Evaluation of Recursive Queries
Part 1: Efficient fixpoint evaluation
"Seminaïve Evaluation"

There are two components:

- Rule Rewriting: Each rule in the program is replaced by a set of rules as follows:
  \[ P \rightarrow \text{recursive w.r.t. } P \] base predicates
  is replaced by
  \[ \delta^P: p_0, p_1, \ldots, p_n \rightarrow q_0, q_1, \ldots, q_m \]
- Special case: \( n=0 \), i.e. no recursive predicates
  \[ \delta^P: () \rightarrow q_0, q_1, \ldots, q_m \]

Naïve Evaluation proceeds as follows:

- Step 1
  \[ s_{g}(2,4), s_{g}(2,5), s_{g}(3,4), s_{g}(3,5) \]
- Iteration 1
  \[ s_{g}(6,8), s_{g}(6,9), s_{g}(7,8), s_{g}(7,9) \]
- Iteration 2
  \[ s_{g}(6,10), s_{g}(7,11) \]
- Iteration 3
  No new tuples

Seminaïve Evaluation proceeds as follows:

- Step 1
  \[ s_{g}(2,4), s_{g}(2,5), s_{g}(3,4), s_{g}(3,5) \]
- Step 2
  \[ s_{g}(6,8), s_{g}(6,9), s_{g}(7,8), s_{g}(7,9) \]
- Iteration 2
  \[ s_{g}(6,10), s_{g}(7,11) \]
- Iteration 3
  No new tuples

Naïve Evaluation

- 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

- Node = SCC
  - p \((x,z)\) :- p
  - ARC: The 'Depends on' relation
  - base predicate
  - There are two components:
    - p  \((x,z)\) :- p
    - q

Program Graph

- Naïve Evaluation
- Seminaïve Evaluation

Notation

- Recursive Predicate
  - \( p \rightarrow^* p \)
- Mutually recursive predicate
  - \( p \rightarrow^* q, q \rightarrow^* p \)
- Strongly connected component (SCC)
  - A maximal set of mutually recursive predicates.
- Linear Rule
  - Only 1 body literal is mutually recursive with head predicate.
  - (i) \( p(X) \rightarrow q(X), q(Y) \)
  - (ii) \( q(X) \rightarrow q(Y) \)
  - (iii) \( q(X) \rightarrow q(Y), b(X,Y) \)
  - (iv) \( q(X) \rightarrow c(X,Y) \)
  - (v) \( q(X) \rightarrow d(X,Y) \)

Example

- \( p(x,z) \rightarrow p(x,y), p(y,z), q(z,w) \)
  - recursive w.r.t. \( p \)
  - base predicates
  - is replaced by
    - (i) \( \delta^P: p(x,y) \rightarrow p(x,y), p(y,z), q(z,w) \)
    - (ii) \( \delta^P: p(x,y) \rightarrow p(x,y), p(y,z), q(z,w) \)

Special case: Linear rule:

- Iteration 1
  \[ s_{g}(2,4), s_{g}(2,5), s_{g}(3,4), s_{g}(3,5) \]
- Iteration 2
  \[ s_{g}(6,8), s_{g}(6,9), s_{g}(7,8), s_{g}(7,9) \]
- Iteration 3
  \[ s_{g}(6,10), s_{g}(7,11) \]
- Iteration 4
  No new tuples

Example

- Iteration 1
  \[ s_{g}(2,4), s_{g}(2,5), s_{g}(3,4), s_{g}(3,5) \]
- Iteration 2
  \[ s_{g}(6,8), s_{g}(6,9), s_{g}(7,8), s_{g}(7,9) \]
- Iteration 3
  \[ s_{g}(6,10), s_{g}(7,11) \]
- Iteration 4
  No new tuples
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 $p_i$, update associated relations as follows:
  $$p_{old} = p_{old} \cup p_{new}$$

$\text{initially: } p_{old} \text{ and } p_{new} = \emptyset$

Some observations
- All known $p_i$ facts.
- $p_{old}$—$p_i$ facts (first) generated in previous iteration.
- $p_{new}$—$p_i$ facts generated in this iteration.
- NO 'INFERENCE' 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

Example
- Given: Call: Q(?)
  Rule: Q IF $P_1$ AND $P_2$ AND $P_n$
  Generate subgoals: $P_1(?)$ $P_2(?)$ ... $P_n(?)$
- Advantage:
  - Computation is ‘focused’ in response to a query.
  - Prolog is a language implemented in such a fashion.
  - Technique is called resolution

Prolong proceed as follows:
- sg(5,1) [1]
  - up(6,2), $\forall Z \forall Z_2$: down(Z, Y), $\exists Y Z$ fails.
  - up(6,2) fails on backtracking, (1) fails.
- up(6,2), $\forall Z \forall Z_2$: down(Z, Y), $\exists Y Z$ fails.
  - up(6,2) fails on backtracking, (1) fails.
- up(2,2), $\forall Z \forall Z_2$: down(Z, Y), $\exists Y Z$ fails.
  - up(2,2) fails on backtracking, (1) fails.
  - sg(2,2), $\exists Y Z$ succeeds with $Z = k$.
- down(7, Y), $\exists Y Z$ succeeds with $Y = r$.