Towards Flexible Task Environments for Comprehensive Evaluation of Artificial Intelligent Systems & Automatic Learners

Kristinn R. Thórisson

Associate Professor, *School of Computer Science*, *Reykjavik University, Iceland*

Co-Founder, *Center for Analysis and Design of Intelligent Agents, Reykjavik U.*

Founder & Managing Director, *Icelandic Institute for Intelligent Machines, Reykjavik, Iceland*

Jordi Bieger, Stephan Schiffel & Deon Garrett

Outline

- Back Story (motivation)
- Identified Issues (problem dissection)
- Derived Requirements (contribution I)
- Draft of a Solution (contribution II)
- Remaining Issues (conclusion)

Back Story

We had implemented a reinforcement learning algorithm with new ideas for handling continous input & output

- RL X - *old version*
- RL X' - *RL X + continous data I/O mod*

Back Story

We asked the questions:

- *Which is better, this particular implementation of reinforcement learner X, or our modified (supposedly enhanced) version of X'?*

- *Can X' even replicate X?*

Back Story

We had:

- Old performance data for X on task T1
- No verification of X'
- No current setup to run T1
- Scarce documentation for T1

Back Story

- Old performance data for X on task T1
- No verification of X'
- Improper documentation for T1
- T1 was a major effort to set up, but provided *valuable data* about X
- X' had features not tested by T1

Back Story

The Conundrum: Should we …

- reconstruct T1 and re-run it for both X and X'?
- construct a new test T2 and run for only X', then compare to old T1 data?
  - If we created a task T2, how would we compare it to T1?
- construct a new task T2 that subsumed T1, run for both X and X'?
  - How would we know that T2 properly subsumed T1?

Back Story

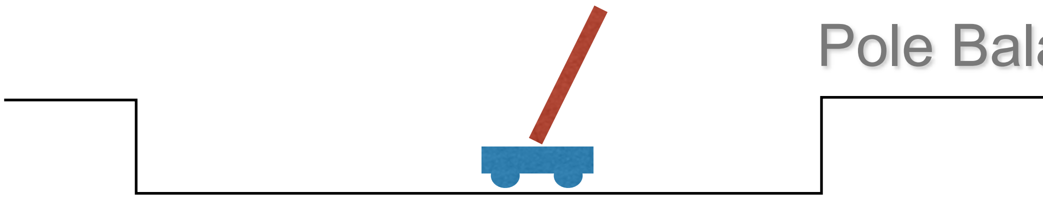
The answer: None of the above

- We did something else…

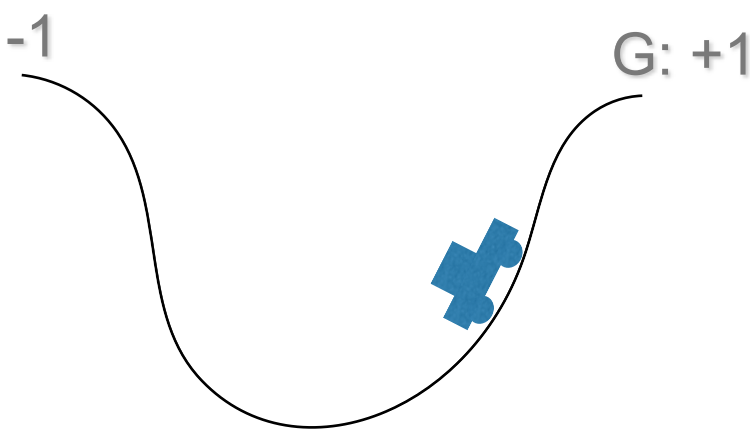
Unanswered Questions

  - Lots of performance data already available for various learners, on various tasks
  - Problem: tasks vary widely
- How do you compare performance of algo X on task A to performance of algo Y on task B?

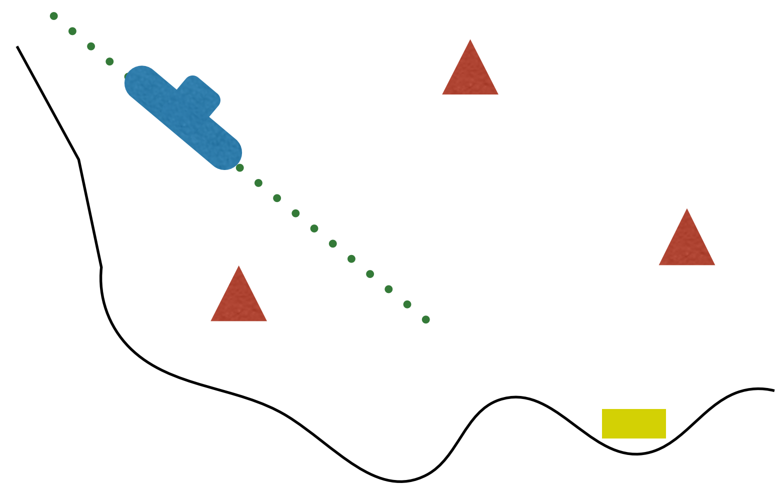
Unanswered Questions



Pole Balancing Task



"Mountain Car" Task



"Diving for Gold" Task

Unanswered Questions

- Lots of tasks already proposed
- Problem: applicable to only a (small) set of targeted learners

- How do you expand a task to become more complex, or with some new features, in a predictable way?

Unaddressed Problems

- Tasks for simple learners (e.g. RL) generally not applicable to AGI-aspiring systems…
- …and vice versa

Unaddressed Problems

- Turing Test

vs

- PacMan

Unaddressed Problems

- Turing Test
- Pong

vs

- Lovelace Test
- PacMan

Unaddressed Problems

- Turing Test ⟷ • Pong

vs

- Lovelace Test ⟷ • PacMan

Unaddressed Problems

- Turing Test
- Pong
- Lovelace Test
- PacMan

Unaddressed Problems

- Turing Test
- Pong
- Lovelace Test
- PacMan

Unaddressed Needs

- Lack of methods for evaluating:
    - Learning Capacity
        - speed, amount, generality
    - Lifelong Learning (and forgetting)
    - Transfer Learning
    - Cognitive Development

What is needed:

A framework for constructing transparent tasks

Why Evaluate AI Systems?

- Assess research progress
- Evaluate strengths/weakneses
- Compare systems and approaches

Goals

- To develop a framework for constructing transparent task-environments, offering:
  - easy construction of abstract task-environments and variants
  - automated generation
  - easy analysis of features of interest

Goals

- What features should task-environments constructed in this framework have?

Desired Features of Framework

- Determinism: Complete (100% repeatability) to partial (or zero)
- Periodicity: support of temporal patterns
- Controllable Continuity: Discretization should be flexibly determinable

Desired Features of Framework

- Asynchronicity: events at arbitrary time-scales
- Dynamism: controllable from highly dynamic to completely static
- Observability: controllable to achieve certain characteristics of tasks with partially-observable variables

Desired Features of Framework

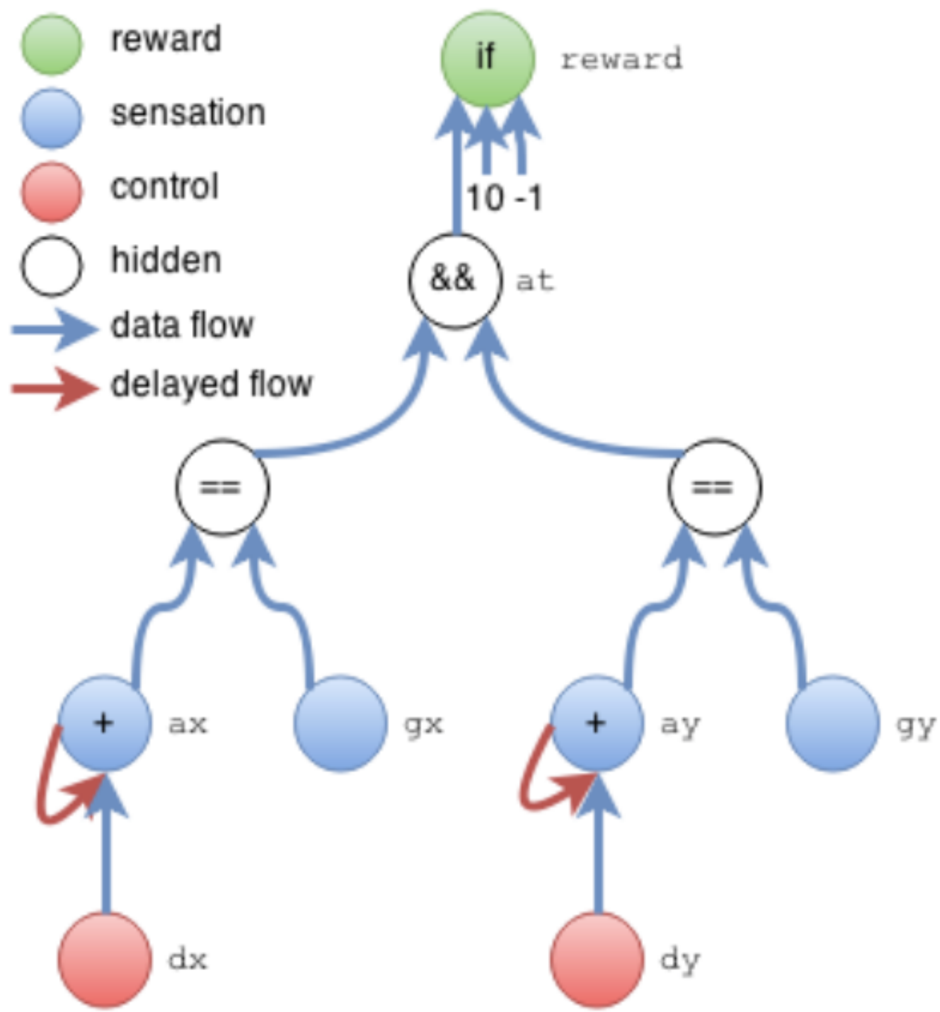
- Controllability: the amount of control a learner has on the task-environment should be tunable
- Multiple Parallell Causal Chains: for systems entertaining multiple simultaneous goals, MPCCs create distractors, noise, long event chains, etc.
- Number of Agents: support of multiple (intelligent) agents with causal effects

Proposed Solution: Early Draft

- Describe the task-environment by a set of time-dependent variables and their relations
- Causal chains constructed via numerous serially related variables
  - Nature of relations determines nature of tasks in predicable ways, e.g. increased control complexity through temporal latencies

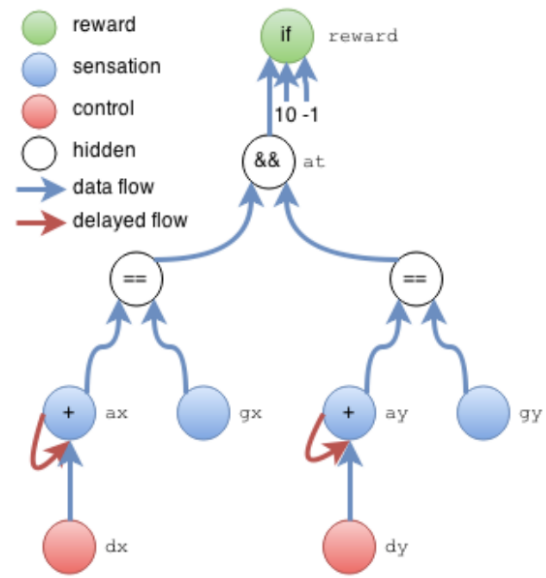
Proposed Solution: Early Draft

- Example task shown as a diagram



Proposed Solution: Early Draft

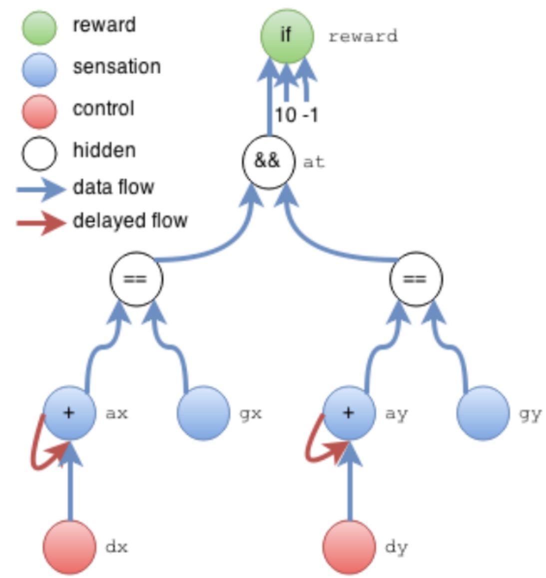
```
Initialization:
   gx = 3   // goal x
   gy = 3   // goal y
   ax = 4   // agent x
   ay = 10  // agent y
Dynamics:
   dx(t) = 0   // step x
   dy(t) = 0   // step y
   ax(t) = ax(t-dt) + dx(t)
   ay(t) = ay(t-dt) + dy(t)
   at(t) = ax(t) == gx(t) &&
           ay(t) == gy(t)
   reward(t) = 10 if at(t) else -1
Terminals:
   reward(t) > 0
Rewards:
   reward(t)
Observations:
   ax(t), ay(t), gx(t), gy(t)
Controls:
   dx(t) = [-1, 0, 1]
   dy(t) = [-1, 0, 1]
```



- Same task shown in program form

Proposed Solution: Early Draft

```
Initialization:
    gx = 3   // goal x
    gy = 3   // goal y
    ax = 4   // agent x
    ay = 10  // agent y
Dynamics:
    dx(t) = 0   // step x
    dy(t) = 0   // step y
    ax(t) = ax(t-dt) + dx(t)
    ay(t) = ay(t-dt) + dy(t)
    at(t) = ax(t) == gx(t) &&
            ay(t) == gy(t)
    reward(t) = 10 if at(t) else -1
Terminals:
    reward(t) > 0
Rewards:
    reward(t)
Observations:
    ax(t), ay(t), gx(t), gy(t)
Controls:
    dx(t) = [-1, 0, 1]
    dy(t) = [-1, 0, 1]
```



- This is a simple task that is **discrete**, **fully observable**, and **static**

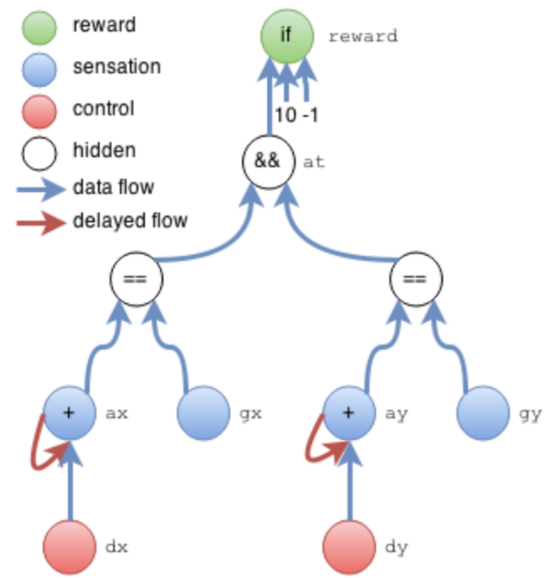
Proposed Solution: Early Draft

```
 1. Initialization:
 2.    gx = 3   // goal x
 3.    gy = 3   // goal y
 4.    ax = 4   // agent x
 5.    ay = 10  // agent y
 6. Dynamics:
 7.    dx(t) = 0   // step x
 8.    dy(t) = 0   // step y
 9.    ax(t) = ax(t-dt) + dx(t)
10.    ay(t) = ay(t-dt) + dy(t)
11.    at(t) = ax(t) == gx(t) &&
               ay(t) == gy(t)
12.    reward(t) = 10 if at(t) else -1
13. Terminals:
```

```
7. dx(t) = dt * cos(angle(t))        8. dy(t) = dt * sin(angle(t))

11. reward(t) = 10 if (ax(t)-gx(t))^2 + (ay(t)-gy(t))^2 < 1 else -1
```

```
17. Observations:
18.    ax(t), ay(t),
19. Controls:
20.    dx(t) = [-1,
21.    dy(t) = [-1,
```



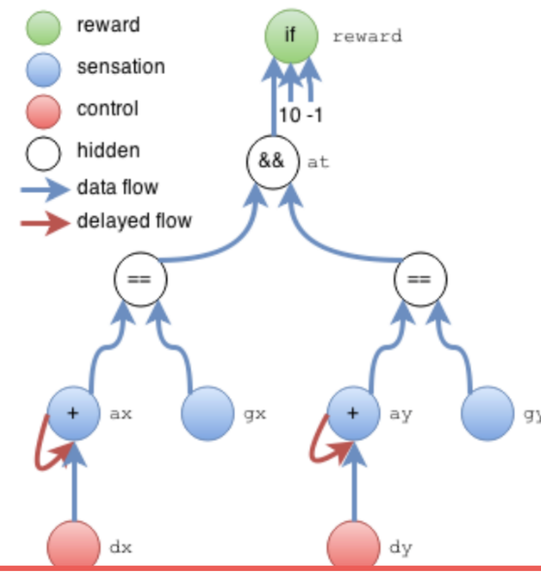
- To make it more **continous** we can e.g. add a float representing angle

Proposed Solution: Early Draft

```
1.  Initialization:
2.     gx = 3   // goal x
3.     gy = 3   // goal y
4.     ax = 4   // agent x
5.     ay = 10  // agent y
6.  Dynamics:
7.     dx(t) = 0   // step x
8.     dy(t) = 0   // step y
9.     ax(t) = ax(t-dt) + dx(t)
10.    ay(t) = ay(t-dt) + dy(t)
11.    at(t) = ax(t) == gx(t) &&
               ay(t) == gy(t)
12.    reward(t) = 10 if at(t) else -1
```

```
17. ax(t-dt), ay(t-dt)          18. gx, gy @ [1:2:]
```

```
16.    reward(t)
17. Observations:
18.    ax(t), ay(t), gx(t), gy(t)
19. Controls:
20.    dx(t) = [-1, 0, 1]
21.    dy(t) = [-1, 0, 1]
```



- To make it harder we can e.g. **decrease observability**

Does This Approach Scale?

- Can this seemingly simplistic approach help us with evaluating AGI-aspiring systems?
- We think so. For instance:
  - Larger-size tasks: Because of its simplicity automated construction should not be possible
  - Given a high-level spec for desired constraints, worst-case scenario: desired properties reached by brute-force

Remaing Work

- Developing good measurements of task-environment complexity
- Developing good measurements of task-environment difficulty (given initial & goal states)
  - Will depend in part on identification and classification of larger task-environment patterns than we have looked at so far