DCC024 Linguagens de Programação

2023.1

Introdução a SML

Haniel Barbosa

ML

- ML foi criada por Robin Milner e outros no começo dos anos 70.
  - Meta-linguagem (ML) para LCF (*Logic for Computable Functions*)
- LCF foi um dos primeiros demonstradores de teoremas automatizados
  - Automatização de raciocínio lógico para demonstração de teoremas
  - Sistemas atuais que usam ML como meta-linguagem são HOL4 e Isabelle.
- ML possui muitos dialetos. Nós usaremos SML, *Standard ML*.
  - Puramente funcional
  - Possui uma especificação formal completa
  - Parte substancial da linguagem possui um compilador verificado formalmente: CakeML.

Interpretando SML

- Usaremos primariamente a linguagem através de um interpretador
  - Nos provê um ambiente de programação interativo
  - sml/nj usado em exemplos

```
- 4 + 3;
```

Interpretando SML

- Usaremos primariamente a linguagem através de um interpretador
  - Nos provê um ambiente de programação interativo
  - sml/nj usado em exemplos

```
- 4 + 3;

val it = 7 : int
```

Interpretando SML

- Usaremos primariamente a linguagem através de um interpretador
  - Nos provê um ambiente de programação interativo
  - sml/nj usado em exemplos

```
- 4 + 3;

val it = 7 : int
```

- Inferência de tipos

Interpretando SML

- Usaremos primariamente a linguagem através de um interpretador
  - Nos provê um ambiente de programação interativo
  - sml/nj usado em exemplos

```
- 4 + 3;

val it = 7 : int
```

- Inferência de tipos
- variável it carrega valor da última expressão avaliada

```
- it;
val it = 7 : int
```

Interpretando SML

- Usaremos primariamente a linguagem através de um interpretador
  - Nos provê um ambiente de programação interativo
  - sml/nj usado em exemplos

```
- 4 + 3;

val it = 7 : int
```

- Inferência de tipos
- variável it carrega valor da última expressão avaliada

```
- it;
val it = 7 : int
```

- Expressão termina com ;

Negação aritmética

```
- 4 - 3;
val it = 1 : int
```

Negação aritmética

```
- 4 - 3;
val it = 1 : int

- 4 + ~3;
val it = 1 : int
```

Negação aritmética

```
- 4 - 3;
val it = 1 : int

- 4 + ~3;
val it = 1 : int
```

- Em SML o “menos unário” é representado por ~

Negação aritmética

```
- 4 - 3;
val it = 1 : int

- 4 + ~3;
val it = 1 : int
```

- Em SML o “menos unário” é representado por ~
- Por que não usar -?

```
- 4 + -3;
```

Negação aritmética

```
- 4 - 3;
val it = 1 : int

- 4 + ~3;
val it = 1 : int
```

- Em SML o “menos unário” é representado por ~
- Por que não usar -?

  ```
  - 4 + -3;
  ```

- Não há sobrecarga do operador -. Ele e ~ possuem tipos diferentes.
- SML é uma linguagem fortemente tipada.

Expressões de tipos primitivos

```
- true;
val it = true : bool
- 1.4;
val it = 1.4 : real
- 14;
val it = 14 : int
- "dcc024";
val it = "dcc024" : string
```

Expressões de tipos primitivos

```
- true;
val it = true : bool
- 1.4;
val it = 1.4 : real
- 14;
val it = 14 : int
- "dcc024";
val it = "dcc024" : string

- #"a";
val it #"a" : char
```

Expressões de tipos primitivos

```
- ~ 1 + 2 - 3 * 4 div 5 mod 6;
```

Expressões de tipos primitivos

```
- ~ 1 + 2 - 3 * 4 div 5 mod 6;

val it = ~1: int
```

Expressões de tipos primitivos

```
- ~ 1 + 2 - 3 * 4 div 5 mod 6;
val it = ~1: int
- ~ 1.0 + 2.0 - 3.0 * 4.0 / 5.0;
```

Expressões de tipos primitivos

```
- ~ 1 + 2 - 3 * 4 div 5 mod 6;

val it = ~1: int

- ~ 1.0 + 2.0 - 3.0 * 4.0 / 5.0;

val it = ~1.4 : real
```

- Os resultados acima indicam a seguinte ordem de precedência:
  - ~ > *, / > div, mod > +, -

Expressões de tipos primitivos

```
- "bibity" ^ "bobity" ^ "boo";
val it = "bibitybobityboo" : string
- 2 < 3;
val it = true : bool
- 2.0 < 3.0;
```

Expressões de tipos primitivos

```
- "bibity" ^ "bobity" ^ "boo";
val it = "bibitybobityboo" : string
- 2 < 3;
val it = true : bool
- 2.0 < 3.0;

val it = true : bool
```

Expressões de tipos primitivos

```
- "bibity" ^ "bobity" ^ "boo";
val it = "bibitybobityboo" : string
- 2 < 3;
val it = true : bool
- 2.0 < 3.0;

val it = true : bool

- #"c" > #"d";
```

Expressões de tipos primitivos

```
- "bibity" ^ "bobity" ^ "boo";
val it = "bibitybobityboo" : string
- 2 < 3;
val it = true : bool
- 2.0 < 3.0;

val it = true : bool

- #"c" > #"d";

val it = false : bool
```

- Alguns operadores primitivos são sobrecarregados
- Caracteres comparados via comparação de inteiros com seus códigos ASCII

Expressões de tipos primitivos

```
- "bibity" ^ "bobity" ^ "boo";
val it = "bibitybobityboo" : string
- 2 < 3;
val it = true : bool
- 2.0 < 3.0;

val it = true : bool

- #"c" > #"d";

val it = false : bool
```

- Alguns operadores primitivos são sobrecarregados
- Caracteres comparados via comparação de inteiros com seus códigos ASCII

```
- "abce" >= "abd";
val it = false : bool
```

Expressões de tipos primitivos

```
- "bibity" ^ "bobity" ^ "boo";
val it = "bibitybobityboo" : string
- 2 < 3;
val it = true : bool
- 2.0 < 3.0;

val it = true : bool

- #"c" > #"d";

val it = false : bool
```

- Alguns operadores primitivos são sobrecarregados
- Caracteres comparados via comparação de inteiros com seus códigos ASCII

```
- "abce" >= "abd";
val it = false : bool
```

- Comparação de strings é lexicográfica com comparação de caracteres

Expressões de tipos primitivos

```
- 1 < 2 orelse 3 > 4;
val it = true : bool
- 1 < 2 andalso not (3 < 4);
val it = false : bool
```

Expressões de tipos primitivos

```
- 1 < 2 orelse 3 > 4;
val it = true : bool
- 1 < 2 andalso not (3 < 4);
val it = false : bool
```

- and é reservado para declaração de funções mutuamente recursivas
- or é indefinido

Expressões de tipos primitivos

```
- 1.3 = 1.3;
```

Expressões de tipos primitivos

```
- 1.3 = 1.3;
```

- Operação indefinida pois real não é um tipo com igualdade definida

```
- op =;
val it = fn : ''a * ''a -> bool
```

Expressões de tipos primitivos

```
- 1.3 = 1.3;
```

- Operação indefinida pois real não é um tipo com igualdade definida

```
- op =;
val it = fn : ''a * ''a -> bool
```

- "=" pode ser aplicado a quaisquer tipos para os quais igualdade é definida
- Por que real não é um desse tipos?

Expressões de tipos primitivos

```
- 1.3 = 1.3;
```

- Operação indefinida pois real não é um tipo com igualdade definida

```
- op =;
val it = fn : ''a * ''a -> bool
```

- "=" pode ser aplicado a quaisquer tipos para os quais igualdade é definida
- Por que real não é um desse tipos?
  - Representação de acordo com padrão IEEE 754
  - Propriedades diferentes daquelas de $\mathbb{R}$

Expressões de tipos primitivos

```
- 1.3 = 1.3;
```

- Operação indefinida pois real não é um tipo com igualdade definida

```
- op =;
val it = fn : ''a * ''a -> bool
```

- "=" pode ser aplicado a quaisquer tipos para os quais igualdade é definida
- Por que real não é um desse tipos?
  - Representação de acordo com padrão IEEE 754
  - Propriedades diferentes daquelas de $\mathbb{R}$
  - $0,1 + 0,2$ em IEEE 754 é $0,30000000000000004$
  - Existem valores a, b, c tais que $(a \times b) \times c \neq a \times (b \times c)$ em IEEE 754

Expressões de tipos primitivos

```
- 1.3 = 1.3;
```

- Operação indefinida pois real não é um tipo com igualdade definida

```
- op =;
val it = fn : ''a * ''a -> bool
```

- "=" pode ser aplicado a quaisquer tipos para os quais igualdade é definida
- Por que real não é um desse tipos?
  - Representação de acordo com padrão IEEE 754
  - Propriedades diferentes daquelas de $\mathbb{R}$
  - $0,1+0,2$ em IEEE 754 é $0,30000000000000004$
  - Existem valores a,b,c tais que $(a \times b) \times c \neq a \times (b \times c)$ em IEEE 754

```
- Real.==(0.1+0.2,0.3);
val it = false : bool
```

```
- Real.==(1.3,1.3);
val it = true : bool
```

Expressões de tipos primitivos

- Como vimos, SML tem sobrecarga de alguns operadores primitivos

```
- 1 * 2;
val it = 2 : int
- 1.0 * 2.0;
val it = 2.0 : real
```

Expressões de tipos primitivos

- Como vimos, SML tem sobrecarga de alguns operadores primitivos

```
- 1 * 2;
val it = 2 : int
- 1.0 * 2.0;
val it = 2.0 : real
```

- Não faz conversão implícita de tipos

```
- 10 / 5;
- 1.0 * 2;
```

Expressões de tipos primitivos

- Como vimos, SML tem sobrecarga de alguns operadores primitivos

```
- 1 * 2;
val it = 2 : int
- 1.0 * 2.0;
val it = 2.0 : real
```

- Não faz conversão implícita de tipos

```
- 10 / 5;
- 1.0 * 2;
```

- Existem conversões explícitas

```
- 1.0 * real(2);
val it = 2.0 : real

- floor(1.0)*2;
val it = 2 : int
```

Avaliação preguiçosa em SML

```
- 1 div 0;
```

Avaliação preguiçosa em SML

```
- 1 div 0;
```

- Gera uma exceção

```
- true orelse 1 div 0 = 0;
```

Avaliação preguiçosa em SML

```
- 1 div 0;
```

▷ Gera uma exceção

```
- true orelse 1 div 0 = 0;

val it = true : bool
```

Expressões condicionais em SML

```
- if 1 < 2 then #"x" else #"y";
val it = #"x" : char
- if 1 > 2 then 34 else 56;
val it = 56 : int
- (if 1 < 2 then 34 else 56) + 1;
val it = 35 : int
```

Expressões condicionais em SML

```
- if 1 < 2 then #"x" else #"y";
val it = #"x" : char
- if 1 > 2 then 34 else 56;
val it = 56 : int
- (if 1 < 2 then 34 else 56) + 1;
val it = 35 : int
```

- Todo if deve ter também o caso do else.

Expressões condicionais em SML

```
- if 1 < 2 then #"x" else #"y";
val it = #"x" : char
- if 1 > 2 then 34 else 56;
val it = 56 : int
- (if 1 < 2 then 34 else 56) + 1;
val it = 35 : int
```

- Todo if deve ter também o caso do else.
  - if ... then ... else ... é também uma expressão
  - Toda expressão deve ter um valor
  - Se um if não tivesse o caso do else seu valor poderia ser indefinido.

Vinculação de nomes a valores em SML

- Nós vimos que it é vinculado ao valor da última expressão avaliada
- Mas como criar nossas próprias vinculações?

```
- val x = 1+2*3;
val x = 7 : int
- x;
val it = 7 : int
```

Vinculação de nomes a valores em SML

- Nós vimos que it é vinculado ao valor da última expressão avaliada
- Mas como criar nossas próprias vinculações?

```
- val x = 1+2*3;
val x = 7 : int
- x;
val it = 7 : int
```

- Nomes podem ser vinculados a diferentes valores (operação destrutiva)

```
- val a = "123";
val a = "123" : string

- val a = 3 + 4;
val a = 7 : int
```

Tipos primitivos estruturados: Tuplas

- SML possui *tuplas* como tipos primitivos

```
- val barney = (1+2, 3.0*4.0, "brown");
val barney = (3,12.0,"brown") : int * real * string

- val point1 = ("red", (300,200));
val point1 = ("red",(300,200)) : string * (int * int)
```

Tipos primitivos estruturados: Tuplas

- SML possui *tuplas* como tipos primitivos

```
- val barney = (1+2, 3.0*4.0, "brown");
val barney = (3,12.0,"brown") : int * real * string

- val point1 = ("red", (300,200));
val point1 = ("red",(300,200)) : string * (int * int)
```

- O que é * no tipo acima?

Tipos primitivos estruturados: Tuplas

- SML possui *tuplas* como tipos primitivos

```
- val barney = (1+2, 3.0*4.0, "brown");
val barney = (3,12.0,"brown") : int * real * string

- val point1 = ("red", (300,200));
val point1 = ("red",(300,200)) : string * (int * int)
```

- O que é * no tipo acima?
  - Construtor de tipos de tuplas

Tipos primitivos estruturados: Tuplas

- SML possui *tuplas* como tipos primitivos

```
- val barney = (1+2, 3.0*4.0, "brown");
val barney = (3,12.0,"brown") : int * real * string

- val point1 = ("red", (300,200));
val point1 = ("red",(300,200)) : string * (int * int)
```

- O que é * no tipo acima?
  - Construtor de tipos de tuplas
- Há operadores primitivos para desconstruir tuplas

```
- #1 (#2 point1);
val it = 300 : int
- #1 (1, 2);
val it = 1 : int
- #1 (1);
```

Tipos primitivos estruturados: Listas

- SML também possui *listas* como tipos primitivos

```
- [1,2,3];
val it = [1,2,3] : int list
- [1.0,2.0];
val it = [1.0,2.0] : real list
- [(1,2),(1,3)];
```

Tipos primitivos estruturados: Listas

- SML também possui *listas* como tipos primitivos

```
- [1,2,3];
val it = [1,2,3] : int list
- [1.0,2.0];
val it = [1.0,2.0] : real list
- [(1,2),(1,3)];

val it = [(1,2),(1,3)] : (int * int) list
```

Tipos primitivos estruturados: Listas

- SML também possui *listas* como tipos primitivos

```
- [1,2,3];
val it = [1,2,3] : int list
- [1.0,2.0];
val it = [1.0,2.0] : real list
- [(1,2),(1,3)];

val it = [(1,2),(1,3)] : (int * int) list
```

- Qual a diferença entre listas e tuplas?

Tipos primitivos estruturados: Listas

- SML também possui *listas* como tipos primitivos

```
- [1,2,3];
val it = [1,2,3] : int list
- [1.0,2.0];
val it = [1.0,2.0] : real list
- [(1,2),(1,3)];

val it = [(1,2),(1,3)] : (int * int) list
```

- Qual a diferença entre listas e tuplas?
- O uso do colchetes é açúcar sintático para o construtor de listas ::

```
- val x = #"c"::[];
val x = [#"c"] : char list
- val y = #"b"::x;
val y = [#"b",#"c"] : char list
- val z = #"a"::y;
val z = [#"a",#"b",#"c"] : char list
```

Tipos primitivos estruturados: Listas

- Similarmente, existem desconstrutores primitivos

```
- hd;
val it = fn : 'a list -> 'a
- tl;
val it = fn : 'a list -> 'a list
```

```
- hd [1,2];
- tl [1,2];
```

Tipos primitivos estruturados: Listas

- Similarmente, existem desconstrutores primitivos

```
- hd;
val it = fn : 'a list -> 'a
- tl;
val it = fn : 'a list -> 'a list


- hd [1,2];
- tl [1,2];

val it = 1 : int
val it = [2] : int list
```

Tipos primitivos estruturados: Listas

- Há também uma operação primitiva para concatenação

```
- [1,2,3]@[4,5,6];
val it = [1,2,3,4,5,6] : int list
```

Tipos primitivos estruturados: Listas

- Há também uma operação primitiva para concatenação

```
- [1,2,3]@[4,5,6];
val it = [1,2,3,4,5,6] : int list
```

- Qual a diferença de complexidade entre :: e @?

Tipos primitivos estruturados: Listas

- Há também uma operação primitiva para concatenação

```
- [1,2,3]@[4,5,6];
val it = [1,2,3,4,5,6] : int list
```

- Qual a diferença de complexidade entre :: e @?

- Há também operações para verificar se uma lista é vazia

```
- [];
val it = [] : 'a list
- nil;
val it = [] : 'a list

- null [];
val it = true : bool
- null [1,2,3];
val it = false : bool
```

Funções em SML

- Construímos programas mais expressivos em SML através de funções

```
- fun square x = x * x;
val square = fn : int -> int
- square 2+1;
val it = 5 : int
```

Funções em SML

- Construímos programas mais expressivos em SML através de funções

```
- fun square x = x * x;
val square = fn : int -> int
- square 2+1;
val it = 5 : int
```

- Funções em SML tomam exatamente um argumento
- Aridades maiores podem ser simuladas por exemplo com tuplas

```
- fun quot(a,b) = a div b;
val quot = fn : int * int -> int
- quot (6,2);
val it = 3 : int
- val pair = (6,2);
val pair = (6,2) : int * int
- quot pair;
val it = 3 : int
```