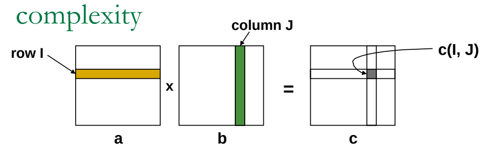
Runtime Analysis and Program Transformations for Dynamic Programs

John Blatz
CS 325/425
April 26, 2006

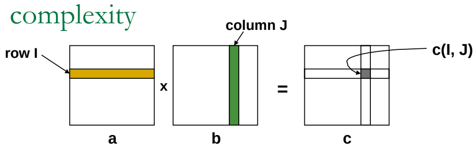
Matrix multiplication & computational complexity




```
for I in 1:n
  for J in 1:n
    c(I,J) = 0
    for K in 1:n
      c(I,J) += a(I,K) * b(K,J)
```

$O(n^3)$

Matrix multiplication & computational complexity



$O(n^3)$

(declaratively) equivalent Dyna program:

```
c(I,J) += a(I,K) * b(K,J)
```


Example: context-free parsing

```
goal += constit(s,0,N) * end(N).
constit(X,I,J) += rewrite(X,W) * word(W,I,J).
constit(X,I,K) += rewrite(X,Y,Z) * constit(Y,I,J) *
  constit(Z,J,K).
```

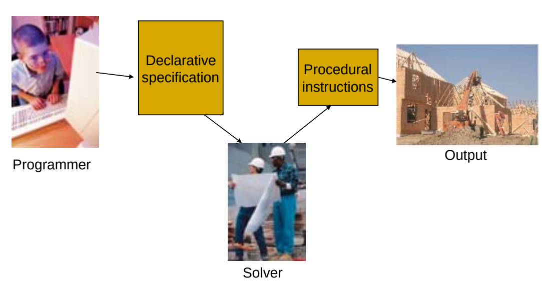
- k grammar symbols (X, Y, Z)
- n words in sentence (I, J, K)
- $O(k^3n^3)$
- Actually just an upper bound! (why?)

Sparsity

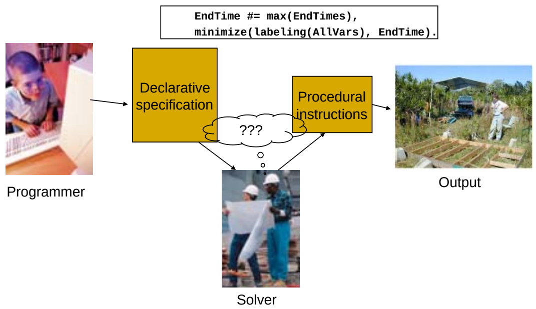
- Runtime of a dynamic rule = total number of ways to instantiate it
- Sparse computations much faster
- Example: multiplication of diagonal matrices
  - Only a and b items that exist are of the form a(I, I) or b(I, I)
  - Asymptotic runtime = $O(n)$ instead of $O(n^3)$

c(I,J) += a(I,K) * b(K,J) → **unification** → **c(I,I) += a(I,I) * b(I,I)**

Building a declarative house



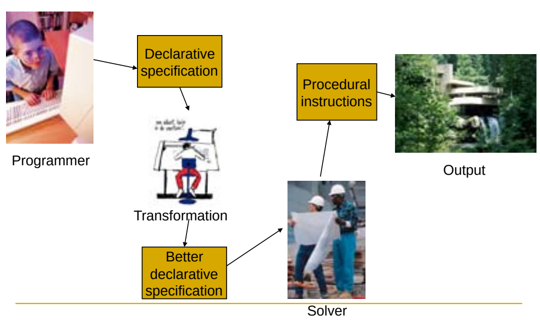
Building a declarative house



Semi-declarative programming

- How can we get the solver to be more efficient?
  - Tell it how to solve the problem:
    - minimize(search(AllVars, 0, smallest, indomain_min, complete, []), EndTime).
  - Explain the problem differently

Building a declarative house



Program transformation examples

```
above(X, Y) :- above(Underling, Y), boss(X, Underling).
```

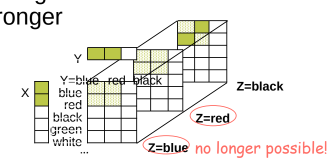
- Prolog will recurse forever on this program
- "Transform" into equivalent program:

```
above(X, Y) :- boss(X, Underling), above(Underling, Y).
```


Program transformation examples

```
X #\= Y,
Y #\= Z,    fuse constraints ⇒   alldifferent([X,Y,Z])
Z #\= X,
```

- Fusing constraints makes arc consistency stronger



alldifferent(X,Y,Z) **and** [X,Y]::[blue,red]

Program transformation examples

```
rooted(t(R,[])) max= iq(R).
unrooted(t(R,[])) max= zero whenever iq(R).
zero := 0.
any(T) max= rooted(T).        any(T) max= unrooted(T).

rooted(t(R,[X|Xs])) max= rooted(t(R,Xs)) + unrooted(X).

unrooted(t(R,[X|Xs])) max= unrooted(t(R,Xs)) + any(X)
```

- Above example computes all possible trees, and so it will run forever
- Transform it to only consider trees that we are interested in

Program transformation examples

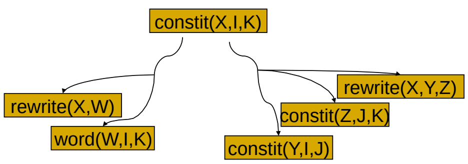
```
rooted(t(R,[])) max= iq(R).
unrooted(t(R,[])) max= zero whenever iq(R).
zero := 0.
any(T) max= rooted(T).          any(T) max= unrooted(T).

rooted(t(R,[X|Xs])) max= rooted(t(R,Xs)) + unrooted(X).
  whenever(interesting(t(R,[X|Xs]))).
unrooted(t(R,[X|Xs])) max= unrooted(t(R,Xs)) + any(X)
  whenever(interesting(t(R,[X|Xs]))).

interesting(X) max= input(X).
interesting(X) max= interesting(t(R,[X|_])).
interesting(t(R,Xs)) max= interesting(t(R,[_|Xs])).
goal max= any(X) whenever input(X).
```


The folding/unfolding paradigm

- Small, basic steps which can be composed
- Has been applied to several declarative languages

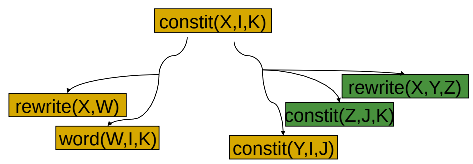
Folding

```
goal += constit(s,0,N) * end(N).
constit(X,I,J) += word(W,I,J) * rewrite(X,W) .
constit(X,I,K) += constit(Y,I,J) * constit(Z,J,K) *
  rewrite(X,Y,Z).
```



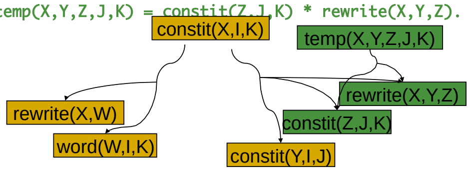
Folding

```
goal += constit(s,0,N) * end(N).
constit(X,I,J) += word(W,I,J) * rewrite(X,W) .
constit(X,I,K) += constit(Y,I,J) * constit(Z,J,K) *
  rewrite(X,Y,Z).
```



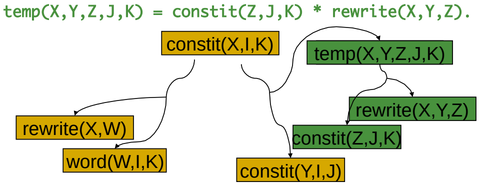
Folding

```
goal += constit(s,0,N) * end(N).
constit(X,I,J) += word(W,I,J) * rewrite(X,W) .
constit(X,I,K) += constit(Y,I,J) * constit(Z,J,K) *
  rewrite(X,Y,Z).
temp(X,Y,Z,J,K) = constit(Z,J,K) * rewrite(X,Y,Z).
```



Folding

```
goal += constit(s,0,N) * end(N).
constit(X,I,J) += word(W,I,J) * rewrite(X,W) .
constit(X,I,K) += constit(Y,I,J) * temp(X,Y,Z,J,K)
temp(X,Y,Z,J,K) = constit(Z,J,K) * rewrite(X,Y,Z).
```



Fully transformed version

k grammar symbols (X,Y,Z)
n positions in string (I,J,K)

```
goal += constit(s,0,N) * end(N).
constit(X,I,J) += word(W,I,J) * rewrite(X,W) .
constit(X,I,K) += constit(Y,I,J) * temp(X,Y,Z,J,K)
temp(X,Y,Z,J,K) = constit(Z,J,K) * rewrite(X,Y,Z).
```

- Still $O(k^3n^3)$ in the worst-case
- But could actually be much faster—why?
  - Many constit(Z,J,K) items, few rewrite(X,Y,Z)
  - Avoids repeating work if temp is already built
  - Fails faster if agenda is poorly ordered
  - Could be followed by another transformation

Folding

temp(X,Y,Z,J,K) =
constit(Z,J,K) * rewrite(X,Y,Z).

- Folding can improve *asymptotic* runtime!

constit(X,I,K) =

```
 constit(y1,I,j1) * constit(z1,j1,K) *    temp(X,y1,z1,j1,K)
   rewrite(X,y1,z1)                        temp(X,y1,z2,j1,K)
 + constit(y1,I,j1)     * constit(z2,j1,K) *  temp(X,y2,z1,j1,K)
   rewrite(X,y1,z2)                        temp(X,y2,z2,j1,K)
 + constit(y2,I,j1)     * constit(z1,j1,K) *  temp(X,y1,z1,j2,K)
   rewrite(X,y2,z1)                        temp(X,y1,z2,j2,K)
 + constit(y2,I,j1)     * constit(z2,j1,K) *  temp(X,y2,z1,j2,K)
   rewrite(X,y2,z2)                        temp(X,y2,z2,j2,K)
 + constit(y1,I,j2)     * constit(z1,j2,K) *
   rewrite(X,y1,z1)
 + constit(y1,I,j2)     * constit(z2,j2,K) *
   rewrite(X,y1,z2)
 + constit(y2,I,j2)     * constit(z1,j2,K) *
```


Folding

temp2(X,Y,J,K) = constit(Z,J,K)
* rewrite(X,Y,Z).

- Sum over values of Z before summing over Y and J

constit(X,I,K) =

```
 constit(y1,I,j1) * constit(z1,j1,K) *
   rewrite(X,y1,z1)
 + constit(y1,I,j1)     * constit(z2,j1,K) *
   rewrite(X,y1,z2)
 + constit(y2,I,j1)     * constit(z1,j1,K) *
   rewrite(X,y2,z1)
 + constit(y2,I,j1)     * constit(z2,j1,K) *
   rewrite(X,y2,z2)
 + constit(y1,I,j2)     * constit(z1,j2,K) *
   rewrite(X,y1,z1)
 + constit(y1,I,j2)     * constit(z2,j2,K) *
   rewrite(X,y1,z2)
 + constit(y2,I,j2)     * constit(z1,j2,K) *
```


Folding

temp2(X,Y,J,K) = constit(Z,J,K)
* rewrite(X,Y,Z).

- Sum over values of Z before summing over Y and J

constit(X,I,K) =

```
constit(y1,I,j1) *  constit(z1,j1,K) * rewrite(X,y1,z1)   temp2(X,y1,j1,K)
                  + constit(z2,j1,K) * rewrite(X,y1,z2)
 + constit(y2,I,j1)     * constit(z1,j1,K) *
   rewrite(X,y2,z1)
 + constit(y2,I,j1)     * constit(z2,j1,K) *
   rewrite(X,y2,z2)
 + constit(y1,I,j2)     * constit(z1,j2,K) *
   rewrite(X,y1,z1)
 + constit(y1,I,j2)     * constit(z2,j2,K) *
   rewrite(X,y1,z2)
 + constit(y2,I,j2)     * constit(z1,j2,K) *
   rewrite(X,y2,z1)
```


Folding

temp2(X,Y,J,K) = constit(Z,J,K)
* rewrite(X,Y,Z).

- Sum over values of Z before summing over Y and J

constit(X,I,K) =

```
constit(y1,I,j1) *    constit(z1,j1,K) * rewrite(X,y1,z1)   temp2(X,y1,j1,K)
                    + constit(z2,j1,K) * rewrite(X,y1,z2)
+ constit(y2,I,j1) *  constit(z1,j1,K) * rewrite(X,y2,z1)   temp2(X,y2,j1,K)
                    + constit(z2,j1,K) * rewrite(X,y2,z2)
+ constit(y1,I,j2) *  constit(z1,j2,K) * rewrite(X,y1,z1)   temp2(X,y1,j2,K)
                    + constit(z2,j2,K) * rewrite(X,y1,z2)
+ constit(y2,I,j2) *  constit(z1,j2,K) * rewrite(X,y2,z1)   temp2(X,y2,j2,K)
                    + constit(z2,j2,K) * rewrite(X,y2,z2)
+ …
```


Folding – best version

k grammar symbols (X,Y,Z)
n positions in string (I,J,K)

```
goal += constit(s,0,N) * end(N).
constit(X,I,J) += word(W,I,J) * rewrite(X,W) .
constit(X,I,K) += constit(Y,I,J) * temp2(X,Y,J,K)
temp2(X,Y,J,K) += constit(Z,J,K) * rewrite(X,Y,Z).
```

- Asymptotic complexity has been reduced!
  - $O(k^2n^3)$ for constit rule (doesn't mention Z)
  - + $O(k^3n^2)$ for temp2 rule (doesn't mention I)

Other names for folding

- Substitution
- Storing intermediate results
- Common subexpression elimination
- Moving an invariant out of a loop
- Building speculatively

Unfolding

pathto(Y) ***max=***
pathto(Z) + edge(Z, Y).

- Unfolding = inverse of folding
- Inlines computation

pathto(X) max= pathto(Y) + edge(Y, X).

pathto(X) max= pathto(Z) + edge(Z,Y) + edge(Y, X).

Pop quiz

- A folding transformation can possibly
  increase *decrease* *not affect*
  the asymptotic time complexity.
- A folding transformation can possibly
  increase *decrease* *not affect*
  the asymptotic space complexity.

Pop quiz

- An unfolding transformation can possibly
  increase *decrease* *not affect*
  the asymptotic time complexity.
- An unfolding transformation can possibly
  increase *decrease* *not affect*
  the asymptotic space complexity.

Maximum independent set in a tree

any(T) = the size of the maximum independent set in T
rooted(T) = the size of the maximum independent set in T that includes T's root
unrooted(T) = the size of the maximum independent set in T that excludes T's root

as before

- rooted(t(R,[])) max= iq(R).
- unrooted(t(_,[])) max= 0.

- any(T) max= rooted(T).
- any(T) max= unrooted(T).

- rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
- unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).

Maximum independent set in a tree

(shorter but harder to understand version: find it automatically?)

- We could actually eliminate "rooted" from the program. Just do everything with "unrooted" and "any."
- Slightly more efficient, but harder to convince yourself it's right.
- That is, it's an optimized version of the previous slide!
- **We can prove it's equivalent by a sequence of folding and unfolding steps—let's see how!**

- any(t(R,[])) max= iq(R).
- unrooted(t(_,[])) max= 0.

- any(T) max= unrooted(T).

- any(t(R,[X|Xs])) max= any(t(R,Xs)) + unrooted(X).
- unrooted(t(R,[X|Xs])) max= unrooted(t(R,Xs)) + any(X).

① any(T) max= rooted(T).
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).

① any(T) max= rooted(T). unfold
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).

① any(T) max= rooted(T). replaced by ⑦, ⑧
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).

- Gray rules are no longer part of the program

① any(T) max= rooted(T).
② any(T) max= unrooted(T). unfold
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).

① any(T) max= rooted(T).
② any(T) max= unrooted(T). replaced by ⑨, ⑩
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑨ any(t(_,[])) max= 0.
⑩ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).

① any(T) max= rooted(T).
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)). unfold
⑨ any(t(_,[])) max= 0.
⑩ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).

- Rules ① and ② are no longer part of the current program
- They were the definition of any(T) in a previous valid program, so we can use them for unfolding

① any(T) max= rooted(T).
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑨ any(t(_,[])) max= 0.
⑩ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑪ any(t(R,[X|Xs])) max= rooted(X) + unrooted(t(R, Xs)).
⑫ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).

replaced by ⑪, ⑫

① any(T) max= rooted(T).
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑨ any(t(_,[])) max= 0.
⑩ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑪ any(t(R,[X|Xs])) max= rooted(X) + unrooted(t(R, Xs)).
⑫ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑬ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).

duplicate

- Duplicating a rule doesn't affect the value computed by max=
- Allowed to do this transformation because of this particular property of max=

① any(T) max= rooted(T).
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑨ any(t(_,[])) max= 0.
⑩ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑪ any(t(R,[X|Xs])) max= rooted(X) + unrooted(t(R, Xs)).
⑫ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑬ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).

fold

- Fold two rules into two rules

① any(T) max= rooted(T).
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑨ any(t(_,[])) max= 0.
⑩ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑪ any(t(R,[X|Xs])) max= rooted(X) + unrooted(t(R, Xs)).
⑫ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑬ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑭ any(t(R,[X|Xs])) max= unrooted(X) + any(t(R, Xs)).

replaced by ⑭

① any(T) max= rooted(T).
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑨ any(t(_,[])) max= 0.
⑩ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑪ any(t(R,[X|Xs])) max= rooted(X) + unrooted(t(R, Xs)).
⑫ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑬ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑭ any(t(R,[X|Xs])) max= unrooted(X) + any(t(R, Xs)).

fold

① any(T) max= rooted(T).
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑨ any(t(_,[])) max= 0.
⑩ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑪ any(t(R,[X|Xs])) max= rooted(X) + unrooted(t(R, Xs)).
⑫ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑬ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑭ any(t(R,[X|Xs])) max= unrooted(X) + any(t(R, Xs)).
⑮ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).

replaced by ⑮

① any(T) max= rooted(T).
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑨ any(t(_,[])) max= 0.
⑩ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑪ any(t(R,[X|Xs])) max= rooted(X) + unrooted(t(R, Xs)).
⑫ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑬ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑭ any(t(R,[X|Xs])) max= unrooted(X) + any(t(R, Xs)).
⑮ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).

fold

① any(T) max= rooted(T).
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑨ any(t(_,[])) max= 0.
⑩ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑪ any(t(R,[X|Xs])) max= rooted(X) + unrooted(t(R, Xs)).
⑫ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑬ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑭ any(t(R,[X|Xs])) max= unrooted(X) + any(t(R, Xs)).
⑮ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑯ any(T) max= unrooted(T).

fold

① any(T) max= rooted(T).
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑨ any(t(_,[])) max= 0.
⑩ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑪ any(t(R,[X|Xs])) max= rooted(X) + unrooted(t(R, Xs)).
⑫ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑬ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑭ any(t(R,[X|Xs])) max= unrooted(X) + any(t(R, Xs)).
⑮ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑯ any(T) max= unrooted(T).

trim

- Nothing relies on rooted anymore, so we can delete it

① any(T) max= rooted(T).
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑨ any(t(_,[])) max= 0.
⑩ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑪ any(t(R,[X|Xs])) max= rooted(X) + unrooted(t(R, Xs)).
⑫ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑬ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑭ any(t(R,[X|Xs])) max= unrooted(X) + any(t(R, Xs)).
⑮ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑯ any(T) max= unrooted(T).

Success!

⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).

⑭ any(t(R,[X|Xs])) max= unrooted(X) + any(t(R, Xs)).

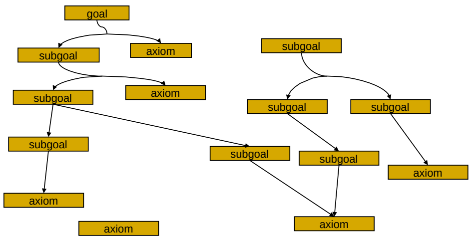
⑯ any(T) max= unrooted(T).

① any(T) max= rooted(T).
② any(T) max= unrooted(T).
③ rooted(t(R,[])) max= iq(R).
④ rooted(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑤ unrooted(t(_,[])) max= 0.
⑥ unrooted(t(R,[X|Xs])) max= any(X) + unrooted(t(R,Xs)).
⑦ any(t(R,[])) max= iq(R).
⑧ any(t(R,[X|Xs])) max= unrooted(X) + rooted(t(R,Xs)).
⑨ any(t(_,[])) max= 0.
⑩ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑪ any(t(R,[X|Xs])) max= rooted(X) + unrooted(t(R, Xs)).
⑫ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑬ any(t(R,[X|Xs])) max= unrooted(X) + unrooted(t(R, Xs)).
⑭ any(t(R,[X|Xs])) max= unrooted(X) + any(t(R, Xs)).
⑮ any(t(R,[X|Xs])) max= any(X) + unrooted(t(R, Xs)).
⑯ any(T) max= unrooted(T).

unfold

- ①②③④⑤⑥⑦⑨⑩⑪⑫⑬⑭⑮⑯
- ① any(T) max= unrooted(T).
- ② any(T) max= rooted(T).

Bottom-up evaluation



Combine all axioms to build all possible subgoals…
many irrelevant to goal!

“Magic Templates” Transformation

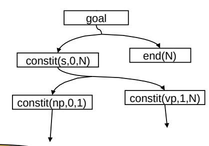
- Simulate top-down execution
- Introduce new “interesting” predicates that keep track of what top-down execution built
- “interesting(foo(x))” means “top-down execution would have examined “foo(x)”
- If in order to build “foo” you need to build “bar”, then interesting(bar) :- interesting(foo).
- Magic predicates will later be used to filter bottom-up construction

relate to “interesting” from max indep set example

What would Prolog do?

```
goal += constit(s,0,N) * end(N).   <-- apply this rule
constit(X,I,J) += rewrite(X,W) * word(W,I,J).
constit(X,I,K) += rewrite(X,Y,Z) * constit(Y,I,J) *   <-- now apply this rule
constit(Z,J,K).
```

- `interesting(goal).`
- `interesting(constit(s,0)) :- interesting(goal).`
- `interesting(rewrite(s)) :- interesting(constit(s,0)).`
- `interesting(constit(np,0)) :- interesting(constit(s,0)), rewrite(s,np,vp).`
- `interesting(constit(vp,1)) :- interesting(constit(s,0)), rewrite(s,np,vp), constit(np,0,1).`



Don’t really need this-- assume we’ve built all the grammar rules beforehand

Generalization

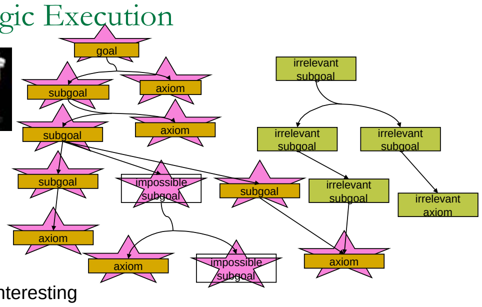
- foo(X, Y) += bar(X, A) * baz(A, Y).
- interesting(bar(X)) |= interesting(foo(X, Y)).
- interesting(baz(A,Y)) |= interesting(foo(X,Y)) & bar(X,A).

The transformed program

```
interesting(goal) = TRUE.
interesting(constit(s,0)) |= interesting(goal).
interesting(constit(Y,I)) |= interesting(constit(X,I)) &
   rewrite(X,Y,Z).
interesting(constit(Z,J)) |= interesting(constit(X,I)) &
   rewrite(X,Y,Z) & constit(Y,I,J).
goal += constit(s,0,N) * end(N) if interesting(goal).
constit(X,I,J) += rewrite(X,W) * word(W,I,J) if
   interesting(constit(X,I)).
constit(X,I,K) += rewrite(X,Y,Z) * constit(Y,I,J) *
   constit(Z,J,K) if interesting(constit(X,I)).
```

Uses forward chaining to simulate backward chaining

Magic Execution



Build interesting filters before other predicates

600.325/425 Declarative Methods – J. Eisner/J. Blatz