Env.lhs

James Cook

Oct 2007

Abstract

Basic sketch of moku's ideas for an "agent-based execution" abstraction in Haskell. The objective is to devise:

- A general execution-strategy-independent way of specifying game behavior,
- An independent abstraction for describing agent behavior in a similarly strategy-independent way, and
- A framework for tying those together with various actual execution strategies.

I believe that this abstraction is equally suitable for continuous-time simulations as turn-based ones.

1 Overview

The abstraction is composed of two parts - "queries" and "actions". These are interfaces between agent and game. From the agent's perspective,

Games may alternately be defined in terms of specific execution strategies if so desired, although that of course will limit the generality of that specific game. The agent-interface may be similarly specialized.

2 Queries

2.1 Base Abstraction

Queries are how agents get information from the environment. The core of the "query" concept is the *EnvQuery* class. The meaning of the type parameters is as follows:

e The type of the environment (world state)

a: The type of an "agent identifier" that the query function may use to determine how to respond to the query. Most execution strategies will attach this to a monad in which the agent runs, hopefully in such a way that the agent will not be able to "forge" it.

q: The type of the "query" - a data type whose values describe the information that the agent is requesting.

r: The type of the "response" to the query.

```
class EnvQuery e a q r
  where
    queryEnv :: e → a → q → r
```

2.2 Monadic Queries

The preferred way to implement a query, if possible, is to implement instances of *EnvQuery*. Standard "Execution Strategies" consist of mappings from *EnvQuery* to *EnvQueryM*, a class defining a monadic action, *query*, that hides the passing of state (and ideally also protects that state from unauthorized access).

If necessary, an *EnvQueryM* instance may be directly defined.

EnvQuery defines one function, *query*, that takes a "query" and returns a "response," as above, but hides the passing of the environment and agent-id.

```
class (Monad m) ⇒ EnvQueryM m e a q r
  where
    query :: q → m r
```

3 Actions

3.1 Base Abstraction

```
class EnvAction e a c
  where
    actEnv :: e → a → c → Either e String
```

3.2 Monadic Actions

```
class (Monad m) ⇒ EnvActionM m e a c
  where
    act :: c → m ()
```

4 A Sample Implementation

```
instance (MonadState e m, MonadReader a m, EnvAction e a c)
  ⇒ EnvActionM m a e c
  where
    act cmd = do
      env ← get
      agent ← ask
      case actEnv env agent cmd of
        Left env' → put env'
        Right err → fail err
instance (MonadState e m, MonadReader a m, EnvQuery e a q r)
  ⇒ EnvQueryM m e a q r
  where
    query q = do
      env   ← get
      agent ← ask
      return (queryEnv env agent q)
```