61A Lecture 31

Wednesday, November 20

Announcements

Announcements

- Project 4 due Thursday 11/21 @ 11:59pm.

Announcements

- Project 4 due Thursday 11/21 @ 11:59pm.
- Extra reader office hours in 405 Soda this week.

Announcements

- Project 4 due Thursday 11/21 @ 11:59pm.
- Extra reader office hours in 405 Soda this week.
  - Wednesday: 5:30pm-7pm
  - Thursday: 5:30pm-7pm

Announcements

- Project 4 due Thursday 11/21 @ 11:59pm.
- Extra reader office hours in 405 Soda this week.
  - Wednesday: 5:30pm-7pm
  - Thursday: 5:30pm-7pm
- Homework 10 due Tuesday 11/26 @ 11:59pm.

Announcements

- Project 4 due Thursday 11/21 @ 11:59pm.
- Extra reader office hours in 405 Soda this week.
  - Wednesday: 5:30pm-7pm
  - Thursday: 5:30pm-7pm
- Homework 10 due Tuesday 11/26 @ 11:59pm.
- Recursive art contest entries will be due Monday 12/2 @ 11:59pm (After Thanksgiving).

Declarative Languages

Databases

Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases store tables and have have methods for adding, editing, and retrieving records.

http://www.headfirstlabs.com/sql_hands_on/

Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases store tables and have have methods for adding, editing, and retrieving records.

The Structured Query Language (SQL) is perhaps the most widely used programming language.

Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases store tables and have have methods for adding, editing, and retrieving records.

The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

http://www.headfirstlabs.com/sql_hands_on/

Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases store tables and have have methods for adding, editing, and retrieving records.

The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

toy_id	toy	color	cost	weight
2	whiffleball	yellow	2.20	0.40
5	frisbee	yellow	1.50	0.20
10	yoyo	yellow	1.50	0.20

http://www.headfirstlabs.com/sql_hands_on/

Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases store tables and have have methods for adding, editing, and retrieving records.

The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

toy_id	toy	color	cost	weight
2	whiffleball	yellow	2.20	0.40
5	frisbee	yellow	1.50	0.20
10	yoyo	yellow	1.50	0.20

Each row is a record

http://www.headfirstlabs.com/sql_hands_on/

Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases store tables and have have methods for adding, editing, and retrieving records.

The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

toy_id	toy	color	cost	weight
2	whiffleball	yellow	2.20	0.40
5	frisbee	yellow	1.50	0.20
10	yoyo	yellow	1.50	0.20

Each row is a record

SQL is an example of a declarative programming language.

Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases store tables and have have methods for adding, editing, and retrieving records.

The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

toy_id	toy	color	cost	weight
2	whiffleball	yellow	2.20	0.40
5	frisbee	yellow	1.50	0.20
10	yoyo	yellow	1.50	0.20

Each row is a record

SQL is an example of a declarative programming language.

It separates *what* to compute from *how* it is computed.

Databases

A table is a collection of records, which are tuples of values organized in columns.

Databases store tables and have have methods for adding, editing, and retrieving records.

The Structured Query Language (SQL) is perhaps the most widely used programming language.

```
SELECT * FROM toy_info WHERE color='yellow';
```

toy_id	toy	color	cost	weight
2	whiffleball	yellow	2.20	0.40
5	frisbee	yellow	1.50	0.20
10	yoyo	yellow	1.50	0.20

Each row is a record

SQL is an example of a declarative programming language.

It separates *what* to compute from *how* it is computed.

The language interpreter is free to compute the result in any way it wants.

http://www.headfirstlabs.com/sql_hands_on/

Declarative Programming

Declarative Programming

Characteristics of **declarative languages:**

Declarative Programming

Characteristics of **declarative languages:**

- A "program" is a description of the desired solution.

Declarative Programming

Characteristics of **declarative languages:**

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

Declarative Programming

Characteristics of **declarative languages:**

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:

Declarative Programming

Characteristics of **declarative languages:**

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes.

Declarative Programming

Characteristics of **declarative languages:**

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes.
- The interpreter carries out execution and evaluation rules.

Declarative Programming

Characteristics of **declarative languages:**

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes.
- The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.

Declarative Programming

Characteristics of **declarative languages**:

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes.
- The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.

Declarative languages often handle only some subset of problems.

Declarative Programming

Characteristics of **declarative languages:**

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes.
- The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.

Declarative languages often handle only some subset of problems.

Declarative Programming

Characteristics of **declarative languages**:

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes.
- The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.

Declarative languages often handle only some subset of problems.

Solve cool problems

as long as they are small

Declarative Programming

Characteristics of **declarative languages**:

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes.
- The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.

Declarative languages often handle only some subset of problems.

Solve cool problems *Limited problem solving*

as long as they are small *on large-scale datasets*

Declarative Programming

Characteristics of **declarative languages**:

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes.
- The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.

Declarative languages often handle only some subset of problems.

Declarative Programming

Characteristics of **declarative languages**:

- A "program" is a description of the desired solution.
- The interpreter figures out how to generate such a solution.

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes.
- The interpreter carries out execution and evaluation rules.

Building a universal problem solver is hard.

Declarative languages often handle only some subset of problems.

The Logic Language

The Logic Language

http://awhimsicalbohemian.typepad.com/.a/6a00e5538b84f3883301538dfa8f19970b-800wi

The Logic Language

The *Logic* language is invented for this course.

The Logic Language

The *Logic* language is invented for this course.

- Based on the Scheme project with ideas from Prolog (1972).

The Logic Language

The *Logic* language is invented for this course.

- Based on the Scheme project with ideas from Prolog (1972).
- Expressions are facts or queries, which contain relations.

http://awhimsicalbohemian.typepad.com/.a/6a00e5538b84f3883301538dfa8f19970b-800wi

The Logic Language

The *Logic* language is invented for this course.

- Based on the Scheme project with ideas from Prolog (1972).
- Expressions are facts or queries, which contain relations.
- Expressions and relations are Scheme lists.

http://awhimsicalbohemian.typepad.com/.a/6a00e5538b84f3883301538dfa8f19970b-800wi

The Logic Language

The *Logic* language is invented for this course.

- Based on the Scheme project with ideas from Prolog (1972).
- Expressions are facts or queries, which contain relations.
- Expressions and relations are Scheme lists.
- For example, **(likes john dogs)** is a relation.

http://awhimsicalbohemian.typepad.com/.a/6a00e5538b84f3883301538dfa8f19970b-800wi

The Logic Language

The *Logic* language is invented for this course.

- Based on the Scheme project with ideas from Prolog (1972).
- Expressions are facts or queries, which contain relations.
- Expressions and relations are Scheme lists.
- For example, **(likes john dogs)** is a relation.
- Implementation fits on a single sheet of paper (next lecture).

http://awhimsicalbohemian.typepad.com/.a/6a00e5538b84f3883301538dfa8f19970b-800wi

The Logic Language

The *Logic* language is invented for this course.

- Based on the Scheme project with ideas from Prolog (1972).
- Expressions are facts or queries, which contain relations.
- Expressions and relations are Scheme lists.
- For example, **(likes john dogs)** is a relation.
- Implementation fits on a single sheet of paper (next lecture).

Today's theme:

http://awhimsicalbohemian.typepad.com/.a/6a00e5538b84f3883301538dfa8f19970b-800wi

The Logic Language

The *Logic* language is invented for this course.

- Based on the Scheme project with ideas from Prolog (1972).
- Expressions are facts or queries, which contain relations.
- Expressions and relations are Scheme lists.
- For example, **(likes john dogs)** is a relation.
- Implementation fits on a single sheet of paper (next lecture).

Today's theme:

http://awhimsicalbohemian.typepad.com/.a/6a00e5538b84f3883301538dfa8f19970b-800wi

Simple Facts

Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:

Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:

- A relation is a Scheme list.

Simple Facts

A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:

- A relation is a Scheme list.
- A fact expression is a Scheme list of relations.

Simple Facts

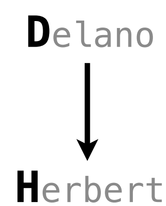
A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:

- A relation is a Scheme list.
- A fact expression is a Scheme list of relations.

```
logic> (fact (parent delano herbert))
```



Simple Facts

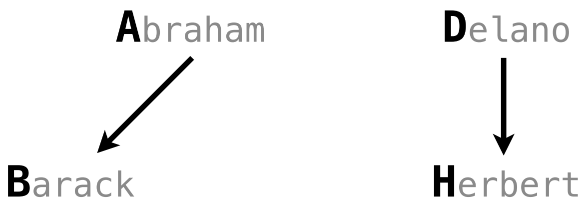
A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:

- A relation is a Scheme list.
- A fact expression is a Scheme list of relations.

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
```



Simple Facts

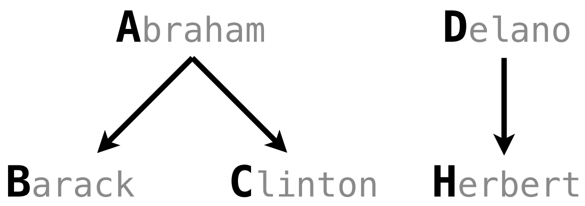
A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:

- A relation is a Scheme list.
- A fact expression is a Scheme list of relations.

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
```



Simple Facts

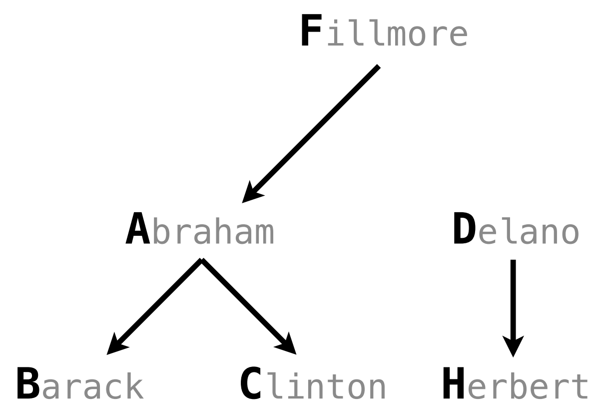
A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:

- A relation is a Scheme list.
- A fact expression is a Scheme list of relations.

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
```



Simple Facts

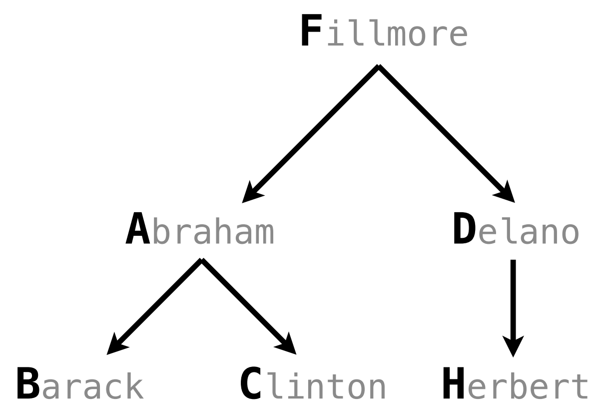
A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:

- A relation is a Scheme list.
- A fact expression is a Scheme list of relations.

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
```



Simple Facts

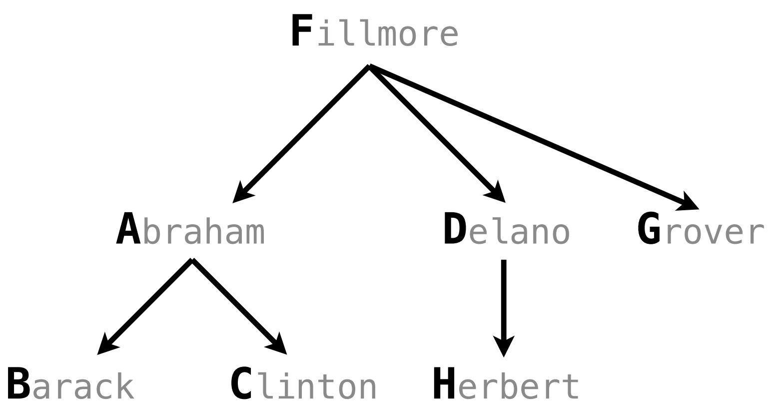
A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:

- A relation is a Scheme list.
- A fact expression is a Scheme list of relations.

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
```



Simple Facts

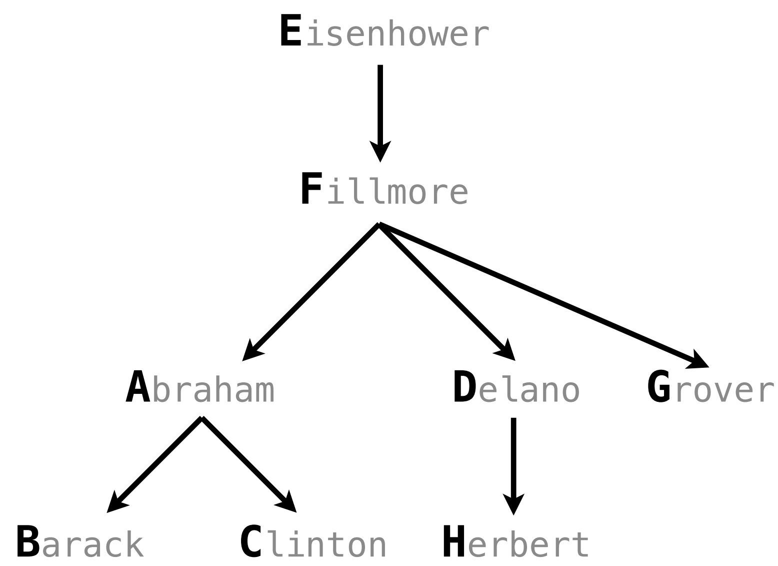
A simple fact expression in the Logic language declares a relation to be true.

Let's say I want to track the heredity of a pack of dogs.

Language Syntax:

- A relation is a Scheme list.
- A fact expression is a Scheme list of relations.

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
```



Relations are Not Procedure Calls

Relations are Not Procedure Calls

`In` *`Logic`*`, a relation is` **`not`** `a call expression.`

Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.

- *Scheme*: the expression **(abs -3)** calls *abs* on -3. It returns 3.

Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.

- *Scheme*: the expression **(abs -3)** calls *abs* on -3. It returns 3.
- *Logic*: **(abs -3 3)** asserts that *abs* of -3 is 3.

Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.

- *Scheme*: the expression **(abs -3)** calls *abs* on -3. It returns 3.
- *Logic*: **(abs -3 3)** asserts that *abs* of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: **(add 1 2 3)**

Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.

- *Scheme*: the expression **(abs -3)** calls *abs* on -3. It returns 3.
- *Logic*: **(abs -3 3)** asserts that *abs* of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: **(add 1 2 3)**

We can ask the Logic interpreter to complete relations based on known facts.

Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.

- *Scheme*: the expression **(abs -3)** calls *abs* on -3. It returns 3.
- *Logic*: **(abs -3 3)** asserts that *abs* of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: **(add 1 2 3)**

We can ask the Logic interpreter to complete relations based on known facts.

(add ? 2 3)

Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.

- *Scheme*: the expression **(abs -3)** calls *abs* on -3. It returns 3.
- *Logic*: **(abs -3 3)** asserts that *abs* of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: **(add 1 2 3)**

We can ask the Logic interpreter to complete relations based on known facts.

(add ? 2 3) **1**

Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.

- *Scheme*: the expression **(abs -3)** calls *abs* on -3. It returns 3.
- *Logic*: **(abs -3 3)** asserts that *abs* of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: **(add 1 2 3)**

We can ask the Logic interpreter to complete relations based on known facts.

(add ? 2 3) **1**

(add 1 ? 3)

Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.

- *Scheme*: the expression **(abs -3)** calls *abs* on -3. It returns 3.
- *Logic*: **(abs -3 3)** asserts that *abs* of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: **(add 1 2 3)**

We can ask the Logic interpreter to complete relations based on known facts.

(add ? 2 3) **1**

(add 1 ? 3) **2**

Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.

- *Scheme*: the expression **(abs -3)** calls *abs* on -3. It returns 3.
- *Logic*: **(abs -3 3)** asserts that *abs* of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: **(add 1 2 3)**

We can ask the Logic interpreter to complete relations based on known facts.

(add <u>?</u> 2 3) **1**

(add 1 <u>?</u> 3) **2**

(add 1 2 <u>?</u>)

Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.

- *Scheme*: the expression **(abs -3)** calls *abs* on -3. It returns 3.
- *Logic*: **(abs -3 3)** asserts that *abs* of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: **(add 1 2 3)**

We can ask the Logic interpreter to complete relations based on known facts.

(add ? 2 3)	**1**
(add 1 ? 3)	**2**
(add 1 2 ?)	**3**

Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.

- *Scheme*: the expression **(abs -3)** calls *abs* on -3. It returns 3.
- *Logic*: **(abs -3 3)** asserts that *abs* of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: **(add 1 2 3)**

We can ask the Logic interpreter to complete relations based on known facts.

(add ? 2 3) **1**

(add 1 ? 3) **2**

(add 1 2 ?) **3**

(? 1 2 3)

Relations are Not Procedure Calls

In *Logic*, a relation is **not** a call expression.

- *Scheme*: the expression **(abs -3)** calls *abs* on -3. It returns 3.
- *Logic*: **(abs -3 3)** asserts that *abs* of -3 is 3.

To assert that 1 + 2 = 3, we use a relation: **(add 1 2 3)**

We can ask the Logic interpreter to complete relations based on known facts.

(add _?_ 2 3)	**1**
(add 1 _?_ 3)	**2**
(add 1 2 _?_)	**3**
(_?_ 1 2 3)	**add**

Queries

Queries

Queries

A *query* contains one or more relations that may contain variables.

Queries

A *query* contains one or more relations that may contain variables.

Variables are symbols starting with **?**

Queries

A *query* contains one or more relations that may contain variables.

Variables are symbols starting with **?**

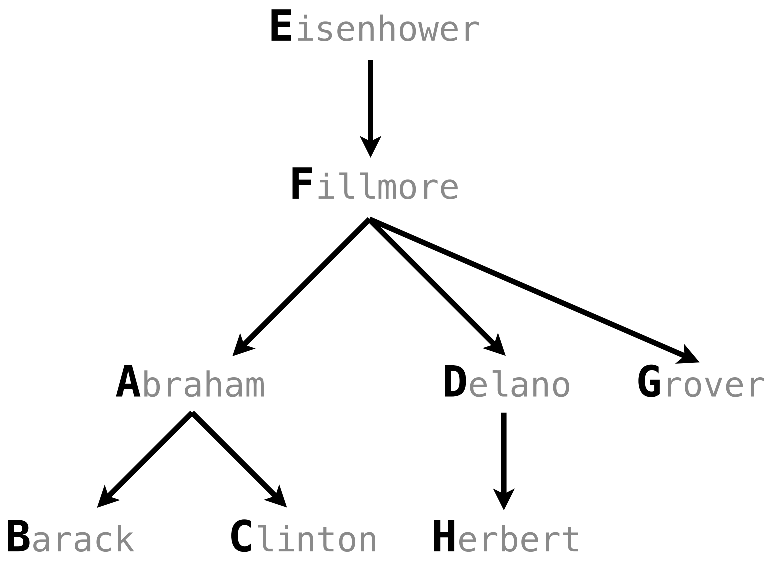
```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
```

Queries

A *query* contains one or more relations that may contain variables.

Variables are symbols starting with **?**

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
```

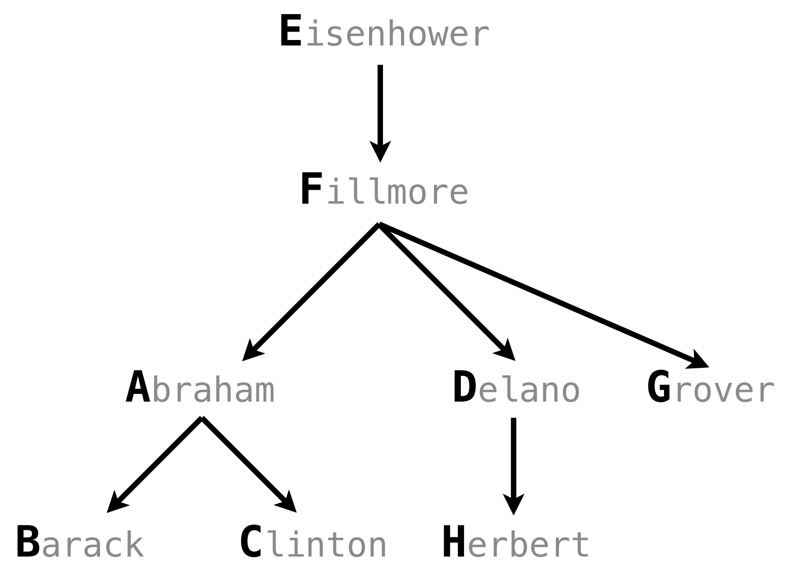


Queries

A *query* contains one or more relations that may contain variables.

Variables are symbols starting with **?**

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))

logic> (query (parent abraham ?puppy))
```



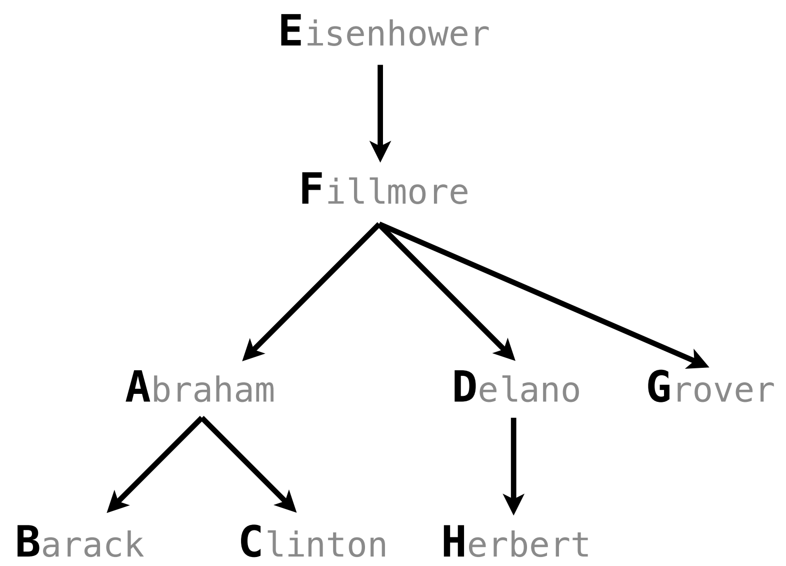
Queries

A *query* contains one or more relations that may contain variables.

Variables are symbols starting with **?**

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
logic> (query (parent abraham ?puppy))
```

A variable can have any name



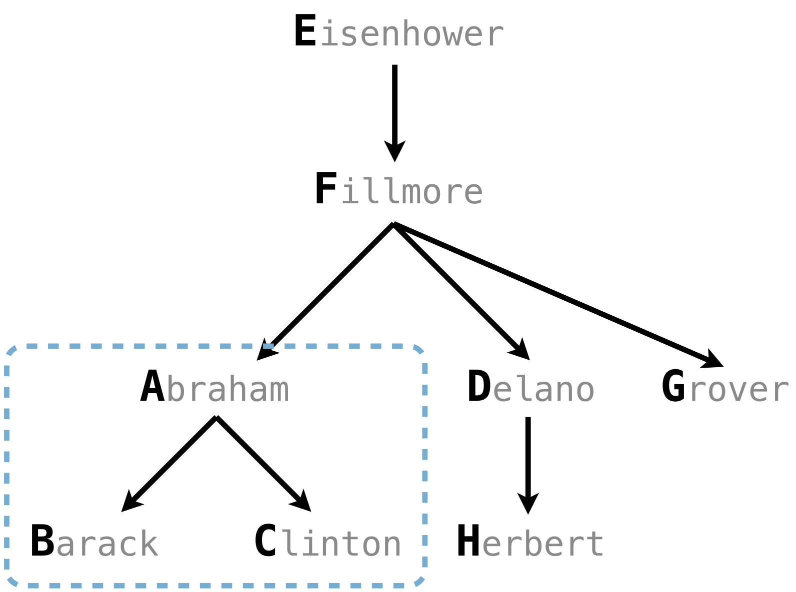
Queries

A *query* contains one or more relations that may contain variables.

Variables are symbols starting with **?**

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))
logic> (query (parent abraham ?puppy))
```

A variable can have any name



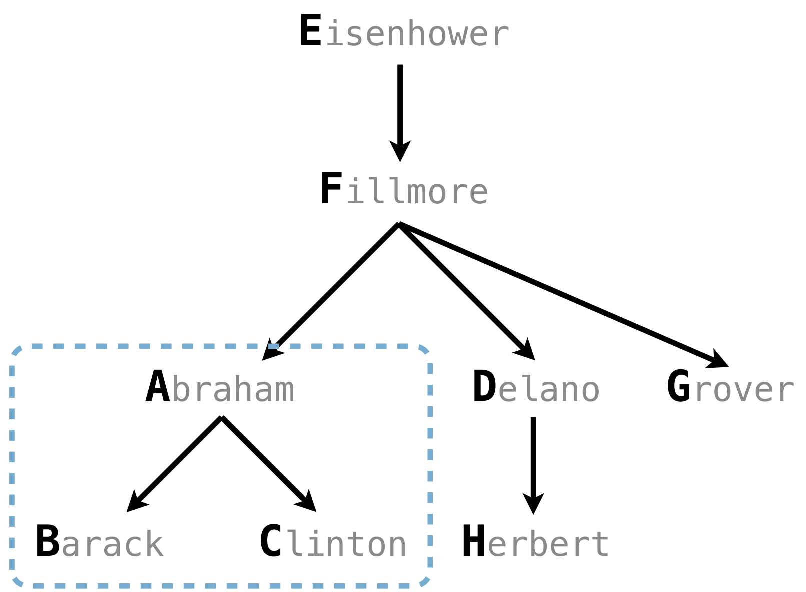
Queries

A *query* contains one or more relations that may contain variables.

Variables are symbols starting with **?**

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))

logic> (query (parent abraham ?puppy))
Success!
```

A variable can have any name



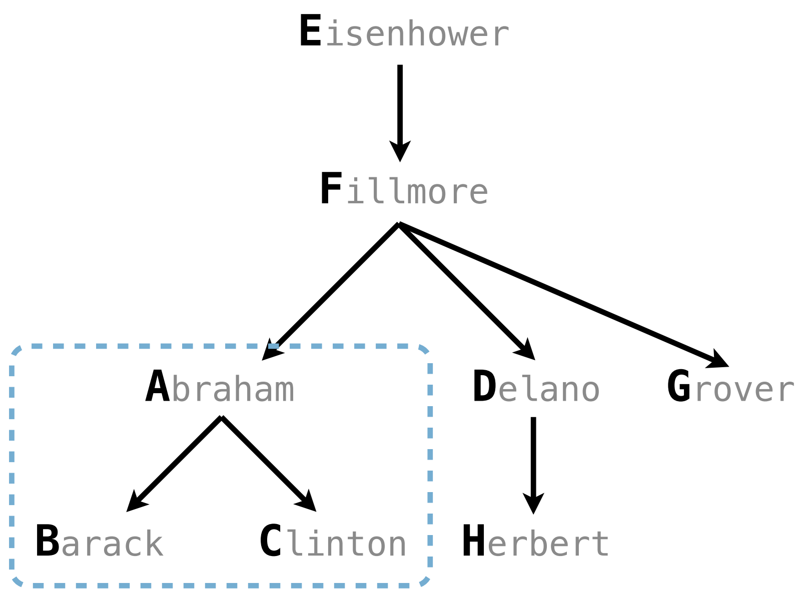
Queries

A *query* contains one or more relations that may contain variables.

Variables are symbols starting with **?**

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))

logic> (query (parent abraham ?puppy))
Success!
puppy: barack
```

A variable can have any name



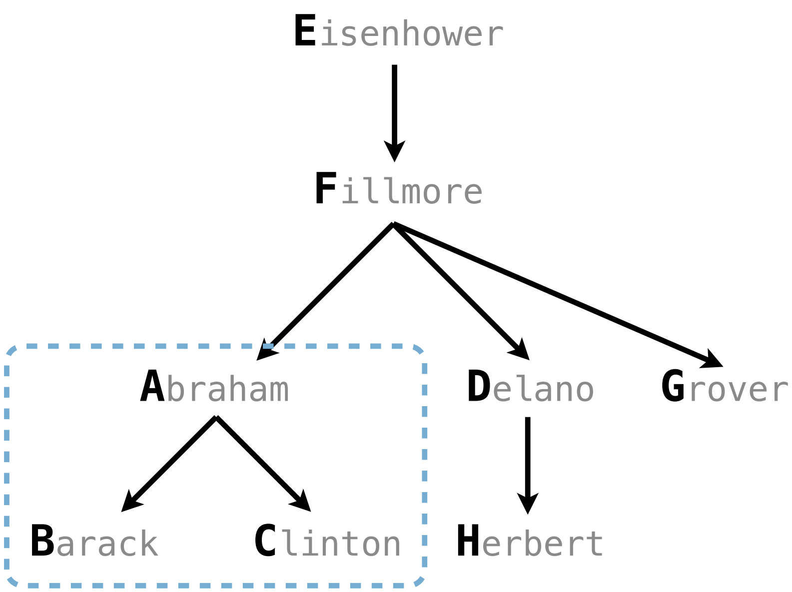
Queries

A *query* contains one or more relations that may contain variables.

Variables are symbols starting with **?**

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))

logic> (query (parent abraham ?puppy))
Success!
puppy: barack
puppy: clinton
```

A variable can have any name



Queries

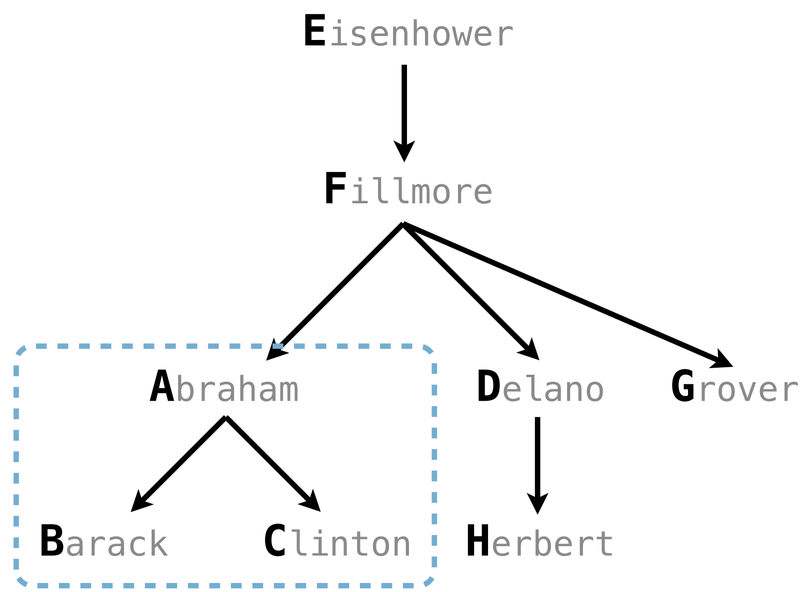
A *query* contains one or more relations that may contain variables.

Variables are symbols starting with **?**

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))

logic> (query (parent abraham ?puppy))
Success!
puppy: barack
puppy: clinton
```

A variable can have any name

Each line is an assignment of variables to values



Queries

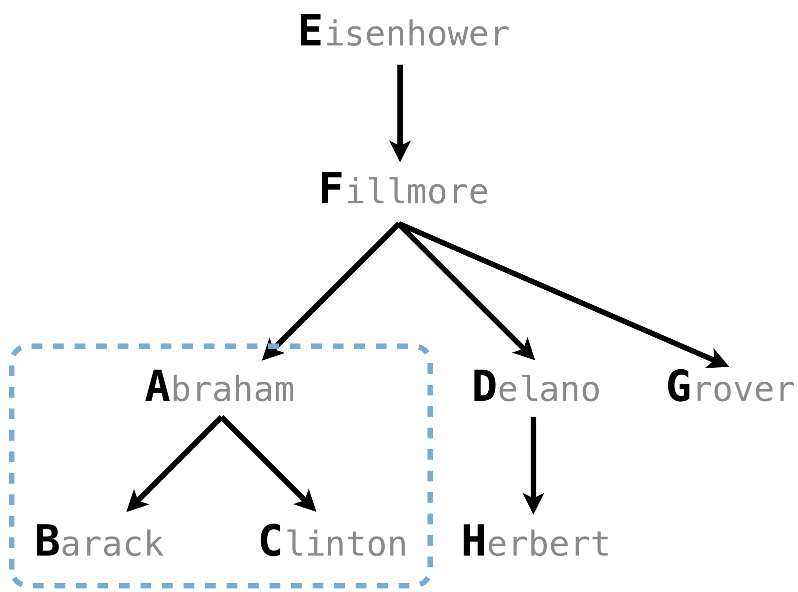
A *query* contains one or more relations that may contain variables.

Variables are symbols starting with **?**

```
logic> (fact (parent delano herbert))
logic> (fact (parent abraham barack))
logic> (fact (parent abraham clinton))
logic> (fact (parent fillmore abraham))
logic> (fact (parent fillmore delano))
logic> (fact (parent fillmore grover))
logic> (fact (parent eisenhower fillmore))

logic> (query (parent abraham ?puppy))
Success!
puppy: barack
puppy: clinton
```

A variable can have any name

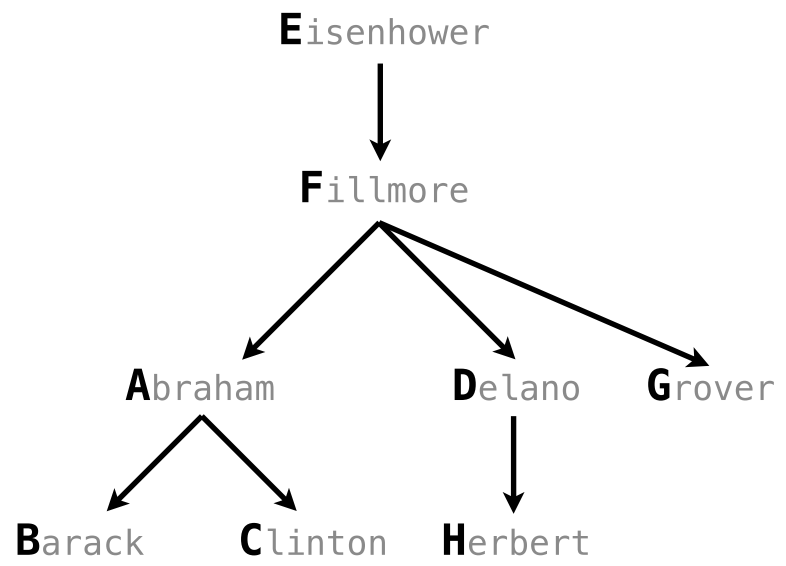
Each line is an assignment of variables to values



(Demo)

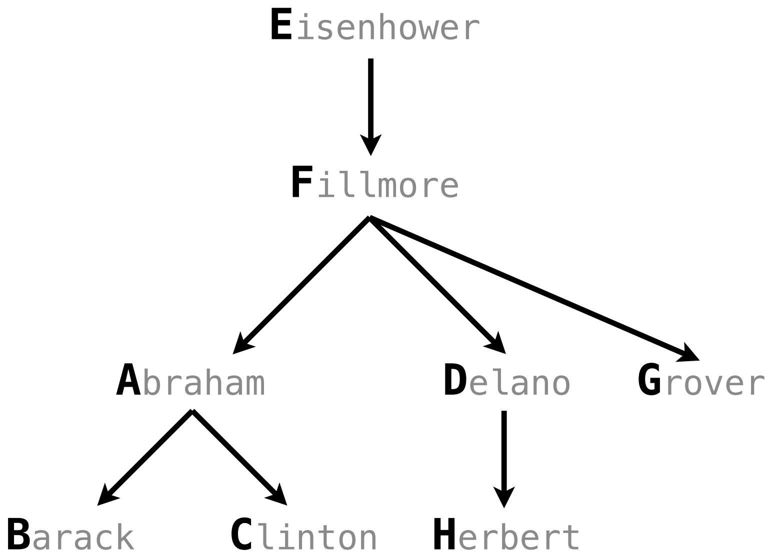
Compound Facts and Queries

Compound Facts



Compound Facts

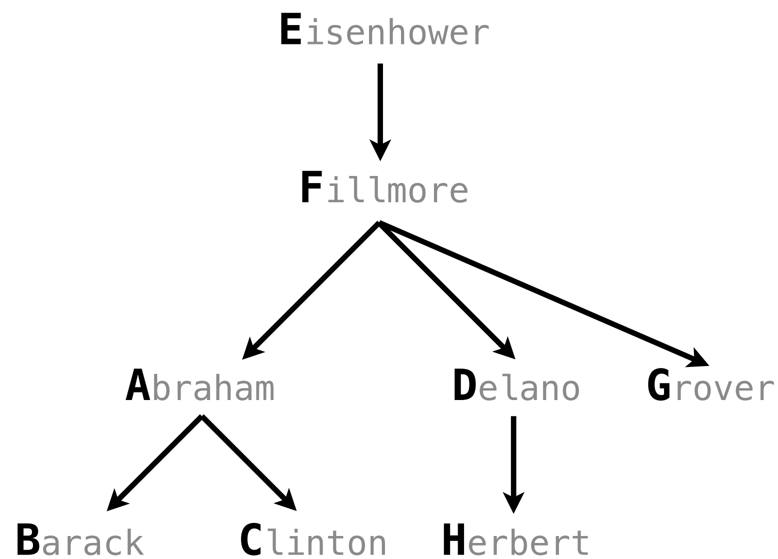
A fact can include multiple relations and variables as well.



Compound Facts

A fact can include multiple relations and variables as well.

```
(fact <conclusion> <hypothesis0> <hypothesis1> ... <hypothesisN>)
```

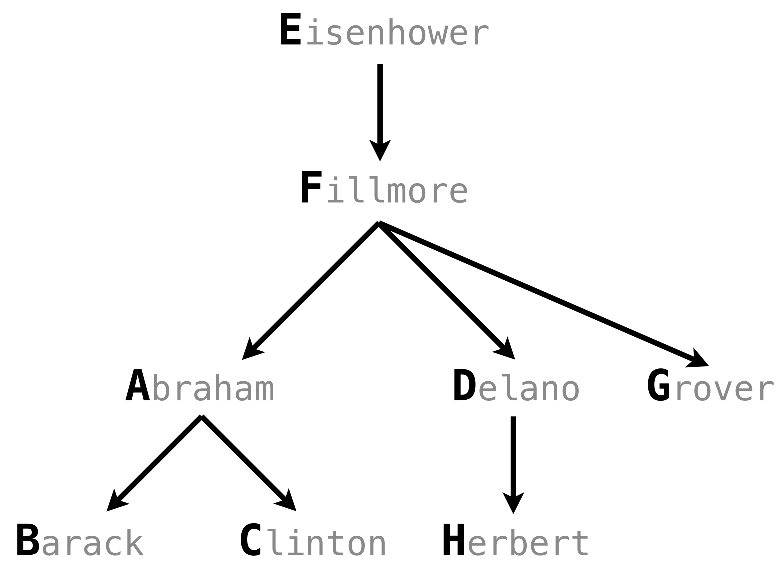


Compound Facts

A fact can include multiple relations and variables as well.

(fact <conclusion> <hypothesis$_0$> <hypothesis$_1$> ... <hypothesis$_N$>)

Means <conclusion> is true if all the <hypothesis$_K$> are true.



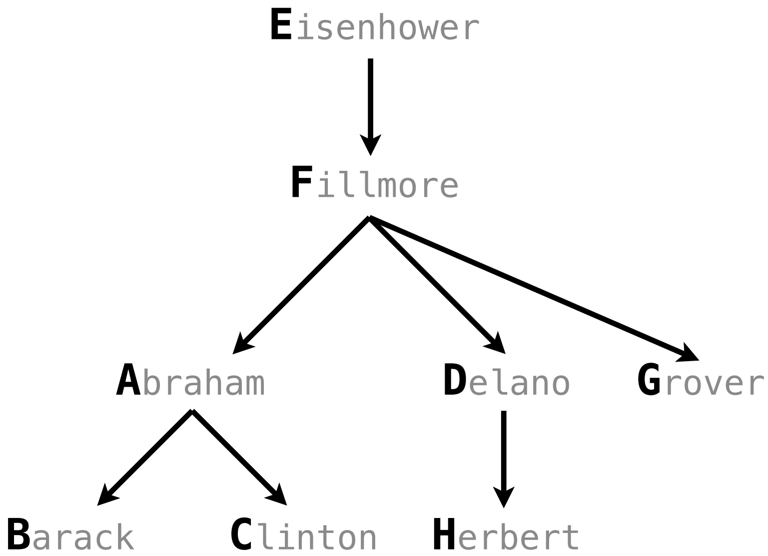
Compound Facts

A fact can include multiple relations and variables as well.

(fact <conclusion> <hypothesis$_0$> <hypothesis$_1$> ... <hypothesis$_N$>)

Means <conclusion> is true if all the <hypothesis$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))
```



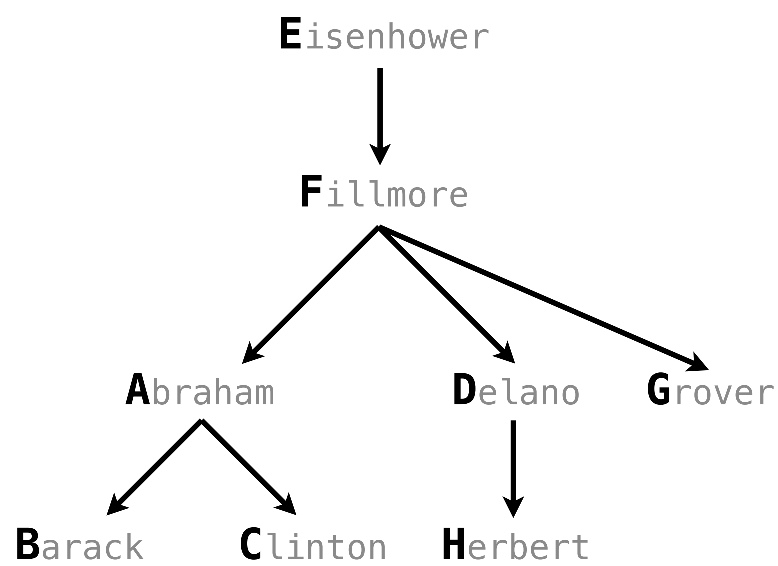
Compound Facts

A fact can include multiple relations and variables as well.

(fact <conclusion> <hypothesis$_0$> <hypothesis$_1$> ... <hypothesis$_N$>)

Means <conclusion> is true if all the <hypothesis$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (child herbert delano))
```



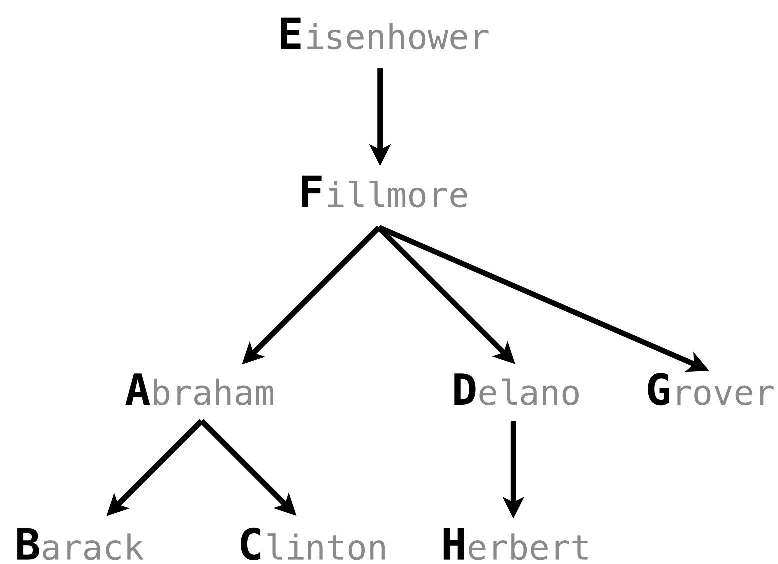
Compound Facts

A fact can include multiple relations and variables as well.

(fact <conclusion> <hypothesis$_0$> <hypothesis$_1$> ... <hypothesis$_N$>)

Means <conclusion> is true if all the <hypothesis$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (child herbert delano))
Success!
```



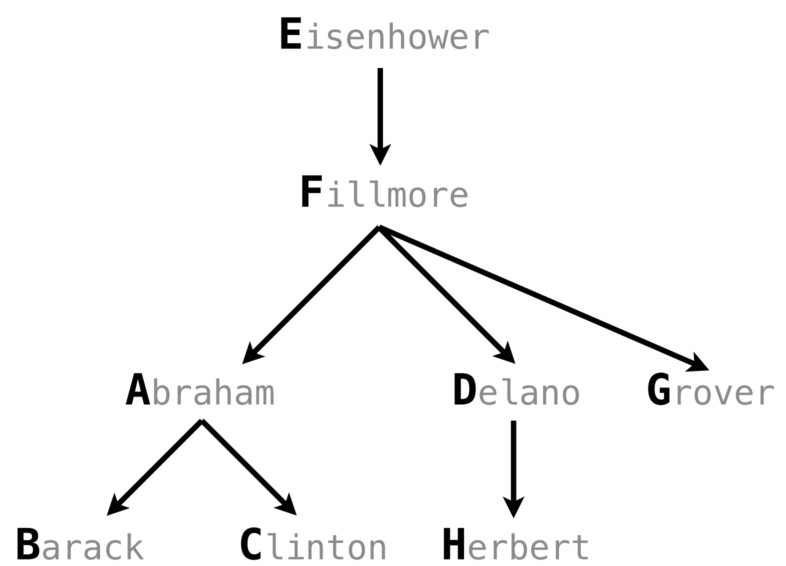
Compound Facts

A fact can include multiple relations and variables as well.

```
(fact <conclusion> <hypothesis0> <hypothesis1> ... <hypothesisN>)
```

Means <conclusion> is true if all the $\langle hypothesis_K \rangle$ are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (child herbert delano))
Success!

logic> (query (child eisenhower clinton))
```



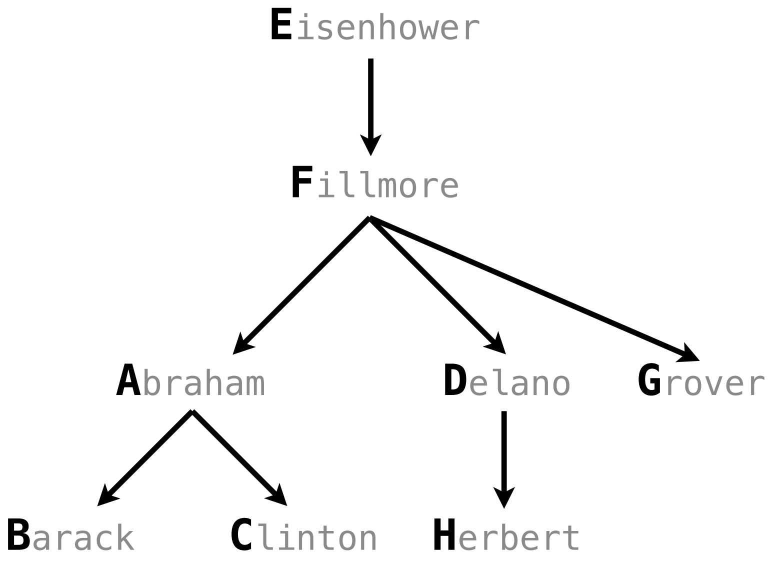
Compound Facts

A fact can include multiple relations and variables as well.

(fact <conclusion> <hypothesis$_0$> <hypothesis$_1$> ... <hypothesis$_N$>)

Means <conclusion> is true if all the <hypothesis$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (child herbert delano))
Success!

logic> (query (child eisenhower clinton))
Failure.
```



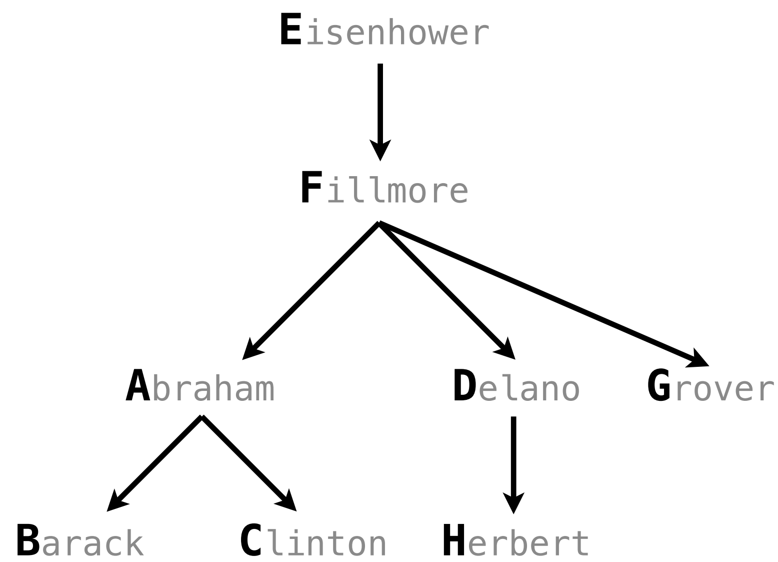
Compound Facts

A fact can include multiple relations and variables as well.

(fact <conclusion> <hypothesis$_0$> <hypothesis$_1$> ... <hypothesis$_N$>)

Means <conclusion> is true if all the <hypothesis$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (child herbert delano))
Success!

logic> (query (child eisenhower clinton))
Failure.

logic> (query (child ?kid fillmore))
```



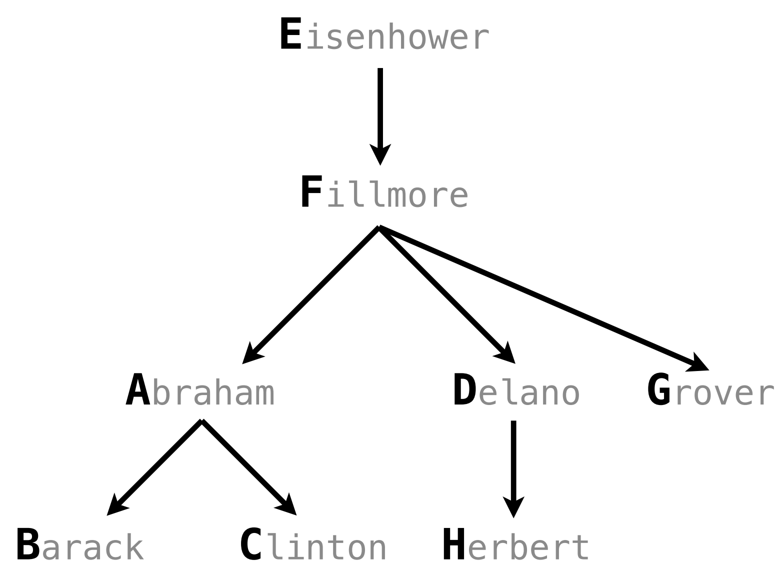
Compound Facts

A fact can include multiple relations and variables as well.

(fact <conclusion> <hypothesis$_0$> <hypothesis$_1$> ... <hypothesis$_N$>)

Means <conclusion> is true if all the <hypothesis$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (child herbert delano))
Success!

logic> (query (child eisenhower clinton))
Failure.

logic> (query (child ?kid fillmore))
Success!
```



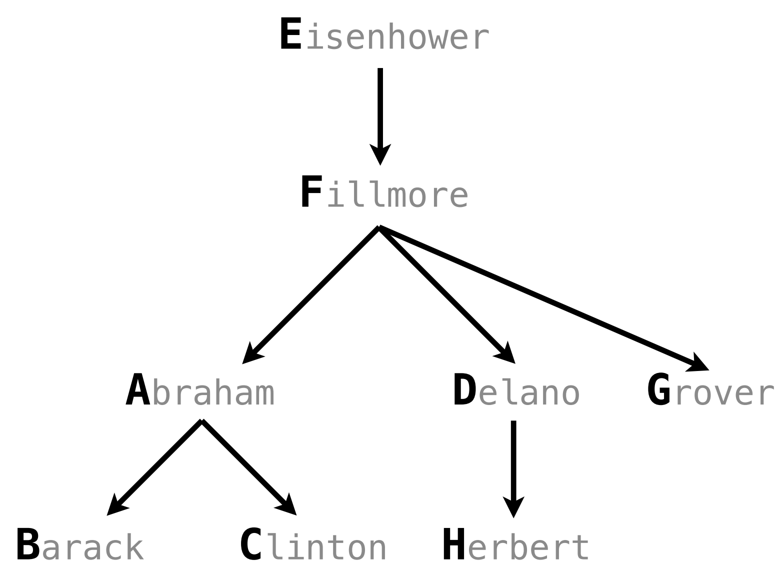
Compound Facts

A fact can include multiple relations and variables as well.

(fact <conclusion> <hypothesis$_0$> <hypothesis$_1$> ... <hypothesis$_N$>)

Means <conclusion> is true if all the <hypothesis$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (child herbert delano))
Success!

logic> (query (child eisenhower clinton))
Failure.

logic> (query (child ?kid fillmore))
Success!
kid: abraham
```



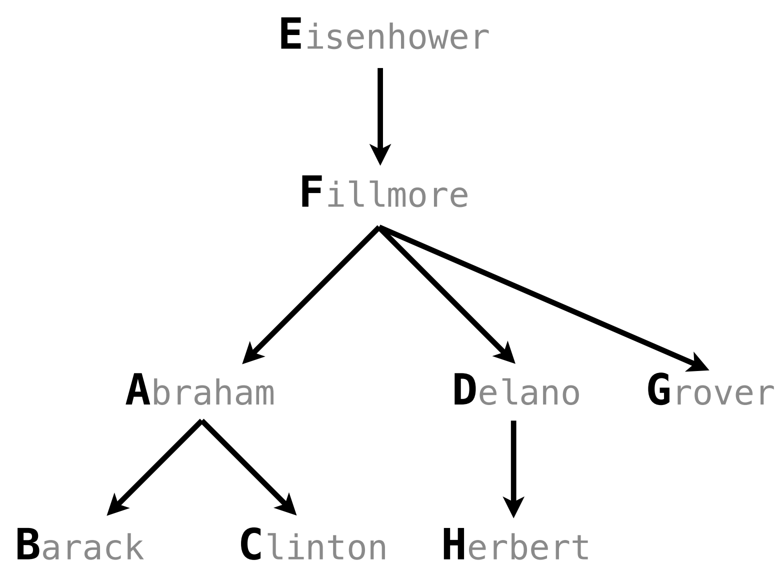
Compound Facts

A fact can include multiple relations and variables as well.

(fact <conclusion> <hypothesis$_0$> <hypothesis$_1$> ... <hypothesis$_N$>)

Means <conclusion> is true if all the <hypothesis$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (child herbert delano))
Success!

logic> (query (child eisenhower clinton))
Failure.

logic> (query (child ?kid fillmore))
Success!
kid: abraham
kid: delano
```



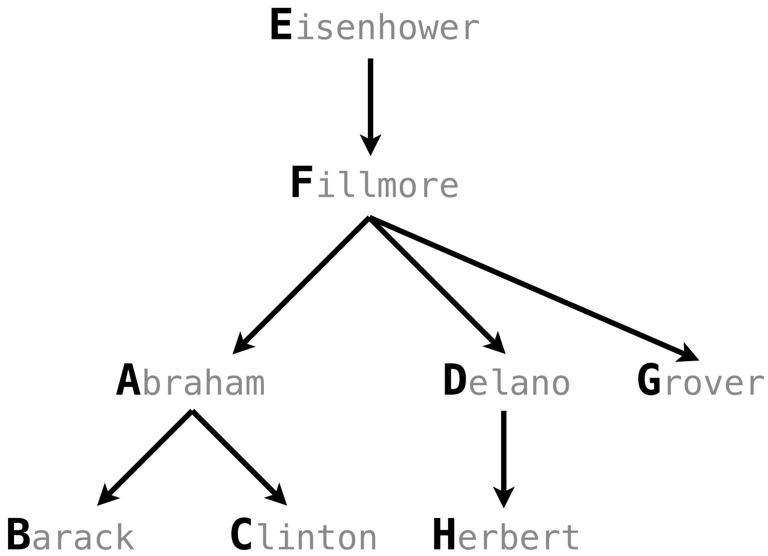
Compound Facts

A fact can include multiple relations and variables as well.

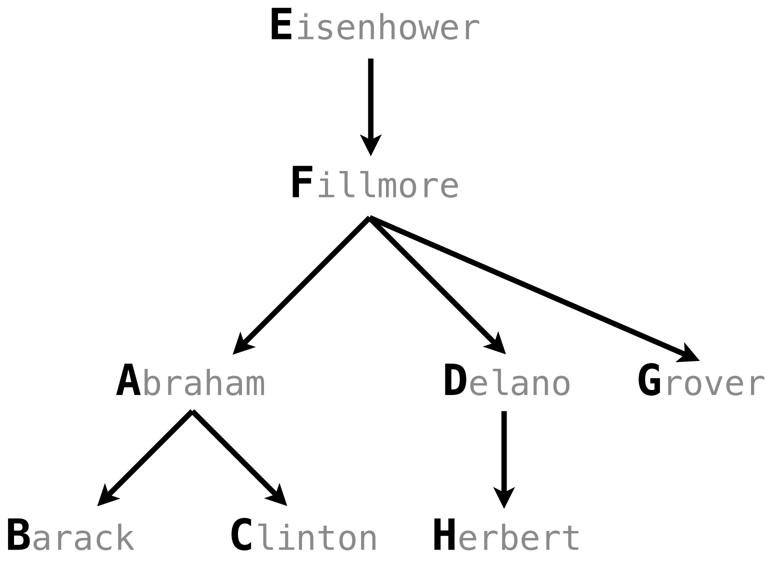
(fact <conclusion> <hypothesis$_0$> <hypothesis$_1$> ... <hypothesis$_N$>)

Means <conclusion> is true if all the <hypothesis$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (child herbert delano))
Success!

logic> (query (child eisenhower clinton))
Failure.

logic> (query (child ?kid fillmore))
Success!
kid: abraham
kid: delano
kid: grover
```

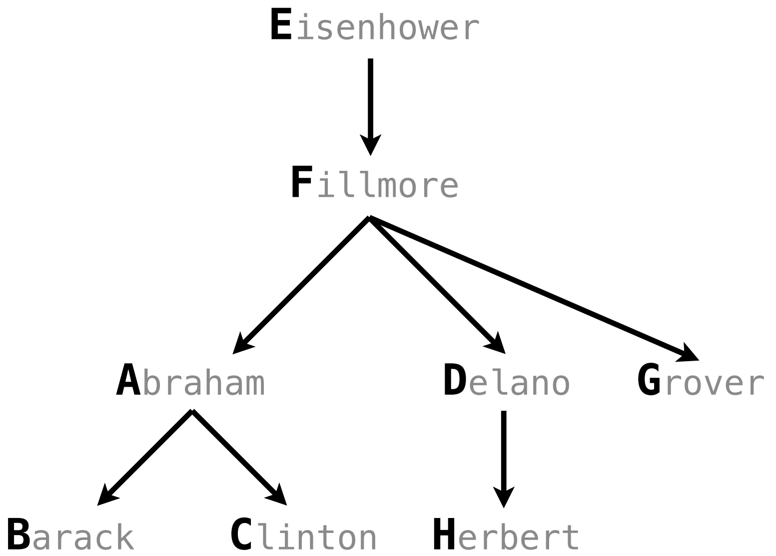


Compound Queries



Compound Queries

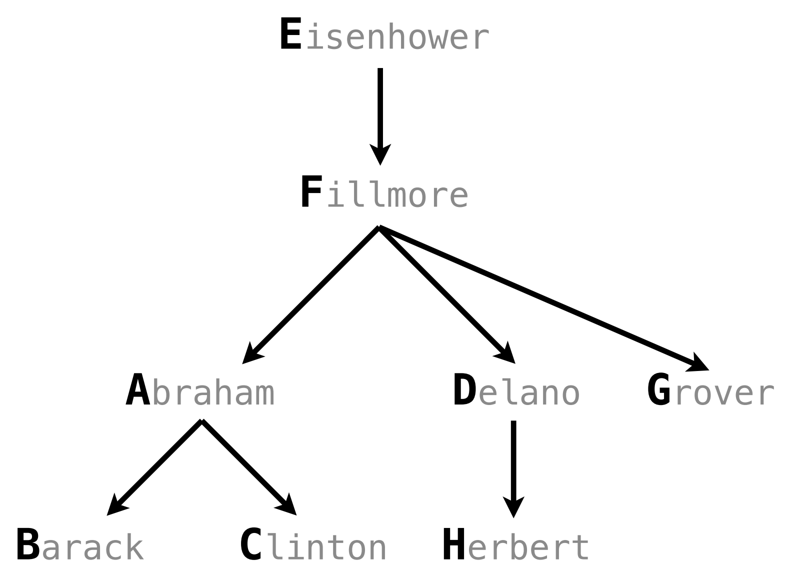
An assignment must satisfy all relations in a query.



Compound Queries

An assignment must satisfy all relations in a query.

```
(query <relation₀> <relation₁> ... <relationN>)
```

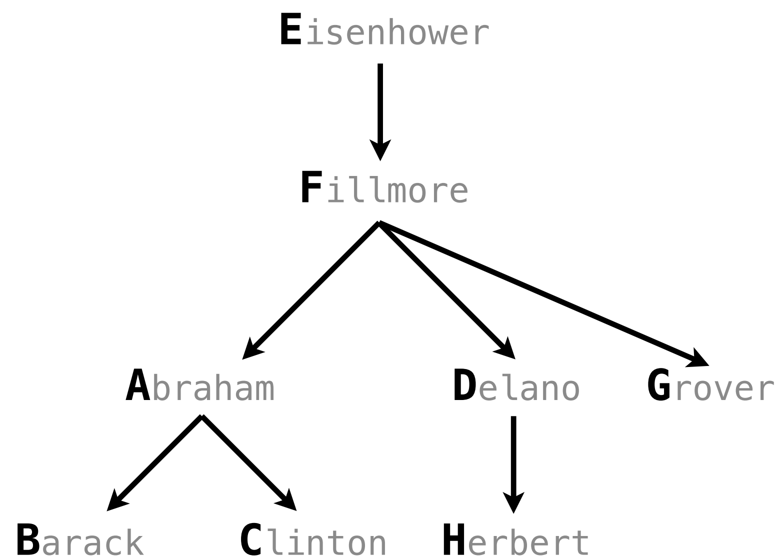


Compound Queries

An assignment must satisfy all relations in a query.

```
(query <relation₀> <relation₁> ... <relationN>)
```

is satisfied if all the <relation$_K$> are true.



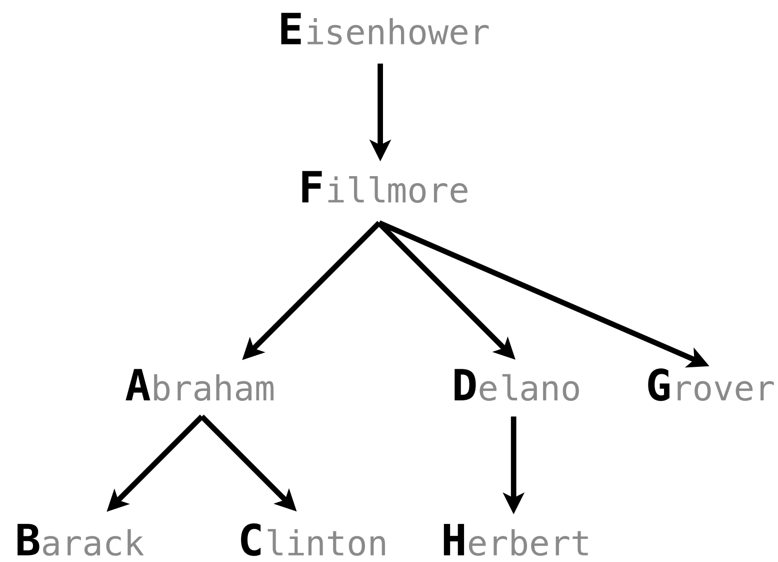
Compound Queries

An assignment must satisfy all relations in a query.

(query <relation$_0$> <relation$_1$> ... <relation$_N$>)

is satisfied if all the <relation$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))
```



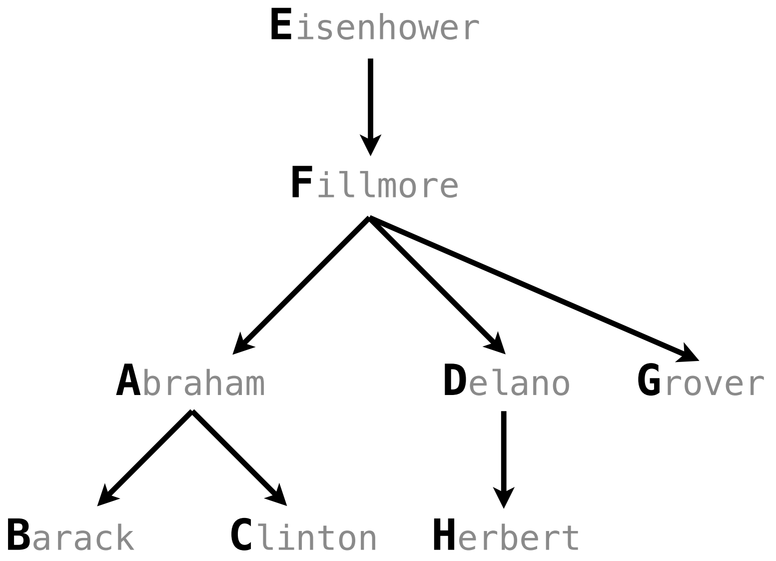
Compound Queries

An assignment must satisfy all relations in a query.

```
(query <relation_0> <relation_1> ... <relation_N>)
```

is satisfied if all the `<relation_K>` are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (parent ?grampa ?kid)
```



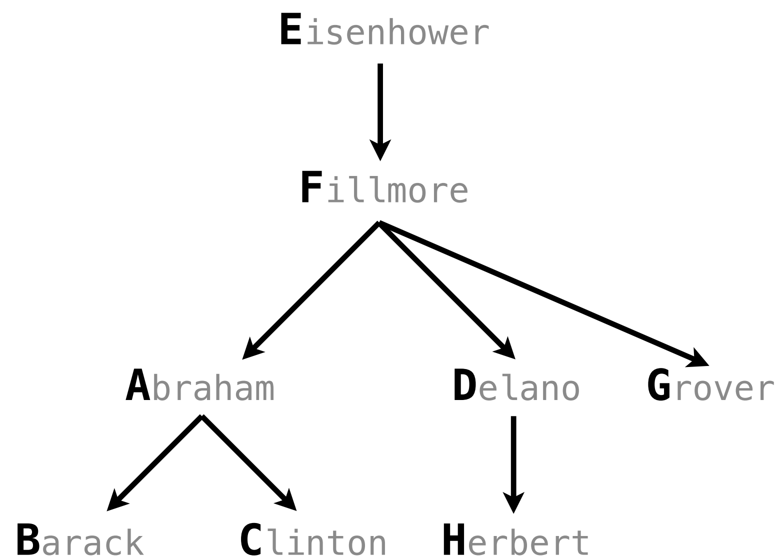
Compound Queries

An assignment must satisfy all relations in a query.

(query <relation$_0$> <relation$_1$> ... <relation$_N$>)

is satisfied if all the <relation$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (parent ?grampa ?kid)
              (child clinton ?kid))
```



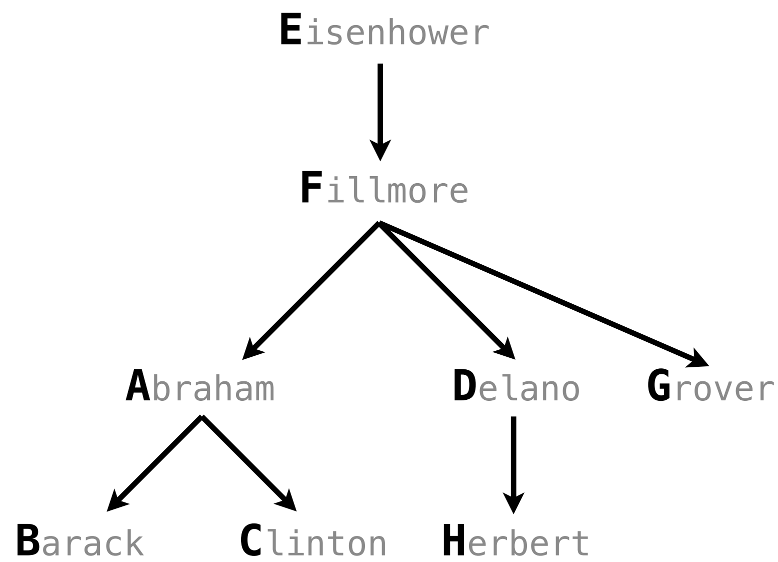
Compound Queries

An assignment must satisfy all relations in a query.

(query <relation$_0$> <relation$_1$> ... <relation$_N$>)

is satisfied if all the <relation$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (parent ?grampa ?kid)
              (child clinton ?kid))
Success!
```



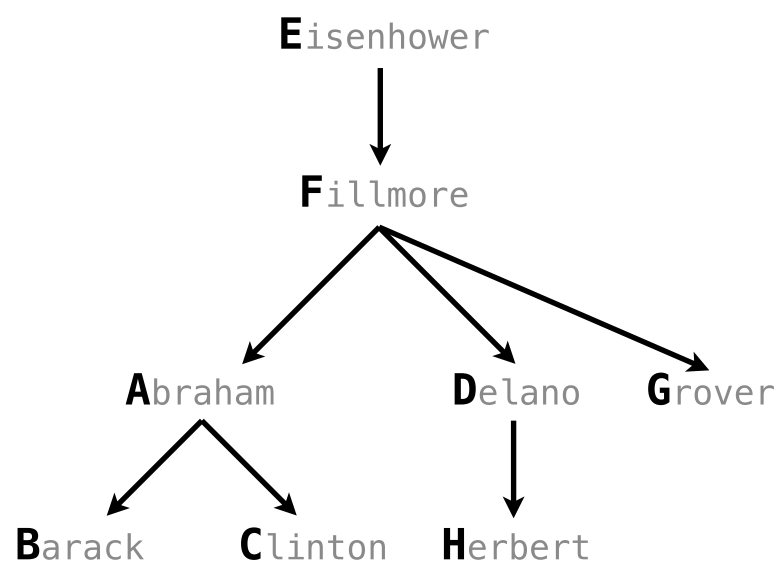
Compound Queries

An assignment must satisfy all relations in a query.

(query <relation$_0$> <relation$_1$> ... <relation$_N$>)

is satisfied if all the <relation$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (parent ?grampa ?kid)
              (child clinton ?kid))
Success!
grampa: fillmore    kid: abraham
```



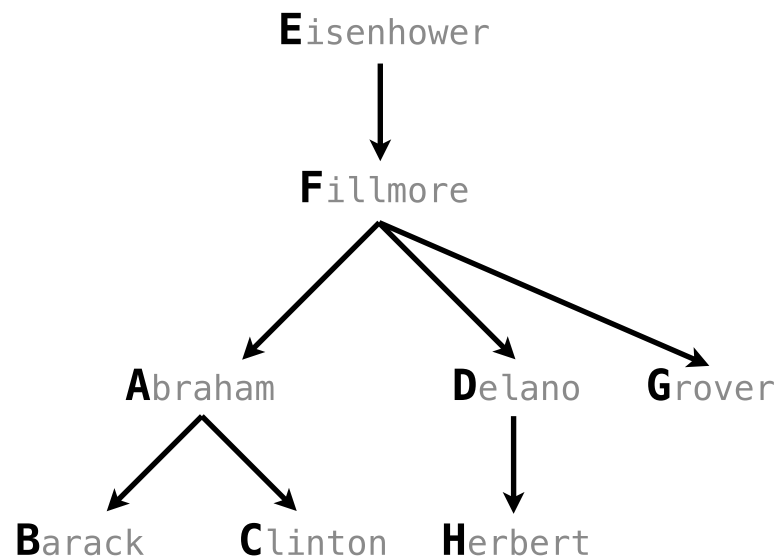
Compound Queries

An assignment must satisfy all relations in a query.

(query <relation$_0$> <relation$_1$> ... <relation$_N$>)

is satisfied if all the <relation$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (parent ?grampa ?kid)
              (child clinton ?kid))
Success!
grampa: fillmore    kid: abraham

logic> (query (child ?y ?x)
```



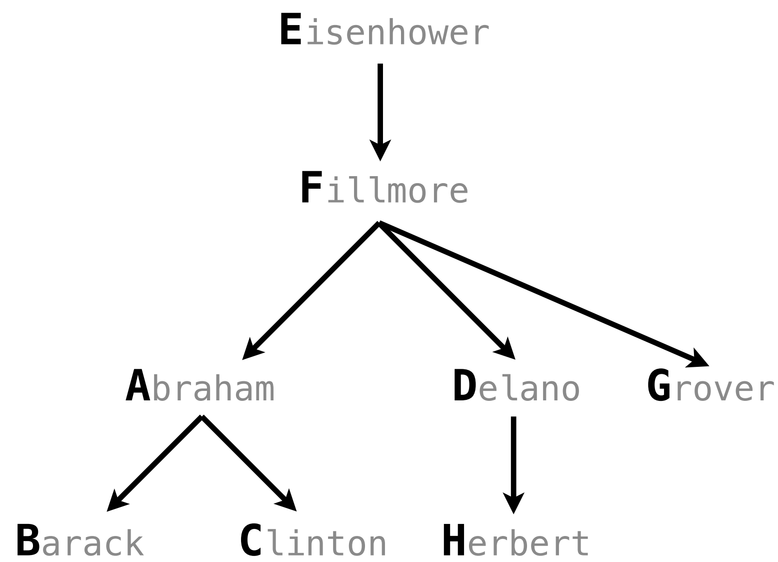
Compound Queries

An assignment must satisfy all relations in a query.

(query <relation$_0$> <relation$_1$> ... <relation$_N$>)

is satisfied if all the <relation$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (parent ?grampa ?kid)
              (child clinton ?kid))
Success!
grampa: fillmore    kid: abraham

logic> (query (child ?y ?x)
              (child ?x eisenhower))
```



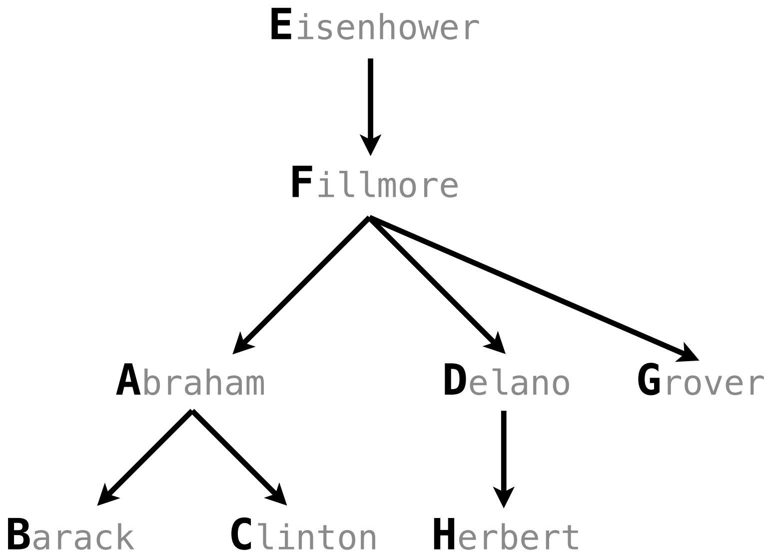
Compound Queries

An assignment must satisfy all relations in a query.

(query <relation$_0$> <relation$_1$> ... <relation$_N$>)

is satisfied if all the <relation$_K$> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (parent ?grampa ?kid)
              (child clinton ?kid))
Success!
grampa: fillmore    kid: abraham

logic> (query (child ?y ?x)
              (child ?x eisenhower))
Success!
```



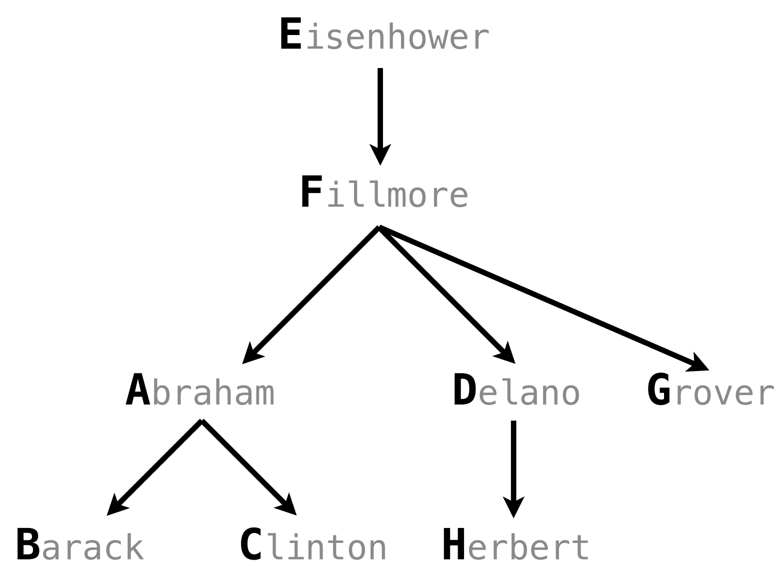
Compound Queries

An assignment must satisfy all relations in a query.

```
(query <relation_0> <relation_1> ... <relation_N>)
```

is satisfied if all the `<relation_K>` are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (parent ?grampa ?kid)
              (child clinton ?kid))
Success!
grampa: fillmore    kid: abraham

logic> (query (child ?y ?x)
              (child ?x eisenhower))
Success!
y: abraham    x: fillmore
```



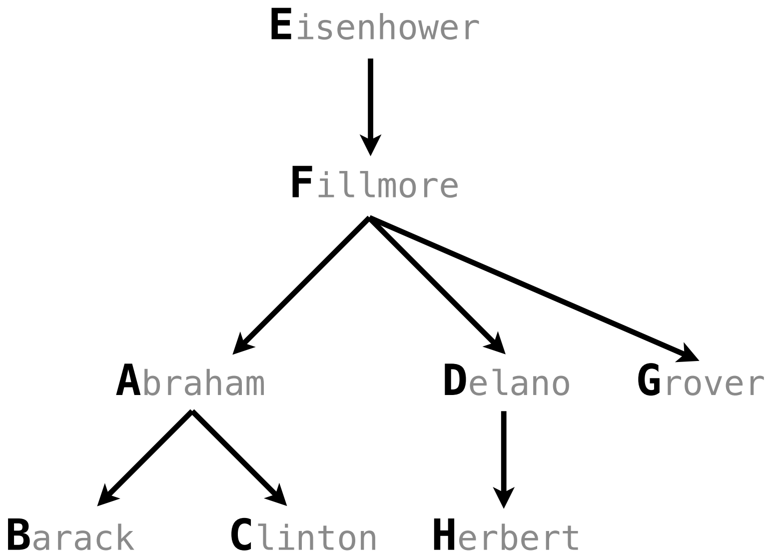
Compound Queries

An assignment must satisfy all relations in a query.

(query <relation₀> <relation₁> ... <relationN>)

is satisfied if all the <relationK> are true.

```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (parent ?grampa ?kid)
              (child clinton ?kid))
Success!
grampa: fillmore    kid: abraham

logic> (query (child ?y ?x)
              (child ?x eisenhower))
Success!
y: abraham    x: fillmore
y: delano     x: fillmore
```



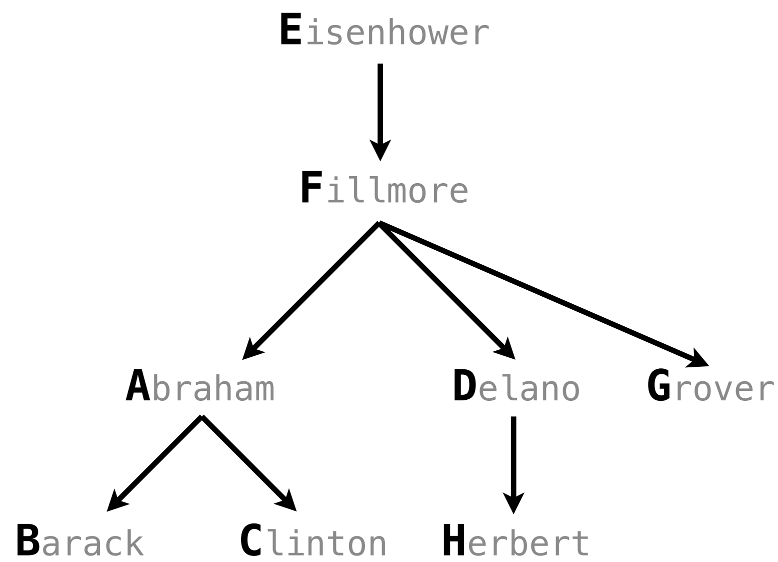
Compound Queries

An assignment must satisfy all relations in a query.

(query <relation$_0$> <relation$_1$> ... <relation$_N$>)

is satisfied if all the <relation$_K$> are true.

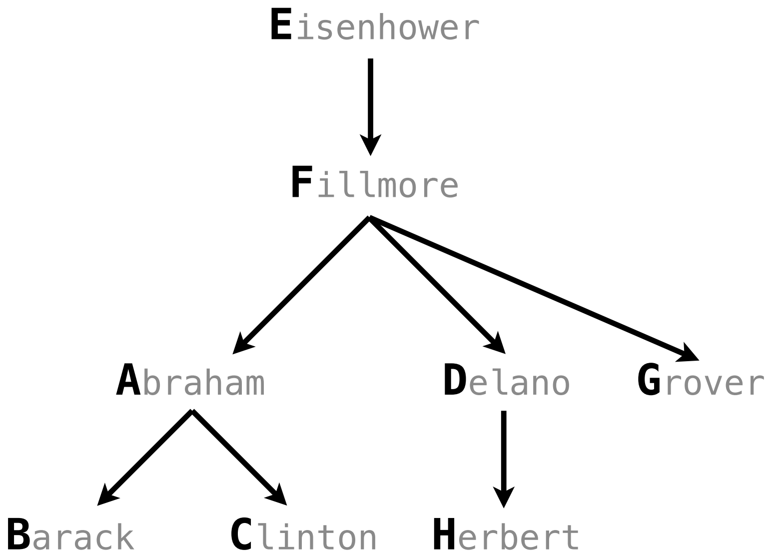
```
logic> (fact (child ?c ?p) (parent ?p ?c))

logic> (query (parent ?grampa ?kid)
              (child clinton ?kid))
Success!
grampa: fillmore    kid: abraham

logic> (query (child ?y ?x)
              (child ?x eisenhower))
Success!
y: abraham    x: fillmore
y: delano     x: fillmore
y: grover     x: fillmore
```



Recursive Facts

Recursive Facts

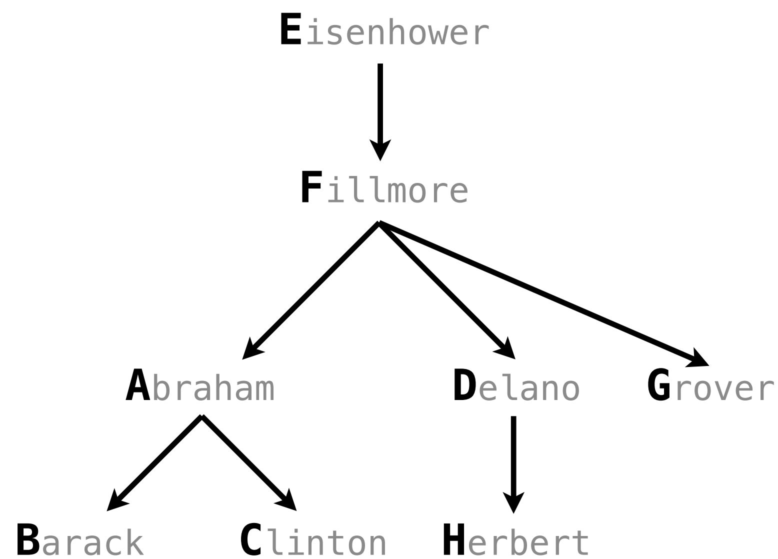
A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.



Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

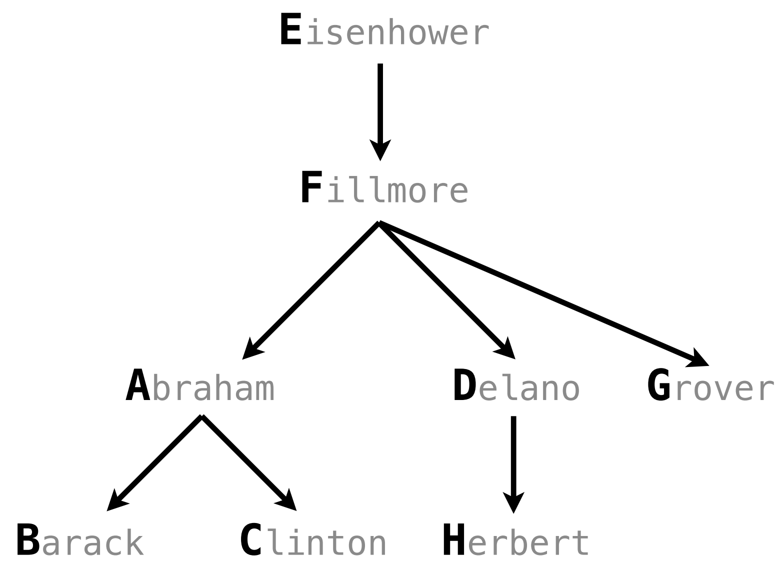
```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
```



Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

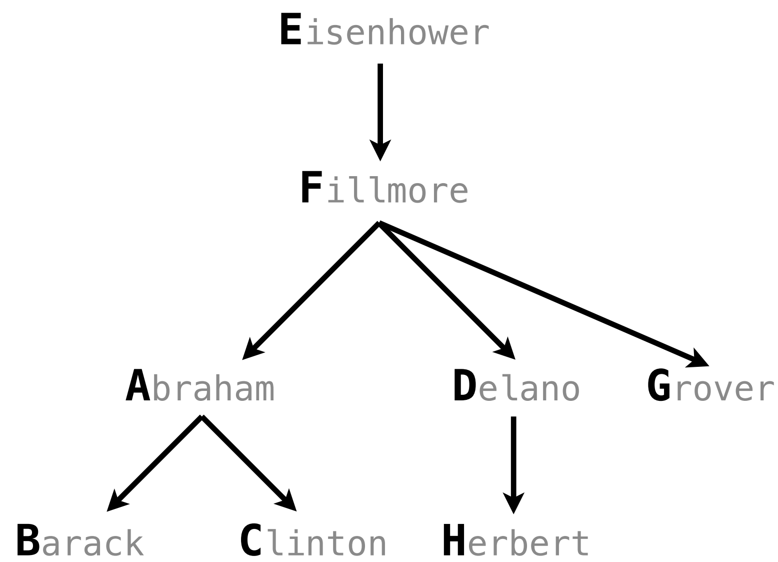
```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))
```



Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

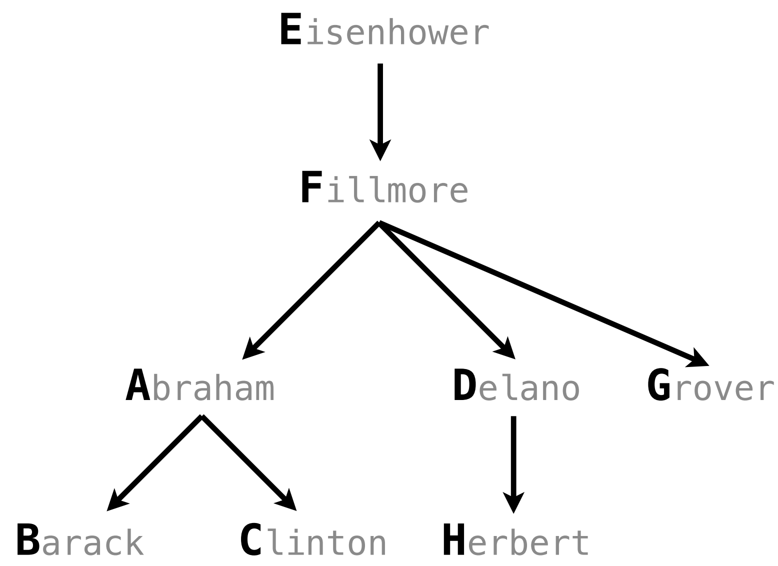
```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))

logic> (query (ancestor ?a herbert))
```



Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

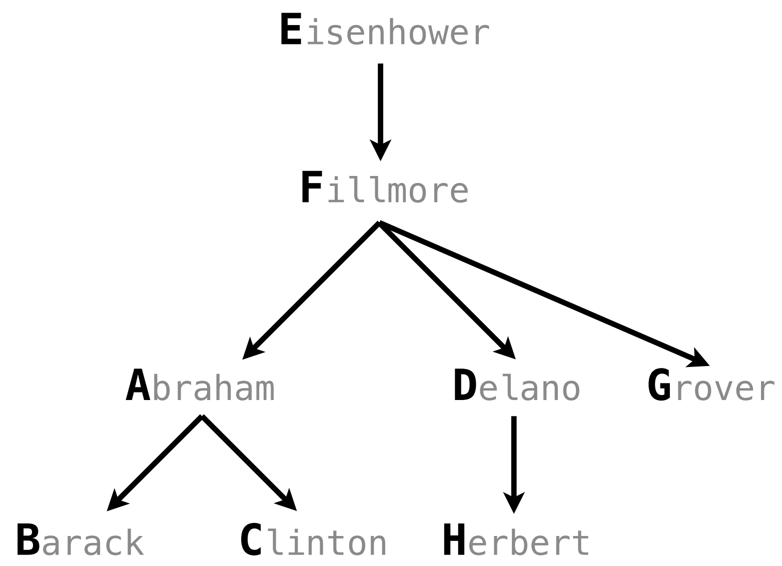
```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))

logic> (query (ancestor ?a herbert))
Success!
```



Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

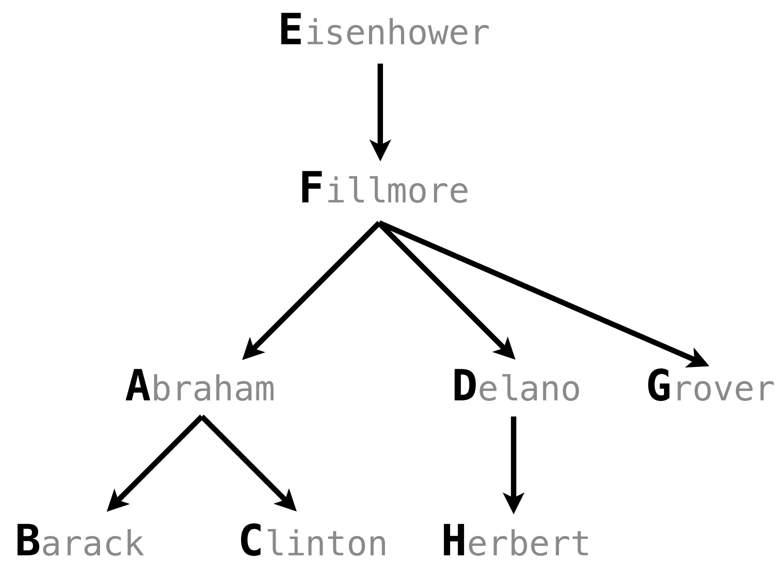
```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))

logic> (query (ancestor ?a herbert))
Success!
a: delano
```



Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

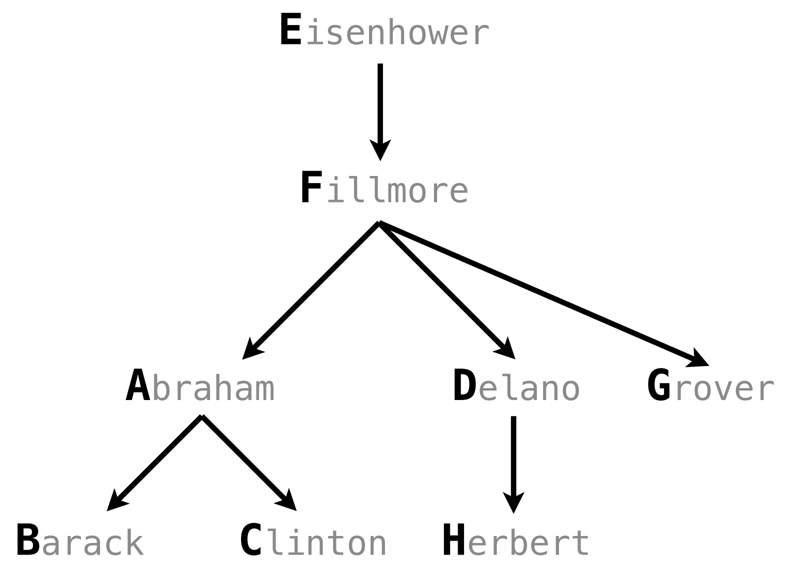
```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))

logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
```



Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

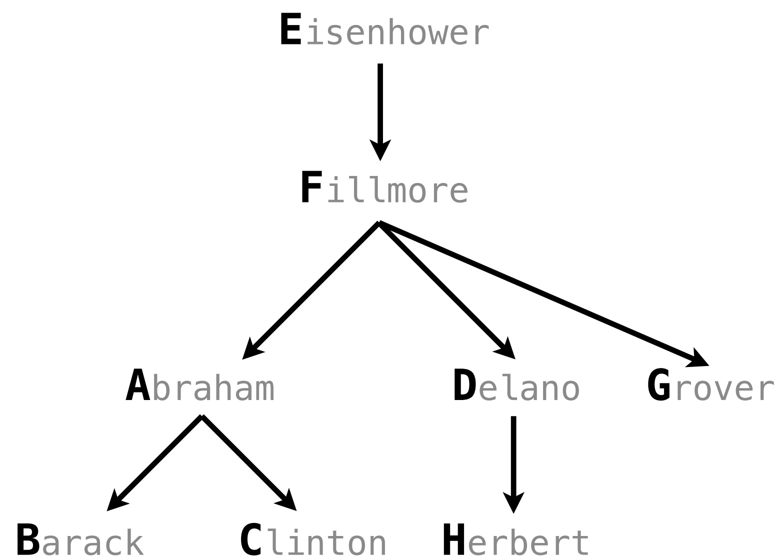
```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))

logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
```



Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

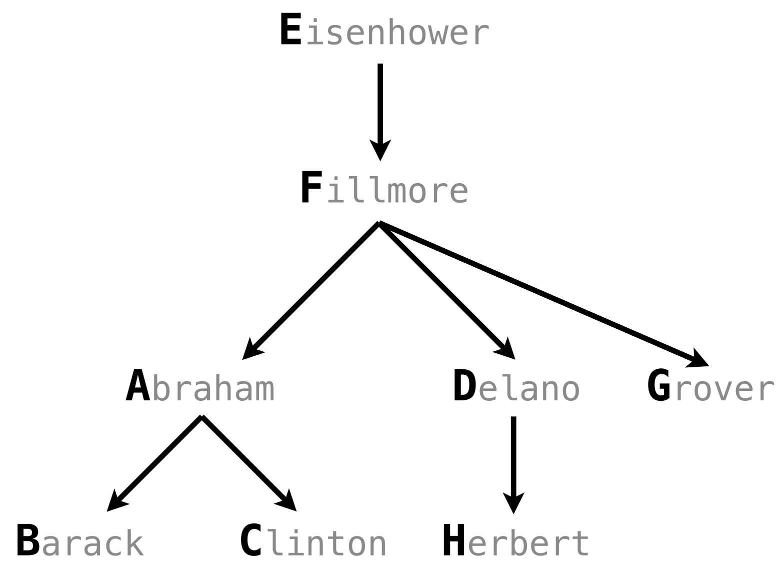
```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))

logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower

logic> (query (ancestor ?a barack)
```



Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

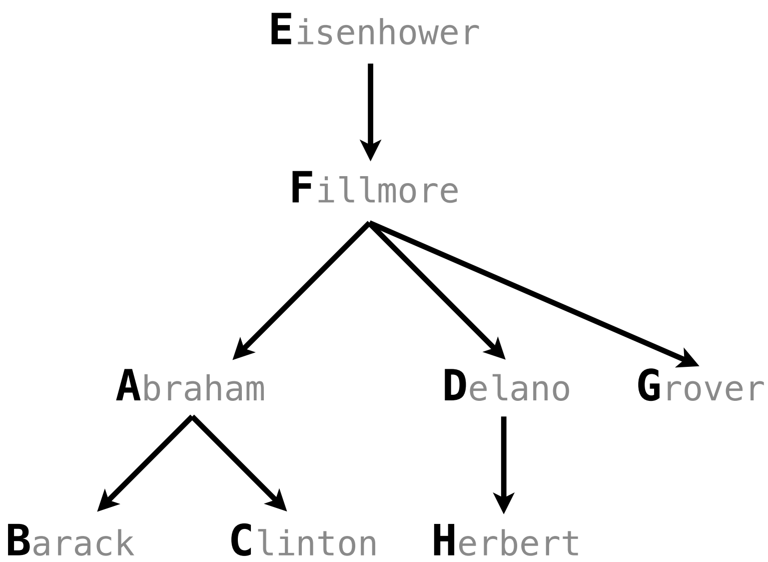
```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))

logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower

logic> (query (ancestor ?a barack)
              (ancestor ?a herbert))
```



Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

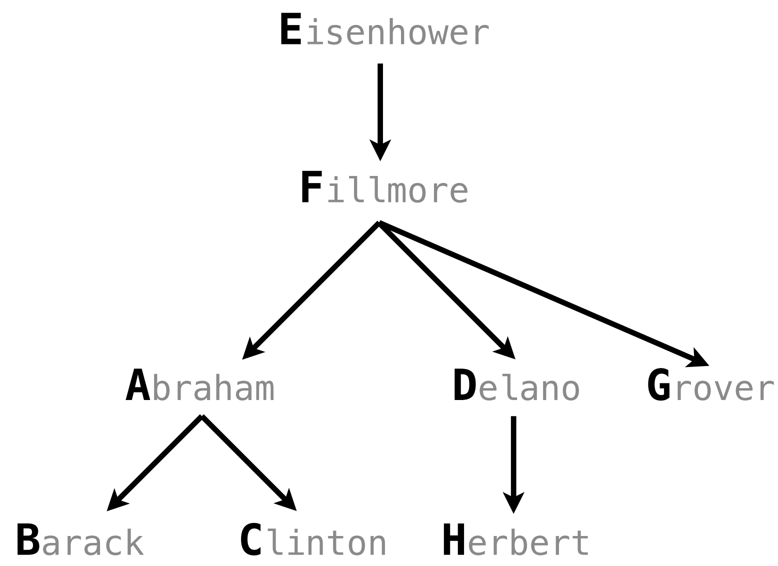
```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))

logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower

logic> (query (ancestor ?a barack)
              (ancestor ?a herbert))
Success!
```



Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

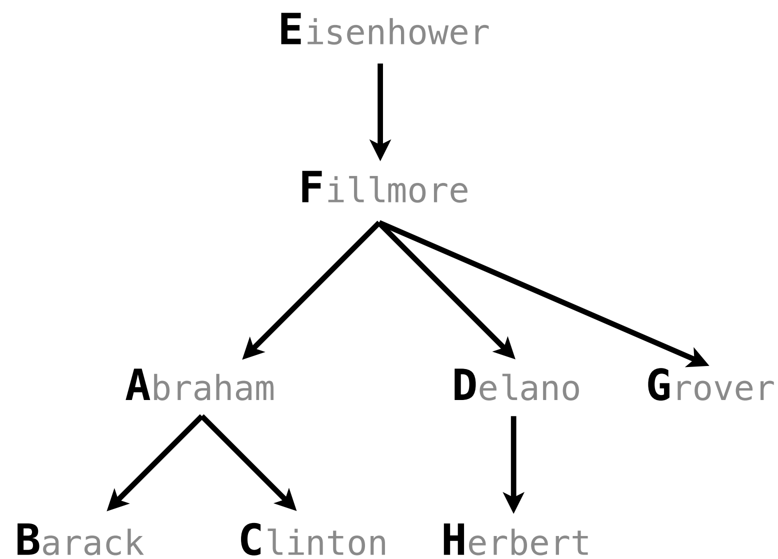
```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))

logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower

logic> (query (ancestor ?a barack)
              (ancestor ?a herbert))
Success!
a: fillmore
```



Recursive Facts

A fact is recursive if the same relation is mentioned in a hypothesis and the conclusion.

```
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))

logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower

logic> (query (ancestor ?a barack)
              (ancestor ?a herbert))
Success!
a: fillmore
a: eisenhower
```



Searching to Satisfy Queries

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
logic> (fact (parent delano herbert))
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
```

```
logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
```

```
logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))
```

```
(parent delano herbert)     ; (1), a simple fact
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))
```

```
(parent delano herbert)     ; (1), a simple fact
(ancestor delano herbert)   ; (2), from (1) and the 1st ancestor fact
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))
```

```
(parent delano herbert)      ; (1), a simple fact
(ancestor delano herbert)    ; (2), from (1) and the 1st ancestor fact
(parent fillmore delano)     ; (3), a simple fact
```

Searching to Satisfy Queries

The Logic interpreter performs a search in the space of relations for each query to find satisfying assignments.

```
logic> (query (ancestor ?a herbert))
Success!
a: delano
a: fillmore
a: eisenhower
logic> (fact (parent delano herbert))
logic> (fact (parent fillmore delano))
logic> (fact (ancestor ?a ?y) (parent ?a ?y))
logic> (fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))
```

```
(parent delano herbert)      ; (1), a simple fact
(ancestor delano herbert)    ; (2), from (1) and the 1st ancestor fact
(parent fillmore delano)     ; (3), a simple fact
(ancestor fillmore herbert)  ; (4), from (2), (3), & the 2nd ancestor fact
```

Hierarchical Facts

Hierarchical Facts

Hierarchical Facts

Relations can contain relations in addition to symbols.

Hierarchical Facts

Relations can contain relations in addition to symbols.

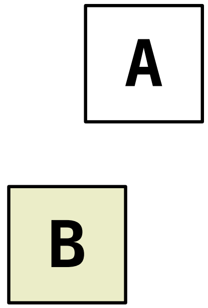
```
logic> (fact (dog (name abraham) (color white)))
```

A

Hierarchical Facts

Relations can contain relations in addition to symbols.

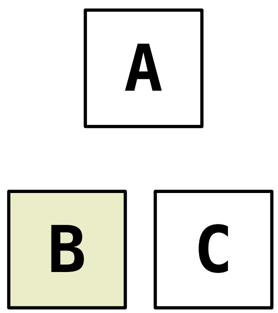
```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
```



Hierarchical Facts

Relations can contain relations in addition to symbols.

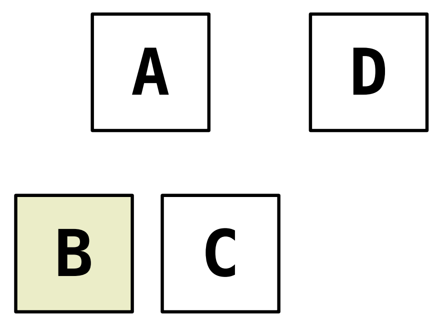
```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
```



Hierarchical Facts

Relations can contain relations in addition to symbols.

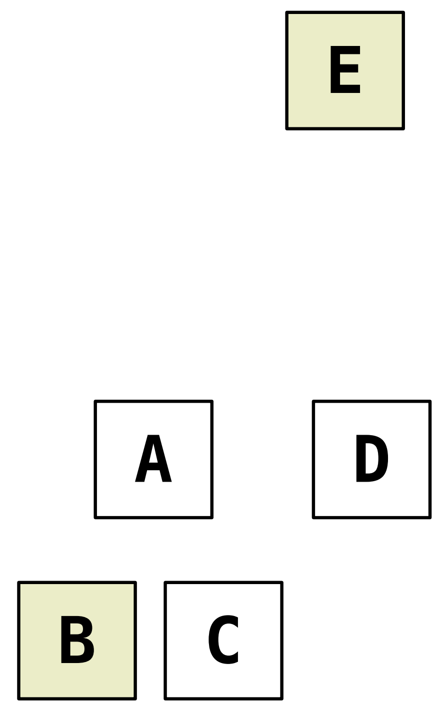
```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
```



Hierarchical Facts

Relations can contain relations in addition to symbols.

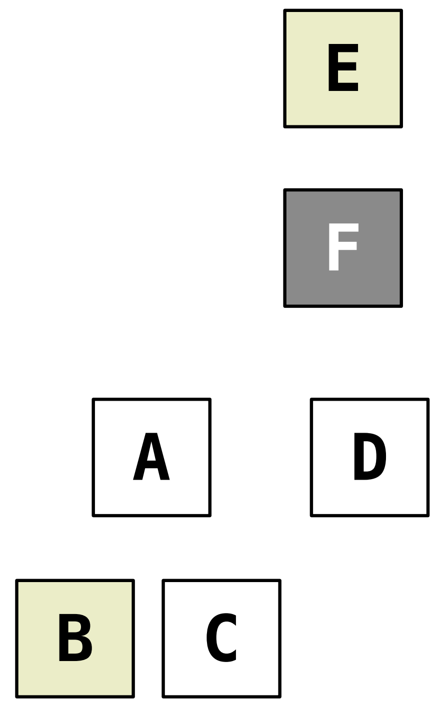
```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
```



Hierarchical Facts

Relations can contain relations in addition to symbols.

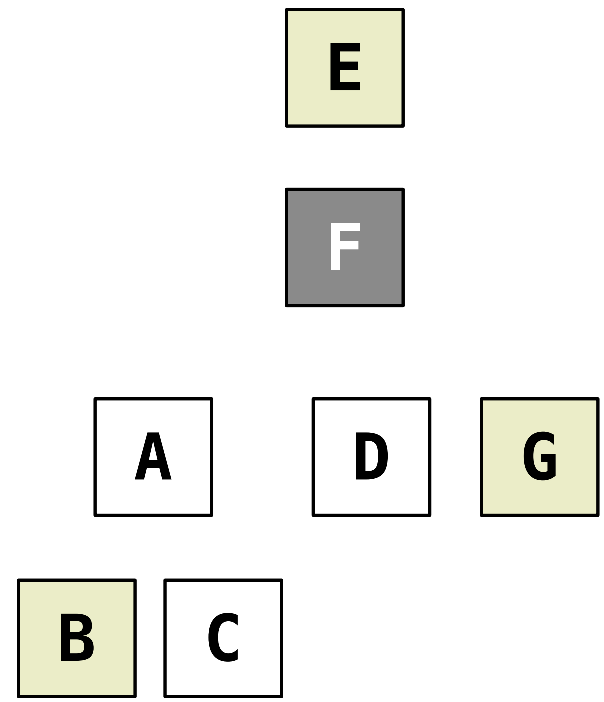
```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
```



Hierarchical Facts

Relations can contain relations in addition to symbols.

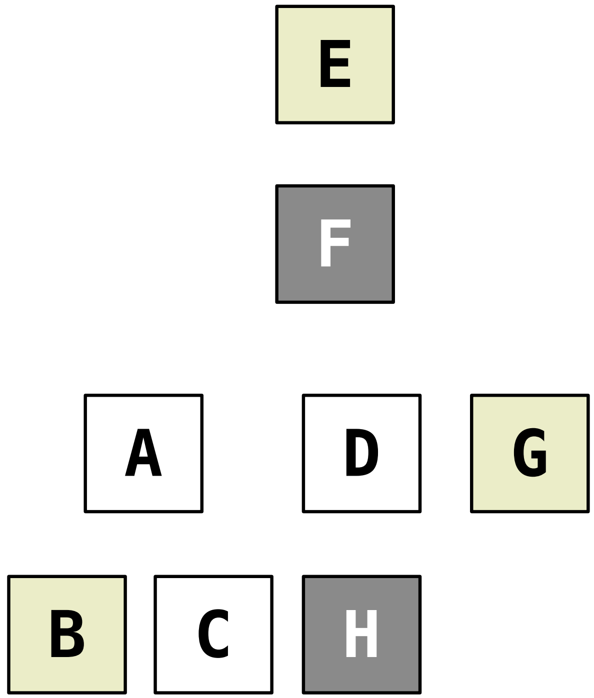
```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
```



Hierarchical Facts

Relations can contain relations in addition to symbols.

```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))
```

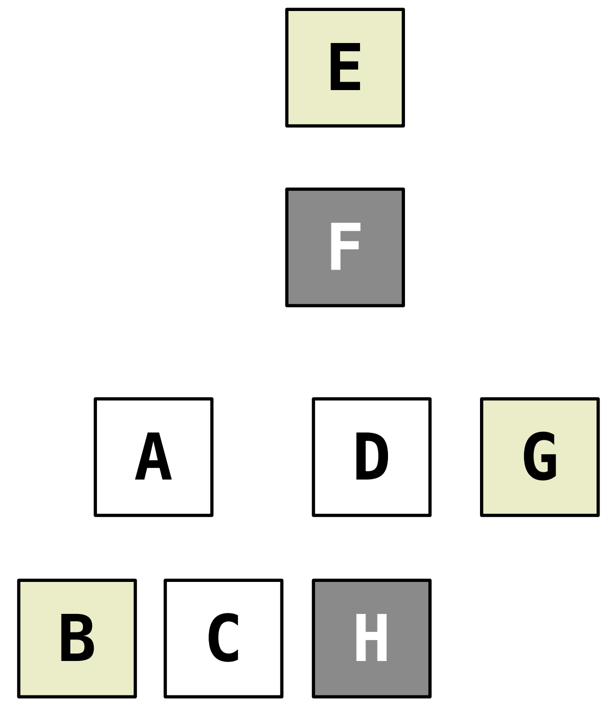


Hierarchical Facts

Relations can contain relations in addition to symbols.

```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))
```

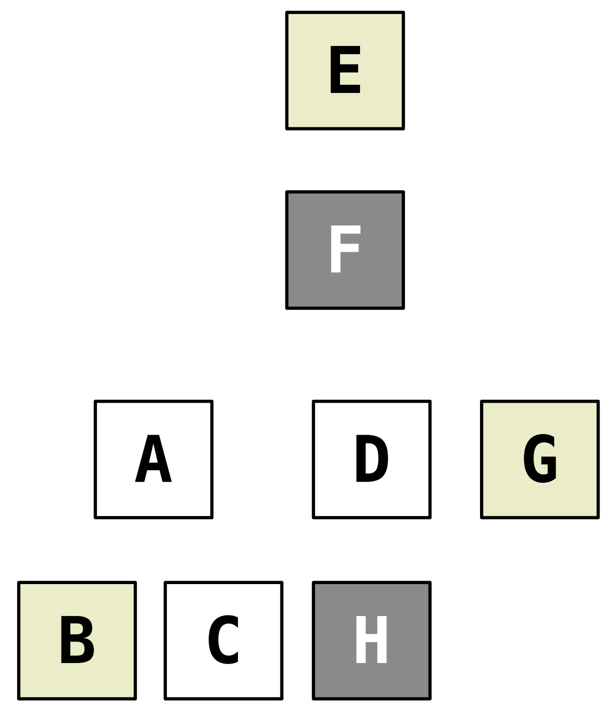
Variables can refer to symbols or whole relations.



Hierarchical Facts

Relations can contain relations in addition to symbols.

```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))
```



Variables can refer to symbols or whole relations.

```
logic> (query (dog (name clinton) (color ?color)))
```

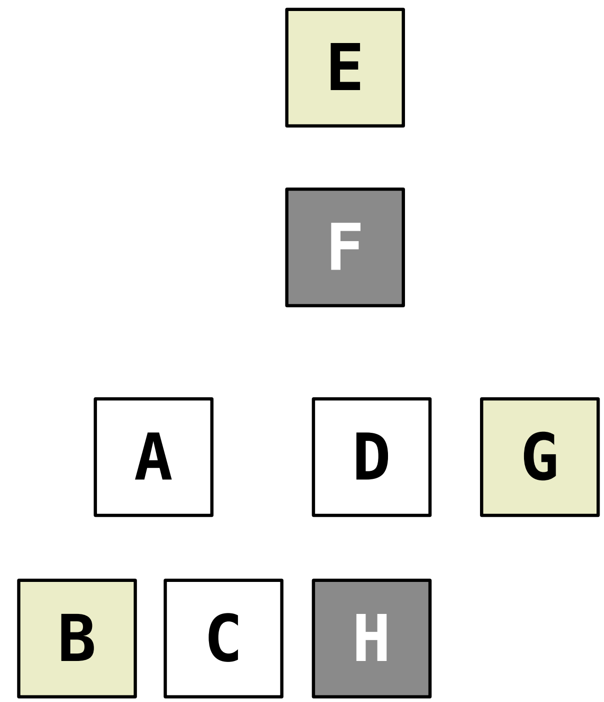
Hierarchical Facts

Relations can contain relations in addition to symbols.

```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))
```

Variables can refer to symbols or whole relations.

```
logic> (query (dog (name clinton) (color ?color)))
Success!
```



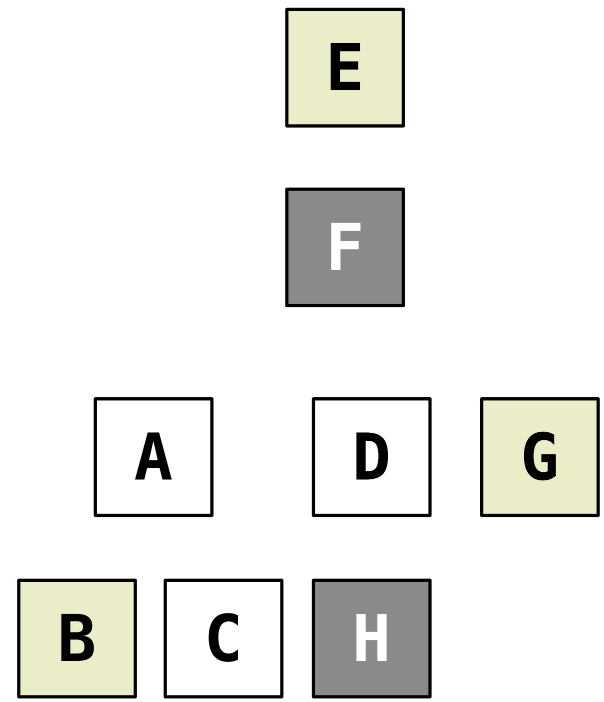
Hierarchical Facts

Relations can contain relations in addition to symbols.

```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))
```

Variables can refer to symbols or whole relations.

```
logic> (query (dog (name clinton) (color ?color)))
Success!
color: white
```



Hierarchical Facts

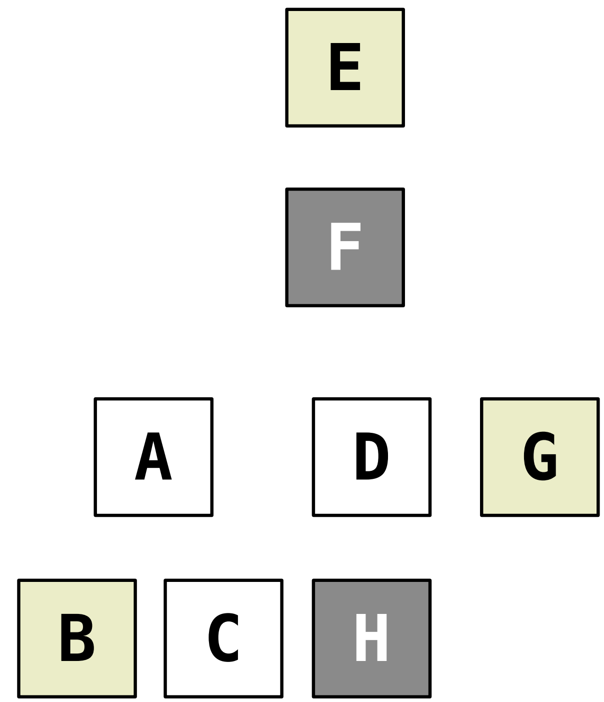
Relations can contain relations in addition to symbols.

```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))
```

Variables can refer to symbols or whole relations.

```
logic> (query (dog (name clinton) (color ?color)))
Success!
color: white
```

```
logic> (query (dog (name clinton) ?stats))
```



Hierarchical Facts

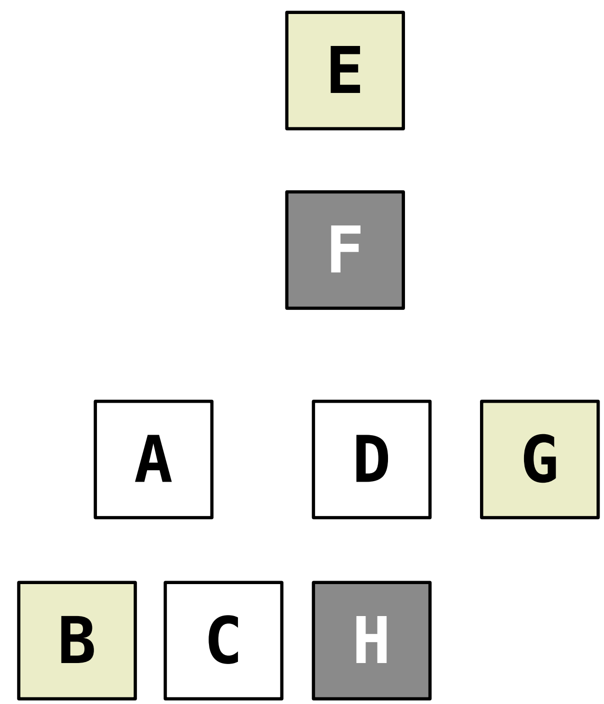
Relations can contain relations in addition to symbols.

```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))
```

Variables can refer to symbols or whole relations.

```
logic> (query (dog (name clinton) (color ?color)))
Success!
color: white
```

```
logic> (query (dog (name clinton) ?stats))
Success!
```



Hierarchical Facts

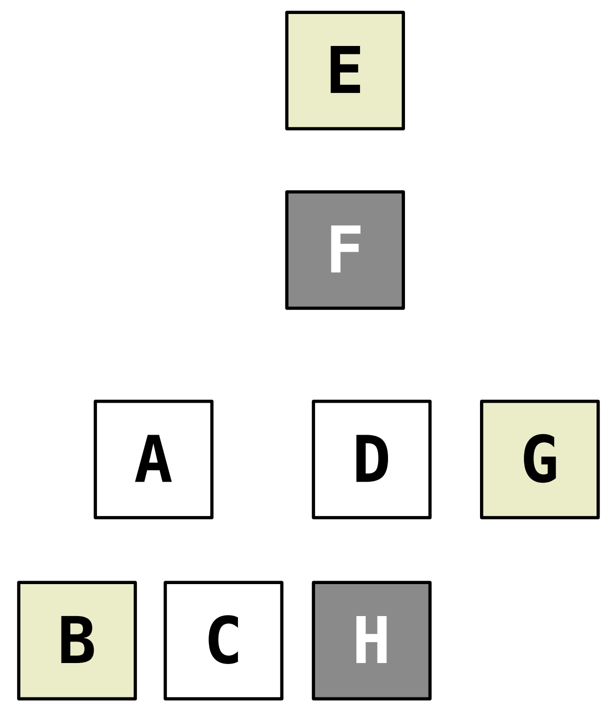
Relations can contain relations in addition to symbols.

```
logic> (fact (dog (name abraham) (color white)))
logic> (fact (dog (name barack) (color tan)))
logic> (fact (dog (name clinton) (color white)))
logic> (fact (dog (name delano) (color white)))
logic> (fact (dog (name eisenhower) (color tan)))
logic> (fact (dog (name fillmore) (color gray)))
logic> (fact (dog (name grover) (color tan)))
logic> (fact (dog (name herbert) (color gray)))
```

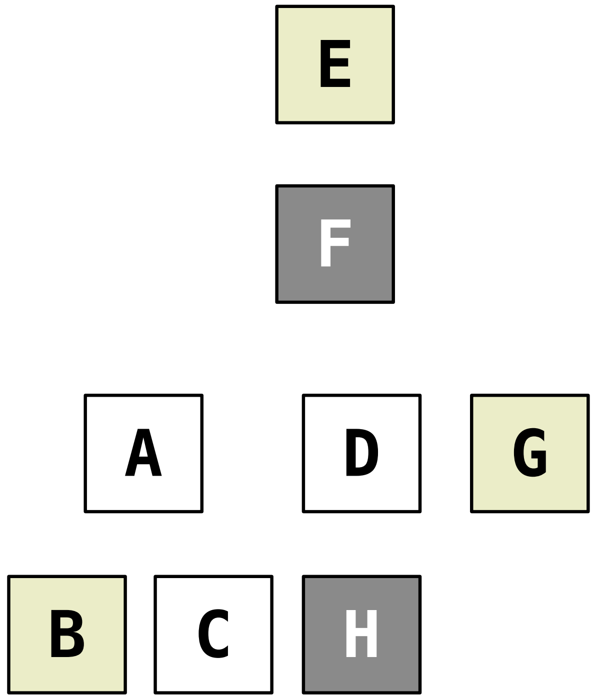
Variables can refer to symbols or whole relations.

```
logic> (query (dog (name clinton) (color ?color)))
Success!
color: white
```

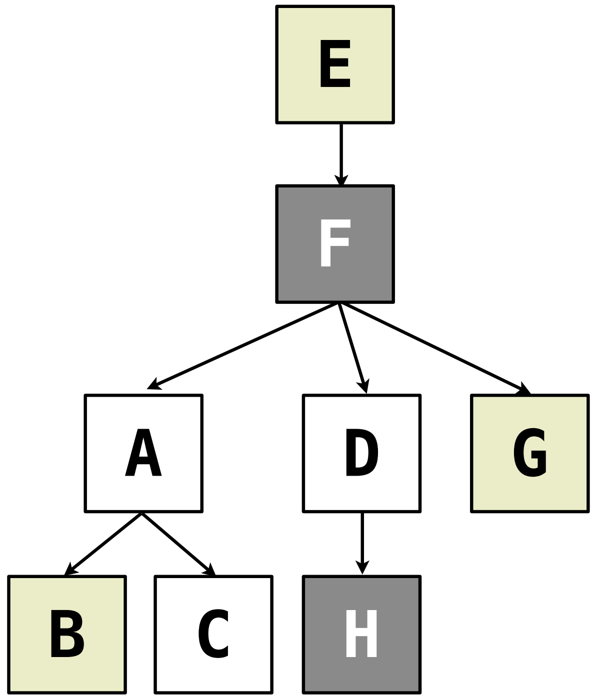
```
logic> (query (dog (name clinton) ?stats))
Success!
stats: (color white)
```



Combining Multiple Data Sources

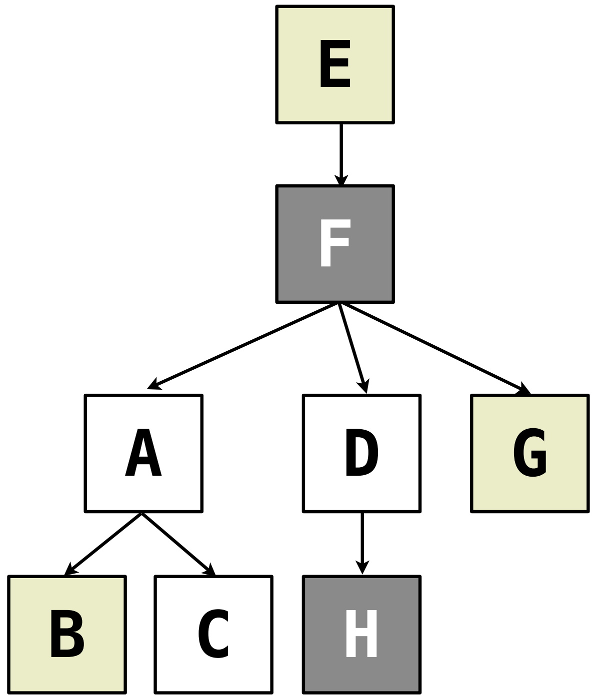


Combining Multiple Data Sources



Combining Multiple Data Sources

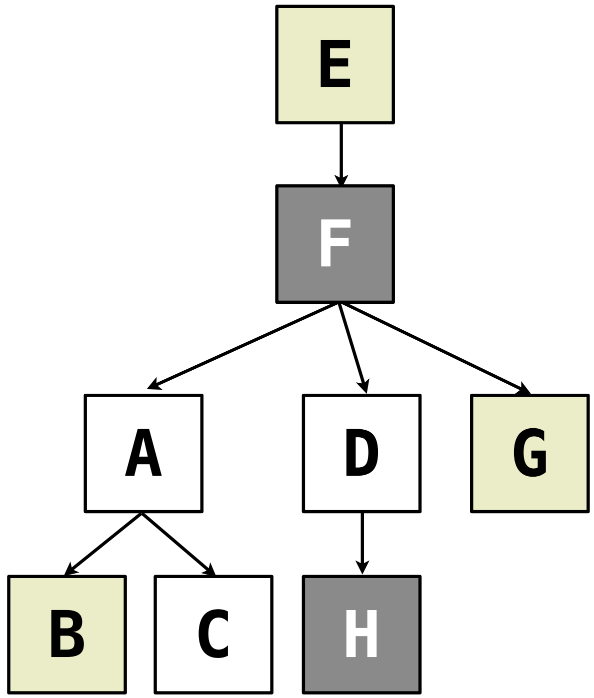
`Which dogs have an ancestor of the same color?`



Combining Multiple Data Sources

`Which dogs have an ancestor of the same color?`

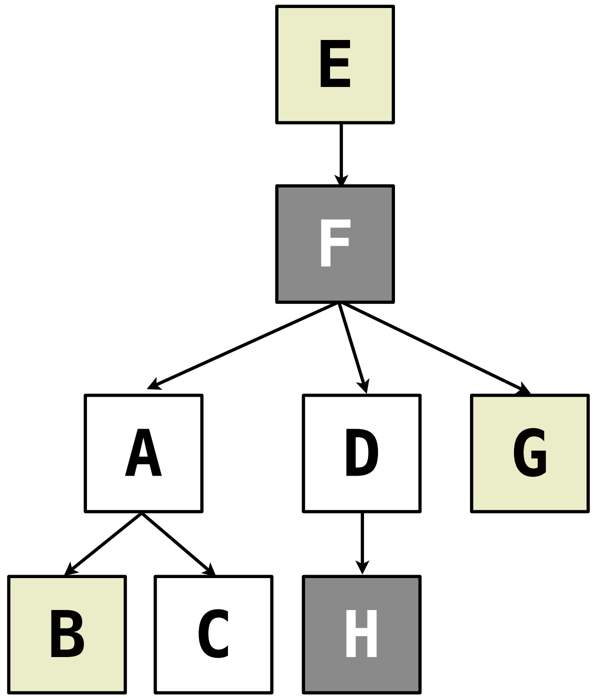
```
logic> (query (dog (name ?x) (color ?fur))
```



Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

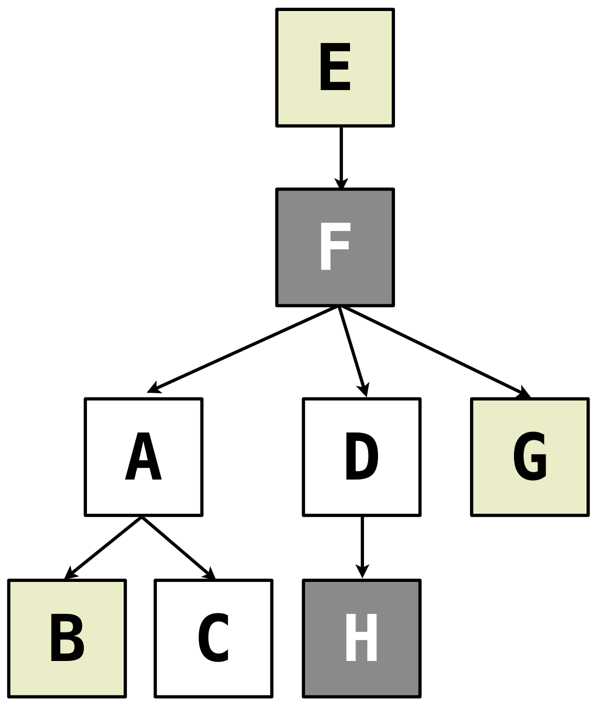
```
logic> (query (dog (name ?x) (color ?fur))
              (ancestor ?y ?x)
```



Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

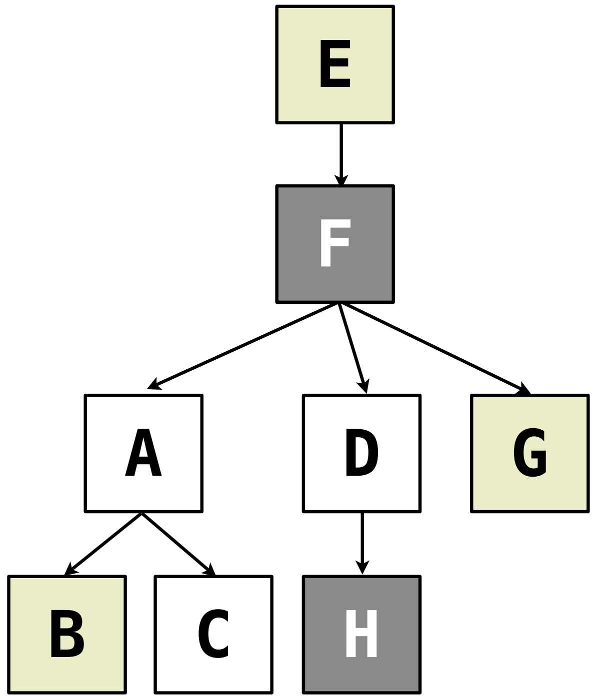
```
logic> (query (dog (name ?x) (color ?fur))
              (ancestor ?y ?x)
              (dog (name ?y) (color ?fur)))
```



Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

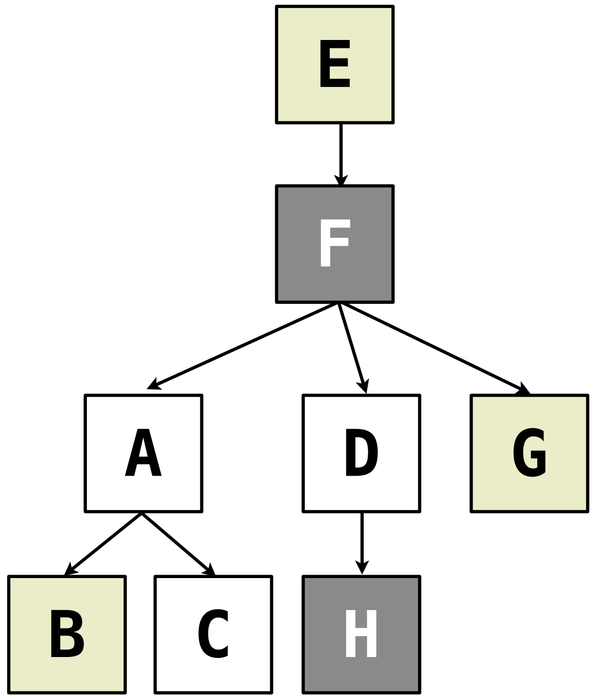
```
logic> (query (dog (name ?x) (color ?fur))
              (ancestor ?y ?x)
              (dog (name ?y) (color ?fur)))
Success!
```



Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

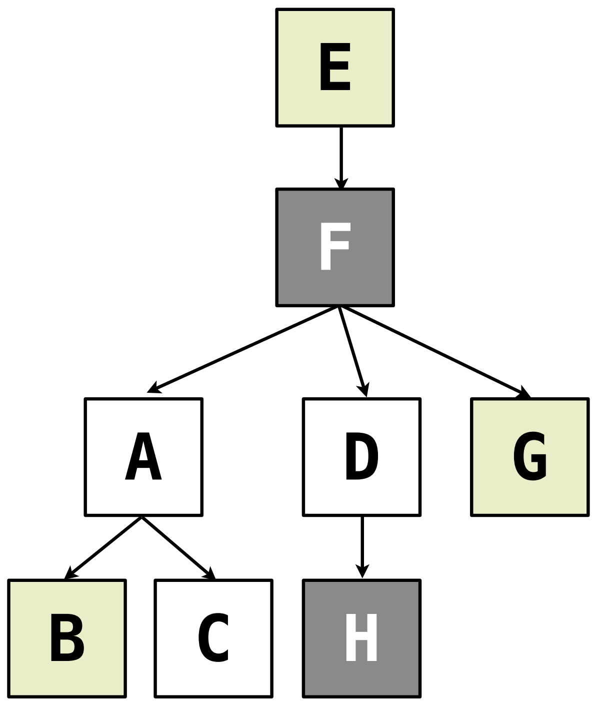
```
logic> (query (dog (name ?x) (color ?fur))
              (ancestor ?y ?x)
              (dog (name ?y) (color ?fur)))
Success!
x: barack    fur: tan     y: eisenhower
```



Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

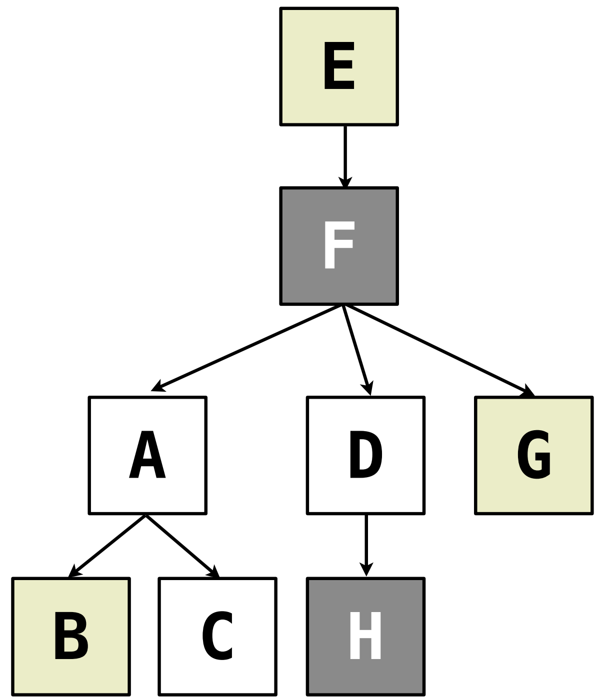
```
logic> (query (dog (name ?x) (color ?fur))
              (ancestor ?y ?x)
              (dog (name ?y) (color ?fur)))
Success!
x: barack    fur: tan     y: eisenhower
x: clinton   fur: white   y: abraham
```



Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

```
logic> (query (dog (name ?x) (color ?fur))
              (ancestor ?y ?x)
              (dog (name ?y) (color ?fur)))
Success!
x: barack    fur: tan     y: eisenhower
x: clinton   fur: white   y: abraham
x: grover    fur: tan     y: eisenhower
```



Combining Multiple Data Sources

Which dogs have an ancestor of the same color?

```
logic> (query (dog (name ?x) (color ?fur))
              (ancestor ?y ?x)
              (dog (name ?y) (color ?fur)))
Success!
x: barack    fur: tan      y: eisenhower
x: clinton   fur: white    y: abraham
x: grover    fur: tan      y: eisenhower
x: herbert   fur: gray     y: fillmore
```