Εξιδανικευμένη Scheme.

Μικρό ιστορικό:

Η αρχή γίνεται με την ISWIM (If you See What I Mean) του Landin (1966).

ISWIM: βασισμένη στο λ-λογισμό, με syntactic sugar,

Idealized Scheme

$$\text{Δύο εκδοχές} \begin{cases} \text{call-by-value}, \ ISWIM_v \quad (v=\text{value}) \\ \text{call-by-name}; \ ISWIM_n \quad (n=\text{name}) \end{cases} \xrightarrow{\text{επέκταση}} \begin{matrix} IS_v \\ IS_n. \end{matrix}$$

---

core syntax: $N ::= x \mid NN \mid \lambda x.M$ (όροι-εκφράσεις του λ-λογισμού)

Operational semantics: Με όρους (μέσω) της SECD-machine. Είναι εικονική μηχανή με βάση την οποία αποτιμώνται οι εκφράσεις του λ-λογισμού.

Plotkin (1975): η SECD-μηχανή είναι ισοδύναμη με τη συνάρτηση $eval_v$

για την οποία ισχύει:

1. $eval_v(V) = V$
2. $eval_v(MN) = eval_v(Q[V/x])$, αν $eval_v(M) = \lambda x Q$
   και $eval_v(N) = V$

όπου V παριστάνει μια τιμή, και τιμές είναι οι μεταβλητές ή οι λ-αφαιρέσεις. $(x, \lambda x N)$. ← η μορφή των τιμών

και η αποτίμηση γίνεται με call_by_value.

β_v-redex $\boxed{(\lambda x.M)\,V}$ (το όρισμα πρέπει να είναι τιμή)

Τι ισοδύναμα προσφέρει η $eval_v$. Το αποτέλεσμα της $eval_v$ (τιμή), προκύπτει αν επαναλαμβανόμενα ανάγουμε το leftmost-outermost β_v-redex, που δεν βρίσκεται στο εσωτερικό μιας λ-αφαίρεσης.

Άλλο εργαλείο για στρατηγική υπολογισμού (αναγωγής) η οποία θα ήθελα επιθυμητό να περιγράψω με φορμαλιστικό τρόπο. Αυτό το κάνουν οι Felleisen et al μέσω των (evaluation) contexts.

Contexts

Context είναι ένας λ-όρος όπου ~~…~~ μια εμφάνιση ενός υποόρου έχει αντικατασταθεί με μία οπή $\square$. (λ-όρος με οπή).

π.χ. $((\lambda x.xx)\square)(\lambda y.y)$. Εάν το context αυτό έχει προκύψει από τον όρο $((\lambda x.xx)M)(\lambda y.y)$ τότε λέμε ότι είναι το context του M.

δηλ. αν E το context και M όρος τότε $E[M]$ είναι τον όρο που προκύπτει από το E αν αντικαταστήσουμε την οπή με το M.

Παράδειγμα

ορίζουμε το context $E ::= \square \mid ME \mid EN \mid \lambda x.E$ (λ-όροι με μία οπή)

Τότε αν το β-redex R εμφανίζεται ως υποόρος στον όρο M, τότε υπάρχει ένα μοναδικό context E ώστε $M = E[R]$.

Η σχέση $\to_\beta$ μπορεί να οριστεί ως εξής:

$$\frac{}{(\lambda x M)N \to_\beta M[N/x]} \qquad \frac{M \to_\beta N}{E[M] \to_\beta E[N]}$$

Πώς μπορούμε να περιγράψουμε με contexts την call-by-value διαδικασία, που περιγράψαμε πιο πάνω;

call-by-value

Οι τιμές V είναι $V ::= x \mid \lambda x.M$

Τα redexes ~~(λx.M)~~ $(\lambda x.M)V$. Έτσι δεν μπορούμε να αναγάγουμε ένα redex $(\lambda x.M)N$ παρά μόνο αν το N είναι τιμή $N \equiv V$. (Call-by-value).

και μια (ντετερμινιστική) διαδικασία αναγωγής ορίζεται με Contexts έτσι:

Contexts

$$E ::= \square \mid EN \mid VE$$

Η αναγωγή (ενός βήματος) $\to_v$ ορίζεται και πάλι ως:

$$\frac{}{(\lambda x.M)V \to_\beta M[V/x]}\,\beta_v \qquad \frac{M \to_{\beta_v} N}{E[M] \to_{\beta_v} E[N]}$$

άρα το β_v ορίζεται με context rewrite rule
$E[(\lambda x.M)V] \to_{\beta_v} E[M[V/x]]$
λόγω μοναδικότητας του E

Ισχύει: Αν M κλειστός όρος τότε είτε είναι τιμή είτε μπορεί να γραφτεί με μοναδικό τρόπο ως $M = E[R]$, όπου R β_v-redex και R είναι το left-most-outermost β_v-redex που δεν βρίσκεται μέσα σε λ-αφαίρεση

το σημειώνουμε $M \propto E[R]$

Άρα η call-by-value αναγωγή που ορίσαμε ανάγει το left-most outer-most β_v-redex έως ότου καταλήξει σε τιμή.

παράδειγμα: αν $E_0 = (\lambda x.M)[\,]$ και $E_1 = [\,]$ τότε

$$(\lambda x M)V = E_0[V] \propto E_1[(\lambda x M)V]$$

Θεώρημα (Plotkin)

$$eval_v(M) = V \iff \underbrace{M \twoheadrightarrow_{\beta_v} V}_{\text{το } M \text{ αποτιμάται στο } V}$$

Λήμμα: (1) $E[M] \xrightarrow[\beta_v]{k} E[N] \implies M \xrightarrow[\beta_v]{k} N$ (k βήματα)

(2) $E[M] \twoheadrightarrow_{\beta_v} V \implies$ υπάρχει τιμή V_0 (τιμή) τ.ω. $E[M] \twoheadrightarrow E[V_0] \twoheadrightarrow_{\beta_v} V$

Δηλ. αν αποτιμάται το M στο $E[M]$, η διαδικασία συνεχίζεται "μέσα" στον υπόορο M έως ότου φτάσουμε σε τιμή V_0 και μετά από αυτό συνεχίζεται η διαδικασία που involve υπόορους του E. Δηλαδή το E ~~[illegible]~~ αντιπροσωπεύει το "υπόλοιπο του υπολογισμού που απομένει να εκτελεστεί, αφού έχει αποτιμηθεί το M". Το E καλείται continuation του M σ' αυτό το σημείο της ακολουθίας υπολογισμού.

Επίσης: $M_0 \to_{\beta_v} \cdots \to_{\beta_v} M_i \to_{\beta_v} \cdots$ και $M_i \equiv E[M]$ και M όχι value

Όπως είδαμε η Scheme περιέχει το call/cc, ενώ ο control που παρέχει τη δυνατότητα στις προγραμματιστές να έχουν απ' ευθείας πρόσβαση σε μια αφαίρεση που αντιπροσωπεύει την current continuation.

Felleisen: Παρουσιάζει την IS (Idealized Scheme) που έχει δύο control operators που αναφέρονται σε continuations.

Σύνταξη IS: προσθέτουμε το $\mathcal{A}$ και $\mathcal{C}$

$$N ::= \dots \mathcal{A}(N) \mid \mathcal{C}(N)$$

$\mathcal{A}$: abort
$\mathcal{C}$: control.

Ισχύει πάλι ότι αν M ένας κλειστός όρος της IS τότε αν δεν είναι τιμή, θα γράφεται μοναδικά ως $M = E[R]$ όπου R ένα β_v-redex ή $R = \mathcal{A}(N)$ ή $R = \mathcal{C}(N)$

Αναγωγές

$E[\mathcal{A}(M)] \to_{\mathcal{A}} M$ (abort), εγκαταλείπει το context και συνεχίζει με το M.

$E[\mathcal{C}(M)] \to_{\mathcal{C}} M\,\lambda z.\mathcal{A}(E[z])$. (control), εγκαταλείπει το context και εφαρμόζει το M σε μια αφαίρεση του εγκαταλελειμμένου context. Εάν το $\lambda z.\mathcal{A}(E[z])$ καλεστεί με τιμή V σε όποιο context E_1 τότε εγκαταλείπεται το context E_1 στο οποίο καλέστηκε και η αποτίμηση επανέρχεται με $E[V]$.

Εάν έχουμε λειτουργικό ορισμό του $\mathcal{C}$ τότε το $\mathcal{A}$ μπορεί να οριστεί από $\mathcal{A}(M) \overset{\text{ορ}}{=} \mathcal{C}(\lambda d.M)$ $d \notin FV(M)$. Τότε

$$E(\mathcal{A}(M)) = E[\mathcal{C}(\lambda d.M)] \to_{\mathcal{C}} (\lambda d.M)\,\lambda z.\mathcal{A}(E[z]))$$

$$\to_{\beta v} M$$

Τι διαφορά έχει το $\mathcal{C}$ με το call/cc της Scheme; Η διαφορά έγκειται στο εξής: Ενώ με το $\mathcal{C}$ το context εγκαταλείπεται πλήρως, με τον call/cc εγκαταλείπεται αλλά επιστρέφει στο σημείο στο οποίο βρισκόταν. Οπότε: Αν το continuation δεν το καλέσουμε ποτέ τότε $\mathcal{C}$ και call/cc δρουν διαφορετικά. Το $\mathcal{C}$ εγκαταλείπει πλήρως ενώ το call/cc διατηρεί το context. Αν το καλέσουμε, τα δύο δρουν ομοίως.

~~π.χ. (+ 4 (control k (* 3 2))) ~>~~

π.χ. $(+\ 4\ (\text{control}\ \lambda k.(*\ 3\ 2))) \rightsquigarrow (\lambda k.(*\ 3\ 2))\ \lambda z.\mathcal{A}((+\ 4\ z))$

$\rightsquigarrow (*\ 3\ 2) \rightsquigarrow 6$

$(+\ 4\ (\text{call/cc}\ \lambda k.(*\ 3\ 2))) \rightsquigarrow E[(\lambda k.(*\ 3\ 2))(\lambda z.\mathcal{A}(+\ 4\ z))]$

$\rightsquigarrow E[(*\ 3\ 2)] \rightsquigarrow (+\ 4\ (*\ 3\ 2)) \rightsquigarrow (+\ 4\ 6) \rightsquigarrow 10$

Αν το call/cc το συμβολίσουμε με $\mathcal{K}$ τότε μπορούμε να το ορίσουμε ως

$$E[\mathcal{K}(M)] \to_{\mathcal{K}} E[M\ \lambda z.\mathcal{A}(E[z])]$$

Αλλά το $\mathcal{K}$ (με την $\to_{\mathcal{K}}$) ορίζεται μέσω του C ως εξής

$$\mathcal{K}_d(M) \overset{\text{ορ.}}{=} C(\lambda k.k(Mk))$$

Τότε $E[\mathcal{K}_d(M)] = E[C(\lambda k.k(Mk))] \to_C (\lambda k.k(Mk))\ \lambda z.\mathcal{A}(E[z])$

$\to_{\beta_v} (\lambda z.\mathcal{A}(E[z]))\ (M\ \lambda z.\mathcal{A}(E[z]))$

$\to_{\beta_v} \mathcal{A}(E[M\ \lambda z.\mathcal{A}(E[z])])$

$\to_{\mathcal{A}} E[M\ \lambda z.\mathcal{A}(E[z])]$ ✓

° Μηχανισμοί catch/throw στην Common Lisp.

• Catch: χρησιμοποιούμε ένα label j για το context στο οποίο βρισκόμαστε έστω E_0 ← (current context).

• throw: Αν το j δεν καλείται ποτέ τότε η έκφραση αποτιμάται "κανονικά".

Αν όμως μία εφαρμογή του j, όπως $E_1[jV]$, συμβαίνει κατά τη διάρκεια του υπολογισμού τότε η τιμή V is "thrown back to" στο σημείο που έχει το label j. Δηλαδή το context E_1 εγκαταλείπεται και συνεχίζουμε με $E_0[V]$.

Μπορούμε να θεωρήσουμε το labeling ως $E_0[\mathcal{K}_d(\lambda j M)]$. έχουμε

έστω $Q = \lambda z.\mathcal{A}(E_0[z])$

$$E_0[\mathcal{K}_d(\lambda j.M)] \to_c (\lambda \kappa.\kappa((\lambda j.M)\kappa))Q$$
$$\to_{\beta v} Q((\lambda j.M)Q)$$
$$\to_{\beta v} Q(M[Q/j])$$

όπου $M[Q/j] \twoheadrightarrow_{\beta v} V$ τότε ~~…~~ καταλήγουμε "κανονικά"

$$\twoheadrightarrow_{\beta v} QV$$
$$\to_{\beta} \mathcal{A}(E_0[V])$$
$$\to_{\mathcal{A}} E_0[V]$$

Αν όμως μια τιμή "is eventually thrown" τότε

$$Q(M[Q/j]) \to_{\beta v} E_1[QV]$$
$$\to_{\beta v} E_1[\mathcal{A}(E_0[V])]$$
$$\to_{\mathcal{A}} E_0[V]$$

δηλ. το E_1 εγκαταλείπεται και η αποτίμηση συνεχίζει με V στο αποθηκευμένο context E_0

- - - -

[76.1]

Παράδειγμα.

1 + catch a in (10 + 41) $\xrightarrow{\text{eval.}}$	1 + [10+41] ⇒ 52	A
1 + catch a in (10 + throw 41 to a)	1 + 41 ⇒ 42	B

$1 + \square = E_0$

A $1 + (\text{call/cc}\ [\lambda j\, (10+41)])$. Context του call/cc

Άρα $E_0[\text{call/cc}\ [\lambda i\, (10+41)]] \rightarrow (\lambda i\, (10+41))\ (\text{escaper}\ \lambda\square.\ E_0[\square])$

$\underbrace{\text{escaper}\ \lambda\square.\ E_0[\square]}_{\{\lambda\square.\ \text{or}\ \lambda\square.\ 1+\square\}}$

//

E_0

$\rightarrow 1 + (\lambda j\, (10+41))\ (\text{escaper}\ \lambda\square.\ 1+\square)$

$\rightarrow 1 + (10+41) \rightarrow 52.$

B $1 + \text{call/cc}\ (\lambda i.\ (10 + j\, 41))$

$\rightarrow 1 + [\lambda i.\ (10 + j\, 41)]\ (\text{escaper}\ \lambda\square.\ 1+\square)$

$\rightarrow 1 + (10 + (\text{escaper}\ \lambda\square.\ 1+\square)\ 41) \rightarrow (1+41) \Rightarrow 42$

$= E_0[v],\ v = 41$