Operatsiooniline semantika

- Keele süntaks määrab programmi struktuuri, semantika programmi tähenduse
- Operatsiooniline semantika defineerib programmi tähenduse tema täitmise kaudu
- Meie kasutame operatsioonilise semantika esitamiseks interpretaatoreid
- Kirjeldatavat keelt nimetatakse objektkeeleks, keelt milles interpretaator realiseeritud metakeeleks
- Meil on metakeeleks Scheme

Objektkeel

- Objektkeele semantika esitamisel eristame kahte liiki väärtusi
  - väljendatavad väärtused (ingl. *expressed value*) on avaldiste võimalikud väärtused
  - tähistatavad väärtused (ingl. *denoted value*) on muutujatega seotud väärtused
- Meie esimeses objektkeeles on mõlemat liiki väärtused samad ja võivad olla kas täisarvud või protseduurid

$$\begin{aligned} \textit{Expressed Value} &= \textit{Number} + \textit{Procedure} \\ \textit{Denoted Value} &= \textit{Number} + \textit{Procedure} \end{aligned}$$

Objektkeel

- Objektkeele BNF ja abstraktne süntaks

⟨*exp*⟩	::=	⟨*integer-literal*⟩	`lit (datum)`
	|	⟨*varref*⟩	
	|	⟨*operator*⟩ ⟨*operands*⟩	`app (rator rands)`
⟨*operator*⟩	::=	⟨*varref*⟩ | `(`⟨*exp*⟩`)`	
⟨*operands*⟩	::=	`()`	
	|	`(`⟨*operand*⟩{`,`⟨*operand*⟩}*`)`	
⟨*operand*⟩	::=	⟨*exp*⟩	
⟨*varref*⟩	::=	⟨*var*⟩	`varref (var)`

Objektkeel

- Näiteid objektkeele avaldistest:

```
5
j
+(3,j)
add1(+(3,j))
(add1)(+(3,j))
```

- Abstrakse süntaksi realiseerime kirjete abil:

```
(define-record lit (datum))
(define-record varref (var))
(define-record app (rator rands))
```

Lihtne interpretaator

- Avaldiste väärtustamine:

```
(define eval-exp
  (lambda (exp)
    (variant-case exp
      (lit (datum) datum)
      (varref (var) (apply-env init-env var))
      (app (rator rands)
        (let ((proc (eval-exp rator))
              (args (eval-rands rands)))
          (apply-proc proc args)))
      (else (error "Invalid expression: " exp)))))
```

Lihtne interpretaator

- Literaali interpreteerime tema endana:

```
(lit (datum) datum)
```

- Muutujate väärtus on määratud keskonnaga:

```
(varref (var) (apply-env init-env var))
```

- Keskonna esitamiseks kasutame lõplikke funktsioone:

```
(define the-empty-env (create-empty-ff))
(define extend-env extend-ff*)
(define apply-env apply-ff)
```

Lihtne interpretaator

- Aplikatsiooni väärtustamiseks väärtustame kõigepealt operaatori ja operandid ja seejärel rakendame protseduuri argumentidele:

```
(app (rator rands)
  (let ((proc (eval-exp rator))
        (args (eval-rands rands)))
    (apply-proc proc args)))
```

- Operandide väärtustamine:

```
(define eval-rands
  (lambda (rands)
    (map eval-exp rands)))
```

Lihtne interpretaator

- Meie esimeses objektkeeles on ainult primitiivsed protseduurid:

⟨*procedure*⟩	::=	⟨*prim-op*⟩	`prim-proc (prim-op)`
⟨*prim-op*⟩	::=	⟨*addition*⟩	`'+`
	|	⟨*subtraction*⟩	`'-`
	|	⟨*multiplication*⟩	`'*`
	|	⟨*increment*⟩	`'add1`
	|	⟨*decrement*⟩	`'sub1`

- Abstraktse süntaksi defineerime kirjena:

```
(define-record prim-proc (prim-op))
```

Lihtne interpretaator

- Protseduuride rakendamine:

```
(define apply-proc
  (lambda (proc args)
    (variant-case proc
      (prim-proc (prim-op)
         (apply-prim-op prim-op args))
      (else (error "Invalid procedure:" proc)))))
```

Lihtne interpretaator

- Primitiivprotseduuride rakendamine:

```
(define apply-prim-op
  (lambda (prim-op args)
    (case prim-op
      ((+) (+ (car args) (cadr args)))
      ((-) (- (car args) (cadr args)))
      ((*) (* (car args) (cadr args)))
      ((add1) (+ (car args) 1))
      ((sub1) (- (car args) 1))
      (else (error "Invalid prim-op:" prim-op)))))
```

Lihtne interpretaator

- Algkeskond seob primitiivsete protseduuride nimed neile vastavate väärtustega:

```
(define prim-op-names '(+ - * add1 sub1))


(define init-env
  (extend-env
    prim-op-names
    (map make-prim-proc prim-op-names)
    the-empty-env))
```

Lihtne interpretaator

- Alternatiivne võimalus protseduuride esitamiseks oleks kasutada metakeele (so. Scheme) protseduure:

```
(define apply-proc apply)
(define prim-op-values
   (list + - * (lambda (x) (+ x 1))
               (lambda (x) (- x 1))))
(define init-env
  (extend-env prim-op-names
              prim-op-values
              the-empty-env))
```

- Eeldab, et metakeeles on protseduurid “esimese-klassi kodanikud”

Lihtne interpretaator

- Interpretaatori käivitamine:

```
(define run
  (lambda (x)
    (eval-exp (parse x))))

(define read-eval-print
  (lambda ()
    (display "--> ")
    (write (eval-exp (parse (read))))
    (newline)
    (read-eval-print)))
```

Lihtne interpretaator

- EOPL lisades on toodud 2 erinevat versiooni parseritest:
  - üks võtab sisendprogrammi stringi kujul

```
> (run "add1 (* (2,3))")
7
> (read-eval-print)
--> "- (10, * (sub1 (3), 3))"
4
```

  - teine Scheme listidena

```
> (read-eval-print)
-->(- 10 (* (sub1 3) 3))
4
```

Tingimusavaldised

- Laiendame objektkeelt tingimusavaldistega:

⟨*exp*⟩ ::= ...

| `if` ⟨*exp*⟩ `then` ⟨*exp*⟩ `else` ⟨*exp*⟩

```
if (test-exp then-exp else-exp)
```

- Vastav abstraktne süntaksipuu:

```
(define-record  if (test-exp then-exp else-exp))
```

- Kõiki nullist erinevaid väärtusi tõlgendame tõestena:

```
(define true-value?
  (lambda (x)
    (not (zero? x))))
```

Tingimusavaldised

- Tingimusavaldiste väärtustamine:

```
(define eval-exp
  (lambda (exp)
    (variant-case exp
      ...
      (if (test-exp then-exp else-exp)
        (if (true-value? (eval-exp test-exp))
            (eval-exp then-exp)
            (eval-exp else-exp)))
      ...
```

Järgmiseks korraks

- Lugeda läbi EOPL ptk. 5.1, 5.2
- “Mängida” interpretaatoriga