Bab VI
Value, Domain dan Type

Value

Suatu nilai (value) adalah hal apapun yang mungkin dapat dievaluasi, disimpan dalam suatu struktur data, dikirimkan sebagai suatu argumentasi atau dikembalikan lagi sebagai hasil.

Suatu perhitungan adalah suatu urutan operasi yang diberlakukan untuk suatu nilai untuk menghasilkan suatu nilai. Dengan demikian nilai-nilai dan operasi adalah dasar perhitungan.

Dalam matematika, kumpulan dari argumen-argumen dan hasil-hasil dari fungsi dikenal dengan domain dan co-domain. Domain akan ditandai sebagai kumpulan dari nilai-nilai yang dikirimkan sebagai argumen-argumen atau dikembalikan sebagai hasil.

Nilai-nilai kebenaran, karakter, integer, real, file, pointers, record, set, prosedur dan abstrak fungsi, lingkungan, perintah dan definisi tidak lain bagian dari bahasa pemrograman.

Dua kategori domain, yaitu :

a. Primitive Domain
   Sifatnya atomic
b. Compound Domain
   Kumpulan dari nilai-nilai yang dibangun dari domain-domain pembangun satu atau domain pembangun yang lain.

Suatu domain adalah satu kumpulan elemen-elemen dan digabungkan dengan sekumpulan operasi.

Domain

Elemen-elemen Teori Domain

Ada beberapa compound domain yang digunakan dalam computer sign, yaitu: array, tupple, record, union, set, list, tree, file, relation, definition dan mapping.

Compound domain dibangun oleh suatu domain pembangun. Suatu domain pembangun adalah :

a. Product domain
b. Sum domain
c. Function domain
d. Power domain
e. Recursive domain

Product Domain

Domain-domain yang dibangun oleh pembangun product domain disebut tuples dalam ML, record dalam Cobol, Pascal dan ADA, struktur dalam C dan C++. Bentuk product domain adalah dasar dari database relational dan pemrograman logic.

Pada kasus biner, product domain pembangun, x, membangun domain A x B dari domain A dan B.

Jika a adalah elemen dari A dan b adalah suatu elemen dari B maka (a,b) adalah suatu elemen dari A x B.

$$A \times B = \{(a,b) \mid a \text{ in } A, b \text{ in } B\}$$

Product Domain : D_0 x … x D_n
Assembly operation: $(a_0,\ldots,a_n)$ in D_0 x…x D_n where a_i in D_i and
D_0 x…x D_n = $\{(a_0,\ldots a_n) \mid a_i \text{ in } D_i\}$
Disassembly operation: $(a_0,\ldots,a_n) \mid i = a_i$ for $0 <= i <= n$

Domain produk disebut "Cartesian" atau "Cross" produk. Dalam Pascal disebut record, sedangkan dalam C disebut suatu struktur.

Sum Domain

Domain-domain yang dibangun oleh pembangun sum domain disebut varian record dalam Pascal dan ADA, Unions dalam Algol-68, Constructor dalam ML.

Dalam kasus biner, sum domain pembangun,+, membangun domain A + B dari domain A dan B.

Jika a adalah elemen dari A dan b adalah elemen dari B maka (A,a) dan (B,b) adalah unsur-unsur A + B

$$A + B = \{(A,a) \mid a \text{ in } A\} \text{ union } \{(B,b) \mid b \text{ in } B\}$$

Dimana A dan B disebut tags dan digunakan untuk membedakan pendukung elemen A dan pendukung elemen B.

Sum domain disebut juga disjoint union atau co-product domain. Didalam Pascal disebut record varian dan dalam C disebut suatu struktur.

Sum domain : D_0 +…+ D_n
Assembly operations: (Di, di) in D_0 +…+ D_n and D_0 +…+ D_n = union_i = 0 n $\{(Di,d) \mid d \text{ in } Di\}$
Disassembly operations: Di(Di,di) = di

Function Domain

Domain-domain yang dibangun oleh pembangun function domain disebut fungsi di dalam Haskell, prosedur di dalam Modula-3 dan proses dalam SR.

Pembangun function domain membentuk domain A → B dari domain A dan B. domain A → B terdiri dari semua fungsi A ke B. A disebut domain dan B disebut co-domain.

(lamda x.e) adalah suatu elemen di A → B dimana e adalah suatu ungkapan yang berisi kejadian dari suatu identifier x, dimana a adalah nilai A merubah kejadian x di dalam e, nilai e[a:x] menghasilkan B.

Function Domain : A → B
Assembly operation: (lamda x.E) in A → B where for all a in A, E[x:a] is a unique value in B
Disassembly operation: (g a) in B, for g in A → b and a in A

Power Domain

Teori set yang menyediakan suatu notasi untuk uraian perhitungan. setL adalah suatu bahasa pemrograman yang didasarkan pada kumpulan (set) dan digunakan untuk menyediakan compiler ADA. Pascal menyediakan operasi set union dan intersection.

Kumpulan dari semua subset dari set adalah power set digambarkan sebagai berikut:

$$P^S = \{ s \mid s \text{ is a subset of } S\}$$

Subtypes dan subranges adalah contoh dari pembangun power set.

Beberapa bahasa menyediakan mekanisme untuk dekomposisi suatu jenis ke dalam subtypes. Enumerasi adalah elemen dari subtype yang lainnya adalah subranges.

Power domain membangun suatu domain dari elemen-elemen set. Untuk domain A, pembangun power domain P() menciptakan domain P(A), suatu kumpulan yang anggotanya adalah subset dari A.

Power Domain : PD
Assembly operations: ø in PD, {a} in PD for a in D, and Si union Sj in PD for Si, Sj in PD

Recursively Defined Domain

Recursively defined domain adalah domain yang didefinisikan dari bentuk

D : … D …

Definisi disebut Recursively sebab nama domain "recurs" pada sisi kanan dari definisi. Recursively defined domain tergantung pada abstrak karena nama domain adalah suatu bagian penting dari definisi domain.

Lebih dari satu set boleh mencukupi suatu recursively defined. Bagaimanapun, mungkin saja ditunjukkan bahwa suatu recursively defined selalu mempunyai solusi terkecil. Solusi terkecil adalah suatu subset solusi yang lain.

Limit Construction
D_0 = null
D_{i+1} = e[D:D_i] for i = 0,…
D = $\lim_i \rightarrow {}_{infty}\ D_i$

Type System

Prosentase besar kesalahan di dalam program adalah dalam kaitan dengan operasi ke object jenis yang bertentangan. Type system telah dikembangkan untuk membantu programmer dalam pendeteksian kesalahan.

Suatu type system adalah satu set aturan untuk mendefinisikan jenis dan menghubungkan suatu type dengan ekspresi dalam bahasa. Suatu type system menolak suatu ekspresi jika tidak menghubungkan suatu type dengan ekspresi. Type checking boleh berjalan pada waktu kompilasi atau waktu berjalan atau kedua-duanya.

Jika kesalahan diharapkan untuk dideteksi pada waktu kompilasi maka suatu static type checking system diperlukan. Satu pendekatan ke static type checking memerlukan programmer untuk menetapkan type masing-masing obyek di dalam program.

Ini mengizinkan compiler untuk melaksanakan type checking sebelum pelaksanaan program dan ini adalah pendekatan yang diambil oleh bahasa seperti Pascal, ADA, C++ dan Java.

Jika pendeteksian kesalahan diharapkan untuk ditunda sampai waktu pelaksanaan, maka dinamic type checking diperlukan.

Di dalam dinamic type checking, masing-masing nilai data berlabel dengan type informasi sehingga lingkungan waktu berjalan dapat memeriksa kecocokan type dan mungkin melaksanakan konversi type jika diperlukan. Bahasa program Lisp, Scheme dan Small-Talk adalah contoh dari bahasa dynamic type.

Type Checking

Suatu bahasa disebut :

- Untyped jika tidak ada type abstrak yang berlaku
- Strong type jika menyelenggarakan type abstrak (operasi mungkin diterapkan hanya untuk type object yang sesuai)
- Type static jika type ekspresi masing-masing dapat ditentukan dari teks program
- Type dynamic jika penentuan type beberapa ekspresi tergantung pada perilaku waktu berjalan program.

Keuntungan dari bahasa Untyped adalah fleksibilitas mereka. Programmer mempunyai kendali penuh atas bagaimana suatu nilai data digunakan tetapi harus mengasumsikan tanggung jawab penuh untuk mendeteksi aplikasi operasi ke type object yang tidak cocok/bertentangan.

Strong type membantu untuk memastikan portabilitas dan keamanan kode dan sering memerlukan programmer dengan tegas menggambarkan type masing-masing object di dalam suatu program. Ini penting juga dalam kumpulan untuk pemilihan operasi yang sesuai dan untuk optimisasi.

Static Type secara luas dikenal sebagai kebutuhan untuk produksi software yang dapat dipercaya dan aman. Type static dipilih ketika efisiensi di

dalam waktu pelaksanaan adalah penting dan kompilator pendukung digunakan untuk mendukung rancang bangun software berjalan.

Dynamic type checking menyiratkan bahwa type dicek pada waktu pelaksanaan dan bahwa tiap-tiap nilai berlabel untuk mengidentifikasi typenya dalam rangka membuat type checking mungkin. Hukuman untuk dynamic type checking adalah biaya waktu dan ruang tambahan.

Type Equivalence (Kesamaan Jenis)

Dua type tak dikenal (satuan object) adalah sama jika mereka berisi elemen-elemen yang sama. Yang sama tidak bisa dikatakan type nama mereka yang dulu, maka tidak diperlukan untuk memisah type union. Kapan type dinamai, ada dua pendekatan utama untuk menentukan apakah dua type sama.

Name Equivalence (kesamaan nama)

Di dalam name equivalence dua type adalah sama jika mereka mempunyai nama yang sama. Type diberi nama berbeda diperlakukan berbeda dan tidak bisa secara kebetulan dicampur hanya karena struktur mereka secara kebetulan adalah sama. Name equivalence perlu definisi type untuk global.

Structural Equivalence (kesamaan struktural)

Di dalam structural equivalence, nama type diabaikan dan elemen-elemen type dibandingkan untuk persamaan. Adalah mungkin bahwa dua type logic yang berbeda boleh menjadi kebetulan yang sama dan dapat dicampur.

Definisi type tidak diperlukan untuk menjadi global. Structural equivalence adalah penting di dalam distribusi pemrograman, dimana program terpisah harus mengkomunikasikan type data.

Definisi N.1:

Dua type T, T’ adalah name equivalence iff T dan T’ adalah nama yang sama.

Dua type T, T’ adalah structural equivalence iff T dan T’ memiliki satuan nilai yang sama.

Tiga aturan berikut yang digunakan untuk menentukan jika dua type adalah structural equivalence :

a. suatu nama type sama secara struktur dengan dirinya sendiri
b. dua type yang sama secara struktur jika mereka dibentuk dengan menerapkan type pembangun yang sama (secara berulang) ke type structural equivalence.
c. Setelah suatu deklarasi type, type n = T, nama type n secara structural setara dengan T.

Type Inference (jenis kesimpulan)

Type inference adalah masalah yang umum dalam menjelmakan untyped atau sintaksis type parsial ke dalam terminologi yang baik.

Deklarasi tetap Pascal adalah suatu contoh type inference, typa nama adalah kesimpulan dari type yang tetap. Dalam Pascal untuk pengulangan type index pengulangan dapat ditarik kesimpulan dari type recursively defined dan dengan begitu indeks pengulangan harus suatu variabel lokal dari pengulangan.

Bahasa pemrograman Miranda dan Haskell adalah type static dan menyediakan strong type inference system sehingga seorang programmer tidak perlu mendeklarasikan type apapun. Bahasa juga mengizinkan para programmer untuk menyediakan spesifikasi type eksplisit.

Suatu type checking harus mampu :

a. menentukan jika suatu program adalah type yang baik dan
b. jika program adalah type yang baik, tentukan type ekspresi manapun di dalam program

Type Declaration (jenis deklarasi)

Bahkan bahasa yang menyediakan suatu type inference system mengizinkan para programmer untuk membuat deklarasi type eksplisit. Sekalipun compiler dapat dengan tepat menyimpulkan type, pembaca manusia mungkin harus meneliti beberapa halaman kode untuk menentukan type suatu fungsi.

Kesalahan kecil oleh programmer dapat menyebabkan compiler mengeluarkan pemberitahuan kesalahan atau untuk menyimpulkan suatu type yang berbeda dibanding yang diharapkan. Karena pertimbangan ini adalah praktek pemrograman yang baik dengan tegas menyatakan type atas semua kecuali kasus yang paling nyata.

Polymorphism

Suatu type system adalah monomorphic jika masing-masing konstanta, variabel, parameter, dan hasil fungsi mempunyai suatu type unik. Type checking suatu system monomorphic adalah type secara langsung. Tetapi system type monomorphic semata-mata tidak memuaskan untuk penulisan software yang bisa dipakai kembali.

System yang sepenuhnya monomorphic jarang. Kebanyakan bahasa pemrograman berisi beberapa operator atau prosedur yang mengizinkan argumentasi lebih dari satu type.

Definisi N.2 :

Monomorphism : tiap-tiap konstanta, variabel, parameter, fungsi dan operator mempunyai suatu type unik.

Pemuatan lebih mengacu pada penggunaan dari sintaksis pengenal tunggal untuk mengacu pada beberapa operasi berbeda yang dibedakan oleh type dan jumlah argumentasi pada operasi.

Polymorphism : suatu operator, fungsi atau prosedur yang mempunyai suatu keluarga type yang terkait dan berorientasi secara seragam atas argumentasinya dengan mengabaikan type.

Suatu operasi polymorphic adalah yang dapat berlaku untuk type yang berbeda tetapi berhubungan dengan argumentasi.

Suatu type system adalah polymorphic jika abstrak beroperasi secara seragam pada argumentasi suatu keluarga type terkait.

Polymorphism type ini kadang-kadang disebut parametric polymorphism.

Type Completeness (jenis kelengkapan)

Prinsip type ini, tidak ada operasi yang dapat berlaku semaunya, terbatas yang berhubungan dengan nilai type.