Inventory Example

Bicycle Factory

Inventory of Bicycle Parts

Hierarchial Structure

Bicycle made up of parts

Each part made up of sub-parts

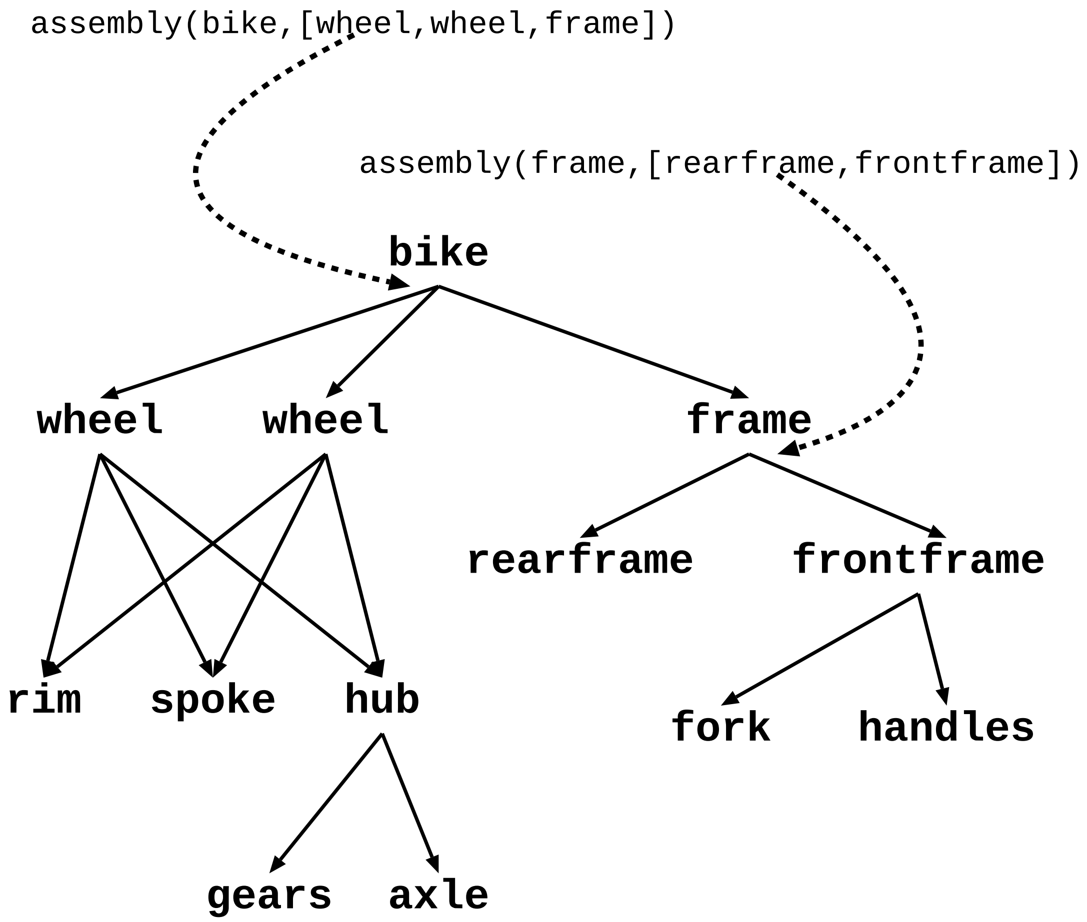
Basic Parts

```
basicpart(rim).
basicpart(spoke).
basicpart(rearframe).
basicpart(handles).
basicpart(gears).
basicpart(bolt).
basicpart(nut).
basicpart(fork).
```

Assemble Parts

```
assembly(bike,[wheel,wheel,frame]).
assembly(wheel,[spoke,rim,hub]).
assembly(frame,[rearframe,frontframe]).
assembly(frontframe,[fork,handles]).
assembly(hub,[gears,axle]).
assembly(axle,[bolt,nut]).
```

Assembly



[rim,spoke,gears,axle,rearframe,fork,handles]

Parts Of

```
partsof(X,[X]) :- basicpart(X).

partsof(X,P) :-
        assembly(X,Subparts),
        partsoflist(Subparts,P).

partsoflist([],[]).
partsoflist([P|Tail],Total) :-
     partsof(P,Headparts),
     partsoflist(Tail,Tailparts),
     append(Headparts,Tailparts,Total).
```

Test

```
+ 1  1  Call: partsof(bike,_67) ?
+ 2  2  Call: basicpart(bike) ?
+ 2  2  Fail: basicpart(bike) ?
+ 2  2  Call: assembly(bike,_326) ?
+ 2  2  Exit: assembly(bike,[wheel,wheel,frame]) ?
+ 3  2  Call: partsoflist([wheel,wheel,frame],_67) ?
+ 4  3  Call: partsof(wheel,_893) ?
+ 5  4  Call: basicpart(wheel) ?
+ 5  4  Fail: basicpart(wheel) ?
+ 5  4  Call: assembly(wheel,_1095) ?
+ 5  4  Exit: assembly(wheel,[spoke,rim,hub]) ?
+ 6  4  Call: partsoflist([spoke,rim,hub],_893) ?
+ 7  5  Call: partsof(spoke,_1662) ?
+ 8  6  Call: basicpart(spoke) ?
+ 8  6  Exit: basicpart(spoke) ?
+ 7  5  Exit: partsof(spoke,[spoke]) ?
+ 9  5  Call: partsoflist([rim,hub],_1656) ?
```

Test

```
+ 32  10  Call: partsoflist([],_5146) ?
+ 32  10  Exit: partsoflist([],[]) ?
+ 33  10  Call: append([bolt,nut],[],_4238) ?
+ 33  10  Exit: append([bolt,nut],[],[bolt,nut]) ?
+ 18  9  Exit: partsoflist([axle],[bolt,nut]) ?
+ 36  9  Call: append([gears],[bolt,nut],_3479) ?
+ 37  10  Call: append([],[bolt,nut],_11574) ?
+ 37  10  Exit: append([],[bolt,nut],[bolt,nut]) ?
+ 36  9  Exit: append([gears],[bolt,nut],
                    [gears,bolt,nut]) ?

X = [spoke,rim,gears,bolt,nut,spoke,rim,
     gears,bolt,nut,rearframe,fork,handles] ?
```

Sentences

Another Hierarchial Structure

Decomposition of Sentences

Sentence

noun_phrase

verb_phrase

noun_phrase

determiner

noun

.

.

.

Sentence Assembly

```
assembly(sentence,[noun_phrase,
                   verb_phrase]).
assembly(noun_phrase,
         [determiner,noun]).
assembly((determiner,[the]).
assembly(noun,[clergyman]).
assembly(noun,[motorcar]).

basicpart(clergyman).
basicpart(motorcar).
```

Accumulators

Program

Traverse a PROLOG structure

Calculate a result depending on structure

Recursive Structure

Accumulator

Answer thus far

Use

The Length of a List

Without Accumulator

```
listlen([],0).
listlen([H|T],N) :- listlen(T,N1),
N is N1 + 1.
```

With Accumulator

```
listlen(L,N) :- listacc(L,0,N).
listacc([],A,A).
listacc([H|T],A,N) :-
A1 is A + 1,
listacc(T,A1,N).
```

Trace

```
| ?- listlen([a,b,c],N).
 + 1  1  Call: listlen([a,b,c],_83) ?
 + 2  2  Call: listacc([a,b,c],0,_83) ?
 + 4  3  Call: listacc([b,c],1,_83) ?
 + 6  4  Call: listacc([c],2,_83) ?
 + 8  5  Call: listacc([],3,_83) ?
 + 8  5  Exit: listacc([],3,3) ?
 + 6  4  Exit: listacc([c],2,3) ?
 + 4  3  Exit: listacc([b,c],1,3) ?
 + 2  2  Exit: listacc([a,b,c],0,3) ?
 + 1  1  Exit: listlen([a,b,c],3) ?

N = 3 ?
```

With Accumulator

Auxiliary Predicate

`lenacc(L,A,N)`

The length of list `L`
when added to `A`
is the number `N`

Structure of this auxilliary function is similar to the original recursive funtion

Flow

Recursive Sub-Goal

The length of the smaller list
(without the head)

Delagation

The production of the final result is delagated
entirely to the recursive subgoal
`lenacc([],A,A).`

Information

All information is provided
by the accumulator

Assembly

Accumulators need not be integers

Assembly Example

`assembly(bike,[wheel,wheel,frame]).`

Before

finding parts of a frame

appending these to the empty list
to give the parts of `[frame]`

finding the parts of the wheel

appending these to the parts of `[frame]`
to get the parts of `[wheel,frame]`

finding the parts of the wheel

appending these to the parts of `[frame]`
to get the parts of `[wheel,frame]`

Wasteful

The list of parts for a sub-part has to be built twice

First

In determining the parts

Second

appending it to the list

Also

Some sub-parts are assemblies

These have to be formed again

Accumulators

Avoids this extra work

Current Inventory is the Accumulator

As a `basicpart` is found it is **accumulated** immediately on the list

With Accumulators

```
partsof(X,P) :- partsacc(X,[],P).

partsacc(X,A,[X|A]) :- basicpart(X).
partsacc(X,A,P) :-
     assembly(X,Subparts),
     partsacclist(Subparts,A,P).

partsacclist([],A,A).
partsacclist([P|Tail],A,Total) :-
     partsacc(P,A,Headparts),
     partsacclist(Tail,Headparts,Total).
```

```
— ?-  partsof(bike,X).
 + 1  1  Call: partsof(bike,_67) ?
 + 2  2  Call: partsacc(bike,[],_67) ?
 + 3  3  Call: basicpart(bike) ?
 + 3  3  Fail: basicpart(bike) ?
 + 3  3  Call: assembly(bike,_523) ?
 + 3  3  Exit: assembly(bike,[wheel,wheel,frame]) ?
 + 4  3  Call: partsacclist([wheel,wheel,frame],[],_67) ?
 + 5  4  Call: partsacc(wheel,[],_1087) ?
 + 6  5  Call: basicpart(wheel) ?
 + 6  5  Fail: basicpart(wheel) ?
 + 6  5  Call: assembly(wheel,_1292) ?
 + 6  5  Exit: assembly(wheel,[spoke,rim,hub]) ?
 + 7  5  Call: partsacclist([spoke,rim,hub],[],_1087) ?

 + 7  5  Exit: partsacclist([spoke,rim,hub],[],
                       [nut,bolt,gears,rim,spoke]) ?
 + 5  4  Exit: partsacc(wheel,[],[nut,bolt,gears,rim,spoke]) ?
 + 31  4  Call: partsacclist([wheel,frame],
                       [nut,bolt,gears,rim,spoke],_67) ?
 + 32  5  Call: partsacc(wheel,[nut,bolt,gears,rim,spoke],_10743) ?
```

Difference Structures

Use of Accumulators
Accumulated List in reverse order

Sometimes this does not matter
Ex. assembly example

Sometimes it does
generating english sentences
(words in opposite order)

Accumulator Organization

Result thus far

Final Result

Difference Structures

Final Result

A Hole for further information

Holes

List with a Hole

`[a,b,c|X]`

The hole

`X`

The hole is passed to the prolog goal
which will fill the hole
(instantiate with something)
and pass back a new hole

Filling Holes

```
Res = [a,b,c|X], p(X,NewHole), NewHole = [z].
```

list with a hole

```
Res = [a,b,c|X]
```

the hole

A predicate to fill in the hole

```
p(X,NewHole)
```

and deliver a new hole

The new hole is filled

```
NewHole = [z]
```

This sequence fills in the structure (Res) with the information provided by the predicate p

No New Info

```
| ?- assert(p(Hole,Hole)).

| ?- Res = [a,b,c|X],
            p(X,NewHole),
            NewHole=[z].
 + 1  1  Call: _55=[a,b,c|_97] ?
 + 1  1  Exit: [a,b,c|_97]=[a,b,c|_97]
 + 2  1  Call: p(_97,_130) ?
 + 2  1  Exit: p(_97,_97) ?
 + 3  1  Call: _97=[z] ?
 + 3  1  Exit: [z]=[z] ?

NewHole = [z],
Res = [a,b,c,z],
X = [z] ?
```

Inserting an element

```
| ?- assert(p([d|NewHole],NewHole)).

| ?- Res = [a,b,c|X],
           p(X,NewHole),
           NewHole=[z].
 + 3  1  Redo: [z]=[z] ?
 + 3  1  Fail: _97=[z] ?
 + 2  1  Redo: p(_97,\_97) ?
 + 2  1  Exit: p([d|_130],\_130) ?
 + 3  1  Call: _130=[z] ?
 + 3  1  Exit: [z]=[z] ?

NewHole = [z],
Res = [a,b,c,d,z],
X = [d,z] ?
```

Parts Example

```
partsof(X,P) :-
     partsacc(X,P,Hole),
     Hole = [].

partsacc(X,[X|Hole],Hole) :-
     basicpart(X).
partsacc(X,P,Hole) :-
     assembly(X,Subparts),
     partsacclist(Subparts,P,Hole).

partsacclist([],Hole,Hole).
partsacclist([P|Tail],Total,Hole) :-
     partsacc(P,Total,Hole1),
     partsacclist(Tail,Hole1,Hole).
```

Basic Part

— ?- partsof(rim,X).
+ 1 1 Call: partsof(rim,_67) ?
+ 2 2 Call: partsacc(rim,_67,_327) ?
+ 3 3 Call: basicpart(rim) ?
+ 3 3 Exit: basicpart(rim) ?
+ 2 2 Exit: partsacc(rim,[rim—˙327],_327) ?
+ 4 2 Call: ˙327=[] ?
+ 4 2 Exit: []=[] ?
+ 1 1 Exit: partsof(rim,[rim]) ?

X = [rim] ?

assembly

```
— ?- partsof(axle,X).
 + 1  1  Call: partsof(axle,˙67) ?
 + 2  2  Call: partsacc(axle,˙67,˙327) ?
 + 3  3  Call: basicpart(axle) ?
 + 3  3  Fail: basicpart(axle) ?
 + 3  3  Call: assembly(axle,˙529) ?
 + 3  3  Exit: assembly(axle,[bolt,nut]) ?
 + 4  3  Call: partsacclist([bolt,nut],˙67,˙327) ?
 + 5  4  Call: partsacc(bolt,˙67,˙1089) ?
 + 6  5  Call: basicpart(bolt) ?
 + 6  5  Exit: basicpart(bolt) ?
 + 5  4  Exit: partsacc(bolt,[bolt—˙1089],˙1089) ?
 + 7  4  Call: partsacclist([nut],˙1089,˙327) ?
 + 8  5  Call: partsacc(nut,˙1089,˙1997) ?
 + 9  6  Call: basicpart(nut) ?
 + 9  6  Exit: basicpart(nut) ?
 + 8  5  Exit: partsacc(nut,[nut—˙1997],˙1997) ?
 + 10  5  Call: partsacclist([],˙1997,˙327) ?
 + 10  5  Exit: partsacclist([],˙327,˙327) ?
 + 7  4  Exit: partsacclist([nut],[nut—˙327],˙327) ?
 + 4  3  Exit: partsacclist([bolt,nut],[bolt,nut—˙327],˙327) ?
 + 2  2  Exit: partsacc(axle,[bolt,nut—˙327],˙327) ?
 + 11  2  Call: ˙327=[] ?
 + 11  2  Exit: []=[] ?
 + 1  1  Exit: partsof(axle,[bolt,nut]) ?

X = [bolt,nut] ?

yes
```

assembly and basic part

```
— ?- partsof(hub,X).
 + 1  1  Call: partsof(hub,_67) ?
 + 2  2  Call: partsacc(hub,_67,_327) ?
 + 3  3  Call: basicpart(hub) ?
 + 3  3  Fail: basicpart(hub) ?
 + 3  3  Call: assembly(hub,_529) ?
 + 3  3  Exit: assembly(hub,[gears,axle]) ?
 + 4  3  Call: partsacclist([gears,axle],_67,_327) ?
 + 5  4  Call: partsacc(gears,_67,_1089) ?
 + 6  5  Call: basicpart(gears) ?
 + 6  5  Exit: basicpart(gears) ?
 + 5  4  Exit: partsacc(gears,[gears—˙1089],_1089) ?
 + 7  4  Call: partsacclist([axle],_1089,_327) ?
 + 8  5  Call: partsacc(axle,_1089,_1997) ?
 + 9  6  Call: basicpart(axle) ?
 + 9  6  Fail: basicpart(axle) ?
 + 9  6  Call: assembly(axle,_2202) ?
 + 9  6  Exit: assembly(axle,[bolt,nut]) ?
 + 10  6  Call: partsacclist([bolt,nut],_1089,_1997) ?
                .
                .
                .
 + 10  6  Exit: partsacclist([bolt,nut],[bolt,nut—˙1997],_1997) ?
 + 8  5  Exit: partsacc(axle,[bolt,nut—˙1997],_1997) ?
 + 17  5  Call: partsacclist([],_1997,_327) ?
 + 17  5  Exit: partsacclist([],_327,_327) ?
 + 7  4  Exit: partsacclist([axle],[bolt,nut—˙327],_327) ?
 + 4  3  Exit: partsacclist([gears,axle],[gears,bolt,nut—˙327],_327) ?
 + 2  2  Exit: partsacc(hub,[gears,bolt,nut—˙327],_327) ?
 + 18  2  Call: ˙327=[] ?
 + 18  2  Exit: []=[] ?
 + 1  1  Exit: partsof(hub,[gears,bolt,nut]) ?

X = [gears,bolt,nut] ?
```