Toward Denotational Semantics of Domain-Specific Modeling Languages for Automated Code Generation

Danielle Gaither, Barrett R. Bryant

{dcg0063, barrett.bryant} @ unt.edu

sell.cse.unt.edu coming soon

Agenda

- Introduction
- Denotational semantics
  - General modeling languages
  - MicroRPG
- Haskell implementation
- Integration with existing tools
- Related work and conclusion

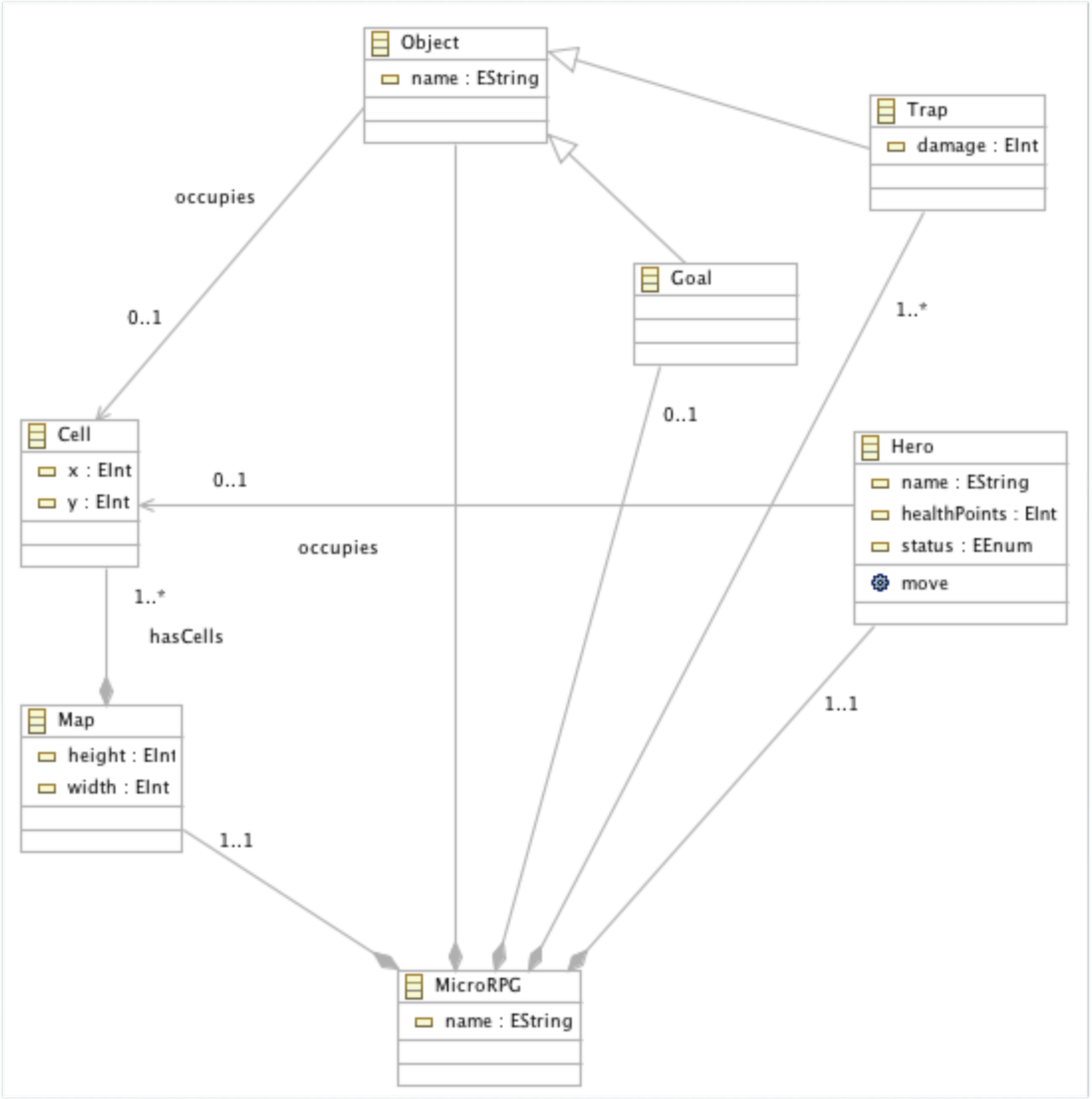
Introduction

- Goal: automated code generation
- Need to establish semantics for source language
- Work to date mostly concerns operational semantics
- Advantages of approach based on denotational semantics

MicroRPG DSL

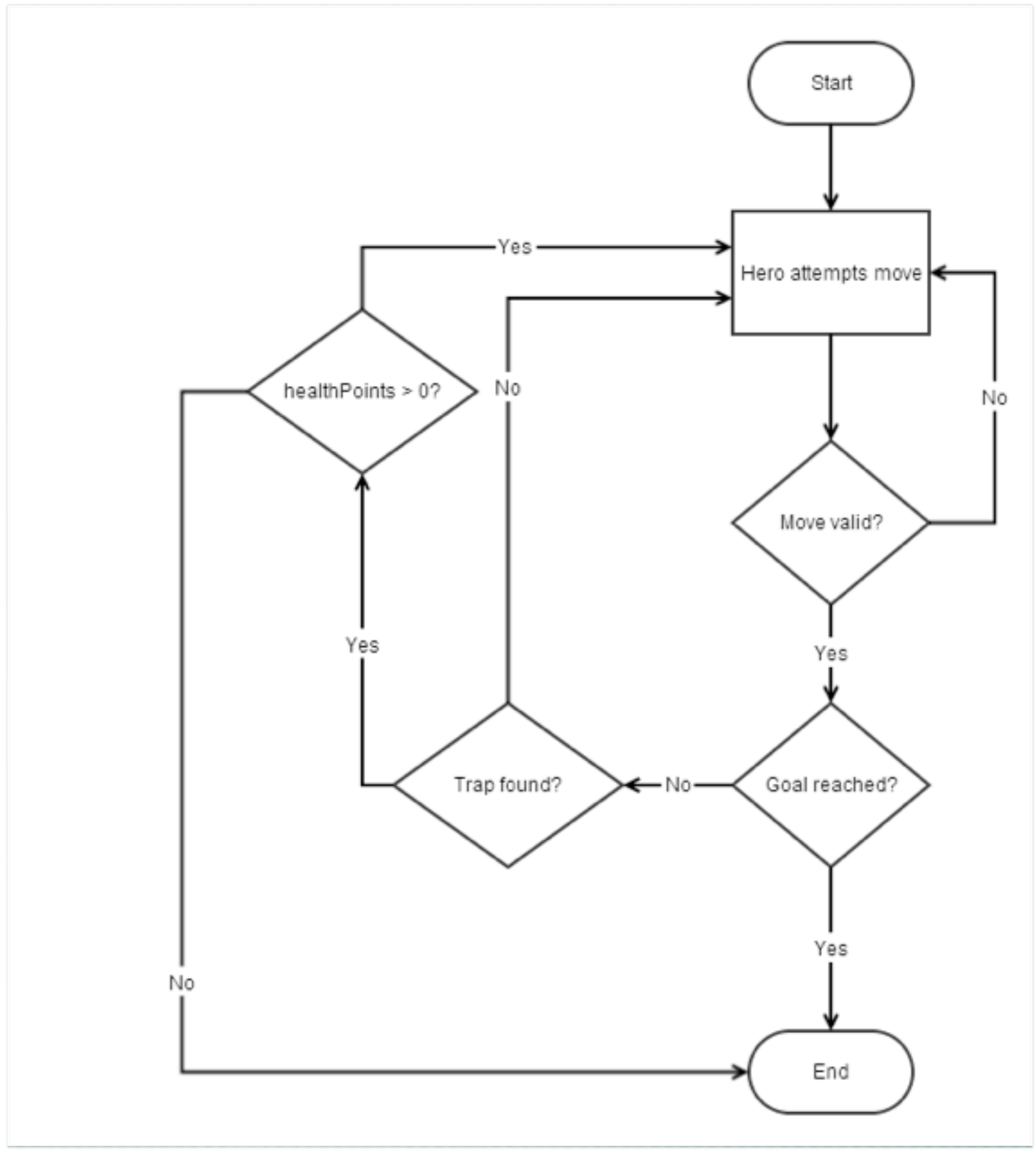
- Simplified version of RPG DSL by Marques et al.[1]
- Metamodel shows components of MicroRPG
- Game ends when hero reaches goal or runs out of health points

[1] E. Marques, V. Balegas, B. Barrocca, A. Barisic and V. Amaral, "The RPG DSL: a case study of language engineering using MDD for generating RPG games for mobile phones," in Workshop on domain-specific modeling, Tucson, AZ, 2012.



MicroRPG metamodel

System Overview



Denotational semantics overview

- Roots in compiler construction
- Syntax analysis checks for conformance to metamodel
- Representations of semantics less standardized than for syntax
- Focusing on graphical modeling languages

Denotational semantics of MicroRPG

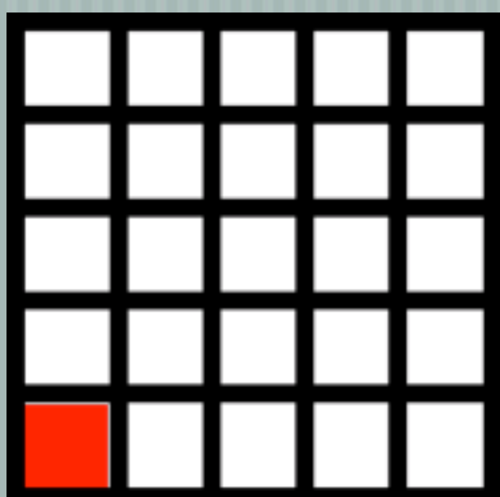
- Syntax domains are classes in metamodel
- Semantic domains include syntax domains + functions that return attributes
- Rules of MicroRPG
  - Not all moves valid
  - Can be specified with constraints

Trying a move

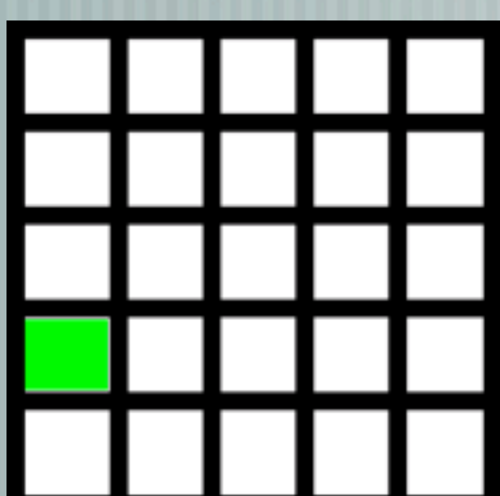
- Need to know:
  - Map of game
  - Current location
  - Desired direction
- If move is invalid, return current space
- Otherwise, return new space

Sample Move

From



To



$$
\begin{aligned}
&\mathit{move}\colon \mathit{GameMap}\times\mathit{Cell}\times\mathit{Direction} \to \mathit{Cell}\\
&\qquad \mathit{move}\ [\![\mathit{gameMap}\ (1{,}1)\ \mathit{Up}]\!]\\
&\qquad = \mathit{tryMove}\ [\![(x1, y1)(x2, y2)]\!]\\
&\qquad = \mathit{tryMove}\ [\![(1{,}1)\ (2{,}1)]\!]\\
&= \text{if } x2 < 1 \text{ or } x2 > \mathit{width}(\mathit{gameMap}) \text{ or}\\
&\quad y2 < 1 \text{ or } y2 > \mathit{height}(\mathit{gameMap}) \text{ then } (x1, y1)\\
&\quad \text{else } (\mathrm{x2}, \mathrm{y2})\\
&= (\mathrm{x2}, \mathrm{y2})\\
&= (1, 2)
\end{aligned}
$$

Denotational semantics of specified move

Why Haskell?

- Amenable to formal verification
- Popular choice for language implementations
- Syntax domains easily translate to Data or Type declarations in Haskell

Sample declarations

```
type Cell = (Int, Int)


data Hero = Hero {
  name :: String,
  position :: Cell,
  health :: Int,
  status :: Status}
 deriving (Show,Read,Eq)
```

Sample method

```
tryMove :: Cell -> Cell -> Int -> Int ->
Cell

tryMove (x1,y1) (x2,y2) width height

  | x2 < 1 = (x1,y1)

  | x2 > width = (x1,y1)

  | y2 < 1 = (x1,y1)

  | y2 > height = (x1,y1)

  | otherwise = (x2,y2)
```

Integration with existing tools

- Marques et al. (see slide 5) used EMF
- Functional languages popular choices for executable denotational semantics
- Denotational semantics can be thought of as DSL in its own right

Language choices

- Haskell
- Scala
- Scheme
- Clojure
- OCaml

Related work

- Lots of work on semantics of metamodels
- Not so much on denotational semantics
- Denotational semantics can be used as a model itself
  - Writing statically verifiable contracts[1]
  - Determine values of financial contracts[2]s

[1] D. Vytiniotis, S. Peyton Jones, D. Rośen and K. Claessen, "HALO: Haskell to logic through denotational semantics," in Principles of Programming Languages, Rome, 2013.
[2] S. Peyton Jones, J.-M. Eber and J. Seward, "Composing contracts: an adventure in financial engineering - Functional pearl," in International Conference on Functional Programming, Montreal, 2000.

Conclusions & future work

- Mapping metamodel components to denotational semantics
- Approach lends itself to implementation in functional language
- Example shown
- Amenable to composition and formal analysis techniques
- Next steps
  - Automating the mapping between model components and code
  - Working with automated proofing systems

Code demo