Aula 4 – Programação Lógica (Prolog)

22705/1001336 - Inteligência Artificial
2019/1 - Turma A
Prof. Dr. Murilo Naldi

naldi@dc.ufscar.br

Agradecimentos

- Agradecimentos pela base do material utilizado nesta aula foi cedido ou adapatado do material dos professores Maria Carmo Nicoletti, Maria Carolina Monard, Solange Rezende, Andréia Bonfante, Heloísa Camargo e Ricardo Cerri.

Sistemas de Dedução

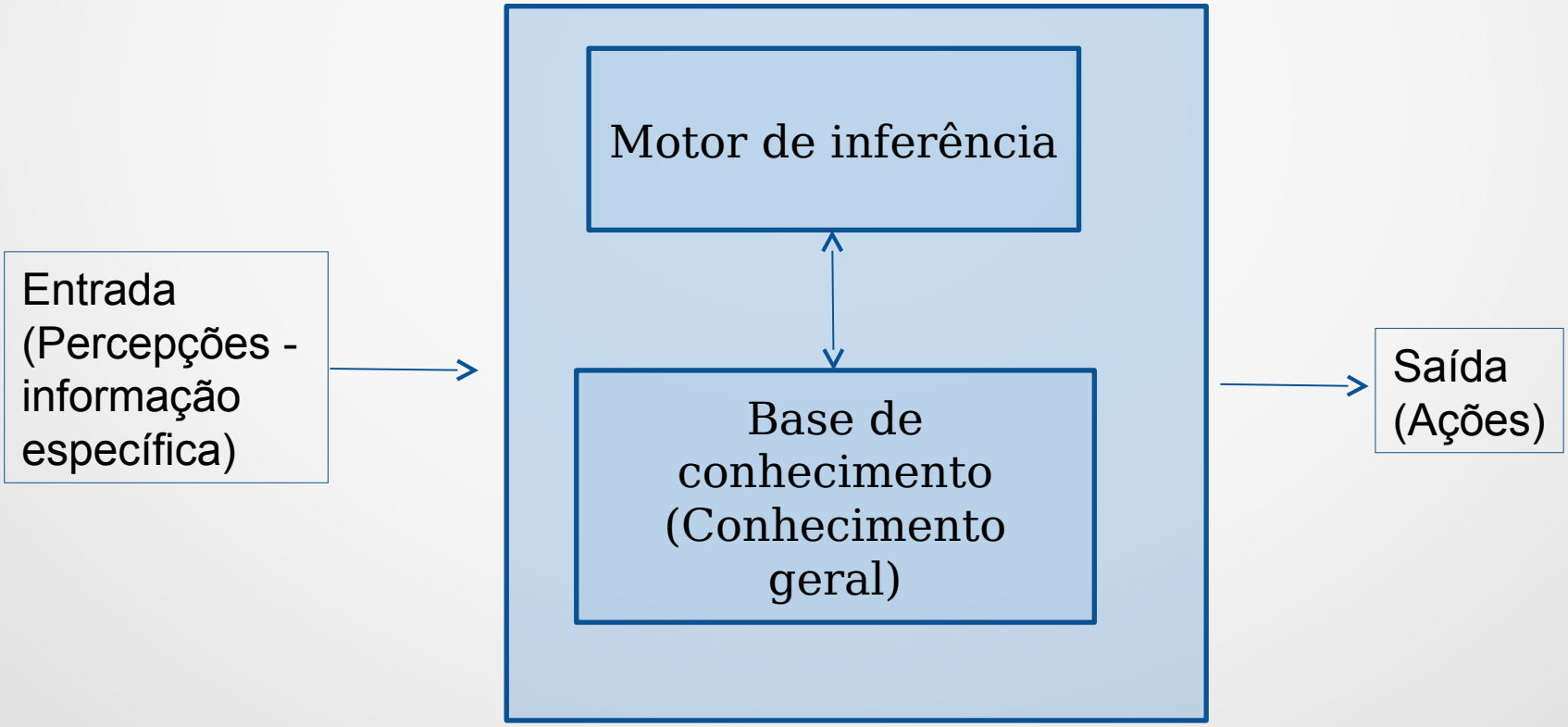
- Consistem em um conjunto de axiomas e regras de inferência que podem ser utilizadas para derivar teoremas
  - Exemplo: Datalog
- Os sistemas de dedução:
  - São insensíveis a ordem das regras
  - Não possui predicados especiais de otimização
  - Funções são simples

Programação lógica

- Tem como base os sistemas de dedução, mas possuem formulações mais complexas e funções especiais, o que os torna mais eficientes
- A linguagem de programação lógica a ser estudada neste curso é a **Prolog** (*PROgrammation en LOGique*)

Porque estudar Prolog?

- Vimos em aulas anteriores que sistemas baseados em conhecimento e agentes inteligentes recebem percepções do ambiente e devem retornar ações



Porque estudar Prolog?

- Vimos também que Sistemas de Produção utilizam um conjunto de regras e fatos (base de conhecimento) e um motor de inferência
  - Portanto, podem ser utilizados dentro dos SBCs
- Prolog é uma linguagem de programação que tem tudo isso e muito mais!
  - Incluindo memorização e retrocesso
  - Corte para otimização e controle
  - Capacidade de consulta
  - Interface com diversas outras linguagens

Programação lógica

- Apropriada para:
  - processamento simbólico, não numérico
  - resolução de problemas que envolvam objetos e relações entre objetos
- Mecanismos Básicos:
  - casamento de padrão
  - estruturas de listas
  - retrocesso (*backtracking*) automático

Prolog

- Programa (lógica): base de fatos + base de regras = base de conhecimento
- Execução (controle): atribuição de valores-verdade a proposições lógicas, pelo mecanismo de inferência (próxima aula)
- Dado um programa e uma proposição lógica, a execução será uma:

RESOLUÇÃO DE TEOREMAS

PARA CLÁUSULAS DE HORN

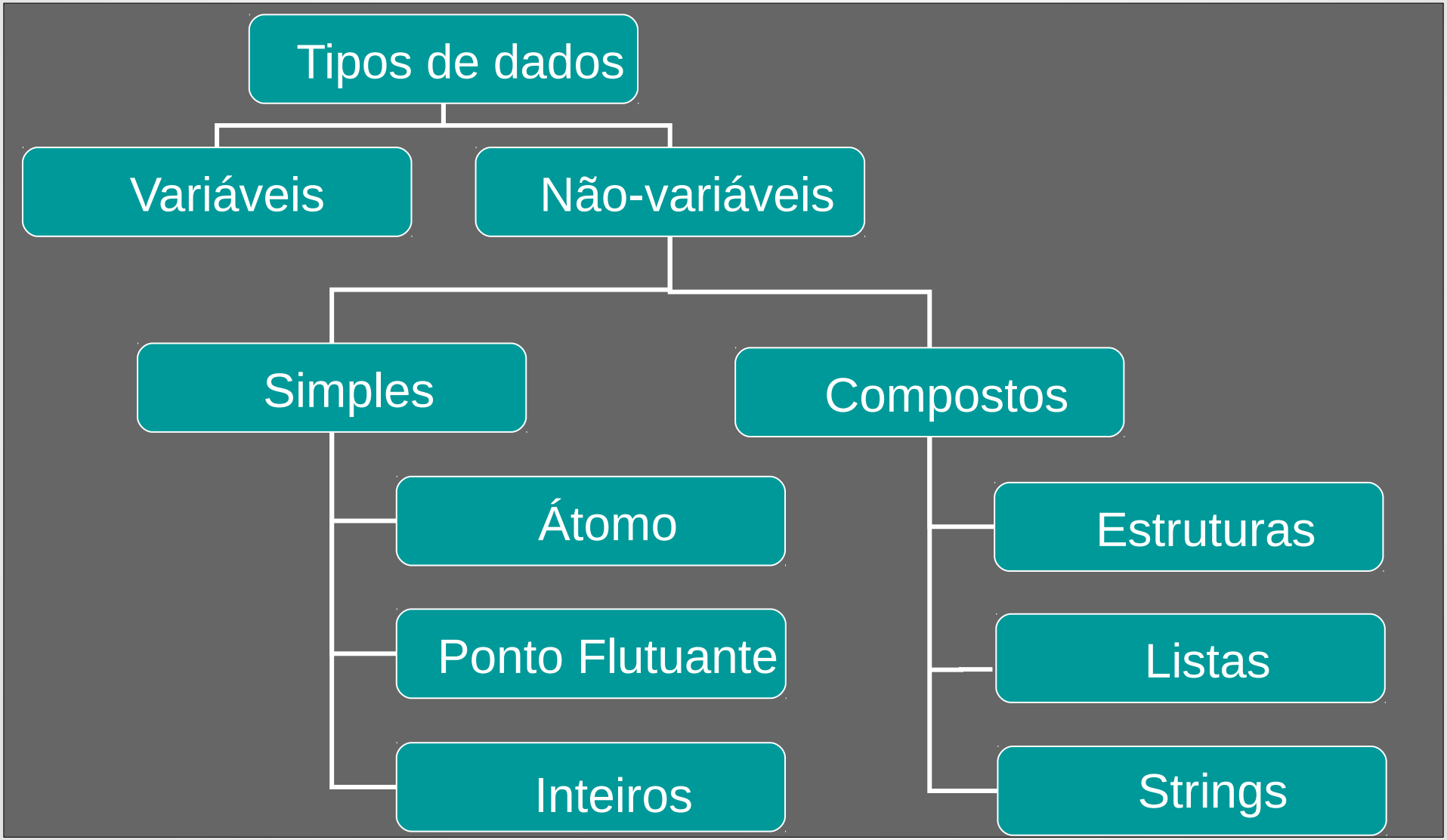
Como funciona?

- Em essência, um programa Prolog consiste de:
  - declaração de fatos a respeito de objetos e suas relações.
  - definição de regras que podem resultar em teoremas.
  - consulta a respeito de objetos e suas relações.

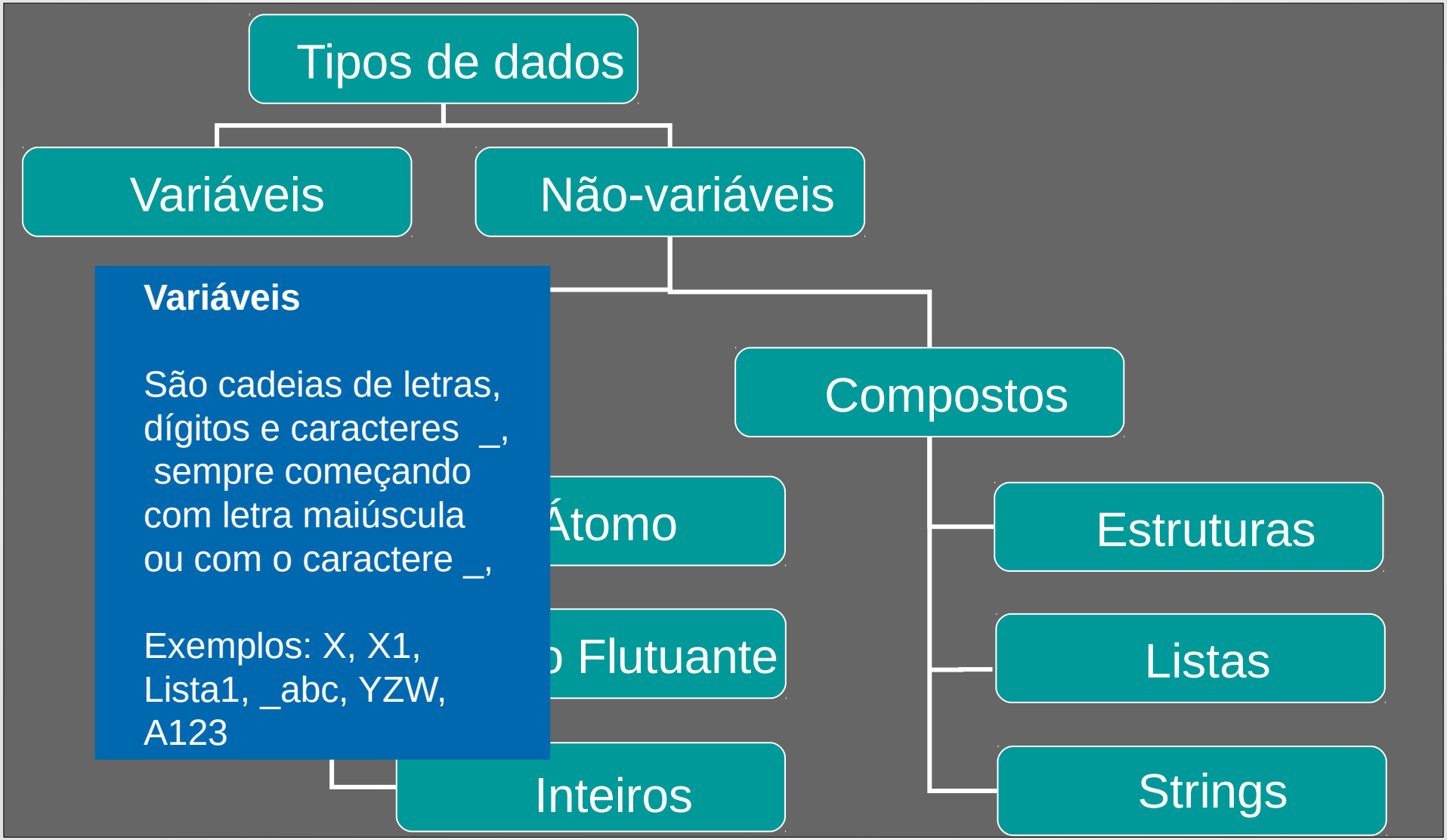
Dados e objetos

- Dados são valores ou ocorrências em estado bruto
  - Base para ser obter informações
    - Base para se obter conhecimento
- São utilizados para representar termos
  - Pessoas, agentes, objetos, etc…
- Prolog
  - Possui uma variedade de tipos de dados

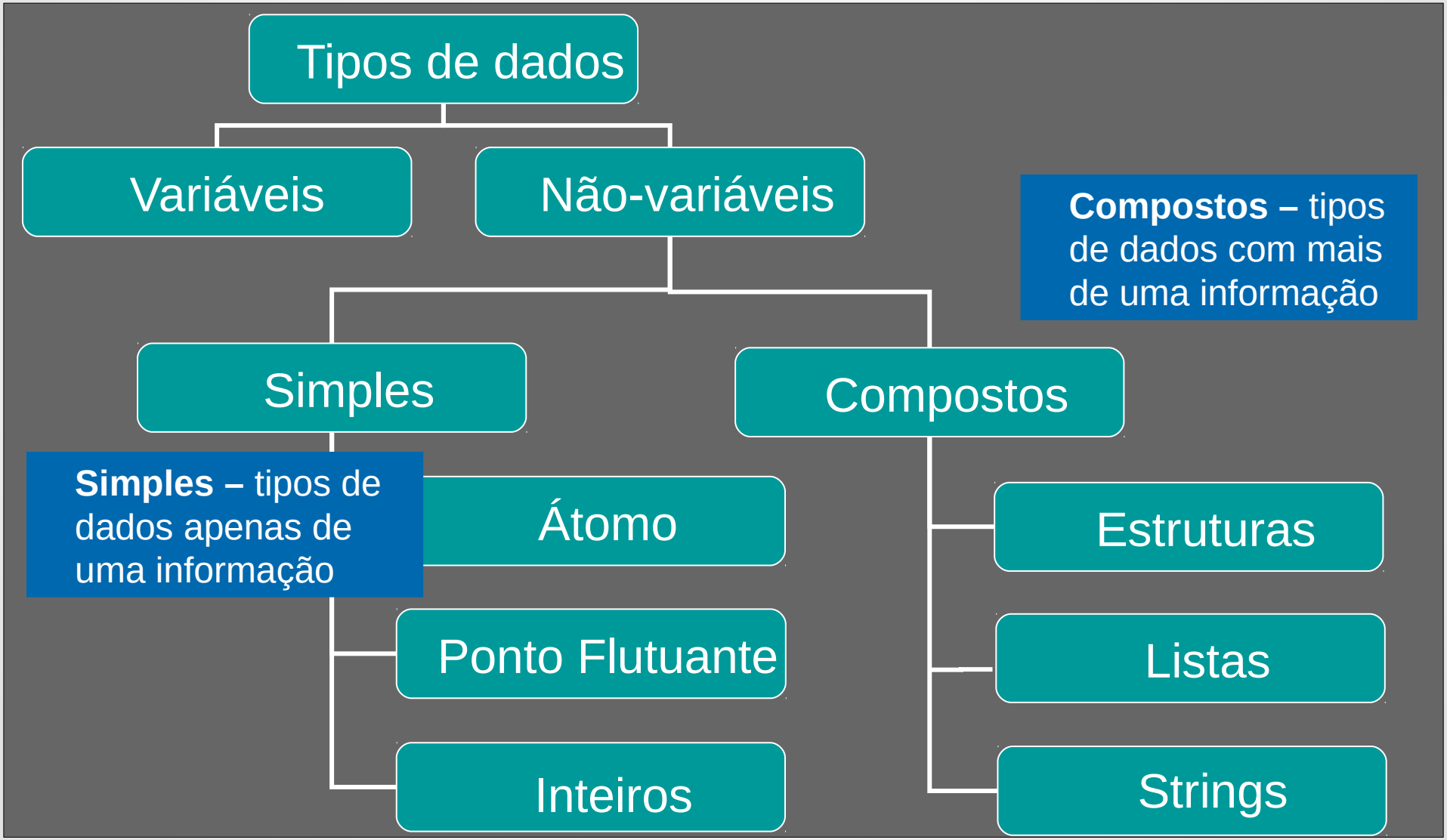
Tipos de dados



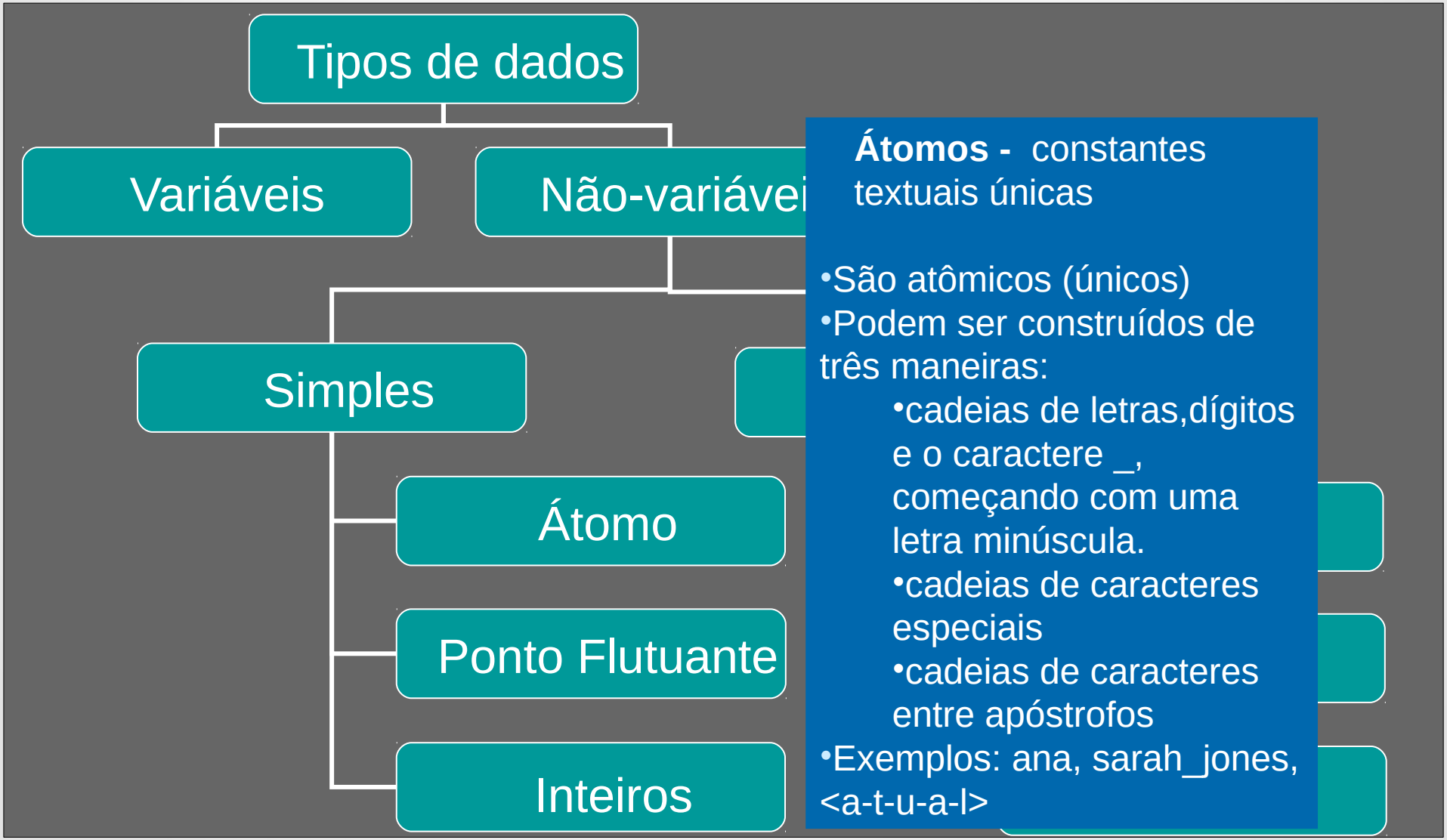
Tipos de dados



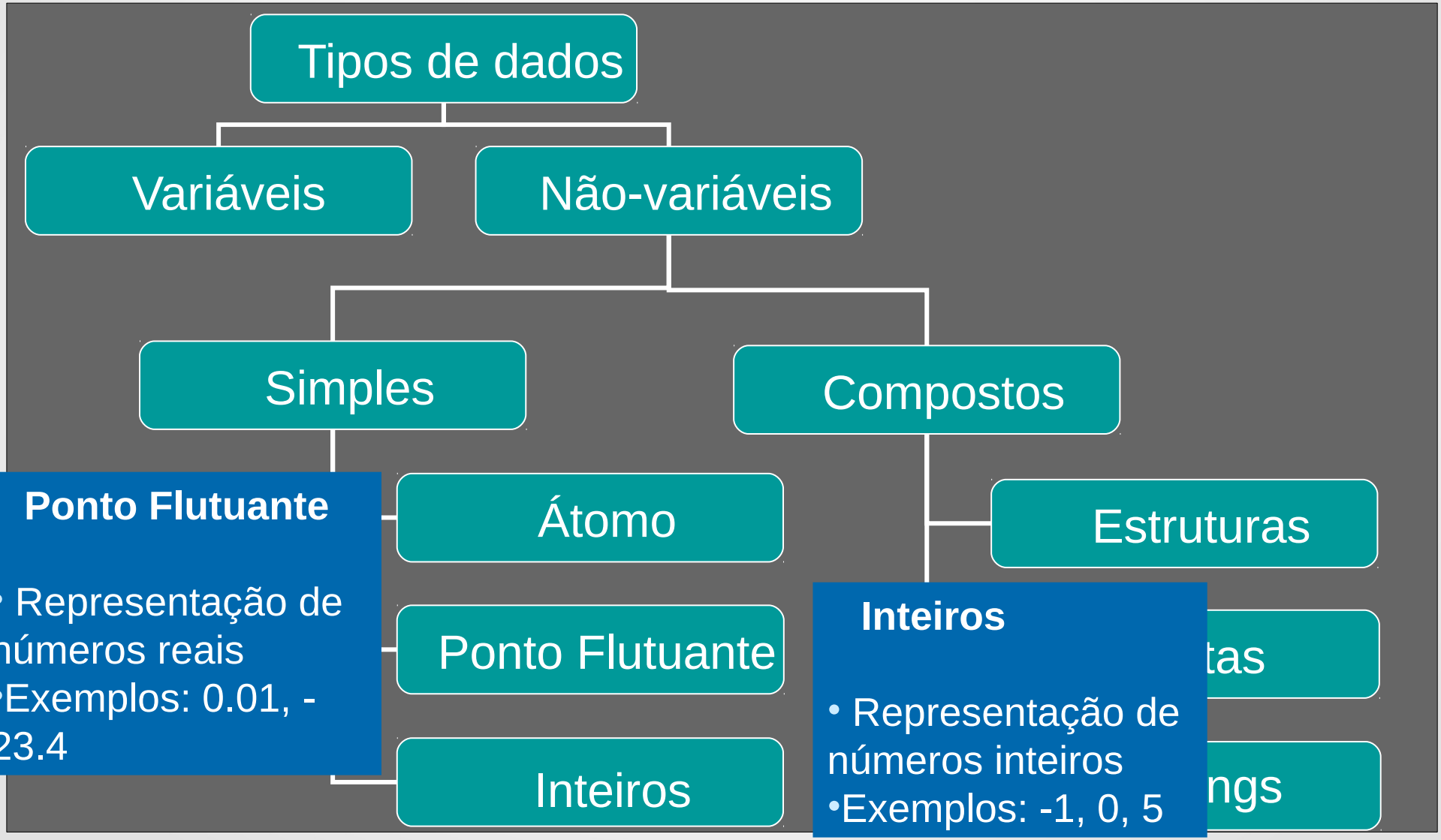
Tipos de dados



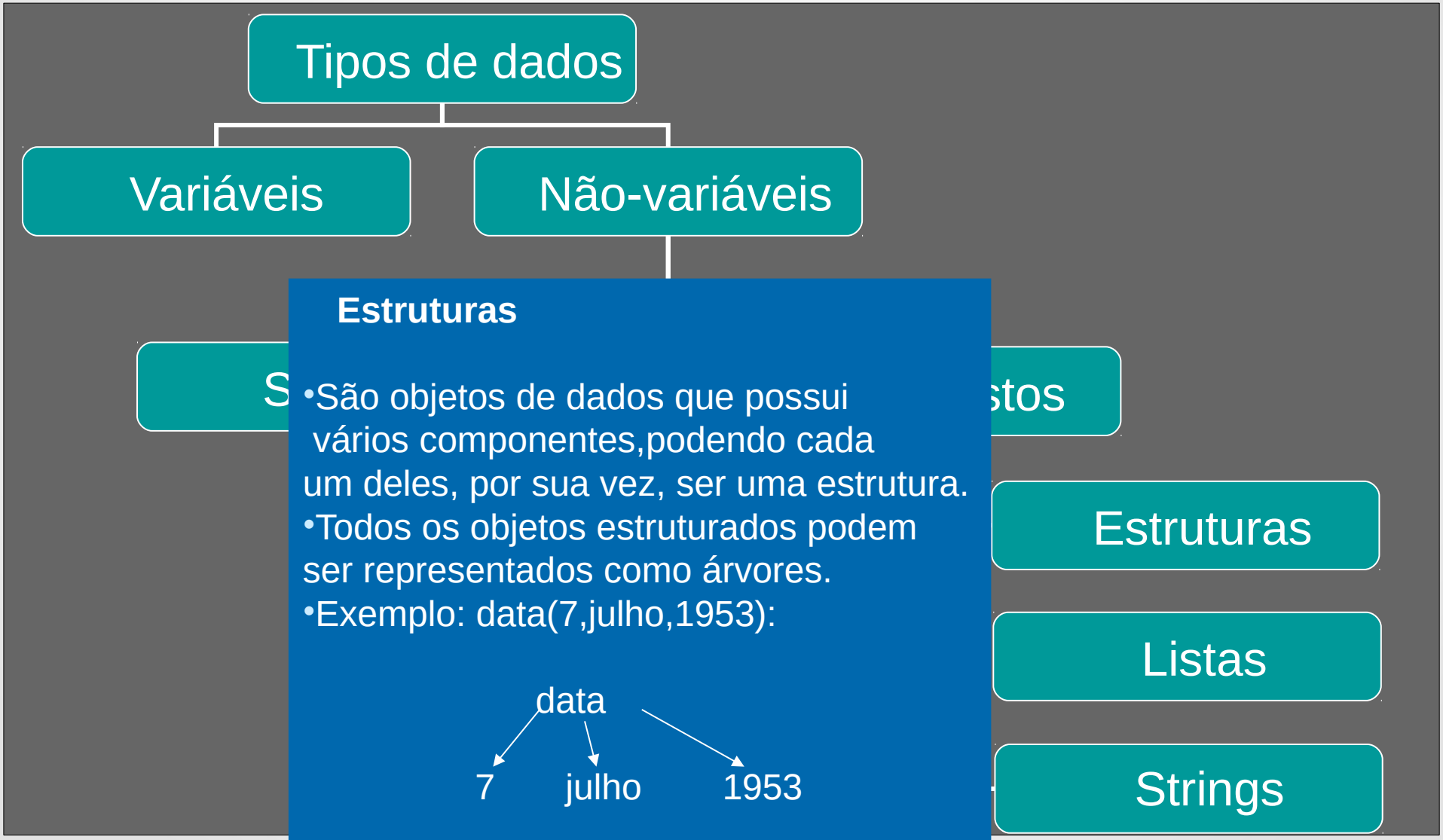
Tipos de dados



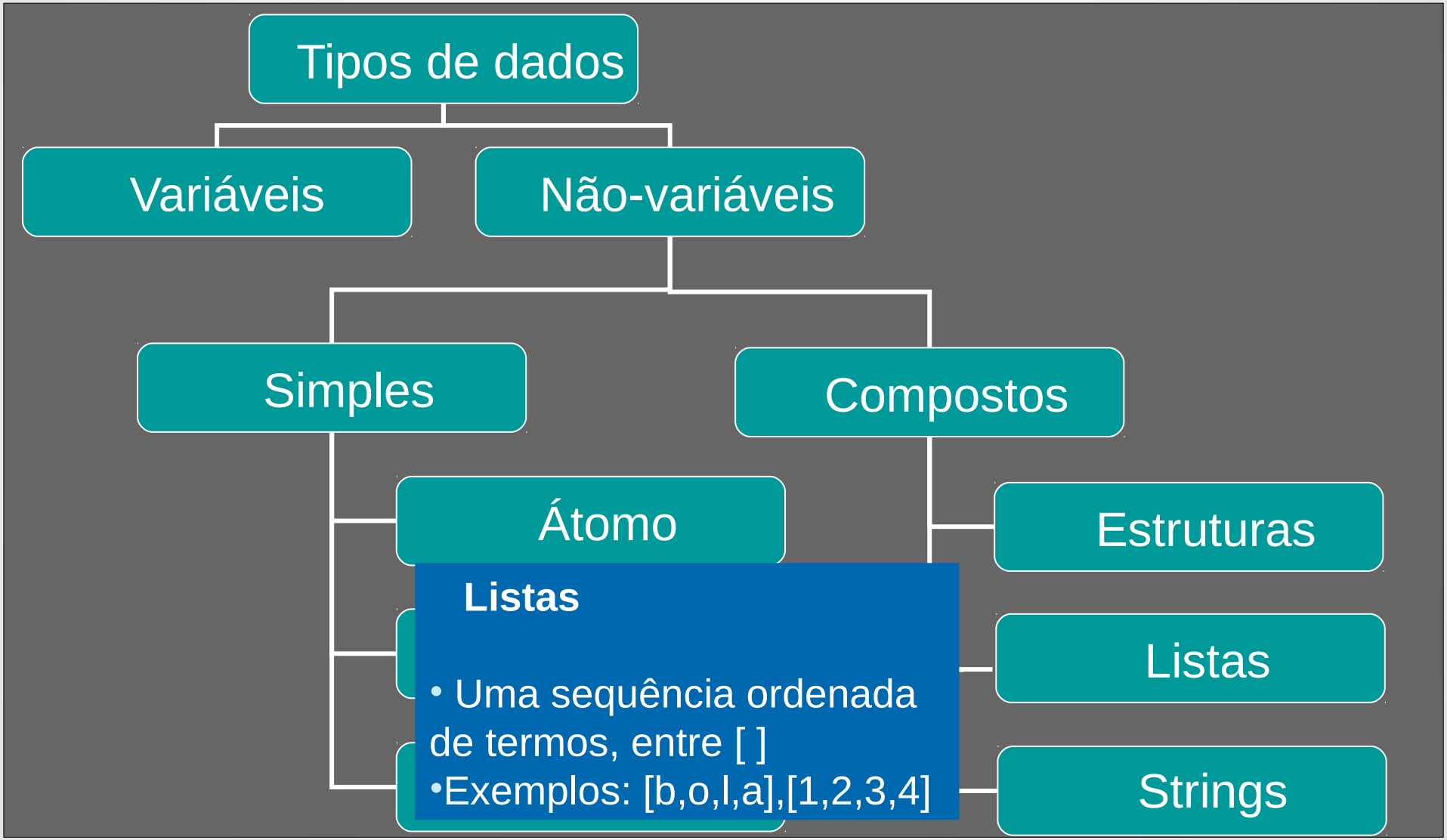
Tipos de dados



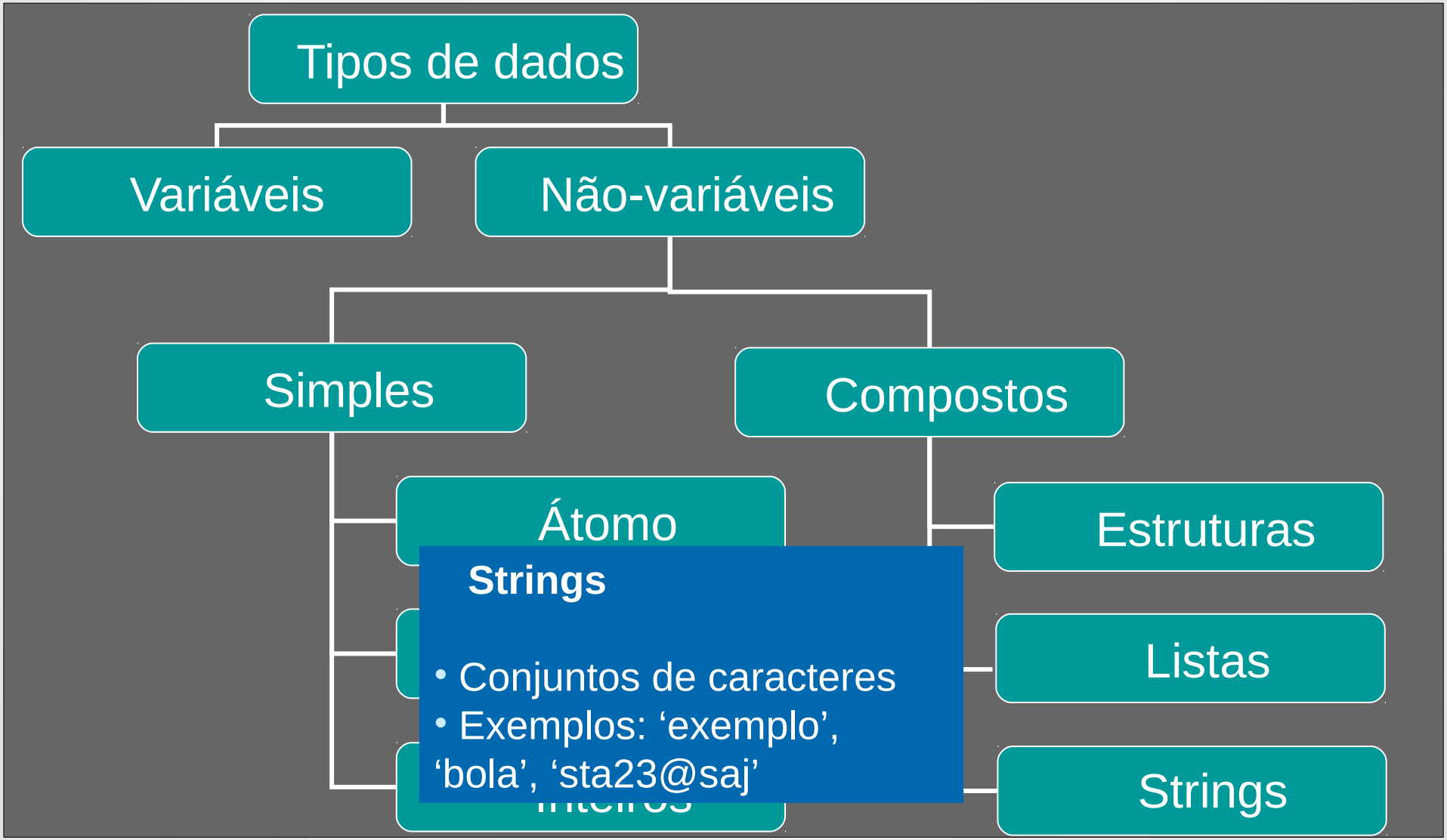
Tipos de dados



Tipos de dados



Tipos de dados



Objetos estruturados

- Objetos estruturados (ou simplesmente estruturas)
- Formado por componentes simples ou estruturas
- Sempre começa com letra minúscula (não variável).
- Também conhecidos como estruturas de **funtor**

Objetos estruturados

- Forma geral: estrutura (funtor) com aridade n
  - símboloDeFuntor(t1, t2, t3,…tn)

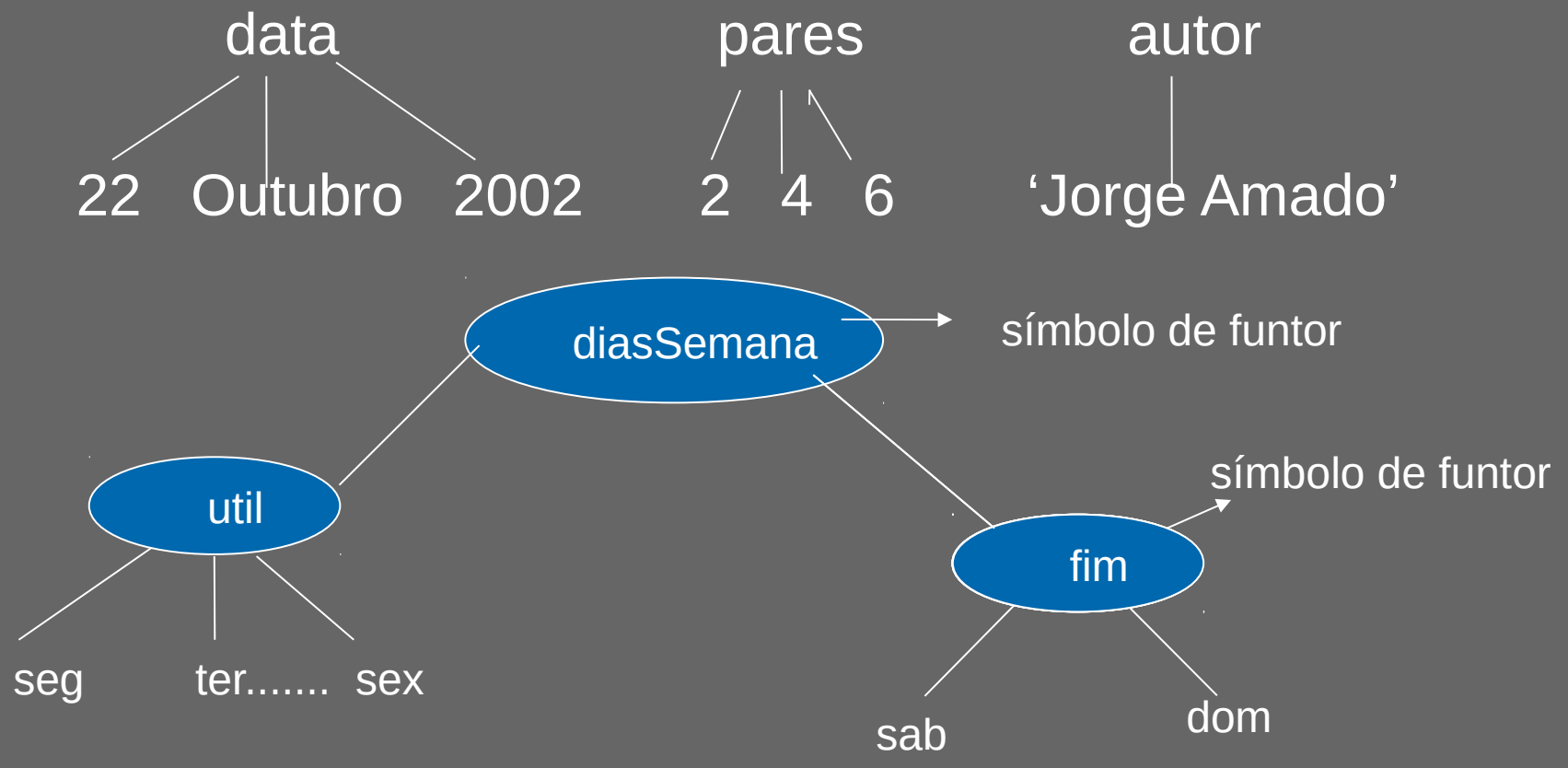
- Exemplos:
- data(22,outubro,2002) //Funtor data, com aridade 3
- pares(2,4,6) //Funtor pares, com aridade 3
- autor(‘Jorge Amado’) //Funtor autor, com aridade 1
- semana(dom,seg,ter,qua,qui,sex,sab)
- //Funtor semana com aridade 7

Objetos estruturados

- Podem ser compostos por outras estruturas
- Exemplo:
  - diasSemana(util(seg,ter,qua,qui,sex),fim(sab,dom))
  - Funtor diasSemana, com aridade 2
    - Termo 1: util(seg,ter,qua,qui,sex)
      - Funtor util, com aridade 5
    - Termo 2: fim(sab,dom)
      - Funtor fim, com aridade 2

Objetos estruturados

- Representação em árvore:



Predicados (relações)

- Prolog trabalha com lógica de primeira ordem (LPO)
  - conhecida como lógica de predicados
- Predicados são características ou relações entre objetos
  - Servem expressar proposições lógicas elementares ou axiomas

Predicados (relações)

- Exemplos:
  - pai_de(joao,pedro): “João é pai de Pedro.”
  - bonita(maria): “Maria é bonita.”
  - gosta_de(ana,vinho): “Ana gosta de vinho.”
  - ordem(1,3,4,8): “1, 3, 4 e 8 estão numa relação de ordem”.

Estruturas X Predicados

- **CUIDADO**
  - Um objeto estruturado tem a **MESMA SINTAXE** que uma relação (estrutura de predicado), porém, a **SEMÂNTICA é DISTINTA**
- Estrutura de objeto é um **TERMO**
- Relação é uma **AFIRMAÇÃO** sobre objetos

Estrutura X Predicado

- Relações possuem valor-verdade associado
  - pai(joao,pedro): é verdade que João é pai de Pedro.
  - bonita(maria): é verdade que Maria é bonita.
  - gosta(ana,vinho): é verdade que Ana gosta de vinho.
- Estruturas de objetos (funtores): não possuem valor-verdade associado
  - data(22,outubro,2002)
  - Pares(2,4,6)
  - autor(‘Jorge Amado’)

Questão Principal:

CONTEXTO

Estrutura X Predicado

- Objetos estruturados só podem ser termos de relações;
- Objetos estruturados não podem ser provados ou explicitados como fatos;
- Exemplo:
  - ama(pai(ricardo),ricardo). V
  - pai(ricardo). X

Cláusulas

- Prolog é composto por cláusulas:
  - Conjunto de fatos
    - Declarações de relações simples
  - Conjunto de regras
    - Implicações lógicas
- Possuem valor-verdade associado e, por isso, não podem ser exclusivamente objetos estruturados!
  - Mas podem ser compostos por eles!

Regras

- Regras (para aplicação lógica):

  Antecedente → Conseqüente

- Em Prolog

  Conseqüente :- Antecedente.

Cabeça **Corpo**

- Cabeça: uma relação SIMPLES (atômica)
- Corpo: uma composição de relações (conjunções, disjunções, negações)

Fatos

- É uma cláusula verdadeira
- Um fato pode ser considerado uma regra com corpo vazio

“Cabeça :- .”

relação(t1,t2,t3,...,tn).

ou seja, só possui cabeça.

Operadores

- Conjunção (^) : separa metas com vírgulas (,), de forma que todas as metas devam ser satisfeitas para que uma cláusula tenha sucesso.
- Negação(¬): é aplicado por meio da função not(P), em que P é uma meta.
- Disjunção(˅) : separa metas com ponto e vírgula (;), de forma que todas as metas de um dos lados da disjunção tenham que ser satisfeitas para ter sucesso.
  - Exemplo:
    - rápido(X) :- ágil(X); veloz(X), not(lento(X)).

Exemplo: conjunto de fatos

```
% base de fatos família real
pai_de(henrique_pai, henrique).          %1
pai_de(henrique_pai, maria).             %2
pai_de(henrique, elizabeth2).            %3
pai_de(henrique, eduardo).               %4
homem(henrique_pai).                     %5
homem(henrique).                         %6
homem(eduardo).                          %7
mulher(catarina).                        %8
mulher(elizabeth1).                      %9
mulher(maria).                           %10
mulher(elizabeth2).                      %11
mulher(ana).                             %12
mulher(jane).                            %13
mae_de(catarina, maria).                 %14
mae_de(ana, elizabeth2).                 %15
mae_de(jane, eduardo).                   %16
mae_de(elizabeth1, henrique).            %17
```

Exemplo : conjunto de regras

%base de regras família real

```
filho_de(Y,X) :- pai_de(X,Y), homem(Y).        %18
pai_ou_mae(X,Y) :- pai_de(X,Y).                %19
pai_ou_mae(X,Y) :- mae_de(X,Y).                %20
predecessor(X,Y) :- pai_de(X,Y).               %21
predecessor(X,Y) :- pai_de(X,Z),
predecessor(Z,Y).                               %22
```

Unificação

- Unificação (ou *matching*) é um das mais importantes operações do Prolog
- Dado dois termos, eles unificam se:
  - São idênticos, ou
  - as variáveis em ambos os termos podem ser instanciadas em objetos, de maneira que após a substituição as variáveis por esses objetos, os termos se tornam idênticos.

Unificação

- As regras que regem se dois termos **S** e **T** unificam são:
  - se **S** e **T** são constantes,então **S** e **T** unificam se, e só se são o mesmo objeto.
  - se **S** for uma variável e **T** for qualquer termo, então unificam.
  - Se **S** e **T** são estruturas, elas unificam se e só se **S** e **T** têm o mesmo funtor e todos elementos correspondentes unificam

Exemplos Unificação

Termo 1	**Termo 2**	**Resultado da Unificação**
henrique	henrique	unificam
eduardo	henrique	não unificam
X	par(a,b)	X = par(a,b)
2.35	Y	Y = 2.35
data(25,maio,Ano)	data(D,maio,1983)	D = 25 Ano = 1983
data(D1,abril,A)	data(D2,M,1900)	D1 = D2 M = abril A = 1900
data(17,marco,2000)	date(17,M,2000)	não unificam
pai_de(X,eduardo)	pai_de(henrique,Y)	X = henrique Y = eduardo

Unificação

- Unificação consiste aplicar uma substituição lógica (θ) de uma variável por um valor na cláusula a fim de que se torne equivalente a outra (e possa ser unificada).
  - Em Prolog, é o mais próximo que temos de atribuição.
- Unificação entre valores ou variáveis simples : =
  - Exemplo: ?- 4 = 4. true
- Igualdade entre valores simples ou variáveis que já possuam valores : ==
  - Exemplo: ?- Y = 2, X = 3, X == Y. false.
- Diferença: \= (possui versões com “<>”)
  - Exemplo: ?- 2 \= 5. true.

Unificação

- Comparação de valores aritméticos:
  - Atribuição de resultado aritmético à variável ou valor simples: is
    - Exemplo: ?- 14 is 2*7. true.
    - Contra-exemplo: ?- 14 = 2*7. false.
  - Quando os dois lados são resultados aritméticos (inclusive tipos diferentes): =:=
    - Exemplo: ?- 2*9 =:= 3*6. true.
    - Contra-exemplo: ?- 2*9 is 3*6. false.

Operadores

- Relacionais:
  - <, >, =<, >=
  - Exemplo: positivo(N) :- N > 0.
    - ?- positivo(4).
    - True.
- Aritméticos
  - +, -, * , /, ** (potência), // (divisão inteira), mod (resto).
  - Exemplo:
    - ? – 5 * 5 > 5 ** 2.
    - false

Consultas

- São o meio de recuperar ou obter nova informação a partir de um conjunto de cláusulas e inferência.
- Podem ser de dois tipos:
  - Confirmação
    - Quando se quer confirmar uma informação
  - Recuperação
    - Quando se quer saber quais valores satisfazem a consulta

Exemplo

- Consulta ou recuperação?

```
| ?- pai_de(henrique,eduardo).
yes
| ?- pai_de(X,maria).
X = henrique_pai ;
no
| ?- pai_de(henrique,X).
X = elizabeth2 ;
X = eduardo
| ?- pai_de(X,eduardo).
X = henrique
| ?- mae_de(X,henrique).
X = elizabeth1
| ?- mae_de(X,maria).
X = catarina;
no
```

```
% base família real
pai_de(henrique_pai, henrique).      %1
pai_de(henrique_pai, maria).         %2
pai_de(henrique, elizabeth2).        %3
pai_de(henrique, eduardo).           %4
homem(henrique_pai).                 %5
homem(henrique).                     %6
homem(eduardo).                      %7
mulher(catarina).                    %8
mulher(elizabeth1).                  %9
mulher(maria).                       %10
mulher(elizabeth2).                  %11
mulher(ana).                         %12
mulher(jane).                        %13
mae_de(catarina, maria).             %14
mae_de(ana, elizabeth2).             %15
mae_de(jane, eduardo).               %16
mae_de(elizabeth1, henrique).        %17
filho_de(Y,X) :- pai_de(X,Y),
homem(Y).                            %18
pai_ou_mae(X,Y) :- pai_de(X,Y).      %19
pai_ou_mae(X,Y) :- mae_de(X,Y).      %20
predecessor(X,Y) :- pai_de(X,Y).     %21
predecessor(X,Y) :- pai_de(X,Z),
predecessor(Z,Y).                    %22
```

Exemplo

```
| ?- mae_de(catarina,X).
X = maria
| ?- mae_de(X,catarina).
no
| ?-mae_de(X,Y).
X = catarina ,
Y = maria ;
X = ana ,
Y = elizabeth2 ;
X = jane ,
Y = eduardo ;
X = elizabeth1 ,
Y = henrique
```

```
% base família real
pai_de(henrique_pai, henrique).      %1
pai_de(henrique_pai, maria).         %2
pai_de(henrique, elizabeth2).        %3
pai_de(henrique, eduardo).           %4
homem(henrique_pai).                 %5
homem(henrique).                     %6
homem(eduardo).                      %7
mulher(catarina).                    %8
mulher(elizabeth1).                  %9
mulher(maria).                       %10
mulher(elizabeth2).                  %11
mulher(ana).                         %12
mulher(jane).                        %13
mae_de(catarina, maria).             %14
mae_de(ana, elizabeth2).             %15
mae_de(jane, eduardo).               %16
mae_de(elizabeth1, henrique).        %17
filho_de(Y,X) :- pai_de(X,Y),
homem(Y).                            %18
pai_ou_mae(X,Y) :- pai_de(X,Y).      %19
pai_ou_mae(X,Y) :- mae_de(X,Y).      %20
predecessor(X,Y) :- pai_de(X,Y).     %21
predecessor(X,Y) :- pai_de(X,Z),
predecessor(Z,Y).                    %22
```

Consulta composta

```
| ?- pai_de(X,elizabeth2),pai_de(X,eduardo).
X = henrique;
no
| ?- pai_de(X,eduardo),pai_de(Y,X).
(Quem é o avô de eduardo?)
X = henrique ,
Y = henrique_pai ;
no
| ?- pai_de(henrique_pai,X),pai_de(X,Y).
(Quem são os netos de henrique_pai?)
X = henrique ,
Y = elizabeth2 ;
X = henrique ,
Y = eduardo ;
no
```

```
% base família real
pai_de(henrique_pai, henrique).      %1
pai_de(henrique_pai, maria).         %2
pai_de(henrique, elizabeth2).        %3
pai_de(henrique, eduardo).           %4
homem(henrique_pai).                 %5
homem(henrique).                     %6
homem(eduardo).                      %7
mulher(catarina).                    %8
mulher(elizabeth1).                  %9
mulher(maria).                       %10
mulher(elizabeth2).                  %11
mulher(ana).                         %12
mulher(jane).                        %13
mae_de(catarina, maria).             %14
mae_de(ana, elizabeth2).             %15
mae_de(jane, eduardo).               %16
mae_de(elizabeth1, henrique).        %17
filho_de(Y,X) :- pai_de(X,Y),
homem(Y).                            %18
pai_ou_mae(X,Y) :- pai_de(X,Y).      %19
pai_ou_mae(X,Y) :- mae_de(X,Y).      %20
predecessor(X,Y) :- pai_de(X,Y).     %21
predecessor(X,Y) :- pai_de(X,Z),
predecessor(Z,Y).                    %22
```

Consulta inferência

Colocada a consulta:

| ?- filho_de(eduardo,henrique).

Não há no programa fatos sobre a relação filho_de.

É necessário usar as regras

A consulta é comparada com a cabeça das regras que definem a relação filho_de, na seqüência.

Ocorre uma unificação entre a consulta e a cabeça da regra %18, com as instanciações:

Y = eduardo

X = henrique

```
% base família real
pai_de(henrique_pai, henrique).      %1
pai_de(henrique_pai, maria).         %2
pai_de(henrique, elizabeth2).        %3
pai_de(henrique, eduardo).           %4
homem(henrique_pai).                 %5
homem(henrique).                     %6
homem(eduardo).                      %7
mulher(catarina).                    %8
mulher(elizabeth1).                  %9
mulher(maria).                       %10
mulher(elizabeth2).                  %11
mulher(ana).                         %12
mulher(jane).                        %13
mae_de(catarina, maria).             %14
mae_de(ana, elizabeth2).             %15
mae_de(jane, eduardo).               %16
mae_de(elizabeth1, henrique).        %17
filho_de(Y,X) :- pai_de(X,Y),
homem(Y).                            %18
pai_ou_mae(X,Y) :- pai_de(X,Y).      %19
pai_ou_mae(X,Y) :- mae_de(X,Y).      %20
predecessor(X,Y) :- pai_de(X,Y).     %21
predecessor(X,Y) :- pai_de(X,Z),
predecessor(Z,Y).                    %22
```

Consulta inferência

A substituída depois de unificada fica:
filho_de(eduardo,henrique):-
pai_de(henrique,eduardo),
homem(eduardo)

O objetivo é substituído pelos
sub-objetivos:
pai_de(henrique,eduardo),
homem(eduardo)
que devem ser verdadeiros.

Os sub-objetivos são fatos no programa.
Logo, o objetivo também é verdadeiro
e a resposta é: yes.

```
% base família real
pai_de(henrique_pai, henrique).      %1
pai_de(henrique_pai, maria).         %2
pai_de(henrique, elizabeth2).        %3
pai_de(henrique, eduardo).           %4
homem(henrique_pai).                 %5
homem(henrique).                     %6
homem(eduardo).                      %7
mulher(catarina).                    %8
mulher(elizabeth1).                  %9
mulher(maria).                       %10
mulher(elizabeth2).                  %11
mulher(ana).                         %12
mulher(jane).                        %13
mae_de(catarina, maria).             %14
mae_de(ana, elizabeth2).             %15
mae_de(jane, eduardo).               %16
mae_de(elizabeth1, henrique).        %17
filho_de(Y,X) :- pai_de(X,Y),
homem(Y).                            %18
pai_ou_mae(X,Y) :- pai_de(X,Y).      %19
pai_ou_mae(X,Y) :- mae_de(X,Y).      %20
predecessor(X,Y) :- pai_de(X,Y).     %21
predecessor(X,Y) :- pai_de(X,Z),
predecessor(Z,Y).                    %22
```

Regras e Disjunção

- Duas ou mais regras sobre a mesma relação indicam formas alternativas de provar um objetivo, portanto correspondem ao operador “ou”.

- Exemplo:
  - Considerando as regras 19 e 20
  - É equivalente a utilizar “;”

```
| ?- pai_ou_mae(X,elizabeth2).
X = henrique ;
X = ana ;
no
```

```
% base família real
pai_de(henrique_pai, henrique).      %1
pai_de(henrique_pai, maria).         %2
pai_de(henrique, elizabeth2).        %3
pai_de(henrique, eduardo).           %4
homem(henrique_pai).                 %5
homem(henrique).                     %6
homem(eduardo).                      %7
mulher(catarina).                    %8
mulher(elizabeth1).                  %9
mulher(maria).                       %10
mulher(elizabeth2).                  %11
mulher(ana).                         %12
mulher(jane).                        %13
mae_de(catarina, maria).             %14
mae_de(ana, elizabeth2).             %15
mae_de(jane, eduardo).               %16
mae_de(elizabeth1, henrique).        %17
filho_de(Y,X) :- pai_de(X,Y),
homem(Y).                            %18
pai_ou_mae(X,Y) :- pai_de(X,Y).      %19
pai_ou_mae(X,Y) :- mae_de(X,Y).      %20
predecessor(X,Y) :- pai_de(X,Y).     %21
predecessor(X,Y) :- pai_de(X,Z),
predecessor(Z,Y).                    %22
```

Regras e Disjunção

```
?- pai_ou_mae(X,eduardo).
X = henrique ;
X = jane ;
no

| ?- pai_ou_mae(jane,X).
X = eduardo

| ?- pai_ou_mae(henrique,Y).
Y = elizabeth2 ;
Y = eduardo ;
no
```

```
% base família real
pai_de(henrique_pai, henrique).      %1
pai_de(henrique_pai, maria).         %2
pai_de(henrique, elizabeth2).        %3
pai_de(henrique, eduardo).           %4
homem(henrique_pai).                 %5
homem(henrique).                     %6
homem(eduardo).                      %7
mulher(catarina).                    %8
mulher(elizabeth1).                  %9
mulher(maria).                       %10
mulher(elizabeth2).                  %11
mulher(ana).                         %12
mulher(jane).                        %13
mae_de(catarina, maria).             %14
mae_de(ana, elizabeth2).             %15
mae_de(jane, eduardo).               %16
mae_de(elizabeth1, henrique).        %17
filho_de(Y,X) :- pai_de(X,Y),
homem(Y).                            %18
pai_ou_mae(X,Y) :- pai_de(X,Y).      %19
pai_ou_mae(X,Y) :- mae_de(X,Y).      %20
predecessor(X,Y) :- pai_de(X,Y).     %21
predecessor(X,Y) :- pai_de(X,Z),
predecessor(Z,Y).                    %22
```

Regras e recursão

- Recursão: operação em que um predicado é usado em sua própria definição.
- Prolog é intrinsecamente recursivo
  - Regras recursivas definem laços
  - Natural da linguagem
- Regras recursivas: definidas em termos delas mesmas.
  - Regra base (sempre antes!)
  - Regra recursiva

Regras Recursivas

```
| ?- predecessor(henrique_pai,X).
X = henrique ;
X = maria;
X = elizabeth2;
X = eduardo ;
no

| ?- predecessor(X,henrique).
X = henrique_pai ;
no

| ?- predecessor(X,eduardo).
X = henrique ;
X = henrique_pai ;
no
```

```
% base família real
pai_de(henrique_pai, henrique).      %1
pai_de(henrique_pai, maria).         %2
pai_de(henrique, elizabeth2).        %3
pai_de(henrique, eduardo).           %4
homem(henrique_pai).                 %5
homem(henrique).                     %6
homem(eduardo).                      %7
mulher(catarina).                    %8
mulher(elizabeth1).                  %9
mulher(maria).                       %10
mulher(elizabeth2).                  %11
mulher(ana).                         %12
mulher(jane).                        %13
mae_de(catarina, maria).             %14
mae_de(ana, elizabeth2).             %15
mae_de(jane, eduardo).               %16
mae_de(elizabeth1, henrique).        %17
filho_de(Y,X) :- pai_de(X,Y),
homem(Y).                            %18
pai_ou_mae(X,Y) :- pai_de(X,Y).      %19
pai_ou_mae(X,Y) :- mae_de(X,Y).      %20
predecessor(X,Y) :- pai_de(X,Y).     %21
predecessor(X,Y) :- pai_de(X,Z),
predecessor(Z,Y).                    %22
```

Consulta e Unificação

- Uma variável só pode ser unificada com um **único termo** em uma cláusula durante uma consulta!
  - A unificação é válida para aquela cláusula.
  - Não existem variáveis globais!
- Variáveis diferentes podem ser unificadas com o **mesmo termo** em uma consulta!
  - Inclusive podem ser unificadas entre si!

Consulta e Unificação

- Exemplo: Definir a relação de irmã

```
pai_ou_mae(tom,bob).
pai_ou_mae(tom,liz).
pai_ou_mae(bob,ana).
pai_ou_mae(bob,pat).
pai_ou_mae(pat,jim).
mulher(ana).
mulher(pat).
irma(X,Y) :- pai_ou_mae(Z,X), pai_ou_mae(Z,Y),
      mulher(X).
```

Consulta e Unificação

- Consulta:

| ?- irma(pat,X).

X = ana ;

X = pat ;

no

A segunda resposta significa Pat é irmã dela mesma.

O Prolog inclui essa resposta porque não há restrição de que X e Y devam ser diferentes.

Retrocesso (*Backtracking*)

- Prolog tenta satisfazer cláusulas meta por meta, segundo a ordem estabelecida
- Em alguns casos, uma unificação feita para satisfazer uma determinada meta pode não satisfazer outra
- Algumas metas podem gerar ciclos infinitos
- Isso faz com que seja necessário retornar o objetivo para metas anteriores e tentar outras unificações
- Esse processo é chamado de **retrocesso**

Exemplo de retrocesso

- Considere que a base de conhecimento contenha os seguintes fatos:

  gosta(joao, jazz).

  gosta(joao, marcia).

  gosta(joao, feijoada).

  gosta(marcia, joao).

  gosta(marcia, feijoada).

- O objetivo é descobrir o que ambos (João e Márcia) gostam:

  ? - gosta(joao,X), gosta(marcia, X).

Retrocesso

?- gosta(joao, X), gosta(marcia, X).

```
gosta(joao, jazz).
gosta(joao, marcia).
gosta(joao, feijoada).
gosta(marcia, joao).
gosta(marcia, feijoada).
```

- Unifica com gosta(joao,jazz).
- θ = {X/jazz}
- Tenta satisfazer o segundo predicado, verificando gosta(marcia,jazz).
- Falha porque não consegue provar gosta(marcia,jazz).
- Faz o retrocesso e tenta satisfazer gosta(joao,X) novamente, excluindo o objeto jazz.

Retrocesso

?- gosta(joao, X), gosta(marcia, X).

- Unifica com gosta(joao,marcia).
- θ = {X/marcia}
- Tenta satisfazer o 2º

  predicado verificando gosta(marcia,marcia).
- Falha.
- Faz o retrocesso e tenta satisfazer gosta(joao,X), excluindo os objetos jazz e marcia.

```
gosta(joao, jazz).
gosta(joao, marcia).
gosta(joao, feijoada).
gosta(marcia, joao).
gosta(marcia, feijoada).
```

Retrocesso

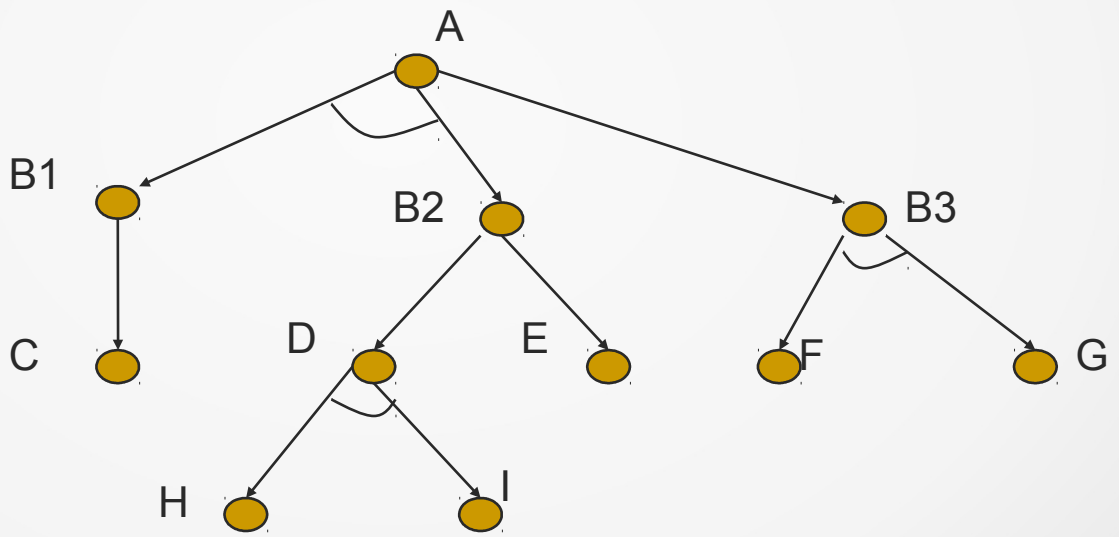
?- gosta(joao, X), gosta(marcia, X).

- Unifica gosta(joao,feijoada).
- {X/feijoada}
- Obtém sucesso com {X/feijoada}, pois gosta(marcia,feijoada).
- X = feijoada

```
gosta(joao, jazz).
gosta(joao, marcia).
gosta(joao, feijoada).
gosta(marcia, joao).
gosta(marcia, feijoada).
```

Árvore AND/OR

- A árvore AND/OR de um programa Prolog representa todos os caminhos que levam a uma solução, isto é, permitem provar o objetivo.
- A :- B1, B2.
- A :- B3.
- B1 :- C.
- B2 :- D.
- B2 :- E.
- B3 :- F, G.
- D :- H, I.

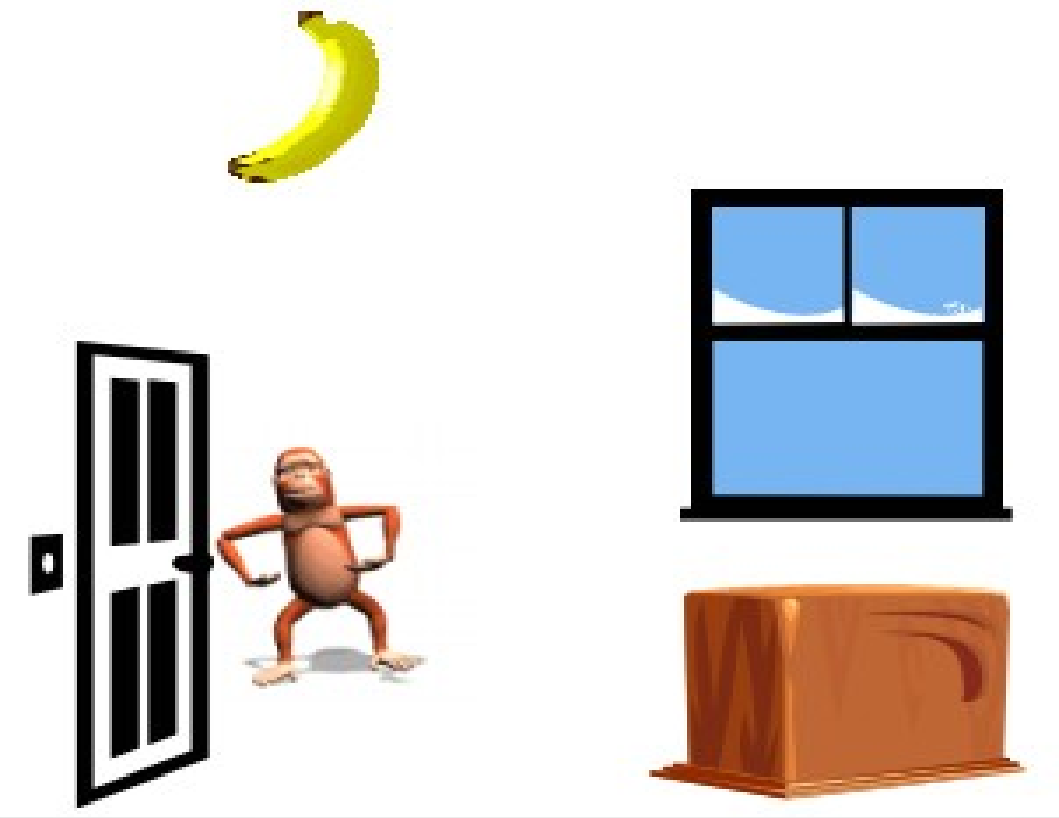


Árvore de execução

- Diferente da árvore AND/OR.
- Representa o processo de execução de uma consulta.
  - Nós representam objetivos a serem provados.
  - Arcos representam a troca de um objetivo por outro(s) por meio de unificação e inferência.
- Permite visualizar passo a passo a execução de uma consulta.

Exemplo árvore execução

- Macaco perto da porta quer comer a banana pendurada. Para isso, ele precisa arrastar a caixa que está perto da janela para perto da porta e subir nela.



Definindo o problema

- O cenário pode ser considerado um problema de estados
  - Cada estado possui informações que definem o cenário
  - Podemos definir um estado utilizando uma estrutura que combine quatro informações:
    - Posição horizontal do macaco (na_porta, na_janela)
    - Posição vertical do macaco (no_chao, acima_caixa)
    - Posição da caixa (na_porta, na_janela)
    - Se o macaco tem ou não tem a banana (tem, nao_tem)

Definindo o cenário

- O estado inicial do cenário é dado pela estrutura:

  estado(na_porta, no_chao, na_janela, nao_tem)
- O estado objetivo (final) do problema é:

  estado(_,_,_,tem)
- onde "_" significa "variável anônima"
  - É um tipo especial de variável
  - Pode ser unificada com qualquer valor, sobre qualquer circunstância.
  - Não é substituída.
  - Significa "tanto faz".

Mudança de estado

- Cada ação do macaco altera o cenário
  - Formalmente iremos modelar ações como:

    acao(*nome*, *estado1*, *estado2*)
- em que:
  - *nome* é nome da ação feita pelo agente (macaco)
  - *estado1* é o estado do cenário antes da ação
  - *estado2* é o estado do cenário depois da ação

Movimentação macaco

- Pode ser para a janela

  acao(mover_janela,

  estado(na_porta,no_chao,Caixa,Banana),

  estado(na_janela,no_chao,Caixa,Banana)).
- ou para a porta

  acao(mover_porta,

  estado(na_janela,no_chao,Caixa,Banana),

  estado(na_porta,no_chao,Caixa,Banana)).

Movimentação caixa

- O macaco pode empurrar a caixa para a porta

  acao(empurrar_porta,
  estado(na_janela,no_chao,na_janela,Banana),
  estado(na_porta,no_chao,na_porta,Banana)).
- ou pode empurrar para a janela

  acao(empurrar_janela,
  estado(na_porta,no_chao,na_porta,Banana),
  estado(na_janela,no_chao,na_janela,Banana)).

Movimentação caixa

- O macaco pode subir na caixa

  acao(subir,

  estado(Pos,no_chao,Pos,Banana),

  estado(Pos,acima_caixa,Pos,Banana)).
- e também pode pegar a banana

  acao(pegar_banana,

  estado(na_porta,acima_caixa,na_porta, nao_tem),

  estado(na_porta,acima_caixa,na_porta,tem)).

Relações entre estados

- É preciso estabelecer relações entre os estados e ações de mudança, a fim de criar um caminho.

  consegue(Estado2):- consegue(Estado1), acao(_,Estado1,Estado2).

- um estado é possível se existir uma ação capaz levar a ele partir de um estado possível.

  consegue(estado(na_porta,no_chao,na_janela, nao_tem)).

- É fato que o macaco está na porta, no chão e sem banana no início do problema, enquanto a caixa está perto da janela.

Fatos e Regras

- Agora temos o conjunto de fatos e regras do problema do macaco.
- Como é um modelo simples, precisamos definir a prioridade das regras a serem ativadas.
  - Ou podemos entrar em laços infinitos!
    - Discutidos na próxima aula!
- Ou seja, a ordem das regras é de extrema importância!

```
%Regras e fatos macaco

acao(pegar_banana,estado(na_porta,acima_caixa,na_porta,nao_tem),estado(na_porta,acima_caixa,na_porta,tem)).
acao(subir,estado(Pos, no_chao, Pos, Banana), estado(Pos, acima_caixa, Pos, Banana)).
acao(empurrar_janela,estado(na_porta,no_chao,na_porta,Banana), estado(na_janela, no_chao,na_janela,Banana)).
acao(empurrar_porta,estado(na_janela,no_chao,na_janela,Banana), estado(na_porta, no_chao,na_porta,Banana)).
acao(mover_janela,estado(na_porta,no_chao,Caixa,Banana), estado(na_janela, no_chao,Caixa,Banana)).
acao(mover_porta,estado(na_janela,no_chao,Caixa,Banana), estado(na_porta, no_chao,Caixa,Banana)).
consegue(estado(na_porta,no_chao,na_janela,nao_tem)).
consegue(Estado2):- consegue(Estado1), acao(_,Estado1,Estado2).
```

Fatos e Regras

- Ordem das regras:
  - 1) Pegar banana
  - 2) Subir na caixa
  - 3) Empurrar caixa
  - 4) Mover
- Regra recursiva:
  - 1) Caso base.
  - 2) Caso recursivo.

```
%Regras e fatos macaco

acao(pegar_banana,estado(na_porta,acim
a_caixa,na_porta,nao_tem),estado(na_port
a,acima_caixa,na_porta,tem)).
acao(subir,estado(Pos, no_chao, Pos,
Banana), estado(Pos, acima_caixa, Pos,
Banana)).
acao(empurrar_janela,estado(na_porta,no_
chao,na_porta,Banana), estado(na_janela,
no_chao,na_janela,Banana)).
acao(empurrar_porta,estado(na_janela,no_
chao,na_janela,Banana), estado(na_porta,
no_chao,na_porta,Banana)).
acao(mover_janela,estado(na_porta,no_ch
ao,Caixa,Banana), estado(na_janela,
no_chao,Caixa,Banana)).
acao(mover_porta,estado(na_janela,no_ch
ao,Caixa,Banana), estado(na_porta,
no_chao,Caixa,Banana)).
consegue(estado(na_porta,no_chao,na_jan
ela,nao_tem)).
consegue(Estado2):- consegue(Estado1),
acao(_,Estado1,Estado2).
```

Consulta

- Com a base no Prolog, hora de fazer a consulta!
  - Será que o macaco consegue pegar a banana?

    ?- consegue(estado(_,_,_,tem)).

    true .
  - E o macaco é bem sucedido ao pegar a banana!
- Mas e a árvore de execução?

Enumerar as regras

- Para poder ver quais regras são ativadas, vamos enumerar as regras
- Basta colocar um *write* na primeira posição após a implicação, com o número que indica a regra visitada
- Assim, podemos saber quais foram ativadas!

```
%Regras e fatos macaco

acao(pegar_banana,estado(na_porta,acim
a_caixa,na_porta,nao_tem),estado(na_port
a,acima_caixa,na_porta,tem)):-write('1 ').
acao(subir,estado(Pos,no_chao,Pos,Banan
a), estado(Pos, acima_caixa, Pos,
Banana)):-write('2 ').
acao(empurrar_janela,estado(na_porta,no_
chao,na_porta,Banana), estado(na_janela,
no_chao,na_janela,Banana)):-write('3 ').
acao(empurrar_porta,estado(na_janela,no_
chao,na_janela,Banana), estado(na_porta,
no_chao,na_porta,Banana)):-write('4 ').
acao(mover_janela,estado(na_porta,no_ch
ao,Caixa,Banana), estado(na_janela,
no_chao,Caixa,Banana)):-write('5 ').
acao(mover_porta,estado(na_janela,no_ch
ao,Caixa,Banana), estado(na_porta,
no_chao,Caixa,Banana)):-write('6 ').
consegue(estado(na_porta,no_chao,na_jan
ela,nao_tem)):-write('7 ').
consegue(Estado2):- write('8.1 '),
consegue(Estado1), write('8.2 '),
acao(_,Estado1,Estado2).
```

Consulta com caminho

- Agora podemos fazer a consulta e obter o caminho de execução da mesma!

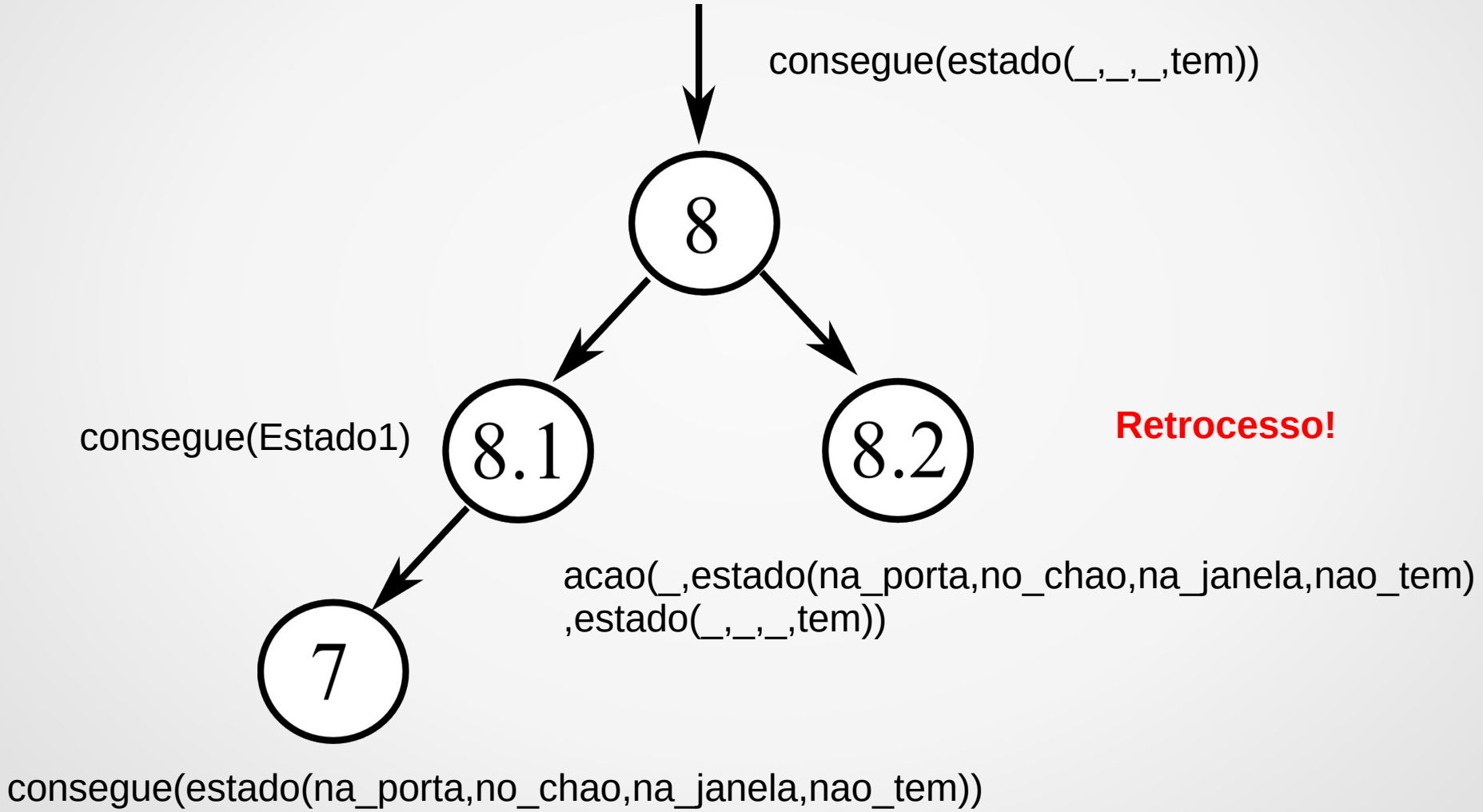
  ?-consegue(estado(_,_,_,tem)).

  8.1 7 8.2 8.1 7 8.2 5 8.2 8.1 7 8.2 5 8.2 2 8.2 4 8.2 6 8.2 8.1 7 8.2 5 8.2 2 8.2 4 8.2 2 8.2 1

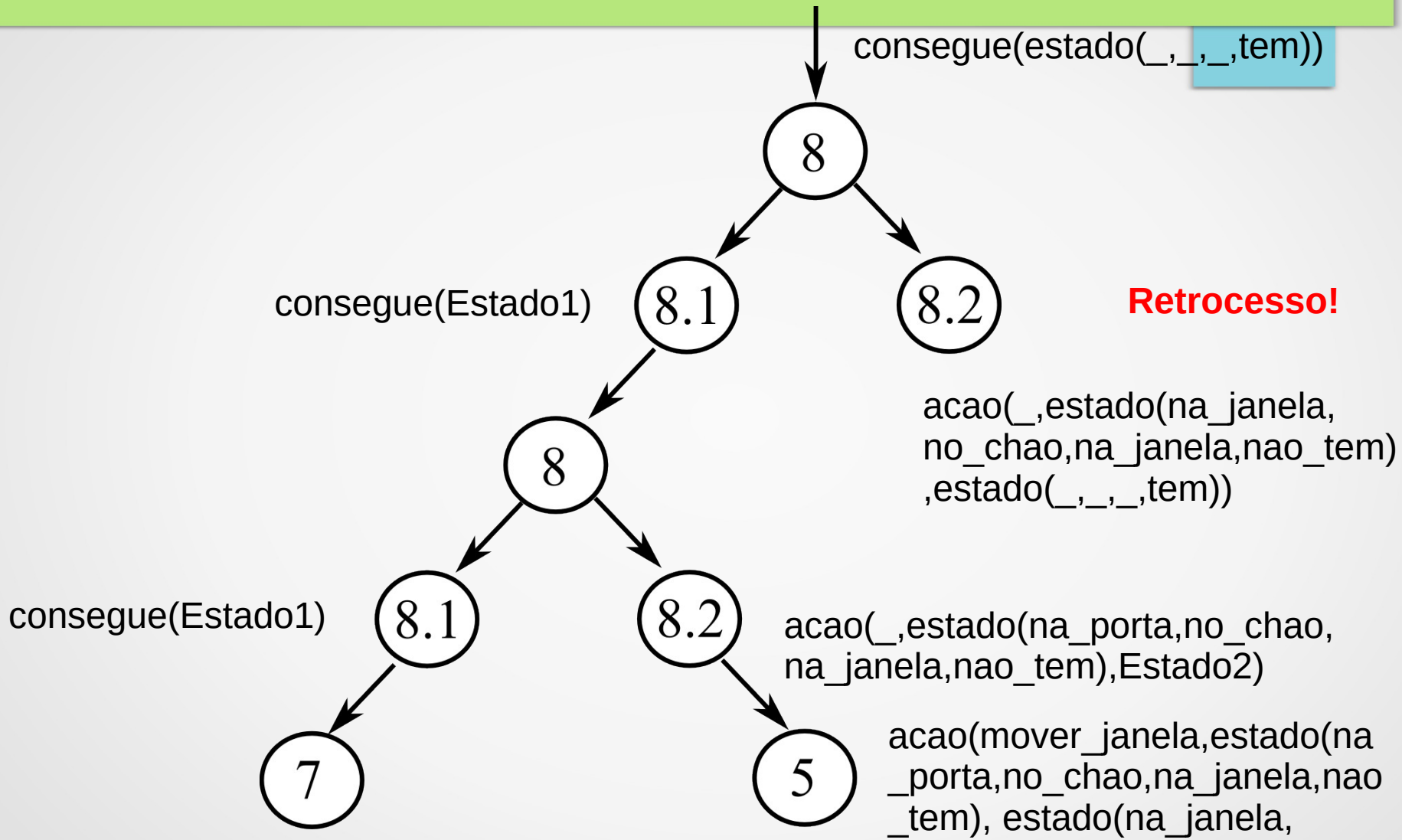
  true

- A sequência de números mostram quais regras foram ativadas!
  - Podemos obter a árvore de execução!

Árvore execução

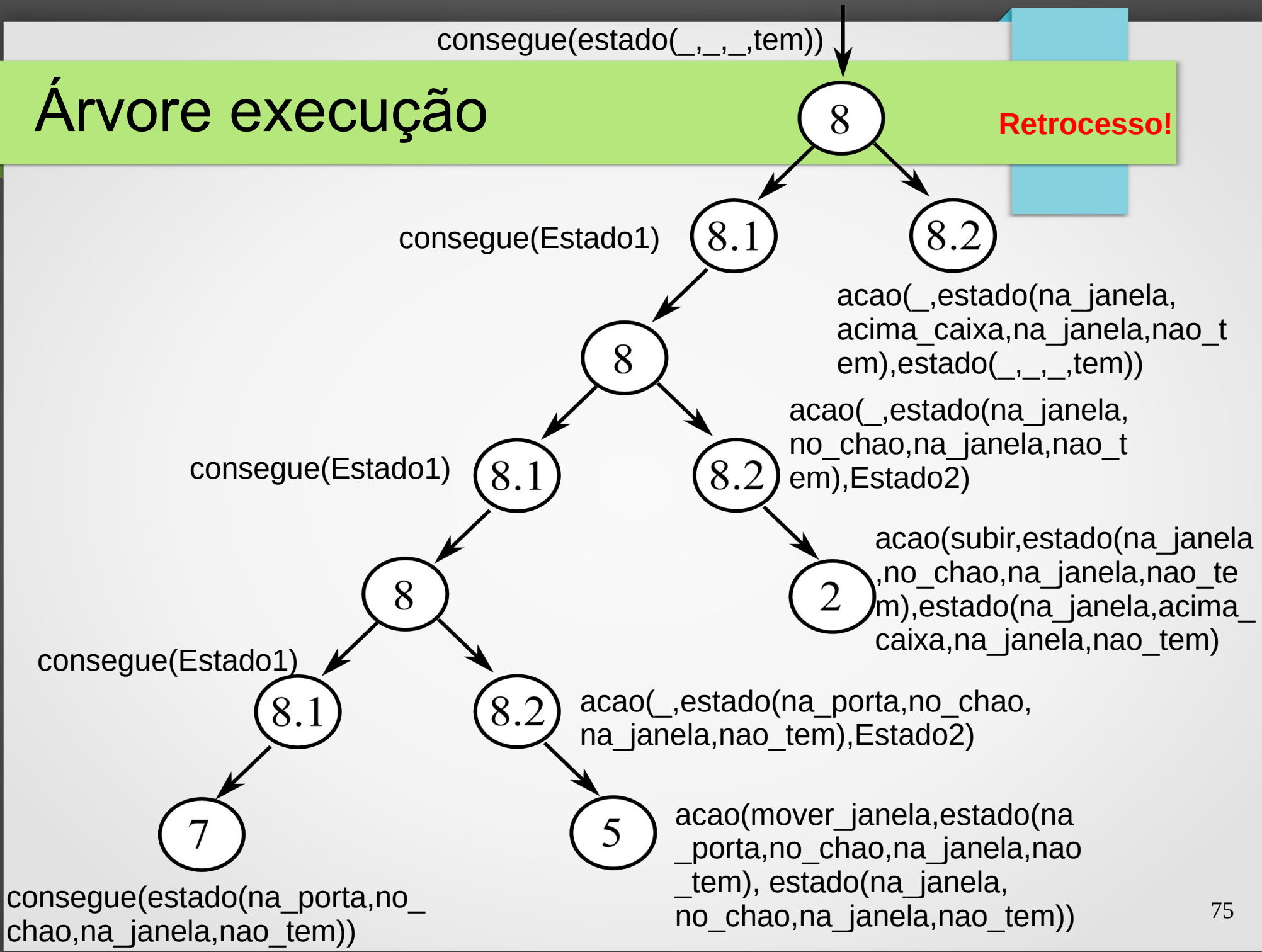


Árvore execução



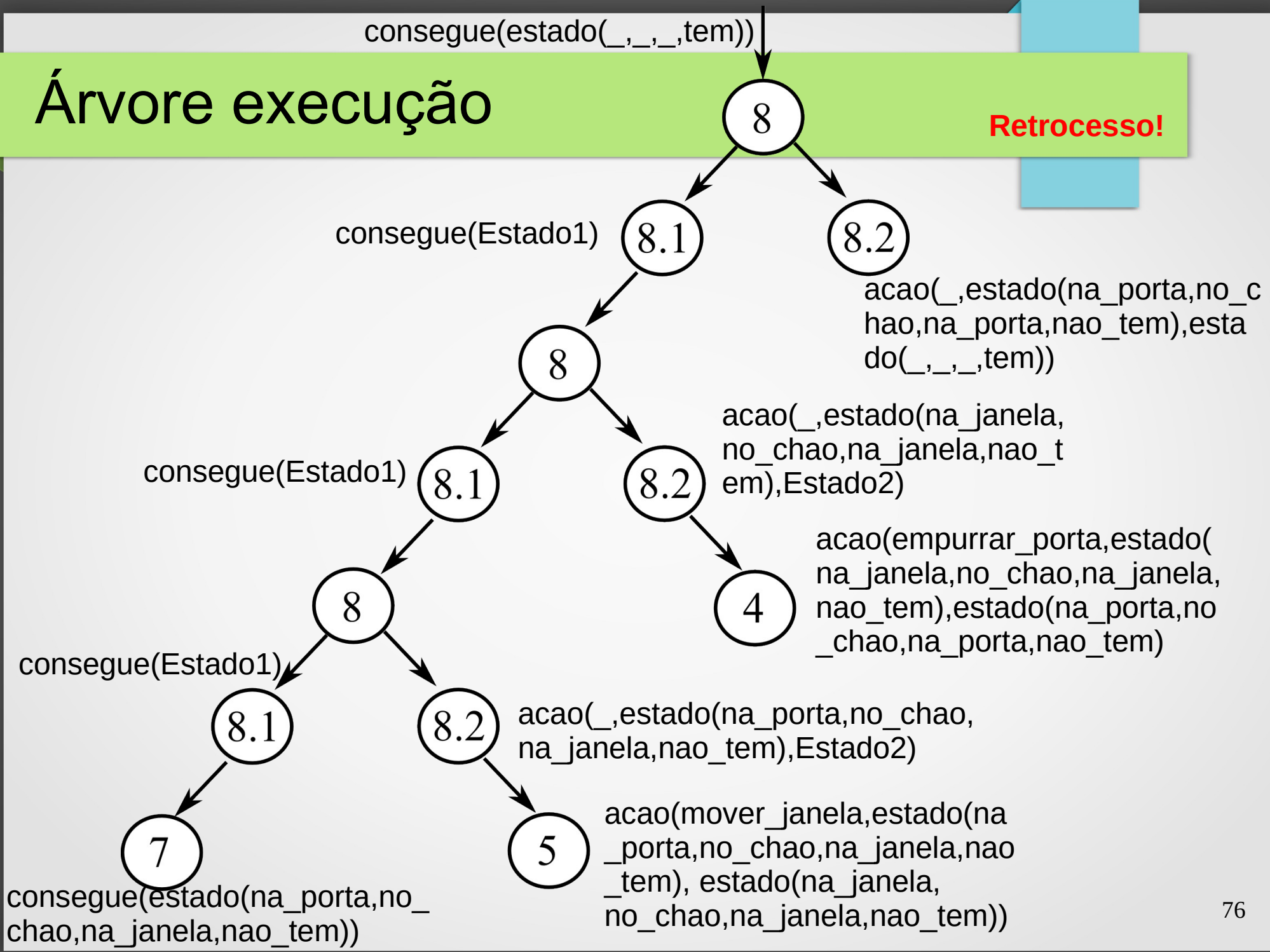
Árvore execução

Retrocesso!

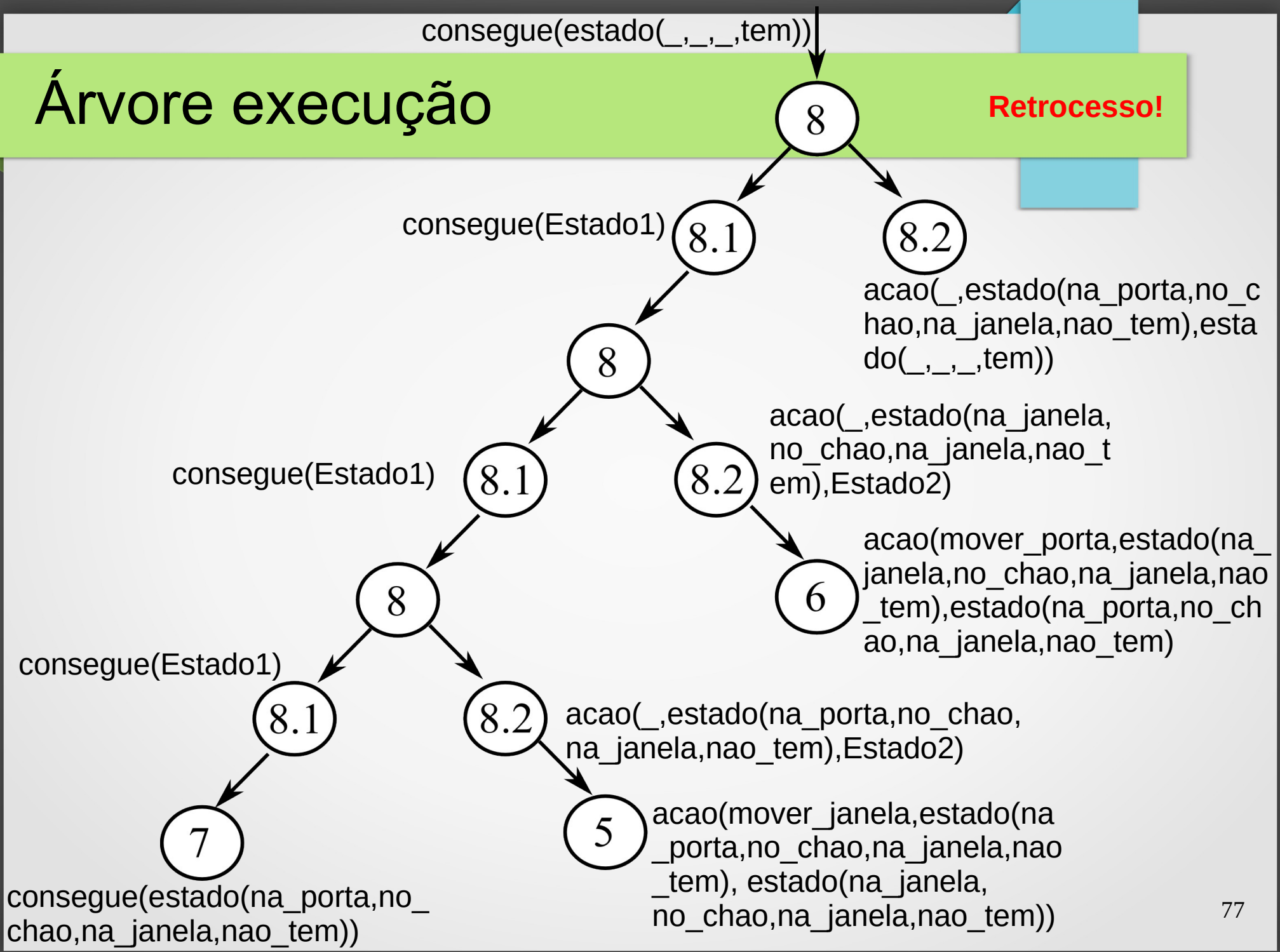


Árvore execução

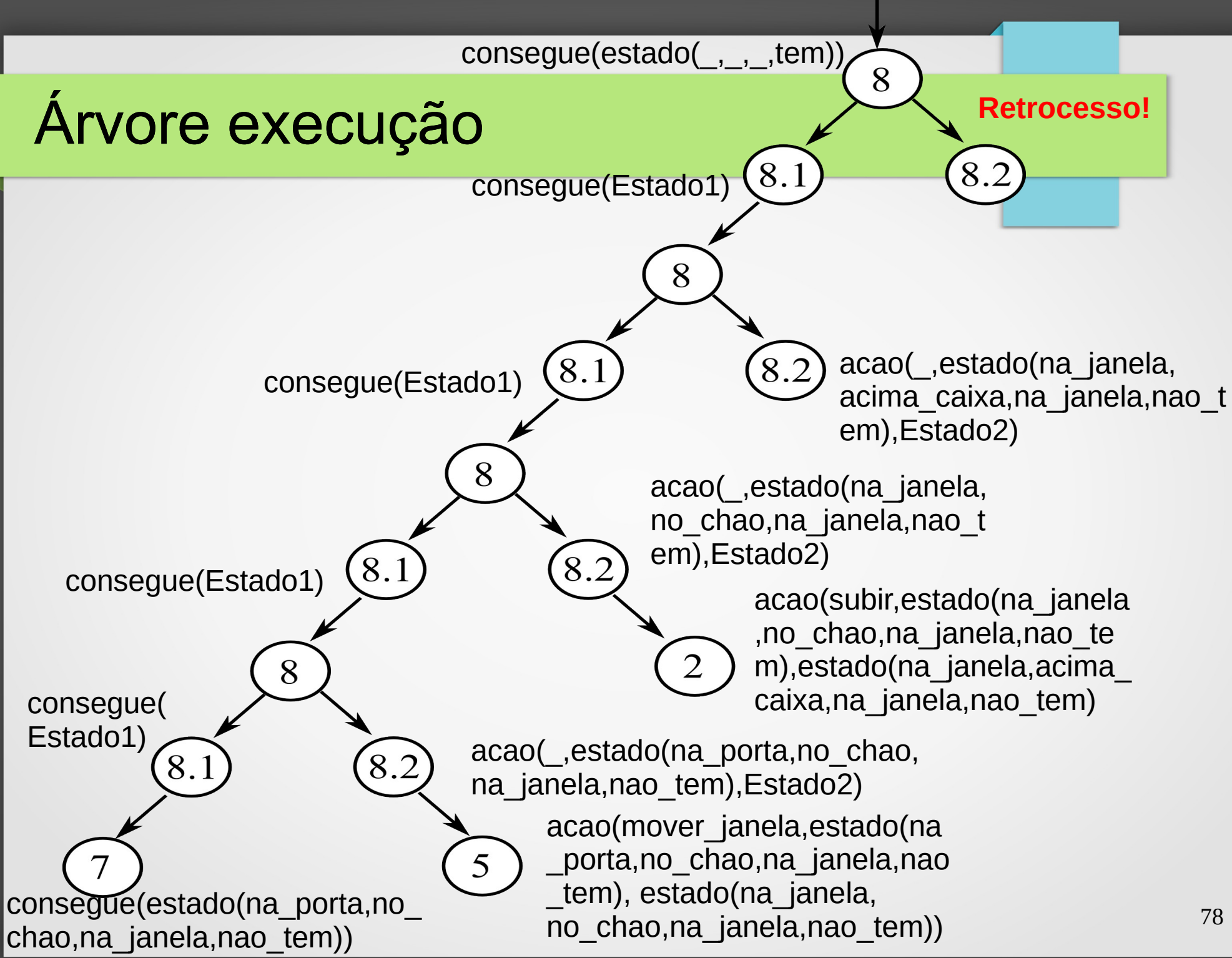
Retrocesso!



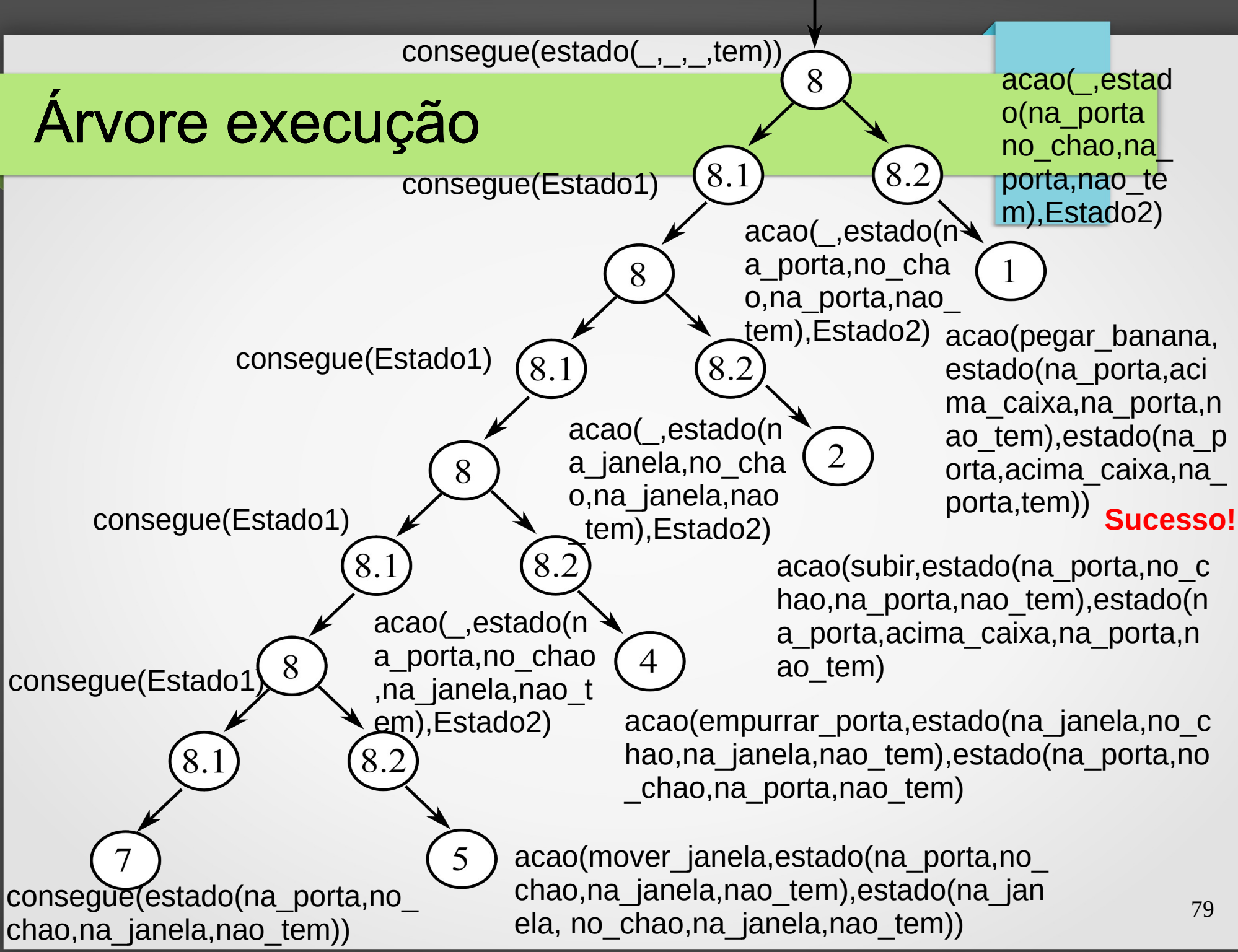
Árvore execução



Árvore execução

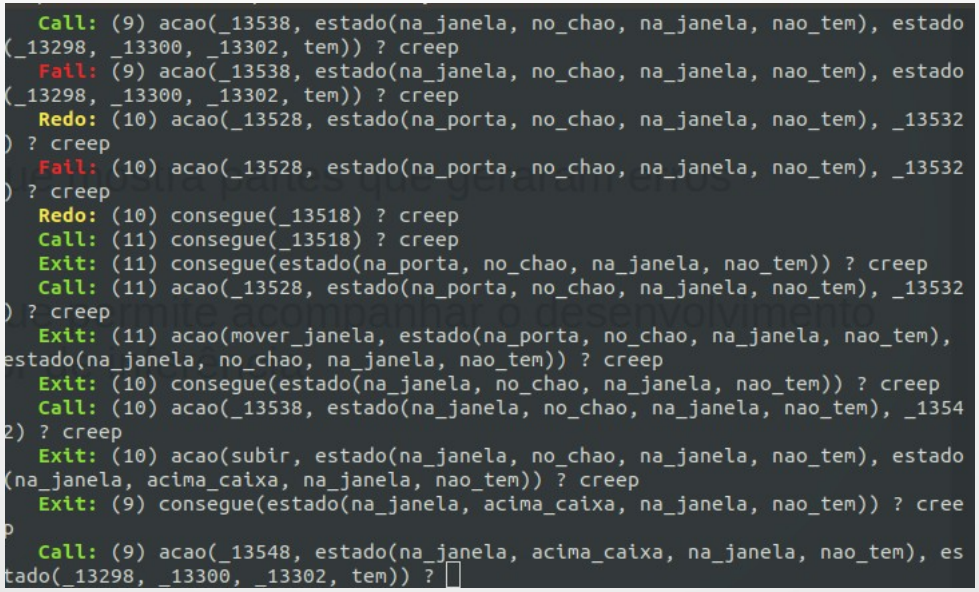


Árvore execução



Trace / Debug

- Debug
  - Modo que mostra partes que geraram erros
- Trace
  - Modo que permite acompanhar o desenvolvimento do motor de inferência



Exercício - Família

- Faça um programa em Prolog no qual seja possível fazer as seguintes consultas:
  - Quem é seu pai?
  - Quem é sua mãe?
  - Quem é seu irmão?
  - Quem é sua irmã?
  - Quem é seu avô?
  - Quem é sua avó?
  - Quem é filho de quem?

Exemplo

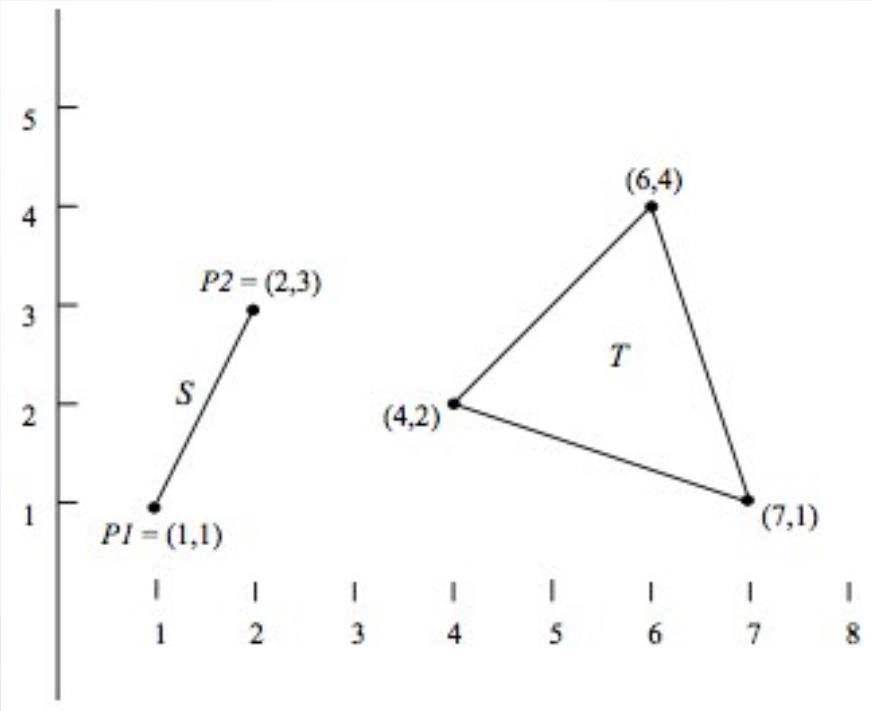
```
%FATOS
sexo(ricardo,m).
sexo(rodrigo,m).
sexo(irineu,m).
sexo(cleide,f).
sexo(dito,m).
sexo(tonha,f).
sexo(chico,m).
sexo(dirce,f).

genitor(irineu,ricardo).
genitor(cleide,ricardo).
genitor(irineu,rodrigo).
genitor(cleide,rodrigo).
genitor(tonha,irineu).
genitor(dito,irineu).
genitor(chico,cleide).
genitor(dirce,cleide).

%REGRAS
mae(X,Z) :- genitor(X,Z),sexo(X,f).
pai(X,Z) :- genitor(X,Z),sexo(X,m).
irmao(X,Z) :- pai(Y,X),pai(Y,Z),diferente(X,Z).
avo(X,Z) :- genitor(X,Y),genitor(Y,Z),sexo(X,m).
avoh(X,Z) :- genitor(X,Y),genitor(Y,Z),sexo(X,f).
filho(X,Z) :- genitor(Z,X),sexo(X,m).
filha(X,Z) :- genitor(Z,X),sexo(X,f).
filhos(X,Z) :- genitor(Z,X).
diferente(X,Z) :- X \= Z.
```

Exercício – Formas Geométricas

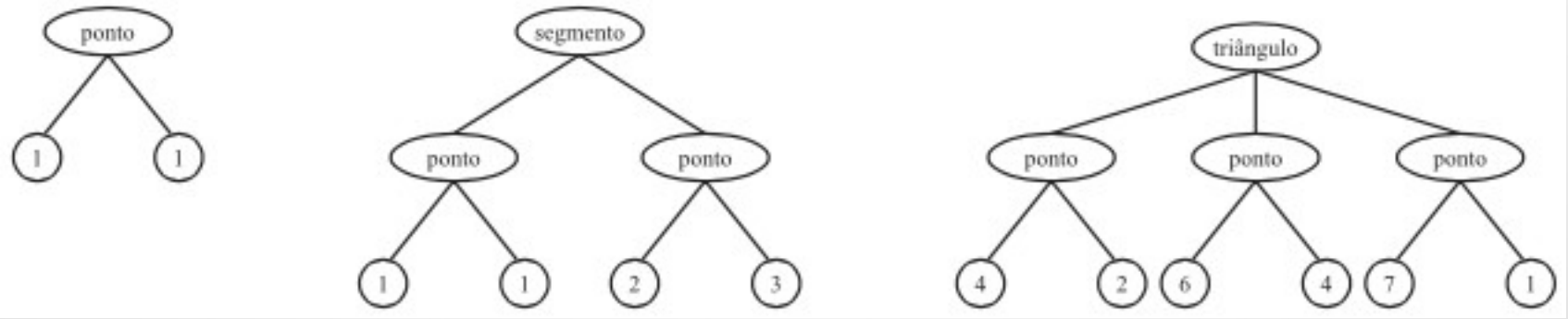
- Qualquer objeto estruturado pode ser representado por uma árvore. A raiz é o funtor e os filhos seus componentes. Se um componente é uma estrutura, ele é uma subárvore da árvore que corresponde a essa estrutura.



83

Exercício – Formas Geométricas

- Se usarmos os funtores **ponto** para pontos, **seg** para segmentos, e **triangulo** para triângulos:
- *P1 = ponto(1,1).*
- *P2 = ponto(2,3).*
- *S = seg(P1,P2) = seq(ponto(1,1), ponto(2,3))*
- *T = triangulo(ponto(4,2), ponto(6,4), ponto(7,1))*



Exercício – Formas Geométricas

- Se usarmos os funtores **ponto** para pontos, **seg** para segmentos, e **triangulo** para triângulos:
  - *P1 = ponto(1,1).*
  - *P2 = ponto(2,3).*
  - *S = seg(P1,P2) = seq(ponto(1,1), ponto(2,3))*
  - *T = triangulo(ponto(4,2), ponto(6,4), ponto(7,1))*
- (A) Represente círculos e polígonos regulares com quatro vértices?
- (B) Os termos abaixo descrevem qual família de triângulos?
- triangulo(ponto(-1,0),P2,P3) = triangulo(P1,ponto(1,0),ponto(0,Y))
- (C) Defina os fatos vertical e horizontal

Exercício – Formas Geométricas

- (A) Represente círculos e polígonos regulares com quatro vértices?
- (B) Os termos abaixo descrevem qual família de triângulos?
  - triangulo(ponto(-1,0),P2,P3) = triangulo(P1,ponto(1,0),ponto(0,Y))
- (C) Defina os fatos vertical e horizontal

```
%(A)
circulo(ponto(X,Y),Raio).
poligono(ponto(X1,Y1),ponto(X2,Y2),ponto(X3,Y3),ponto(X4,Y4)).

%(B)
%Triangulos isósceles

%(C)
ponto(X1,Y1).
ponto(X2,Y2).
seg(ponto(X1,Y1),ponto(X2,Y2)).
vertical(seg(ponto(X,Y1),ponto(X,Y2))).
horizontal(seg(ponto(X1,Y),ponto(X2,Y))).
```