Programski jezik PREV

pri predmetu Prevajalniki v študijskem letu 2014/15

(delovna verzija: 4. april 2015)

Boštjan Slivnik

1 Leksikalna pravila

- *Ključne besede*:
  `arr else for fun if rec then typ var where while`
- *Imena atomarnih podatkovnih tipov*:
  `logical integer string`
- *Konstante atomarnih podatkovnih tipov:*
  - `logical` `true` `false`
  - `integer` Poljubno predznačeno zaporedje števk.
  - `string` Poljubno (lahko prazno) zaporedje znakov z ASCII kodami med vključno 32_{10} in 126_{10}, ki je obdano z enojnima navednicama (“`’`”, ASCII koda 39_{10}); izjema je sam znak “`’`”, ki je podvojen.
- *Imena*:
  Poljubno zaporedje črk, števk in podčrtajev, ki se ne začne s števko in ni ne ključna beseda ne ime ali konstanta atomarnega podatkovnega tipa.
- *Ostali simboli*:
  `+ - * / % & | ^ ! == != < > <= >= ( ) [ ] { } : ; . , =`
- *Komentarji*:
  Komentar je poljubno besedilo, ki se začne z znakom “`#`” (ASCII koda 35_{10}) in se razteza do konca vrstice.
- *Belo besedilo*:
  Presledek (ASCII koda 32_{10}), tabulator (ASCII koda 9_{10}) in znaka za konec vrstice (ASCII kodi 10_{10} in 13_{10}) predstavljajo belo besedilo.

2 Sintaksna pravila

source $\longrightarrow$ *definitions*

definitions $\longrightarrow$ *definition*
definitions $\longrightarrow$ *definitions* `;` *definition*

definition $\longrightarrow$ *type_definition*
definition $\longrightarrow$ *function_definition*
definition $\longrightarrow$ *variable_definition*

type_definition $\longrightarrow$ `typ` identifier `:` *type*

type $\longrightarrow$ identifier

type ⟶ `logical`
type ⟶ `integer`
type ⟶ `string`
type ⟶ `arr [` int_const `]` *type*
type ⟶ `rec {` *components* `}`
type ⟶ `^` *type*

components ⟶ *component*
components ⟶ *components* `,` *component*

component ⟶ identifier `:` *type*

function_definition ⟶ `fun` identifier `(` *parameters* `)` `:` *type* `=` *expression*

parameters ⟶ *parameter*
parameters ⟶ *parameters* `,` *parameter*

parameter ⟶ identifier `:` *type*

expression ⟶ *logical_ior_expression*
expression ⟶ *logical_ior_expression* `{ WHERE` *definitions* `}`

logical_ior_expression ⟶ *logical_ior_expression* `|` *logical_and_expression*
logical_ior_expression ⟶ *logical_and_expression*

logical_and_expression ⟶ *logical_and_expression* `&` *compare_expression*
logical_and_expression ⟶ *compare_expression*

compare_expression ⟶ *additive_expression* `==` *additive_expression*
compare_expression ⟶ *additive_expression* `!=` *additive_expression*
compare_expression ⟶ *additive_expression* `<=` *additive_expression*
compare_expression ⟶ *additive_expression* `>=` *additive_expression*
compare_expression ⟶ *additive_expression* `<` *additive_expression*
compare_expression ⟶ *additive_expression* `<` *additive_expression*
compare_expression ⟶ *additive_expression*

additive_expression ⟶ *additive_expression* `+` *multiplicative_expression*
additive_expression ⟶ *additive_expression* `-` *multiplicative_expression*
additive_expression ⟶ *multiplicative_expression*

multiplicative_expression ⟶ *multiplicative_expression* `*` *prefix_expression*
multiplicative_expression ⟶ *multiplicative_expression* `/` *prefix_expression*
multiplicative_expression ⟶ *multiplicative_expression* `%` *prefix_expression*
multiplicative_expression ⟶ *prefix_expression*

prefix_expression ⟶ `+` *prefix_expression*
prefix_expression ⟶ `-` *prefix_expression*
prefix_expression ⟶ `^` *prefix_expression*
prefix_expression ⟶ `!` *prefix_expression*
prefix_expression ⟶ *postfix_expression*

postfix_expression ⟶ *postfix_expression* `^`
postfix_expression ⟶ *postfix_expression* `.` identifier
postfix_expression ⟶ *postfix_expression* `[` *expression* `]`
postfix_expression ⟶ *atom_expression*

atom_expression ⟶ log_constant
atom_expression ⟶ int_constant
atom_expression ⟶ str_constant
atom_expression ⟶ identifier
atom_expression ⟶ identifier `(` *expressions* `)`

atom_expression ⟶ `{` *expression* `=` *expression* `}`
atom_expression ⟶ `{ if` *expression* `then` *expression* `}`
atom_expression ⟶ `{ if` *expression* `then` *expression* `else` *expression* `}`
atom_expression ⟶ `{ while` *expression* `:` *expression* `}`
atom_expression ⟶ `{ for` identifier `=` *expression* `,` *expression* `,` *expression* `:` *expression* `}`
atom_expression ⟶ `(` *expressions* `)`

expressions ⟶ *expression*
expressions ⟶ *expressions* `,` *expression*

variable_definition ⟶ `var` identifer `:` *type*

3 Semantična pravila

Območja vidnosti

- Ime je vidno v celotnem območju vidnosti (od začetka do konca ne glede na mesto definicije).
- Izraz oblike *expression* `{ WHERE` *definitions* `}` ustvari novo vgnezdeno območje vidnosti: izraz in vse definicije so znotraj novega vgnezdenega območja vidnosti.
- Definicija funkcije ustvari novo vgnezdeno območje vidnosti, ki se začne za imenom funkcije in se razteza do konca definicije funkcije.

Območja definiranosti

- Imena komponent posameznega zapisa so definirana v svojem območju definiranosti.
- Vsa ostala imena so definirana v enem samem skupnem območju definiranosti.

Tipiziranost

Podatkovni tipi:

- `logical`, `integer` in `string` opisujejo tipe LOGICAL, INTEGER in STRING, zaporedoma.
- Če je vrednost konstante int_const enaka n in *type* opisuje tip τ, tedaj

  `arr [` int_const `]` *type*

  opisuje tip $\mathrm{ARR}(n, \tau)$.
- Če $type_i$ opisuje tip τ_i za $i \in \{1, 2, \ldots, n\}$, tedaj

  `rec {` $\mathrm{identifier}_1$ `:` $type_1$ `,` $\mathrm{identifier}_2$ `:` $type_2$ `, ... ,` $\mathrm{identifier}_n$ `:` $type_n$ `}`

  opisuje tip $\mathrm{REC}(\tau_1, \tau_2, \ldots, \tau_n)$.
- Če *type* opisuje tip τ, tedaj `^` *type* opisuje tip $\mathrm{PTR}(\tau)$.

Deklaracije:

- Deklaracija tipa

  `typ` identifier `:` *type* `,`

  pri kateri *type* opisuje tip τ, določa, da identifier opisuje tip τ.
- Deklaracija funckije

  `fun` identifier
  `(` $\mathrm{identifier}_1$ `:` $type_1$ `,` $\mathrm{identifier}_2$ `:` $type_2$ `, ... ,` $\mathrm{identifier}_n$ `:` $type_n$ `)`
  `:` *type* `=` *expression* `,`

  pri kateri (a) $type_i$ opisuje tip τ_i za $i \in \{1, 2, \ldots, n\}$, (b) *type* opisuje tip τ in (c) je *expression* tipa τ, določa, da je funkcija identifier tipa $\tau_1 \times \tau_2 \times \ldots \times \tau_n \to \tau$.

- Deklaracija spremenljivke

  `var` identifier `:` *type* ,

  pri kateri *type* opisuje tip τ, določa, da je spremenljivka identifier tipa τ.
- Deklaracija parametra ali komponente

  identifier `:` *type* ,

  pri kateri *type* opisuje tip τ, določa, da je parameter ali komponenta identifier tipa τ.

Izrazi:

- log_const, int_const in str_const so tipa LOGICAL, INTEGER in STRING, zaporedoma.
- Če je *expression* tipa LOGICAL, je `!` *expression* tipa LOGICAL.
- Če je *expression* tipa INTEGER, sta `+` *expression* in `-` *expression* tipa INTEGER.
- Če je *expression* tipa τ, je `^` *expression* tipa PTR(τ).
- Če je *expression* tipa PTR(τ), potem je *expression* `^` tipa τ.
- Če sta $expression_1$ in $expression_2$ tipa LOGICAL, potem je

  $$expression_1 \ op \ expression_2 \quad \text{pri } op \in \{\texttt{\&}, \texttt{|}\}$$

  tipa LOGICAL.
- Če sta $expression_1$ in $expression_2$ tipa INTEGER, je

  $$expression_1 \ op \ expression_2 \quad \text{pri } op \in \{\texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{\%}\}$$

  tipa INTEGER.
- Če sta $expression_1$ in $expression_2$ tipa $\tau \in \{\text{LOGICAL}, \text{INTEGER}, \text{PTR}(\tau)\}$, je

  $$expression_1 \ op \ expression_2 \quad \text{pri } op \in \{\texttt{==}, \texttt{!=}, \texttt{<=}, \texttt{>=}, \texttt{<}, \texttt{>}\}$$

  tipa LOGICAL.
- Če je $expression_1$ tipa ARR(n, τ) in je $expression_2$ tipa INTEGER, je

  $$expression_1 \ \texttt{[} \ expression_2 \ \texttt{]}$$

  tipa τ.
- Če je *expression* tipa REC($\tau_1, \tau_2, \ldots, \tau_n$) in je identifier ime i-te komponenta tega tipa, je izraz

  expression `.` identifier

  tipa τ_i.
- Če je identifier tipa $\tau_1 \times \tau_2 \times \ldots \times \tau_n \to \tau$ in so $expression_i$ tipa τ_i za $i \in \{1, 2, \ldots, n\}$, je izraz

  identifier `(` $expression_1$ `,` $expression_2$ `,` `...` `,` $expression_n$ `)`

  tipa τ.
- Če je *expression* tipa τ, je izraz oblike

  expression `{ where` *definitions* `}`

  tipa τ.
- Če sta $expression_1$ in $expression_2$ tipa $\tau \in \{\text{LOGICAL}, \text{INTEGER}, \text{STRING}, \text{PTR}(\tau)\}$, je

  `{` $expression_1$ `=` $expression_2$ `}`

  tipa τ.

- Če je *expression* tipa LOGICAL, so

  `{ while` *expression* `:` *expression′* `}` ,
  `{ if` *expression* `then` *expression′* `}` in
  `{ if` *expression* `then` *expression′* `else` *expression″* `}`

  tipa VOID.
- Če so identifier, $expression_1$, $expression_2$ in $expression_3$ tipa INTEGER, je

  `{ for` identifier `=` $expression_1$ `,` $expression_2$ `,` $expression_3$ `:` *expression′* `}`

  tipa VOID.
- Če so $expression_i$ tipa τ_i za $i \in \{1, 2, \ldots, n\}$, je

  `(` $expression_1$ `,` $expression_2$ `, ... ,` $expression_n$ `)`

  tipa τ_n.