Evaluation of Recursive Queries
Part 1: Efficient fixpoint evaluation
“Seminaïve Evaluation”
Bottom-up evaluation

- **Naïve**
  
  **Repeat**
  Apply all rules
  **Until** no new tuples generated

- **Seminaïve**
  - If a rule is applied in iteration N, at least one body fact must be a fact generated in iteration N-1 (and not before!).
  - No application is repeated.
Example

Naïve Evaluation proceed as follows:

Step(1)
sg(2,4), sg(2,5), sg(3,4), sg(3,5)

Step(2)
Iteration 1
sg(6,8), sg(6,9), sg(7,8), sg(7,9)

Iteration 2
sg(6,8), sg(6,9), sg(7,8), sg(7,9), sg(10,11)

Iteration 3
No new tuples

Seminaïve Evaluation proceed as follows:

Step(1)
sg(2,4), sg(2,5), sg(3,4), sg(3,5)

Step(2)
Iteration 1
sg(6,8), sg(6,9), sg(7,8), sg(7,9)

Iteration 2
sg(6,8), sg(6,9), sg(7,8), sg(7,9), sg(10,11)

Iteration 3
No new tuples
Notation

- Recursive Predicate
  - $p \rightarrow^* p$
- Mutually recursive predicate
  - $p \rightarrow^* q, \quad q \rightarrow^* p$
- Strongly connected component (SCC)
  - A maximal set of mutually recursive predicates.
- Linear Rule
  - Only 1 body literal is mutually recursive with head predicate.

\[
\begin{align*}
p(x) & : q1(x), q2(y). \\
qu1(x) & : q2(x). \\
qu2(x) & : q1(y), b(x,y). \\
qu2(x) & : c(x,y). \\
c(x,y) & : d(x), d(y).
\end{align*}
\]

Program Graph
- Node = SCC
- ARC : The ‘Depends on’ relation ‘→’
Seminaïve evaluation

There are two components:

- Rule Rewriting: Each rule in the program is replaced by a set of rules as follows:

\[
p : \leftarrow p_1, p_2, \ldots, p_n, q_1, q_2, \ldots, q_m
\]

is replaced by

\[
\delta p_{new}(\cdot) : \leftarrow \delta p_{1}^{old}, p_2, \ldots, p_n, q_1, q_2, \ldots, q_m.
\]

\[
\delta p_{new}(\cdot) : \leftarrow p_1^{old}, \delta p_{2}^{old}, \ldots, p_n, q_1, q_2, \ldots, q_m.
\]

\[
\vdots
\]

\[
\delta p_{new}(\cdot) : \leftarrow p_1^{old}, p_2^{old}, \ldots, p_{n-1}^{old}, \delta p_{n}^{old}, q_1, q_2, \ldots, q_m.
\]

Special case: \( n=0 \), i.e. no recursive predicates

\[
\delta p_{new}(\cdot) : \leftarrow q_1, q_2, \ldots, q_m.
\]
**Example**

\[
p(x,z) :\neg p_1(x,y), p_2(y,z), q(z,w)
\]

is replaced by

(i). \[ \delta p_{\text{new}}^{\text{old}}(x,z) :\neg \delta p_1^{\text{old}}(x,y), p_2(y,z), q(z,w). \]

(ii). \[ \delta p_{\text{new}}^{\text{old}}(x,z) :\neg p_1^{\text{old}}(x,y), \delta p_2^{\text{old}}(y,z), q(z,w). \]

**Special case – ‘Linear’ rule:**
\[
sg(x,z) :\neg u(x,v), sg(v,w), d(w,z).
\]

\[
\delta sg_{\text{new}}^{\text{old}}(x,z) :\neg u(x,v), \delta sg^{\text{old}}(v,w), d(w,z).
\]
Seminaïve evaluation – Part 2

Rule Evaluation
- Repeatedly apply rule in ‘iterations’ until no new facts.
- Iteration 1---Use all rules
- Later iterations---Use only recursive rules

In each iteration:
- Apply rules
- For each non-base predicate $p$, update associated relations as follows:

Initially: $p_i, \delta p_i^\text{old}$ and $\delta p_i^\text{new} = \emptyset$
Seminaive evaluation – Cont.

- Some observations
  - $p_i$ — All known $p_i$ facts.
  - $\delta p_i^{\text{old}}$ — $p_i$ facts (first) generated in previous iteration.
  - $\delta p_i^{\text{new}}$ — $p_i$ facts generated in this iteration.
  - $p_i^{\text{old}}$ — $p_i - \delta p_i^{\text{old}}$ (∴ generated before prev. iteration)

  NO ‘INFERANCE’ is ever repeated!

- A refinement of rule evaluation:
  - Go “node by node, bottom-up” in program graph.

  Evaluation order = 1, 2, 3, 4, 5
  or 3, 1, 2, 4, 5
Top-down evaluation

- **Given:**
  
  Call: \( Q(?) \)
  
  Rule: \( Q \text{ IF } P_1 \land P_2 \ldots \land P_n \)

  **Generate subgoals:**
  
  \( P_1(?) \ P_2(?) \ldots P_n(?) \)

- **Advantage:**
  
  - Computation is ‘focused’ in response to a query.

  - **Prolog is a language implemented in such a fashion.**
    
    - Technique is called *resolution*
Example

```
Example
up(2,1)  down(1,4)
up(3,1)  down(1,5)
up(6,2)  down(4,8)
up(7,3)  down(4,9)
up(10,7) down(9,11)

r1: sg(X,Y) :- up(X,Z), down(Z,Y)

r2: sg(X,Y) :- up(X,Z), sg(Z1,Z2), down(Z2,Y)

r3: sg(6,Y) ?
```

Prolog proceed as follows:

```
sg(6,y)?
(r1)
up(6,Z)?(Z=2); down(2,Y)? fails;
up(6,Z)? Fails on backtracking; (r1) fails.

(r2)
up(6,Z1)?(Z1=2);
sg(2,Z2)?
(r1)
up(2,Z')?(Z'=1); down(1,Y')?(Y'=Z2=4);
sg(2,Z2)? succeeds with Z2 =4;
down(4,Y)?(Y=8);
sg(6,Y)? succeeds with Y=8
```