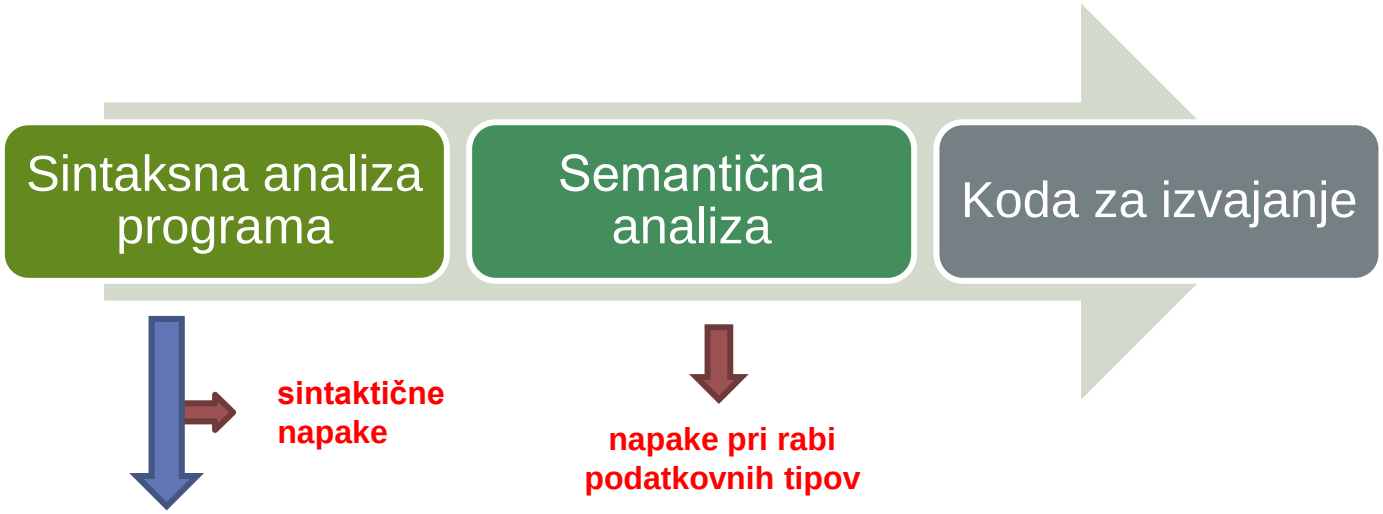
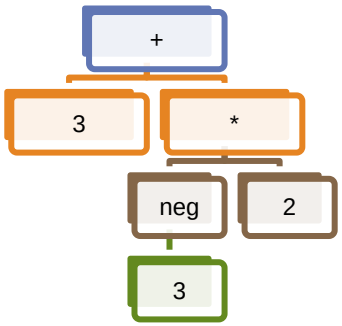
PROGRAMIRANJE
2014/15

implementacija interpreterja za JAIS

Izvajanje programa





ABSTRAKTNO SINTAKSNO DREVO

sintaksno analizo lahko poenostavimo, ker so izrazi v Racketu že sami po sebi drevesne strukture !!!

Naš pristop

- preskočimo fazo sintaksne analize in razčlenjevanja s podajanjem AST, ki je že v izvornem programskem jeziku 0

- sintakso ciljnega jezika X lahko definiramo z uporabo lastnih podatkovnih tipov (`struct`)

- primer: **JAIS (Jezik Aritmetičnih Izračunov v Slovenščini)**
  - rekurzivna funkcija za računanje z izrazi
  - izrazi za:
    - definicijo konstant (konst)
    - definicijo logičnih vrednosti (bool)
    - negacijo (negiraj)
    - seštevanje (sestej)
    - vejanje (ce-potem-sicer)

Preverjanje pravilnosti programa

- primer interpreterja za JAIS:

```
(define (jais e)
  (cond [(konst? e) e]    ; vrnemo izraz v ciljnem jeziku
        [(bool? e) e]
        [(negiraj? e)
         (let ([v (jais (negiraj-e e))])
           (cond [(konst? v) (konst (- (konst-int v)))]
                 [(bool? v) (bool (not (bool-b v)))]
                 [#t (error "negacija nepričakovanega izraza")]))]
        [(sestej? e)
         (let ([v1 (jais (sestej-e1 e))]
               [v2 (jais (sestej-e2 e))])
           (if (and (konst? v1) (konst? v2))
               (konst (+ (konst-int v1) (konst-int v2)))
               (error "seštevanec ni številka")))]
        [#t (error "sintaksa izraza ni pravilna")]))
```

preverjanje ustreznosti podatkovnih tipov (semantika) že izvajamo

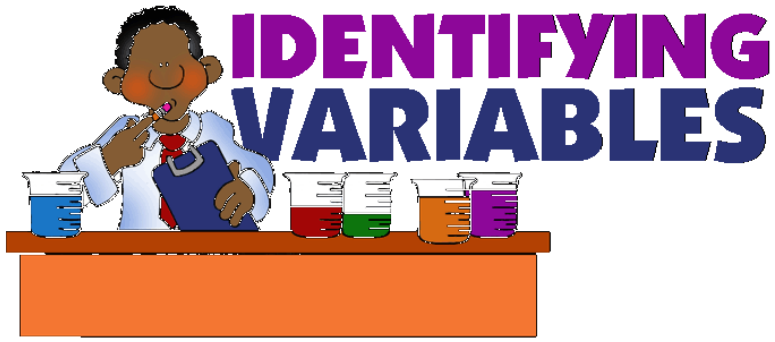
- preverjanje ustreznosti podatkovnih tipov
- preverjanje pravilne sintakse?
  - delno preverja že Racket `(jais (negiraj 1 2 3))`
  - napaka: `(jais (negiraj (konst "lalala")))`
  - potrebno dopolniti kodo, da preverja tudi pravilno sintakso konstant!

Razširitve

- **razširitve preprostega jezika**

  1. definiranje spremenljivk
  2. definiranje lokalnih okolij
  3. definiranje funkcij (funkcijskih ovojnic)
  4. definiranje makrov

potrebujemo znanje o delovanju teh elementov, ki se ga učimo od začetka predmeta

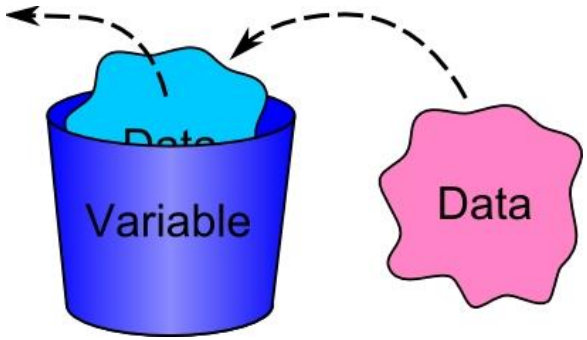


Definiranje spremenljivk in lokalnega okolja

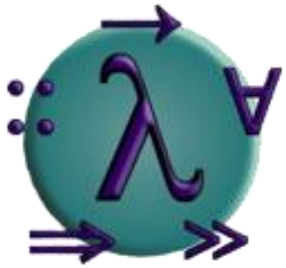
- spremenljivko beremo vedno iz trenutnega okolja (torej potrebujemo okolje)

- okolje prenašamo v spremenljivki jezika 0, ki hrani vrednosti spremenljivk X
  - okolje je na začetku prazno
  - primerna struktura je seznam parov `(ime_spremenljivke . vrednost)`
  - deklaracija nove spremenljivke doda v okolje nov par
  - shranjevanje najprej evalvira podani izraz, nato shrani vrednost

- dostop do spremenljivk
  - preverjanje, ali je spremenljivka definirana
    - če je, vrnemo vrednost
    - sicer napaka



Definiranje funkcij



- potrebujemo strukturo, ki bo hranila funkcijsko ovojnico
  (ne uporabljamo je v sintaksi programa, temveč samo pri izvajanju)

```
(struct ovojnica (okolje funkcija) #:transparent)
```

- v `okolje` shranimo okolje, kjer je funkcija definirana (leksikalni doseg!), v `funkcija` pa funkcijsko kodo

- kako izvesti funkcijski klic?

```
(klici ovojnica argument)
```

  - `ovojnica` mora biti evalvirana v primerek tipa `ovojnica`, sicer napaka
  - `argument` mora biti vrednost (konstanta, boolean), ki je argument funkcije
  - izvajanje:
    - `ovojnica-funkcija` evalviramo v okolju `ovojnica-okolje`, ki ga razširimo z:
      - imenom in vrednostjo argumenta `argument`
      - imenom funkcije, povezano z ovojnico (za rekurzijo)

Optimizacija ovojnic

- okolje v ovojnici lahko vsebuje spremenljivke, ki jih funkcija ne potrebuje
  - senčene spremenljivke iz zunanjega okolja
  - spremenljivke, ki so definirane v funkciji in senčijo zunanje
  - spremenljivke, ki v funkciji ne nastopajo
- ovojnice so lahko prostorsko zelo potratne, če so obsežne
- rešitev: zmanjšamo število spremenljivk v okolju ovojnice na nujno potrebne
- primeri nujno potrebnih spremenljivk
  - `(lambda (a) (+ a b c))`
  - `(lambda (a) (let ([b 5]) (+ a b c)))`
  - `(lambda (a) (+ b (let ([b c]) (* b 5))))`

Implementacija makro sistema

- makro sistem:
  - nadomeščanje (neprijazne) sintakse z drugačno (lepšo)
  - širitev sintakse osnovnega jezika
- v našem interpreterju (JAIS) lahko makro sistem implementiramo kar s funkcijami v jeziku Racket
- primeri

```
(define (in e1 e2)
  (ce-potem-sicer e1 e2 (bool #f)))
```

```
> (jais3 (in (bool #f) (bool #f)))
(bool #f)
> (jais3 (in (bool #f) (bool #t)))
(bool #f)
> (jais3 (in (bool #t) (bool #f)))
(bool #f)
> (jais3 (in (bool #t) (bool #t)))
(bool #t)
```

```
(define (vsota-sez sez)
  (if (null? sez)
      (konst 0)
      (sestej (car sez)
              (vsota-sez (cdr sez)))))
```

```
> (vsota-sez (list (konst 3) (konst 5)
  (konst 2)))

(sestej (konst 3) (sestej (konst 5)
(sestej (konst 2) (konst 0))))
```

- je tak makro sistem higieničen?