PROLOG
(FACTOS, REGRAS E UNIFICAÇÃO)

Parcialmente adaptado de

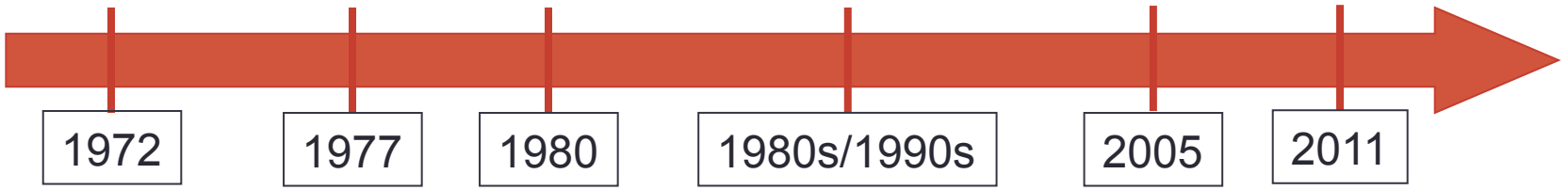
http://www.learnprolognow.org/

Prolog

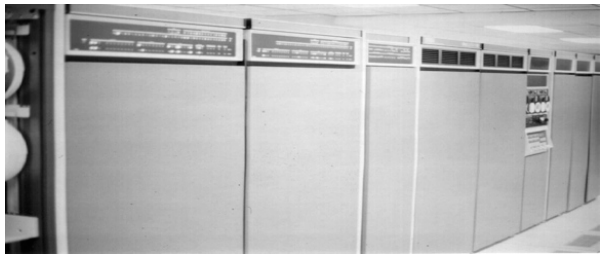
- "PROgramação em LÓGica"
- Declarativa
- Muito diferente das outras linguagens de programação (procedimentais)
- Adequada para tarefas baseadas em conhecimento

História do Prolog

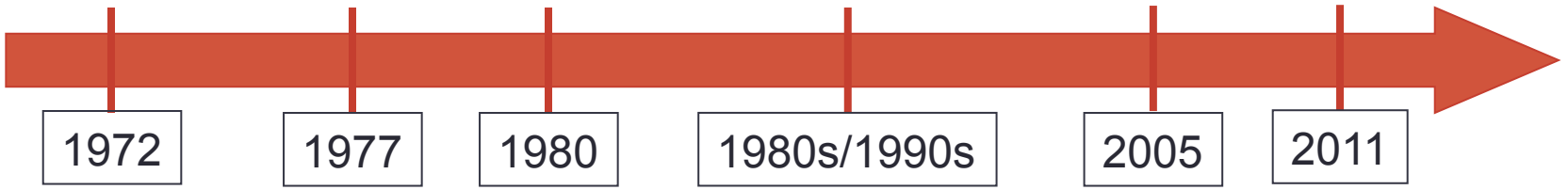
primeiro interpretador
de Prolog
(Colmerauer e Roussel)



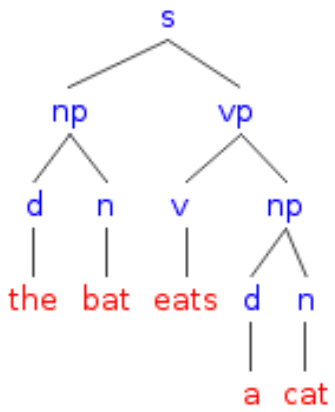
História do Prolog



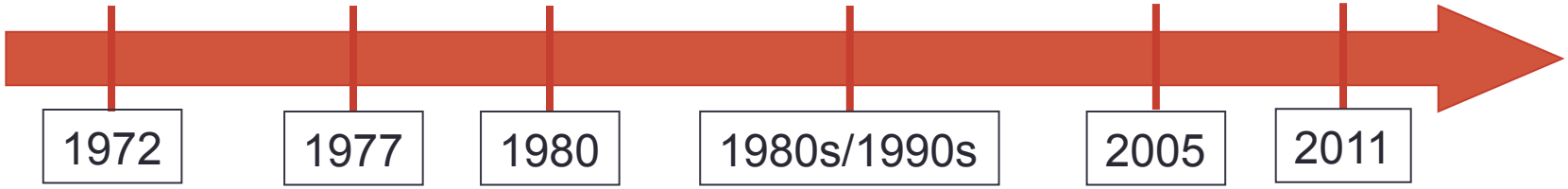
implementação do compilador
DEC10
(Warren)



História do Prolog

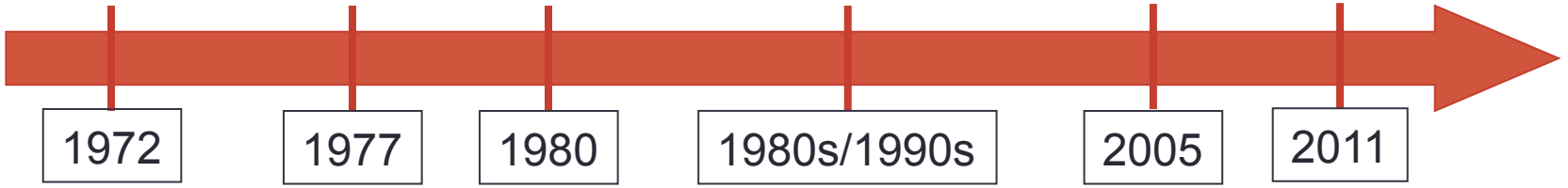


implementação de
Gramáticas de Cláusulas Definidas
(Pereira e Warren)

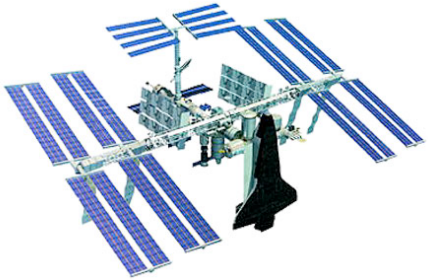


História do Prolog

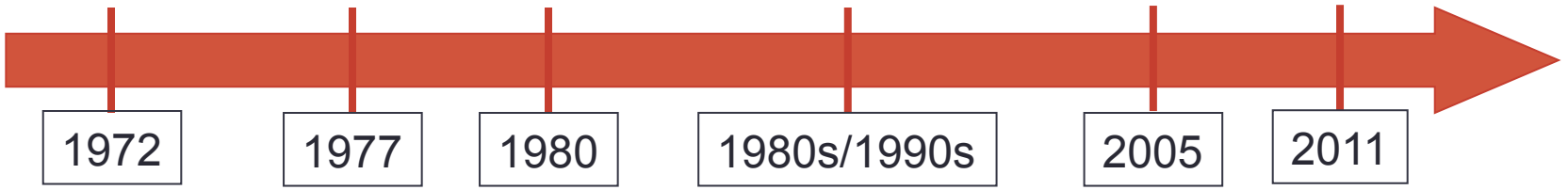
Prolog ganha popularidade especialmente na Europa e no Japão



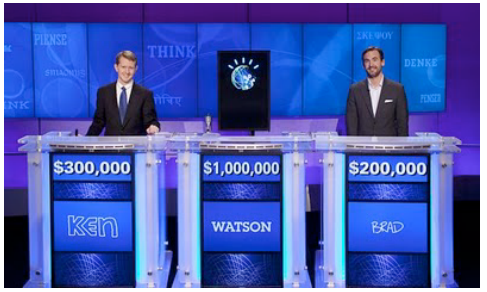
História do Prolog



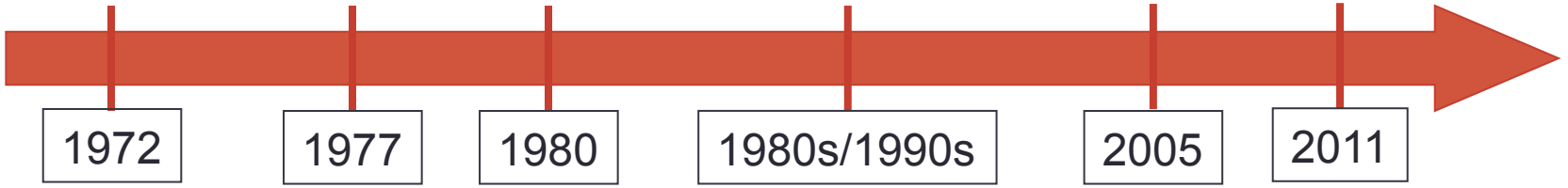
Prolog usado para programar interface de língua natural Estação Espacial Internacional (NASA)



História do Prolog



Prolog usado para programar parte do Watson



Ideia básica do Prolog

- Descrever o problema numa base de conhecimento em lógica
- Fazer interrogações
- O Prolog deduz novos factos que são consequência lógica
- O Prolog devolve as suas deduções como respostas às interrogações

Implicações

- Pensar declarativamente, não procedimentalmente
  - Difícil
  - Exige uma abordagem diferente
- Linguagem de alto-nível
  - Não tão eficiente como, por exemplo, C
  - Adequada para modelação rápida
  - Útil em muitas aplicações em IA

Base de Conhecimentos 1

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

```
?-
```

Base de Conhecimentos 1

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

```
?- woman(mia).
```

Base de Conhecimentos 1

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

```
?- woman(mia).
yes
?-
```

Base de Conhecimentos 1

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

```
?- woman(mia).
yes
?- playsAirGuitar(jody).
```

Base de Conhecimentos 1

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

```
?- woman(mia).
yes
?- playsAirGuitar(jody).
yes
?-
```

Base de Conhecimentos 1

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

```
?- woman(mia).
yes
?- playsAirGuitar(jody).
yes
?- playsAirGuitar(mia).
```

Base de Conhecimentos 1

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

```
?- woman(mia).
yes
?- playsAirGuitar(jody).
yes
?- playsAirGuitar(mia).
no
?-
```

Base de Conhecimentos 1

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

```
?- tattoed(jody).
```

Base de Conhecimentos 1

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

```
?- tattoed(jody).
no
?-
```

Base de Conhecimentos 1

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

```
?- tattoed(jody).
ERROR: predicate tattoed/1
    not defined.
?-
```

Base de Conhecimentos 1

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

```
?- party.
```

Base de Conhecimentos 1

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

```
?- party.
yes
?-
```

Base de Conhecimentos 1

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

```
?- party.
yes
?- rockConcert.
```

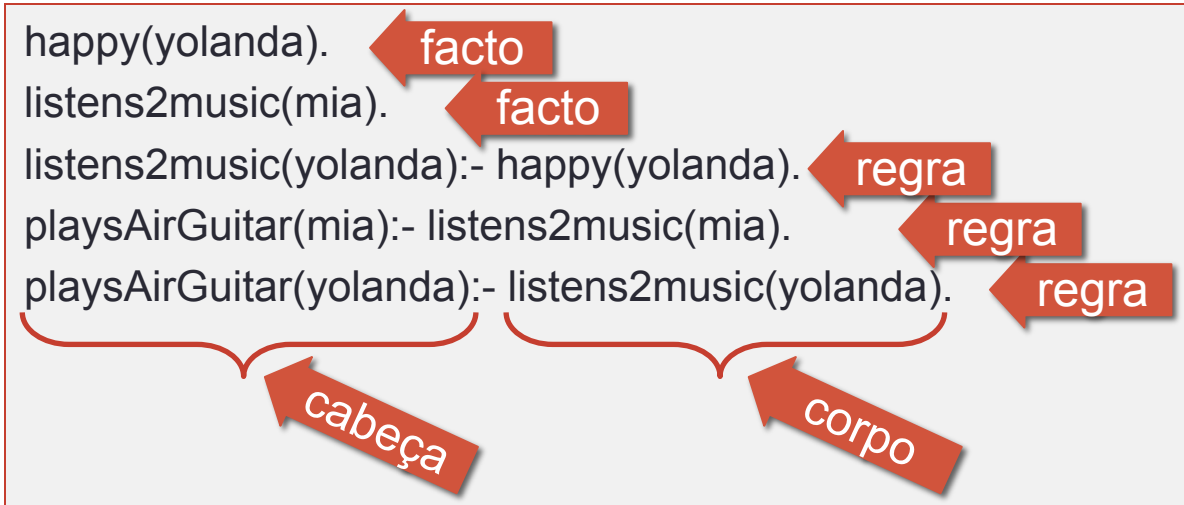
Base de Conhecimentos 1

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

```
?- party.
yes
?- rockConcert.
no
?-
```

Base de Conhecimentos 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```



Base de Conhecimentos 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

```
?-
```

Base de Conhecimentos 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

```
?- playsAirGuitar(mia).
yes
?-
```

Base de Conhecimentos 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

```
?- playsAirGuitar(mia).
yes
?- playsAirGuitar(yolanda).
yes
```

Cláusulas

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

*Há 5 **cláusulas** nesta base de conhecimentos:*
2 factos e 3 regras.

O fim de uma cláusula é assinalado com um ponto

Predicados

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

*Há 3 **predicados** nesta base de conhecimentos:*
happy, listens2music, e playsAirGuitar

Base de Conhecimentos 3

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
```

Representação da Conjunção

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
```

- *A virgula "," representa a conjunção em Prolog*

Base de Conhecimentos 3

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
```

```
?- playsAirGuitar(vincent).
no
?- playsAirGuitar(butch).
yes
```

Representação da Disjunção

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
```

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch); listens2music(butch).
```

Prolog e Lógica

- Operadores
  - Implicação **:-**
  - Conjunção **,**
  - Disjunção **;**
- Uso do modus ponens
- Negação

Base de Conhecimentos 4

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

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

```
?- woman(X).
```

Base de Conhecimentos 4

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

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

```
?- woman(X).
X=mia
```

Base de Conhecimentos 4

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

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

```
?- woman(X).
X=mia;
X=jody
```

Base de Conhecimentos 4

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

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

```
?- woman(X).
X=mia;
X=jody;
X=yolanda
```

Base de Conhecimentos 4

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

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

```
?- woman(X).
X=mia;
X=jody;
X=yolanda;
no

?-
```

Base de Conhecimentos 4

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

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

```
?- woman(X).
X=mia;
X=jody;
X=yolanda;
no

?- loves(marsellus,X), woman(X).
```

Base de Conhecimentos 4

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

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

```
?- woman(X).
X=mia;
X=jody;
X=yolanda;
no

?- loves(marsellus,X), woman(X).
X=mia
yes

?-
```

Base de Conhecimentos 4

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

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

```
?- woman(X).
X=mia;
X=jody;
X=yolanda;
no

?- loves(marsellus,X), woman(X).
X=mia
yes

?- loves(pumpkin,X), woman(X).
```

Base de Conhecimentos 4

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

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

```
?- woman(X).
X=mia;
X=jody;
X=yolanda;
no

?- loves(marsellus,X), woman(X).
X=mia
yes

?- loves(pumpkin,X), woman(X).
no

?-
```

Base de Conhecimentos 5

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

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

```
?- jealous(marsellus,W).
```

Base de Conhecimentos 5

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

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

```
?- jealous(marsellus,W).
W=vincent
```

Base de Conhecimentos 5

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

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

```
?- jealous(marsellus,W).
W=vincent;
W=marsellus
```

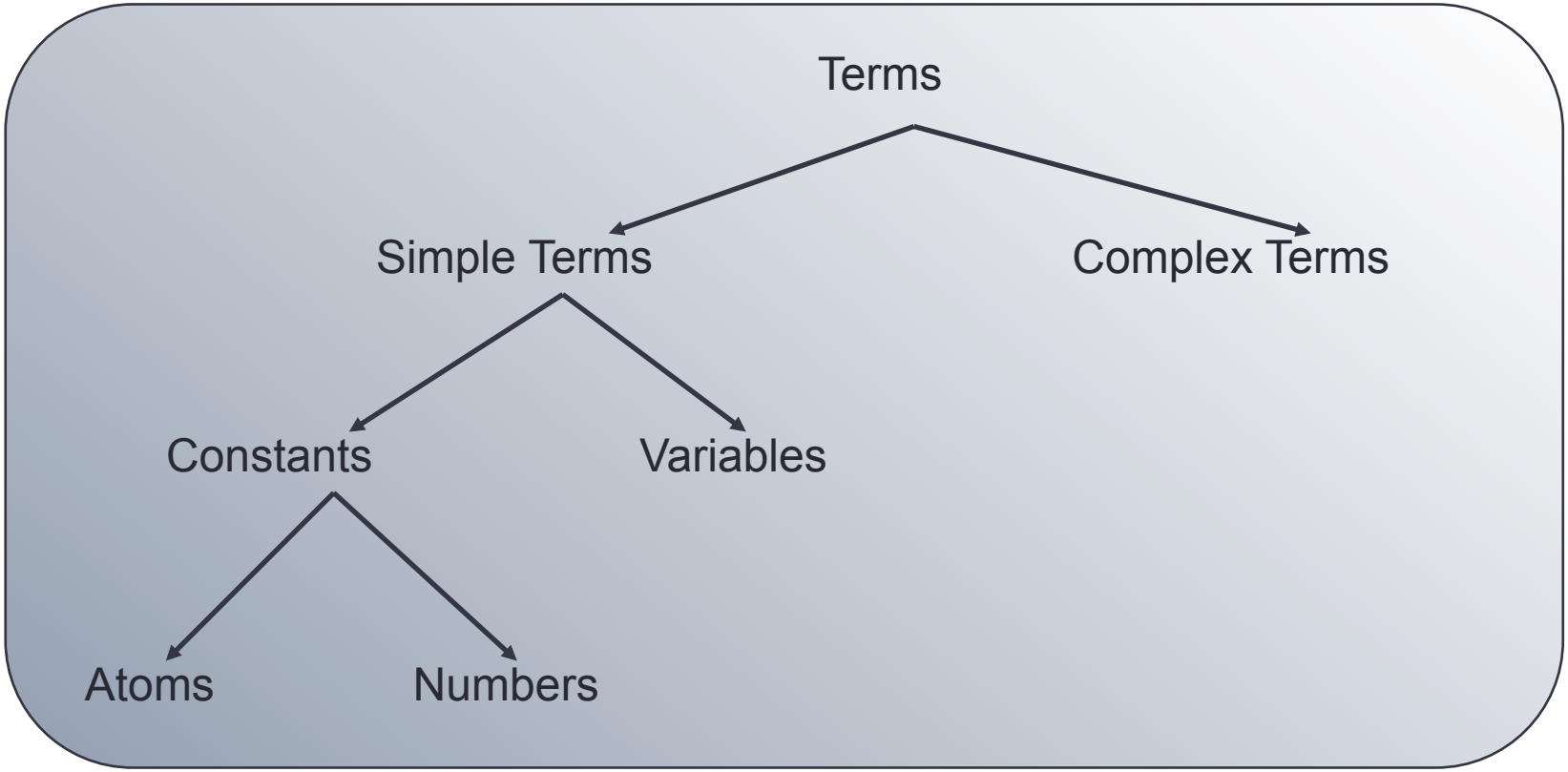
Base de Conhecimentos 5

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

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

```
?- jealous(marsellus,W).
W=vincent;
W=marsellus;
no

?-
```

Sintaxe do Prolog

- O que são os factos, regras e interrogações?



Átomos

- Uma sequência de caracteres de letras maiúsculas, minúsculas, números ou ‘underscore’, começando com uma letra minúscula
  - *Exemplos*: **butch**, **big_kahuna_burger**, **playGuitar**

- Uma sequência arbitrária de caracteres entre plicas
  - *Exemplos*: **'Vincent'**, **'Five dollar shake'**, **'@$%'**
  -
- Uma sequência de caracteres especiais
  - *Exemplos*: **: , ; . :-**

Números

- Inteiros: 12, -34, 22342
- Reais: 34573.3234

Variáveis

- Uma sequência de caracteres de letras maiúsculas, minúsculas, números ou ‘underscore’, começando com uma letra maiúsculas ou ‘underscore’

- Exemplos:
  X, Y, Variable, Vincent, _tag

Termos Complexos

- Átomos, números e variáveis são a base dos termos complexos
- Os termos complexos são construídos a partir de um functor seguido de uma sequência de argumentos
- Os argumentos são colocados entre parênteses, separados por vírgulas
- O functor tem que ser um átomo

Exemplos de termos complexos

- Exemplos anteriores:
  - playsAirGuitar(jody)
  - loves(vincent, mia)
  - jealous(marsellus, W)

- Termos complexos dentro de termos complexos :
  - hide(X,father(father(father(butch))))

Aridade

- Ao número de argumentos de um termo complexo chama-se a sua aridade

- Exemplos:

woman(mia)	é um termo com aridade 1
loves(vincent,mia)	é um termo com aridade 2
father(father(butch))	é um termo com aridade 1

A aridade é importante

- Podem-se definir dois predicados com o mesmo functor mas com aridade diferente
- Neste caso, são dois predicados diferentes
- A aridade de um predicado é geralmente indicada com o sufixo "/" seguido de um número para indicar a aridade

Exemplo de Aridade

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

- Esta base de conhecimento define
  - happy/1
  - listens2music/1
  - playsAirGuitar

Unificação

- O Prolog unifica **woman(X)** com **woman(mia)**, instanciando a variável **X** com o átomo **mia**.
- Definição:
  - Dois termos unificam se forem iguais, ou se contêm variáveis que podem ser instanciadas de maneira a que os termos resultantes fiquem iguais
- Significa que:
  - **mia** e **mia** unificam
  - **42** e **42** unificam
  - **woman(mia)** e **woman(mia)** unificam
- E que:
  - **vincent** e **mia** não unificam
  - **woman(mia)** e **woman(jody)** não unificam

Unificação

- E os seguintes termos?
  - **mia** e **X**
  - **woman(Z)** e **woman(mia)**
  - **loves(mia,X)** e **loves(X,vincent)**

Instanciações

- Quando o Prolog unifica dois termos faz todas as instanciações necessárias de modo que os termos fiquem iguais

Unificação em Prolog

1. Se T_1 e T_2 forem constantes, então T_1 e T_2 unificam se forem o mesmo átomo, ou o mesmo número.
2. Se T_1 for uma variável e T_2 um termo, então T_1 e T_2 unificam, e T_1 é instanciado com T_2. (e vice versa)
3. Se T_1 e T_2 são termos complexos, unificam se:
   a) Tiverem o mesmo functor e aridade, e
   b) Todos os argumentos correspondentes unificam, e
   c) As instanciações das variáveis são compatíveis.
4. Dois termos unificam se e só se unificam de acordo com as cláusulas 1. a 3.

Unificação em Prolog: =/2

```
?- mia = mia.
yes

?- mia = vincent.
no

?- mia = X.
X=mia
yes

?- X=mia, X=vincent.
no
```

- Ao passar pelo primeiro objectivo, instancia X com **mia**, por isso já não pode unificar com **vincent**. Logo o segundo objectivo falha.

Unificação em Prolog: =/2

```
?- k(s(g),Y) = k(X,t(k)).
X=s(g)
Y=t(k)
Yes

?- k(s(g),t(k)) = k(X,t(Y)).
X=s(g)
Y=k
yes

?- loves(X,X) = loves(marsellus,mia).
no
```

Unificação em Prolog: =/2

- O Prolog não usa o algoritmo standard de unificação. Exemplo:

  ?- father(X) = X.

- Os termos unificam ou não?

- De acordo com o algoritmo anterior, não. Termos infinitos
  - X=father(…
  - Implementações (antigas) ficarão sem memória.
- Nalgumas implementações mais sofisticadas:

  ?- father(X) = X.

  X=father(father(father(…))))

  yes
- O algoritmo standard de unificação executa um occurs check pelo que o resultado será

  ?- father(X) = X.

  no
- Podemos instruir o Prolog para executar o occurs_check:

  ?- unify_with_occurs_check(father(X), X).

  no

Programação com Unificação

```
vertical(line(point(X,Y),point(X,Z))).
horizontal(line(point(X,Y),point(Z,Y))).
```

```
?- vertical(line(point(1,1),point(1,3))).
yes

?- vertical(line(point(1,1),point(3,2))).
no

?- horizontal(line(point(1,1),point(2,Y))).
Y = 1;
no

?- horizontal(line(point(2,3),Point)).
Point = point(_554,3);
no
```

A resposta é um termo complexo, que foi construído apenas usando unificação, e nada mais. Nomeadamente, não recorre a inferência usando modus ponens.

Criar termos complexos a partir da unificação é um dos mecanismos mais poderosos do Prolog.