Primeira Prova de Análise Estática de Programas
- DCC888 -
Ciência da Computação

Nome(s): _____
"Eu dou minha palavra de honra que não trapacearei neste exame."

Número(s) de matrícula: _____

As regras do jogo:

- Vocês podem consultar qualquer material para fazer esta prova, porém vocês não podem se comunicar com colegas do curso (exceto seu colega de equipe, naturalmente) durante o período de prova.
- O período de prova termina às 14h59:59 do dia 9 de Maio.
- As respostas da prova devem ser enviadas para `fernando_at_dcc.ufmg.br`.
- Respostas podem ser escritas à mão, ou em um editor de texto. Quando forem escritas à mão, envie para o instrutor a cópia digital das respostas.
- Seja honesto e lembre-se: **você deu sua palavra de honra**.

Alguns conselhos:

- Escreva sempre algo nas questões, a fim de ganhar algum crédito parcial.
- Fique à vontade para escrever para a lista de discussão com dúvidas. Note que não há garantias que respostas chegarão a tempo. Não envie dúvidas via quaisquer outros meios que não a lista de discussão.
- Se não entender a questão, escreva a sua interpretação junto à resposta.
- A prova não é difícil, ela é divertida, então aproveite!

Tabela 1: Pontos acumulados (para uso do instrutor)

Questão 1	Questão 2	Questão 3	Questão 4	Questão 5	Extra

Questão Extra (0.5 Pontos): Escrevi o soneto mais bonito da língua portuguesa, e teria feito muito mais, se não fossem os Maragatos. Soube deixar umas saudades pelo caminho (deserto e imenso), mas 28 foi muito pouco... quem sou?

Esta questão refere-se a uma linguagem de programação muito simples, PARSER, que permite a leitura de cadeias de caracteres. Essa linguagem de programação possui nove tipos diferentes de comandos, os quais são mostrados na figura abaixo, em sintaxe de SML/NJ:

Auxiliary functions

```
type     Var = string

datatype  Val = INT of int
          | CHAR of char
          | BOOL of bool

exception Halt of Val list

fun update state vr vl =
  fn x => if x = vr then vl else state x
```

Grammar of our programming language, and meaning of instructions

datatype inst = INIT of Var * Val	*Var = Val*
| READ of Var	*Var = Read()*
| INC of Var	*Var = Var + 1*
| DEC of Var	*Var = Var - 1*
| CMP of Var * Var * Val	*Var = Var == Val ? true : false*
| IF of Var * inst * inst	*if (Var) then inst else inst*
| LOOP of Var * inst	*While (Var != true) do inst*
| SEQ of inst list	*inst_1, inst_2, ..., inst_n*
| HALT	*throw new Exception("argh!")*

Ao lado de cada uma das instruções de PARSER é mostrado o seu significado. Um interpretador para PARSER, também escrito em sintaxe de SML/NJ, pode ser visto logo abaixo. Esse interpretador está disponível em `https://homepages.dcc.ufmg.br/~fernando/classes/dcc888/previousExams/Parser.sml`

The interpreter:

```
step (INIT(vr, vl)) input state = (input, update state vr vl)

step HALT input state = raise Halt input
  | step (READ vr) ((CHAR ch)::input) state = (input, update state vr (CHAR ch))

step (INC vr) input state =
  let
    val (INT n) = state vr
  in
    (input, update state vr (INT (n+1)))
  end

step (DEC vr) input state =
  let
    val (INT n) = state vr
  in
    (input, update state vr (INT (n-1)))
  end

step (SEQ nil) input state = (input, state)

step (SEQ (i::insts)) input state =
  let
    val (i_input, i_state) = step i input state
  in
    step (SEQ insts) i_input i_state
  end

step (IF (vr, instsT, instsF)) input state =
  let
    val (BOOL vr_vl) = state vr
  in
    if vr_vl
    then step instsT input state
    else step instsF input state
  end

step (LOOP (vr, insts)) in st =
  let
    val (BOOL vr_vl) = st vr
  in
    if vr_vl then (in, st)
    else
      step (SEQ [insts, LOOP(vr, insts)]) in st
  end

step (CMP (dst, vr, vl)) input state =
  let
    val vl_vr = state vr
    val res = if vl_vr = vl then true else false
    val vl_res = BOOL res
  in
    (input, update state dst vl_res)
  end
```

Conforme pode ser visto na implementação do interpretador, a interpretação de um programa escrito em PARSER recebe uma lista de caracteres como entrada, mais um *estado*. O estado é uma função que associa variáveis (`Var`) a valores (`Val`). Três programas diferentes, escritos em PARSER, podem ser vistos na Figura 1:

1. O primeiro deles, (a), lê até dois caracteres zero (`#"0"`) da entrada. Caso o primeiro, ou o segundo caracteres sejam diferentes de zero, então o programa termina *normalmente*.
2. O segundo programa, (b), lê a string $0^4 1^4 *$, formada por quatro caracteres zero, e quatro caracteres 1, i.e., (`#"1"`), mais quaisquer caracteres após os oito símbolos iniciais.
3. Finalmente, o terceiro programa, (c), simplesmente implementa um laço eterno.

Junto aos dois primeiros programas pode ser visto a maneira como eles são interpretados em SML, e a lista que cada um deles produz ao final do processo de interpretação.

(a)
```
val bb0 = SEQ [INIT("i", INT 1), INIT("b", BOOL false)]
val bb1 = SEQ [READ("v"), CMP("b", "i", INT 2), CMP("x", "v", CHAR #"0")]
val bb2 = IF ("x",  INC "i", INIT("b", BOOL true))
val bb3 = SEQ [bb0, LOOP("b", SEQ [bb1, bb2])]

- val (i, s) = step bb3 [CHAR #"0", CHAR #"0", CHAR #"0"] (fn x => INT 0) ;
val i = [CHAR #"0"] : Val list
val s = fn : Var -> Val
```

(b)
```
val bb4 = SEQ [INIT("i", INT 0), INIT("b", BOOL false)]
val bb5 = SEQ [READ("v"), CMP("b", "i", INT 3), CMP("x", "v", CHAR #"0")]
val bb6 = IF ("x",  INC "i", HALT)
val bb7 = SEQ [bb4, LOOP("b", SEQ [bb5, bb6])]
val bb8 = SEQ [INIT("b", BOOL false), LOOP("b", SEQ [READ("v"), CMP("b", "i",
INT 1), CMP("x", "v", CHAR #"1"), IF("x", DEC("i"), HALT)])]
val bb9 = SEQ [bb7, bb8]

- val (i, s) = step bb9 [CHAR #"0", CHAR #"0", CHAR #"0", CHAR #"0", CHAR #"1", CHAR #"1", CHAR #"1", CHAR #"1",
CHAR #"1", CHAR #"1"] (fn x => INT 0) ;
val i = [CHAR #"1",CHAR #"1"] : Val list
```

(c)
```
val bb10 = SEQ [INIT("b", BOOL false), LOOP("b", SEQ [])]
```

Figura 1: Exemplos de programas escritos em PARSER.

As questões 2, 3 e 4 estão divididas em duas opções. **Você deve escolher somente uma delas**. Cada opção possui as mesmas três questões relacionadas. As opções são:

- Projetar uma análise de *Formato de Entrada*. O objetivo dessa análise é representar a sequência de caracteres que é lida por um programa. Por exemplo, uma resposta aceitável para o programa visto na figura 1 (a) seria $00|0\overline{0}|\overline{0}$, isto é, uma sequência com zero, um ou até dois caracteres `#"0"`. Para o programa visto na Figura 1, você poderia reportar $0^4 1^4$, o que seria ótimo, ou simplesmente $0 * 1*$, o que também seria aceitável. PARSER permite ler qualquer tipo de string aceita por uma máquina de Turing, mas você pode querer aproximar essas strings como expressões regulares, por exemplo.

  Em geral, vocês não vão conseguir representar perfeitamente a entrada lida por um programa escrito em PARSER. É um problema indecidível saber se um certo caractere pode ser lido por um programa qualquer. Assim, você precisa *aproximar* sua análise estática. Sua aproximação pode ser uma sub-aproximação ou uma super-aproximação, isto é, ou você reporta uma string que certamente precisa ser lida pelo programa, ou uma string que "pode"ser lida pelo programa. Nesse último caso, se você reportar uma string que não pode ser lida, então sua análise está errada. No primeiro caso, por outro lado, se você reportar uma string que não pode ser lida, então sua analise está errada.

- Projetar uma análise de *terminação*. O problema de decidir se um programa termina ou não é indecidível; porém, existem vários programas que podemos dizer que terminam. Por exemplo, qualquer programa escrito em PARSER que não possua laços vai terminar, seja com erro (via `HALT`), seja normalmente. Por outro lado, códigos como aquele visto na Figura 1 (c) não terminam.

  Sua análise de terminação deve indicar quais programas certamente irão terminar. Ela é conservadora: se ela indicar que um programa necessariamente termina, então o programa deve terminar. Por outro lado, se ela indicar que um programa pode não terminar, então o dito programa pode terminar ou não. Contudo, a sua análise não pode ser trivial: deve haver uma classe "razoável"de programas contendo laços que ela corretamente marca como terminantes. Faz parte da questão determinar qual é essa classe de programas.

1. Escreva Grafos de Fluxo de Controle (*Control flow graphs*) para cada um dos programas usados como exemplo.

   (a) (3 Pontos) Programa (a)
   (b) (4 Pontos) Programa (b)
   (c) (3 Pontos) Programa (c)

   Note que existem muitas formas de transformar os programas escritos em PARSER em grafos de fluxo de controle. Você pode representar as instruções `LOOP` e `IF` simplesmente como arestas no grafo, por exemplo. A representação que você escolher é parte da questão.

2. Essa questão diz respeito aos reticulados usados para definir a sua opção de análise estática.

   (a) (4 Pontos) Sua análise vai associar informação a que tipo de estrutura? Por exemplo, a nomes de variáveis? A pares formados por pontos de programa e nomes de variáveis? Nesse último caso, como você define pontos de programa?
   (b) (3 Pontos) Qual é o conjunto a partir do qual é formado o reticulado que você vai usar?
   (c) (3 Pontos) Como é a ordem parcial desse reticulado? Quais seriam os limites superiores e inferiores dessa estrutura algébrica? Seu reticulado possui altura infinita ou finita? Você poderia desenhar um diagrama mostrando uma parte de sua estrutura?

3. Escreva funções de transferência para cada uma das nove instruções da linguagem PARSER. Essas funções de transferência, naturalmente, irão depender de como informação está associada à sintaxe de um programa. Você pode usar conjuntos *IN/OUT*, por exemplo, ou associar informação ao nome de variáveis, ou a arestas do grafo de fluxo de controle de um programa, etc.

   (a) (1 Ponto) `INIT`
   (b) (1 Ponto) `READ`
   (c) (1 Ponto) `INC`
   (d) (1 Ponto) `DEC`
   (e) (1 Ponto) `CMP`
   (f) (1 Ponto) `IF`
   (g) (1 Ponto) `HALT`
   (h) (1 Ponto) `SEQ`
   (i) (2 Ponto) `LOOP`

4. Mostre como sua análise estática iria funcionar para cada um dos três programas vistos na Figura 1:

   (a) (1 Ponto) Como o usuário de sua análise estática poderia saber qual a string será lida por um programa, ou se o programa termina ou não? Em outras palavras, qual informação resume o resultado da análise estática?
   (b) (3 Pontos) Programa (a)
   (c) (3 Pontos) Programa (b)
   (d) (3 Pontos) Programa (c)

5. Essa questão diz respeito a terminação.

   (a) (5 Pontos) Mostre que sua análise termina para qualquer programa. Sua demonstração deve ser geral, isto é, você não pode baseá-la em exemplos.
   (b) (5 Pontos) Qual a complexidade assimptótica de sua análise?