english: the lightest weight programming language of them *all*

hugo liu & henry lieberman

mit media laboratory

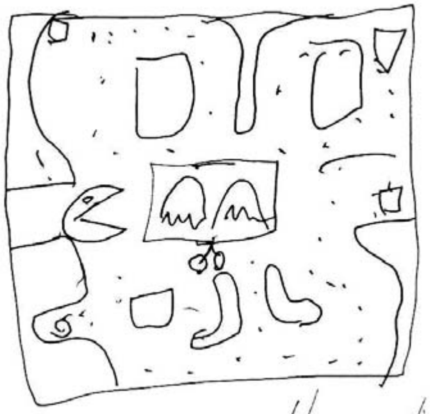
lightweight languages 2004

simplicity

at the mit media laboratory in cambridge, massachusetts

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*



The yellow dots
are set through a maze
and Pac-man, The chomping
yellow circle has to collect them
There are blinking big yellow
dots that allow you to eat
the gosts ( ).

a non-programmer's description of Pacman
(courtesy: Pane *et alii,* 2001)

talk overview

meta*for*: visualising stories as code

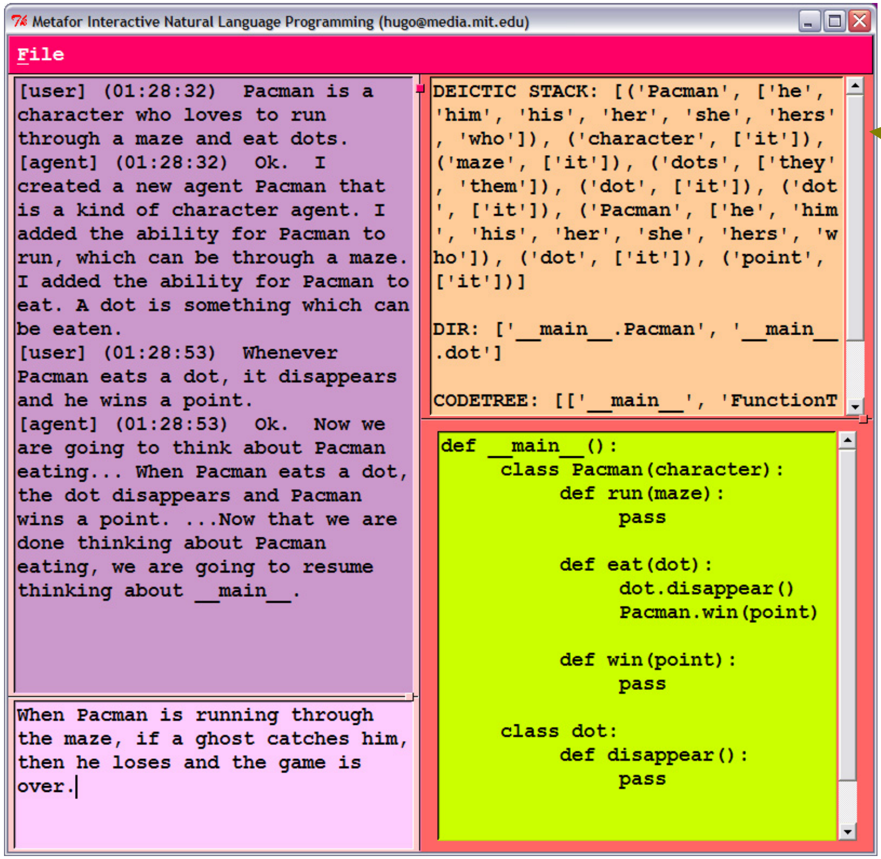
a theory of programmatic semantics for NL

“common sense” knowledge for the interpreter

implementation

meta*for*: visualising stories as code

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



Metafor Interactive Natural Language Programming (hugo@media.mit.edu)
File

[user] (01:28:32) Pacman is a character who loves to run through a maze and eat dots.
[agent] (01:28:32) Ok. I created a new agent Pacman that is a kind of character agent. I added the ability for Pacman to run, which can be through a maze. I added the ability for Pacman to eat. A dot is something which can be eaten.
[user] (01:28:53) Whenever Pacman eats a dot, it disappears and he wins a point.
[agent] (01:28:53) Ok. Now we are going to think about Pacman eating... When Pacman eats a dot, the dot disappears and Pacman wins a point. ...Now that we are done thinking about Pacman eating, we are going to resume thinking about __main__.

When Pacman is running through the maze, if a ghost catches him, then he loses and the game is over.

```
DEICTIC STACK: [('Pacman', ['he',
'him', 'his', 'her', 'she', 'hers'
, 'who']), ('character', ['it']),
('maze', ['it']), ('dots', ['they'
, 'them']), ('dot', ['it']), ('dot
', ['it']), ('Pacman', ['he', 'him
', 'his', 'her', 'she', 'hers', 'w
ho']), ('dot', ['it']), ('point',
['it'])]

DIR: ['__main__.Pacman', '__main__
.dot']

CODETREE: [['__main__', 'FunctionT
```

```
def __main__():
    class Pacman(character):
        def run(maze):
            pass

        def eat(dot):
            dot.disappear()
            Pacman.win(point)

        def win(point):
            pass

    class dot:
        def disappear():
            pass
```

under-the-hood debug window

story rendered as code

(in Python as shown)

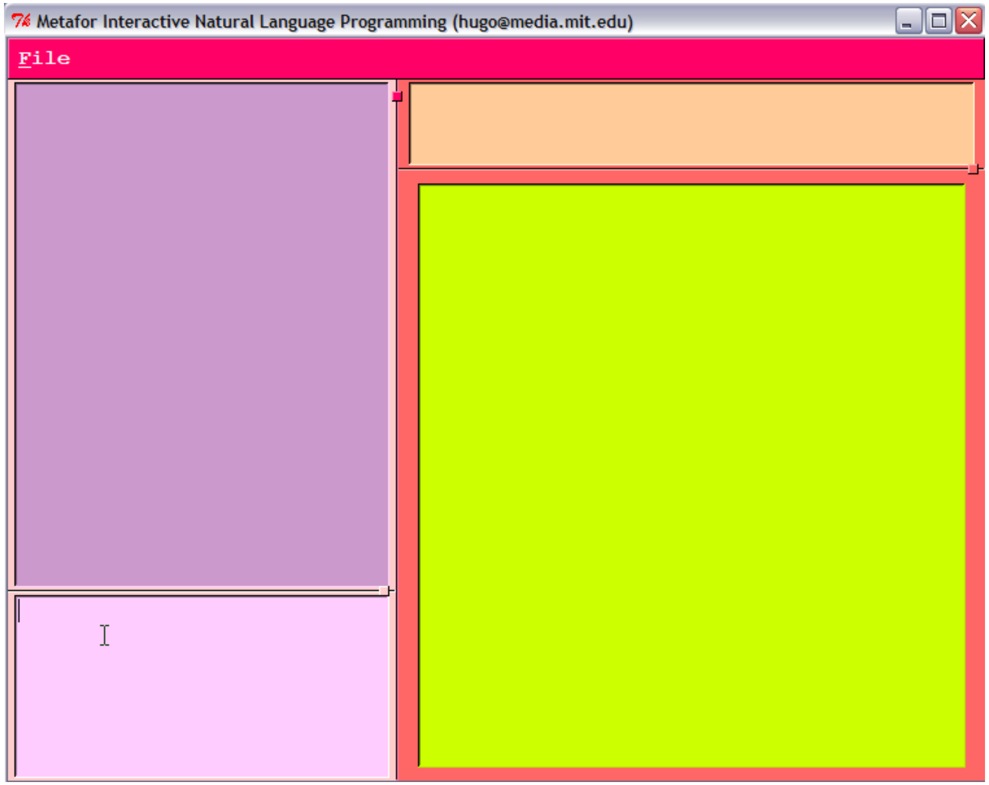
user enters story here

simplicity

at the mit media laboratory in cambridge, massachusetts

metafor demo

click here



Metafor Interactive Natural Language Programming (hugo@media.mit.edu)
File

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 ←→ objects
    - *e.g.* “**the martini**”
  - verbs ←→ functions
    - *e.g.* “**make a drink**”
  - adjectives ←→ properties
    - *e.g.* “**sweet drinks**”
  - adverbials ←→ parameters
    - *e.g.* “**quickly make a drink**”

further basic programmatic features

- verb-arg structure ←→ function-arg structure
  - *e.g.* “**give the drink to the customer**”
- conventions for prototype ←→ inheritance
  - *e.g.* “**a martini is *a drink* which …**”
- attachment semantics ←→ an object’s parts
  - *e.g.* “**the customer’s age**” ←→ **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 ←→ procedural attachment
    - *e.g.* “**the drink** ***which Bill would like the best***”

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
```

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 → listType
  - e.g., body contains functions → 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

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)

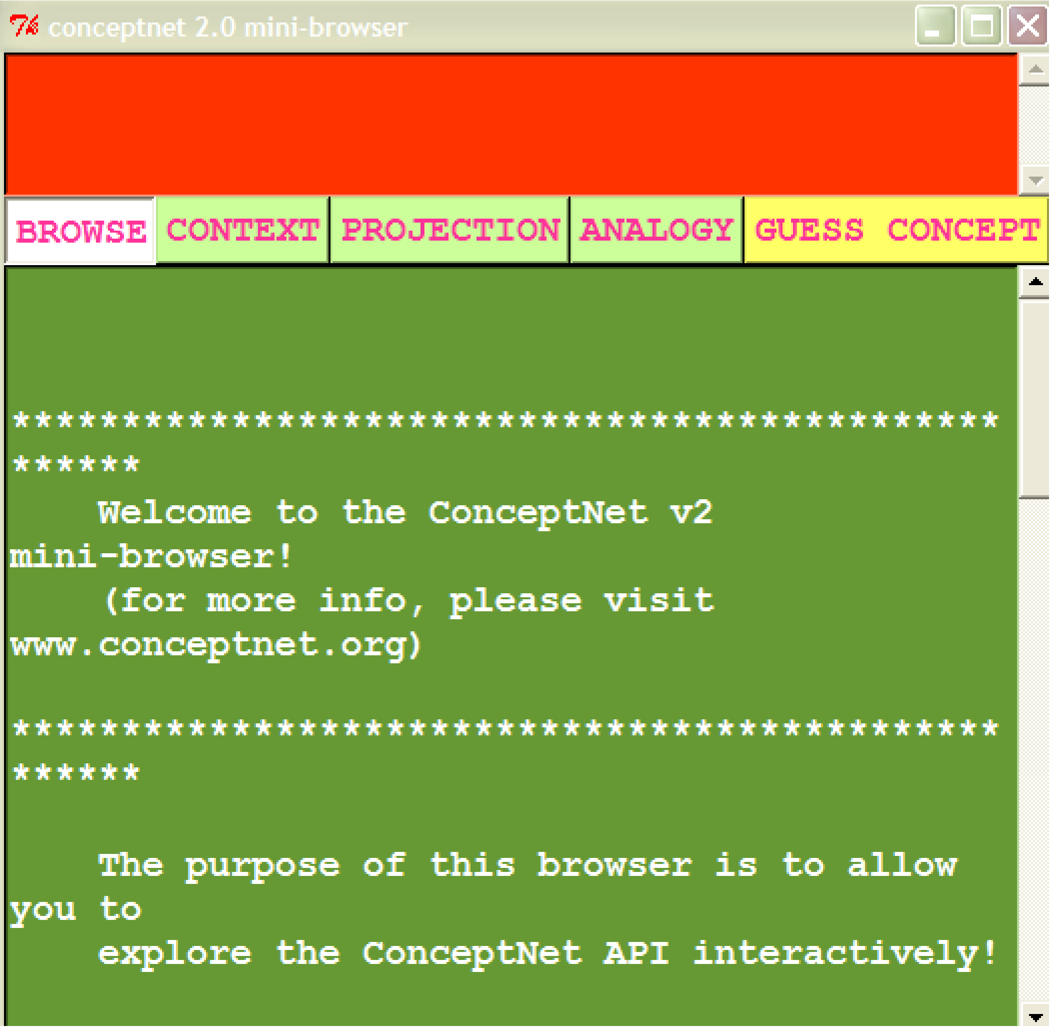
conceptNet: a source of background semantics

- **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**

```
CapableOf(x,y)  → x.y()
LocationOf(x,y)  → y.x
PropertyOf(x,y)  → x.y
PartOf(x,y)  → x.y
IsA(x,y)  → class x(y)
EffectOf(w.x,y.z)  → 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”)

implementation basics

- **we parsed the input text into syntactic frames**
  - **{verb: ‘parse’, subj: ‘us’, obj:’input text’, obj2: ‘into syntactic frame’}**
  - 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

readings

- Hugo Liu and Henry Lieberman: 2005, Metafor: Visualizing Stories as Code. Proceedings of the 2005 ACM International Conference on Intelligent User Interfaces, to appear
- Hugo Liu and Henry Lieberman: 2004b, Toward a Programmatic Semantics of Natural Language. Proceedings of VL/HCC'04: the 20th IEEE Symposium on Visual Languages and Human-Centric Computing. pp. 281-282. September 26-29, 2004, Rome. IEEE Computer Society Press.
- Henry Lieberman and Hugo Liu: 2004a, Feasibility Studies for Programming in Natural Language. H. Lieberman, F. Paterno, and V. Wulf (Eds.) Perspectives in End-User Development, to appear. Kluwer.