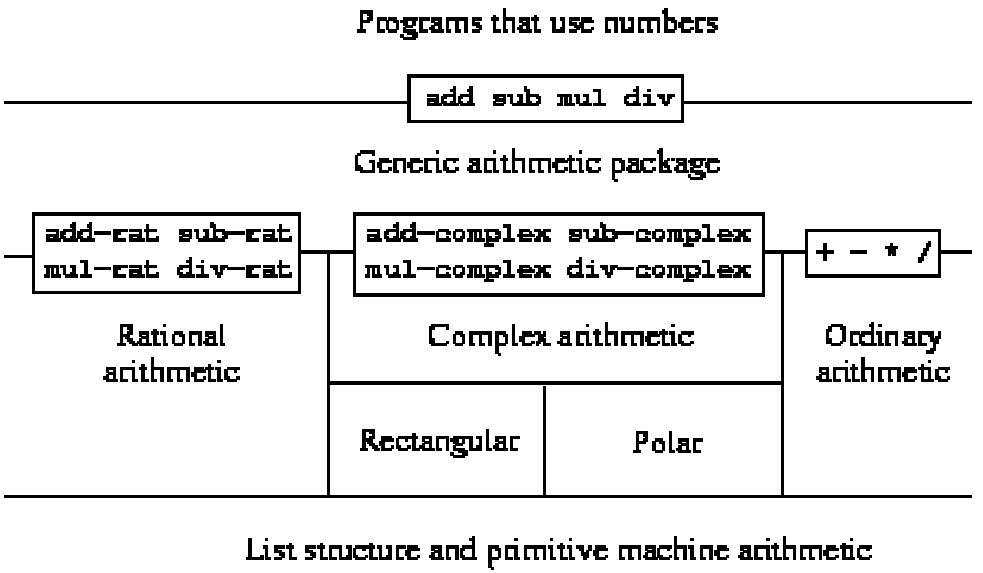
SICP - ABSZTRAKT ADATTÍPUSOK

Általános megfontolások

- Szeretnénk különböző fajta adatokat egységesen kezelni, a részleteket elrejtve
- Ezt egységesen és transzparens módon kívánjuk megoldani, hogy a modellünk esetleges bővítése/változása ne vonja maga után a teljes kód átírását
- A rendszerünk felépítését a következő fólia mutatja

A rendszerünk felépítése



Általános aritmetikai eljárások

- Az általános műveleteket az alábbi módon definiáljuk. Minden szám-típusnak saját címkéje (*tag*) lesz, így a hozzá tartozó eljárások hatáskörébe fog tartozni.

```
(define (add x y) (apply-generic 'add x y))
(define (sub x y) (apply-generic 'sub x y))
(define (mul x y) (apply-generic 'mul x y))
(define (div x y) (apply-generic 'div x y))
```

- Már csak installálni kell a kész csomagokat (lásd forrás)

```
(install-scheme-number-package)
(install-rational-package)
(install-complex-package)
```

Különböző adattípiusok kombinált használata

- Az eddigi eljárásaink szépen működnek, viszont mi történik, ha egy valós számot akarok egy komplexhez adni? Ezt nem lehet kényelmesen megtenni, legfeljebb úgy, ha a csomag használója külön egy komplex számot képez a valósból. Ez nyilván nem egy szép és felhasználóbarát megoldás.
- Egy kicsit jobb megoldás, ha minden egyes csomagon belül az összes létező kombinációt külön kezeljük. Így viszont egy új osztály bevezetésekor gyakorlatilag az egész programot át kellene írni. Valamint az sem tisztázott, hogy egy valós-komplex összeadás a komplex vagy a valós csomag hatáskörébe tartozik.
- Coercion:
  - egy művelet elvégzése előtt meggyőződünk, hogy definiálva van-e az argumentumokra az adott művelet.
  - Ha igen, akkor a megfelelő eljárásnak továbbadjuk a végrehajtást.
  - Ha nem, akkor coercion-t próbálunk végrehajtani. Először megpróbáljuk az első argumentumot a másodikra alakítani, majd ha nem meggy, akkor a másodikat az elsőre. Ha most sem járunk szerencsével, akkor feladjuk.

Coercion

```
(define (apply-generic op . args)
   (let ((type-tags (map type-tag args)))
     (let ((proc (get op type-tags)))
       (if proc
           (apply proc (map contents args))
           (if (= (length args) 2)
               (let ((type1 (car type-tags))
                     (type2 (cadr type-tags))
                     (a1 (car args))
                     (a2 (cadr args)))
                 (let ((t1->t2 (get-coercion type1 type2))
                       (t2->t1 (get-coercion type2 type1)))
                   (cond (t1->t2
                          (apply-generic op (t1->t2 a1) a2))
                         (t2->t1
                          (apply-generic op a1 (t2->t1 a2)))
                         (else
                          (error "No method for these types"
                                 (list op type-tags))))))
               (error "No method for these types"
                      (list op type-tags)))))))
```

- Így már csak legfeljebb n^2 eljárást kell írni egy n típusból álló rendszerhez.
- Ennek ellenére még így sem elég absztrakt a megfogalmazás. Valamint nem kezelhető az az eset sem, amikor mindkét operandust egy harmadik típusúvá alakítva végezhető csak el egy művelet.

Adattípusok hierarchiája

- Észrevehető, hogy sok esetben egy világos hierarchia állítható fel a típusaink között, a legáltalánosabbtól a legspeciálisabbig haladva.
- Egy ilyen hierarchikus megoldás azért is nagyon kényelmes, mert egy új típus felvételekor elég csak a hierarchiában elfoglalt helyét és a szomszédaihoz szükséges konverziókat megadni. Azzal nem szükséges foglalkozni, hogy a többi típushoz képest hol helyezkedik el, mert ezen információk következnek a világunk felépítéséből.
- Implementáció: minden típushoz hozzárendelünk egy eljárást, ami a toronyban az eggyel magasabb szintre emeli (ha lehetséges).
- Probléma: amennyiben egy típusnak több szülőtípusa is lehet, akkor nem tudunk mit csinálni (pl. sokszögek), nem tudjuk eldönteni, hova kellene felemelni az adattípusunkat. Erre a problémára pillanatnyilag nincs általánosan elfogadott jó megoldás (a SICP szerint).

SZIMBOLIKUS ALGEBRA

Általános megfogalmazás, gondolatok

- A szimbolikus műveletvégzés és adatreprezentáció egy tipikus példája az eddig elmondottak hasznosságára, és egy elég komplex rendszer is ahhoz, hogy érdemes legyen magas szinten foglalkozni vele.
- Szokásos absztrakiók:
  - primitív objektumok: konstansok és változók
  - a primitíveken tipikusan lineáris kombinációt, polinomizálást, valamint valós- és trigonometrikus függvényeket szoktunk értelmezni
- Egy komplett, jól működő rendszer implementálása nagyon nagy feladat, ahol nagyon sok ötletre van szükség. Maga a SICP is „csak” egy polinomaritmetikát ismerő rendszert készít.
- Ilyesmi házi feladat?

ÖSSZEFOGLALÁS

Amit a fontosnak tartok

- Az adatok rétegekbe csomagolásával nagyon szép és elegáns absztrakciók készülhetnek, amikkel tényleg könnyű dolgozni.
- Bár méréseket nem végeztem, felmerült bennem néhány kérdés:
  - Mi az a bonyolultsági szint, ahol már átláthatóbb egy ilyen megoldás, mint egy egyszerű, de specifikus?
  - Mennyire hatékony? Elég sok munkát kell végezni adott esetben, hogy kibontsuk a struktúrákat, majd transzformálgassuk őket igény szerint.