

Seminário em F*

Daniella Vasconcellos
e Laís Van Vossen
MFO

Cristiano Damiani Vasconcellos
26/07/2023

Introdução

O F*, também conhecido como FStar (F Estrela) é uma linguagem de programação e assistente de provas baseada em tipos dependentes, mas que também desempenha outros papéis:

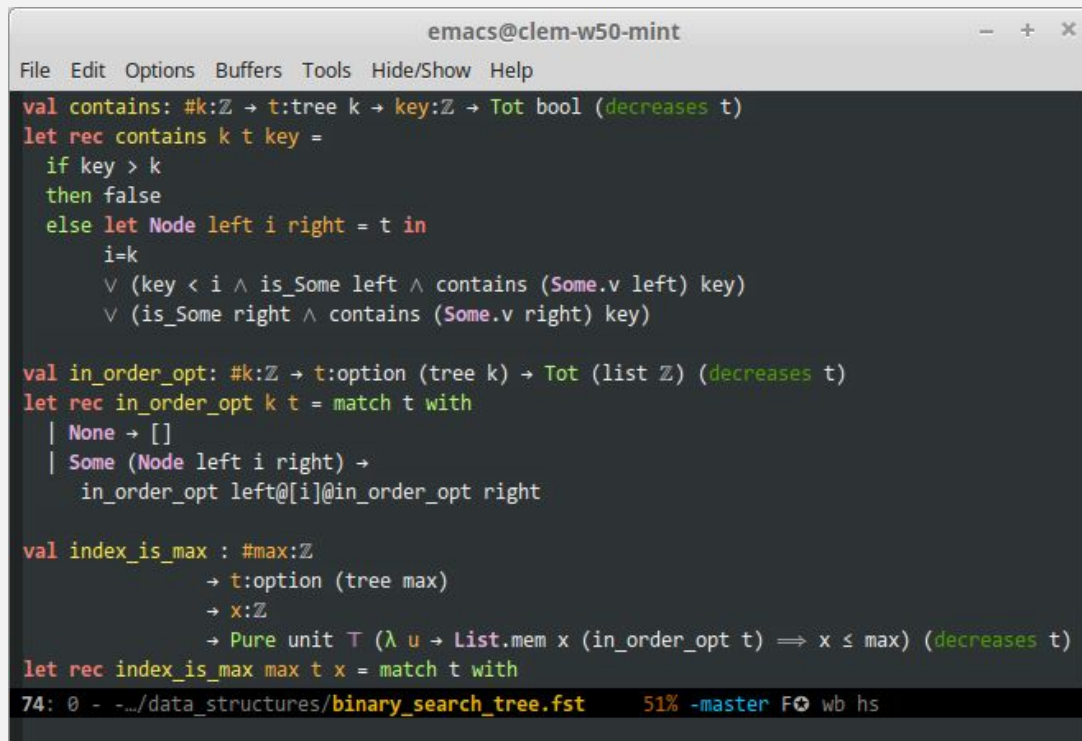
1. Uma linguagem de programação de propósito geral, que incentiva a programação funcional de ordem superior com efeitos;
2. Um compilador, que traduz programas F* para OCaml, F#, C ou Wasm;
3. Um assistente de provas;
4. Um mecanismo de verificação de programas, utilizando solucionadores SMT para automatizar provas de programas;
5. Um sistema de metaprogramação, que suporta procedimentos de automação de provas.

Histórico

- O F* é um projeto de **código aberto no GitHub** em desenvolvimento por pesquisadores de várias instituições, incluindo a Microsoft Research, MSR-Inria, Inria, Rosario e Carnegie-Mellon
- O nome F no F* é uma homenagem ao **System F**, já o asterisco foi concebido como um tipo de operador de ponto fixo, e o F* foi projetado para ser uma espécie de **ponto fixo de todas essas linguagens** (Fable, F7, F9, F5, FX, F#)
- Linguagem **menos madura** e conhecida do que ferramentas como o Isabelle e o Coq, porém com uma **comunidade crescente e ativa no GitHub**

Como programar em F*

- Programas em F* podem ser feitos em qualquer editor de texto, no entanto, a maioria dos usuários utiliza o **Emacs** em conjunto com o **fstar-model.el** que oferece várias utilidades para a edição e verificação de arquivos em F*



The screenshot shows the Emacs editor window titled 'emacs@clem-w50-mint'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Hide/Show', and 'Help'. The code is written in F* and defines functions for a binary search tree. The code is as follows:

```
val contains: #k:Z → t:tree k → key:Z → Tot bool (decreases t)
let rec contains k t key =
  if key > k
  then false
  else let Node left i right = t in
    i=k
    ∨ (key < i ∧ is_Some left ∧ contains (Some.v left) key)
    ∨ (is_Some right ∧ contains (Some.v right) key)

val in_order_opt: #k:Z → t:option (tree k) → Tot (list Z) (decreases t)
let rec in_order_opt k t = match t with
| None → []
| Some (Node left i right) →
  in_order_opt left@[i]@in_order_opt right

val index_is_max : #max:Z
  → t:option (tree max)
  → x:Z
  → Pure unit T (λ u → List.mem x (in_order_opt t) ⇒ x ≤ max) (decreases t)
let rec index_is_max max t x = match t with
```

The status bar at the bottom shows '74: 0 - ~/data_structures/binary_search_tree.fst 51% -master F w b h s'.

Estrutura sintática básica

- **Módulos:** possuem a extensão .fst e contêm a definição de funções base da linguagem
- **Comentários:** podem ser de uma linha (//) ou de múltiplas linhas ((* *)).

```
3  (*  
4  Este é um comentário de  
5  múltiplas linhas  
6  *)  
7  
8  // Este é um comentário de uma única linha  
9
```

Estrutura sintática básica

- **Módulo Prims - Primitivos:** módulo que contém os tipos primitivos da linguagem, exemplos:
 - False: não possui elementos
 - Unit: elementos únicos
 - Booleans: true ou false, possuem os operadores not, && e ||
 - Condicionais: checam valores e condições, podendo ser if, then ou else

```
17 val condicionais_operadores (b1:bool) (b2:bool) (b3:bool) : nat
18 let condicionais_operadores b1 b2 b3=
19     (* se b1 e b2 forem verdadeiros, ou b3 for verdadeiro,
20     então retorna 18, senão retorna 20 *)
21     if b1 && b2 || b3
22     then 18
23     else 20
```

Estrutura sintática básica

- **Módulo Prims - Primitivos:** módulo que contém os tipos primitivos da linguagem, exemplos:
 - Inteiros (int): elementos do conjunto dos inteiros (0, 1, 2, ...) com os seguintes operadores matemáticos:
 - -, +, /, %, op_Multiply, <, <=, >, >=

```
25 let maior_que_dobro (x:int) (y:int): bool =  
26     // verifica se o valor de x é maior que y + y  
27     x > y + y  
28 let checa_par (x:int) (y:int): bool =  
29     (* verifica se o resto da divisão  
30     de x por 2 é menor que 1, ou seja, se x é par *)  
31     x % 2 < 1
```

Estrutura sintática básica

- **Tipos de Refinamento Booleano no F*:**
permitem especificar propriedades refinadas sobre os valores em programas
 - Adição de condições lógicas expressas por valores booleanos nas especificações de tipos.
 - `let nat = x:int{x >= 0}`
- **Flexibilidade na Definição de Refinamentos:**
 - Possibilidade de definir refinamentos arbitrários de qualquer escolha.

```
1 let empty = x:int { false } // conjunto vazio
2 let zero = x:int{ x = 0 } // tipo que contem um elemento 0
3 let pos = x:int { x > 0 } // os números positivos
4 let neg = x:int { x < 0 } // os números negativos
5 let even = x:int { x % 2 = 0 } // os números pares
6 let odd = x:int { x % 2 = 1 } // os números ímpares
```


■ Verificação de programas

- Tipos dependentes para especificação e verificação precisa
 - Capacidade de definir tipos que dependem de valores em F^* ;
 - Permite expressar e verificar propriedades mais detalhadas dos programas;
 - Exemplos: ordenação de listas, paridade de números inteiros.
- Verificação formal e raciocínio sobre **corretude** e **segurança**:
 - Verificação estática de propriedades antes da execução do programa;
 - Especificação de pré e pós-condições, invariantes e lemas;
 - Forte garantia de corretude e proteção contra falhas de segurança.

Teoria dos tipos

- Utilização de **lemas**;
- Prova formal estabelece **conexão entre propriedades e programa correspondente**;
- Especificação precisa de tipos dependentes e refinamentos booleanos;
- Construção de provas formais para corretude e segurança dos programas.

Exemplo: Quicksort

```
1 let rec sorted (l:list int)
2   : bool
3   = match l with
4     | [] -> true
5     | [x] -> true
6     | x :: y :: xs -> x <= y && sorted (y :: xs)
7
8 let rec mem (#a:eqtype) (i:a) (l:list a)
9   : bool
10  = match l with
11    | [] -> false
12    | hd :: tl -> hd = i || mem i tl
13
14 forall l. sorted (sort l) /\ (forall i. mem i l <==> mem i (sort l))
15
16 let rec sort (l:list int)
17   : Tot (list int) (decreases (length l))
18   = match l with
19     | [] -> []
20     | pivot :: tl ->
21       let hi, lo = partition ((<=) pivot) tl in
22       append (sort lo) (pivot :: sort hi)
```

■ Integração com linguagens

- **Flexibilidade** e praticidade;
- **C:** Geração de código C a partir de especificações em F* para alto desempenho e interação com sistemas existentes;
- **OCaml:** Compartilha semânticas e recursos com o F*, facilitando a incorporação de código OCaml e aproveitando bibliotecas existentes;
- **F#:** Roda na plataforma .NET, aproveitando a biblioteca e infraestrutura disponíveis.

Conclusão

- F* é uma linguagem **poderosa** e **versátil**;
- Abordagem **inovadora** para **programação funcional** e **verificação** de programas;
- Potencial para **desenvolvimento seguro** e confiável;
- Comunidade em **constante crescimento**.

Referências

- 2020. **Proof-oriented Programming in F***. Microsoft Research.
- 2016. Nikhil Swamy, Cătălin Hrițcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Jean-Karim Zinzindohoue, and Santiago Zanella-Béguelin. **Dependent Types and Multi-Monadic Effects in F***. SIGPLAN Not. 51, 1 (jan 2016), 256–270.
<https://doi.org/10.1145/2914770.2837655>



Obrigada

**UDESC – Universidade do Estado de
Santa Catarina**

lais.vossen@edu.udesc.br

daniella.vasconcellos@edu.udesc.br