english: the lightest weight programming language of them all

hugo liu & henry lieberman
mit media laboratory

simplicity
at the mit media laboratory in cambridge, massachusetts

lightweight languages 2004
programming is storytelling

- every program tells a story
  - objects ~ characters
  - behaviours ~ personality
- traditionally expressed in programming languages
  - easy for computers
  - difficult for people to read, understand, and author fluently


simplicity
courtesy: Pane et alii, 2001

at the mit media laboratory in cambridge, massachusetts
talk overview

metafor: visualising stories as code

a theory of programmatic semantics for NL

“common sense” knowledge for the interpreter

implementation

simplicity
at the mit media laboratory in cambridge, massachusetts
metafor: visualising stories as code

user enters story here

history of dialogue with system agent, who explains what was understood

under-the-hood debug window

story rendered as code (in Python as shown)
metafor demo

click here
putting metafor in context

- **scope & limitations**
  - only generates non-executable ‘scaffolding code’
  - cannot convert arbitrary English into fully specified code
  - however, broad coverage sufficient for brainstorming, program “outlining”
- **related work**
  - machine translation and interlingua
  - pseudo-nl domain languages
    - nl interface to MOOs (Bruckman, 1997)
    - Natural Language SQL interfaces, e.g. MS-SQL
  - case tools for UML requirements engineering
    - exploits structure of requirements documents
    - keyword-parse into flowchart (Hars & Marchewka, 1996)
    - grammar-based parsers: NL-OOPS (Mich, 1996); (Lee & Bryant, 2002)
  - user-supervised outlining: UTEL (Tam et alii, 1998)
a theory of programmatic semantics for natural language

• **natural language has an inherent programmatic regularity**
  – resembles object-oriented and agent-oriented programming
  – relies heavily on prototyping, and common sense knowledge

• **to oversimplify…**
  – noun phrases $\leftrightarrow$ objects  
    - *e.g.* “the martini”
  – verbs $\leftrightarrow$ functions  
    - *e.g.* “make a drink”
  – adjectives $\leftrightarrow$ properties  
    - *e.g.* “sweet drinks”
  – adverbials $\leftrightarrow$ parameters  
    - *e.g.* “quickly make a drink”

*simplicity*

at the mit media laboratory in cambridge, massachusetts
further basic programmatic features

• verb-arg structure $\leftrightarrow$ function-arg structure
  – *e.g.* “give the drink to the customer”
• conventions for prototype $\leftrightarrow$ inheritance
  – *e.g.* “a martini is a drink which …”
• attachment semantics $\leftrightarrow$ an object’s parts
  – *e.g.* “the customer’s age” $\leftrightarrow$ customer.age
  – *e.g.* “a bar with a bartender”
  – *e.g.* “some stools in the bar”
  – *e.g.* “the bar has some customers”
scoping

• conditionals
  – subjunctive constructions
    • If the drink is on the menu, then make it
    • Should the customer not ordered, the bartender would not have made the drink
    • In the case that the drink is expensive, he won’t order it.
  – implied
    • The customer may order a sweet drink (auxiliary)
    • Sometimes he orders a sweet drink and ... (set theoretic)

• when
  – when the drink is sweet, order it. (topical object)
  – when the customer orders it, the bartender makes it. (topical agent action)
set-theoretic features

- tendency not to express loop structure (cf. pane et alii, 2001)
- *dynamic* reference
  - The customer *buys* some of the sweet drinks under $2.

```
map(customer.buy,
    filter(lambda sdu2: some(sdu2),
    filter(lambda sweet_drink: sweet_drink.price < 2,
    filter(lambda drink: 'sweet' in drink.properties,
    menu.drinks))))
```

- set-theoretic semantics
  - comparatives/superlatives (“the cheaper/cheapest drink”);
  - subsets (e.g. “*all* drinks have,” “*some* drinks ..while *others*…”)
  - complementizer $\leftrightarrow$ procedural attachment
    - e.g. “the drink *which Bill would like the best*”

**simplicity**

at the mit media laboratory in cambridge, massachusetts
representational dynamism

• **for nl, underlying representation is fluid**
  
a) There is a bar. (atom)
b) The bar contains two customers. (unimorphic list)
c) It also contains a waiter. (unimorphic wrt. persons)
d) It also contains some stools. (polymorphic list)
e) The bar opens and closes. (class / agent)
f) The bar is a kind of store. (inheritance class)
g) Some bars close at 6pm. (subclass or instantiatable)

• **nominalization (i.e. casting an adjective as a noun)**
  
The drink is sweet.
The drink has sweetness.
dynamic refactoring

• ambiguity never killed anybody!
  – conventional programming often forces a programmer to make inessential decisions about representation details far too early in the design and programming process

• sour apple martini
  ➔ class sour_apple_martini

• sour apple martini, sweet apple martini, sour grape martini
  ➔ class martini:
    def __init__(self, flavor='sour', fruit='apple'):
      self.flavor, self.fruit = flavor, fruit

simplicity
at the mit media laboratory in cambridge, massachusetts
metafor’s generic functions
(full_name, arguments, body)

• cf. generic functions in CLOS
• dynamic type inspector is heuristic
  – e.g., body contains two similarly typed elements $\rightarrow$ listType
  – e.g., body contains functions $\rightarrow$ classType
  – propagates symmetry in peer objects
    • apple has color, therefore, strawberry has color
  – predefined functions for flow control statements
• inspector assumes simplicity
  – adds complexity only as necessary
  – irresolvable representational conflicts formulated as question and fed back to user via dialog
  – uses referential cues (anaphoric reference, appositives) to aid in disambiguation

simplicity
at the mit media laboratory in cambridge, massachusetts
narrative stance equivalences

• bar has part customer
  a) I want to make a bar with a customer. (1st p. programmer)
  b) There is a customer in the bar. (3rd p. narrator)
  c) I am a customer sitting on a stool. (1st p. customer)
  d) The bartender said, “Here is a customer” (mixed p. playwright)
prototypes & background semantics

• **thought is inherently metaphorical** (Lakoff & Johnson, 1980)
  - e.g. system for “time” is partially structured by “money”
  - e.g. “academic repartee” structured by “war”
  - Narayanan (1997) maps linguistic metaphors to schemas

• **personification:** *partial inheritance from person prototype*

• **to complicate things,**
  - in NL, not just object inheritance, but also system inheritance

• **what are some background semantics KBS?**
  - Cyc (Lenat, 1980)
  - ConceptNet (Liu & Singh, 2004)
conceptNet: a source of background semantics

- semantic network with 300,000 nl nodes, 1.6 million edges
- contains defeasible world knowledge
  - e.g.
    - “kicking someone causes pain”
    - “a lemon is sour”
- some mappings to programmatic knowledge
  - $\text{CapableOf}(x, y) \rightarrow x.y()$
  - $\text{LocationOf}(x, y) \rightarrow y.x$
  - $\text{PropertyOf}(x, y) \rightarrow x.y$
  - $\text{PartOf}(x, y) \rightarrow x.y$
  - $\text{IsA}(x, y) \rightarrow \text{class } x(y)$
  - $\text{EffectOf}(w.x, y.z) \rightarrow w.x() : y.z$
heuristic type inference with conceptnet

** Click here **

Welcome to the ConceptNet v2 mini-browser!
(for more info, please visit www.conceptnet.org)

The purpose of this browser is to allow you to explore the ConceptNet API interactively!
other advanced features

• declaration-execution equivalence
  – e.g. “there are some sweet drinks”; “buy some sweet drinks”)
  – e.g. “the bartender makes the drinks”; “when … the bartender makes the drinks”)

• anaphora / deixis (e.g. “he”, “this”, “here”)

• lazy evaluation (e.g. “the cheapest drink”)

simplicity
at the mit media laboratory in cambridge, massachusetts
implementation basics

• we parsed the input text into syntactic frames
  – using the MontyLingua NLP package (pypi)

• semantic recognition agents
  – conceptNet: recognition of default semantic types
    • e.g.: ‘bins’ are likely containers
  – wordNet (Fellbaum, 1998): sets of objects
    • e.g.: colors: red, orange, yellow, green...

• programmatic interpreter
  – resolve textual references to existing objects
  – handle special structures
    • e.g. scoping statements, lists, quotes, flow control
  – map VSOO structures to some action or change
  – update deictic discourse stack, scope, and interpretive context (i.e. declarative versus procedural)
vaporware

- exploit the richness of verb-argument structure
  - FrameNet (Fillmore, 1968)
  - Levin’s verb classes & alternations (1993)
- accounting for the implied behaviour of verbs
  - “the effect of x giving something to y is that y receives it”
- refine the scaffolding code
  - meaning negotiation through dialogue
  - guide interaction with a programming “plan”,
    - a la programmer’s apprentice (Rich & Waters, 1990)
brainstorming & outlining with metafor

• **metafor makes user accountable for the consequences of their language**
  – exposes implied knowledge / knowledge gaps
  – exposes metaphorical structure of thought
    • e.g. “there is a way for the bartender to…”

• **as a constructionist educational tool**
  – hypothesis: *precise storytelling is a requisite for good programming*
  – a programming “tutor” for novices proficient in *reading* but not *writing* code

*simplicity*

at the MIT Media Laboratory in Cambridge, Massachusetts
readings

