CS-602 - Design of Problem Solvers

Chapter 4 - Query Evaluation and Optimization

Dr. Mahdi Khemakhem

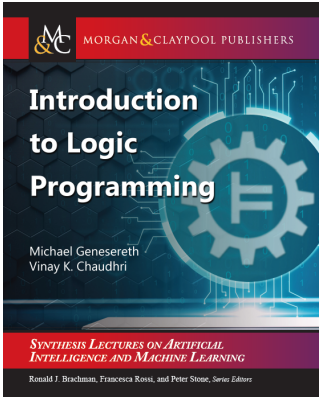
Department of Computer Science
College of Computer Engineering and Science
Prince Sattam bin Abdulaziz University

AY - 2025/2026

Outline

Materials

- **Textbook:** "Introduction to logic programming", 2022, by *Genesereth, Michael, and Vinay K. Chaudhri*
- $\Rightarrow$ **Read Chapters 5 and 6 for this chapter's material**



Outline

Preview

- **In chapter 3, we saw:**
  - Various ways of **querying** a dataset **to find just the information that we need**.
  - A way of **updating** the information in a dataset.
- **In this chapter, we look at:**
  - A way of **comparing** and **evaluating queries containing variables**. (see Section 2)
  - A variety of techniques for **optimizing queries**, **i.e., converting expensive queries into semantically equivalent queries that are computationally less complex**. (see Section 3)

Outline

Introduction

- In chapter 3, we described the *semantics of queries in terms of instances of query rules*.
  - While the definition is easy to understand and mathematically precise, *enumerating instances is not a practical method for computing answers to queries*.
- In this section, *we present an algorithm that produces the same results but in a more efficient manner*.
  - We begin this section with a discussion of *evaluation of queries without variables*.
  - In subsection 2.2, we look at a way of *comparing expressions containing variables*.
  - In subsection 2.3, we show *how to combine that technique with the procedure described here to produce an evaluation procedure for queries with variables*.
  - In subsection 2.4, we close with an *analysis of the computational complexity of our evaluation algorithm*.

Evaluating Ground Queries (1/3)

- **If a query contains multiple query rules, we check whether the body of each rule is true.**
- **The procedure for determining whether the body is true depends on the type of the body:**
  1. **If the body of a query rule is a single atom**, we check whether that atom is contained in our dataset. **If so, the body is true.**
  2. **If the body is a negated atom**, we check whether the atom is contained in our dataset. **If so, the body is false. Otherwise, the body is true.**
  3. **If the body is a conjunction of literals**, we first execute this procedure on the first conjunct. **If the answer is true, we move on to the next conjunct and so forth until we are done. If the answer to any one of the conjuncts is false, then the value of the body as a whole is false.**

Evaluating Ground Queries (2/3)

Example (1/2)

Consider the dataset shown below. There are four symbols `a`, `b`, `c`, and `d`; and there is a single binary predicate `p`. {`p(a,b)`, `p(b,c)`, `p(c,d)`}

Now, imagine that we are asked to evaluate the query shown below.

```
goal(a) :- p(a,c)
goal(b) :- p(a,b) &
p(b,a)
goal(c) :- p(c,d) &
~p(d,c)
```

1. **The body `p(a,c)` is an atom that is not in the dataset ⇒ nothing is contributed to our result.**
2. **The body `p(a,b) & p(b,a)` is a conjunction. `p(a,b)` is true but `p(b,a)` is false, so the conjunction as a whole is false ⇒ nothing is contributed to our result.**
3. **The body `p(c,d) & ~p(d,c)` is a conjunction ⇒ `p(c,d)` is true, `p(d,c)` is false so the negation is true, so the conjunction as a whole is true ⇒ we add the head of our rule `goal(c)` to our result.**

Evaluating Ground Queries (3/3)

Example (2/2)

The value of a query with multiple rules is the union of the values of each of the rules in the query.

- **Dataset:** {p(a,b), p(a,c), p(b,c), p(c,d)}
- **Query:**
  goal(a) :- p(a,b)
  goal(b) :- ∼p(b,c)
  goal(c) :- p(c,d) & ∼p(d,c)
- **Result:**
  {goal(a)} ∪ {} ∪ {goal(c)}
  {goal(a), goal(c)}

Matching

Matching is the process of **determining whether a pattern** (**an expression with or without variables**) **matches an instance** (**an expression without variables**).

- i.e., whether the two expressions can be made identical by appropriate **substitutions** for the variables in the pattern.

Substitution

A *substitution* is a **finite set of pairs** of **variables** and **terms**, called **replacements**.

$$\{X \leftarrow a,\ Y \leftarrow b\}$$

The result of applying a **substitution** σ to an **expression** Φ is the expression $\Phi\sigma$ obtained from Φ by **replacing every occurrence of every variable with a binding in the substitution by the term to which it is bound**.

$$p(X,b)\{X \leftarrow a,\ Y \leftarrow b\} = p(a,b)$$
$$q(X,Y,X)\{X \leftarrow a,\ Y \leftarrow b\} = q(a,b,a)$$

Matcher

A **substitution** σ is a *matcher* for a **pattern** and an **instance if and only if applying the substitution to the pattern results in the given instance**.

$$\texttt{p(X,b)\{X←a, Y←b\}} = \texttt{p(a,b)}$$
$$\texttt{q(X,Y,X)\{X←a, Y←b\}} = \texttt{q(a,b,a)}$$

Here, $\texttt{\{X←a, Y←b\}}$ is a **matcher** for $\texttt{p(X,b)}$ and $\texttt{p(a,b)}$.
It is also a **matcher** for $\texttt{q(X,Y,X)}$ and $\texttt{q(a,b,a)}$.

Matching Procedure (1/6)

- The **matching procedure** **assumes a representation of expressions as sequences of subexpressions**.
  - **For example**, the expression `p(X,b)` can be thought of as a sequence with three elements, namely, the **predicate** `p`, the **variable** `X`, and the **symbol** `b`.
- We start the **procedure** with two **expressions** and a **substitution** (which is initially empty). We then **recursively process the two expressions**, **comparing the subexpressions at each point**. Along the way, we **expand the substitution with variable assignments as described below** (see next slide).

Matching Procedure (2/6)

1. If the **pattern is a symbol** and the **instance is the same symbol**, then the **procedure succeeds, returning the unmodified substitution as result**.
2. If the **pattern is a symbol** and the **instance is a different symbol** or a **compound expression**, then the **procedure fails**.
3. If the **pattern is a variable with a binding**, we **compare the binding for the variable with the given instance**. If they are **identical**, the **procedure succeeds, returning the unmodified substitution as result**; **otherwise** it **fails**.
4. If the **pattern is a variable without a binding**, we **include a binding for the variable in the given instance** and we **return that substitution as a result**.
5. If the **pattern is a compound expression** and the **instance is a compound expression of the same length**, we **iterate across the pattern and the instance**.
6. If the **pattern is a compound expression** and the **instance is a symbol or a compound expression of a different length**, we **fail**.

Matching Procedure (3/6)

- If we fail to match a sub-pattern and a sub-instance at any point in this process, the procedure as a whole fails.
- If we finish this recursive comparison of the pattern and the instance, the procedure as a whole succeeds and the accumulated substitution at that point is the resulting matcher.

Matching Procedure (4/6)

Example (1/3)

As an example of this procedure in operation, consider the **process of matching** the **pattern** `p(X,Y)` and the **instance** `p(a,b)` with the **initial substitution** $\{\}$.

A trace of the execution of the procedure for this case is shown below:

```
Compare:  p(X,Y), p(a,b), {}
    Compare:  p, p, {}
    Result:  {}
    Compare:  X, a, {}
    Result:  {X ← a}
    Compare:  Y, b, {X ← a}
    Result:  {X ← a, Y ← b}
Result:  {X ← a, Y ← b}
```

We show the beginning of a comparison with a line labeled `Compare` together with the **expressions being compared** and the **input substitution**.

We show the **result of each comparison** with a line labeled `Result`. The **indentation** shows the **depth of recursion of the procedure**.

Matching Procedure (5/6)

Example (2/3)

As another example, consider the **process of matching** the **pattern** `p(X,X)` and the **instance** `p(a,a)`. A trace is shown below. By the time we compare the last arguments in the two expressions, `X` is bound to `a`, **so the match succeeds**.

```
Compare:  p(X,X), p(a,a), {}
    Compare:  p, p, {}
    Result:  {}
    Compare:  X, a, {}
    Result:  {X ← a}
    Compare:  X, a, {X ← a}
    Result:  {X ← a}
Result:  {X ← a}
```

Matching Procedure (6/6)

Example (3/3)

As a final example, consider the **process of trying to match** the **pattern** `p(X,X)` and the **expression** `p(a,b)`. By the time we reach this point, `X` is bound to `a` and the corresponding instance is `b`. **Since the pattern is a symbol and the instance is a different symbol, the match attempt fails**.

```
Compare:  p(X,X), p(a,b), {}
    Compare:  p, p, {}
    Result:  {}
    Compare:  X, a, {}
    Result:  {X ← a}
    Compare:  X, b, {X ← a}
    Result:  false
Result:  false
```

Evaluating Queries with Variables (1/7)

- **Evaluating queries with variables** is **complicated** by the fact that there can be **multiple variable bindings** that **make the body of a rule true**; and, consequently, there can be **multiple possible answers**.
  - **In some cases**, we **want just a single answer**; **in some cases**, we **want several answers**; and, **in other cases**, we **want all answers**.

$\rightarrow$In what follows, we talk about a **procedure for generating all answers**. The procedures for the other cases are analogous.

Evaluating Queries with Variables (2/7)

- **In finding the extension for an arbitrary query rule** (i.e., one with or without variables), **we start with the query rule and an empty substitution**.
- **Rather than simply checking whether the body is true or false**, as in the ground case, **we compute the set of all variable bindings for which the body is true** and, for each of these, **we include in our extension the result of applying that variable binding to the head of the rule**.
  $\rightarrow$**The procedure for computing these variable bindings depends on the type of the body of the rule.**

Evaluating Queries with Variables (3/7)

To illustrate this procedure, consider the dataset shown below.
Dataset: {p(a,b), p(a,c), p(b,c), p(c,d)}

Example: Evaluating Atoms

Consider the query goal(Y) :- p(a,Y). The pattern p(a,Y) matches two factoids in our dataset, and so there are two results goal(a), goal(c).
Call eval: p(a,Y), {}
Exit eval: {{Y←b}, {Y←c}} (two results)

Evaluating Queries with Variables (4/7)

Example: Evaluating Conjunctions (1/2)

Consider the query `goal(Y) :- p(a,Y) & p(Y,d)`. The pattern `p(a,Y)` matches just two factoids in our dataset `p(a,b)` and `p(a,c)`. Given {`Y`←`b`}, the pattern `p(Y,d)` does not match any factoids; given {`Y`←`c`}, the pattern `p(Y,d)` matches just `p(c,d)`. Thus, there is just one answer in this case.

Dataset: {`p(a,b)`, `p(a,c)`, `p(b,c)`, `p(c,d)`}

Query: `goal(Y) :- p(a,Y) & p(Y,d)`

Call eval: `p(a,Y) & p(Y,d)`, {}

Exit eval: {{`Y`←`c`}} (just one result)

Value of query: {`goal(c)`}

Evaluating Queries with Variables (5/7)

Example: Evaluating Conjunctions (2/2)

Dataset: {p(a,b), p(a,c), p(b,c), p(c,d)}

Query: goal(Y) :- p(a,Y) & p(Y,Z)

Call eval: p(a,Y) & p(Y,Z), {}

Exit eval: {{Y←b, Z←c},{Y←c,Z←d}} (two results)

Value of query: {goal(b),goal(c)}

Evaluating Queries with Variables (6/7)

Example: Evaluating Negations

Consider the queries `goal(Y) :- ~p(Y,d)` with the bindings {Y←b} and {Y←c}.
Dataset: {p(a,b), p(a,c), p(b,c), p(c,d)}
Call eval: ~p(Y,d), {Y←b}
Exit eval: {{Y←b}} (just one result)

Call eval: ~p(Y,d), {Y←c}
Exit eval: {} (no results)

Evaluating Queries with Variables (7/7)

Example: Evaluating Conjunction with Negations

Consider the query `goal(Y) :- p(a,Y) & ~p(Y,d)`. We would again find two answers to the first conjunct, namely {Y←b} and {Y←c}.

→Given the first answer to the first conjunct {Y←b}, the negation ~p(Y,d) is **satisfied**, and so the conjunction is **true**.

→Given the second answer to the first conjunct {Y←c}, the negation ~p(Y,d) **fails** and so there is no answer in this case.

Call eval: p(a,Y) & ~p(Y,d), {}
  Call eval: p(a,Y),{}
  Exit eval: {{Y←b},{Y ←c}}
  Call eval: ~p(Y,d) {Y←b}
  Exit eval: {{Y←b}} (just one result)
  Call eval: ~p(Y,d) {Y←c}
  Exit eval: {} (no results)
Exit eval: {{Y←b}} (just one result)

Computational Analysis (1/11)

One **important feature** of **query evaluation algorithm** is that **computational analysis** is **straightforward**.

Assumptions

- **All rules applied**.
- Subgoals processed left to right (**standard left-to-right implementation**).
- **No indexing of datasets** and **no caching of results once they are computed**.
  $\rightarrow$ We will **first** consider analysis with **no indexing**. **Then** we will look at analysis with **indexed data**.
- **Optimization not considered** (until next section 3):
  - Dropping redundant rules or subgoals
  - Reordering of subgoals
  - Caching

Computational Analysis (2/11)

Consider the query shown below.

```
goal(a,c) :- p(a,Y) & p(Y,c)
```

What is the cost of evaluating this query? (without indexing)

- In the worst case, there are n^2 factoids (predicates) in the database, where n is the number of ground terms in our language. So, we need n^2 steps to evaluate the first conjunct `p(a,Y)`.
- There are at most n factoids that have `a` as first argument. For each of these, we look at n^2 possibilities for the second conjunct `p(Y,c)`.

Hence, the cost of computing the instance can be expressed as follows: $n^2 + n \times n^2 = n^2 + n^3$ (**For $n = 3$ we have 36 unifications**).

Computational Analysis (3/11)

Now consider the general version of this query shown below.

```
goal(X,Z) :- p(X,Y) & p(Y,Z)
```

What is the cost of computing all answers to this query? (without indexing)

- In the worst case, there are n^2 facts in the database, where n is the number of objects in the domain. So, we need n^2 steps to evaluate the first subgoal `p(X,Y)`.
- There are at most n^2 possible values for `Y`. For each of these, we look at n^2 possibilities for the second subgoal `p(Y,Z)`.

The resulting cost is shown as follows: $n^2 + n^2 \times n^2 = n^2 + n^4$ (**For** $n = 3$ **we have 90 unifications**).

Computational Analysis (4/11)

Full Indexing

In **full indexing**, each factoid appears on the list of factoids is associated with each constant in that factoid.

Example: {p(a,b), p(b,c), q(b), q(c)}
Index on p: {p(a,b), p(b,c)}
Index on q: {q(b), q(c)}
Index on a: {p(a,b)}
Index on b: {p(a,b), p(b,c), q(b)}
Index on c: {p(b,c), q(c)}

Computational Analysis (5/11)

Full Indexing/Worst Case

Example: {p(a,a), p(b,a), p(c,a), p(a,b) p(b,b), p(c,b), p(a,c) p(b,c) p(c,c)}
Index on p: {p(a,a),..., p(c,c)}
Index on a: {p(a,a), p(a,b), p(a,c), p(b,a), p(c,a)}
Index on b: {p(a,b), p(b,a), p(b,b), p(b,c), p(c,b)}
Index on c: {p(a,c), p(b,c), p(c,a), p(c,b), p(c,c)}

Computational Analysis (6/11)

Example with Indexing (1/2)

```
goal(a,c) :- p(a,Y) & p(Y,c)
```

- **Cost of computing `goal(a,c)` without indexing**

$$n^2 + n \times n^2 = n^2 + n^3$$

For $n = 3$ we have 36 unifications.

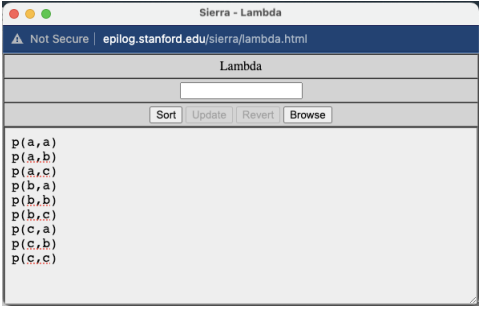
- **Cost of computing `goal(a,c)` with indexing**

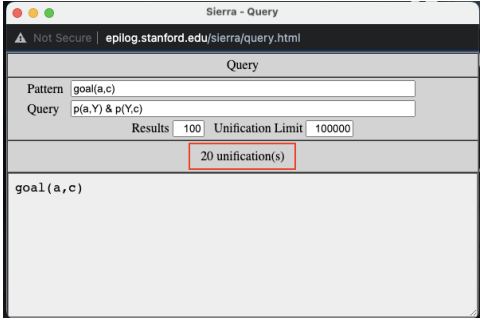
$$(2n - 1) + n \times (2n - 1) = 2n^2 + n - 1$$

For $n = 3$ we have 20 unifications.

Computational Analysis (7/11)

Example with Indexing (1/2)





Computational Analysis (8/11)

Example with Indexing (2/2)

```
goal(X,Z) :- p(X,Y) & p(Y,Z)
```

- **Cost of computing `goal(X,Z)` without indexing**

$$n^2 + n^2 \times n^2 = n^2 + n^4$$

For $n = 3$ we have 90 unifications.

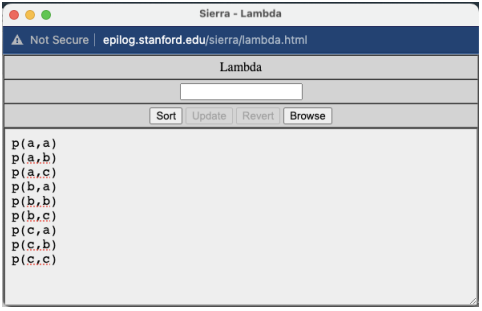
- **Cost of computing `goal(X,Z)` with indexing**

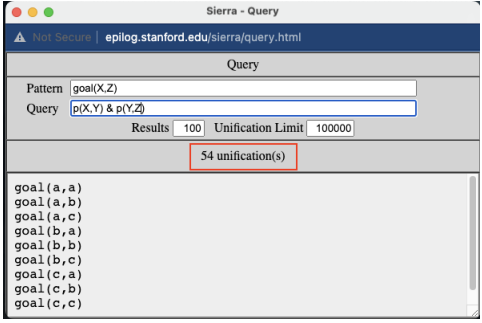
$$n^2 + n^2 \times (2n - 1) = n^2 + 2n^3 - n^2 = 2n^3$$

For $n = 3$ we have 54 unifications.

Computational Analysis (9/11)

Example with Indexing (2/2)





Computational Analysis (10/11)

It is instructive to **compare** the **cost of computing answers** using the present algorithm with the **cost of computing answers** by enumerating all instances of rules and, for each such instance, checking whether the body of each such instance is true or false (**in accordance with the semantics described in the preceding chapter**).

Computational Analysis (11/11)

- In the preceding example, the query has just three variables. Consequently, for a domain with n objects, there are n^3 instances.
- To evaluate each of these instances, we must compare each of our two subgoals to every factoid in our dataset, and in the worst case there are n^2 of these.

The overall cost is shown below.

$$n^3 \times (n^2 + n^2) = n^5 + n^5 = 2n^5$$

Our algorithm is clearly better in this case, and the relative benefits are greater when we consider sparse datasets[1].

→In such cases, the **"semantics-based" algorithm** must still look at all instances, but **our algorithm** needs to look at only those instances that are derived form factoids in the dataset.

[1] Datasets that do not include all possible factoids

Key Takeaways

Key Takeaways

- **Pattern matching** is the process of finding a **matcher** that makes a **pattern identical to an instance**.
- A **matching procedure** can be defined that recursively compares the subexpressions of the pattern and the instance, building up a matcher along the way.
- **Evaluating queries with variables** involves finding all variable bindings that make the body of a rule true, and applying these bindings to the head of the rule to generate answers.
- The **computational analysis** of query evaluation is straightforward, and indexing can significantly improve performance.

Exercise 4.1

For each of the following patterns and instances, say whether the instance matches the pattern and, if so, give the corresponding matcher.

1. `p(X,Y)` and `p(a,a)`
2. `p(X,Y)` and `p(a,f(a))`
3. `p(X,f(Y))` and `p(a,f(a))`
4. `p(X,X)` and `p(2,min(2,4))`
5. `p(X,min(2,4))` and `p(2,2)`

Solution 4.1

1. Yes, {X←a, Y←a}
2. Yes, {X←a, Y←f(a)}
3. Yes, {X←a, Y←a}
4. No
5. No

Exercise 4.2

Suppose that we want to find all `goal(X,Y,Z)` such that `p(X,Y) & q(Y,Z)`.
Select the formula that captures the worst case complexity of our standard evaluation algorithm for this query (assuming no indexing of the dataset). The symbol n here represents the total number of objects in the domain.

1. $2n^2 + 2n^3$
2. $2n^2 + 2n^4$
3. $2n^2 + n^3 + n^4$

$\rightarrow$**We have $2n^2$ entries (facts) in the dataset. The worst case total cost $2n^2 + n^2 \times 2n^2 = 2n^2 + 2n^4$**

Exercise 4.3

For each of the queries shown below, select the expression that captures the worst case complexity of our standard evaluation algorithm without indexing. The symbol n represents the total number of objects in the domain.

```
goal(X,Y) :- p(X,Y) & q(Y)
```

1. $n^4 + n^3 + n^2 + n$
2. $2n^4 + 2n^3 + n^2 + n$
3. $2n^4 + 2n^3$

$\rightarrow$We have $n^2 + n$ entries (facts) in the dataset. The worst case total cost
$(n^2 + n) + n^2 \times (n^2 + n) = n^2 + n + n^4 + n^3$

```
goal(X,Y) :- p(X,Y) & q(Y) & q(Z)
```

1. $n^4 + n^3 + n^2 + n$
2. $2n^4 + 2n^3 + n^2 + n$
3. $2n^4 + 2n^3$

$\rightarrow$We have $n^2 + n$ entries (facts) in the dataset. The worst case total cost
$(n^2 + n) + n^2 \times ((n^2 + n) + 1 \times (n^2 + n))$
$= n^2 + n + n^4 + n^3 + n^4 + n^3 =$
$2n^4 + 2n^3 + n^2 + n$

Warning

Homework 4.1 should be submitted on Blackboard before the next course session!

Homework 4.1

Given the following Dataset: {p(a,a), p(a,b), p(a,c), p(b,a), p(b,b), p(b,c), p(c,a), p(c,b), p(c,c), q(a), q(b), q(c)}

1. Test the two queries seen in Exercise 4.3 using the Sierra IDE.
2. How many unifications have been performed (**with indexing**)? Give the result screenshot for each query.
3. For each query, compare the number of obtained unifications with the number of unifications **without indexing**.

Outline

Introduction

- **Two queries** are **semantically equivalent if and only if they produce identical results for a given dataset**.
  - In such cases, **it does not matter which query we write if all we care about is getting the right answers!**
  - On the other hand, **it is possible that one query is computationally better than another!**
- In this section, **we look at a variety of techniques for optimizing queries**, i.e., *converting expensive queries into semantically equivalent queries that are computationally less complex.*

Subgoal Ordering (1/7)

- One very common source of **inefficiency in evaluation** stems from **non-optimal ordering of subgoals within queries**.
- Frequently, **it is possible to find better orderings** just by **looking at the form of the queries involved** even without looking at the data to which the queries are applied.

Subgoal Ordering (2/7)

Example (1/3)

As an example of inefficiency due to poor subgoal ordering, consider the query shown below.

```
goal(X,Y) :- p(X) & r(X,Y) & q(X)
```

To evaluate this query, our algorithm would first examine all $n^2 + 2n$ facts to find those that match `p(X)`. There would be at most n answers. For each of these, the algorithm would again look at $n^2 + 2n$ facts to find those that match `r(X,Y)`. **There would be n for each value of `X`.** Finally, for each of these pairs, the algorithm would again examine at $n^2 + 2n$ facts to find those that match `q(X)`. The grand total is shown below.

$$(n^2 + 2n) + n \times ((n^2 + 2n) + \mathbf{n} \times (n^2 + 2n)) = n^4 + 3n^3 + 3n^2 + 2n$$

$\rightarrow$ **For $n = 3$ we have 195 unifications without indexing.**

Subgoal Ordering (3/7)

Example (2/3)

If we change the order of the last query conditions by moving `q` before `r` we obtain the the query shown below.

```
goal(X,Y) :- p(X) & q(X) & r(X,Y)
```

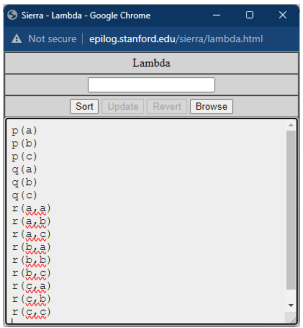
To evaluate this query, our algorithm would again examine all $n^2 + 2n$ facts to find those that match `p(X)`. There would be at most n answers. For each of these, the algorithm would again look at $n^2 + 2n$ facts to find those that match `q(X)`. **There would be at least one for each value of** `X`. Finally, for each such `X`, the algorithm would examine $n^2 + 2n$ facts to find those that match `r(X,Y)`. The grand total is shown below.

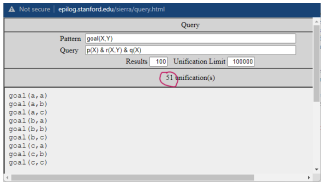
$$(n^2 + 2n) + n \times ((n^2 + 2n) + \mathbf{1} \times (n^2 + 2n)) = 2n^3 + 5n^2 + 2n$$

$\rightarrow$ **For $n = 3$ we have 105 unifications without indexing.**

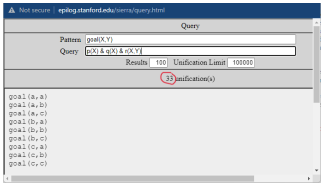
Subgoal Ordering (4/7)

Example (3/3)





→ **We have 51 unifications with indexing.**



→ **We have 33 unifications with indexing.**

Subgoal Ordering (5/7)

- A **simple method for reordering subgoals** consists to **assemble a new body** for the query **incrementally**, →*picking* a subgoal on each step and *removing* it from the list of remaining subgoals to be considered.
- **To make choice**, the method examines the **remaining subgoals in left-to-right order**.
  - If **it encounters a subgoal with all variables bounded by subgoals already chosen**, then **that subgoal is added to the new query and removed from the list of remaining subgoals**.
  - If **not**, **the method removes the first remaining subgoal from the list, adds it to the new query, updates its list of bound variables, and moves on to the next step**.

Subgoal Ordering (6/7)

Example/ Reordering Method (1/2)

Consider the first query of the last example `goal(X,Y) :- p(X) & r(X,Y) & q(X)`

- At the start, the list of remaining subgoals consists of all three subgoals in the query.
  - RemainingSubgoals:{`p(X), r(X,Y), q(X)`}
  - NewQuery:`newgoal(X,Y) :-`
  - BoundedVariables:{}
- At this point, none of the remaining three subgoals is ground, so the method chooses the first subgoal `p(X)`, adds it to its new query, and puts `X` on the list of bound variables.
  - RemainingSubgoals:{`r(X,Y), q(X)`}
  - NewQuery:`newgoal(X,Y) :- p(X)`
  - BoundedVariables:{`X`}

Subgoal Ordering (7/7)

Example/ Reordering Method (2/2)

- On the second step, the method looks at the remaining two subgoals. The subgoal `r(X,Y)` contains the unbound variable `Y` and so it is not chosen. By contrast, all of the variables in the subgoal `q(X)` are bound, so the method outputs this subgoal next. On the third step, the final subgoal is added forming the second query shown above.
  - RemainingSubgoals:{`r(X,Y)`}
  - NewQuery:`newgoal(X,Y) :- p(X) & q(X)`
  - BoundedVariables:{`X`}
- On the third step, the final subgoal is added forming the second query shown above.
  - RemainingSubgoals:{}
  - NewQuery:`newgoal(X,Y) :- p(X) & q(X) & r(X,Y)`
  - BoundedVariables:{`X, Y`}

Subgoal Removal (1/7)

Another **common source of inefficiency in evaluation** stems from the presence of **redundant subgoals** within queries.

- In many cases, **it is possible to detect and eliminate such redundancies.**

Subgoal Removal (2/7)

Example

- As a simple example of the problem, consider the query shown below.

```
goal(X,Y) :- p(X,Y) & q(Y) & q(Z)
```

Relation goal is true of `X` and `Y` if `p` is true of `X` and `Y` and `q` is true of `Y` and `q` is also true of some `Z`.

- It should be clear that the subgoal `q(Z)` is redundant here. If there is a value for `Y` such that `q(Y)` is true, then that value for `Y` also works as a value for `Z`. Consequently, we can drop the `q(Z)` subgoal (while retaining `q(Y)`), resulting in the query shown below.

```
goal(X,Y) :- p(X,Y) & q(Y)
```

Subgoal Removal (3/7)

In general, it is easy to **determine which subgoals can be removed** and **which need to be retained**.

- The basic idea is to **assemble a new body for the query incrementally**, picking a subgoal on each step, checking for redundancy, and adding the subgoal once if it is not redundant.

Subgoal Removal (4/7)

Example/ Redundancy Elimination (1/2)

As an illustration of this method in action, consider the following query: `goal(X,Y) :- p(X,Y) & q(Y) & q(Z)`

- The method first computes the variables in the head of the query, namely `[X,Y]` and initializes the variable `newquery` to a new query with the same head as the original query.
- It then iterates over the body of the query, adding subgoals to the new query once they are checked for redundancy.
- On the first step of the iteration, the method focuses on `p(X,Y)`. It creates a dataset consisting of instances of the other subgoals, namely `q(x1)` and `q(x2)`; it then tries to derive `p(x0,x1)`; and, in this case, it fails. So `p(X,Y)` is added to the new query (`newquery=p(X,Y)`).

Subgoal Removal (5/7)

Example/ Redundancy Elimination (2/2)

- On the second step, the method focuses on `q(Y)`. It creates a dataset consisting of instances of the other subgoals, namely `p(x0,x1)` and `q(x2)`; it then tries to derive `q(x1)`; and, once again, it fails. So `q(Y)` is added to the new query (`newquery=p(X,Y) & q(Y)`).
- Finally, the method focuses on `q(Z)`. It creates a dataset consisting of instances of the other subgoals, namely `p(x0,x1)` and `q(x1)`; it then tries to derive an instance of `q(Z)`. Note that `Z` is not bound here since it does not occur as a head variable or a variable in any of the other subgoals. In this case, the test succeeds; and so `q(Z)` is not added to the new query.

Subgoal Removal (6/7)

This method is sound in that **it removes only redundant subgoals**.

- As a result, **any query produced by this method is equivalent to the query it is given as input**.
- **Unfortunately, the method is not complete**.
  - There are redundant subgoals that **it is does not detect**.
  - The problem arises when **multiple redundant subgoals share variables** that prevent the method from detecting the redundancy.

Subgoal Removal (7/7)

Example/ Redundancy Elimination Failure!

As an illustration of the limitation of redundancy elimination method, consider the following query:

```
goal(X) :- p(X,Y) & q(X,Y) & p(X,Z) & q(X,Z)
```

The relation goal is true of `X` if `p` is true of `X` and `Y` and `q` is true of `X` and `Y` and `p` is true of `X` and `Z` and `q` is true of `X` and `Z`.

- **Clearly, the last two subgoals are redundant with the first two subgoals.**
- Unfortunately, our method does not detect that either of the subgoals in either pair is redundant because of the variables shared with the other subgoal of the pair. Try it.

→Detecting this sort of redundancy can be done mechanically by considering subsets of subgoals and not just individual subgoals.

Rule Removal (1/2)

An analogous **form of inefficiency in evaluation** stems from the presence of **redundant rules**.

- **As with redundant subgoals within rules**, it is often easy to **detect and eliminate such redundancies**.

Rule Removal (2/2)

Example/ Redundant Rule

As an example of the problem, consider the rules shown below.

```
goal(X) :- p(X,b) & q(b) & r(Z)
goal(X) :- p(X,Y) & q(Y)
```

Relation `goal` is true of `X` if `p` is true of `X` and `Y` and `q` is true of `b` and `r` is true of `Z`.
Relation `goal` is true of `X` if `p` is true of `X` and `Y` and `q` is true of `Y`.

- Any answer produced by the first rule here is also produced by the second rule, so the **first rule is redundant and can be eliminated**.
- The trick to detecting such redundancies is to recognize that the **second rule subsumes the first**, i.e., *all of the answers produced by the first rule are produced by the second rule*.

Example (*SEND + MORE = MONEY*)

The equation "*SEND + MORE = MONEY*" is a classic example of a cryptarithmetic puzzle, **where each letter represents a digit**, and we need to find a **unique digit for each letter** such that the equation holds true.

In this case, we are trying to find values for S, E, N, D, M, O, R, Y such that the sum of *SEND* and *MORE* equals *MONEY*.

```
  S E N D
+ M O R E
---------
M O N E Y
```

Here's one possible solution $S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2$ where the equation holds true:

```
  9 5 6 7
+ 1 0 8 5
---------
1 0 6 5 2
```

The equation is satisfied with these values, and each letter represents a unique digit.

Example ($SEND + MORE = MONEY$)

We can formulate this problem up as a query. First, we have a dataset listing the allowable digits.

{digit(1), digit(2), digit(3), digit(4), digit(5), digit(6), digit(7), digit(8), digit(9), digit(0)}

We use 8 variables S,E,N,D,M,O,R, and Y in the goal where each variable should receive a distinct digit.

Example (*SEND + MORE = MONEY*)

We write a query listing the **eight letters as goal values**, with subgoals to fix the **ranges of these variables**, **disjointness constraints**, and **additional constraints** to capture the arithmetic conditions.

```
puzzle(S,E,N,D,M,O,R,Y):- digit(S) & digit(E) & digit(N) & digit(D) & digit(M) & digit(O) &
digit(R) & digit(Y) &distinct(S,O) &distinct(E,S) &
distinct(N,S) & distinct(N,E) &distinct(D,S) & distinct(D,E) & distinct(D,N) &
distinct(M,O) & distinct(M,S) & distinct(M,E) & distinct(M,N) & distinct(M,D) &
distinct(O,S) & distinct(O,E) & distinct(O,N) & distinct(O,D) & distinct(O,M) &
distinct(R,S) & distinct(R,E) & distinct(R,N) & distinct(R,D) & distinct(R,M) &
distinct(R,O) &distinct(Y,S) & distinct(Y,E) & distinct(Y,N) & distinct(Y,D) &
distinct(Y,M) & distinct(Y,O) & distinct(Y,R) &
evaluate(plus(times(S,1000),times(E,100),times(N,10),D),SEND) &
evaluate(plus(times(M,1000),times(O,100),times(R,10),E),MORE) &
evaluate(plus(times(M,10000),times(O,1000),times(N,100),times(E,10),Y),MONEY) &
evaluate(plus(SEND,MORE),MONEY)
```

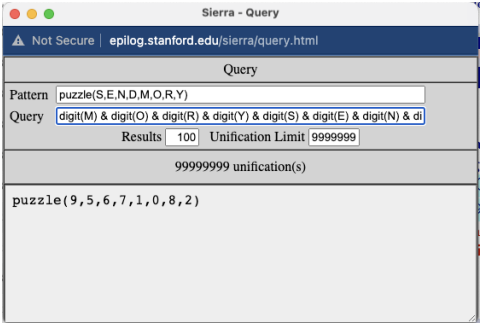
Example (*SEND + MORE = MONEY*)

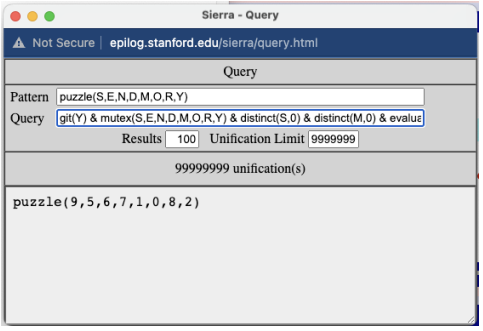
The last query can be simplified as follows:

```
puzzle(S,E,N,D,M,O,R,Y):-
digit(S) & digit(E) & digit(N) & digit(D) &
digit(M) & digit(O) & digit(R) & digit(Y) &
mutex(S,E,N,D,M,O,R,Y) &
distinct(S,O) & distinct(M,O) &
evaluate(plus(times(S,1000),times(E,100),times(N,10),D),SEND) &
evaluate(plus(times(M,1000),times(O,100),times(R,10),E),MORE) &
evaluate(plus(times(M,10000),times(O,1000),times(N,100),times(E,10),Y),MONEY) &
evaluate(plus(SEND,MORE),MONEY)
```

Given the way this query is written, the **evaluation process will take a long time**. $\rightarrow$There are $10^8 = 100,000,000$ possible variable bindings.

Example (*SEND + MORE = MONEY*)





Example (*SEND + MORE = MONEY*)

We can use subgoal ordering to transform this query into one that is easier to evaluate. We simply move the inequalities up to the point where the variables are bound. See below.

```
puzzle(S,E,N,D,M,O,R,Y):- digit(S) & distinct(S,0) & digit(E) & distinct(E,S) &
digit(N) & distinct(N,S) & distinct(N,E) &  digit(D) & distinct(D,S) & distinct(D,E) &
distinct(D,N) &
digit(M) & distinct(M,0) & distinct(M,S) & distinct(M,E) & distinct(M,N) &distinct(M,D) &
digit(O) & distinct(O,S) & distinct(O,E) & distinct(O,N) & distinct(O,D) &distinct(O,M) &
digit(R) & distinct(R,S) & distinct(R,E) & distinct(R,N) & distinct(R,D) &distinct(R,M) &
distinct(R,O) &
digit(Y) & distinct(Y,S) & distinct(Y,E) & distinct(Y,N) & distinct(Y,D) & distinct(Y,M) &
distinct(Y,O) &
distinct(Y,R) &
evaluate(plus(times(S,1000),times(E,100),times(N,10),D),SEND) &
evaluate(plus(times(M,1000),times(O,100),times(R,10),E),MORE) &
evaluate(plus(times(M,10000),times(O,1000),times(N,100),times(E,10),Y),MONEY) &
evaluate(plus(SEND,MORE),MONEY)
```

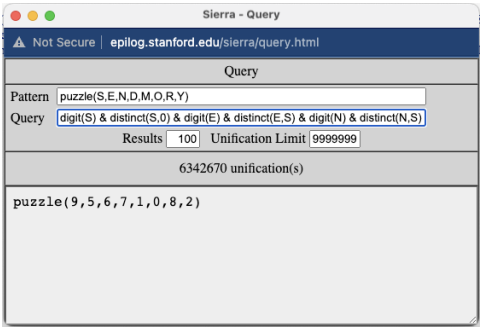
Having done this, we eliminate many of the possibilities before they are even

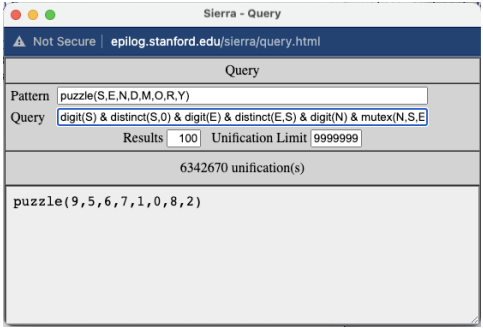
Example (*SEND + MORE = MONEY*)

The last query can be simplified as follows:

```
puzzle(S,E,N,D,M,O,R,Y):-
digit(S) & distinct(S,O) &
digit(E) & distinct(E,S) &
digit(N) & mutex(N,S,E) &
digit(D) & mutex(D,S,E,N) &
digit(M) & distinct(M,O) & mutex(M,S,E,N,D) &
digit(O) & mutex(O,S,E,N,D,M) &
digit(R) & mutex(R,S,E,N,D,M,O) &
digit(Y) & mutex(Y,S,E,N,D,M,O,R) &
evaluate(plus(times(S,1000),times(E,100),times(N,10),D),SEND) &
evaluate(plus(times(M,1000),times(O,100),times(R,10),E),MORE) &
evaluate(plus(times(M,10000),times(O,1000),times(N,100),times(E,10),Y),MONEY) &
evaluate(plus(SEND,MORE),MONEY)
```

Example (*SEND + MORE = MONEY*)





Key Takeaways

Key Takeaways

- **The order of subgoals** in a query can have a **significant impact** on the efficiency of evaluation.
- **Reordering subgoals** to place more **selective subgoals earlier** in the query can **reduce the number of unifications** needed during evaluation.
- **Redundant subgoals** within queries can **increase evaluation time** without affecting the results.
- **Eliminating redundant subgoals** can **improve efficiency**.
- **Redundant rules** can also **slow down evaluation**.
- **Detecting and removing redundant rules** can **streamline the evaluation process**.

Exercise 4.4

For the following group of query rules, say which rule is best in terms of worst case evaluation complexity using our standard algorithm without indexing.

1. `goal(X,Y,Z) :- p(X,Y) & q(X,X) & r(X,Y,Z)`
2. `goal(X,Y,Z) :- p(X,Y) & r(X,Y,Z) & q(X,X)`
3. `goal(X,Y,Z) :- q(X,X) & p(X,Y) & r(X,Y,Z)`
4. `goal(X,Y,Z) :- q(X,X) & r(X,Y,Z) & p(X,Y)`
5. `goal(X,Y,Z) :- r(X,Y,Z) & p(X,Y) & q(X,X)`
6. `goal(X,Y,Z) :- r(X,Y,Z) & q(X,X) & p(X,Y)`

Solution 4.4

The best rule is the third one: `goal(X,Y,Z) :- p(X,X) & p(X,Y) & r(X,Y,Z)`

- The first subgoal `p(X,X)` binds `X` to a single value.
- The second subgoal `p(X,Y)` binds `Y` to a single value (since `X` is already bound).
- The third subgoal `r(X,Y,Z)` binds `Z` to a single value (since both `X` and `Y` are already bound).

Exercise 4.5

For each of the following groups of query rules, select the alternative that is equivalent to the first rule in the group.

1.
```
goal(X) :- p(X,Y) & q(Y) & q(Z)
goal(X) :- p(X,Y)
goal(X) :- p(X,Y) & q(Y)
goal(X) :- p(X,Y) & q(Z)
```
2.
```
goal(X) :- p(X) & q(X) & q(W)
goal(X) :- p(X)
goal(X) :- p(X) & q(X)
goal(X) :- p(X) & q(W)
```
3.
```
goal(X,Y,Z) :- p(X,Y) & q(Y) & q(Z) & q(W)
goal(X,Y,Z) :- p(X,Y) & q(Y) & q(Z)
goal(X,Y,Z) :- p(X,Y) & q(Y) & q(W)
goal(X,Y,Z) :- p(X,Y) & q(Z) & q(W)
```

Solution 4.5

1. The third rule is equivalent to the first one: `goal(X) :- p(X,Y) & q(Y)`
2. The third rule is equivalent to the first one: `goal(X) :- p(X) & q(X)`
3. The second rule is equivalent to the first one: `goal(X,Y,Z) :- p(X,Y) & q(Y) & q(Z)`

Exercise 4.6

For each of the following pairs of queries, say whether the first query subsumes the second, i.e., whether the set of answers to the first query contains the answers to the second query.

1. ```
   goal(X) :- p(X,Y)
   goal(X) :- p(X,a) & p(X,b)
   ```
2. ```
   goal(X) :- p(X,a)
   goal(X) :- p(X,Y)
   ```
3. ```
   goal(X) :- p(X,Y) & p(X,Z)
   goal(X) :- p(X,b) & q(b)
   ```


   ```

Solution 4.6

1. Yes, the first query subsumes the second one.
2. No, the first query does not subsume the second one.
3. No, the first query does not subsume the second one.

Warning

Homework 4.2 should be submitted on Blackboard before the next course session!

Homework 4.2

For the following group of query rules, say which rule is best in terms of worst case evaluation complexity using our standard algorithm without indexing.

1. `goal(X,Y,Z) :- p(X,Y) & q(a,b) & r(X,Y,Z)`
2. `goal(X,Y,Z) :- p(X,Y) & r(X,Y,Z) & q(a,b)`
3. `goal(X,Y,Z) :- q(a,b) & p(X,Y) & r(X,Y,Z)`
4. `goal(X,Y,Z) :- q(a,b) & r(X,Y,Z) & p(X,Y)`
5. `goal(X,Y,Z) :- r(X,Y,Z) & p(X,Y) & q(a,b)`
6. `goal(X,Y,Z) :- r(X,Y,Z) & q(a,b) & p(X,Y)`

End of Chapter 4