[**12 Individuals and Relations**](#_gus8oj3340c2) **1**

[12.1 Exploiting Structure Beyond Features](#_n1f16xv2mk9v) 1

[12.2 Symbols and Semantics](#_pggp2dqfyz0z) 1

[12.3 Datalog: A Relational Rule Language](#_kajah79p5qyl) 1

[12.3.1 Semantics of Ground Datalog](#_qivcs14unz02) 2

[12.3.2 Interpreting Variables](#_hkjh8le90txb) 3

[12.3.3 Queries with Variables](#_sn8ins6md9fn) 4

[12.4 Proofs and substitutions](#_32bwj1qmui41) 4

[✏️ - Exercise Session 7 - 1: Unification](#_f4bw6zx9tan) 5

[12.4.1 Bottom-up Procedure with Variables](#_tdn5o331ns44) 6

[📖 - Bottom-up Proof Procedure](#_pf84cb4zihm9) 7

[📖 - Top-down Proof Procedure (propositional)](#_9k1pjzgzopak) 8

[✏️ - Exercise Session 7 - 2: Bottom-up and Top-down](#_3fo4j4i3i9e7) 10

[12.4.2 Definite Resolution with Variables](#_1ribyzhyguko) 12

[12.5 Function Symbols](#_6dfqy2rpsu8n) 15

[12.5.1 Proof Procedures with Function Symbols](#_3ght91vslbu8) 15

[✏️ - Exercise session 8 - DLV](#_ugu5b928eg5) 16

[Exercise 1 - DLV Semantic](#_y8vh9dipaer0) 16

[🧮 - Book Exercises](#_k758qdy5didp) 21

[📁 - .PDF exercise](#_avqsvgocjjc6) 31

# 12 Individuals and Relations

## 12.1 Exploiting Structure Beyond Features

* **Individuals**: things in the domain
* **Relations**: what is true about these individuals. Can be a unary relationship or multiple individuals. The relation can be true or false

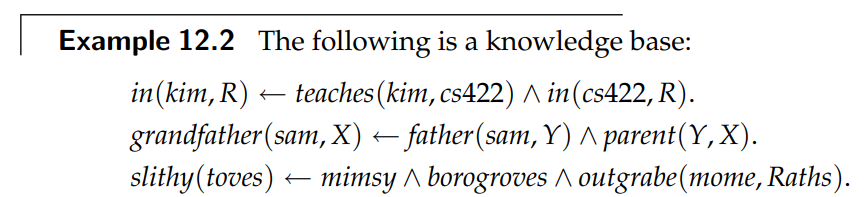
## 12.2 Symbols and Semantics

What is the correct answer is defined independently of how it is computed. The correctness of a knowledge base is defined by the semantic

## 12.3 Datalog: A Relational Rule Language

* **Variable**: word starting with upper-case letter or the underscore. *X, Room, B4, Raths*
* **Constant**: word starting with lower-case letter, or a number constant or a string
* **Predicate symbol**: word starting with lower-case letter. Distinguible by the constants by their context. *Ki, r123, grandfather and hejduk* can be constants or predicate, while *725* is a constant
* **Term**: either a variable or a constant
* **Atom (or atomic symbol)**: is a predicate with terms inside *teaches(sue, math), sunny, father(bill, Y), happy(mike)*
* **Body**: atom or conjunction of atoms
* **Define clause**: either an atom (named atomic clause) or of the form a ← b called rule, where a is the head and b is the body each clauses ends with .
* **Knowledge base**: set of define clauses
* **Query**: is of the form *ask b*where *b* is a body
* **Expression**: either a term, or atom, a definite clause or a query

To comment: use %



An expression is ground if it does not contain any variables. *teaches(fred, cs322)* is ground, *teaches(Bob, math)* is not ground

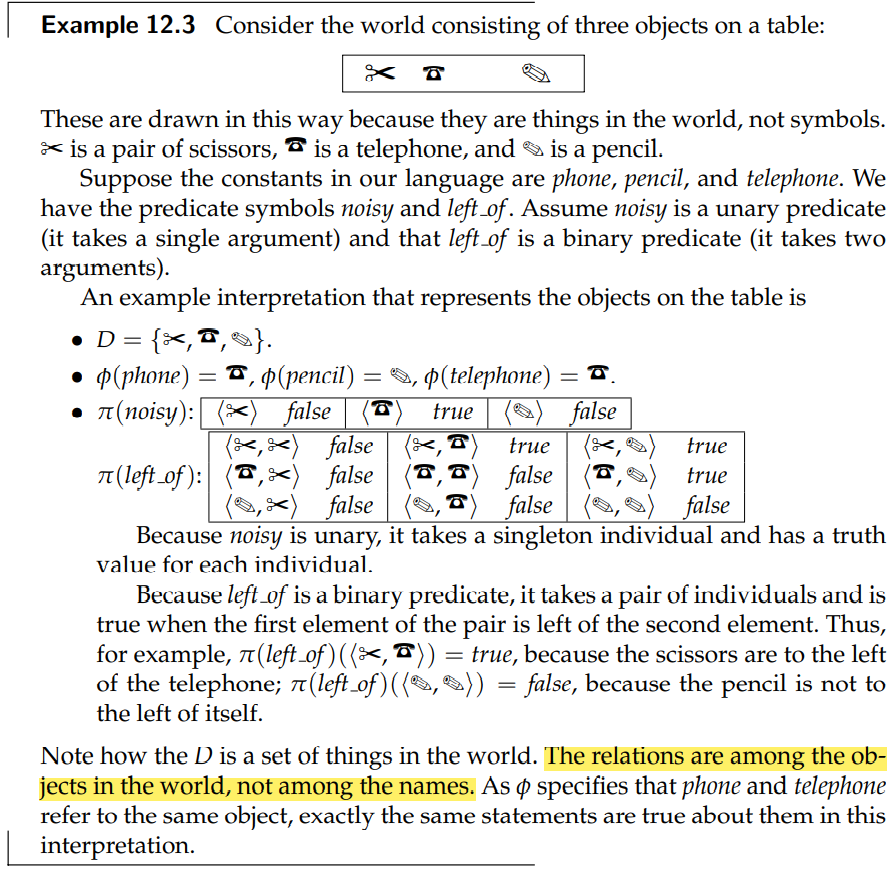
### 12.3.1 Semantics of Ground Datalog

An interpretation is a triple *I* =〈 D, Φ, π 〉where,

* D is a non-empty set called the domain. Elements of D are individuals
* Φ is a mapping that assign to each constant an element of D
* π is a mapping that assign to each n-ary predicate symbols a function from Dn into {true, false}

Φ is a function from names into individuals in the world. The constant *c* is said to **denote** the individual Φ(c)

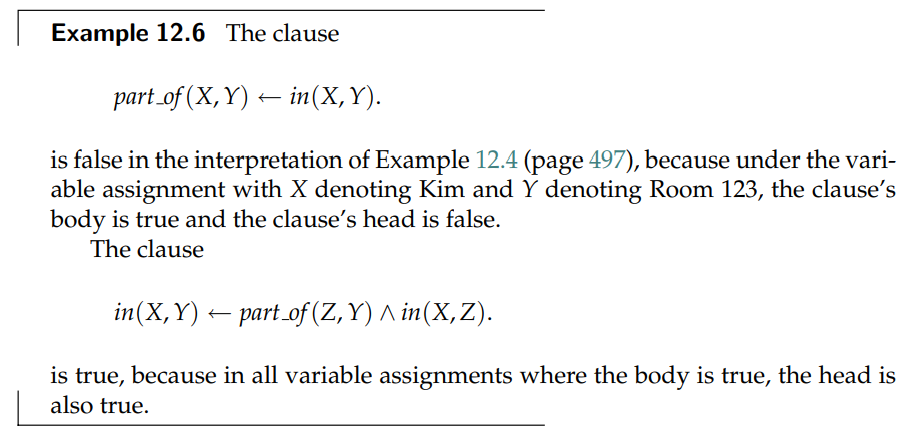
π(p) specifies whether the relation denoted by n-ary predicate symbol p is true or false for each n-tuple of individuals



### 12.3.2 Interpreting Variables

If a variable X appears in a clause C, then claiming that C is true in an interpretation means that C is true no matter which individual from the domain is denoted by X

**Universal quantification**: a clause is true in an interpretation if is true for all variables assignments, the variables are said to be universally quantified in the scope of the clause



### 12.3.3 Queries with Variables

Queries are used to ask whether something is a logical consequence of a knowledge base.

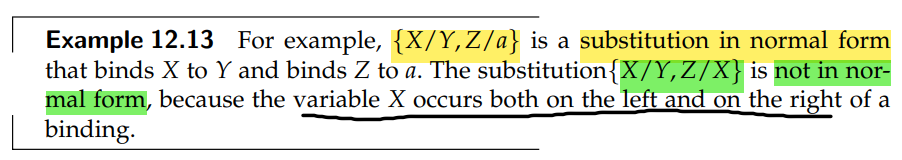
* Propositional queries: a user can ask yes-or-no questions
* Queries with variables: the system returns the values of the variables that make the query a logical consequence of the knowledge base

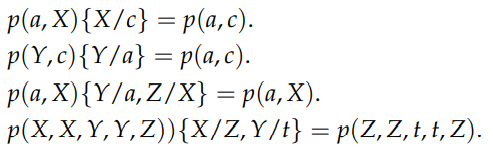
## 12.4 Proofs and substitutions

An instance of a clause is obtained by uniformly substituting terms for variables in the clause. All occurences of a particular variable are replaced by the same term

**Substitution**: finite set of the form {V1/t1, …, Vn/tn} where each Vi is a distinct variable and each ti is a term. The element Vi is a binding for variable Vi. Substitutions can apply to clause, atoms and terms

A substitution is in **normal form** if no Vi appears in any tj





The substitution {X/Y, Z/a} of the clause p(X,Y) ← q(a, Z, X, Y, Z) is the clause p(Y,Y) ← q(a, a, Y, Y, a)

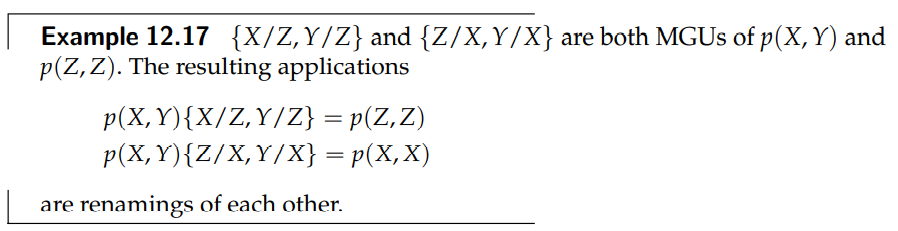
**Unifier**: a substitution that when applied to each expression gives the same expression. Expression can have many unifiers

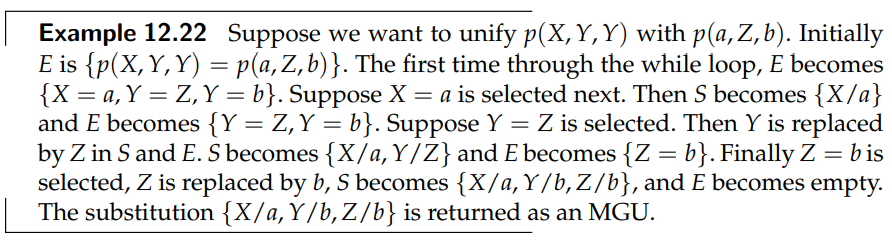
{X/a, Y/b} is a unifier of t(a, Y, c) and t(X, b, c) as

t(a, Y, c) {X/a, Y/b} = t(a, b, c)

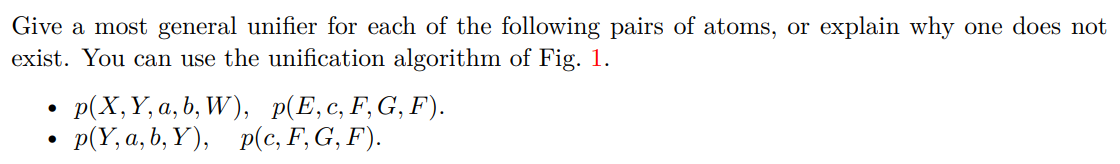
t(X, b, c} {X/a, Y/b} = t(a, b, c)

**Most general unifier (MGU)** is a unifier that maps everything to the same variable. Are like {X/Z, Y/Z}. If two expressions have a unifier, they have also at least one MGU





### ✏️ - Exercise Session 7 - 1: Unification



*E* = { p(X, Y, a, b, W) = p(E, c, F, G, F) } and S = {}

So we have {X = E, Y = c, a = F, b = G, W = F} and we specify that all X must be substituted with E, that is S = {X/E}. We can then remove X = E from the set *E*.

*E* = {Y = c, a = F, b = G, W = F}, also Y should be substituted with c, that is S = {X/E, Y/c} and *E* becomes *E* = {a = F, b = G, W = F}

F should be substituted with a, that is S = {X/E, Y/c, F/a} and E = {b = G, W = F}

G should be substituted with b, that is S = {X/E, Y/c, F/a, G/b} and E = {W = F}

Now we have that W should be substituted with F, but F is substituted with a, so W/a, that is S = {X/E, Y/c, F/a, G/b, W/a} and E = {}

E = { p(Y, a, b, Y ) = p(c, F, G, F)} and S = {}, that is E = {Y = c, a = F, b = G, Y = F}

Y should be substituted with c, that is E = {a = F, b = G, Y = F} and S = {Y/c}

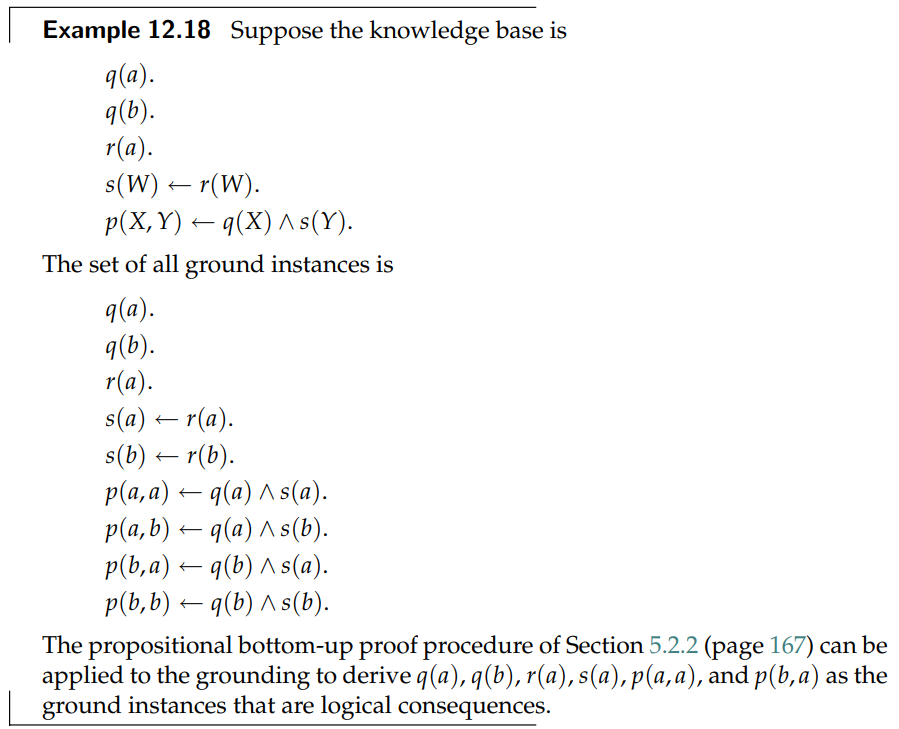
F should be substituted with a, that is E = {b = G, Y = F} and S = {Y/c, F/a}

G should be substituted with b, that is E = {Y = F} and S = {Y/c, F/a, G/b}

Y should be substituted with F, but we already have Y/c and F/a, since F and Y should be mapped to the same value, there is an inconsistency and it cannot be unified.

### 12.4.1 Bottom-up Procedure with Variables

**Ground instance**: obtained by substituting costants for variables in a clause. The constants are those appearing in the knowledge base or in the query. If there are no constants in the query you must invent one .

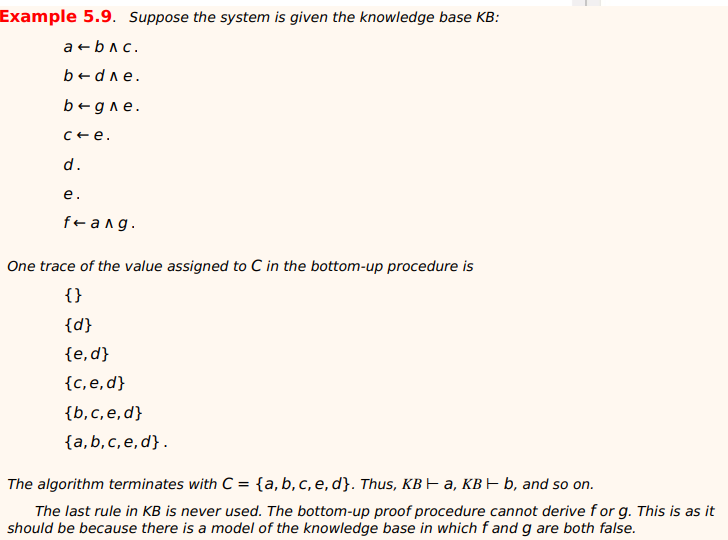


### 📖 - Bottom-up Proof Procedure

Can be used to derive all logical consequences of a knowledge base. Is called bottom-up because each part is built on the structure already complete. It builds on atoms that have already been established.

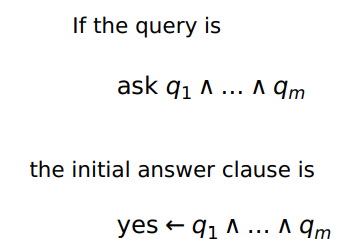
It is based on **rule of derivation**: *if “ h ← a1 ^ 12 ^ … ^ am” is a definite clause in the knowledge base, and each ai has been derived, then h can be derived*.

An atomic clause corresponds to the case of m = 0 and can always be immediately derive in the knowledge base



### 📖 - Top-down Proof Procedure (propositional)

This approach starts from a query and tries to find definite clauses that support the query. The top-down proof procedure can be understood in terms of the **answer clause**. An answer clause is of the form *yes ← a1 ^ a2 ^ … ^ am* where *yes* is a special atom that is going to be true exactly when the answer to the query is “yes”



The top-down algorithm selects an atom in the body of the answer clause (suppose a1), the atom is called subgoal. The algorithm proceeds by doing steps of resolution. Then it chooses a definite clause in the knowledge base with a1 as the head. If there is no such clause, the algorithm fails.

Note that any atom in the body can be selected, and if one selection does not lead to a proof, other selections do not need to be tried.

If we have

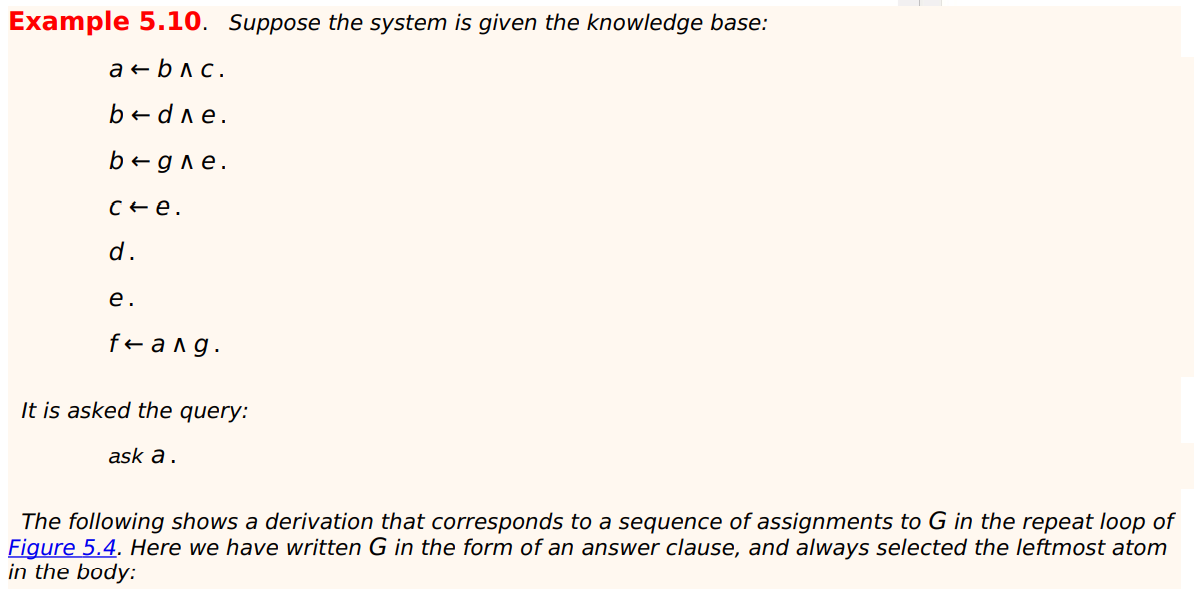
*yes ← a1 ^ a2 ^ … ^ am*

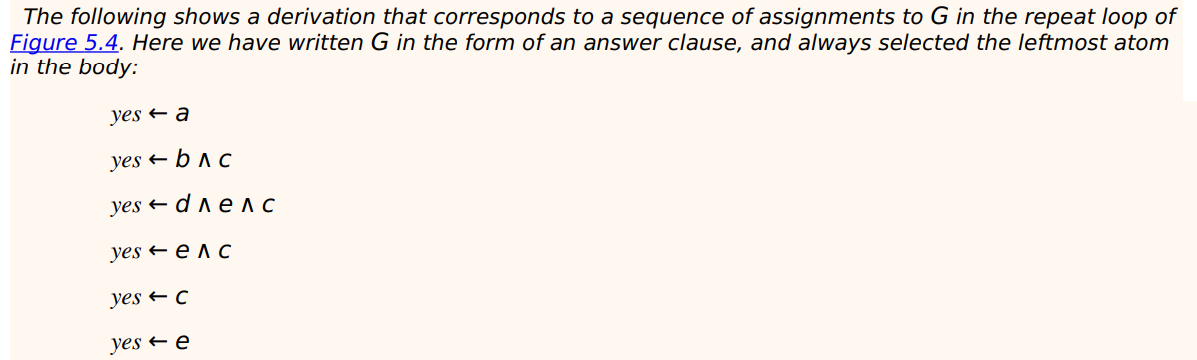
*a1 ← b1 ^ … bp*

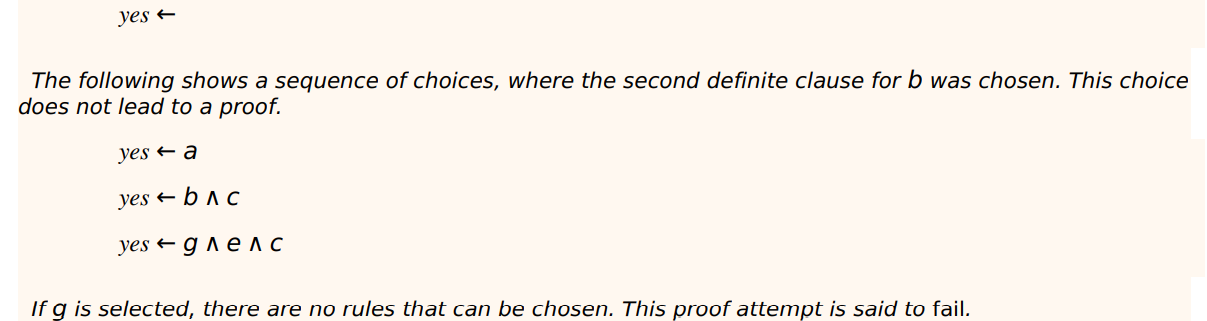
Then the answer clause is

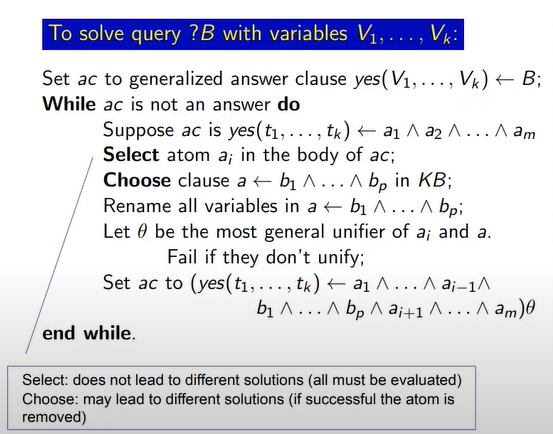
*yes ← b1 ^ … bp ^ a2 ^ … ^ am*

The subgoal in the answer clause is replaced by the body of the chosen definite clause

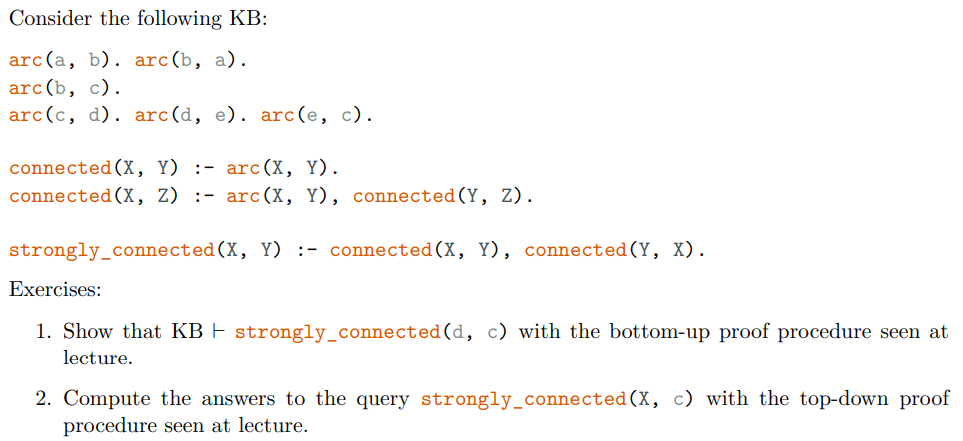








### ✏️ - Exercise Session 7 - 2: Bottom-up and Top-down



**Number 1. Bottom-up**

Start with C = ø, and select all the clauses that must be always satisfied, that is

C = { arc(a,b), arc(b,a), arc(b,c), arc(c,d), arc(d,e), arc(e,c) }

From the clause *connected(X,Y) :- arc(X,Y)* we get

C = { arc(a,b), arc(b,a), arc(b,c), arc(c,d), arc(d,e), arc(e,c), connected(a,b), connected(b,a), connected(b,c), connected(c,d), connected(d,e), connected(e,c) }

(Since we are looking to see if the clause *strongly\_connected(d,c)* is present in the knowledge base, we can stop deriving everything and just focus on what we actually need)

From the clause *connected(X,Z) :- arc(X,Y), connected(Y,Z)* we get:

C = { arc(a,b), arc(b,a), arc(b,c), arc(c,d), arc(d,e), arc(e,c), connected(a,b), connected(b,a), connected(b,c), connected(c,d), connected(d,e), connected(e,c), … , connected(d,c)}

From the clause *strongly\_connected(X,Y) :- connected(X,Y), connected(Y,X)* by substituting X/d, Y/c, we get:

C = { arc(a,b), arc(b,a), arc(b,c), arc(c,d), arc(d,e), arc(e,c), connected(a,b), connected(b,a), connected(b,c), connected(c,d), connected(d,e), connected(e,c), … , connected(d,c), **strongly\_connected(c,d)**}

So we have KB |- **strongly\_connected(c,d)**

**Number 2. Top-down**

First of al name all the clauses in the program

1. arc(a,b).
2. arc(b,a).
3. arc(b,c).
4. arc(c,d).
5. arc(d,e).
6. arc(e,c).
7. connected(X,Y) :- arc(X,Y)
8. connected(X,Z) :- arc(X,Y), connected(Y,Z)
9. strongly\_connected(X,Y) :- connected(X,Y), connected(Y,X)

Remember:

* Select: does not lead to different solutions (all must be evaluated)
* Choose: may lead to different solutions (if successful the atom is removed)

*ac* (Answer Set) is *yes(X) :- strongly\_connected(X,c)*

1. Select *strongly\_connected(X,c}*[[1]](#footnote-0). Choose clause (9) and rename the variables such that it becomes *strongly\_connected(X0,Y0) :- connected(X0,Y0), connected(Y0,X0)*. Then apply the mgu θ = {X/X0 , Y0/c} between *strongly\_connected(X,c)* and *strongly\_connected(X0,Y0)[[2]](#footnote-1)*

*yes(X0) :- connected(X0,c), connected(c,X0)*

1. Select *connected(X0,c)*, choose clause (7) and rename the variables such that it becomes *connected(X1,Y1) :- arc(X1,Y1)*. Then apply the mgu between *connected(X0,c)* and *connected(X1,Y1)*, θ = {X0/X1, Y1/c}[[3]](#footnote-2)

yes(X1) *:- arc(X1,c), connected(c,X1)*

1. Select *arc(X1,c)* and choose the clause (6). Apply the mgu between *arc(X1,c)* and *arc(e,c)*. θ = {X1/e}.

*yes(e) :- arc(e,c), connected(c, e)*

We can remove *arc(e,c)* because it is the clause (6), we have

*yes(e) :- connected(c, e)*

1. Select *connected(c, e)* and choose clause (8) and rename the variables such that it becomes *connected(X2,Z2) :- arc(X2,Y2), connected(Y2,Z2)*. Then apply the mgu between *connected(c, e)* and *connected(X2,Z2)*. θ = {X2/c, Z2/e}

yes(e) :- *arc(c,Y2), connected(Y2,e)*

1. Select *arc(c,Y2)* and choose clause (4). Apply the mgu between *arc(c,Y2)* and *arc(c,d)*. θ = {Y2/d}

*yes(e) :- arc(c,d), connected(d,e)*

We can remove *arc(c,d)* because it is the clause (4), we have

*yes(e) :- connected(d,e)*

1. Select *connected(d,e)* and choose the clase (7) and rename the variables such that it becomes *connected(X3,Y3) :- arc(X3,Y3)*. Then apply the mgu between *connected(d,e)* and *connected(X3,Y3).* θ = {X3/d, Y3/e}

*yes(e) :- arc(d,e)*

1. Select *arc(d,e)* and clause (5). Then apply the mgu between *arc(d,e)* and *arc(d,e)*

*yes(e) :-*

So *e* is a possible substitution of the variable in *strongly\_connected(X,c)*

### 12.4.2 Definite Resolution with Variables

The propositional top-down procedure can be extended to the case with variables by allowing instances of rules to be used in the derivation

**Generalised answer clause**: is in the form *yes(t1, … , tk) ← a1 ^ a2 ^ … ^ am*

Where t1, …, tk are terms (either variables or constants X, bob … ) and a1, … am are atoms (predicate with terms inside *teaches(X, bob)* )

Use *yes* to do **answer extraction**: determining which instances of the query variables are a logical consequence of the knowledge base

Start with *yes(V1, … , Vk) ← q* where *V1, … , Vk* are the variables that appear in *q*. It means that *yes(V1, … , Vk)* is true if the corresponding instance of the query is true

At each state the algorithm selects an atom ai in the body of the generalised answer clause. Then it chooses a clause in the knowledge base whose head unifies with ai.

*yes(t1, … , tk) ← a1 ^ a2 ^ … ^ ai ^ am*

*ai ← b1 ^ … ^ bp*

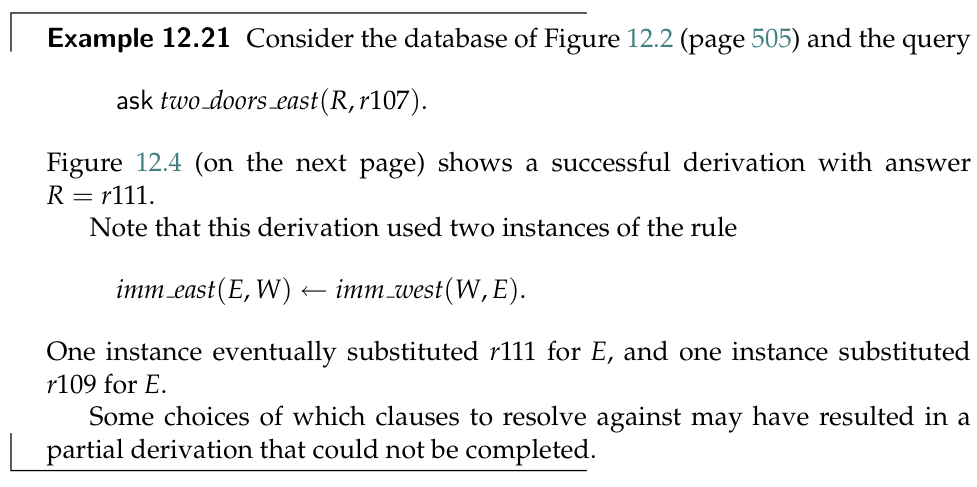
Becomes

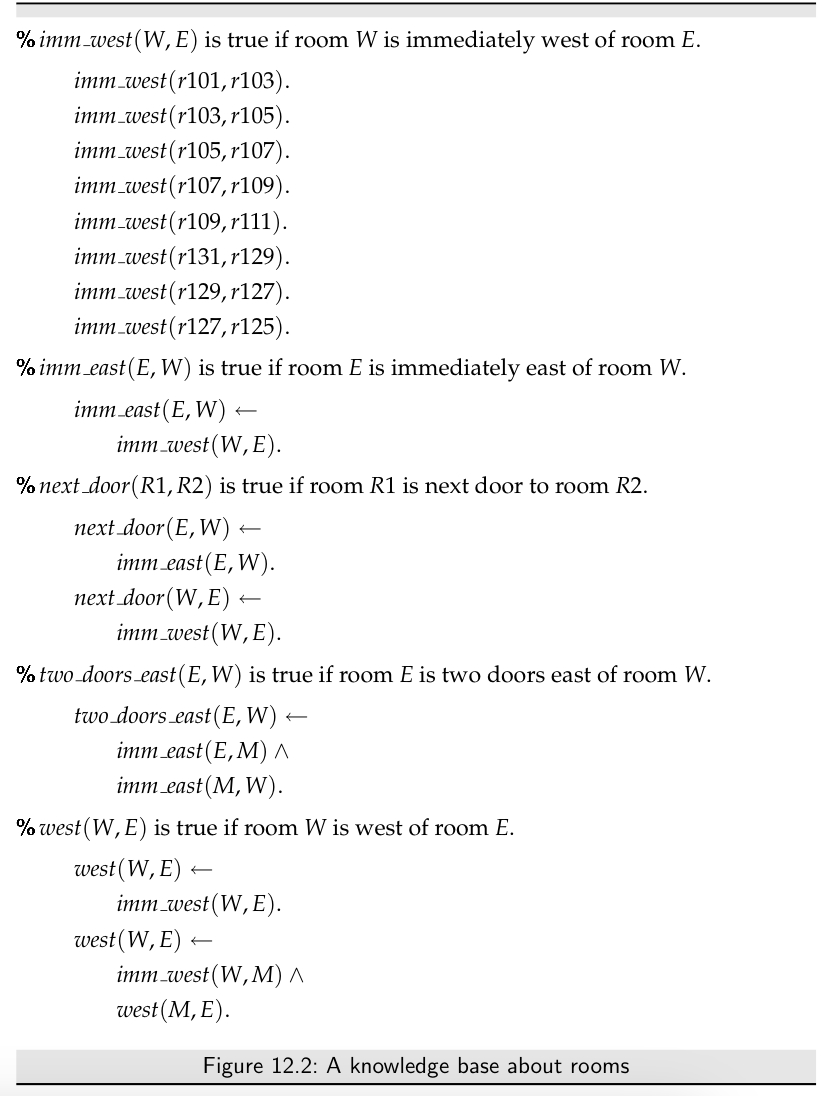
*yes(t1, … , tk) ← a1 ^ a2 ^ … ^ b1 ^ … ^ bp ^ am*

When the body of the chosen clause has replaced ai in the answer clause, apply MGU to the whole answer clause

The main difference between the variable top-down procedure and the propositional top-down procedure is that for clauses with variables the proof procedure must take *copies* of clauses from the knowledge base. The copying renames the variables in the clause with new names (in order to remove name clashes between variables)

If all choices fail, the algorithm fails and there are no derivations





Ask *two\_door\_east(R, r107)*.

Answer set (ac): *yes(R) :- two\_door\_east(R, r107)*

1. Select *two\_door\_east(R, r107)* and choose *two\_door\_east(E,W) :- imm\_east(E,M), imm\_east(M,W)* and rename the variables such that it becomes *two\_door\_east(E1,W1) :- imm\_east(E1,M1), imm\_east(M1,W1)*. Then apply the mgu between *two\_door\_east(R, r107)* and *two\_door\_east(E1,W1)*. θ = {R/E1, W1/r107}

*yes(E1) :- imm\_east(E1,M1), imm\_east(M1,*r107*)*

1. Select *imm\_east(E1,M1)* and choose *imm\_east(E,W) :- imm\_west(W,E)* and rename the variables such that it becomes *imm\_east(E2,W2) :- imm\_west(W2,E2)*. Then apply the mgu between *imm\_east(E1,M1)* and *imm\_east(E2,W2)*. θ = {E1/E2, M1/W2}

yes(E2) :- *imm\_west(W2, E2), imm\_east(W2,* r107*)*

1. Select *imm\_west(W2, E2)* and choose *imm\_west(r109, r111)*. Then apply the mgu between *imm\_west(W2, E2)* and *imm\_west(r109, r111)*. θ ={W2/r109, E2/r111}

yes(r111) :- *imm\_west(r109, r111), imm\_east(r109,* r107*)*

We can remove the clause *imm\_west(r109, r111)*, so

yes(r111) :- *imm\_east(r109,* r107*)*

1. Select *imm\_east(r109,* r107*)* and choose *imm\_east(E, W) :- imm\_west(W, E)* and rename the variables such that it becomes *imm\_east(E3, W3) :- imm\_west(W3, E3)*. Then apply the mgu θ = {E3/r109, W3/r107}

yes(r111) :- *imm\_west(r107,* r109*)*

We can remove the clause *imm\_west(r107, r111)*, so

yes(r111) :-

## 12.5 Function Symbols

Using constants to name each individual means that the knowledge base can only represent a finite number of individuals. However, many cases exist in which you want to reason about a potentially infinite set of individuals.

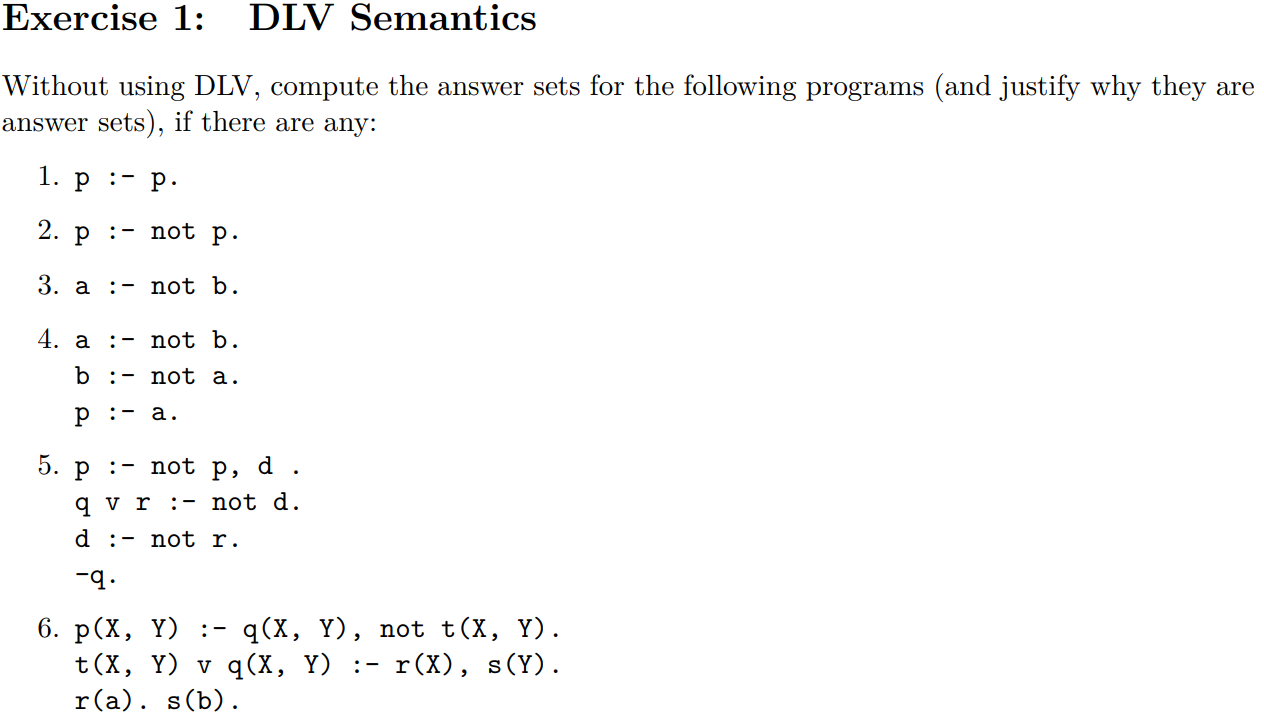
A **function symbol** is a word starting with a lower-case letter

### 12.5.1 Proof Procedures with Function Symbols

The use of function symbols involves infinitely many terms

## ✏️ - Exercise session 8 - DLV

### Exercise 1 - DLV Semantic



**p :- p.**

The only answer set is the empty interpretation {}. The set {p} is also a model but is not minimal, and so it is not an answer set

**p :- not p.**

Two possible choices

* {}

Compute the transformation, start with *p :- not p.*

*not p* is not in our interpretation and we can remove it, we are left with only *p*. But it is not a model because *p* should be inside the interpretation but it is not

* {p}

*p :- not p,* we need to remove this rule, left with an empty program. And p is not minimal, not an answer set

This particular program does not have any answer set

**a :- not b.**

* {}

a not an answer set

* {a}

start with *a :- not b* and delete *not b*. Now we only have *a*. This is a model

* {b}

start with *a :- not b* and delete *not b*. Now we only have *a*. This is not a model

* {a,b}

Not a model

**a :- not b.**

**b :- not a.**

**p :- a.**

* {}

a :- not b.

b :- not a.

p :- a.

Need to delete *not b* and *not a* so we have:

a.

b.

p :- a.

{} not a model

* {a,b}

a :- not b. ⇐ delete this rule because there is *not b* and *b* is in our model

b :- not a. ⇐ delete this rule because there is *not a* and *a* is in our model

p :- a.

We have

p :- a. ⇐ *a* is true but *p* is not true, does not satisfy the program

* {b}

a :- not b. ⇐ delete this rule because there is *not b* but *b* is in our model

b :- not a. ⇐ delete *not a* by definition

p :- a.

We have

b.

p :- a.

*{b}* is an answer set because *b* is satisfied and *a* is not true

* {a}

a :- not b. ⇐ delete the negation as failure *not b*

b :- not a. ⇐ delete this rule because there is *not a* and *a* is in our model

p :- a.

We have

a.

p :- a.

The first clause is satisfied but the second one is not. Not an answer set

* {a,p}

a :- not b. ⇐ delete the negation as failure *not b*

b :- not a. ⇐ delete this rule because there is *not a* and *a* is in our model

p :- a.

We have

a.

p :- a.

{a,p} It is an answer set

**p :- not p, d.**

**q v r :- not d.**

**d :- not r.**

**-q.**

Observation:

1. *-q* must always satisfied
2. *p* should never be contained by any interpretation, because as soon as p is in the interpretation you get rid of the rule *p :- not p, d.* But *p* does not appear anywhere else in the model
3. Can not add *q* or there would be a contradiction

* {-q}

p :- not p, d. ⇐ delete *not p* by definition

q v r :- not d. ⇐ delete *not d* by definition

d :- not r. ⇐ delete *not r* by definition

-q.

We have

p :- d.

q v r :-

d.

-q.

Not satisfied *d* is not in our model and we don’t have either *q* or *r*

* {-q, d}

p :- not p, d. ⇐ delete *not p* by definition

q v r :- not d. ⇐ delete this entire rule (*d* is in the model)

d :- not r. ⇐ delete *not r* by definition

-q.

We have

p :- d.

d.

-q.

We have *-q* and *d* but *p* is missing in our model, it is not an answer set

* {-q, d, p}

p :- not p, d. ⇐ delete this entire rule (*p* is in the model)

q v r :- not d. ⇐ delete this entire rule (*d* is in the model)

d :- not r. ⇐ delete *not r* by definition

-q.

We have

d.

-q.

But this is not a model because it is not minimal

* {-q, r}

p :- not p, d. ⇐ remove *not p*

q v r :- not d. ⇐ remove *not d*

d :- not r. ⇐ remove the entire rule because it contains *not r* but we have r in the model

-q.

We have

p :- d.

q v r.

-q.

It is a model. We have the choice of q or r

**p(X, Y) :- q(X, Y), not t(X, Y).**

**t(X, Y) v q(X, Y) :- r(X), s(Y).**

**r(a).**

**s(b).**

First compute the grounding, substitute any variables with all the possible combinations

p(a,a) :- q(a,a), not t(a,a).

p(a,b) :- q(a,b), not t(a,b).

p(b,a) :- q(b,a), not t(b,a).

p(b,b) :- q(b,b), not t(b,b).

t(a,a) v q(a,a) :- r(a), s(a).

t(a,b) v q(a,b) :- r(a), s(b).

t(b,a) v q(b,a) :- r(b), s(a).

t(b,b) v q(b,b) :- r(b), s(b)

r(a).

s(b).

Start with {r(a). s(b)} but then you have this rule. And you need to make a choice

* {r(a). s(b). t(a,b).}

p(a,a) :- q(a,a), not t(a,a). ⇐ remove the negation as failure literal

p(a,b) :- q(a,b), not t(a,b). ⇐ remove entirely this rule

p(b,a) :- q(b,a), not t(b,a). ⇐ remove the negation as failure literal

p(b,b) :- q(b,b), not t(b,b). ⇐ remove the negation as failure literal

t(a,a) v q(a,a) :- r(a), s(a).

t(a,b) v q(a,b) :- r(a), s(b).

t(b,a) v q(b,a) :- r(b), s(a).

t(b,b) v q(b,b) :- r(b), s(b)

r(a).

s(b).

We get

p(a,a) :- q(a,a). ⇐ ok, the body is false

p(b,a) :- q(b,a). ⇐ ok, the body is false

p(b,b) :- q(b,b). ⇐ ok, the body is false

t(a,a) v q(a,a) :- r(a), s(a). ⇐ ok, the body is false

t(a,b) v q(a,b) :- r(a), s(b). ⇐ the body is true and we have *t(a,b)*

t(b,a) v q(b,a) :- r(b), s(a). ⇐ ok, the body is false

t(b,b) v q(b,b) :- r(b), s(b) ⇐ ok, the body is false

r(a).

s(b).

This is an answer set

* {r(a). s(b). q(a,b).}

p(a,a) :- q(a,a), not t(a,a).

p(a,b) :- q(a,b), not t(a,b). ⇐ this rule is true, but we don’t have *p(a,b)*

p(b,a) :- q(b,a), not t(b,a).

p(b,b) :- q(b,b), not t(b,b).

t(a,a) v q(a,a) :- r(a), s(a).

t(a,b) v q(a,b) :- r(a), s(b).

t(b,a) v q(b,a) :- r(b), s(a).

t(b,b) v q(b,b) :- r(b), s(b)

r(a).

s(b).

Not a model

* {r(a). s(b). q(a,b)., p(a,b)}

p(a,a) :- q(a,a), not t(a,a). ⇐ remove the negation as failure literal

p(a,b) :- q(a,b), not t(a,b). ⇐ remove the negation as failure literal

p(b,a) :- q(b,a), not t(b,a). ⇐ remove the negation as failure literal

p(b,b) :- q(b,b), not t(b,b). ⇐ remove the negation as failure literal

t(a,a) v q(a,a) :- r(a), s(a). ⇐ ok, the body is false

t(a,b) v q(a,b) :- r(a), s(b). ⇐ the body is true, but is ok because we have *q(a,b)*

t(b,a) v q(b,a) :- r(b), s(a). ⇐ ok, the body is false

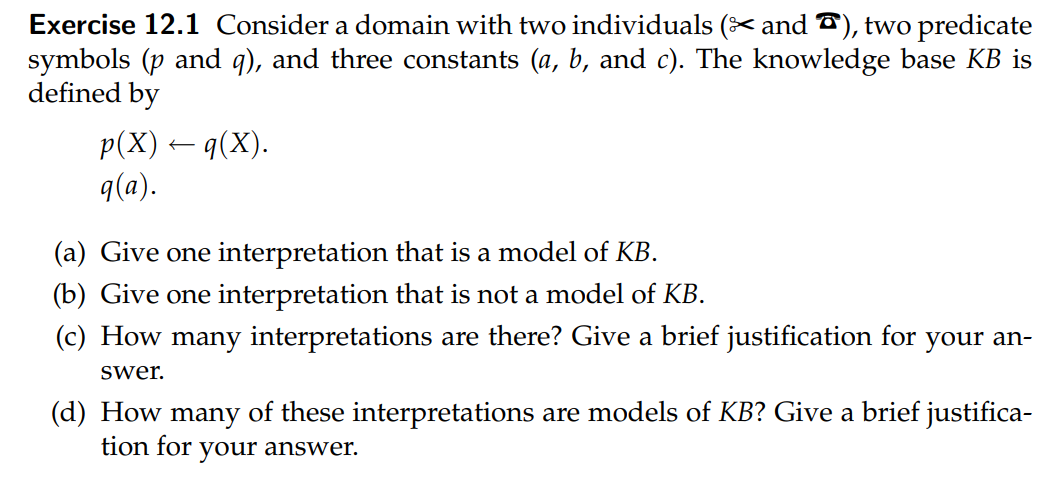
t(b,b) v q(b,b) :- r(b), s(b) ⇐ ok, the body is false

r(a).

s(b).

So this is a model

## 🧮 - Book Exercises



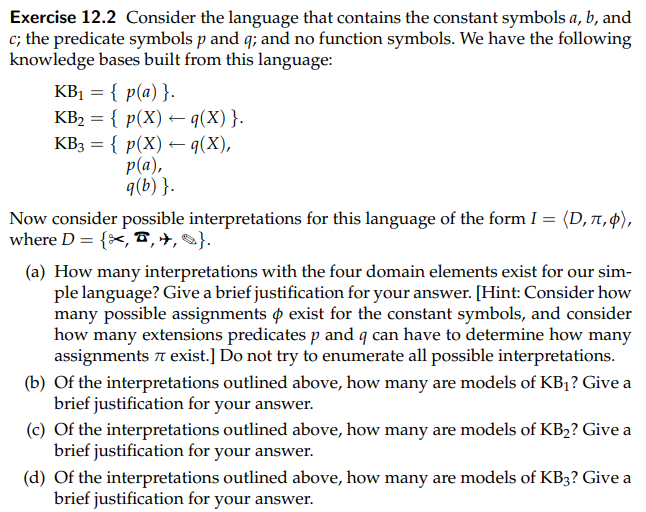
*I* = ⟨D, φ, π⟩

D = {a,b,c}

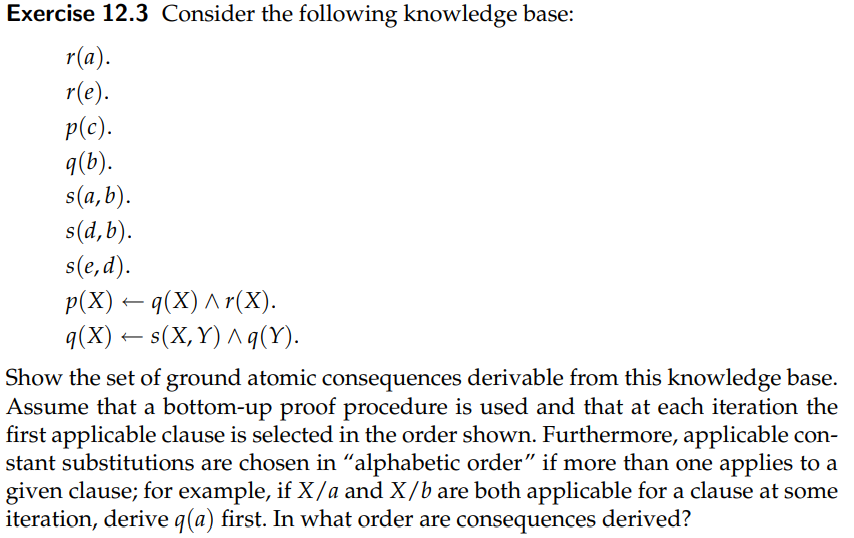
Φ(a) = something

Φ(b) = something else

Φ(c) = another thing



1. There exist 24 possible interpretations. This is because each individual of the domain can have a true or a false assignment with θ



**Ground set**

1. r(a).
2. r(e).
3. p(c).
4. q(b).
5. s(a,b).
6. s(d,b).
7. s(e,d).
8. p(a) ← q(a) ^ r(a).
9. p(b) ← q(b) ^ r(b).
10. p(c) ← q(c) ^ r(c).
11. p(d) ← q(d) ^ r(d).
12. p(e) ← q(e) ^ r(e).
13. q(a) ← s(a,a) ^ q(a).
14. q(a) ← s(a,b) ^ q(b).
15. q(a) ← s(a,c) ^ q(c).
16. q(a) ← s(a,d) ^ q(d).
17. q(a) ← s(a,e) ^ q(e).
18. q(b) ← s(b,a) ^ q(a).
19. q(b) ← s(b,b) ^ q(b).
20. q(b) ← s(b,c) ^ q(c).
21. q(b) ← s(b,d) ^ q(d).
22. q(b) ← s(b,e) ^ q(e).
23. q(c) ← s(c,a) ^ q(a).
24. q(c) ← s(c,b) ^ q(b).
25. q(c) ← s(c,c) ^ q(c).
26. q(c) ← s(c,d) ^ q(d).
27. q(c) ← s(c,e) ^ q(e).
28. q(d) ← s(d,a) ^ q(a).
29. q(d) ← s(d,b) ^ q(b).
30. q(d) ← s(d,c) ^ q(c).
31. q(d) ← s(d,d) ^ q(d).
32. q(d) ← s(d,e) ^ q(e).
33. q(e) ← s(e,a) ^ q(a).
34. q(e) ← s(e,b) ^ q(b).
35. q(e) ← s(e,c) ^ q(c).
36. q(e) ← s(e,d) ^ q(d).
37. q(e) ← s(e,e) ^ q(e).

Start with C = {}

C = {r(a). r(e). p(c). q(b). s(a,b). s(d,b). s(e,d)}

C = {r(a). r(e). p(c). q(b). s(a,b). s(d,b). s(e,d), q(a)14. p(a)8. q(d)29. q(e)36. p(e)12}





**Ask *two\_door\_east(R, r107)***

Create the answer set clause *yes(R) :- two\_door\_east(R,r107)*

1. Select *two\_door\_east(R,r107)* and choose *two\_door\_east(E,W) :- imm\_east(E,M), imm\_east(M,W)* and rename the variables such that it becomes *two\_door\_east(E0,W0) :- imm\_east(E0,M0), imm\_east(M0,W0)*. Apply mgu between *two\_door\_east(R,r107)* and *two\_door\_east(E0,W0)*. Θ = {R/E0, W0/r107}

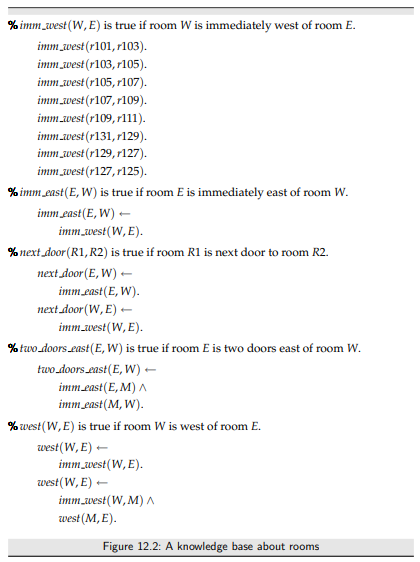
yes(E0) :- *imm\_east(E0,M0), imm\_east(M0, r107).*

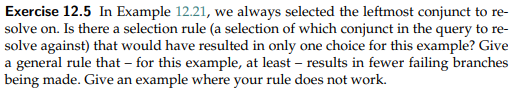
1. Select *imm\_east(E0, M0)* and choose *imm\_east(E, W) :- imm\_west(W,E)* rename the variables such that it becomes *imm\_east(E1, W1) :- imm\_west(W1,E1)*. Then apply the mgu between *imm\_east(E0,M0)* and *imm\_east(E1, W1)*. Θ = {E0/E1, M0/W1}

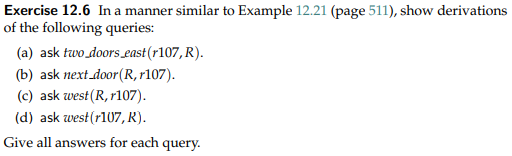
yes(E1) :- *imm\_west(W1, E1), imm\_east(W1, r107).*

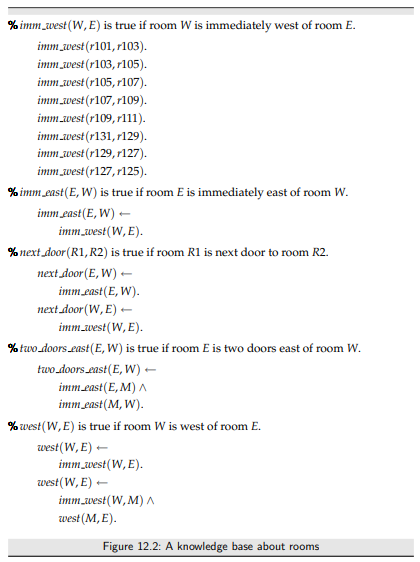
1. Select *imm\_east(W1, r107)* and choose *imm\_east(E,W) :- imm\_west(W,E)* and rename the variables such that it becomes *imm\_east(E2,W2) :- imm\_west(W2, E2)*. Then apply the mgu between *imm\_east(W1, r107)* and *imm\_east(E2,W2)*. Θ = {W1/E2, W2/r107}

yes(E1) :- *imm\_west(E2, E1), imm\_west(r107, E2)*









**Ask two\_doors\_east(r107, R).**

Create the answer set clause

*yes(R) :- two\_doors\_east(r107, R).*

1. Select *two\_doors\_east(r107, R)* and choose the clause *two\_doors\_east(E,W) :- imm\_east(E,M) ^ imm\_east(M,W)* and rename the variables such that it becomes *two\_doors\_east(E0,W0) :- imm\_east(E0,M0) ^ imm\_east(M0,W0)* and apply the mgu with *two\_doors\_east(r107, R)*. Θ = {E0/r107, R/W0)

*yes(W0) :- imm\_east(r107, M), imm\_east(M, W0)*

1. Select *imm\_east(r107, M)* and choose *imm\_east(E,W) :- imm\_west(W,E)* and rename the variables such that it becomes *imm\_east(E1,W1) :- imm\_west(W1,E1)*. Apply the mgu between *imm\_east(r107, M)* and *imm\_east(E1,W1)*. Θ = {E1/r107, M/W1}

*yes(W0) :- imm\_west(W1, r107), imm\_east(W1, W0)*

1. Select *imm\_east(W1, W0)* and choose *imm\_east(E,W) :- imm\_west(W,E)* and rename the variables such that it becomes *imm\_east(E2,W2) :- imm\_west(W2,E2)*. Then apply the mgu between *imm\_east(W1, W0)* and *imm\_east(E2,W2) :- imm\_west(W2,E2)*. Θ = {W1/E2, W0/W2}

*yes(W2) :- imm\_west(E2, r107), imm\_west(W2, E2)*

1. Select *imm\_west(E2, r107)* and choose *imm\_west(r105, r107)* and apply the mgu. Θ = {E2/r105}

*yes(W2) :- imm\_west(r105, r107), imm\_west(W2, r105)*

Since *imm\_west(r105, r107)* is satisfied, we can remove it

*yes(W2) :- imm\_west(W2, r105)*

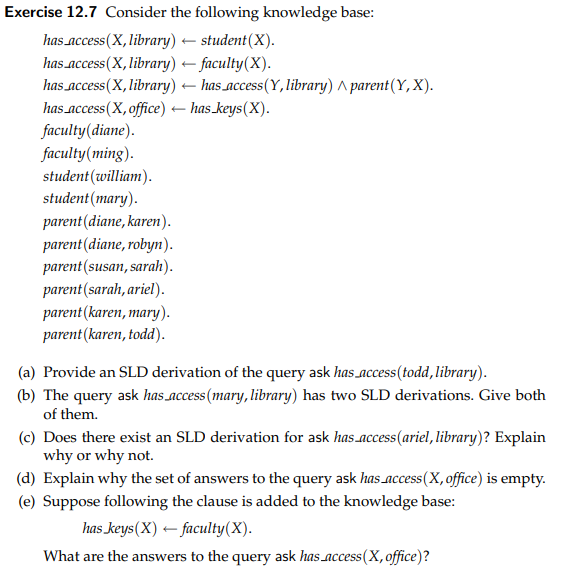
1. Select *imm\_west(W2, r105)* and choose *imm\_west(r103, r105)* and apply the mgu. Θ = {W2/r103}

*yes(r103) :- imm\_west(r103, r105)*

Since *imm\_west*(r103, r105) is satisfied, we can remove it

*yes(r103) :-*

So r103 is a possible substitution of R that satisfies *Ask two\_doors\_east(r107, R).*



**SLD** **derivation of** *has\_access(todd, library)*

a) start with C ={} and then start propagate

C = {faculty(diane), faculty(ming), student(william), student(mary), parent(diane, karen), parent(diane, robyn), parent(susan, sarah), parent(sarah, ariel), parent(karen,mary), parent(karen, todd) }

Using *has\_access(X, library) :- student(X)*. We can try to substitute X with the constants *william* and *mary*, we get:

C = {faculty(diane), faculty(ming), student(william), student(mary), parent(diane, karen), parent(diane, robyn), parent(susan, sarah), parent(sarah, ariel), parent(karen,mary), parent(karen, todd), has\_acess(william, library), has\_access(mary, library)}

Using *has\_access(X, library) :- faculty(X)*. We can try to substitute X with the constants *diane* and *ming*, we get:

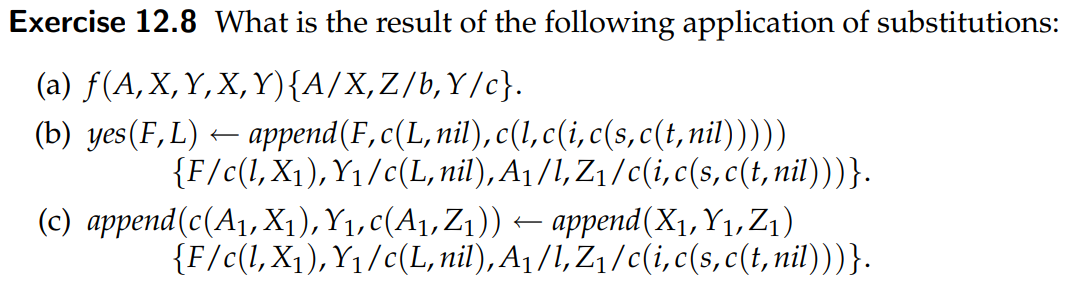
C = {faculty(diane), faculty(ming), student(william), student(mary), parent(diane, karen), parent(diane, robyn), parent(susan, sarah), parent(sarah, ariel), parent(karen,mary), parent(karen, todd), has\_acess(william, library), has\_access(mary, library), has\_access(diane, library), has\_access(ming, library)}

Using *has\_access(X, library) :- has\_access(Y, library), parent(X,Y)* we can substitute X with *karen* and Y with *diane,* we get

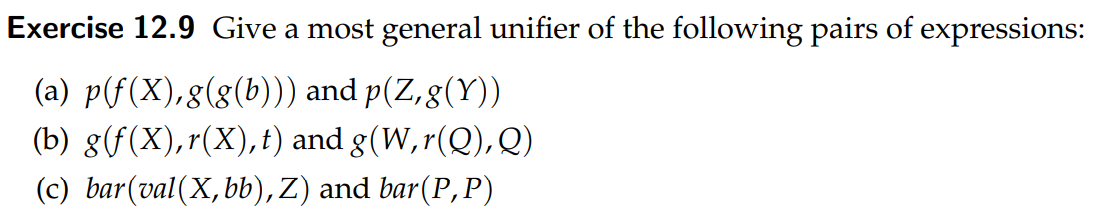
C = {faculty(diane), faculty(ming), student(william), student(mary), parent(diane, karen), parent(diane, robyn), parent(susan, sarah), parent(sarah, ariel), parent(karen,mary), parent(karen, todd), has\_acess(william, library), has\_access(mary, library), has\_access(diane, library), has\_access(ming, library), has\_access(karen, library)}

Using *has\_access(X, library) :- has\_access(Y, library), parent(X,Y)* we can substitute X with *todd* and Y with *karen,* we get

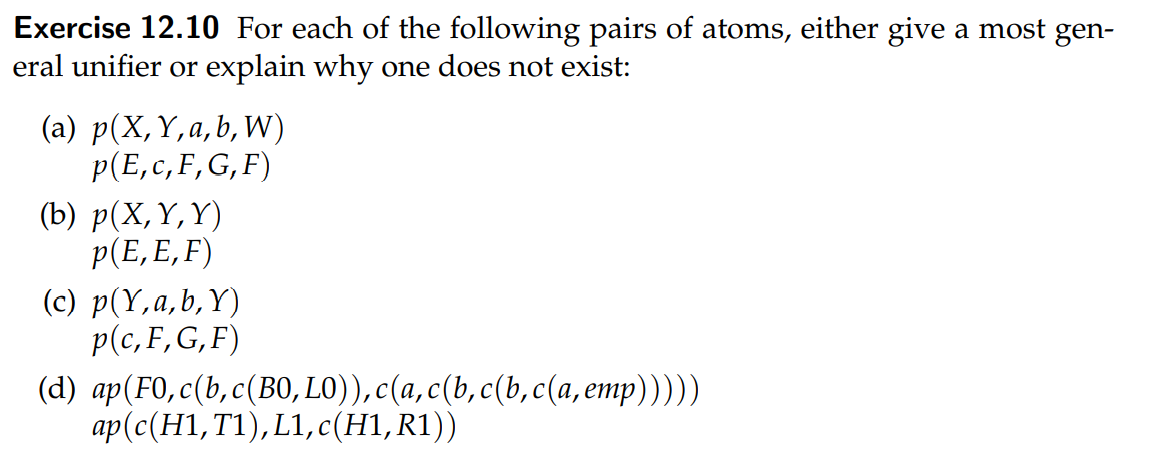
C = {faculty(diane), faculty(ming), student(william), student(mary), parent(diane, karen), parent(diane, robyn), parent(susan, sarah), parent(sarah, ariel), parent(karen,mary), parent(karen, todd), has\_acess(william, library), has\_access(mary, library), has\_access(diane, library), has\_access(ming, library), has\_access(karen, library), has\_access(todd, library)}



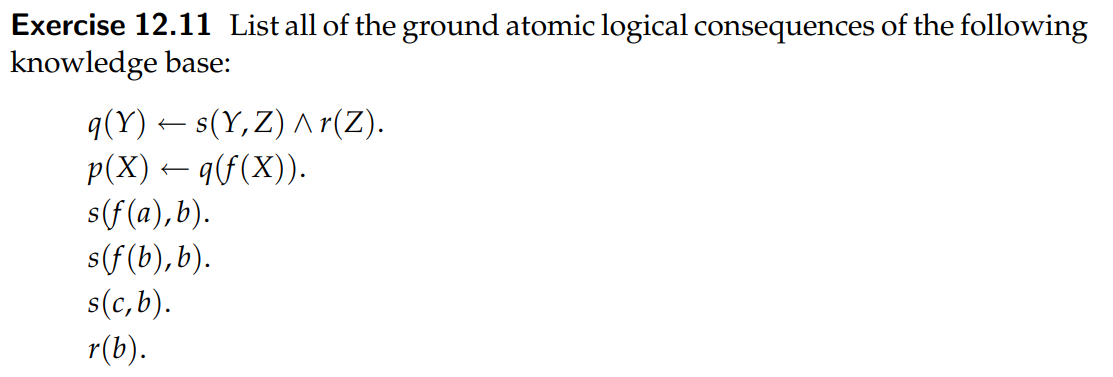
1. f(X, X, c, X, c)
2. yes(c(l,X1), L) :- append(c(l,X1), c(L,nil), c,(l,c(i,c(s,c(t,nil)))))
3. append(c(l, X1), c(L,nil), c(l, c(i,c(s,c(t,nil)))))) :- append(X1, c(L,nil), c(i,c(s,c(t,nil)))))

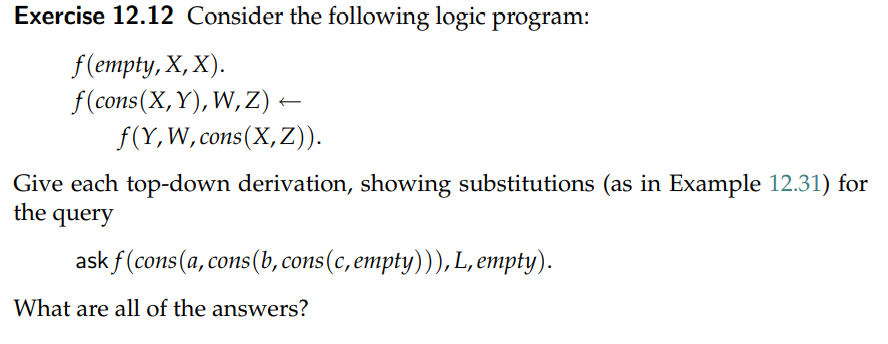


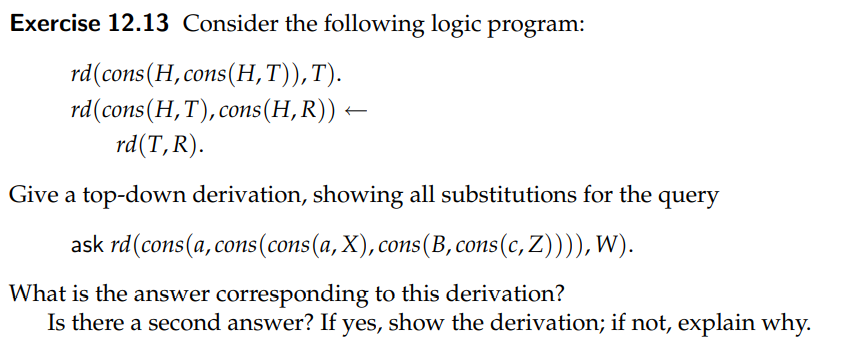
1. { f(X)/Z, g(b)/Y}
2. {f(X)/W, r(X)/r(Q), Q/t}
3. {val(X,bb)/P, Z/P}



1. {X/E, Y/c, F/a, G/b, W/a[[4]](#footnote-3)}
2. {X/E, Y/Z, Y/Z}
3. NO. Because we would have Y/c, F/a and Y/F
4. WTF

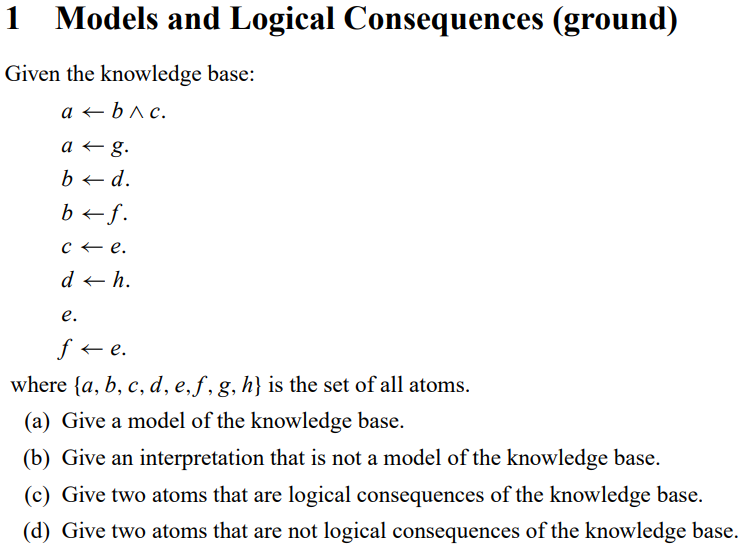




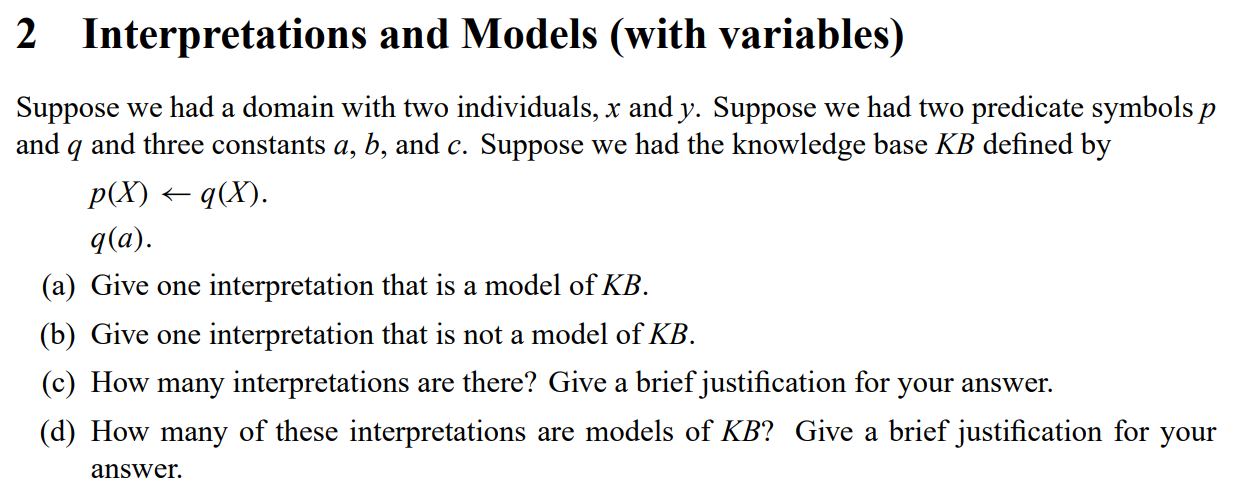


## 📁 - .PDF exercise

Link file: <https://www.cs.ubc.ca/~poole/ci/problems/ch2/probs_sols.pdf>



1. A model is when all the atoms are true, another model is when *e,f,c,b,a* are true (is the minimal model)
2. There are a lot of possibilities. An interpretation where *a* is false but *b* and *c* are true is not a model of the knowledge base (the others atoms can be all true or all false, does not matters because already a:- b,c is not a model)
3. *e* and *c* are a logical consequence of the knowledge base (also *c* and *f* are). Can be two of *e,f,c,b,a*
4. *d* and *h* are not a logical consequence of the knowledge base (also *g*)



1. D = {x,y}

θ(a) = x, θ(b) = x, θ(c) = x

π(p)(x) = TRUE

π(q)(x) = TRUE

π(p)(y) = TRUE

π(q)(y) = TRUE

1. D = {x,y}

θ(a) = x, θ(b) = x, θ(c) = x

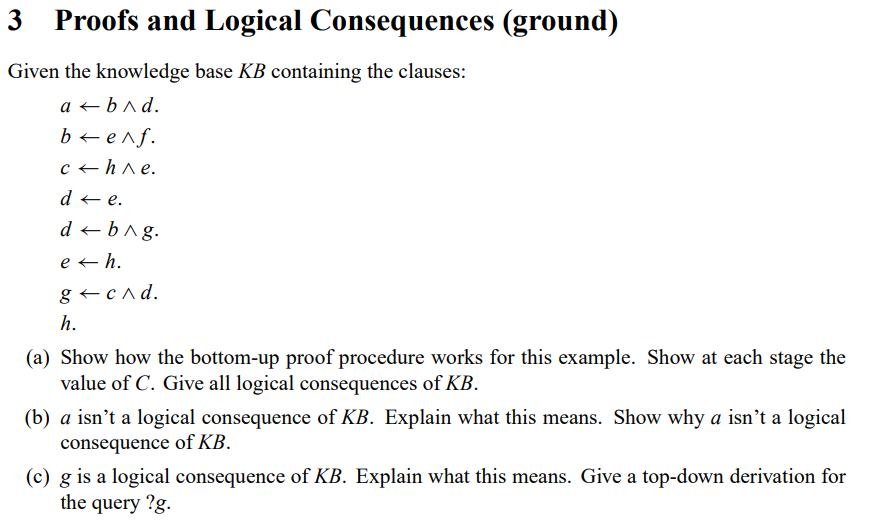
π(p)(x) = TRUE

π(q)(x) = FALSE

π(p)(y) = FALSE

π(q)(y) = TRUE

1. There are 23 possible θ (2 interpretations for *a*, 2 for *b* and 2 for *c*).



1. Can not select any other atom because there is only one [↑](#footnote-ref-0)
2. That is *strongly\_connected(X0,c)* [↑](#footnote-ref-1)
3. That is connected(X1,c) [↑](#footnote-ref-2)
4. Because W/F and F/A [↑](#footnote-ref-3)