A Model of a Fragment of English

Kenneth Lai

Brandeis University

October 12, 2022

Announcements

- For tomorrow (but can probably wait until next Tuesday)
  - Read van Eijck and Unger Chapter 8
- For 10/19
  - HW2 due
  - Paper Presentation Ideas due

Today's Plan

- Paper Presentation Ideas: Discourse and Dialogue, and Multimodal Semantics
- Predicate Logic Exercises
- User-defined Data Types
- A Model of a Fragment of English

Discourse and Dialogue

- AMR for dialogue: Bonial et al. 2020. Dialogue-AMR: Abstract Meaning Representation for Dialogue. Proceedings of LREC.
- TalkMoves in the classroom: Suresh et al. 2022. The TalkMoves Dataset: K-12 Mathematics Lesson Transcripts Annotated for Teacher and Student Discursive Moves. Proceedings of LREC.
- Also see SIGDIAL, SemDial venues

Multimodal Semantics

- Aligning images and text with semantic role labels: Bhattacharyya et al. 2022. Aligning Images and Text with Semantic Role Labels for Fine-Grained Cross-Modal Understanding. Proceedings of LREC.
- Aligning visual and textual vector spaces: Yun, Kim, and Jung. 2022. Modality Alignment between Deep Representations for Effective Video-and-Language Learning. Proceedings of LREC.

- Also see Multimodal Semantic Representations workshop

Semantics of Predicate Logic

- **Exercise 5.18** Translate the following sentences into predicate logic, making sure that their truth conditions are captured.
  - Someone walks and someone talks.
  - No wizard cast a spell or mixed a potion.
  - Every balad that is sung by a princess is beautiful.
  - If a knight finds a dragon, he fights it.

Semantics of Predicate Logic

- **Exercise 5.18** Translate the following sentences into predicate logic, making sure that their truth conditions are captured.
  - Someone walks and someone talks.
    - $\exists x(\text{Person}(x) \wedge \text{Walk}(x)) \wedge \exists y(\text{Person}(y) \wedge \text{Talk}(y))$
  - No wizard cast a spell or mixed a potion.
    - $\neg\exists x(\text{Wizard}(x) \wedge (\exists y(\text{Spell}(y) \wedge \text{Cast}(x, y)) \vee \exists z(\text{Potion}(z) \wedge \text{Mix}(x, z))))$
  - Every balad that is sung by a princess is beautiful.
    - $\forall x((\text{Ballad}(x) \wedge \exists y(\text{Princess}(y) \wedge \text{Sing}(y, x))) \rightarrow \text{Beautiful}(x))$
  - If a knight finds a dragon, he fights it.
    - $\forall x \forall y((\text{Knight}(x) \wedge \text{Dragon}(y) \wedge \text{Find}(x, y)) \rightarrow \text{Fight}(x, y))$

Computational Semantics
Day 2: Meaning representations and (predicate) logic

Jan van Eijck[1] & Christina Unger[2]

[1]CWI, Amsterdam, and UiL-OTS, Utrecht, The Netherlands
[2]CITEC, Bielefeld University, Germany

ESSLLI 2011, Ljubljana

Type definitions

General form:

$$\texttt{data}\ \text{type\_name}\ (\text{type\_parameters}) = \text{constructor}_1\ t_{11} \dots t_{1i} \mid \text{constructor}_2\ t_{21} \dots t_{2j} \mid \dots \mid \text{constructor}_n\ t_{n1} \dots t_{nk}$$

This can be used to create:

- enumeration types
- composite types
- recursive types
- parametric types

Example: Enumeration types

$$\text{data type\_name (type\_parameters)} = \text{constructor}_1\ t_{11} \ldots t_{1i} \mid \text{constructor}_2\ t_{21} \ldots t_{2j} \mid \ldots \mid \text{constructor}_n\ t_{n1} \ldots t_{nk}$$

Examples:

```
module Day2 where
--data Bool = True | False

data Season = Spring | Summer | Autumn | Winter

data Temperature = Hot | Cold | Moderate
```

Example: Enumeration types

Now, we can define a function using objects of type `Season` and `Temperature`.

```
weather :: Season -> Temperature
weather Summer = Hot
weather Winter = Cold
weather _      = Moderate
```

Example: Enumeration types

Now, we can define a function using objects of type Season and Temperature.

```
weather :: Season -> Temperature
weather Summer = Hot
weather Winter = Cold
weather _      = Moderate
```

But user-defined types do not automatically have operators for equality, ordering, show, etc.

```
> weather Spring
No instance for (Show Temperature)
arising from a use of ‘print’ at <interactive>:1:0-13
```

Instance declarations for `Show`

In order to display user-defined types, we can either define the function `show :: Typename -> String` explicitly ...

```
instance Show Season where
  show Spring = "Spring"
  show Summer = "Summer"
  show Autumn = "Autumn"
  show Winter = "Winter"
```

Instance declarations for Show

In order to display user-defined types, we can either define the function `show :: Typename -> String` explicitely ...

```
instance Show Season where
  show Spring = "Spring"
  show Summer = "Summer"
  show Autumn = "Autumn"
  show Winter = "Winter"
```

... or derive it.

```
data Season = Spring | Summer | Autumn | Winter
              deriving Show
```

Example: Composite types

$$\text{data type\_name (type\_parameters)} = \text{constructor}_1\ t_{11} \dots t_{1i} \mid \text{constructor}_2\ t_{21} \dots t_{2j} \mid \dots \mid \text{constructor}_n\ t_{n1} \dots t_{nk}$$

Examples:

```
data Book = Book Int String [String]

data Color = White | Black | RGB Int Int Int
```

Example: Recursive types

$$\begin{aligned} \texttt{data}\ \text{type\_name}\ (\text{type\_parameters}) = {} & \text{constructor}_1\ t_{11} \dots t_{1i} \\ | {} & \text{constructor}_2\ t_{21} \dots t_{2j} \\ | {} & \dots \\ | {} & \text{constructor}_n\ t_{n1} \dots t_{nk} \end{aligned}$$

Example:

```
data Tree = Leaf | Branch Tree Tree
```

Example: Polymorphic types

$$\text{data type\_name (type\_parameters)} = \text{constructor}_1\ t_{11} \dots t_{1i} \mid \text{constructor}_2\ t_{21} \dots t_{2j} \mid \dots \mid \text{constructor}_n\ t_{n1} \dots t_{nk}$$

Examples:

```
data Maybe a = Nothing | Just a

data List a = Nil | Cons a (List a)

data Tree a = Leaf a | Branch (Tree a) (Tree a)
```

Summary of 9/14 Discussion

Things in model	Expression	Type
relations	verbs	`String`
entities	nouns	`String`
?	adjectives	`String`
truth values	sentences	`String`

Summary of 9/14 Discussion

Things in model	Expression	Type
relations	verbs	`String`
entities	nouns	`String`
?	adjectives	`String`
truth values	sentences	`String`

- How to represent a model in Haskell?

Summary of 9/14 Discussion

Things in model	Expression	Type
relations	verbs	`String`
entities	nouns	`String`
?	adjectives	`String`
truth values	sentences	`String`

- How to represent a model in Haskell?
- Truth values (`True`, `False`) are objects of type `Bool`

A Model of a Fragment of English

- Declare a data type `Entity`

```
data Entity = A | B | C | D | E | F | G
            | H | I | J | K | L | M | N
            | O | P | Q | R | S | T | U
            | V | W | X | Y | Z | Unspec
     deriving (Eq,Show,Bounded,Enum)
```

A Model of a Fragment of English

- Declare a data type `Entity`

```
data Entity = A | B | C | D | E | F | G
            | H | I | J | K | L | M | N
            | O | P | Q | R | S | T | U
            | V | W | X | Y | Z | Unspec
     deriving (Eq,Show,Bounded,Enum)
```

- We can put all of our entities in a list

```
entities :: [Entity]
entities =  [minBound..maxBound]
```

A Model of a Fragment of English

- Proper names are interpreted as entities

```
snowWhite, alice, dorothy, goldilocks, littleMook, atreyu
                                                   :: Entity

snowWhite  = S
alice      = A
dorothy    = D
goldilocks = G
littleMook = M
atreyu     = Y
```

A Model of a Fragment of English

- Proper names are interpreted as entities

```
snowWhite, alice, dorothy, goldilocks, littleMook, atreyu
                                                 :: Entity

snowWhite  = S
alice      = A
dorothy    = D
goldilocks = G
littleMook = M
atreyu     = Y
```

- Not all nouns are interpreted as entities, though
  - Common nouns such as *girl* and *dwarf* are more like sets of entities, or properties of entities (unary relations)