**(set! <symbol> <výraz>)**

1) v hiearchii vyhledej symbol <symbol>, začni aktuálním prostředím a pokračuj přes rodiče až ke globálnímu.

2) Pokud <symbol> nemá vazbu v žádném prostředí, nahlas chybu

3) nahraď vazbu symbolu <symbol> na hodnotu vzniklou vyhodnocením <výraz>

**Box** = mutovatelný kontainer pro hodnotu

**Metody předávání argumentu procedurám**

L-Hodnota – místo v paměti kam ukládáme

R-Hodnota – Obsah který ukládáme

*1) Volání hodnotou*

Volané procedure jsou předány R-hodnoty argumentů

Vázány v lokálním prostředí

Volaná procedura nemůže přiřazovat hodnoty přes argumenty (C, Scheme, Lisp…)

*2) Volání odkazem*

L-Argumenty

Má k dispozici odkazy na úložiště hodnot

Přiřazení do proměnné probíhá v těle procedury, ale změna hodnoty argumentu probíhá v prostředí volání procedury

*3) Volání výsledkem*

L- hodnoty jsou uchovávány v lokálním prostředí

Po dokončení se převedou kopie lokálních hodnot na paměťová místa předaných argumentů

*4) Volání jménem*

Předána jména argumentů

Při každém nalezení volání argumentu zastupovaného jménem je jméno opět vyhodnoceno

**Makra**

1. pohled -Rozšíření syntaxe jazyka

Dáno definicí transformačního procesu

Po načtení makra následuje makroexpanze preprocesorem a po dokončení všech makroexpanzí dojde k vyhodnocení kódu

+ Preprocessor a eval jsou zcela nezávislé

+/- Preprocesor může být aktivován hned po načtení výrazu, což staví makra “mimo jazyk”

- Nejsou element prvního řádu

2. pohled – Makra jsou Speciální element jazyka

Element ukazující na transformační proceduru

Makra jsou element 1. řádu

Potřeba rozšířit eval o uživatelsky definované speciální formy

+ Lze s nimi pracovat jako s daty – mohou dynamicky vznika a zanikat v běhu program

+ Existence anonymního makra

- k makroexpanzi dochází až v evalu

Pokud je F1 makro, jehož transformační procedura je T, pak:

1) F´= Apply[T, E1….En)

(výsledek aplikace T na nevyhodnocené argumenty)

2) Výsledek vyhodnocení F element E v prostředí P je definován jako

F = eval [F´, P]

Výsledek vyhodnocení element F´v prostředí P

(Transformační procedura makra v Dr Racket se neaplikuje v prostředí vzniku ale v prostředí počátečních vazeb (počáteční prostředí je takové prostředí ve kterém nelze definovat vazby tzn. počáteční =/= globální)

**Hygienická makra**

Bezpečná makra

Definována pomocí přepisovacích pravidel

+ odpadají složité kvazikvotované výrazy

+ nemůže nastat symbol capture

+ v souladu s lexikálním rozsahem platnosti

1) jestliže je v těle makra definována vazba na dosud nepoužitý symbol, je v těle makra transparentně přejmenován tak, aby nedošlo ke kolizi)

2)Při vyhodnocení těla se všechny vazby výskytů volných symbolů (, které nebyly vytvořeny lokálně v rámci makra) hledají v prostředí definice makra (lexikální předek). (při použití makra nezáleží na vazbách v prostředí použití makra)

- některá makra se tak nedělají pohodlně

**Streamy**

Aplikace líného vyhodnocování

Líně vyhodnocované seznamy

Prázdný seznam = proud

Tečkový pár, kde (e . p) e = element p = příslib proudu

**Nekonečné streamy**

“potenciálně nekonečná lineární datová struktura”

Tzn. Stream-cdr se nikdy nedostane na konec

(aktuální prvek . příslib pokračování)

Rekurzivní procedura bez limitní podmínky

**Postfixový generator**

= množina seznamů, kde pro každé n € N system obsahuje seznam délky n

= pro každé dva s, t € S platí, že buď je s prefix t nebo t je prefix s

**Nekonečný stream příslušný postfixovému generátoru S** je element reprezentující posloupnost (Ei) I jdoucí od 0 k nekonečnu, kde Ei je element na i-té pozici libovolného seznamu s € S mající alespoň i+1 prvků

Použití – 1) řízení průběhu výpočtu pomocí dat

2) není dopředu známa velikost dat, nebo nelze odhadnout kolik dat je třeba zpracovat (např. vstupně výstupní operace (ve scheme přes porty))

**Aktuální pokračování** –

1) Kontext – procedura 1 argumentu reprezentující výpočet, který nastane okamžitě po vyhodnocení výrazu

2) Úniková procedura – procedura, po jejíž aplikaci se ukončí zbylý výpočet a jako výsledek je okamžitě vrácena hodnota její aplikace

3) aktuální prostředí/Kontinuace – úniková procedura vytvořená z kontextu aktuálního výrazu

Call/cc <příjemce> - musí být procedura jednoho argumentu

Vytvoří se context v aktuálně vyhodnocovaném výrazu

<příjemce> je zavolán s argumentem (aktulní pokračování) = procedura vzniklá vyhodnocením (escaper <kontext>)

Korutiny – reprezentují vzájemně se přepínající podprogramy

Programy:

(define-macro prog1 (lambda x

`(if ,(null? x)

(set! x 0)

(let ((s ,(car x)))

(begin ,@(cdr x)

s)))))

(define a (mcons '() '()))

(define b (mcons '() '()))

(set-mcdr! a b)

(set-mcar! a b)

(set-mcar! b a)

(set-mcdr! b a)

(define a (mcons 1 '()))

(define b (mcons '() '()))

(set-mcdr! a b)

(set-mcdr! b b)

(set-mcar! b a)

(define counter (let ((value 0)) (lambda x (if (null? x) value (set! value (+ value (car x))) ))))

(define-syntax do-times

(syntax-rules ()

((do-times (x x1) body ... )

(call/cc (lambda (exit)

(define break exit)

(let loop((x 0)

(y x1)

)

(if (= y (+ x 1))

(begin body ...)

(begin body ...

(loop (+ x 1) y))))) ))))

(define list-index

(lambda (e l)

(call/cc

(Lambda (exit)

(let poc ((l l))

(zadání)