A Computational Theory of the Communication of Problem Solving Knowledge between Parents and Children

Bo Morgan

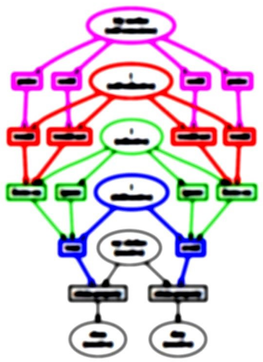
bo@mit.edu

PhD Student
MIT Media Lab
Cambridge, MA

PhD Proposal Defense

MIT Media Lab

Cambridge, MA
May 19th 2010



No Computational Models Currently Exist

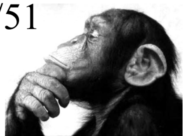
- We are modeling an area that has very few other computational models that exist.
- We are not saying this is the way the world works.
- We are describing and opening up the possibility of building a space of models that could be tested in order to discover how the world works.
- Our goal is to make working models and testable theories of self-reflective and self-conscious processes and knowledge.
- We consider a working model to be a computer implementation and simulation of the processes in question.

The Story of Muddy Carol

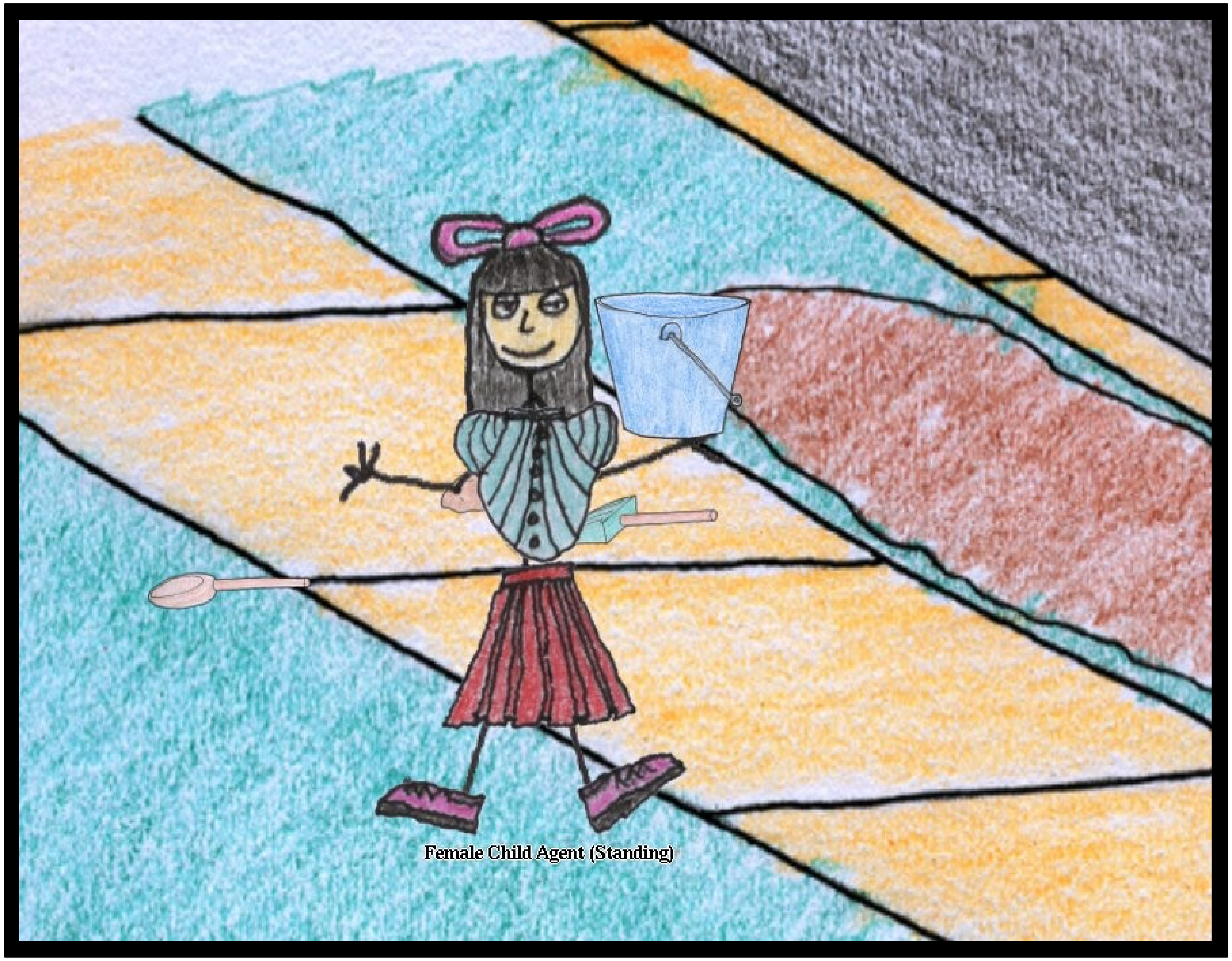
(1) A little girl named Carol is playing alone in the mud. She wants to fill her cup with mud, and first tries to do this with her fork, but this fails because the mud slips through. She succeeds by using her spoon.

(2) A stranger scolds Carol for playing in the mud. “That is a naughty thing to do.” Carol feels anxious, alarmed, and afraid. Overcome by fear and the urge to escape, she interrupts her present goal and runs to her parent's protection.

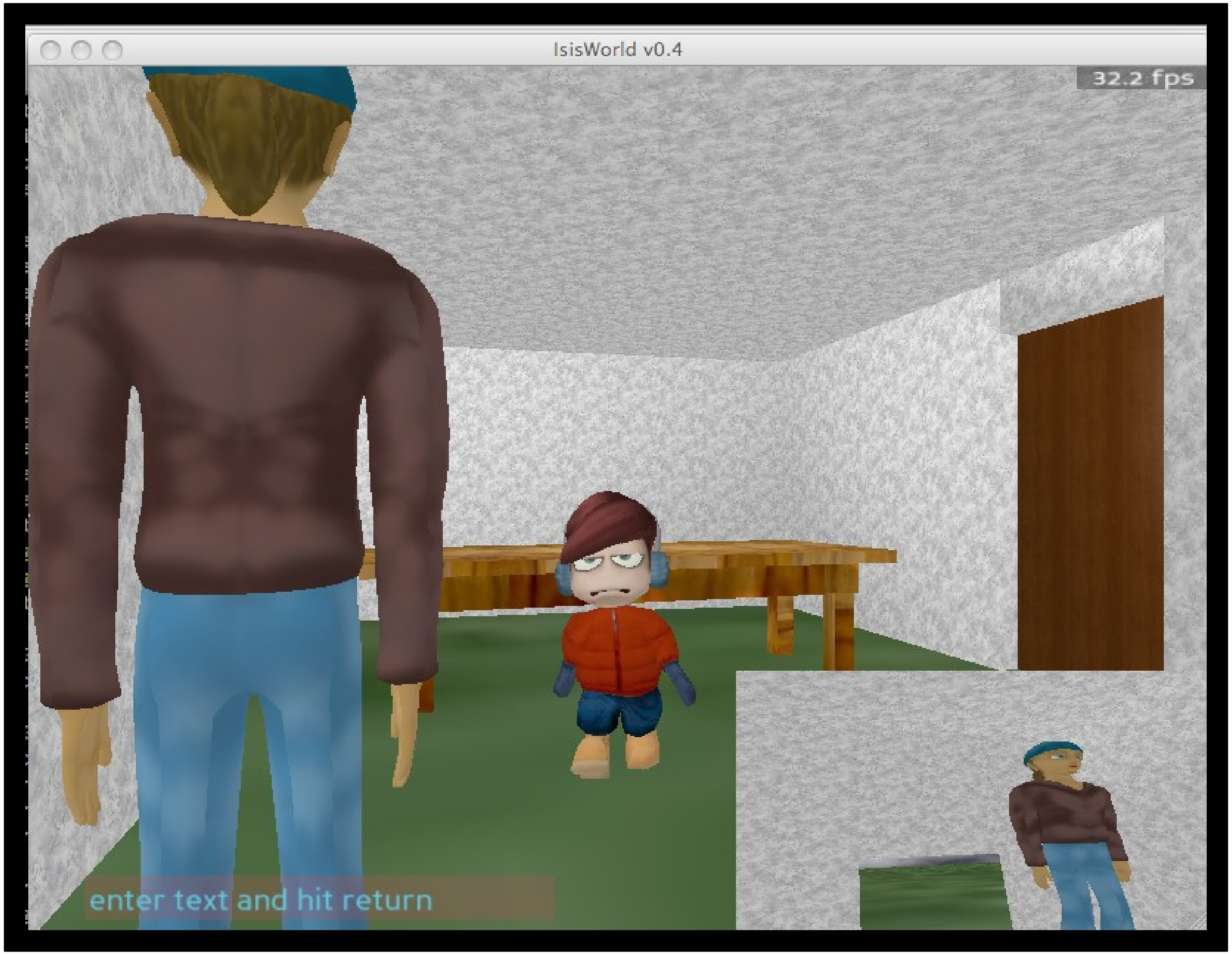
(3) Carol returns to her mother for help, but instead of defense or encouragement, all she gets is a reproof, her mother scolds, “What a disgusting mess you've made! See all the mud on your clothes and your face." Carol, ashamed, begins to cry.



Little Carol Simulation



IsisWorld Simulation



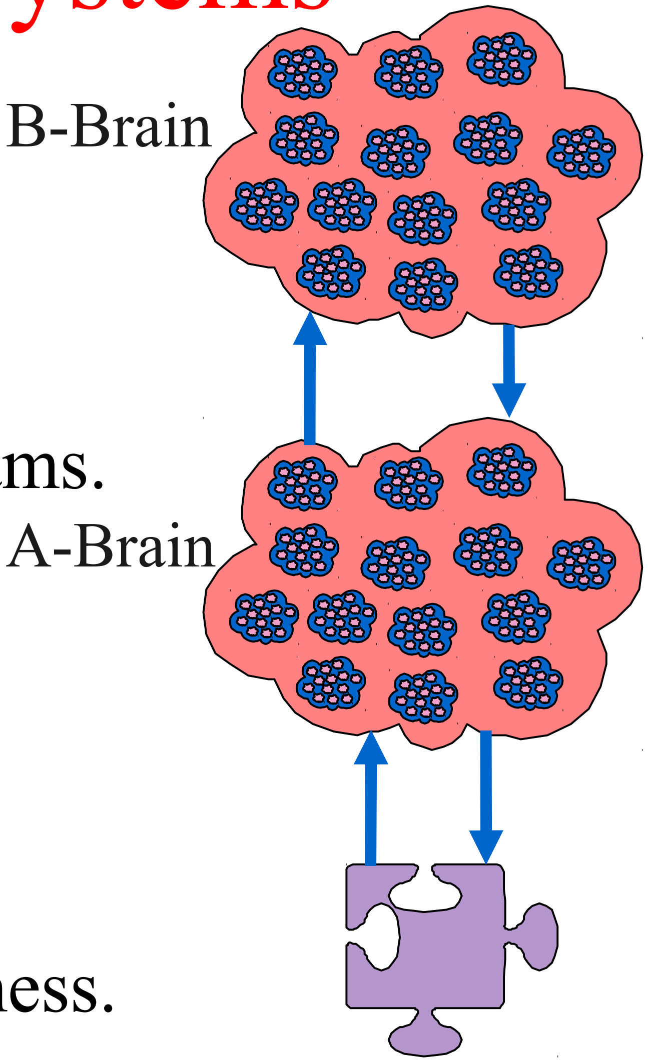
Smith, D. and Morgan, B.; " IsisWorld: An open source commonsense simulator for AI researchers"; AAAI 2010 Workshop on Metacognition; April 2010

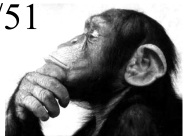
Layered Control Systems

Causal Reflective Programming allows programs to monitor one another allowing intricate failure tolerant "responsible" programs.

A new field of algorithmic machine learning and pattern recognition!

Novel models of Human consciousness.



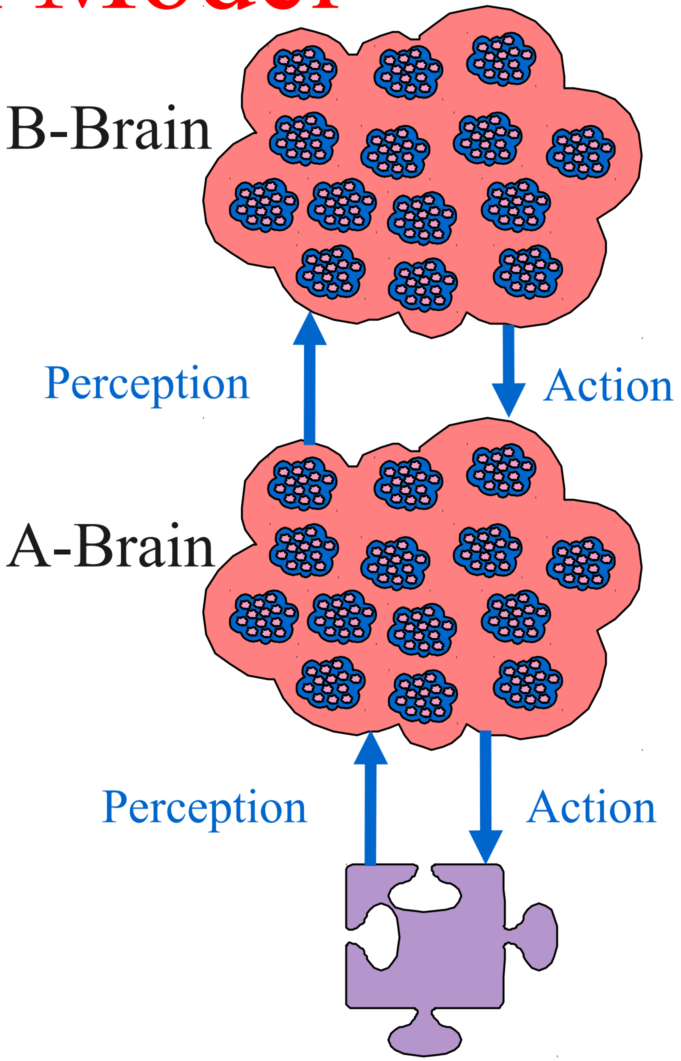


Reflective Control Model

The reflective model emphasizes that sometimes humans solve problems in their minds in addition to the physical world.

A few different types of critical reflection:

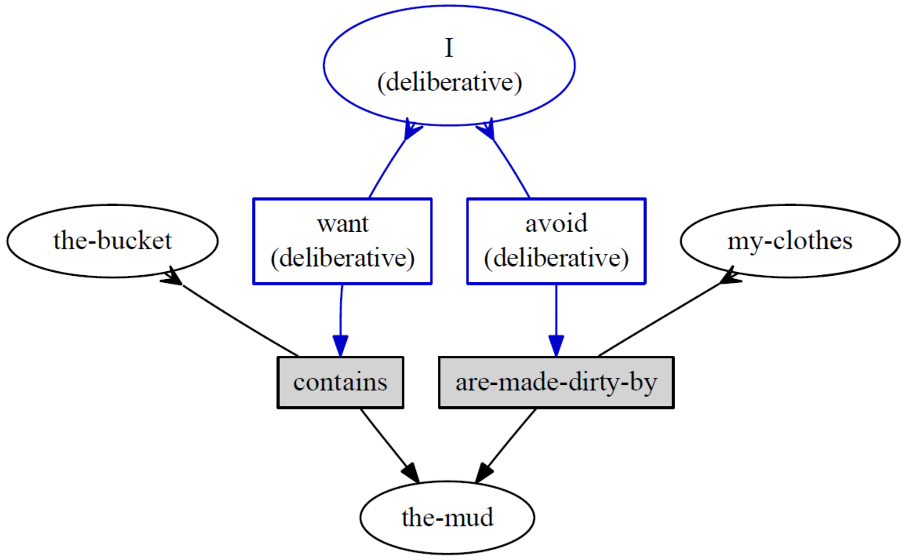
- faults in plans
- faults in planning processes
- faults in knowledge
- conflicts in goals
- credit assignment for failure/success



Deliberative Knowledge

English Interpretation:

"I avoid
getting my clothes dirty."

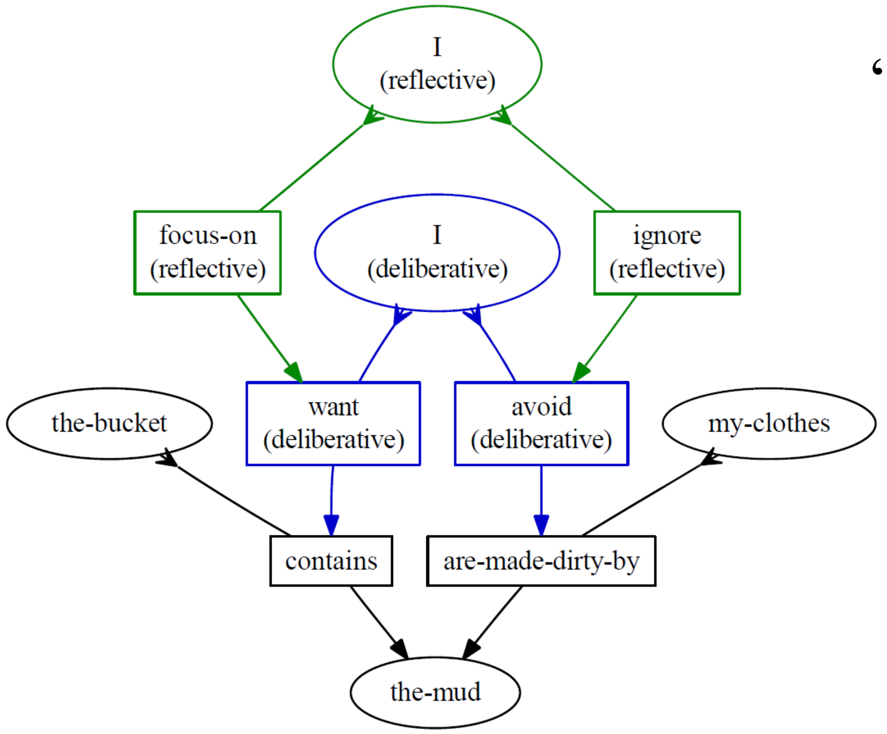


Basic goals in our model are represented by what we call "deliberative knowledge" or "the deliberative layer."

Deliberate Commonsense Reasoning

- Similar inspiring stories are remembered.
- Inspiring stories are critically debugged and analogically adapted to the present situation.
- Bug free parts of plans are executed.
- Need powerful graph matching library to operate over complex procedural physical action memories.

Reflective Knowledge



English Interpretation:

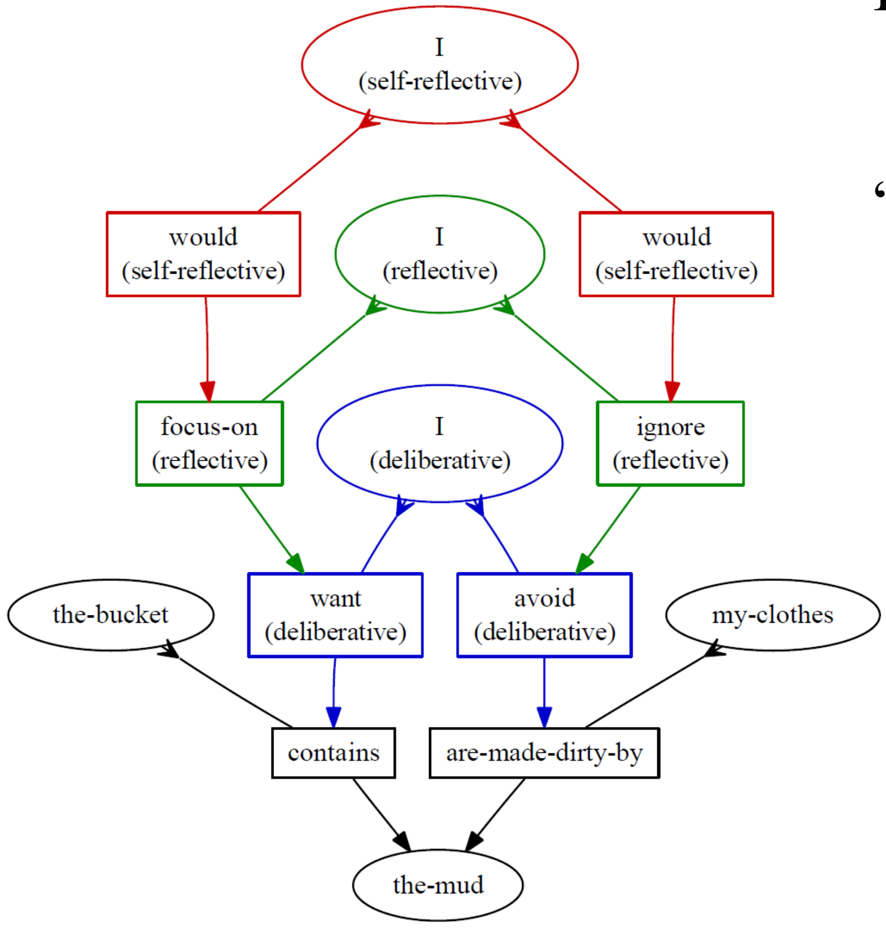
"I ignore
avoiding
getting my clothes dirty."

Selecting, constraining, and otherwise reasoning about groups of deliberative goals is handled by what we call "reflective knowledge" or "the reflective layer."

Reflective Control of Deliberation

- Similar inspiring stories are remembered about previously successful deliberations.
- Inspiring stories are critically debugged and analogically adapted to the present situation.
- Bug free parts of plans are executed.
  - E.g. constellations of non-conflicting deliberation rules.
  - E.g. analogies between visual and spatial domains were previously helpful in this problem domain.
  - E.g. we appear to be making very slow progress toward any story that resembles a complete plan, so let's try thinking in larger jumps and shut down our slower and more careful incremental thinking.

Self-Reflective Knowledge



English Interpretation:

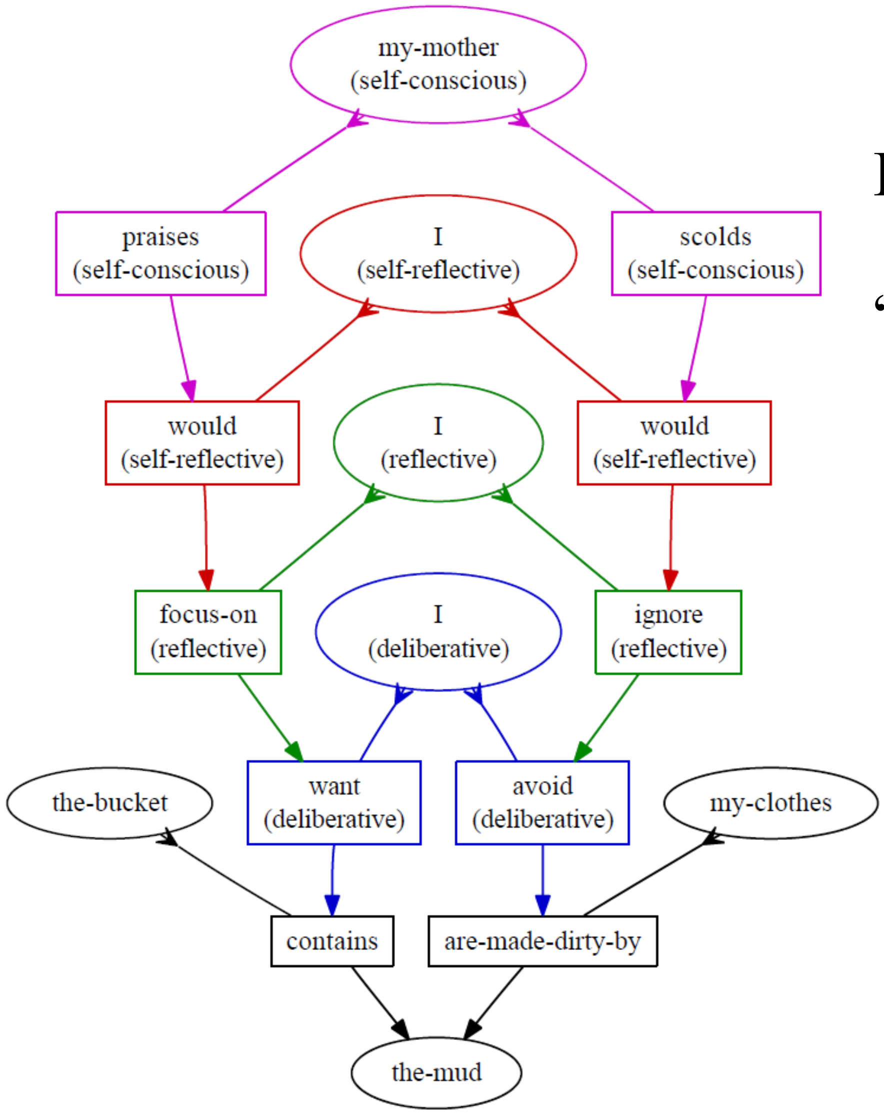
"I would
ignore
avoiding
getting my clothes dirty."

Self-models of what one would or would not do (i.e. personality traits) are represented as what we call "self-reflective knowledge" or "the self-reflective layer."

Self-Reflective Layer

- Self and Other difference models
  - Good or bad at accomplishing which types of goals?
  - Good or bad at using which types of resources?
  - E.g. Ralph never takes out the recycling unless he is explicitly asked, while Carol takes out the recycling and the trash without being asked.
- Constellations of non-conflicting Self-control resources with conflict suppressing constraints
  - E.g. when trying to imagine what my friend must be thinking about herself, I cannot also plan my relationship with my boss.
- Common bugs are learned as critics that debug conflicting configurations of reflective mind for specific problem types.
  - E.g. we have previously found it useful to think in larger jumps in this domain and incremental and detailed planning has never before resulted in a complete plan for these types of problems.

Self-Conscious Knowledge



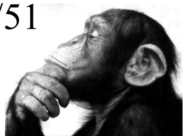
English Interpretation:

"My mother scolds if
I would
ignore
avoiding
getting my clothes dirty."

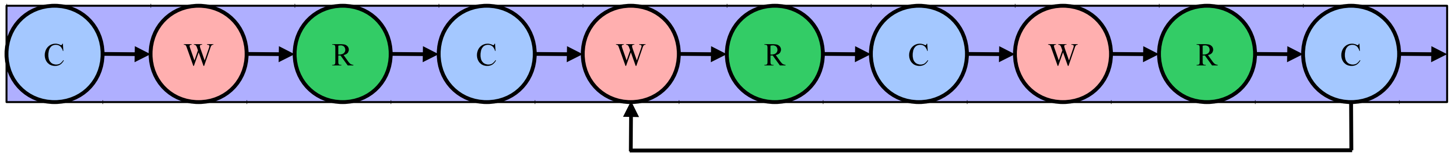
Imprimer opinions, inherited personality trait constraints, and other top-level goals are referred to as "self-conscious knowledge," or "the self-conscious layer."

Self-Conscious Layer

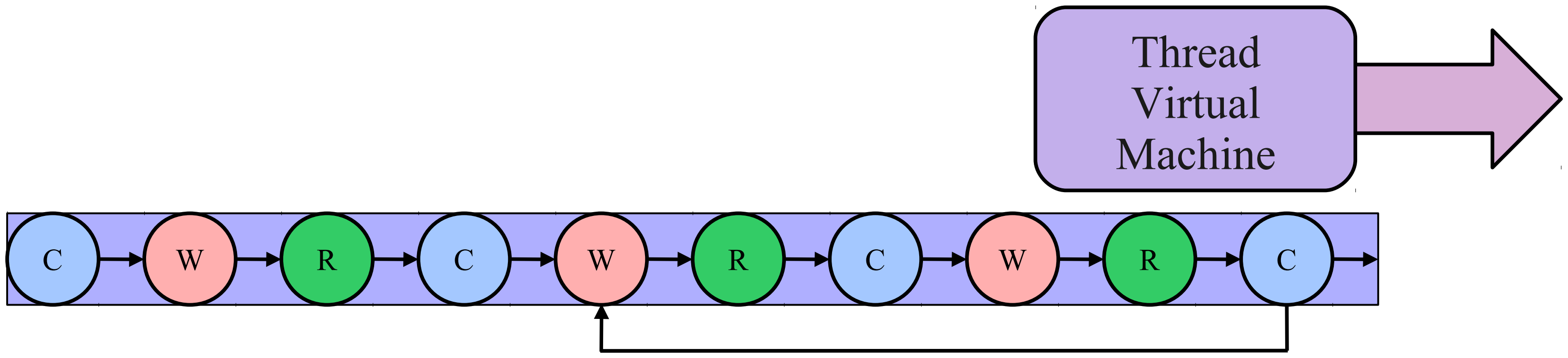
- Inspiring stories about ourselves and relationships with others are adapted to the current context.
  - E.g. my friends hugged me and carried me around (praised me) as a hero when I won the game for my team.
  - E.g. my mother was disappointed in me when she found out that I had lied to my friends.
- Constraints suppress stories that result in a loss of respect, trust, friendship, love, or other beneficial relationships between self- and other-models.
  - E.g. my athletic self might lose respect for my friend-making self if I start smoking cigarettes in order to be cool.
  - E.g. I may lose a friendship if I act against my friend's goals.
  - E.g. I may lose the love of my parents if I reject their culture.
  - E.g. I may lose the respect of my friends if I reject our gang culture.



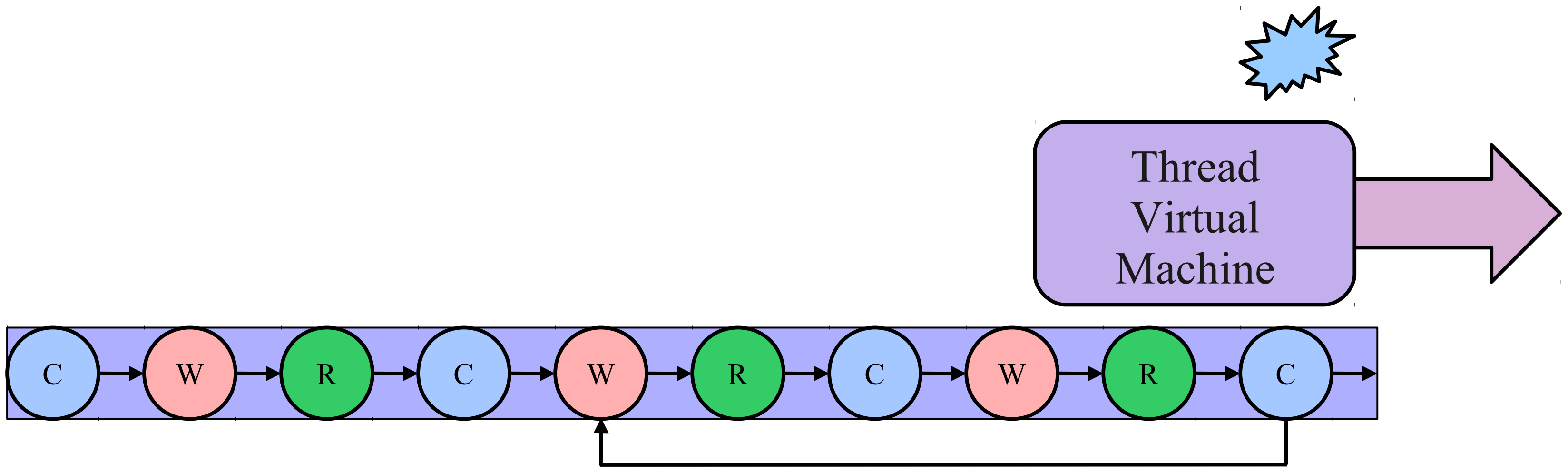
Recognition of Critical Patterns



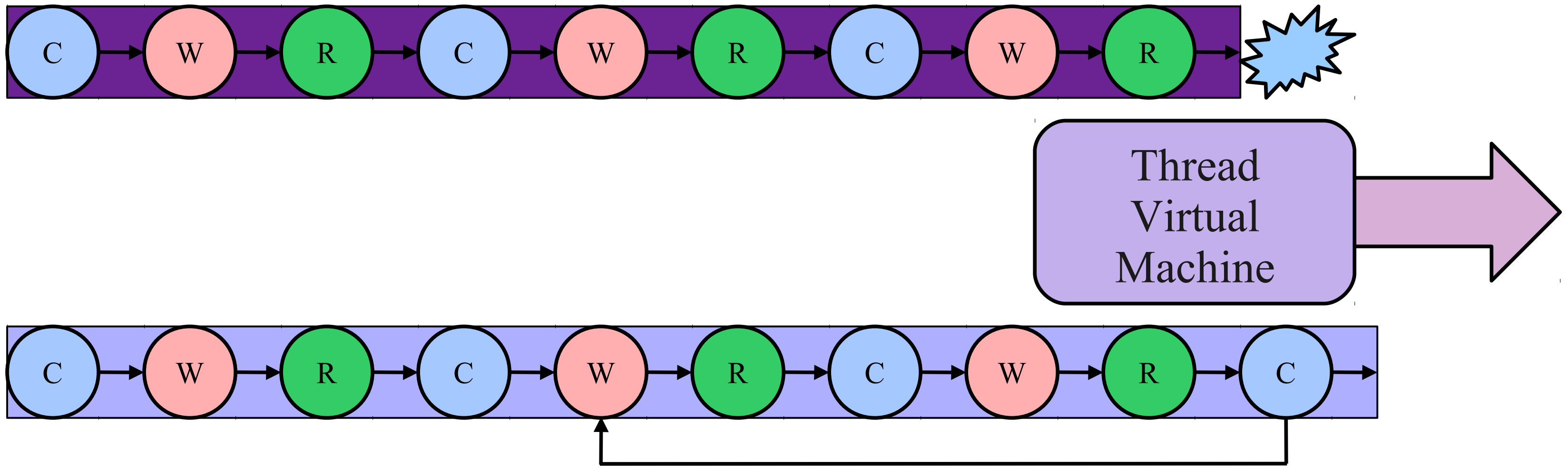
Recognition of Critical Patterns



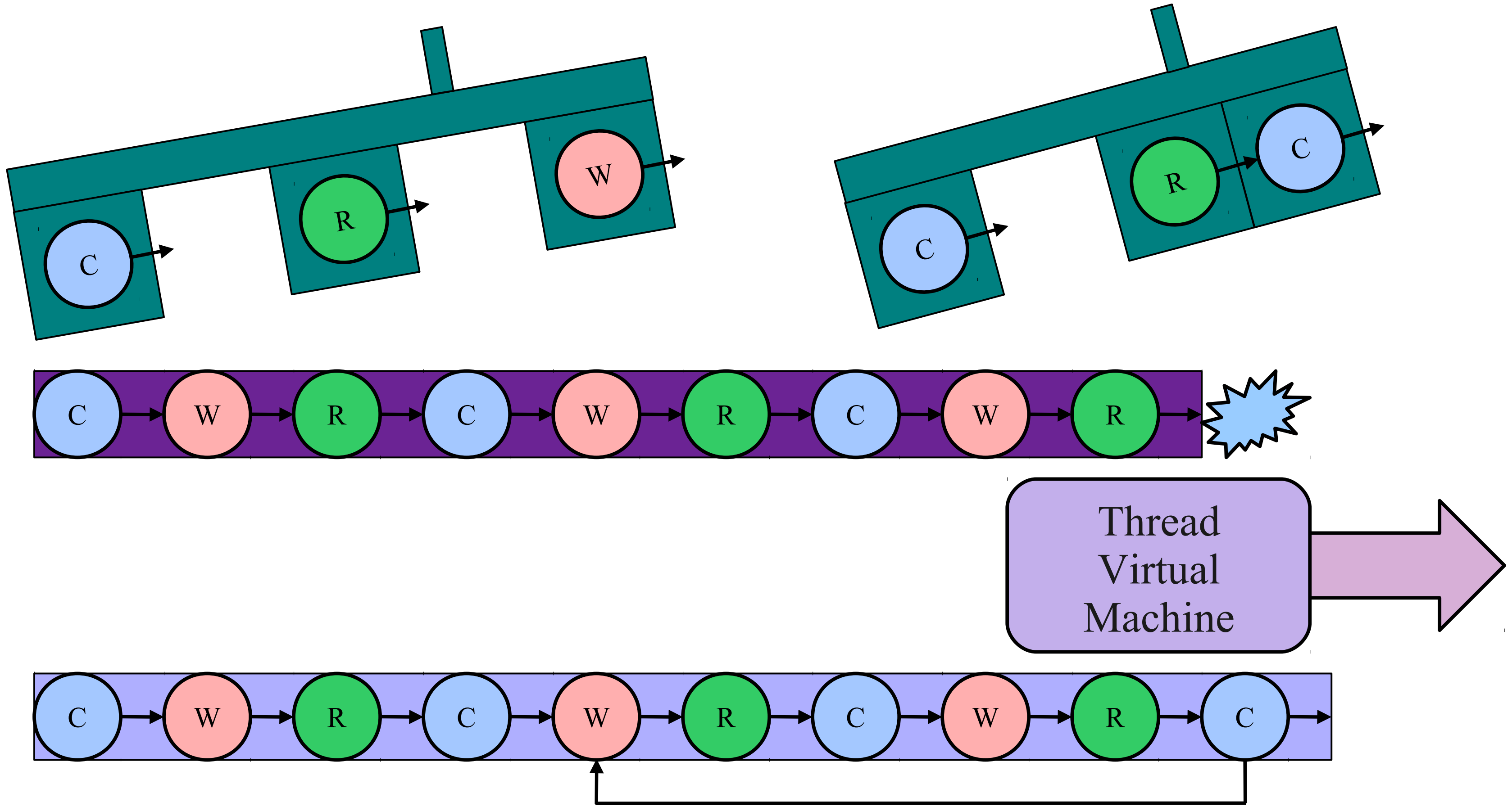
Recognition of Critical Patterns



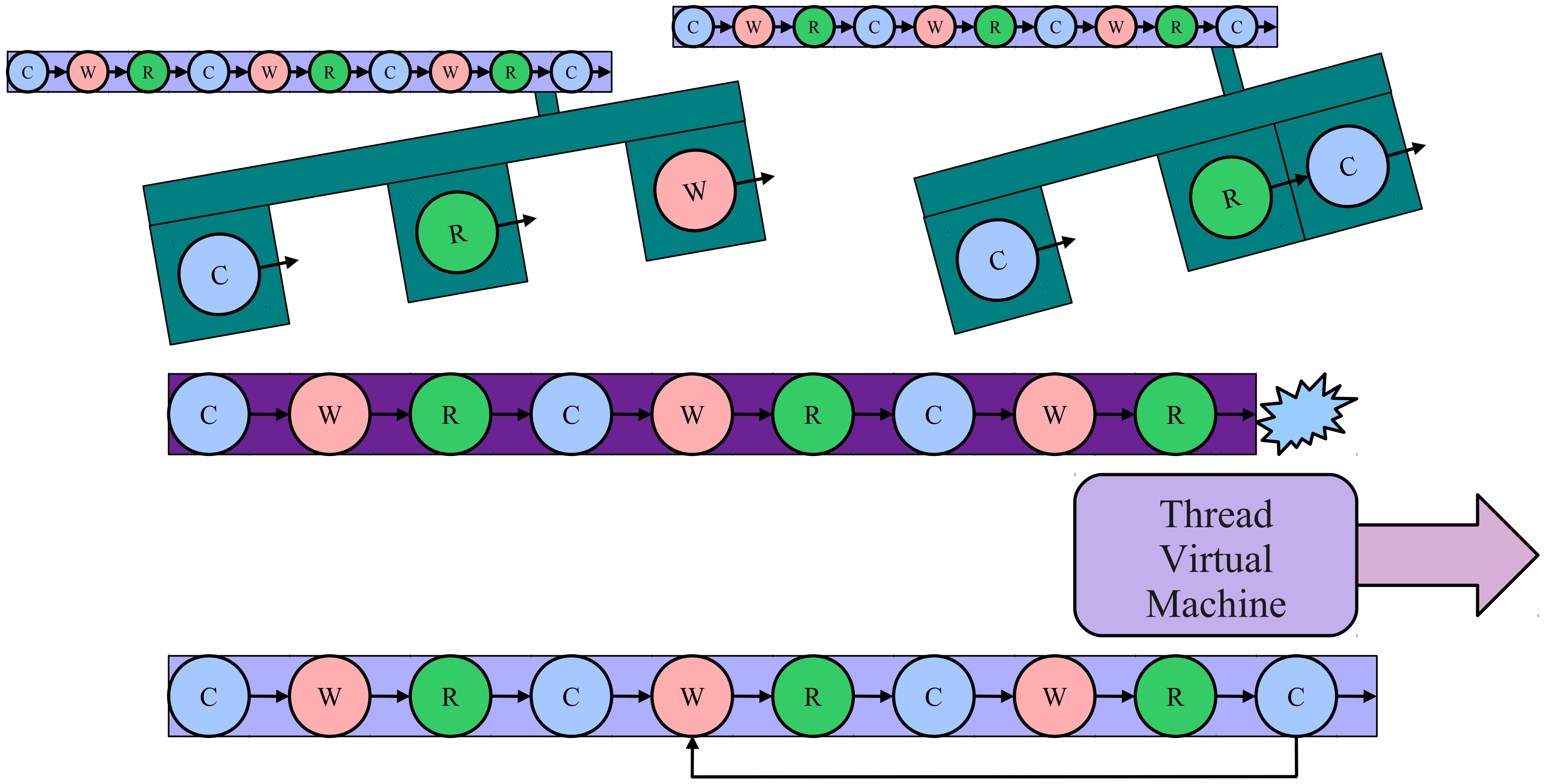
Recognition of Critical Patterns

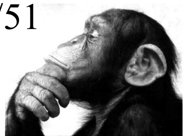


Recognition of Critical Patterns



Recognition of Critical Patterns

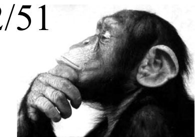




Examples of Reflective Computation

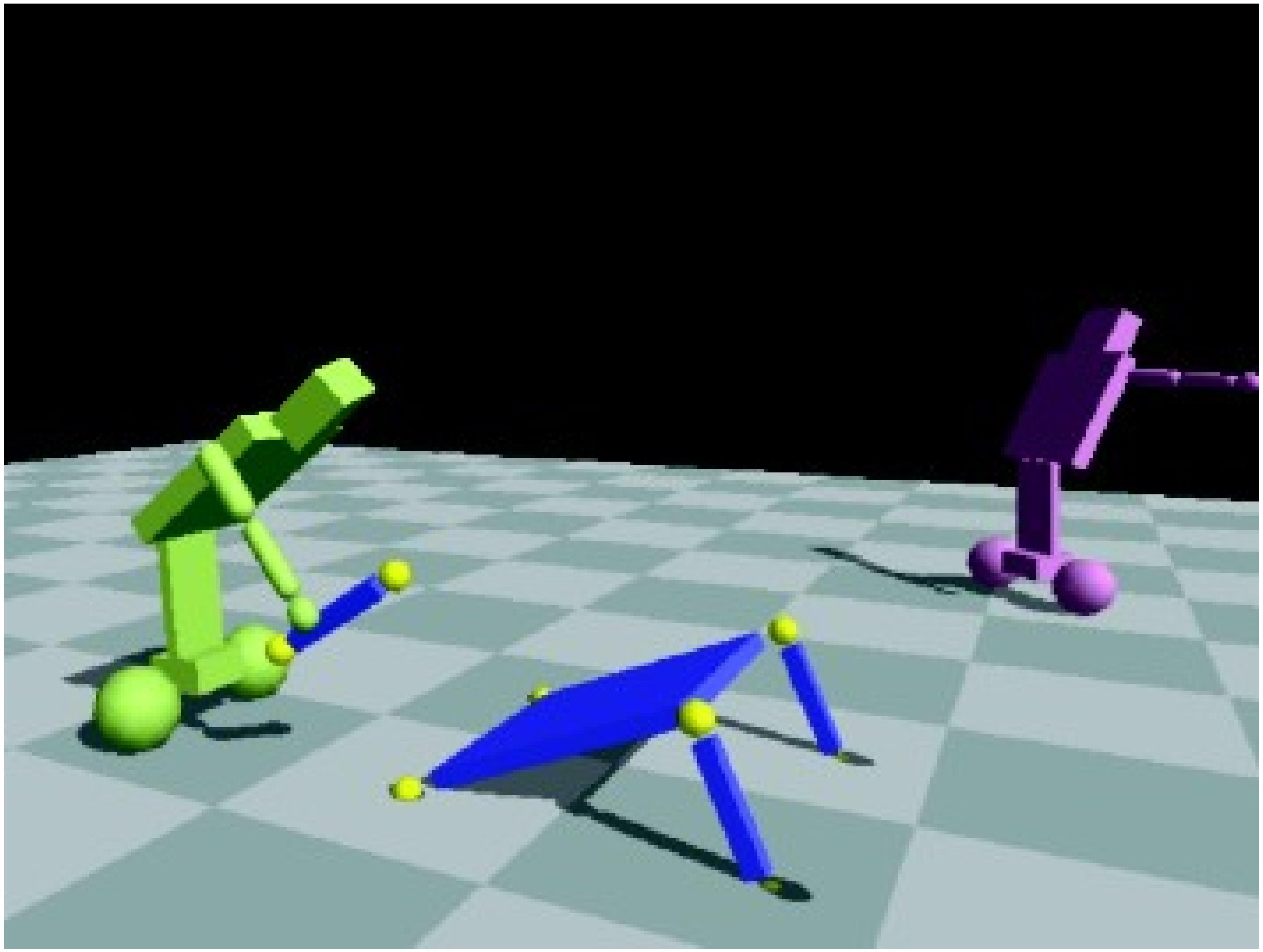
- to keep performance statistics,
- to keep information for debugging purposes,
- stepping and tracing facilities,
- interfacing (e.g. graphical output, mouse input),
- computation about which computation to pursue next (also called reasoning about control),
- self-optimization,
- self-modification (e.g. in a learning system), and
- self-activation (e.g. through monitors and daemons).

Maes, P.; “Concepts and Experiments in Computational Reflection"; OOPSLA '87 Proceedings; 1987 October



EM1 building a table

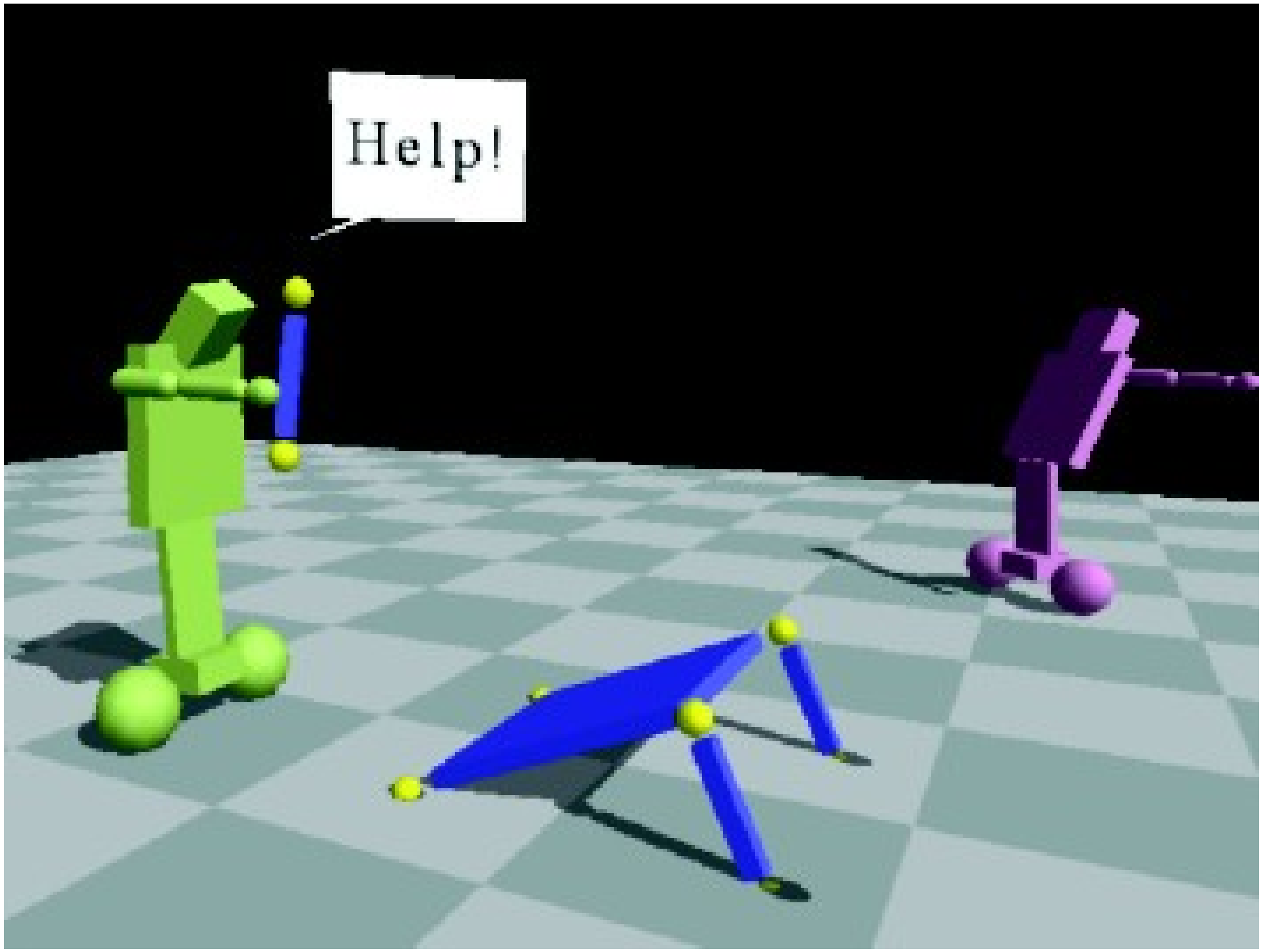
1.



Singh, P. “EM1: An Architecture for Reflective Commonsense Thinking” PhD Thesis, MIT Media Lab. June 2005

EM1 building a table

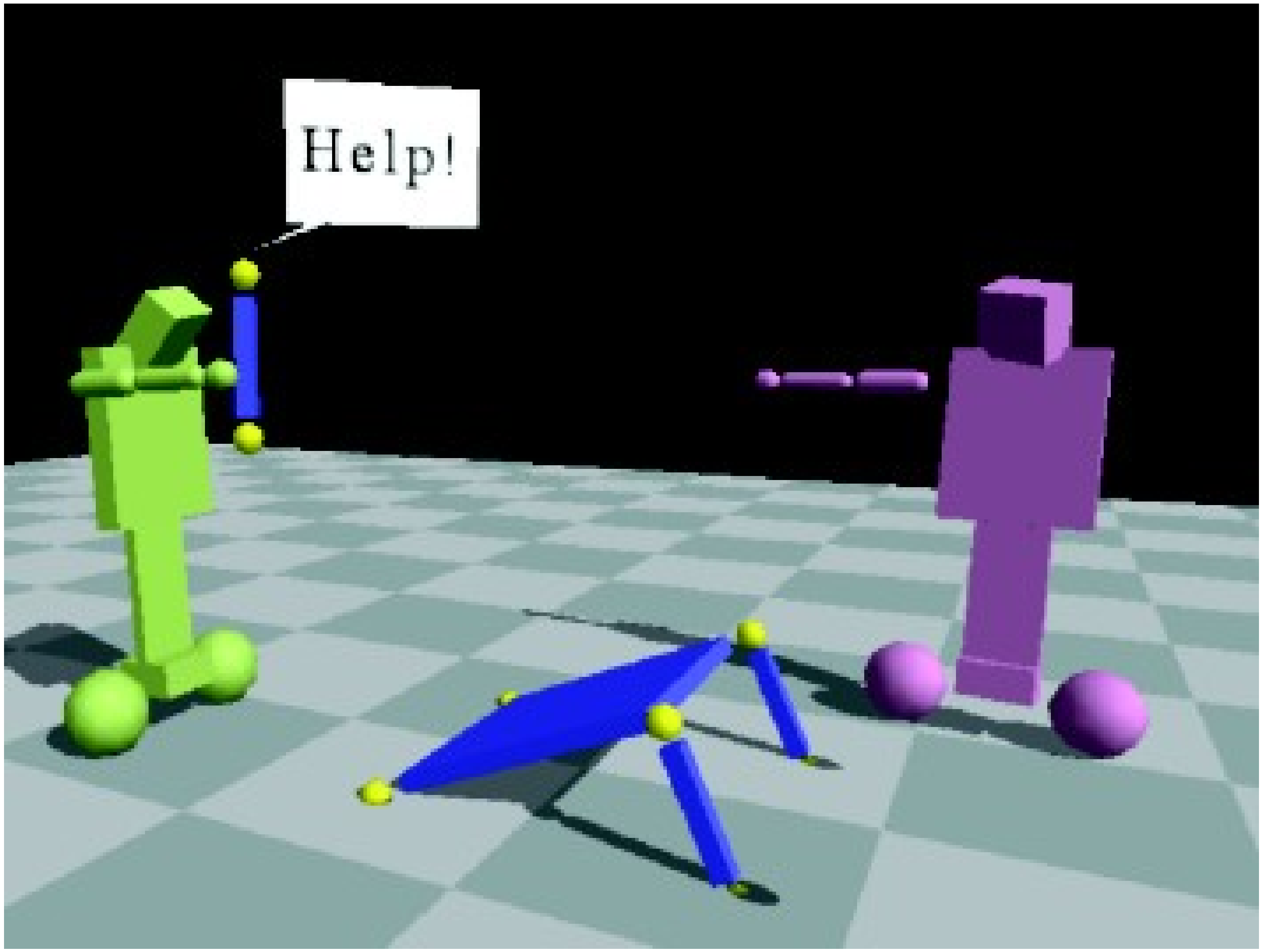
2.



Singh, P. "EM1: An Architecture for Reflective Commonsense Thinking" PhD Thesis, MIT Media Lab. June 2005

EM1 building a table

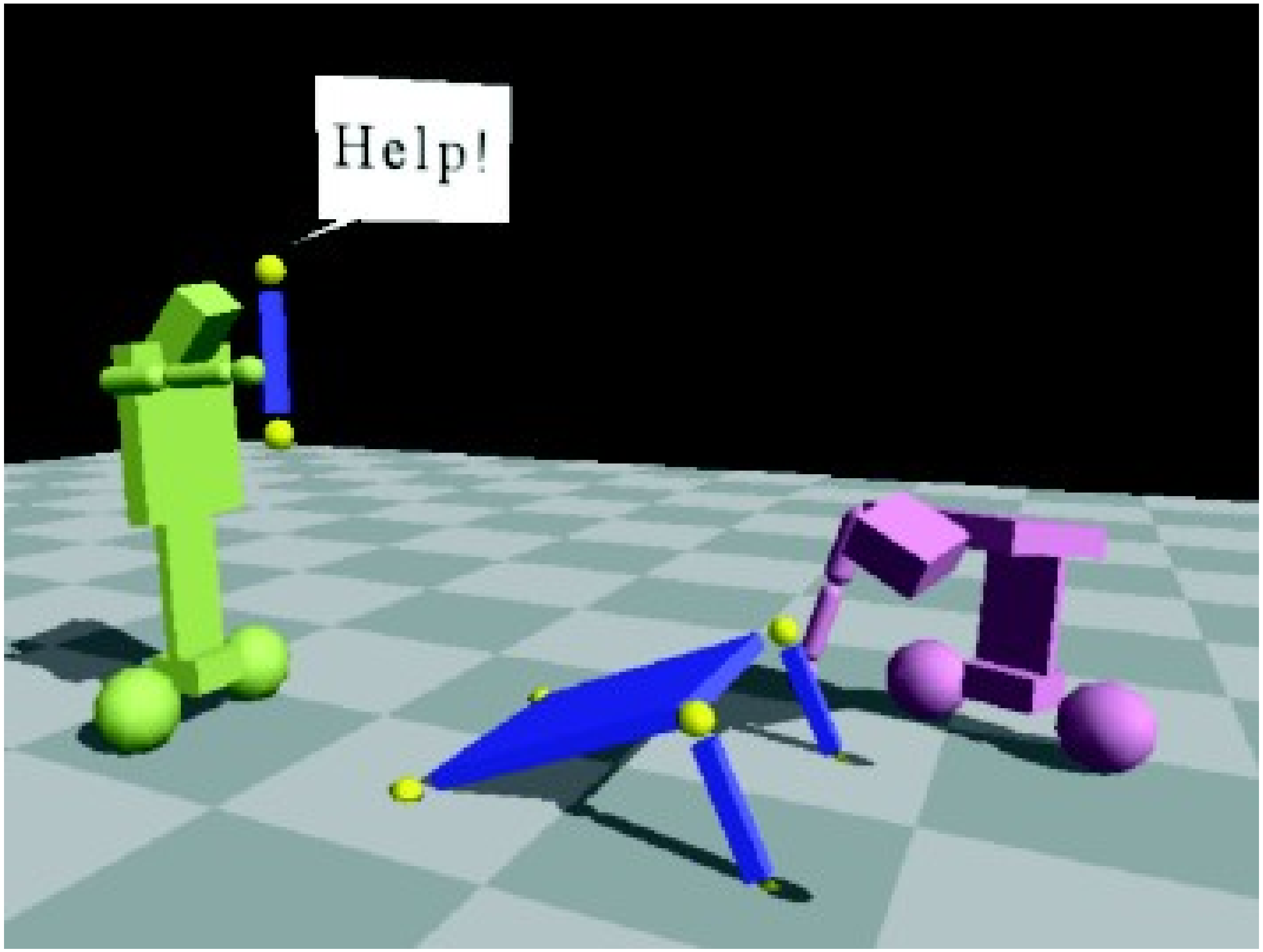
3.



Singh, P. "EM1: An Architecture for Reflective Commonsense Thinking" PhD Thesis, MIT Media Lab. June 2005

EM1 building a table

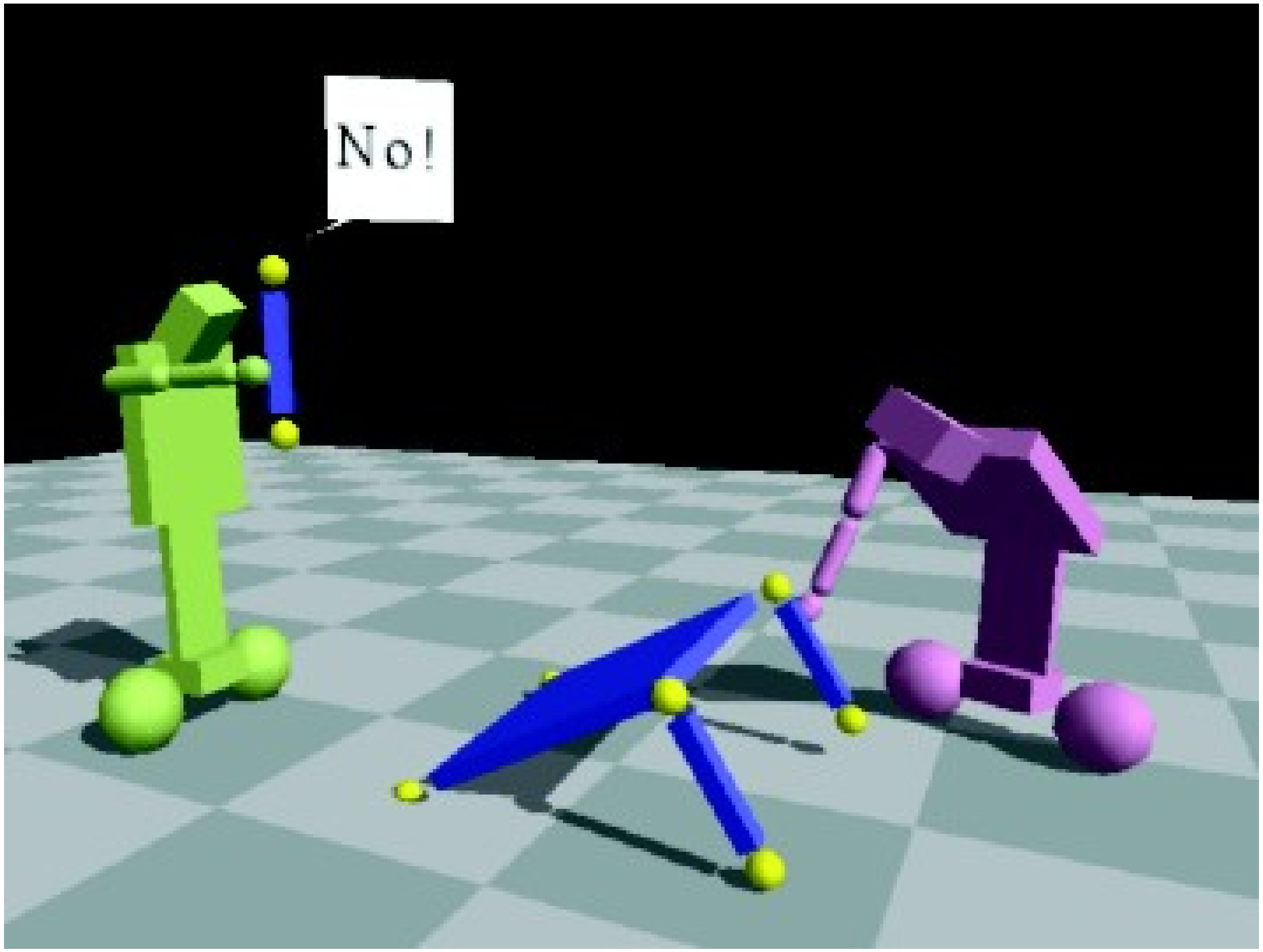
4.



Singh, P. "EM1: An Architecture for Reflective Commonsense Thinking" PhD Thesis, MIT Media Lab. June 2005

EM1 building a table

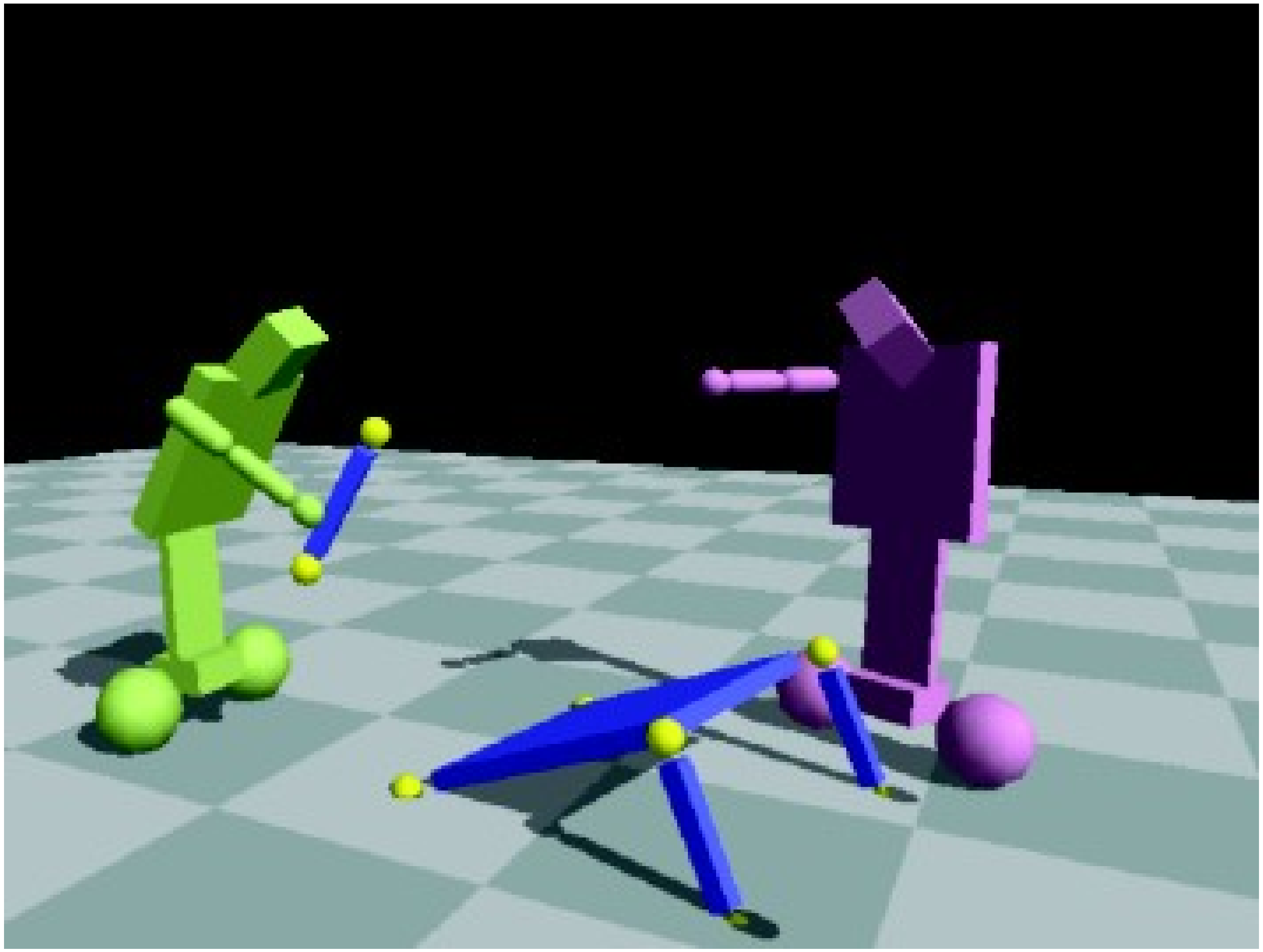
5.



Singh, P. “EM1: An Architecture for Reflective Commonsense Thinking” PhD Thesis, MIT Media Lab. June 2005

EM1 building a table

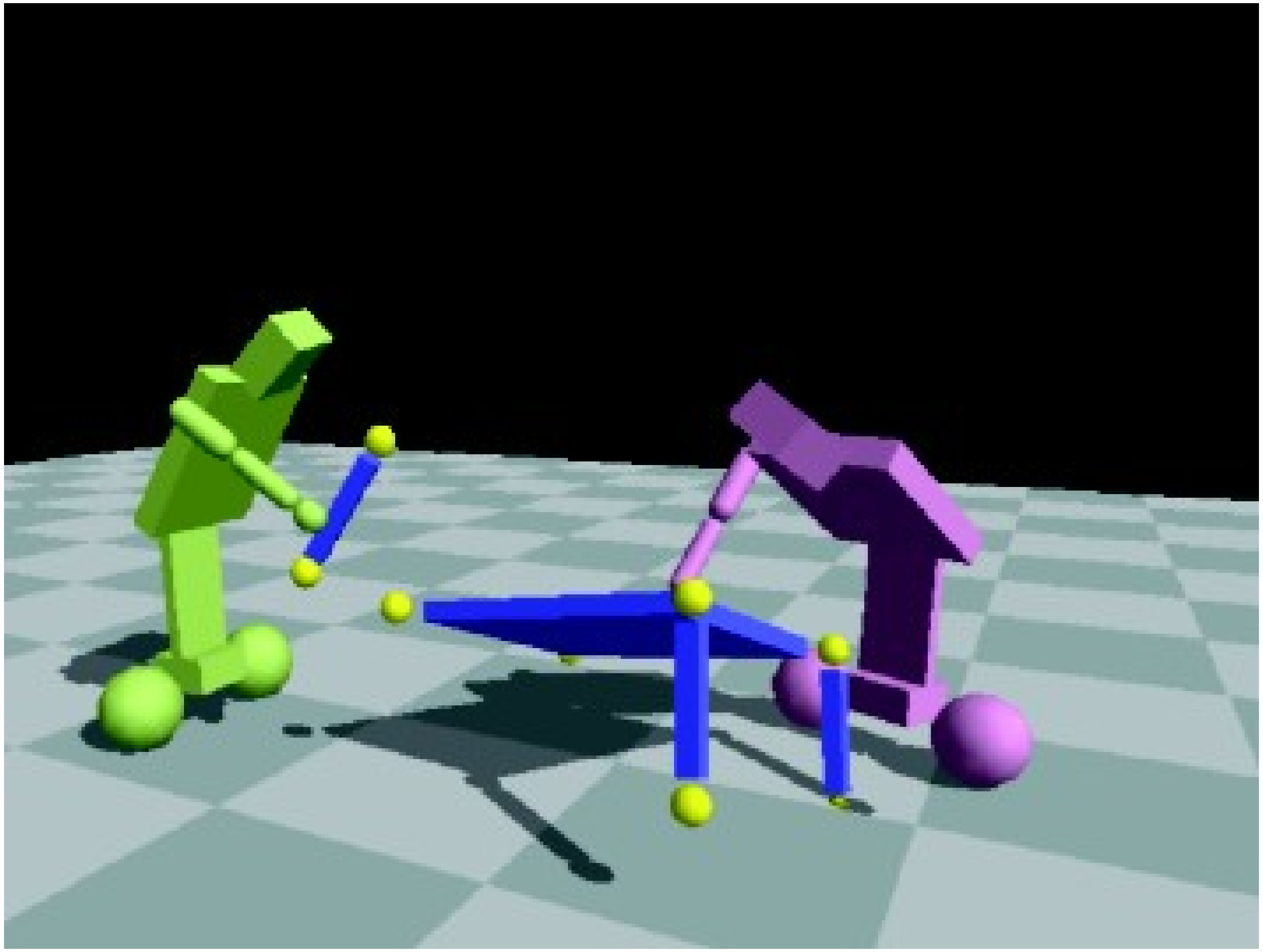
6.



Singh, P. "EM1: An Architecture for Reflective Commonsense Thinking" PhD Thesis, MIT Media Lab. June 2005

EM1 building a table

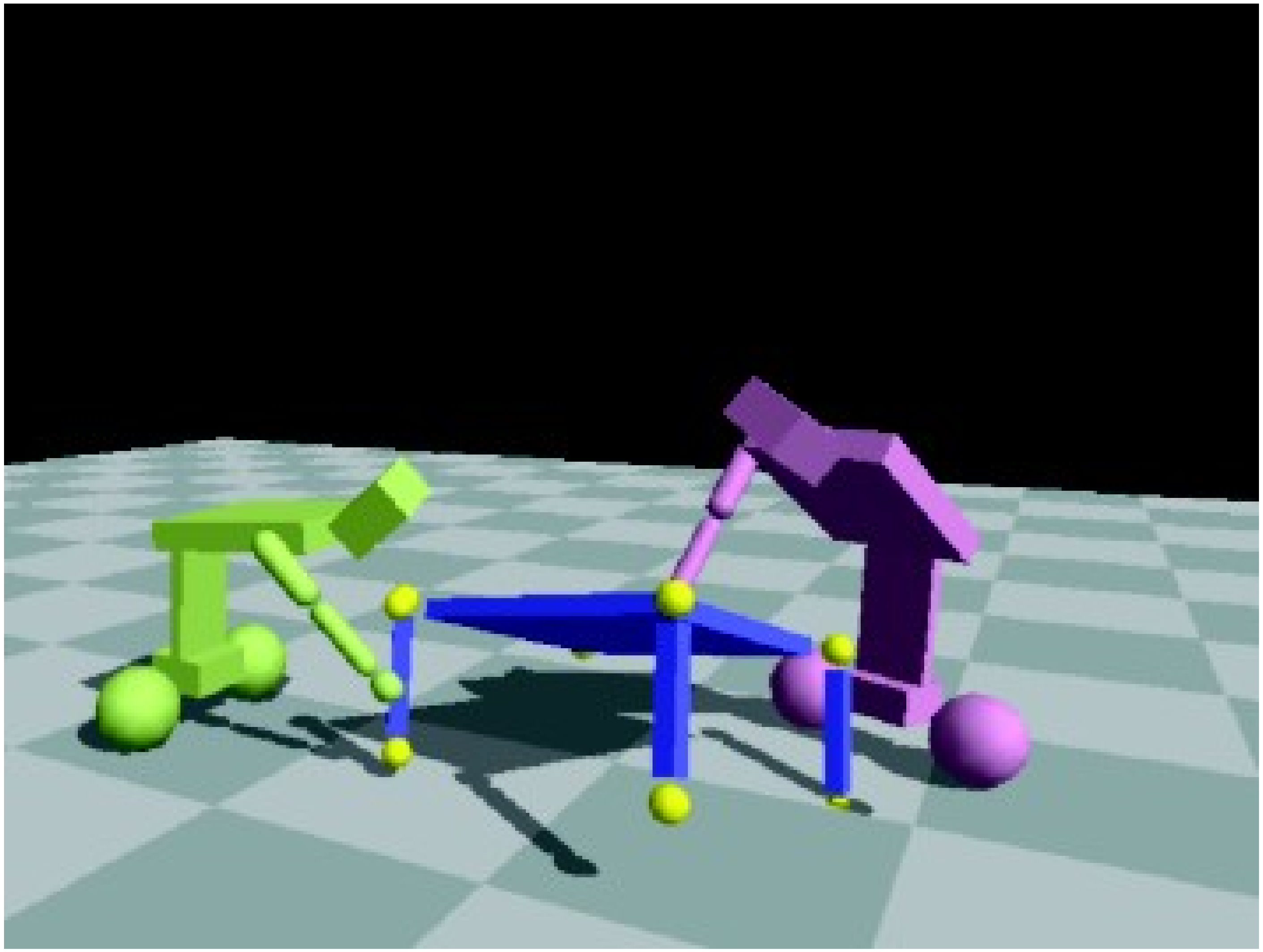
7.



Singh, P. “EM1: An Architecture for Reflective Commonsense Thinking” PhD Thesis, MIT Media Lab. June 2005

EM1 building a table

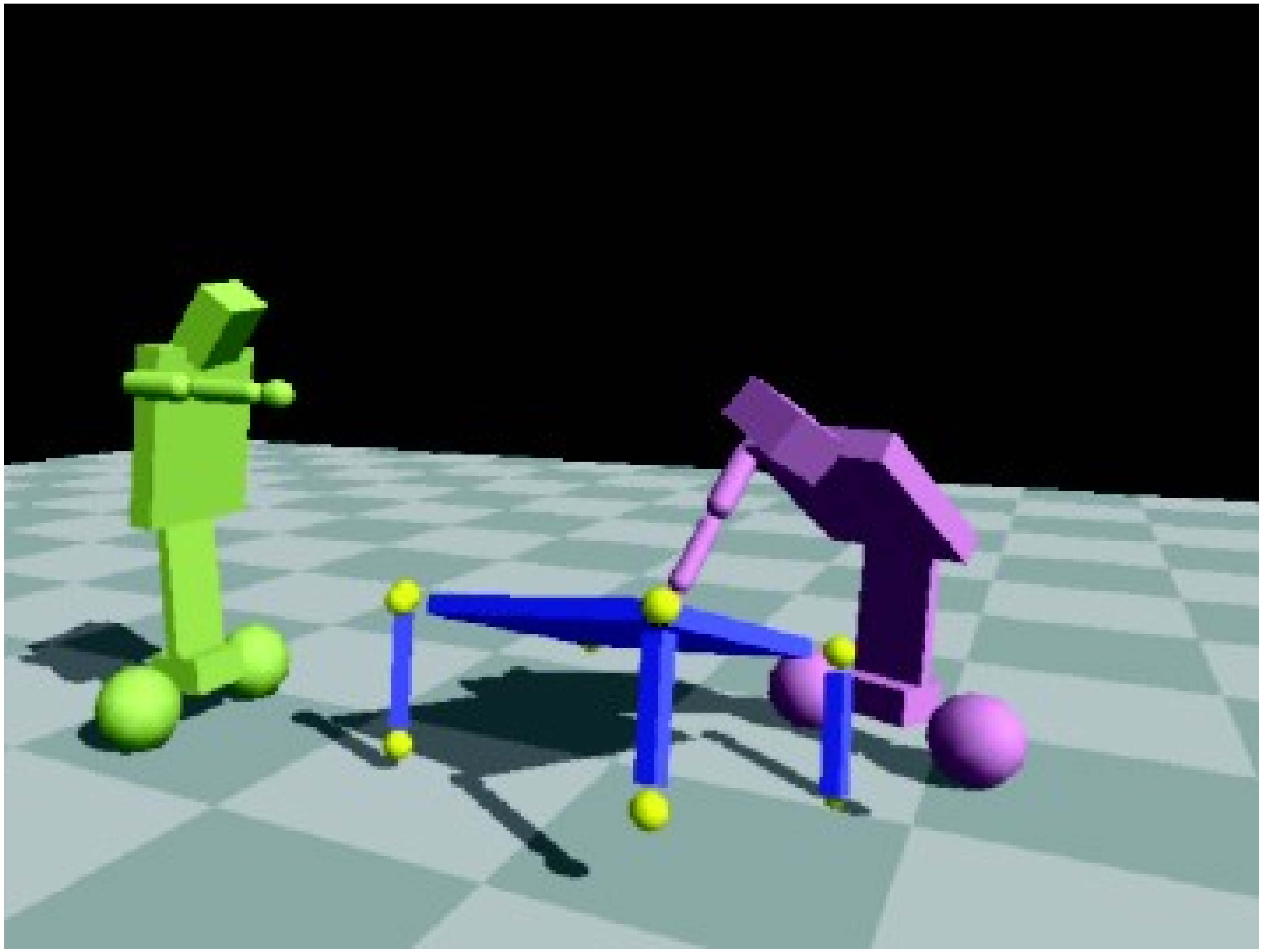
8.



Singh, P. "EM1: An Architecture for Reflective Commonsense Thinking" PhD Thesis, MIT Media Lab. June 2005

EM1 building a table

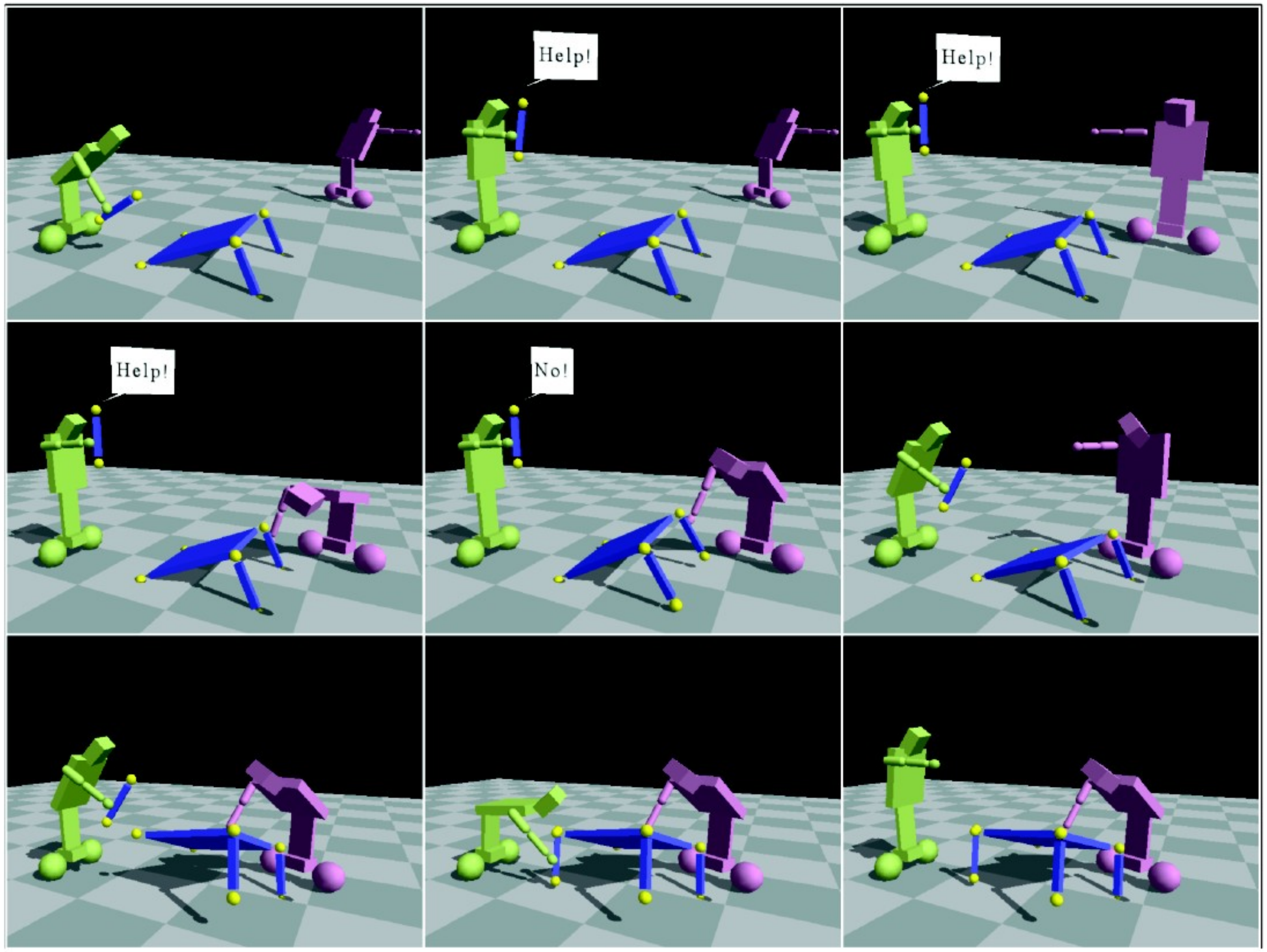
9.



Singh, P. "EM1: An Architecture for Reflective Commonsense Thinking" PhD Thesis, MIT Media Lab. June 2005

EM1 building a table

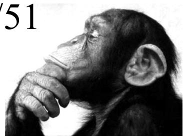
1–9.



Singh, P. "EM1: An Architecture for Reflective Commonsense Thinking" PhD Thesis, MIT Media Lab. June 2005

Problems with EM1 Architecture

- Procedural code is not traced.
- Critics and selectors are specified in logic.
  - Imperfect reasoning is not possible.
  - Logic cannot specify process control.
- Reasoning is very slow with only 21 narratives in the architecture memory.
- Parallel theory with no parallel implementation.
- Expensive (~$5,000) proprietary versions of Lisp and Prolog used.
  - Never released as a public software package.
- Self-reflective, Self-conscious, and Self-ideal critics are future work.
- Critics and selectors are not learned from experience.

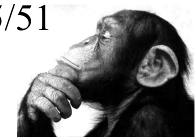


Partial Solution
A new programming language

- Procedural code is traced.
- Critics and selectors are specified as procedural code.
  - Language has control statements.
  - Can be traced or stepped, so process control can also be reflective.
- Written in pure C for speed on large computers (tested allocating 1000 virtual threads allocating a total of 9 gigabytes of RAM on 8 processor cores)
- Distributed peer-to-peer memory layer for grassroots software collaboration.
- Open, free for all platforms, download on-line now: http://funk2.org/
  - Publicly developed project: git://neuromin.de/funk2.git

The Rest of the Solution
A new cognitive architecture

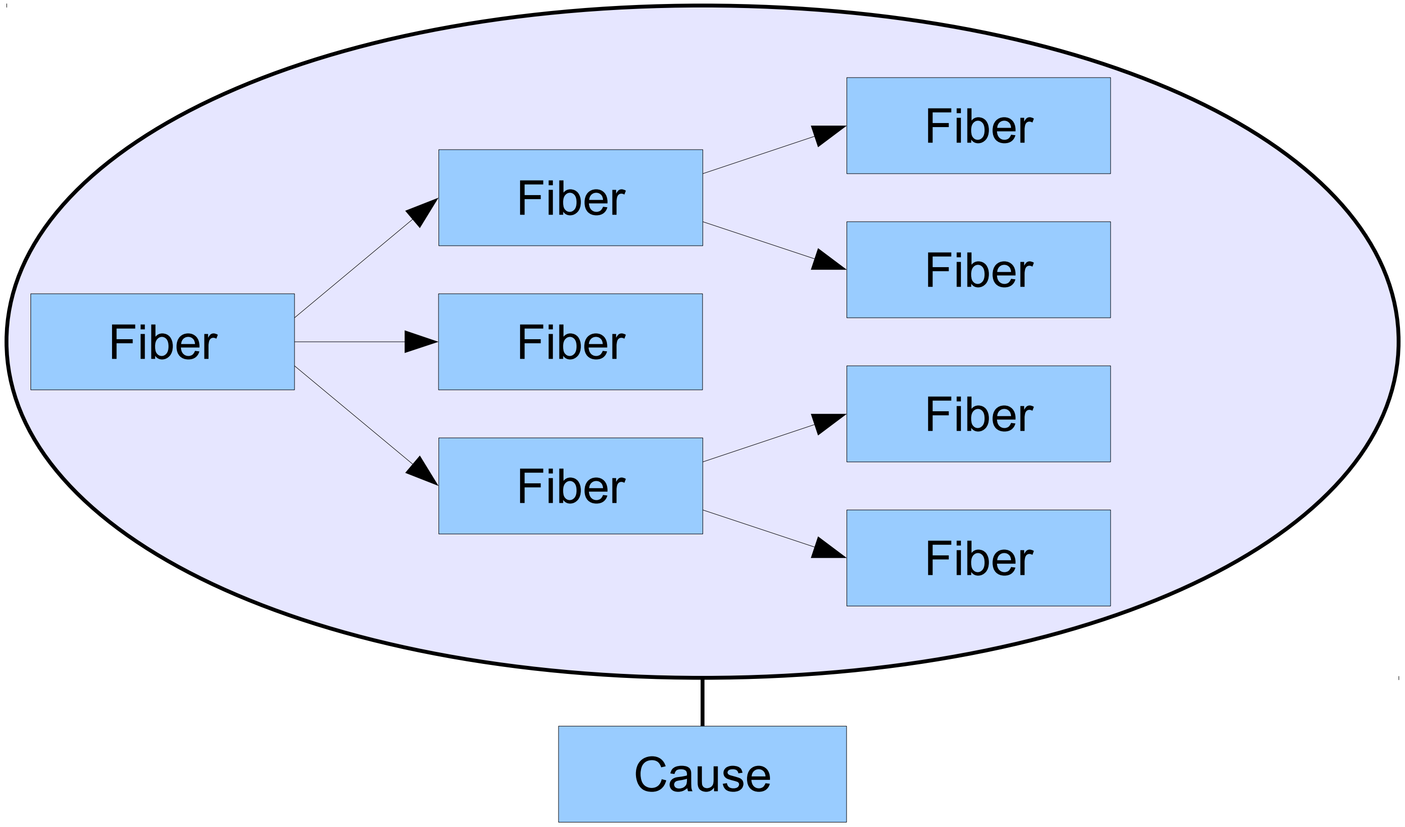
- Self-reflective and Self-conscious critics will exist.
- Critics and selectors will be learned from experience.



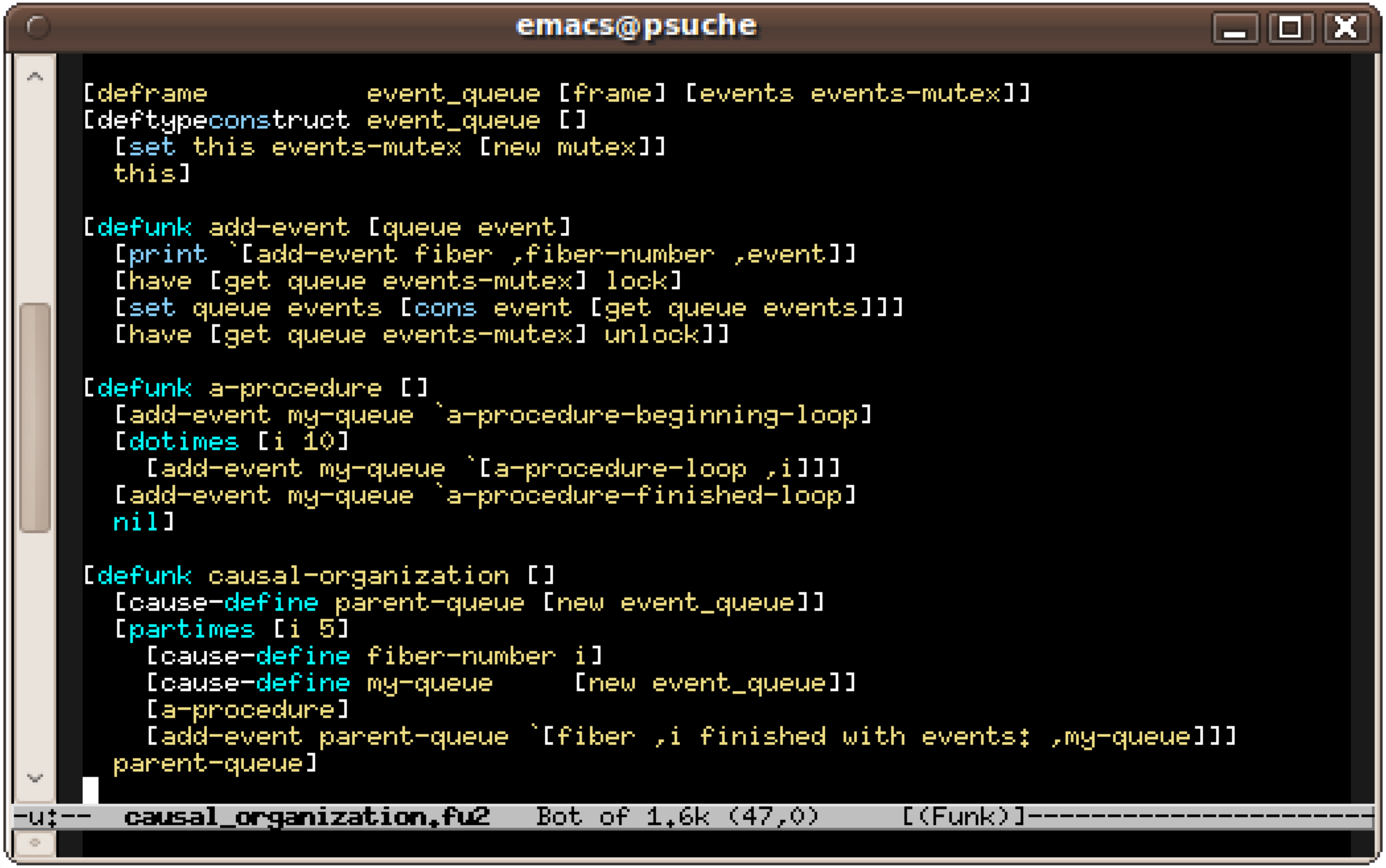
Procedural Code is Causally Traced

- "Cause" objects organize reflective procedural tracing.
- All memory is labeled by creation cause.
- Causes are branched at semantic control points, such as parallel thread creation or reflective activation.
- Cause objects are at the locus of all memory reads and writes.
- Memory can be traced by mutation, allowing causal rewindable memory.
- Bytecode branches and function jumps can be causally traced through assigned callbacks.

Cause Objects Organize Tracing

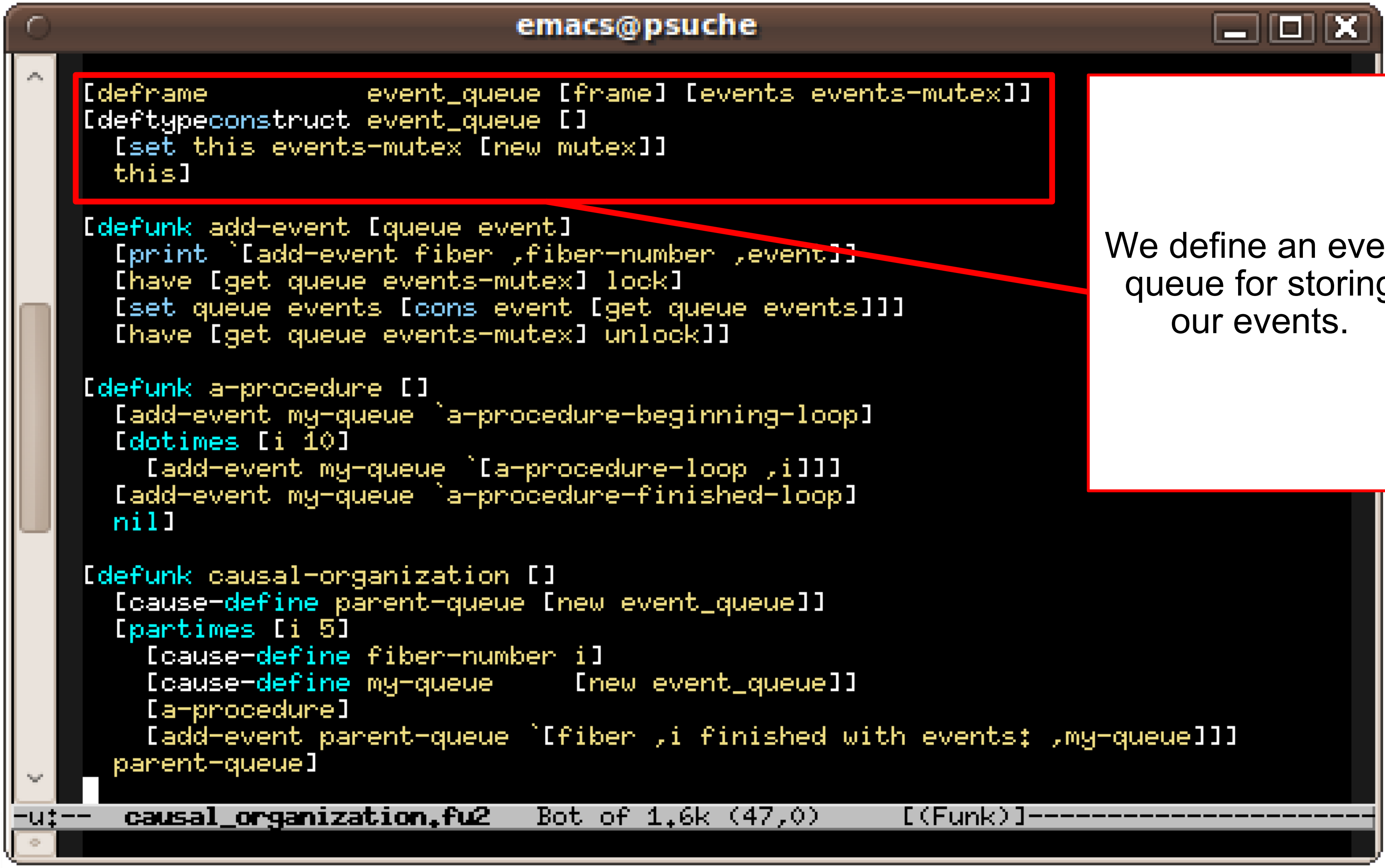


Cause Objects Organize Tracing



```
[deframe             event_queue [frame] [events events-mutex]]
[deftypeconstruct event_queue []
  [set this events-mutex [new mutex]]
  this]

[defunk add-event [queue event]
  [print `[add-event fiber ,fiber-number ,event]]
  [have [get queue events-mutex] lock]
  [set queue events [cons event [get queue events]]]
  [have [get queue events-mutex] unlock]]

[defunk a-procedure []
  [add-event my-queue `a-procedure-beginning-loop]
  [dotimes [i 10]
    [add-event my-queue `[a-procedure-loop ,i]]]
  [add-event my-queue `a-procedure-finished-loop]
  nil]

[defunk causal-organization []
  [cause-define parent-queue [new event_queue]]
  [partimes [i 5]
    [cause-define fiber-number i]
    [cause-define my-queue     [new event_queue]]
    [a-procedure]
    [add-event parent-queue `[fiber ,i finished with events: ,my-queue]]]
  parent-queue]

-u:--  causal_organization.fu2   Bot of 1.6k (47,0)     [(Funk)]--------------------
```

Cause Objects Organize Tracing

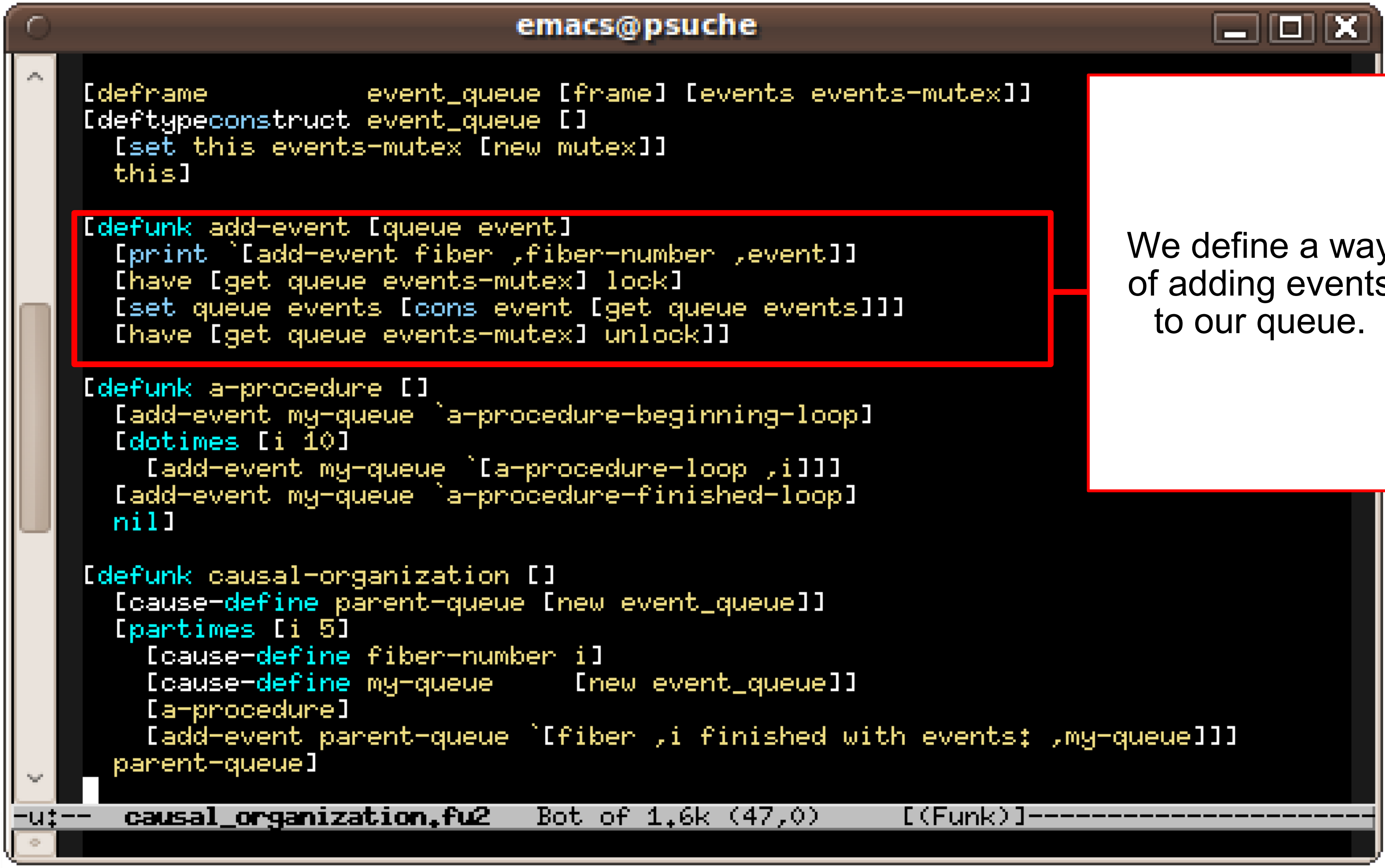


```
[deframe            event_queue [frame] [events events-mutex]]
[deftypeconstruct event_queue []
  [set this events-mutex [new mutex]]
  this]

[defunk add-event [queue event]
  [print `[add-event fiber ,fiber-number ,event]]
  [have [get queue events-mutex] lock]
  [set queue events [cons event [get queue events]]]
  [have [get queue events-mutex] unlock]]

[defunk a-procedure []
  [add-event my-queue `a-procedure-beginning-loop]
  [dotimes [i 10]
    [add-event my-queue `[a-procedure-loop ,i]]]
  [add-event my-queue `a-procedure-finished-loop]
  nil]

[defunk causal-organization []
  [cause-define parent-queue [new event_queue]]
  [partimes [i 5]
    [cause-define fiber-number i]
    [cause-define my-queue     [new event_queue]]
    [a-procedure]
    [add-event parent-queue `[fiber ,i finished with events: ,my-queue]]]
  parent-queue]
```

```
-u:--  causal_organization.fu2   Bot of 1.6k (47,0)      [(Funk)]--------------------
```

We define an event queue for storing our events.

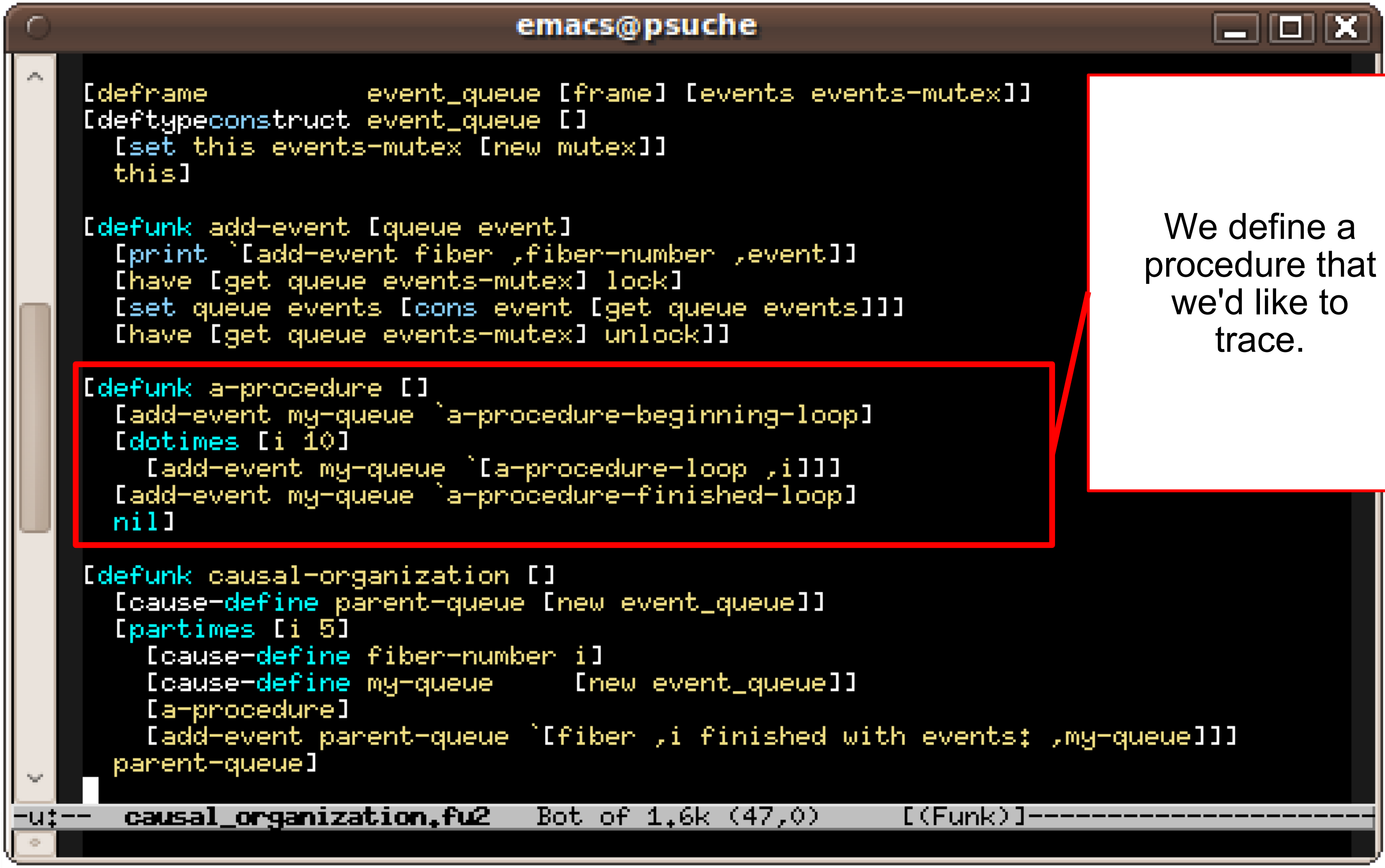
Cause Objects Organize Tracing



```
[deframe           event_queue [frame] [events events-mutex]]
[deftypeconstruct event_queue []
  [set this events-mutex [new mutex]]
  this]

[defunk add-event [queue event]
  [print `[add-event fiber ,fiber-number ,event]]
  [have [get queue events-mutex] lock]
  [set queue events [cons event [get queue events]]]
  [have [get queue events-mutex] unlock]]

[defunk a-procedure []
  [add-event my-queue `a-procedure-beginning-loop]
  [dotimes [i 10]
    [add-event my-queue `[a-procedure-loop ,i]]]
  [add-event my-queue `a-procedure-finished-loop]
  nil]

[defunk causal-organization []
  [cause-define parent-queue [new event_queue]]
  [partimes [i 5]
    [cause-define fiber-number i]
    [cause-define my-queue     [new event_queue]]
    [a-procedure]
    [add-event parent-queue `[fiber ,i finished with events: ,my-queue]]]
  parent-queue]

-u:--  causal_organization.fu2   Bot of 1.6k (47,0)     [(Funk)]--------------------
```

We define a way of adding events to our queue.

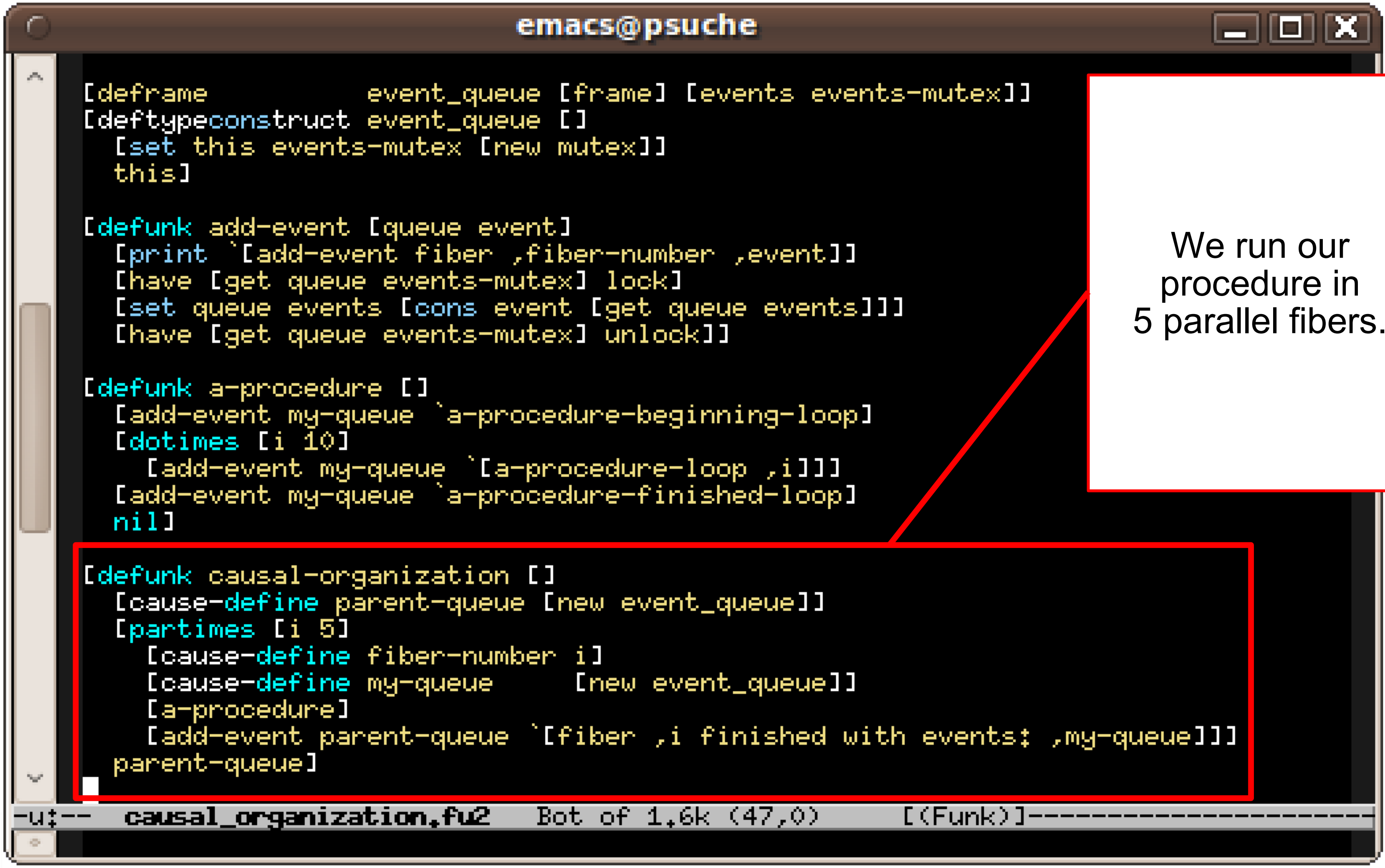
Cause Objects Organize Tracing


emacs@psuche

```
[deframe          event_queue [frame] [events events-mutex]]
[deftypeconstruct event_queue []
  [set this events-mutex [new mutex]]
  this]

[defunk add-event [queue event]
  [print `[add-event fiber ,fiber-number ,event]]
  [have [get queue events-mutex] lock]
  [set queue events [cons event [get queue events]]]
  [have [get queue events-mutex] unlock]]

[defunk a-procedure []
  [add-event my-queue `a-procedure-beginning-loop]
  [dotimes [i 10]
    [add-event my-queue `[a-procedure-loop ,i]]]
  [add-event my-queue `a-procedure-finished-loop]
  nil]

[defunk causal-organization []
  [cause-define parent-queue [new event_queue]]
  [partimes [i 5]
    [cause-define fiber-number i]
    [cause-define my-queue     [new event_queue]]
    [a-procedure]
    [add-event parent-queue `[fiber ,i finished with events: ,my-queue]]]
  parent-queue]

-u:--  causal_organization.fu2   Bot of 1.6k (47,0)     [(Funk)]----------------------
```

We define a procedure that we'd like to trace.

Cause Objects Organize Tracing

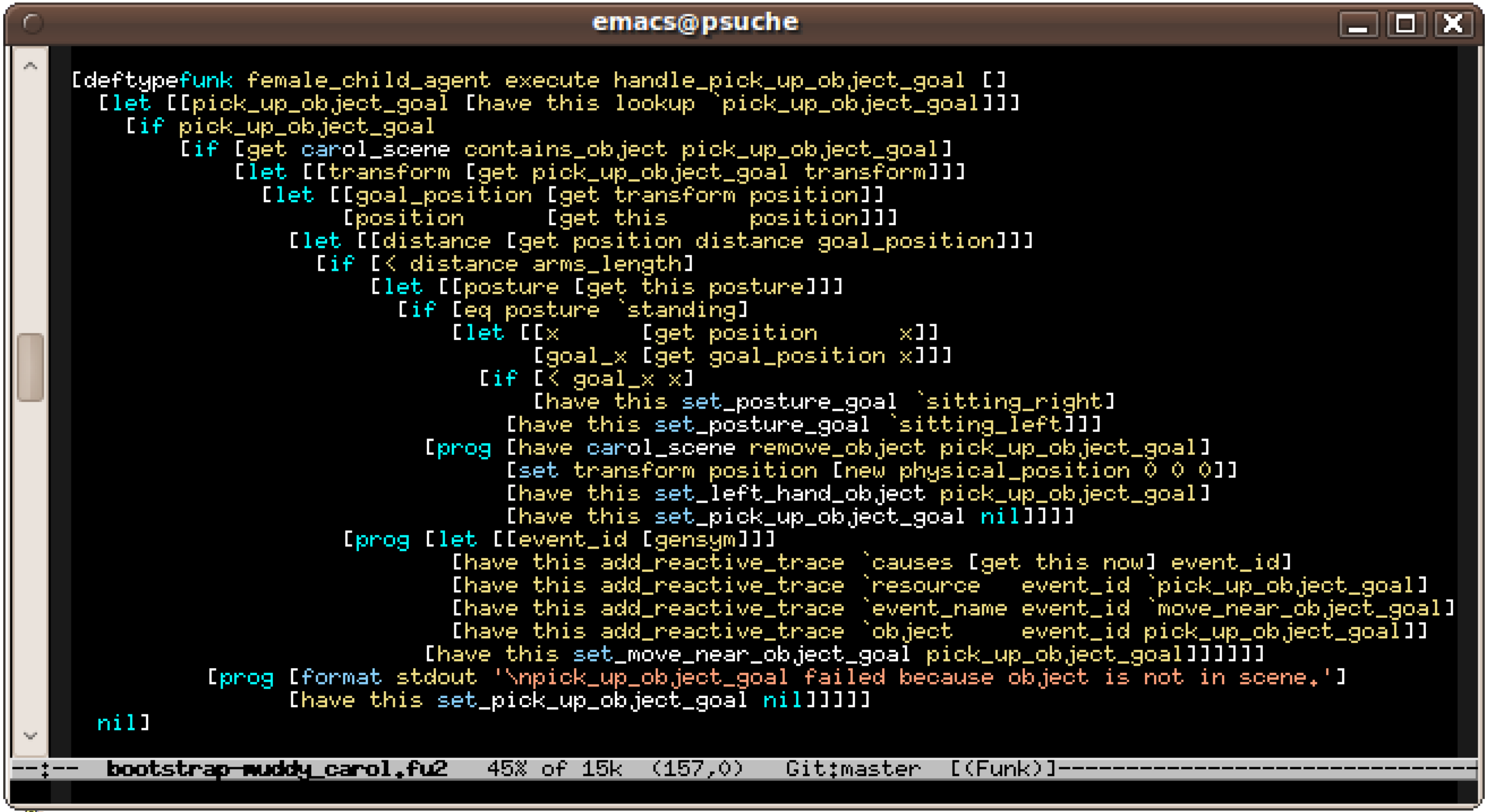


emacs@psuche

```
[deframe            event_queue [frame] [events events-mutex]]
[deftypeconstruct event_queue []
  [set this events-mutex [new mutex]]
  this]

[defunk add-event [queue event]
  [print `[add-event fiber ,fiber-number ,event]]
  [have [get queue events-mutex] lock]
  [set queue events [cons event [get queue events]]]
  [have [get queue events-mutex] unlock]]

[defunk a-procedure []
  [add-event my-queue `a-procedure-beginning-loop]
  [dotimes [i 10]
    [add-event my-queue `[a-procedure-loop ,i]]]
  [add-event my-queue `a-procedure-finished-loop]
  nil]

[defunk causal-organization []
  [cause-define parent-queue [new event_queue]]
  [partimes [i 5]
    [cause-define fiber-number i]
    [cause-define my-queue     [new event_queue]]
    [a-procedure]
    [add-event parent-queue `[fiber ,i finished with events: ,my-queue]]]
  parent-queue]

-u:--  causal_organization.fu2   Bot of 1.6k (47,0)     [(Funk)]--------------------
```

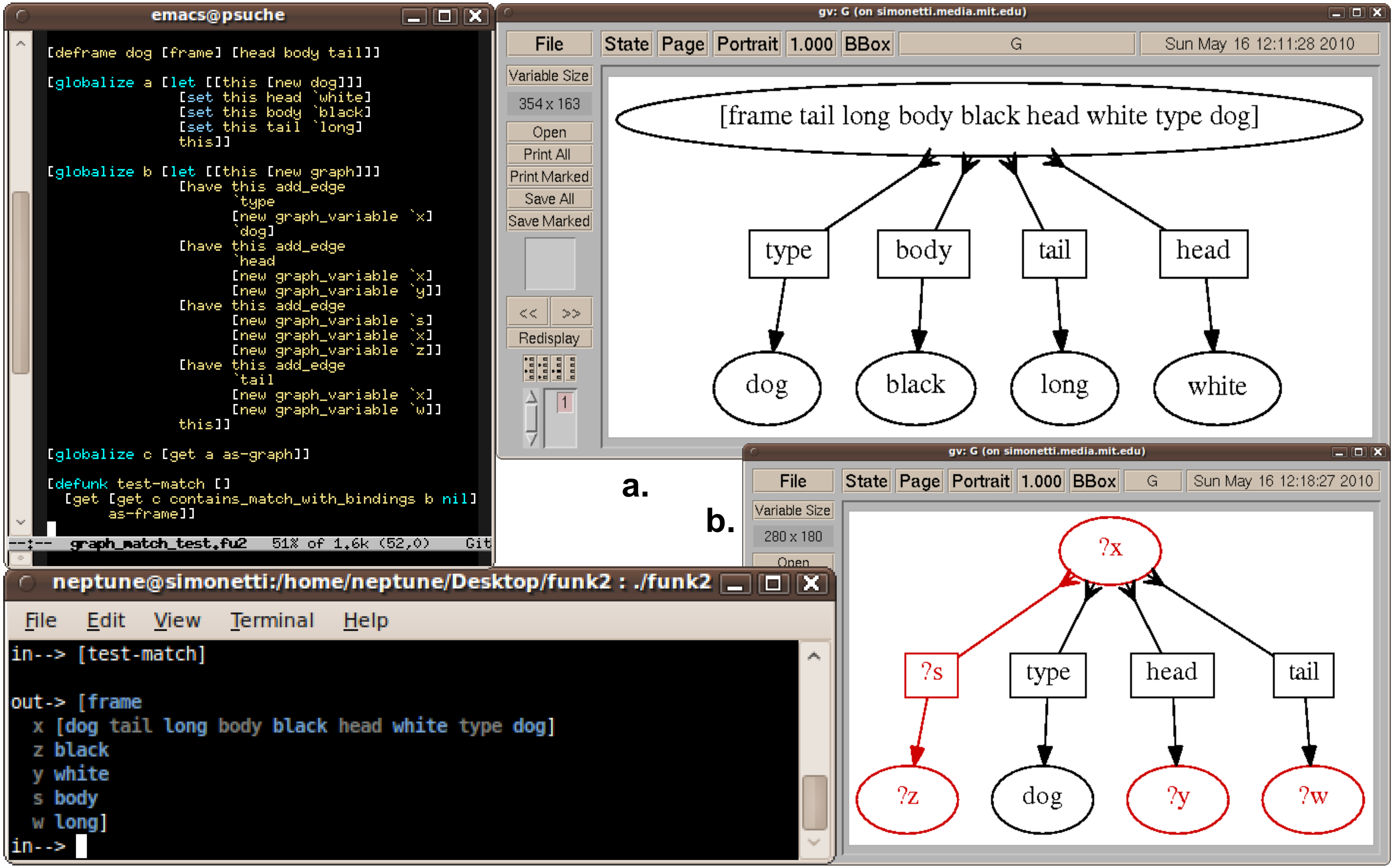
We run our procedure in 5 parallel fibers.

Procedural Critic-Selector (not Logic)



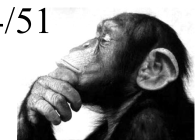
```
[deftypefunk female_child_agent execute handle_pick_up_object_goal []
  [let [[pick_up_object_goal [have this lookup `pick_up_object_goal]]]
    [if pick_up_object_goal
        [if [get carol_scene contains_object pick_up_object_goal]
            [let [[transform [get pick_up_object_goal transform]]]
              [let [[goal_position [get transform position]]
                    [position      [get this      position]]]
                [let [[distance [get position distance goal_position]]]
                  [if [< distance arms_length]
                      [let [[posture [get this posture]]]
                        [if [eq posture `standing]
                            [let [[x      [get position      x]]
                                  [goal_x [get goal_position x]]]
                              [if [< goal_x x]
                                  [have this set_posture_goal `sitting_right]
                                [have this set_posture_goal `sitting_left]]]
                          [prog [have carol_scene remove_object pick_up_object_goal]
                                [set transform position [new physical_position 0 0 0]]
                                [have this set_left_hand_object pick_up_object_goal]
                                [have this set_pick_up_object_goal nil]]]]
                    [prog [let [[event_id [gensym]]]
                            [have this add_reactive_trace `causes [get this now] event_id]
                            [have this add_reactive_trace `resource   event_id `pick_up_object_goal]
                            [have this add_reactive_trace `event_name event_id `move_near_object_goal]
                            [have this add_reactive_trace `object     event_id pick_up_object_goal]]
                          [have this set_move_near_object_goal pick_up_object_goal]]]]]]
          [prog [format stdout '\npick_up_object_goal failed because object is not in scene.']
                [have this set_pick_up_object_goal nil]]]]]
  nil]
--:--  bootstrap-muddy_carol.fu2   45% of 15k  (157,0)   Git:master  [(Funk)]---------------------------
```

Graphs are General Representation

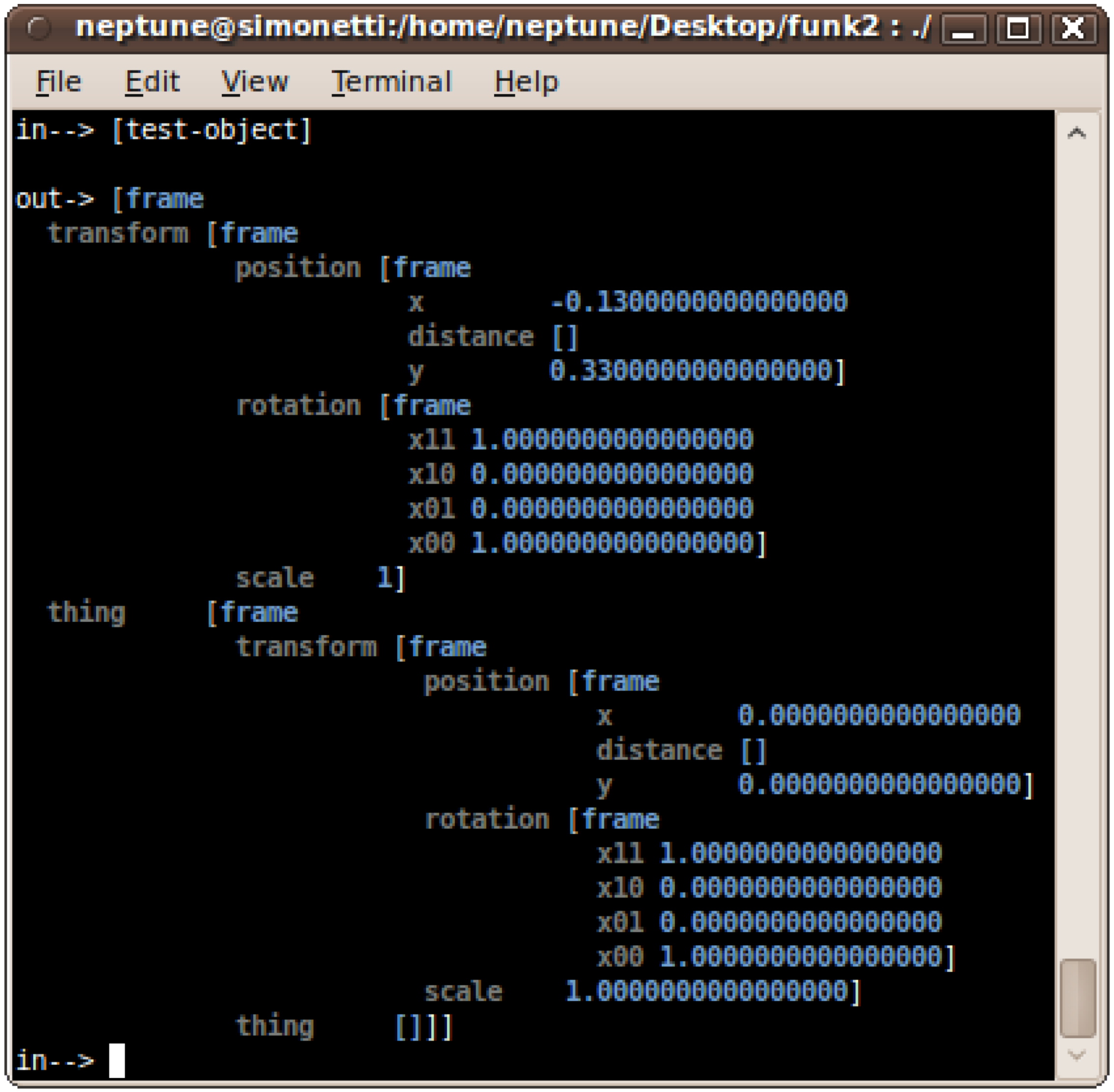


```
[deframe dog [frame] [head body tail]]

[globalize a [let [[this [new dog]]]
               [set this head `white]
               [set this body `black]
               [set this tail `long]
               this]]

[globalize b [let [[this [new graph]]]
               [have this add_edge
                     `type
                     [new graph_variable `x]
                     `dog]
               [have this add_edge
                     `head
                     [new graph_variable `x]
                     [new graph_variable `y]]
               [have this add_edge
                     [new graph_variable `s]
                     [new graph_variable `x]
                     [new graph_variable `z]]
               [have this add_edge
                     `tail
                     [new graph_variable `x]
                     [new graph_variable `w]]
               this]]

[globalize c [get a as-graph]]

[defunk test-match []
  [get [get c contains_match_with_bindings b nil]
       as-frame]]
--:--  graph_match_test.fu2    51% of 1.6k  (52,0)    Git
```

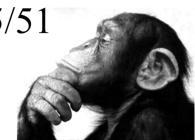
```
in-> [test-match]

out-> [frame
   x [dog tail long body black head white type dog]
   z black
   y white
   s body
   w long]
in->
```



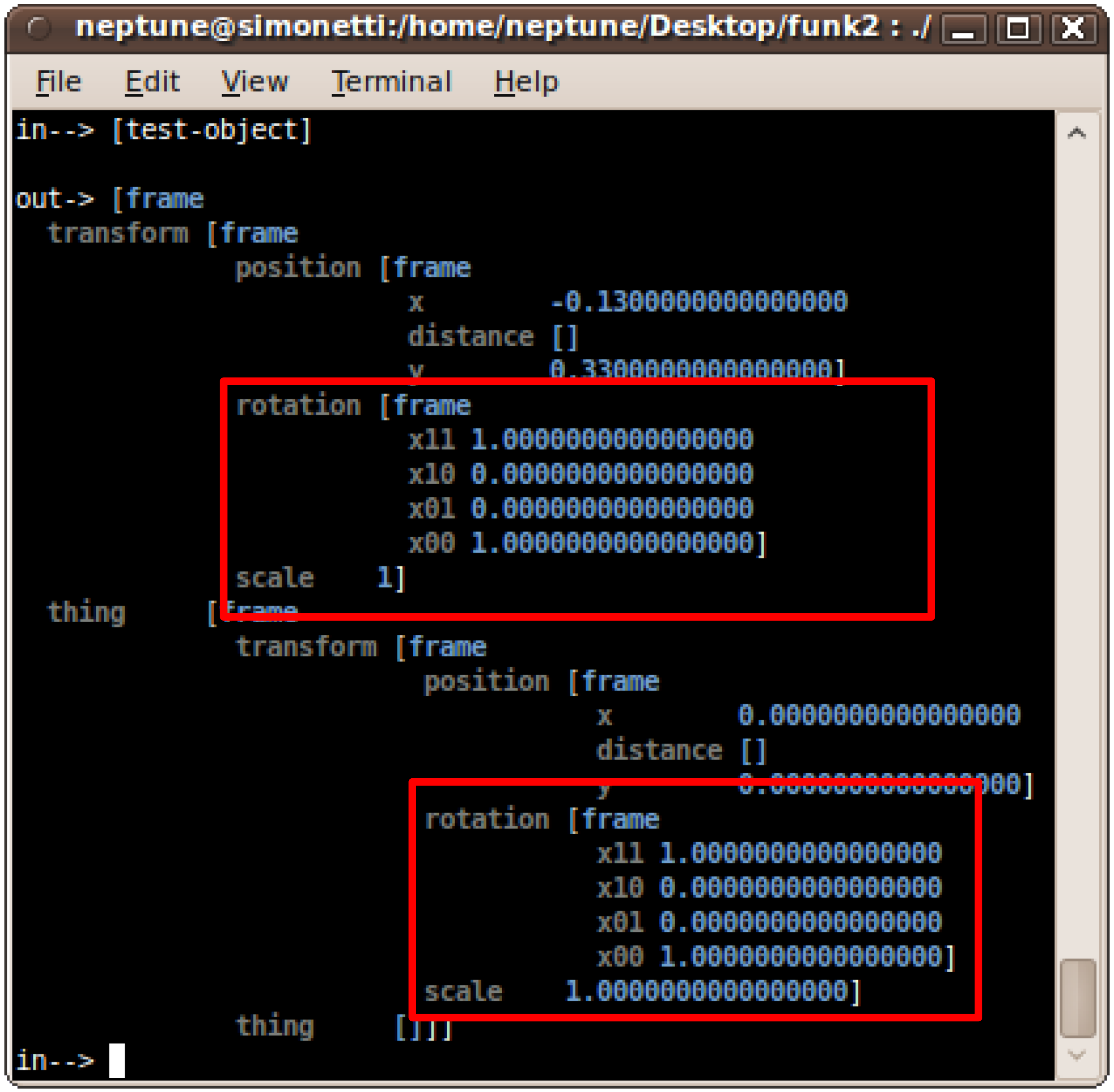
Matching Larger Patterns


neptune@simonetti:/home/neptune/Desktop/funk2 : ./
File Edit View Terminal Help

```
in--> [test-object]

out-> [frame
  transform [frame
            position [frame
                      x        -0.13000000000000000
                      distance []
                      y        0.33000000000000000]
            rotation [frame
                      x11 1.0000000000000000
                      x10 0.0000000000000000
                      x01 0.0000000000000000
                      x00 1.0000000000000000]
            scale    1]
  thing     [frame
            transform [frame
                        position [frame
                                  x        0.0000000000000000
                                  distance []
                                  y        0.0000000000000000]
                        rotation [frame
                                  x11 1.0000000000000000
                                  x10 0.0000000000000000
                                  x01 0.0000000000000000
                                  x00 1.0000000000000000]
                        scale    1.0000000000000000]
            thing     []]]
in-->
```

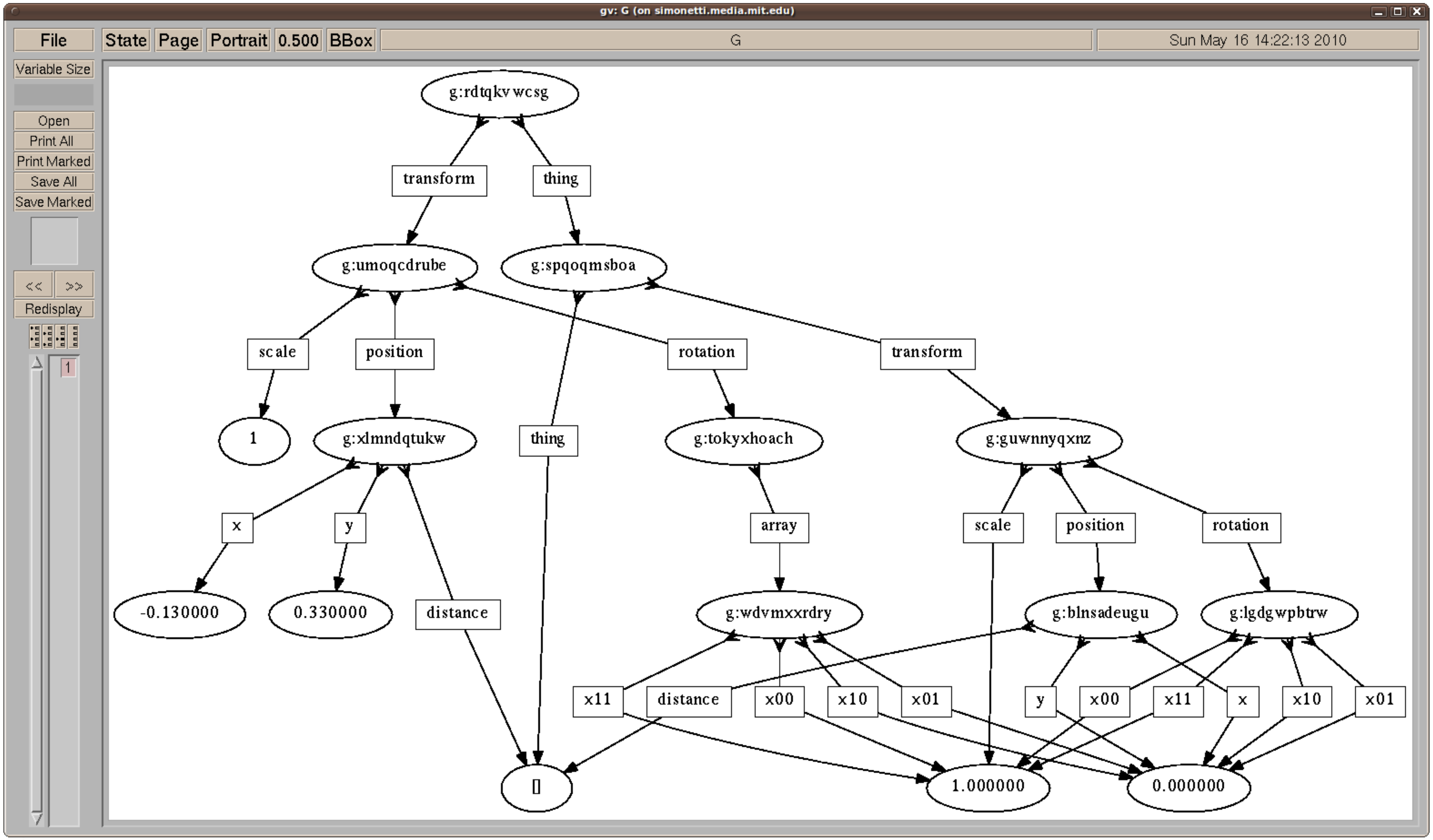


Matching Larger Patterns


neptune@simonetti:/home/neptune/Desktop/funk2 : ./
File Edit View Terminal Help

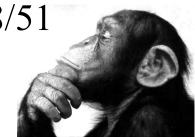
```
in--> [test-object]

out-> [frame
  transform [frame
            position [frame
                       x        -0.1300000000000000
                       distance []
                       y        0.3300000000000000]
            rotation [frame
                       x11 1.0000000000000000
                       x10 0.0000000000000000
                       x01 0.0000000000000000
                       x00 1.0000000000000000]
            scale    1]
  thing     [frame
            transform [frame
                        position [frame
                                   x        0.0000000000000000
                                   distance []
                                   y        0.0000000000000000]
                        rotation [frame
                                   x11 1.0000000000000000
                                   x10 0.0000000000000000
                                   x01 0.0000000000000000
                                   x00 1.0000000000000000]
                        scale    1.0000000000000000]
            thing    []]]
in--> 
```

Matching Larger Patterns

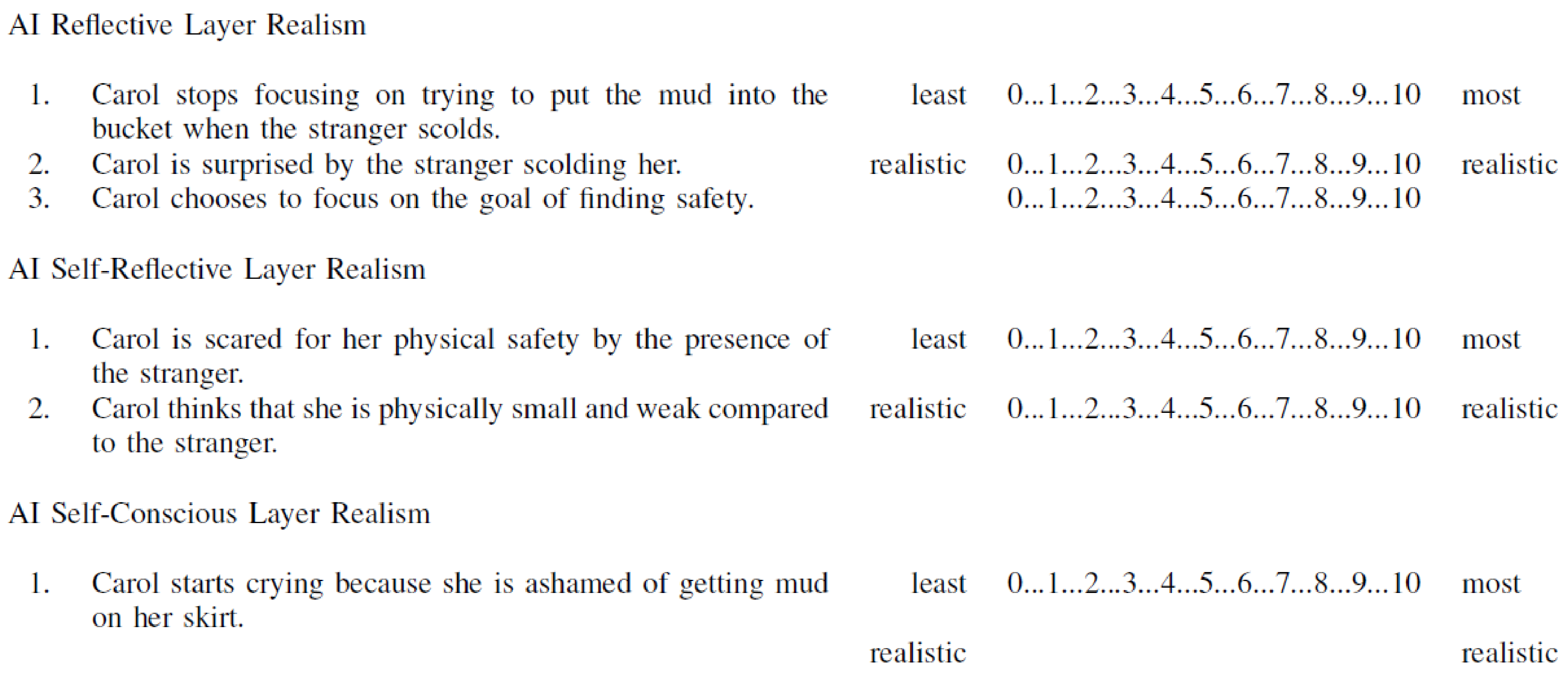


Evaluation

- A collection of social situations and tasks in the IsisWorld commonsense simulation environment.
- Three quantitative evaluations
  - Compare to relational reinforcement learning algorithms.
  - Compare to explanation-based learning (EBL) algorithms.
  - Measure performance degradation with “knockout” models.
- One qualitative evaluation
  - Questionnaire to measure plausibility and realism of mental events at different layers.



Qualitative Evaluation



AI Reflective Layer Realism

1.	Carol stops focusing on trying to put the mud into the bucket when the stranger scolds.	least	0...1...2...3...4...5...6...7...8...9...10	most
2.	Carol is surprised by the stranger scolding her.	realistic	0...1...2...3...4...5...6...7...8...9...10	realistic
3.	Carol chooses to focus on the goal of finding safety.		0...1...2...3...4...5...6...7...8...9...10	

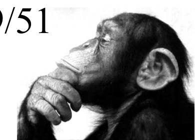
AI Self-Reflective Layer Realism

1.	Carol is scared for her physical safety by the presence of the stranger.	least	0...1...2...3...4...5...6...7...8...9...10	most
2.	Carol thinks that she is physically small and weak compared to the stranger.	realistic	0...1...2...3...4...5...6...7...8...9...10	realistic

AI Self-Conscious Layer Realism

1.	Carol starts crying because she is ashamed of getting mud on her skirt.	least	0...1...2...3...4...5...6...7...8...9...10	most
		realistic		realistic

Example questions evaluating the realism of the execution of mental resources within the top three layers of our proposed model.



Timeline

- May 2011 – Defend PhD
- March–April 2011 – Finish writing dissertation
- January–February 2011 – Evaluation
- December 2010 – Self-conscious reasoning
- November 2010 – Self-reflective reasoning
- October 2010 – Reflective reasoning
- September 2010 – Case-based reasoning in IsisWorld
- August 2010 – Analogical graph mapping
- June 2010 – Lattice graph matching

Only the beginning...

This work exists thanks to the
inspiration, advice, and support of

Marvin Minsky	Society of Mind Group	MIT Media Lab
Joseph Paradiso	Responsive Environments Group	MIT Media Lab
Henry Lieberman	Software Agents Group	MIT Media Lab
Dustin Smith	Software Agents Group	MIT Media Lab
Gerald Sussman	Neutral Computer Science Research	MIT CS+AI Lab
Michael Cox	IPTO	DARPA
Newton Howard	Mind Machine Project	MIT

and many others...