
KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

Michal Krupka
krupka.inf.upol.cz
michal.krupka@upol.cz
17. listopadu 12, 771 46 Olomouc

Paradigmata programování 2

Slovníček Scheme-Lisp

verze z 27. února 2019

Většina prvků jazyka Scheme má přesný nebo velmi podobný protipól v Common Lispu. Tento slovníček uvádí překlad všech symbolů ze standardu R^{5}RS do Common Lispu jako první pomoc uživatelům jazyka Scheme při programování v Common Lispu.

Slovníček obsahuje překlad všech operátorů ze standardu R^{5}RS , bez ohledu na to, jestli jsme o nich v prvním semestru mluvili.

Při používání slovníčku je třeba být seznámen se základními rozdíly mezi Common Lispem a Scheme uvedenými na přednášce. Je-li vysvětlení ve slovníčku příliš stručné, je třeba sáhnout po definici ve standardu.

1 Základní výrazy

Scheme	**Common Lisp**	**Poznámka pro CL**
`(define a x)`	`(defvar a x)` `(defparameter a x)`	Pouze je-li `define` na nejvyšší úrovni. Lokální `define` (o kterém jsme nemluvili) v CL obdobu nemá (je třeba použít jiný nástroj, např. `let`).
`(quote x), 'x`	`(quote x), 'x`	
`(lambda (x1...xn) ...)`	`(lambda (x1...xn) ...)`	
`(lambda (x1...xn . y) ...)`	`(lambda (x1...xn &rest y) ...)`	
`(lambda y ...)`	`(lambda (&rest y) ...)`	
`(define f (lambda (x1...xn) ...))`	`(defun f (x1...xn) ...)`	Nastavuje ale funkční vazbu.
`(define f (lambda (x1...xn . y) ...))`	`(defun f (x1...xn &rest y) ...)`	Nastavuje ale funkční vazbu.
`(define f (lambda y ...))`	`(defun f (&rest y) ...)`	Nastavuje ale funkční vazbu.
`if`	`if`	Je-li hodnota a v `(if a b)` `nil`, vrací `nil` (ve Scheme nedefinováno).
`(set! a x)`	`(setf a x)`	`(setf a x)` vrací hodnotu `x`.

2 Odvozené výrazy

Scheme	Common Lisp	Poznámka pro CL
`cond`	`cond`	Základy stejné, místo `else` psát `t`, další drobné rozdíly.
`case`	`case`	Drobné rozdíly.
`and, or`	`and, or`	
`let, let*`	`let, let*`	Inicializace nemusí být uvedena, pak navazuje na `nil`, např. `(let (a (b) (c nil)) ...)` navazuje `a`, `b` i `c` na `nil`. Tzv. pojmenované `let` neexistuje.
`letrec`		Varianta neexistuje, viz ale `labels`.
`begin`	`progn`	
`do`	`do`	
`delay`		Neexistuje.
`quasiquote, unquote, unquote-splicing`		Neexistují. Nutno používat zkratky "``", ","a ",@".
`` `(x ,y ,@z) ``	`` `(x ,y ,@z) ``	

3 Makra

Scheme	Common Lisp	Poznámka pro CL
`let-syntax, letrec-syntax, syntax-rules`		V CL nejsou hygienická makra. Obyčejná makra pomocí `defmacro`

3.1 Standardní procedury: predikáty rovnosti

Scheme	Common Lisp	Poznámka pro CL
`eqv?`	`eql`	Přibližně
`eq?`	`eq`	Přibližně
`equal?`	`equal, equalp`	Přibližně

3.2 Standardní procedury: čísla

Scheme	Common Lisp	Poznámka pro CL
`(number? x)`	`(numberp x), (typep x 'number)`	
`(complex? x)`	`(or (complexp x) (rationalp x)) (typep x '(or complex rational))`	`complexp` vrací `nil` pro racionální čísla
`(real? x)`	`(realp x), (typep x 'real)`	
`(rational? x)`	`(rationalp x), (typep x 'rational)`	
`(integer? x)`	`(integerp x), (typep x 'integer)`	
`exact?, inexact?`		CL nezná pojem přesného čísla
`=, <, >, <=, >=`	`=, <, >, <=, >=`	
`zero?, positive?, negative?, odd?, even?`	`zerop, plusp, minusp, oddp, evenp`	
`max, min`	`max, min`	

Scheme	Common Lisp	Poznámka pro CL
`+, -, *, /`	`+, -, *, /`	
`abs`	`abs`	
`quotient`	`truncate`	Jako druhou hodnotu vrací zbytek.
`remainder, modulo`	`rem, mod`	
`gcd, lcm`	`gcd, lcm`	
`numerator, denominator`	`numerator, denominator`	
`floor, truncate, ceiling, round`	`floor, truncate, ceiling, round`	Je možno zadat i dělitel (default `1`), vrací podíl (zaokrouhlený) a zbytek.
`rationalize`	`rationalize`	Nemá druhý argument; počítá vždy co nejpřesněji.
`exp, log, sin, cos, tan, asin, acos, atan`	`exp, log, sin, cos, tan, asin, acos, atan`	`log` připouští druhý parametr — základ (default e)
`sqrt`	`sqrt`	
`expt`	`expt`	
`make-rectangular`	`complex`	
`make-polar`		Není, je třeba použít `abs` a `cis`.
`real-part, imag-part`	`real-part, imag-part`	
`magnitude, angle`	`abs, phase`	
`exact->inexact, inexact->exact`		CL nezná pojem přesného čísla.
`number->string`	`prin1-to-string`	Jedna z obecných funkcí pro tisk do řetězce; netýká se pouze čísel. Soustava se nastavuje dynamickou proměnnou `*print-base*`.
`string->number`	`parse-integer, read-from-string`	Přibližně

4 Standardní procedury: logické typy

Scheme	Common Lisp	Poznámka pro CL
`not`	`not`	
`(boolean? x)`	`(typep x 'boolean)`	

5 Standardní procedury: páry a seznamy

Seznam v Lispu je něco trochu jiného, než seznam ve Scheme: je to buď prázdný seznam (tedy symbol `nil`), nebo pár. Mezi seznamy tedy patří i tečkové seznamy a kruhové seznamy (o kterých se dozvíme tento semestr).

Seznamům, jak je známe ze Scheme, se anglicky říká *proper lists*. Česky je budeme nazývat *čisté seznamy*.

Scheme	Common Lisp	Poznámka pro CL
`(pair? x)`	`(consp x), (typep x 'cons)`	
`cons, car, cdr`	`cons, car, cdr`	
`(set-car! x y), (set-cdr! x y)`	`(setf (car x) y)` `(setf (cdr x) y)`	Vrací `y`.
`caar, caddr, ...`	`caar, caddr, ...`	
`(null? x)`	`(null x), (typep x 'null), (not x)`	Viz též `endp`.
`(list? x)`		V Lispu je seznam něco jiného.
`list`	`list`	
`length`	`list-length`	`length` je rovněž použitelné, dělá ale něco mírně jiného.

Scheme	Common Lisp	Poznámka pro CL
`append`	`append`	
`reverse`	`reverse`	
`(list-tail list n)`	`(nthcdr n list)`	Přibližně.
`(list-ref list n)`	`(nth n list)`, `(elt list n)`	`nth` je přesný překlad, `elt` je obecnější.
`memq, memv, member`	`member`	Je obecnější, porovnávací funkce se zadává parametrem.
`assq, assv, assoc`	`assoc`	Je obecnější, porovnávací funkce se zadává parametrem.

6 Standardní procedury: symboly

Scheme	Common Lisp	Poznámka pro CL
`(symbol? x)`	`(symbolp x)`, `(typep x 'symbol)`	
`symbol->string`	`symbol-name`	
`string->symbol`	`intern`	Přibližně. Standardně je třeba používat velká písmena.

6.0.1 Standardní procedury: znaky

Scheme	Common Lisp	Poznámka pro CL
`(char? x)`	`(characterp x)`, `(typep x 'character)`	
`char=?, char<?, char>?, char<=?, char>=?`	`char=, char<, char>, char<=, char>=`	Všechny akceptují lib. počet argumentů.
`char-ci=?, char-ci<?, char-ci>?, char-ci<=?, char-ci>=?`	`char-equal, char-lessp, char-greaterp, char-not-greaterp, char-not-lessp`	Všechny akceptují lib. počet argumentů.
`char-alphabetic?`	`alpha-char-p`	
`char-numeric?`	`digit-char-p`	Volitelně je možno zadat soustavu, jako true vrací příslušnou číselnou hodnotu.
`char-whitespace?`		Není.
`char-upper-case?`	`upper-case-p`	
`char-lower-case?`	`lower-case-p`	
`char->integer`	`char-code`	
`integer->char`	`code-char`	
`char-upcase, char-downcase`	`char-upcase, char-downcase`	

7 Standardní procedury: řetězce

Scheme	Common Lisp	Poznámka pro CL
`(string? x)`	`(stringp x)`, `(typep x 'string)`	
`(make-string k),`	`(make-string k)`, `(make-sequence 'string k)`	

Scheme	Common Lisp	Poznámka pro CL
`(make-string k fill)`	`(make-string k :initial-element fill)` `(make-sequence 'string k :initial-element fill)`	
`string-length`	`length`	Pracuje nejen s řetězci.
`(string-ref str k)`	`(char str k)`, `(elt str k)`, `(aref str k)`	Přesná obdoba je `char`, `elt` a `aref` jsou obecnější.
`(string-set! str k char)`	`(setf (char str k) char)`, `(setf (elt str k) char)`, `(setf (aref str k) char)`	Vrací `char`.
`string=?`, `string<?`, `string>?`, `string<=?`, `string>=?`	`string=`, `string<`, `string>`, `string<=`, `string>=`	Všechny akceptují lib. počet argumentů.
`string-ci=?`, `string-ci<?`, `string-ci>?`, `string-ci<=?`, `string-ci>=?`	`string-equal`, `string-lessp`, `string-greaterp`, `string-not-greaterp`, `string-not-lessp`	Všechny akceptují lib. počet argumentů.
`substring`	`subseq`	Pracuje nejen s řetězci, poslední parametr je volitelný.
`(string-append str1 ...)`	`(concatenate 'string str1 ...)`	Pracuje nejen s řetězci; aby byl výsledek řetězec, je třeba použít `'string` jako první parametr.
`(string->list str)`, `(list->string lst)`	`(coerce str 'list)`, `(coerce lst 'string)`	
`string-copy`	`copy-seq`	Pracuje nejen s řetězci.
`string-fill!`	`fill`	Pracuje nejen s řetězci, volitelně lze určit rozmezí.

8 Standardní procedury: vektory

Vektor v Common Lispu je libovolné jednorozměrné pole. Proto lze tyto funkce použít i na textové řetězce.

Scheme	Common Lisp	Poznámka pro CL
`(vector? x)`	`(vectorp x)`, `(typep x 'vector)`	
`(make-vector k)`,	`(make-vector k)`, `(make-sequence 'vector k)`	
`(make-vector k fill)`	`(make-vector k :initial-element fill)` `(make-sequence 'vector k :initial-element fill)`	
`vector`	`vector`	
`vector-length`	`length`	
`(vector-ref vec k)`	`(elt vec k)`, `(aref vec k)`	
`(vector-set! vec k x)`	`(setf (elt vec k) x)`, `(setf (aref vec k) x)`	
`(vector->list vec)`, `(list->vector lst)`	`(coerce vec 'list)`, `(coerce lst 'vector)`	
`vector-fill!`	`fill`	Pracuje nejen s vektory, volitelně lze určit rozmezí.

9 Standardní procedury: řízení běhu

Scheme	Common Lisp	Poznámka pro CL
`(procedure? x)`	`(functionp x)`, `(typep x 'function)`	
`apply`	`apply`	
`map`	`mapcar`	Mírnější podmínky. Pozor, funkce `map` dělá něco jiného.
`for-each`	`mapc`	
`force`		Neexistuje.
`call-with-current-continuation`		Neexistuje.
`values`	`values`	Mírnější podmínky.
`call-with-values`	`multiple-value-call`	Přibližně.
`dynamic-wind`	`unwind-protect`	Velmi přibližně; `unwind-protect` je speciální operátor a je jednodušší, protože není `call-with-current-continuation`.

10 Standardní procedury: `eval`

Ve Scheme přijímá `eval` druhý parametr — prostředí, v němž se vyhodnocení provede. Tento parametr může nabývat pouze hodnot vracených procedurami `scheme-report-environment`, `null-environment` a `interaction-environment`. V CL druhý parametr chybí a vyhodnocení se provádí v aktuálním dynamickém prostředí.

Scheme	Common Lisp	Poznámka pro CL
`eval`	`eval`	Rozdíly viz výše.
`scheme-report-environment`, `null-environment`, `interaction-environment`		Neexistují, viz výše.

11 Standardní procedury: vstup a výstup

Scheme	Common Lisp	Poznámka pro CL
`call-with-input-file`, `call-with-output-file`	`with-open-file`	Přibližně. Makro s více možnostmi.
`input-port?`, `output-port?`	`input-stream-p`, `output-stream-p`	
`current-input-port`, `current-output-port`	`*debug-io*`, `*error-output*`, `*query-io*`, `*standard-input*`, `*standard-output*`, `*trace-output*`	Dynamické proměnné, výběr závisí na účelu.
`with-input-from-file`, `with-output-to-file`	`with-open-file`	Přibližně. Makro s více možnostmi.
`open-input-file`, `open-output-file`	`open`	Více možností, zadávají se parametry.
`close-input-port`, `close-output-port`	`close`	
`read`	`read`	
`read-char`	`read-char`	
`peek-char`	`peek-char`	
`eof-object?`		Neexistuje, konec souboru se zjišťuje jinak.

Scheme	Common Lisp	Poznámka pro CL
`char-ready?`		Neexistuje, používat read-char-no-hang
`write`	`prin1`	Jedna z mnoha funkcí pro zápis objektu. Neplést s funkcí `write`.
`display`	`princ`	Jedna z mnoha funkcí pro zápis objektu.
`newline`	`terpri`	Viz též `fresh-line`.
`write-char`	`write-char`	

12 Standardní procedury: systémové rozhraní

Scheme	Common Lisp	Poznámka pro CL
`load`	`load`	Více možností
`transcript-on`, `transcript-off`		Neexistuje, dělá se jinak.