Facts, Rules, and Queries

TK2ICM: Logic Programming (2nd Term 2018-2019)

M. Arzaki

Computing Laboratory, School of Computing
Telkom University

SoC Tel-U

January 2019

Acknowledgements

This slide is compiled using the materials in the following sources:
Books:

1. P. Blackburn, J. Bos, K. Striegnitz, *Learn Prolog Now!* (Chapter 1-6, 10,11), London: College Publications (available online), 2006. **[LPN]**
2. M. Bramer, *Logic Programming with Prolog* (Chapter 1-9), 2nd Edition, Springer, 2013. **[LPwP]**
3. I. Bratko, *Prolog Programming for Artificial Intelligence* (Chapter 1-3, 5,6,8,9), Pearson Education, 2001. (advanced reference). **[PPAI]**
4. K. H. Rosen, *Discrete Mathematics and Its Applications* (Chapter1), 7th Edition, 2012.
5. M. Ben-Ari, *Mathematical Logic for Computer Science* (Logic Programming Sections), 2nd Edition, 2000.

Lecture slides and lecture notes:

1. *Prolog Programming* by Kristina Striegnitz.
2. *Learn Prolog Now!* by Patrick Blackburn, Johan Bos, and Kristina Striegnitz.
3. *Logic Programming* at Fasilkom UI by A. A. Krisnadhi and A. Saptawijaya.
4. *Computational Logic Part 2: Logic Programming* at Fasilkom UI by L. Y. Stefanus.
5. *Logic Programming* at ILLC, University of Amsterdam by U. Endriss.
6. *Functional Programming* at Fasilkom UI by A. Azurat.
7. *Bahasa Prolog* at FPMIPA UPI by Munir.
8. Other available sources online.

Some of the pictures are taken from the above resources. This slide is intended for academic purpose at FIF Telkom University. You are prohibited for using these slides for professional purpose outside Telkom University, unless with the author authorization. If you have any suggestions/comments/questions related with the material on this slide, send an email to `<pleasedontspam>@telkomuniversity.ac.id`.

Contents

Contents

Contents

Knowledge Base 1 (KB1)

Knowledge base 1 (KB1) is simply a collection of facts. Facts are used to state things that are unconditionally true of the domain of interest. For example, we can state that Mia, Jody, and Yolanda are women, and that Jody plays air guitar, using the following four facts:

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
```

We can ask Prolog whether Mia is a woman by posing the query:

Knowledge Base 1 (KB1)

Knowledge base 1 (KB1) is simply a collection of facts. Facts are used to state things that are unconditionally true of the domain of interest. For example, we can state that Mia, Jody, and Yolanda are women, and that Jody plays air guitar, using the following four facts:

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
```

We can ask Prolog whether Mia is a woman by posing the query:
`?- woman(mia).`
Prolog will answer:

Knowledge Base 1 (KB1)

Knowledge base 1 (KB1) is simply a collection of facts. Facts are used to state things that are unconditionally true of the domain of interest. For example, we can state that Mia, Jody, and Yolanda are women, and that Jody plays air guitar, using the following four facts:

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
```

We can ask Prolog whether Mia is a woman by posing the query:
`?- woman(mia).`
Prolog will answer:
`true.`

Similarly, we can ask whether Jody plays air guitar by posing the following query:
`?- playsAirGuitar(jody).`

Similarly, we can ask whether Jody plays air guitar by posing the following query:
`?- playsAirGuitar(jody).`
Prolog will again answer `true`, because this is one of the facts in KB1.

However, suppose we ask whether Mia plays air guitar:
`?- playsAirGuitar(mia).`

Similarly, we can ask whether Jody plays air guitar by posing the following query:
`?- playsAirGuitar(jody).`
Prolog will again answer `true`, because this is one of the facts in KB1.

However, suppose we ask whether Mia plays air guitar:
`?- playsAirGuitar(mia).`
We'll get false as the answer because Prolog concludes that `playsAirGuitar(mia)` does not follow from KB1.

Exercise

What happens if we pose the following query?

1. `?- playsAirGuitar(vincent).`
2. `?- tatooed(jody).`

Contents

Knowledge Base 2 (KB2)

```
listensToMusic(mia).
happy(yolanda).
playsAirGuitar(mia) :- listensToMusic(mia).
playsAirGuitar(yolanda) :- listensToMusic(yolanda).
listensToMusic(yolanda):- happy(yolanda).
```

- KB2 contains two facts, `listensToMusic(mia)` and `happy(yolanda)`. The last three items are rules.
- Rules state information that is conditionally true of the domain of interest. For example, the first rule says that Mia plays air guitar if she listens to music, and the last rule says that Yolanda listens to music if she if happy.

Remark

If a knowledge base contains a rule `head :- body`, and Prolog knows that body follows from the information in the knowledge base, then Prolog can infer head.

Exercise

What happens if we pose the following query?

1. `?- playsAirGuitar(mia).`
2. `?- playsAirGuitar(yolanda).`

KB2 contains five clauses, namely three rules and two facts. KB2 consists of three predicates (or procedures): `listensToMusic`, `happy`, and `playsAirGuitar`.

Contents

Knowledge Base 3 (KB3)

```
happy(vincent).
listensTomusic(butch).
playsAirGuitar(vincent):- listensTomusic(vincent),happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listensTomusic(butch).
```

The comma (,) express conjunction (an $\wedge$ operator in logic).

Exercise

What happens if we pose the following query?

1. `?- playsAirGuitar(vincent).`
2. `?- playsAirGuitar(butch).`

Nota that KB3 two rules with exactly the same head, namely:

- `playsAirGuitar(butch):- happy(butch).`
- `playsAirGuitar(butch):- listensToMusic(butch).`

This is a way of stating that Butch plays air guitar if either he listens to music, or if he is happy.

There is another way of expressing disjunction in Prolog. We could replace the pair of rules given above by the single rule

- `playsAirGuitar(butch):-happy(butch);listensToMusic(butch).`

KB3 becomes:

```
happy(vincent).
listensTomusic(butch).
playsAirGuitar(vincent):- listensTomusic(vincent),happy(vincent).
playsAirGuitar(butch):- happy(butch);listensTomusic(butch).
```

Contents

Knowledge Base 4 (KB4)

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent,mia).
loves(marcellus,mia).
loves(pumpkin,honey_bunny).
loves(honey_bunny,pumpkin).
```

Exercise

What happens if we pose the following query?

1. `?- woman(X).`
2. `?- loves(marcellus,X),woman(X).`
3. `?- loves(pumpkin,X), woman(X).`

Contents

Knowledge Base 5 (KB5)

```
loves(vincent,mia).
loves(marsellus,mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

jealous(X,Y):- loves(X,Z), loves(Y,Z).
```

Exercise

1. What happens if we pose the query `?- jealous(marcellus,W).`
2. Suppose we wanted Prolog to tell us about all the jealous people: what query should we pose?

Contents

Knowledge Base 6 (KB6)

```
wizard(harry).
wizard(ron).
wizard(hermione).

muggle(uncle_vernon).
muggle(aunt_petunia).

chases(crookshanks, scabbars).
```

Exercise

What happens if we pose the query:

1. `?- wizard(harry).`
2. `?- chases(crookshanks,scabbars).`
3. `?- muggle(harry).`
4. `?- wizard(dumbledore).`
5. `?- witch(hermione).`

Exercise

What happens if we pose the query:

1. `?- muggle(X).`
2. `?- chases(X,Y).`
3. `?- chases(X,X).`

Contents

Knowledge Base (KB7)

```
eating(dudley).

happy(aunt_petunia) :- happy(dudley).
happy(uncle_vernon) :- happy(dudley), unhappy(harry).
happy(dudley) :- kicking(dudley,harry).
happy(dudley) :- eating(dudley).
```

Remark

From KB6, we observe several correspondence of Prolog with logical operators.

Knowledge Base (KB7)

```
eating(dudley).

happy(aunt_petunia) :- happy(dudley).
happy(uncle_vernon) :- happy(dudley), unhappy(harry).
happy(dudley) :- kicking(dudley,harry).
happy(dudley) :- eating(dudley).
```

Remark

From KB6, we observe several correspondence of Prolog with logical operators.

1. $\rightarrow$ operator, e.g., if `happy(dudley)` is true, then `happy(aunt_petunia)` is true. The `:-` corresponds to $\leftarrow$ operator.

Knowledge Base (KB7)

```
eating(dudley).

happy(aunt_petunia) :- happy(dudley).
happy(uncle_vernon) :- happy(dudley), unhappy(harry).
happy(dudley) :- kicking(dudley,harry).
happy(dudley) :- eating(dudley).
```

Remark

From KB6, we observe several correspondence of Prolog with logical operators.

1. $\rightarrow$ operator, e.g., if `happy(dudley)` is true, then `happy(aunt_petunia)` is true. The `:-` corresponds to $\leftarrow$ operator.
2. $\wedge$ operator, e.g., if `happy(dudley)` is true and `unhappy(harry)` is true, then `happy(uncle_vernon)` is true. The `,` corresponds to $\wedge$ operator.

Knowledge Base (KB7)

```
eating(dudley).

happy(aunt_petunia) :- happy(dudley).
happy(uncle_vernon) :- happy(dudley), unhappy(harry).
happy(dudley) :- kicking(dudley,harry).
happy(dudley) :- eating(dudley).
```

Remark

From KB6, we observe several correspondence of Prolog with logical operators.

1. $\rightarrow$ operator, e.g., if `happy(dudley)` is true, then `happy(aunt_petunia)` is true. The `:-` corresponds to $\leftarrow$ operator.
2. $\wedge$ operator, e.g., if `happy(dudley)` is true and `unhappy(harry)` is true, then `happy(uncle_vernon)` is true. The `,` corresponds to $\wedge$ operator.
3. $\vee$ operator, e.g., if `kicking(dudley,harry)` is true or `eating(dudley)` is true, then `happy(dudley)` is true. The last two rules can be expressed as a single rule `happy(dudley) :- kicking(dudley,harry); eating(dudley).`

Exercise

What happens if we pose the following query:

1. `?- happy(dudley).`
2. `?- happy(aunt_petunia).`
3. `?- happy(uncle_vernon).`
4. `?- happy(X).`

Contents

Knowledge Base (KB8)

```
father(albert,james).
father(james,harry).

mother(ruth,james).
mother(lili,harry).

wizard(lili).
wizard(ruth).
wizard(albert).

wizard(X):-
                father(Y,X),wizard(Y),
                mother(Z,X),wizard(Z).
```

KB8 defines three predicates, `father/2`, `mother/2`, and `wizard/1`.

Remark

For any predicate `something`, the notation `something/n` where `n` is an integer denotes that `something` has arity `n`. Like in predicate that we've seen in Mathematical Logic, the arity is the number of inputs in a predicate.

Exercise

What happens if we pose the following query:

1. `?- wizard(james).`
2. `?- wizard(harry).`
3. `?- wizard(Wizard).`

Exercise

What query should we pose if we want to find a wizard whose mother is also a wizard?

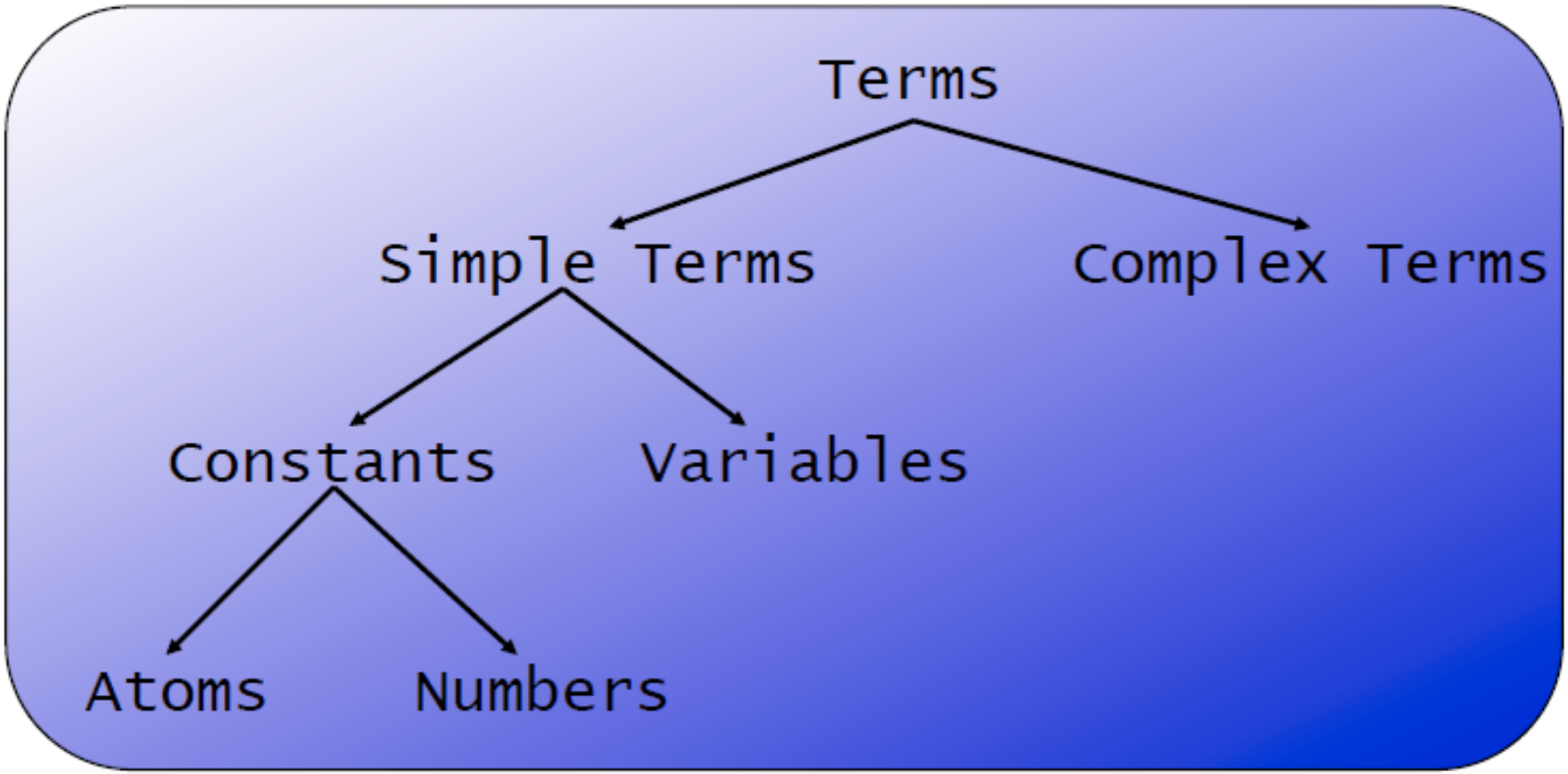
Contents

Terms

What are exactly facts, rules, and queries built out of?

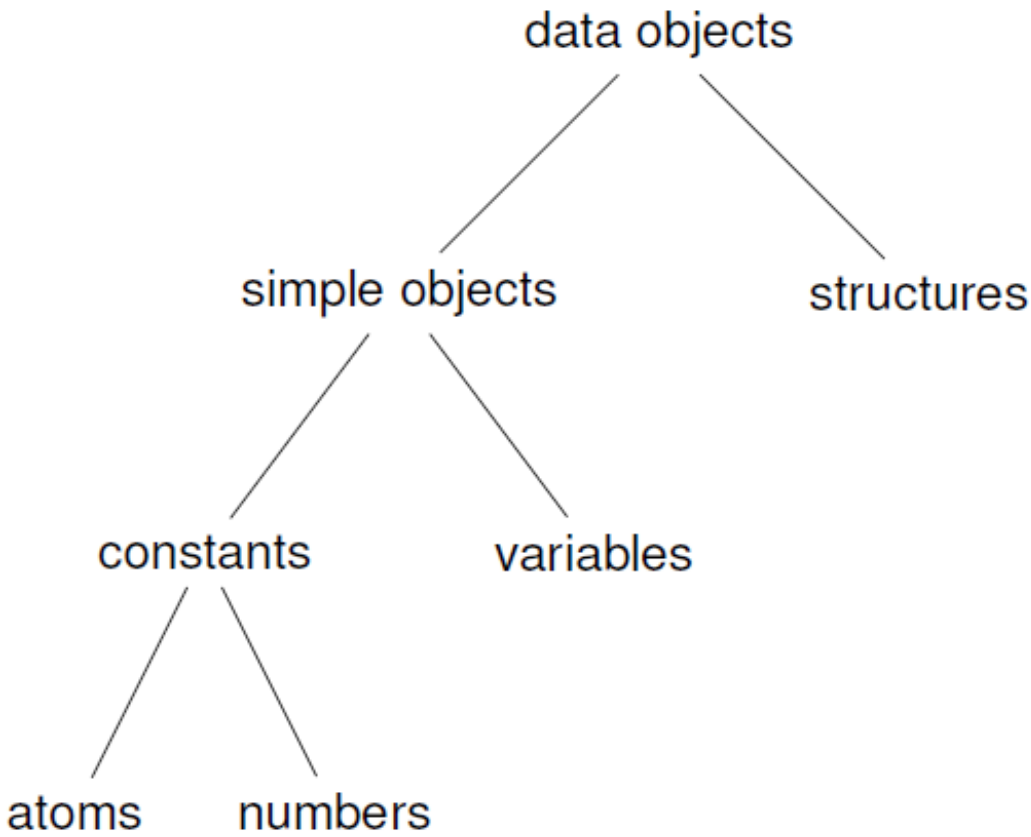
Terms

What are exactly facts, rules, and queries built out of? They built of Prolog terms.



Data Objects

Terms are sometimes called as *data objects*, and complex terms are also called as *structures*.



Atoms

- A string of characters made up of uppercase letters, lowercase letters, digits,and the underscore character, that **begins with a lowercase letter**.

Atoms

- A string of characters made up of uppercase letters, lowercase letters, digits,and the underscore character, that **begins with a lowercase letter**. e.g.: `butch`, `playGuitar`, `wizard`, `bandung`, `informatika`
- An arbitrary sequence of character **enclosed in single quotes.**

Atoms

- A string of characters made up of uppercase letters, lowercase letters, digits,and the underscore character, that **begins with a lowercase letter**.
  e.g.: `butch`, `playGuitar`, `wizard`, `bandung`, `informatika`
- An arbitrary sequence of character **enclosed in single quotes.**
  e.g.: `'Vincent'`, `'Five dollar shake'`, `'@$%'`, `'kuliah TK2 ICM'`
- Strings of special characters:

Atoms

- A string of characters made up of uppercase letters, lowercase letters, digits,and the underscore character, that **begins with a lowercase letter**.
  e.g.: `butch`, `playGuitar`, `wizard`, `bandung`, `informatika`
- An arbitrary sequence of character **enclosed in single quotes.**
  e.g.: `'Vincent'`, `'Five dollar shake'`, `'@$%'`, `'kuliah TK2 ICM'`
- Strings of special characters:
  e.g.: `<- - - - >`, `=>`, `<=`, `======>`, `<======`, `:`, `;`, `,`

Numbers

- Integers, e.g.: `1`, `31`, `-90`, `23`, the range of integers in Prolog depends on its implementation.
- Real numbers (represented as floats), e.g.: `2.5`, `e`, `pi`, `sqrt(2)`, the range of real numbers in Prolog depends on its implementation.
- Real numbers are not used very much in typical Prolog programming because Prolog is primarily a language for symbolic, non-numerical computation.

Some Built-in Predicates

- `atom(X)`
  returns `true` if `X` currently stands for an atom.
- `integer(X)`
  returns `true` if `X` currently stands for an integer.
- `float(X)`
  returns `true` if `X` currently stands for a real number which is not an integer.
- `number(X)`
  returns `true` if `X` currently stands for a number.
- `atomic(X)`
  returns true if `X` currently stands for an atom or a number.

Exercise

Try to pose the following queries:

1. `?- atom(logic_programming).`
2. `?- integer(1998).`
3. `?- float(1998).`
4. `?- integer(19.98).`
5. `?- float(19.98).`
6. `?- float(sqrt(1998)).`
7. `?- atomic(sqrt(1998)).`
8. `?- atomic(computing).`
9. `?- atomic(informatika).`
10. `?- atomic(<==>).`

Variables

A variable is a string of uppercase letters, lowercase letters, digits and underscore characters that **starts either with an uppercase letter or with underscore**.

Examples

`X`, `Y`, `Variable`, `_tag`, `X_526`, `List`, `Head`, `Tail`, `Input`, `Output`, `_input`.

The variable _ (that is, a single underscore character) is rather special. It's called the anonymous variable, and we discuss it in a later lecture.

Singleton Variables

Definition

A singleton variable is a variable that appears only one time in a clause. It can always be replaced by _, i.e., the anonymous variable.

Suppose we have a predicate `parent(X,Y)`, we can define the predicate `hasachild(X)` from the predicate `parent(X,Y)` as follows:

- `hasachild(X):- parent(X,_).` or
- `hasachild(X):- parent(X,_Y).`

In the above definition, we do not have any interest about the child of `X`. In predicate logic, the above rules is equivalent to
$\forall X\,(\exists Y\ parent\,(X,Y) \rightarrow hasachild\,(X))$.

Variables Checking

We use the predicate `var/1` to check whether a simple term (or a simple object) is a variable or not. Similarly, we use `nonvar/1` to check whether a simple term (or a simple object) is not a variable or not.

- `var(X)` succeeds if `X` is currently an uninstantiated variable.
- `nonvar(X)` succeeds if `X` is not a variable, or `X` is already uninstantiated variable.

Exercise

Try to test the following queries:

1. `?- var(Z).`
2. `?- var(_).`
3. `?- nonvar(bandung).`
4. `?- nonvar(Bandung).`

Complex Terms (Structures)

- Atoms, numbers and variables are building blocks for **complex terms (structures)**.
- Complex terms are built out of a **functor** directly followed by a sequence of **arguments**.
  - Arguments are put in round brackets and separated by commas.
  - The functor must be an atom.

Examples

The following expression are complex terms:

Examples

The following expression are complex terms:

1. `playsAirGuitar(jody)`, `playsAirGuitar` is a functor, jody is an argument.
2. `loves(vincent,mia)`, `loves` is a functor, `vincent` and `mia` are arguments.
3. `jealous(marsellus,W)`, `jealous` is a functor, `marsellus` and `W` are arguments.
4. `date(31,jan,2018)`, date is a functor, 31, `jan`, and 2018 are arguments.
5. `hide(X,father(father(father(butch))))` is a complex term that has complex terms inside of it.
6. `vertical(line(point(X,Y),point(X,Z))` is a complex term that has complex terms inside of it.

Grounds and Nongrounds

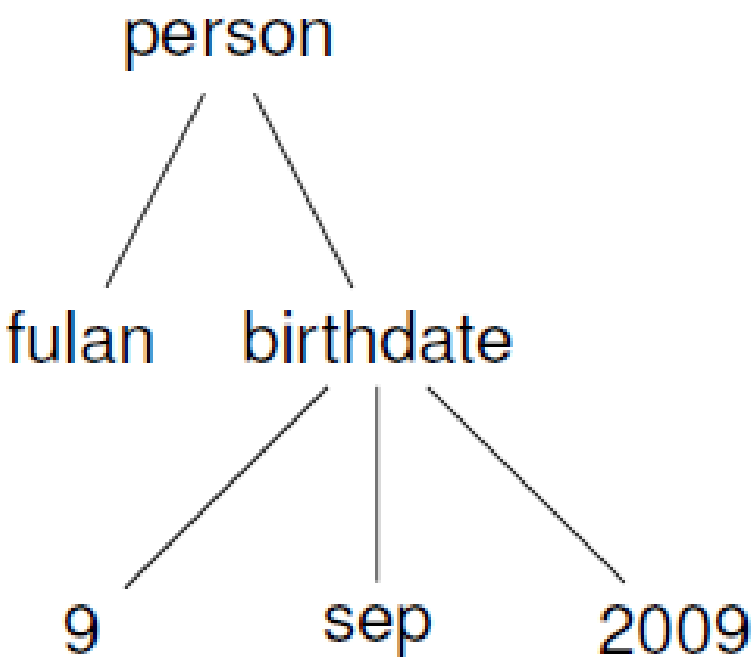
Terms where variables do not occur are called **grounds**; otherwise, they are called **nonground**.

Examples

The term `loves(vincent,mia)` is ground, whereas the term `loves(vincent,X)` is nonground.

Complex Terms as Trees

- All complex terms (structures) can be seen as a tree.
- The root of the tree is the functor, and the offsprings of the root are the components.
  If a component of a complex term is also a complex term, then it is a subtree of the tree that corresponds to the whole complex term.
- The complex term `person(fulan, birthdate(9, sep, 2009))` is represented as the tree:



Functors

Each functor is defined by two things:

- the name, whose syntax is like atoms;
- the arity, i.e., the number of arguments.

The two functors of the following complex terms are different

- `uncle(X)` and `uncle(X,Y)`;
- `point(X,Y)` and `point(X,Y,Z)`.

Arity

The number of arguments a complex term has is called its **arity**.

Examples

1. `woman(mia)` is a term with arity 1.
2. `loves(vincent,mia)` is a term with arity 2.
3. `father(father(butch))` is a term with arity 1.

Remark

We can define two predicates with the same functor but with different arity. However, Prolog would treat this as two different predicates. In Prolog documentation, arity of a predicate is usually indicated with the suffix "/" followed by a number to indicate the arity.

We recall KB8.

```
father(albert,james).
father(james,harry).

mother(ruth,james).
mother(lili,harry).

wizard(lili).
wizard(ruth).
wizard(albert).

wizard(X):-
                father(Y,X),wizard(Y),
                mother(Z,X),wizard(Z).
```

KB8 defines three predicates, `father/2`, `mother/2`, and `wizard/1`.