Artificially Intelligent Haskell
Overview

- About Aetion
- What we do, and how we do it
- J qy "HR"j cu"*cpf "j cupÍv4"j gm gf
About Aetion

- Approach to AI based on widening regions of confidence based on explanatory necessity
- Most useful in domains with many overlapping possibilities or sparse data
- Currently supported by DoD contracts
The Challenges

- Frequent Changes
  - Problem domains change
  - New domains
  - Changing approaches to existing domains

- Want to minimize code impact of changes
The First Solution

- Minimize work on things not directly relevant to current applications
- Fill in the gaps later.
The First Solution

- Minimize work on things not directly relevant to current applications
- Fill in the gaps later.

But:
- No general pattern for extension
- Most problems required non-localized changes.
The Second Solution

- Find an abstraction and stick with it!
- Blueprint for Second-system Effect
- How did we make it work anyway?
FP’s contributions

- The types are cool
- The code is short
- The ideas are good
The Types are Cool

- Find many bugs before they happen
- Separating effectful and non-effectful code
- BUT: extensible records would be handy
The Code is Short

- Big advantage is not in writing, but in rewriting.
The Code is Short

- Big advantage is not in writing, but in rewriting.
- 10K lines of Haskell vs. 30K lines of Java
The Ideas are Good

- Monads, arrows, continuations, various combinator libraries, GADTs, even higher order functions.
- Makes over-generality easy.
The Result

Three projects:

- Generalized data/change management (SNOWDRIFT)
  - Maintains relational view of data with constant time relation traversal
  - Provides priority-based process scheduling
  - Maintains tree of event handlers [Grust 1999]
  - Starts lightweight threads based on thrown events
Event Handlers

Ideal:

```
[ h |- confidence <|- (1 +)
  | i <- h |- linkedFrom Supports
 , i |- belief == Accepted ]
```
Event Handlers

Ideal:

\[
[ h |- confidence <= (1 +) \\
  | i <- h |- linkedFrom Supports \\
  , i |- belief == Accepted ]
\]

Actual, almost:

\[
[ h |- confidence <= (1 +) \\
  | ModAttr i Belief _ Accepted = theEvent \\
  , h <- i |- linkedFrom Supports ]
\]
The Result

- Three projects:
  - Generalized data/change management
  - Inference rules (WINTERMUTE)
    - Compositional constructors
      - `makeNamed "blah"
      - `makeLocated (4,5)
      - `makeExplainerer `addMutable
    - Triggers propagate changes based on introduced relationships
The Result

- Three projects:
  - Generalized data/change management
  - Inference rules
  - Application-specific hackery (PAWPRINTS)
    - Mostly pure Haskell
    - Mostly pure Haskell
The Result

- Three projects:
  - Generalized data/change management
  - Inference rules
  - Application-specific hackery

- All work pretty well!
Rogue’s Gallery

- Concurrency

\[ h \vdash attr \leftarrow val \]

But what about?

\[ h_1 \vdash attr_1 \leftarrow val_1 \]
\[ h_2 \vdash attr_2 \leftarrow val_2 \]
Rogue’s Gallery

- Something more like:

\[ h_1, h_2 \vdash \]
\[ \text{attr1 } h_1 \leftarrow \text{val1} \]
\[ \text{attr2 } h_2 \leftarrow \text{val2} \]

- But this doesn’t compose
Rogue’s Gallery

- Records (Compositional Construction)

```haskell
data Entity e => Thing e = Thing Stuff e

makeFoo >>= makeBar >>= makeThing >>= addMutable

Then, we want uniform access to composed objects:

Class ThingLike t
  where getStuff :: t -> Stuff
        setStuff :: Stuff -> t -> t
```
Rogue’s Gallery

- Records (continued)

But then composition is hard:

```haskell
data Entity e => Thing e = Thing Stuff e

instance (c e) => c (Entity e)
  where ...
```
What Didn’t Work

- Client/Server Interface
  - Serialization requirements forced some design decisions.
  - Frequently pushing changes removed many possible uses of laziness
What Didn’t Work

- Client/Server Interface
  - Serialization requirements forced some design decisions.
  - Frequently pushing changes removed many possible uses of laziness.

- But: that’s mostly our fault.

- What’s the better way?
What Didn’t Work

- Management buy-in
  - Maintenance and maturity of libraries
  - Tools-related projects
What Can Work Better

- Libraries working together
- Cross-platform compatibility
- More tools development
  - Debugging
  - Profiling — performance seems like a black art
- Programmer interest

- But: chicken and egg problem?
Thank you