• The cut permits us to force the evaluation of a series of subgoals on the right-hand side of a rule not to be retried if the right-hand side succeeds once.

• You can think about the cut as a conditional statement.

• The cut is implemented by !.

• Example 1:

\[
f(x) = \text{if } x=0 \text{ then } 1 \text{ else } 5
\]

**PROLOG**:

\[
f(0,1),
\]

\[
f(X,5) :- X>0.
\]

is the same as:

\[
f(0,1) :- !, 
\]

\[
f(X,5) :- . 
\]

---

**Natural language processing**

• Write a program that is effectively a BNF grammar, which, when executed, will parse sentences in a natural language.

• Consider the following BNF grammar:

\[
s \rightarrow np \ vp \\
np \rightarrow det \ n \\
vp \rightarrow tv \ np \\
\rightarrow iv \\
det \rightarrow \text{the} \\
\rightarrow a \\
\rightarrow an \\
n \rightarrow \text{giraffe} \\
\rightarrow \text{apple} \\
iv \rightarrow \text{dreams} \\
tv \rightarrow \text{eats} \\
\rightarrow \text{dreams}
\]

**Representation in Prolog**

• We represent a sentence using a list.

• We can write Prolog rules that partition a sentence into its grammatical categories using the structure defined by the BNF grammar.

• For example:

\[
s \rightarrow np \ vp
\]

is represented by:

\[
s(X,Y) :- np(X,U),vp(U,Y). \\
X \text{ is the sentence being parsed and } Y \text{ represents the resulting tail of the list that will remain to parse if this rule succeeds (to be applied).}
\]
• Assume we want to parse "the giraffe dreams". We write the query:

```
s([the, giraffe, dreams], X).
```

? - s([the, giraffe, dreams], []).
Call: ( 7) s([the, giraffe, dreams], []) ?

• Assume we want to parse "the giraffe sleeps". We write the query:

```
s([the, giraffe, sleeps], X).
```

The result will be "No".

• Assume we want all the sentences parsed by the grammar.

```
s(Sentence, Y).
```

If we modify slightly each rule we can add the capability to generate a parse tree directly from the grammar.

This is done by adding an additional parameter to each rule and appropriate variables to hold the intermediate values that are derived.

For example, the parse tree of "the giraffe dreams" can be represented by:

```
s(np(det(the), n(giraffe)), vp(iv(dreams)))
```

Here is the modified Prolog program:

```
s(s(NP, VP)) -> np(NP), vp(VP).
np(NP) -> det(DT), n(N).
vp(VP) -> iv(VP).
v(t(V), np(NP)) -> t(V), np(NP).
det(det(the)) -> [the].
det(det(a)) -> [a].
n(n(giraffe)) -> [giraffe].
n(n(apple)) -> [apple].
iv(iv(dreams)) -> [dreams].
tv(tv(dreams)) -> [dreams].
v(t(v(dreams))) -> [eats].
```

Here are some possible queries:

```
s(Tree, [the, giraffe, dreams], X).
s(Tree, Sentence, X).
```

• We use a notation called Definite Clause Grammar (DCG).

• This notation is close from the notation of context-free grammars rules.

• We use the operator --> instead of :->.

• We remove the variables from the rules.

• But the meaning and the arity of the predicates do not change.

• DCG representation of the previous BNF grammar:

```
s --> np, vp.
np --> det, n.
vp --> iv.
det --> [the].
n --> [giraffe].
n --> [apple].
v --> [dreams].
tv --> [dreams].
v --> [eats].
```

• Queries are the same as previously.

DCG representation of the JAY concrete syntax

```
expression --> conjunction, ['|'], expression ; conjunction.
conjunction --> relation, ['&'], conjunction ; relation.
relation --> addition, ['<='], relation ;
            addition, ['>='], relation ;
            addition, ['='], relation ;
            addition, ['<'], relation ;
            addition, ['>'], relation ;
            addition, ['->'], relation ;
            addition, ['=->'], relation ;
            addition.
addition --> term, ['+'], addition ;
            term, ['-'], addition ;
            term.
term --> factor, ['*'], term ;
            factor, ['/'], term ;
            factor.
factor --> ['('], expression, [')'] ; [id] ; [lit].
```
expression(expression(\textit{C}, [\textit{E}] \rightarrow \textit{conjunction}(\textit{C}, [\textit{E}]), expression(\textit{C})).

\textit{conjunction} \rightarrow \textit{conjunction}(\textit{C}, \textit{C}).

\textit{relation}(A, B, C) \rightarrow \textit{addition}(\textit{A}, \textit{B}, \textit{C}).

\textit{addition}(\textit{A}, \textit{B}, \textit{C}) \rightarrow \textit{relation}(\textit{A}, \textit{B}, \textit{C}).

\textit{term}(\textit{factor}(\textit{F}), \textit{term}(\textit{F}), \textit{term}(\textit{T})) \rightarrow \textit{factor}(\textit{F}).

\textit{factor}(\textit{F}) \rightarrow \textit{identifiable}(\textit{F}).