Declarative versus procedural meaning

There are two ways of understanding a Prolog program.

For example, consider the abstract query

$\mathcal{P}$:- $\mathcal{Q}$, $\mathcal{R}$.

where $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$ are objects.

This clause can be read in two ways:

1. $\mathcal{P}$ is true if $\mathcal{Q}$ and $\mathcal{R}$ are true.
2. To prove $\mathcal{P}$, *first* prove $\mathcal{Q}$ and *next* prove $\mathcal{R}$.

Declarative versus procedural meanings (cont)

Note the difference in the wording and the ordering.

In the first case, we speak about **truth** and the order in which the truth values are obtained is actually not completely defined. For example, in this context, "$\mathcal{Q}$ and $\mathcal{R}$ are true" is equivalent to "$\mathcal{Q}$ and $\mathcal{R}$ are true", because $\mathcal{Q}$ must be true and, separately, $\mathcal{R}$ too.

In the second case, we speak about the **process** of obtaining the truth values in details. In this context, "$\mathcal{P}$ and $\mathcal{Q}$ are true" may **not** be equivalent to "$\mathcal{Q}$ and $\mathcal{P}$ are true" because one may be more efficient (e.g. if $\mathcal{P}$ is false, there is no need to prove $\mathcal{Q}$) or one may not terminate (e.g. if $\mathcal{P} = \mathcal{Q}$).

Declarative meaning

Informally, the declarative meaning of a Prolog program is as follows.

A goal $\mathcal{G}$ is true (or logically follows from the program) if

1. *there is a clause C in the program*
2. *of which an instance I can be deduced such that*
   - 2.1 *the head of I is identical to $\mathcal{G}$,*
   - 2.2 *all the goals of the body of I are true.*

Note that it is not said how to find C and I, and no ordering of the goals is imposed (they just all must be true).

Procedural meaning

Given a list $\mathcal{G}$ of goals, a list $\mathcal{C}$ of clauses and the identity substitution σ,

1. *if $\mathcal{G}$ is empty, then end with σ and* **success**;
2. *if $\mathcal{C}$ is empty then* **fail**; *let G_1 be the first goal and C_1 the first clause;*
3. *let $\overline{C}_1$ be an instance of C_1 containing no variable in common with $\mathcal{G}$;*
4. *if the head of $\overline{C}_1$ does not match the head of G_1, restart with the remaining clauses;*
5. *let σ' be the resulting substitution, $\mathcal{G}'$ the remaining goals and $\mathcal{B}$ the goals in the body of $\overline{C}_1$; restart with the list of goals made of $\sigma'(\mathcal{G}')$ and $\sigma'(\mathcal{B})$ and substitution $\sigma' \circ \sigma$;*
6. *if it failed, restart with $\mathcal{G}$ and the remaining clauses in $\mathcal{C}$* (backtracking).

Procedural meaning/Example

The declarative meaning can be seen as an *abstraction* of the procedural meaning, i.e. it hides certain aspects of it. Consider

```
big(bear).              % Clause 1
big(elephant).          % Clause 2
small(cat).             % Clause 3
brown(bear).            % Clause 4
black(cat).             % Clause 5
gray(elephant).         % Clause 6
dark(Z) :- black(Z).    % Clause 7
dark(Z) :- brown(Z).    % Clause 8

?- dark(X), big(X).     % What is dark and big?
```

Procedural meaning/Example

The list of goals is $\mathcal{G} = (G_1, G_2)$, where $G_1 = \texttt{dark(X)}$ and $G_2 = \texttt{big(X)}$.

There is no match between G_1 and the facts, until clause 7.

Clause 7 has no variable in common with G_1.

The matching between the head of clause 7, `dark(Z)`, and G_1 succeeds with substitution $\sigma' = \{\texttt{X} = \alpha, \texttt{Z} = \alpha\}$.

Let us start again with the list of goals $(\texttt{black}(\alpha), \texttt{big}(\alpha))$, and the substitution σ'.

Actually, it is enough to restrict σ' to the variables in G_1, so let us take instead

$$\sigma' = \{\texttt{X} = \alpha\}$$

Procedural meaning/Example (cont)

Now the list of goals is $\mathcal{G} = (G_1, G_2)$, where $G_1 = \texttt{black}(\alpha)$ and $G_2 = \texttt{big}(\alpha)$, and $\sigma = \{\texttt{X = } \alpha\}$.

The first clause whose head matches $\texttt{black}(\alpha)$ is clause 5, which contains no variable. The substitution resulting of the matching is $\sigma' = \{\alpha \texttt{ = cat}\}$.

Let us start again with the list of goals reduced to `big(cat)` (since clause 5 is a fact, so has no body) and substitution $\sigma' \circ \sigma = \{\texttt{X = cat}\}$.

But no clause has a head matching `big(cat)`, so let us backtrack and try another match below clause 5.

There is none.

Procedural meaning/Example (cont)

So we must backtrack further and reconsider the list of goals is $\mathcal{G} = (G_1, G_2)$, where $G_1 = \texttt{dark(X)}$ and $G_2 = \texttt{big(X)}$, and the identity substitution σ.

The clause after clause 7, whose head matches G_1 is clause 8. It does not have common variables with G_1. The resulting substitution is $\sigma' = \{\texttt{X} = \beta\}$.

Let us start again with the list of goals $(\texttt{brown}(\beta), \texttt{big}(\beta))$ and the substitution

$$\sigma' \circ \sigma = \sigma'$$

Procedural meaning/Example (cont)

Now the list of goals is $\mathcal{G} = (G_1, G_2)$, where $G_1 = \texttt{brown}(\beta)$ and $G_2 = \texttt{big}(\beta)$, and $\sigma = \{\texttt{X = } \beta\}$.

The first clause whose head matches G_1 is clause 4, which contains not variable (it is a fact).

The matching leads to substitution $\sigma' = \{\beta \texttt{ = bear}\}$.

Let us start again with the list of goals `big(bear)` and the substitution

$$\sigma' \circ \sigma = \{\texttt{X = bear}\}$$

Procedural meaning/Example (cont)

Now the the list of goals is $\mathcal{G} = (G_1)$, where $G_1 = \texttt{big(bear)}$, and the substitution

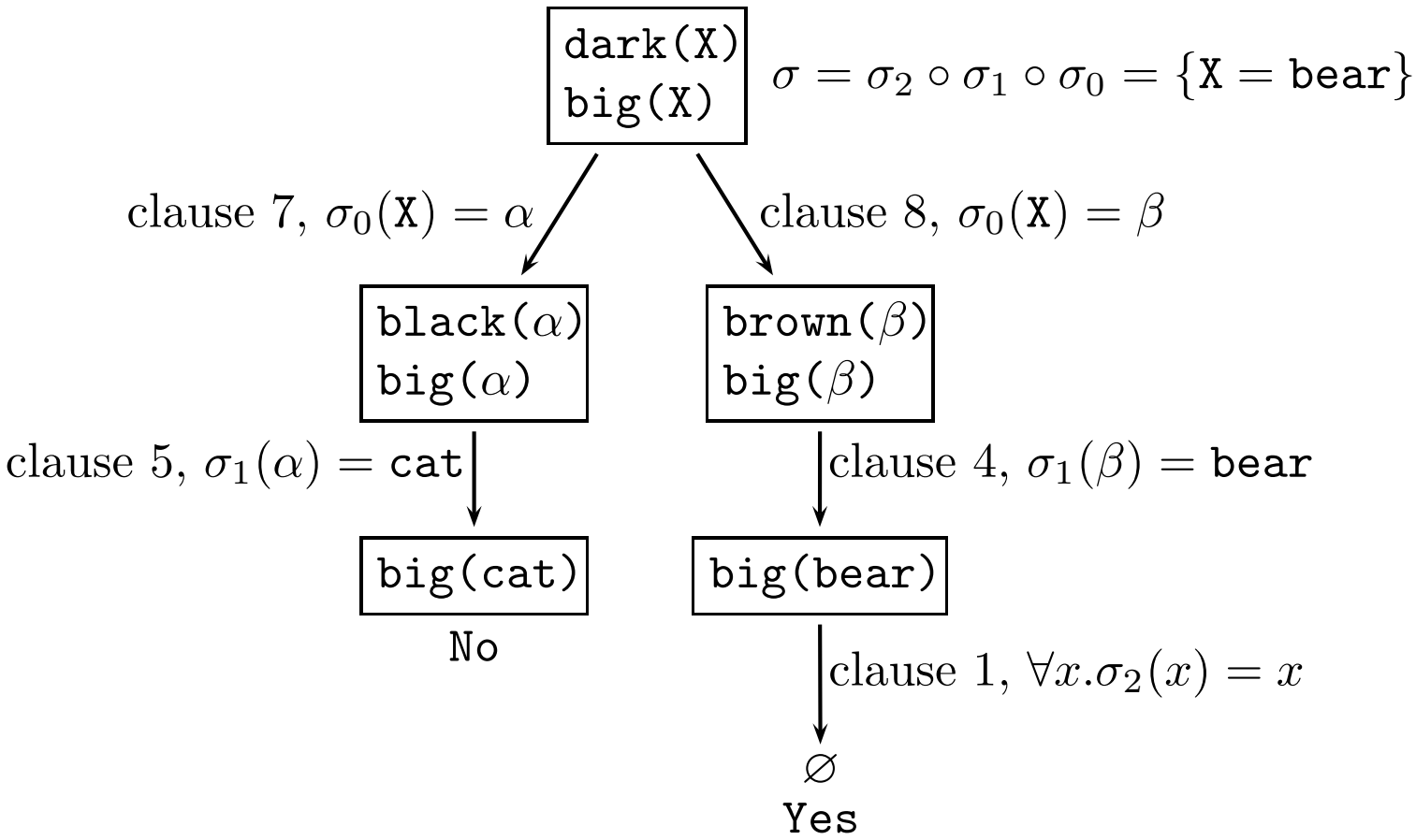
$$\sigma = \{\texttt{X = bear}\}$$

The first clause whose head matches G_1 is clause 1. It has no variable. The resulting substitution is the identity. Since it is a fact, it is proven and there is no new (sub-)goals.

This means that the interpreters ends with the positive result

```
?- dark(X), big(X).   % What is dark and big?
X = bear
Yes
```

Procedural meaning/Example (cont)



Procedural meaning/Example (cont)

The corresponding proof tree is

$$\langle 8, \{\texttt{Z = BEAR}\}\rangle \dfrac{\dfrac{\langle 4\rangle}{\texttt{brown(bear)}}}{\texttt{dark(bear)}} \qquad \dfrac{\langle 1\rangle}{\texttt{big(bear)}}}{\texttt{dark(bear),big(bear)}}$$

Declarative versus procedural meaning (resumed)

Given a Prolog program, it is possible to provide several programs which have the same declarative meaning but potentially different procedural meanings by playing on the order of the clauses and the order of the goals in the bodies. For example, consider again the relation `ancestor` page 36:

```
ancestor(X,Y) :- parent(X,Y).                    % Version 1
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).

ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).     % Version 2
ancestor(X,Y) :- parent(X,Y).

ancestor(X,Y) :- parent(X,Y).                    % Version 3
ancestor(X,Y) :- ancestor(Z,Y), parent(X,Z).

ancestor(X,Y) :- ancestor(Z,Y), parent(X,Z).     % Version 4
ancestor(X,Y) :- parent(X,Y).
```

Declarative versus procedural meaning (resumed)

Let x and y be some data object. Given a query

```
?- ancestor(x, y).
```

The procedural behaviours of the different versions are as follows:

- versions 1 and 2 always allow an answer to be found;
- versions 3 and 4 always loop forever.

Consider the examples:

```
?- ancestor(liz,jim).
?- ancestor(tom,pat).
```

Declarative versus procedural meaning (resumed)

What about the following variations?

```
ancestor(X,Y) :- parent(X,Y).                    % Version 3bis
ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y).

ancestor(X,Y) :- ancestor(X,Z), parent(Z,Y). % Version 4bis
ancestor(X,Y) :- parent(X,Y).
```