Logic Programming
Lists, Sets, and Trees

Michael Genesereth
Computer Science Department
Stanford University

Programme

General Examples

- Lists
- Sorted Lists
- Sets

Application

- Natural Language Processing

Predefined Functions

- List Functions
- List Aggregate
- Conversion Functions

Linked Lists

Linked Lists

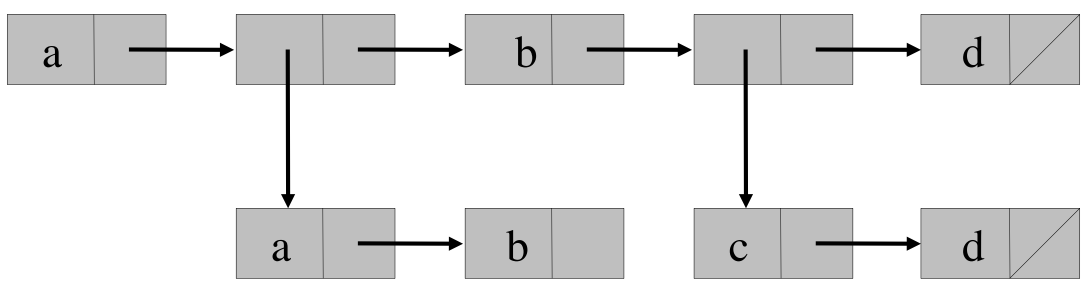
Flat Lists

$$[a, b, c, d]$$

Nested Lists

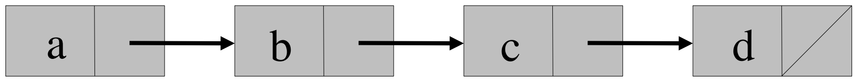
$$[a, [a, b], b, [c, d], d]$$

Linked List



Representation

Example



Representation as Term

```
cons(a,cons(b,cons(c,cons(d,nil))))
```

Syntactic Sugar

```
a!b!c!d!nil
```

```
[a,b,c,d]
```

Vocabulary

Symbols: `a`, `b`, `c`, `d`, ..., `nil`
Binary Constructor: `cons`
Binary Predicate: `mem`, `among`
Ternary Predicate: `append`, `repend`

Membership

Example:

```
mem(b, cons(a,cons(b,cons(c,nil))) )
mem(b, (a!(b!(c!nil))) )
mem(b, [a,b,c] )
```

Definition :

```
mem(X,cons(X,Y))
mem(X,cons(Y,Z)) :- mem(X,Z)
```

Unsafe Rule!!

```
mem(X,X!Y)
mem(X,Y!Z) :- mem(X,Z)
```

Unsafe Rule!!

Globally Safe?

Containment

Example:

```
among(b,cons(a,cons(cons(b,nil),cons(d,nil))))
among(b,[a,[b],d])
```

Definition:

```
among(X,X)
among(X,cons(Y,Z)) :- among(X,Y)
among(X,cons(Y,Z)) :- among(X,Z)

among(X,X)
among(X,Y!Z) :- among(X,Y)
among(X,Y!Z) :- among(X,Z)
```

Concatenation

Example:

```
app(cons(a,cons(b,nil)),
    cons(c,cons(d,nil)),
    cons(a,cons(b,cons(c,cons(d,nil)))))
app([a,b],[c,d],[a,b,c,d])
```

Definition :

```
app(nil,Y,Y)
app(cons(X,Y),Z,cons(X,W)) :- app(Y,Z,W)

app(nil,Y,Y)
app(X!Y,Z,X!W) :- app(Y,Z,W)
```

Reverse Concatenation

Example:

```
repend(cons(a,cons(b,nil)),
       cons(c,cons(d,nil)),
       cons(b,cons(a,cons(c,cons(d,nil)))))
repend([a,b],[c,d],[b,a,c,d])
```

Definition :

```
repend(nil,L,L)
repend(cons(X,L),M,N) :- repend(L,cons(X,M),N)

repend(nil,L,L)
repend(X!L,M,N) :- repend(L,X!M,N)
```

mem in Sierra

Try:

```
mem(b,[a,b,c,d])
mem(X,[a,b,c,d])
mem(b,L)
```
(find 5)

append in Sierra

Try:

```
app([a,b],[c,d],L)
app([a,b],L,[a,b,c,d])
app(L,M,[a,b,c,d])
```

Sorted Lists

Sorted Lists

Unsorted List

[1,3,2]

Sorted List

[1,2,3]

Multiple Occurrences

[1,2,2,3]

Vocabulary

Symbols: `1`, `2`, `3`, `4`, ..., `nil`
Binary Constructor: `cons`
Unary Predicate: `sorted`
Binary Predicate: `leq`
Ternary Predicate: `insert`, `merge`

Sorted Lists

```
sorted(nil)
sorted([X])
sorted(cons(X,cons(Y,L))) :-
  leq(X,Y) & sorted(cons(Y,L))
```

```
sorted(nil)
sorted([X])
sorted(X!Y!L) :- leq(X,Y) & sorted(Y!L)
```

Concatenation (Version 1)

```
merge(X,Y,Z) :- append(X,Y,W) & sort(W,Z)
```

Concatenation (Version 2)

```
insert(X,nil,[X])
insert(X,Y!L,X!Y!L) :- leq(X,Y)
insert(X,Y!L,Y!M) :- ~leq(X,Y) & insert(X,L,M)
```

```
merge(nil,Y,Y)
merge(X!L,Y,Z) :- merge(L,Y,W) & insert(X,W,Z)
```

Sets

Sorted Lists

Set

`{4,1,3,2}`

Order does not matter

`{4,1,3,2} = {2,3,1,4} = {1,2,3,4}`

Multiple occurrences do not matter

`{1,2,2,3,3,4} = {1,2,3,4}`

Representation of Sets as Sorted Lists

`[1,2,3,4]`

Vocabulary

Symbols: `1`, `2`, `3`, `4`, ..., `nil`
Binary Constructor: `cons`
Binary Predicate: `less`, `mem`, `subset`
Ternary Predicate: `intersection`, `union`

Membership

Version 1:

```
mem(X,X!L)
mem(X,Y!L) :- mem(X,L)
```

Version 2:

```
mem(X,X!L)
mem(X,Y!L) :- less(Y,X) & mem(X,L)
```

Subsets

```
subset(nil,Y)
subset(X!L,Y) :- mem(X,Y) & subset(L,Y)
```

Intersection

```
intersection(nil,Y,nil)

intersection(X!L,M,X!N) :-
  mem(X,M) & intersection(L,M,N)

intersection(X!L,M,N) :-
  ~mem(X,M) & intersection(L,M,N)
```

Union

```
union(nil,Y,Y)

union(X!L,M,N) :-
  mem(X,M) & union(L,M,N)

union(X!L,M,X!N) :-
  ~mem(X,M) & union(L,M,N)
```

Union

```
union(nil,Y,Y)

union(X!L,M,N) :-
  mem(X,M) & union(L,M,N)

union(X!L,M,X!N) :-
  ~mem(X,M) & union(L,M,N)
```

Natural Language Processing

Pseudo English

Good Sentences:

- Mary likes Pat.
- Mary likes Pat and Quincy.
- Pat and Quincy like Mary.

Bad Sentences:

- Mary Pat likes.
- Likes and Mary Pat Quincy.

Backus Naur Form (BNF)

```
<sentence> ::= <np> <vp>
<np> ::= <noun>
<np> ::= <noun> "and" <noun>
<vp> ::= <verb> <np>
<noun> ::= "mary" | "pat" | "quincy"
<verb> ::= "like" | "likes"
```

Internal Representation

English sentence:

Mary likes Pat and Quincy.

Our representation:

```
[mary, likes, pat, and, quincy]
```

Logical Grammar

```
sentence(Z) :- append(X,Y,Z) & np(X) & vp(Y)
```

Logical Grammar

```
sentence(Z) :- append(X,Y,Z) & np(X) & vp(Y)

np([X]) :- noun(X)
np(X!and!Y) :- noun(X) & np(Y)
```

Logical Grammar

```
sentence(Z) :- append(X,Y,Z) & np(X) & vp(Y)

np([X]) :- noun(X)
np(X!and!Y) :- noun(X) & np(Y)

vp(X!Y) :- verb(X) & np(Y)
```

Logical Grammar

```
sentence(Z) :- append(X,Y,Z) & np(X) & vp(Y)

np([X]) :- noun(X)
np(X!and!Y) :- noun(X) & np(Y)

vp(X!Y) :- verb(X) & np(Y)

noun(mary)
noun(pat)
noun(quincy)
```

Logical Grammar

```
sentence(Z) :- append(X,Y,Z) & np(X) & vp(Y)

np([X]) :- noun(X)
np(X!and!Y) :- noun(X) & np(Y)

vp(X!Y) :- verb(X) & np(Y)

noun(mary)
noun(pat)
noun(quincy)

verb(like)
verb(likes)
```

Examples

Sentences:

√ Mary likes Pat.

√ Mary likes Pat and Quincy.

√ Pat and Quincy like Mary.

Not Sentences:

× Mary Pat likes.

× Likes and Mary Pat Quincy.

Glitch

Sentences:

Mary likes Pat.

Mary likes Pat and Quincy.

Pat and Quincy like Mary.

Allowed but not sentences in natural English:

Mary like Pat.

Pat and Quincy likes Mary.

How can we enforce subject-verb number agreement?

Augmented Logical Grammar

```
sentence(Z) :- append(X,Y,Z) & np(X,N) & vp(Y,N)

np([X],0) :- noun(X)
np(X!and!Y,1) :- noun(X) & np(Y,N)

vp(X!Y,M) :- verb(X,M) & np(Y,N)

noun(mary)
noun(pat)
noun(quincy)

verb(like,1)
verb(likes,0)
```

Augmented Logical Grammar

```
sentence(Z) :- append(X,Y,Z) & np(X,N) & vp(Y,N)

np([X],0) :- noun(X)
np(X!and!Y,1) :- noun(X) & np(Y,N)

vp(X!Y,M) :- verb(X,M) & np(Y,N)

noun(mary)
noun(pat)
noun(quincy)

verb(like,1)
verb(likes,0)
```

Augmented Logical Grammar

```
sentence(Z) :- append(X,Y,Z) & np(X,N) & vp(Y,N)

np([X],0) :- noun(X)
np(X!and!Y,1) :- noun(X) & np(Y,N)

vp(X!Y,M) :- verb(X,M) & np(Y,N)

noun(mary)
noun(pat)
noun(quincy)

verb(like,1)
verb(likes,0)
```

Augmented Logical Grammar

```
sentence(Z) :- append(X,Y,Z) & np(X,N) & vp(Y,N)

np([X],0) :- noun(X)
np(X!and!Y,1) :- noun(X) & np(Y,N)

vp(X!Y,M) :- verb(X,M) & np(Y,N)

noun(mary)
noun(pat)
noun(quincy)

verb(like,1)
verb(likes,0)
```

Augmented Logical Grammar

```
sentence(Z) :- append(X,Y,Z) & np(X,N) & vp(Y,N)

np([X],0) :- noun(X)
np(X!and!Y,1) :- noun(X) & np(Y,N)

vp(X!Y,M) :- verb(X,M) & np(Y,N)

noun(mary)
noun(pat)
noun(quincy)

verb(like,1)
verb(likes,0)
```

List-Oriented Builtins

List Functions

```
evaluate(length([1,2,3]),3)
evaluate(minimum([1,2,3]),1)
evaluate(maximum([1,2,3]),3)
evaluate(sum([1,2,3]),6)
evaluate(mean([1,2,3]),2)

evaluate(reverse([a,b,c]),[c,b,a])
evaluate(append([a,b],[c]),[a,b,c])
evaluate(revappend([a,b],[c]),[b,a,c])
```

Aggregates

Dataset:

```
p(a,1)
p(a,2)
p(a,3)
```

Examples:

```
evaluate(setofall(Y,p(a,Y)),[1,2,3])
evaluate(length(setofall(Y,p(a,Y))),3)
evaluate(sum(setofall(Y,p(a,Y))),6)
```

Sundry

Expressions:

```
evaluate(listify(p(a,b)),[p,a,b])
evaluate(delistify([p,a,b]),p(a,b))
```

Matching:

```
evaluate(submatches("321-1245",".2."),["321","124"])
evaluate(matches("321-1245","(.)-(.)"),["1-1","1","1"])
```

Strings:

```
evaluate(readstring("p(a,b)"),p(a,b))
evaluate(stringify(p(a,b)),"p(a,b)")
```