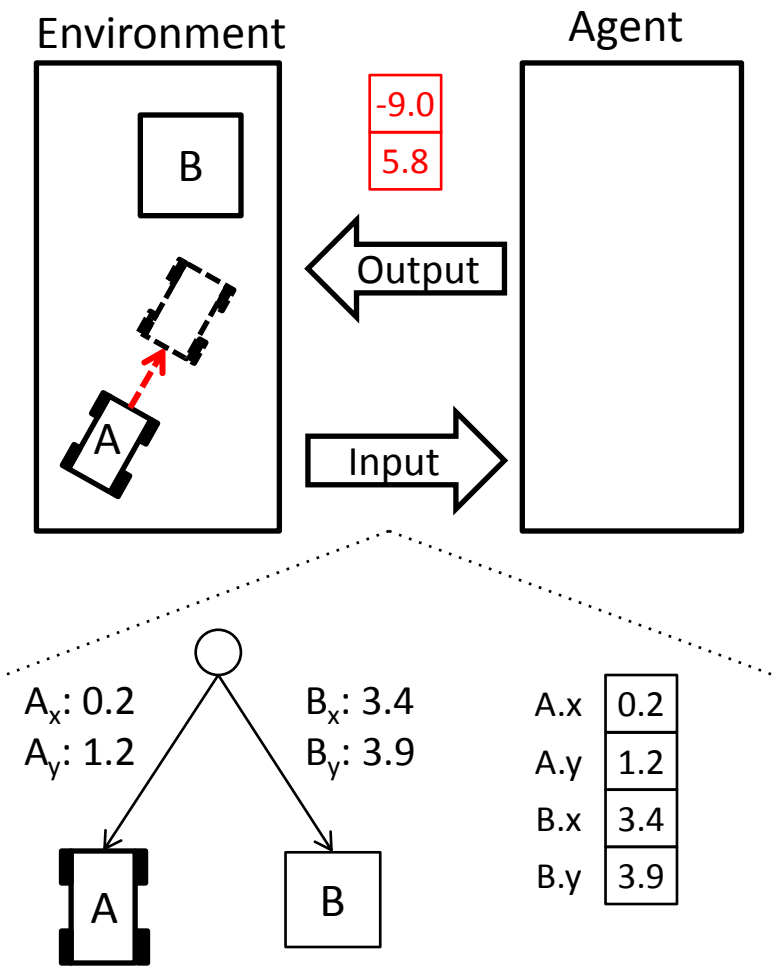
Augment SVS to allow agents to automatically learn continuous control

- Agent autonomously derives a set of relational actions that it can plan over symbolically
- SVS learns how to translate relational actions to continuous output





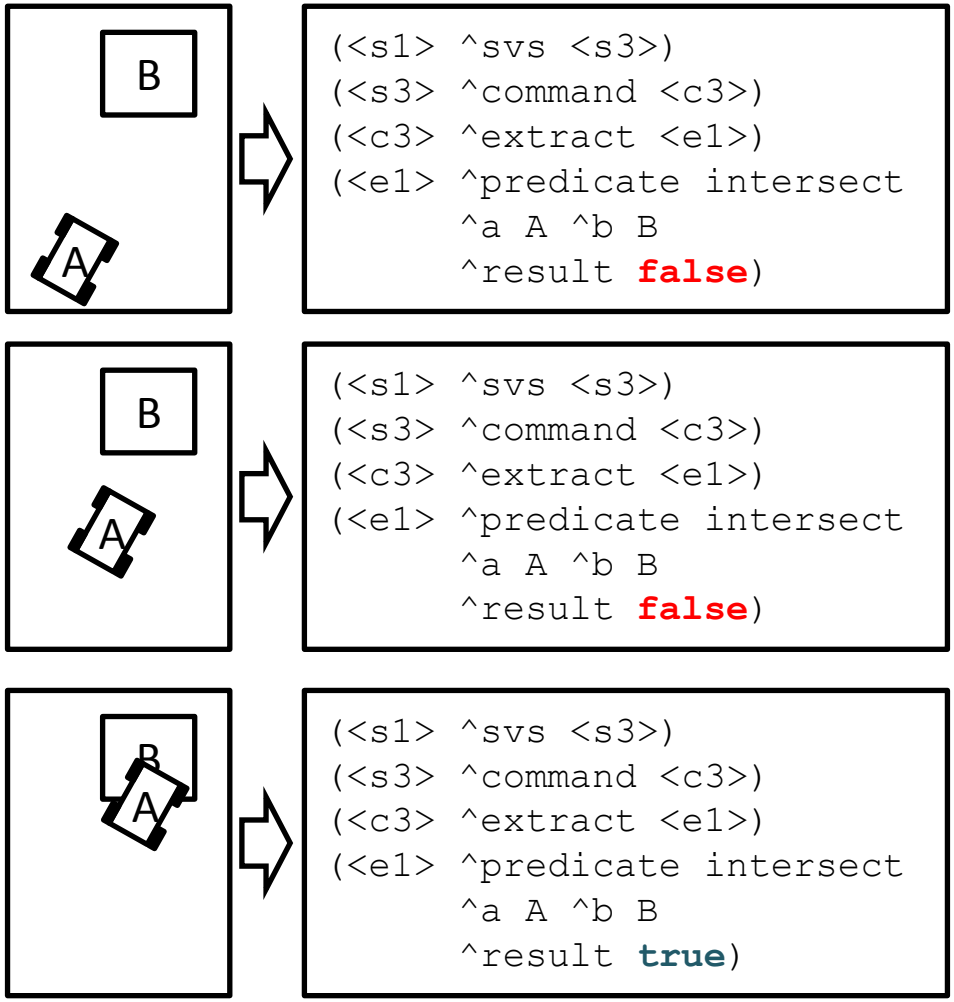
Environment Assumptions



- Input to the agent is a scene graph
- Output is fixed-length vector of continuous numbers
  - Agent doesn't know a priori what numbers represent
- Agent runs in lock-step with environment
- Fully observable
- Some noise tolerable

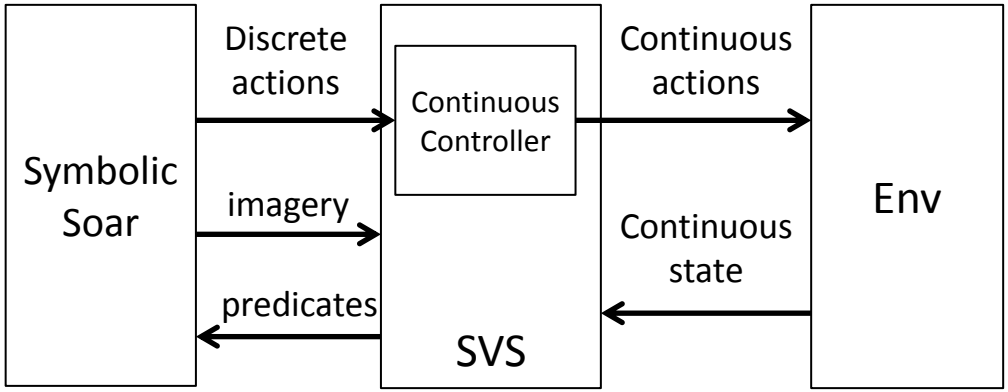
Relational Actions

- The value of an extracted predicate is the smallest unit of change that's distinguishable to symbolic Soar
- Each potential predicate value change is a *relational action*
  - Combinations of predicates?
- +intersect(A, B)



Continuous Controller

- Takes a relational action and translates into a multi-step trajectory of continuous-valued outputs to environment $(u_1, u_2, \ldots, u_n)$
- One predicate change takes multiple decision cycles
- May fail to find a trajectory that changes predicate value



Continuous Planning

- Find the trajectory that will lead to the predicate change the fastest

Continuous Model

$f(x, u) \rightarrow y$

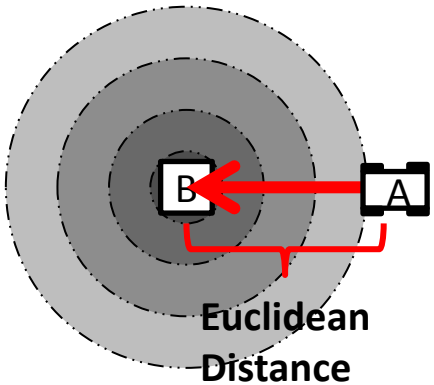
Learned (subject of next talk) or hand-coded

Vector of transform values in flattened scene graph

Objective Function

$g(x) \rightarrow [-\infty, \infty]$

Given a priori for each predicate



intersect(A, B)=true

Control Function

$H_{f,g}(x) \rightarrow u = argmin_u[g(f(x, u))]$

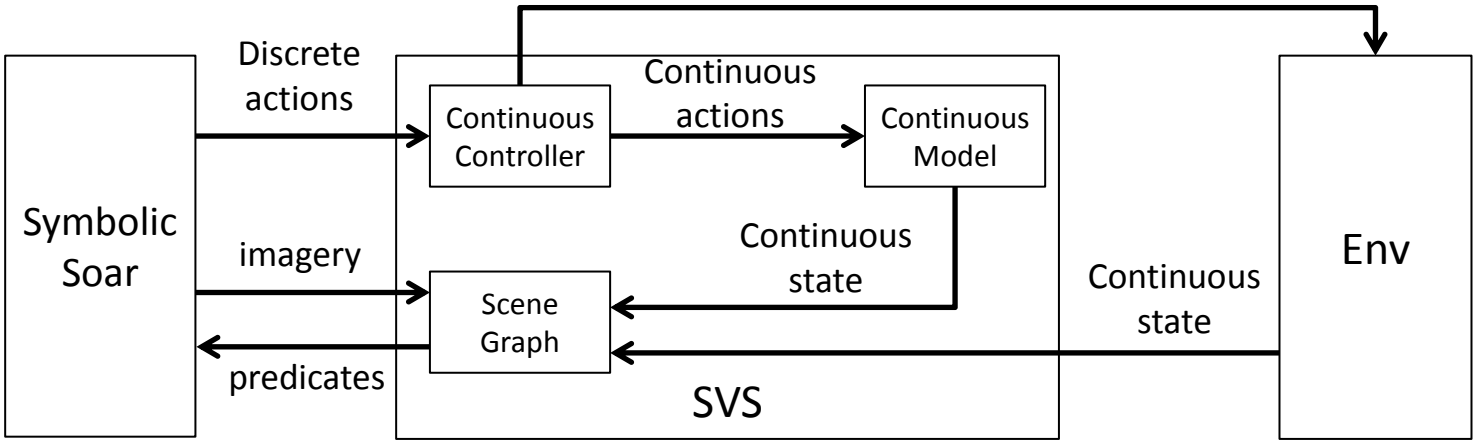
Fixed in architecture

```
GreedySearch(s, objective):
    for u in sample(outputs):
        y = continuous-model(s, u)
        o = objective(y)
        if o < best:
            best = o
            best-output = u
    return best-output
```

Downhill Simplex

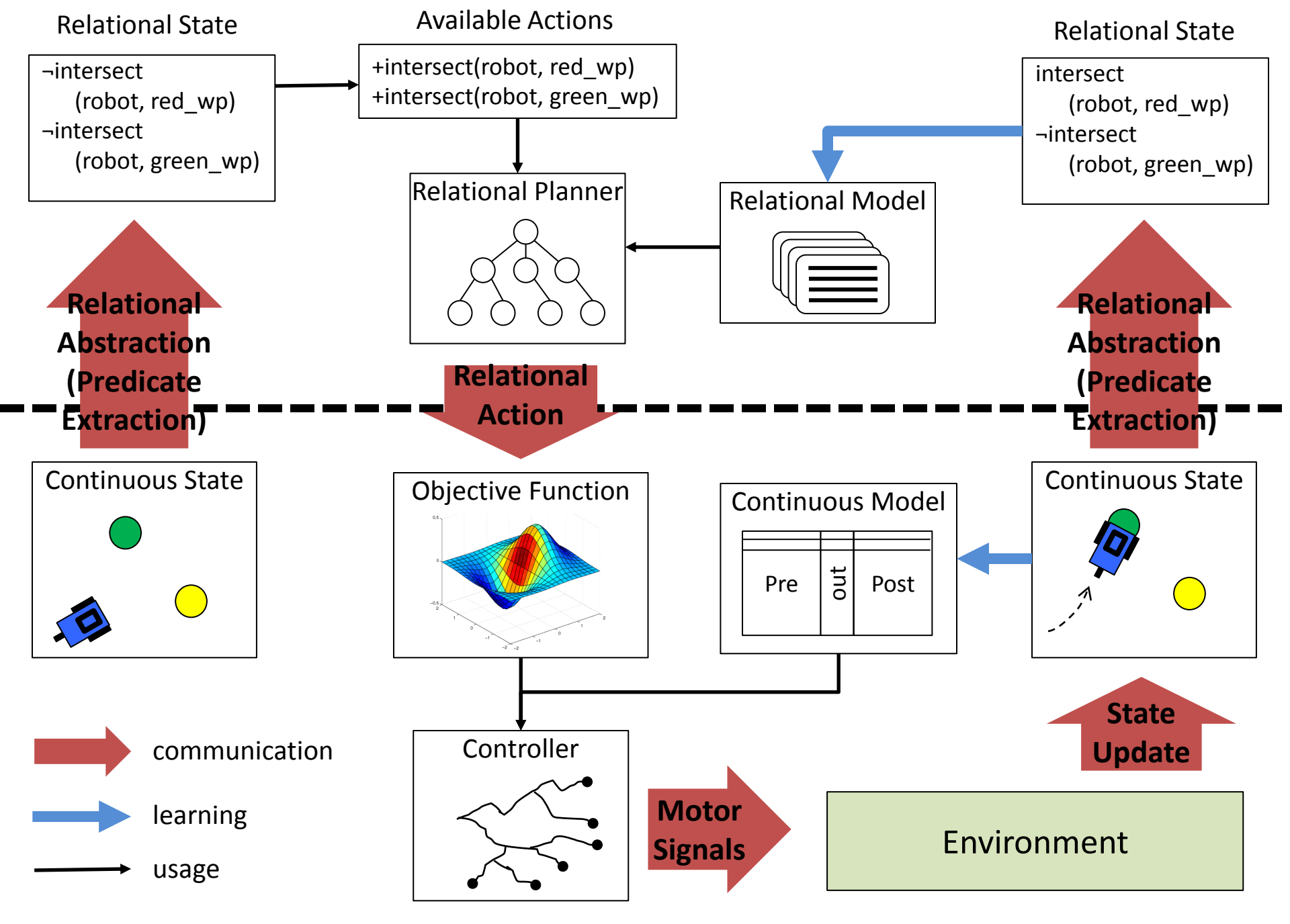
Imagery with Control

- Agent can perform relational prediction by simulating a trajectory using continuous model on the internal scene graph, instead of sending it out to the environment
- Modified scene graph updates predicates as usual



Summary

- Sam focused on theory of translating continuous environment state into relational symbolic representations
- I've added theory about where relational actions come from and how to ground them into continuous trajectories in the environment
- Ultimate goal: agents that can learn relational abstractions and plan over them in continuous environments


Relational State
¬intersect
(robot, red_wp)
¬intersect
(robot, green_wp)
Available Actions
+intersect(robot, red_wp)
+intersect(robot, green_wp)
Relational State
intersect
(robot, red_wp)
¬intersect
(robot, green_wp)
Relational Planner
Relational Model
Relational Abstraction (Predicate Extraction)
Relational Action
Relational Abstraction (Predicate Extraction)
Continuous State
Objective Function
Continuous Model
Pre
out
Post
Continuous State
State Update
Controller
Motor Signals
Environment
communication
learning
usage

New Implementation

- Integrated into kernel rather than communicate on IO link
- Removes dependencies on external libraries (WildMagic, CGAL) to ease installation
- Structured code to be extensible
  - Easy to add new predicates, commands, models
- Main idea: keep it clean enough that future students won't throw it away and start over (happened twice already)

Nuggets & Coal

Nuggets

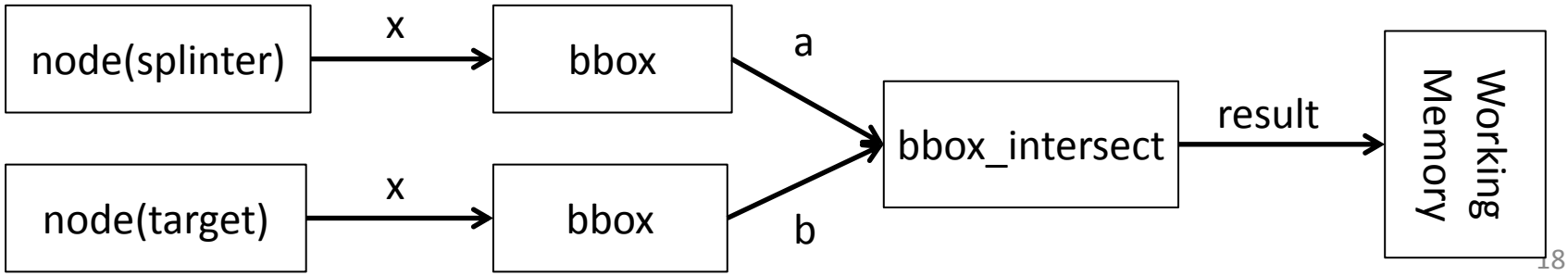
- Adds a task independent mechanism for continuous control
- Usable implementation, soon to be released

Coal

- Dropped some functionality from previous implementations
- Not optimized, haven't measured performance in non-trivial domains

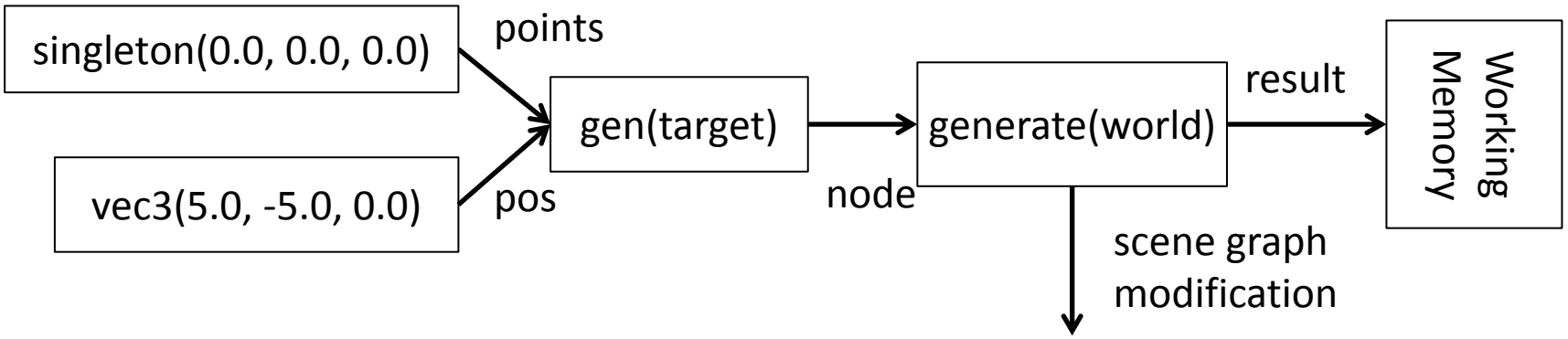
Extract Rule

```
sp {cursor-target-intersect
   (state <s> ^superstate nil
              ^svs ( ^command <c>
                     ^spatial-scene (
                         ^child.id splinter
                         ^child.id target)))
-->
   (<c> ^extract <e>)
   (<e> ^type bbox_intersect ^a <a> ^b <b>)
   (<a> ^type bbox ^x <n1>)
   (<n1> ^type node ^name splinter)
   (<b> ^type bbox ^x <n2>)
   (<n2> ^type node ^name target)}
```



Generate Rule

```
sp {gen-target
   (state <s> ^superstate nil
              ^svs.command <c>)
-->
   (<c> ^generate <g>)
   (<g> ^parent world
        ^node <n>)
   (<n> ^type gen
        ^name target
        ^points.type singleton
        ^pos <pos>)
   (<pos> ^type vec3 ^x 5.0 ^y -5.0 ^z 0.0)}
```



Control Rule

```
sp {seek-target
   (state <s> ^superstate nil
              ^svs ( ^command <c>
                     ^spatial-scene ( ^child.id splinter
                                      ^child.id target)))
-->
   (<c> ^control <ctl>)
   (<ctl> ^type simplex ^depth 20 ^outputs <out>
          ^objective <obj> ^model model1)
   (<out> ^left <left> ^right <right>)
   (<left>  ^min -1.0 ^max 1.0 ^inc 1.0)
   (<right> ^min -1.0 ^max 1.0 ^inc 1.0)
   (<obj> ^name euclidean ^a splinter ^b target)}
```