Example of Naive (1)

```
Local(Id, Loc, 1) :- Local0(Id, Loc).
Local(Id, Loc, K) :- Op1(Id, In, Loc),
   Local(In, Loc, M), Sum(M, 1, K).
Local(Id, Loc, K) :- Op2(Id, In1, In2, Loc),
   Local(In1, Loc, M), Local(In2, Loc, N),
   Sum(M, N, J), Sum(J, 1, K).
Local0(r, node1).
Local0(s, node1).
Local0(q, node1).
Local0(u, node2).
Op1(s1, r, node1).
Op1(s2, u, node2).
Op2(j1, s1, u1, node1).
Op2(u1, s, q, node1).
Op2(j2, j1, s2, node1).
```

f0 = ∅

Example of Naïve (2)

```
Local(Id, Loc, 1) :- Local0(Id, Loc).
Local(Id, Loc, K) :- Op1(Id, In, Loc),
   Local(In, Loc, M), Sum(M, 1, K).
Local(Id, Loc, K) :- Op2(Id, In1, In2, Loc),
   Local(In1, Loc, M), Local(In2, Loc, N),
   Sum(M, N, J), Sum(J, 1, K).
Local0(r, node1).
Local0(s, node1).
Local0(q, node1).
Local0(u, node2).
Op1(s1, r, node1).
Op1(s2, u, node2).
Op2(j1, s1, u1, node1).
Op2(u1, s, q, node1).
Op2(j2, j1, s2, node1).
```

f1 = I_P(f0) =

```
Local0(r, node1).
Local0(s, node1).
Local0(q, node1).
Local0(u, node2).
Op1(s1, r, node1).
Op1(s2, u, node2).
Op2(j1, s1, u1, node1).
Op2(u1, s, q, node1).
Op2(j2, j1, s2, node1).
```


Example of Naïve (3)

```
Local(Id, Loc, 1) :- Local0(Id, Loc).
Local(Id, Loc, K) :- Op1(Id, In, Loc),
   Local(In, Loc, M), Sum(M, 1, K).
Local(Id, Loc, K) :- Op2(Id, In1, In2, Loc),
   Local(In1, Loc, M), Local(In2, Loc, N),
   Sum(M, N, J), Sum(J, 1, K).
Local0(r, node1).
Local0(s, node1).
Local0(q, node1).
Local0(u, node2).
Op1(s1, r, node1).
Op1(s2, u, node2).
Op2(j1, s1, u1, node1).
Op2(u1, s, q, node1).
Op2(j2, j1, s2, node1).
```

f2 = I_P(f1) =

```
Local(r, node1, 1).
Local(s, node1, 1).
Local(q, node1, 1).
Local(u, node2, 1).
Local0(r, node1).
Local0(s, node1).
Local0(q, node1).
Local0(u, node2).
Op1(s1, r, node1).
Op1(s2, u, node2).
Op2(j1, s1, u1, node1).
Op2(u1, s, q, node1).
Op2(j2, j1, s2, node1).
```


Example of Naïve (4)

```
Local(Id, Loc, 1) :- Local0(Id, Loc).
Local(Id, Loc, K) :- Op1(Id, In, Loc),
   Local(In, Loc, M), Sum(M, 1, K).
Local(Id, Loc, K) :- Op2(Id, In1, In2, Loc),
   Local(In1, Loc, M), Local(In2, Loc, N),
   Sum(M, N, J), Sum(J, 1, K).
Local0(r, node1).
Local0(s, node1).
Local0(q, node1).
Local0(u, node2).
Op1(s1, r, node1).
Op1(s2, u, node2).
Op2(j1, s1, u1, node1).
Op2(u1, s, q, node1).
Op2(j2, j1, s2, node1).
```

f3 = I_P(f2) =

```
Local(r, node1, 1).
Local(s, node1, 1).
Local(q, node1, 1).
Local(u, node2, 1).
Local(s1, node1, 2).
Local(u1, node1, 3).
Local(s2, node2, 2).
Local0(r, node1).
Local0(s, node1).
Local0(q, node1).
Local0(u, node2).
Op1(s1, r, node1).
Op1(s2, u, node2).
Op2(j1, s1, u1, node1).
Op2(u1, s, q, node1).
Op2(j2, j1, s2, node1).
```


Example of Naïve (5)

```
Local(Id, Loc, 1) :- Local0(Id, Loc).
Local(Id, Loc, K) :- Op1(Id, In, Loc),
   Local(In, Loc, M), Sum(M, 1, K).
Local(Id, Loc, K) :- Op2(Id, In1, In2, Loc),
   Local(In1, Loc, M), Local(In2, Loc, N),
   Sum(M, N, J), Sum(J, 1, K).
Local0(r, node1).
Local0(s, node1).
Local0(q, node1).
Local0(u, node2).
Op1(s1, r, node1).
Op1(s2, u, node2).
Op2(j1, s1, u1, node1).
Op2(u1, s, q, node1).
Op2(j2, j1, s2, node1).
```

f4 = I_P(f3) =

```
Local(r, node1, 1).
Local(s, node1, 1).
Local(q, node1, 1).
Local(u, node2, 1).
Local(s1, node1, 2).
Local(u1, node1, 3).
Local(j1, node1, 6).
Local(s2, node2, 2).
Local0(r, node1).
Local0(s, node1).
Local0(q, node1).
Local0(u, node2).
Op1(s1, r, node1).
Op1(s2, u, node2).
Op2(j1, s1, u1, node1).
Op2(u1, s, q, node1).
Op2(j2, j1, s2, node1).
```


Example of Naïve (6)

```
Local(Id, Loc, 1) :- Local0(Id, Loc).
Local(Id, Loc, K) :- Op1(Id, In, Loc),
   Local(In, Loc, M), Sum(M, 1, K).
Local(Id, Loc, K) :- Op2(Id, In1, In2, Loc),
   Local(In1, Loc, M), Local(In2, Loc, N),
   Sum(M, N, J), Sum(J, 1, K).
Local0(r, node1).
Local0(s, node1).
Local0(q, node1).
Local0(u, node2).
Op1(s1, r, node1).
Op1(s2, u, node2).
Op2(j1, s1, u1, node1).
Op2(u1, s, q, node1).
Op2(j2, j1, s2, node1).
```

f5 = I_P(f4) = f4

```
Local(r, node1, 1).
Local(s, node1, 1).
Local(q, node1, 1).
Local(u, node2, 1).
Local(s1, node1, 2).
Local(u1, node1, 3).
Local(j1, node1, 6).
Local(s2, node2, 2).
Local0(r, node1).
Local0(s, node1).
Local0(q, node1).
Local0(u, node2).
Op1(s1, r, node1).
Op1(s2, u, node2).
Op2(j1, s1, u1, node1).
Op2(u1, s, q, node1).
Op2(j2, j1, s2, node1).
```


Problem with Computed Predicates

```
Local(Id, Loc, 1) :- Local0(Id, Loc).
Local(Id, Loc, K) :- Op1(Id, In, Loc),
   Local(In, Loc, M), Sum(M, 1, K).
Local(Id, Loc, K) :- Op2(Id, In1, In2, Loc),
   Local(In1, Loc, M), Local(In2, Loc, N),
   Sum(M, N, J), Sum(J, 1, K).

Local0(r, node1).
Op1(s1, r, node1).
Op1(s1, s1, node1).
```

$f1 = I_P(\varnothing)$

```
Local0(r, node1).
Op1(s1, r, node1).
Op1(s1, s1, node1).
-------------------
```

$f2 = I_P(f1)$

```
Local(r, node1, 1).
Local0(r, node1).
Op1(s1, r, node1).
Op1(s1, s1, node1).
-------------------
```

$f3 = I_P(f2)$

```
Local(r, node1, 1).
Local(s1, node1, 3).
Local0(r, node1).
Op1(s1, r, node1).
Op1(s1, s1, node1).
```


Some Refinements of Naive

- Start with f = facts(P)
- Use dependency graph
  - Nodes are the different predicates of P
  - Have an edge `p` → `q` if there is a rule of the form `q(…) :- …, p(), … .`

The paper uses a more detailed version of the dependency graph than the book, where there are nodes for rules as well.

Example Dependency Graph

```
Needed(D, M, C) :- MajAll(M, C), Offered(D, M).
Needed(D, M, C) :- DegAll(D, C), Offered(D, M).

MajAll(M, C) :- MajReq(M, C).
MajAll(M, C) :- MajAll(M, C1), Prereq(C, C1).

DegAll(D, C) :- DegReq(D, C).
DegAll(D, C) :- DegAll(D, C1), Prereq(C, C1).
```

```
                 Needed

     MajAll        DegAll        Offered

MajReq       Prereq       DegReq
```


Evaluate in Order of Dependency Graph

- Compute `DegAll` from `DegReq` and `Prereq`
- Compute `MajorAll` from `MajReq` and `Prereq`
- Compute `Needed` from `MajAll`, `DegAll`, `Offered`

Semi-Naïve Evaluation

Naïve evaluation discovers the same fact over and over.

- E.g., derive `local(r, node1, 1)` repeatedly

Idea: When evaluating a rule, make sure we use at least one newly discovered fact.

- Otherwise, we derive a fact we've already seen

Example

Consider the rule

```
Local(Id, Loc, K) :- Op2(Id, In1, In2, Loc),
  Local(In1, Loc, M), Local(In2, Loc, N),
  Sum(M, N, J), Sum(J, 1, K).
```

When computing f4 = I_P(f3).

- If we use facts `Op2(u1, s, q, node1)`, `Local(s, node1, 1)`, `Local(q, node1, 1)`, we derive `Local(u1, node1, 3)`, which we already have
- If we use `Op2(j1, s1, u1, node1)`, `Local(s1, node1, 2)`, `Local(u1, node1, 3)`, we get

Implementing Semi-Naive

Notice after f1, only new facts are `Local` facts

Suppose at each step in Naïve we determine

`NewLocal(Id, Loc, K)` if `Local(Id, Loc, K)` in $f - f_{old}$.

We can work with a modified set of rules

```
Local(Id, Loc, K) :- Op1(Id, In, Loc),
  NewLocal(In, Loc, M), Sum(M, 1, K).
Local(Id, Loc, K) :- Op2(Id, In1, In2, Loc),
  NewLocal(In1, Loc, M), Local(In2, Loc, N),
  Sum(M, N, J), Sum(J, 1, K).
Local(Id, Loc, K) :- Op2(Id, In1, In2, Loc),
  Local(In1, Loc, M), NewLocal(In2, Loc, N),
  Sum(M, N, J), Sum(J, 1, K).
```


One Other Modification

If P′ is the modified program, then instead of

$f = I_P(f)$

we need

$f = I_{P'}(f) \cup f$.

Note: Semi-Naïve doesn't guarantee we won't generate a fact at multiple steps, if there are distinct "proofs" of it

```
MajReq(eng, wr310). MajReq(eng, wr408).
Prereq(wr309, wr310).
Prereq(wr401, wr408).
Prereq(wr309, wr401).
```

Directed Methods

Suppose we have the query

```
:- Local(I, node2, K).
```

We could push the constant into the rules:

```
Local(Id, node2, 1) :- Local0(Id, node2).

Local(Id, node2, K) :- Op1(Id, In, node2),
  Local(In, node2, M), Sum(M, 1, K).

Local(Id, node2, K) :- Op2(Id, In1, In2, node2),
  Local(In1, node2, M), Local(In2, node2, N),
  Sum(M, N, J), Sum(J, 1, K).
```


Top-Down Evaluation

What Prolog uses

- Start with the first goal in the list
- If you find a matching rule head, replace that goal by the rule body (with appropriate bindings)
- Continue with the next goal.
- Might have to backtrack if you get stuck or want more answers.

Top-Down Example

Initial goal

```
:- Local(u1, L, C).
```

Try different rules

```
Local(Id, Loc, 1) :- Local0(Id, Loc).

Local(Id, Loc, K) :- Op1(Id, In, Loc),
   Local(In, Loc, M), Sum(M, 1, K).

Local(Id, Loc, K) :- Op2(Id, In1, In2, Loc),
   Local(In1, Loc, M), Local(In2, Loc, N),
   Sum(M, N, J), Sum(J, 1, K).
```


Have a New Goal (List)

```
:- Op2(u1, In1, In2, Loc),
   Local(In1, Loc, M), Local(In2, Loc, N),
   Sum(M, N, J), Sum(J, 1, C).
```

Try Op2 facts

```
Op2(j1, s1, u1, node1).

Op2(u1, s, q, node1).
```

Next Goal List

```
:- Local(s, node1, M), Local(q, node1, N),
Sum(M, N, J), Sum(J, 1, C).
```

Try matching first goal

```
Local(Id, Loc, 1) :- Local0(Id, Loc).
```

Which gives

```
:- Local0(s, node1), Local(q, node1, N),
Sum(1, N, J), Sum(J, 1, C).
```

Try Local0 facts

```
Local0(r, node1).

Local0(s, node1).
```


Continuing

```
:- Local(q, node1, N),
Sum(1, N, J), Sum(J, 1, C).
```

Try matching first goal

```
Local(Id, Loc, 1) :- Local0(Id, Loc).
```

Which gives

```
:- Local0(q, node1), Sum(1, 1, J), Sum(J, 1, C).
```

Try Local0 facts

```
Local0(r, node1).

Local0(s, node1).

Local0(q, node1).
```


Finishing up

```
:- Sum(1, 1, J), Sum(J, 1, C).

Sum(1, 1, 2).

:- Sum(2, 1, C).

Sum(2, 1, 3).
```

Success!

Top-Down Methods

Sometimes you can propagate constants statically (at "compile" time), before any computation.

However, not all constants are available at compile time.

Example of Constant Propagation

Consider query

```
:- Needed(ba, eng, C).
```

for our program

```
Needed(D, M, C) :- MajAll(M, C), Offered(D, M).
Needed(D, M, C) :- DegAll(D, C), Offered(D, M).

MajAll(M, C) :- MajReq(M, C).
MajAll(M, C) :- MajAll(M, C1), Prereq(C, C1).

DegAll(D, C) :- DegReq(D, C).
DegAll(D, C) :- DegAll(D, C1), Prereq(C, C1).
```


Can Do Some Constant Propagation Right Away

Modified program

```
Needed(ba, eng, C) :- MajAll(eng, C), Offered(ba, eng).
Needed(ba, eng, C) :- DegAll(ba, C), Offered(ba, eng).

MajAll(eng, C) :- MajReq(eng, C).
MajAll(eng, C) :- MajAll(eng, C1), Prereq(C, C1).

DegAll(ba, C) :- DegReq(ba, C).
DegAll(ba, C) :- DegAll(ba, C1), Prereq(C, C1).
```


Consider Evaluation of `MajAll`

Suppose we have the facts

```
MajReq(eng, wr310).
MajReq(eng, lit481).
```

We'll derive (by the first rule)

```
MajAll(eng, wr310).
MajAll(eng, lit481.
```

Specializing the second rule:

```
MajAll(eng, C) :- MajAll(eng, wr310), Prereq(C, wr310).
MajAll(eng, C) :- MajAll(eng, lit481),
                  Prereq(C, lit481).
```


New Sub-queries

Pose the new queries

```
:- Prereq(C, wr310).
:- Prereq(C, lit481).
```

which can be used to instantiate the rules together with the `MajAll` facts.

The Iterative Query-Sub-Query (QSQ) Method

Combine queries with the same *query form* (pattern of free and bound positions) into a *generalized query*:

```
:- Prereq(C, {wr310, lit481}).
```

Use generalized query:

- Evaluate against extensional DB (if appropriate)
- Plug results into rules for new facts
- Use new facts to generate new sub-queries
- Combine sub-queries with existing generalized query, if possible

Example

Suppose we have the facts:

```
Prereq(wr301, wr310).
Prereq(wr301, wr418).
Prereq(wr200, wr301).
Prereq(lit480, lit481).
Prereq(lit217, lit218).
```

Which match the generalized query?

```
:- Prereq(C, {wr310, lit481}).
```


Example (2)

Plug

```
Prereq(wr301, wr310).
Prereq(lit480, lit481).
```

plus

```
MajAll(eng, wr310).
MajAll(eng, lit481).
```

into

```
MajAll(eng, C) :- MajAll(eng, C1), Prereq(C, C1).
```

to get

```
MajAll(eng, wr301).
MajAll(eng, lit480).
```


Example (3)

Use

```
MajAll(eng, wr301).
MajAll(eng, lit480).
```

with

```
MajAll(eng, C) :- MajAll(eng, C1), Prereq(C, C1).
```

to get new sub-queries

```
:- Prereq(C, wr301).
:- Prereq(C, lit480).
```

Expand generalized query

```
:- Prereq(C, {wr310, lit481, wr301, lit480}).
```


Example (4)

Evaluate generalized query again

```
Prereq(wr301, wr310).
Prereq(wr301, wr418).
Prereq(wr200, wr301).
Prereq(lit480, lit481).
Prereq(lit217, lit218).
```

Which match the new generalized query?

```
:- Prereq(C, {wr310, lit481, wr301, lit480}
```


Example (5)

Plug

```
Prereq(wr301, wr310).
Prereq(wr201, wr301).
Prereq(lit480, lit481).
```

plus

```
MajAll(eng, wr310).
MajAll(eng, lit481).
MajAll(eng, wr301).
MajAll(eng, lit480).
```

into

```
MajAll(eng, C) :- MajAll(eng, C1), Prereq(C, C1).
```

Example (6)

We get

```
MajAll(eng, wr301).
MajAll(eng, lit480).
MajAll(eng, wr200).
```

Plus we expand the generalized query to

```
:- Prereq(C,
      {wr310, lit481, wr301, lit480, wr200}).
```


Example (7)

Evaluate generalized query again

```
Prereq(wr301, wr310).
Prereq(wr301, wr418).
Prereq(wr200, wr301).
Prereq(lit480, lit481).
Prereg(lit217, lit218).
```

Which match the new generalized query?

```
:- Prereq(C,
      {wr310, lit481, wr301, lit480, wr200}).
```


Example (8)

The answers we get are the same for the last version of the generalized query:

```
Prereq(wr301, wr310).
Prereq(wr201, wr301).
Prereq(lit480, lit481).
```

So we will get no new facts nor new sub-queries.