Specificații formale ale produselor software

Prof.dr.Victor-Valeriu PATRICIU, asist.ing.Cristian CODREANU
Catedra de Calculatoare, Academia Tehnică Militară, București

Lucrarea prezintă unele aspecte ce țin de specificarea cerințelor unui proiect software. Specificațiile eronate pentru un produs software compromit realizarea acestuia, sau duc la o comportare neconformă cu scopul inițial. Corectarea unor deficiențe de specificare pe parcursul derulării proiectului comportă un volum de muncă foarte mare, de aceea ideal este ca specificația produsului să fie facută corect de la început. Limbajul Z permite scrierea de specificații de produse independent de suportul de implementare, permițând testarea consistenței și verificarea proprietăților impuse produsului. A doua parte a lucrării schițează descrierea algoritmului DES (Data Encription Standard) în limbajul Z și demonstrează o proprietate a algoritmului, pe baza calculului cu predicate.
Cuvinte cheie: *criptografie computațională, limbaj Z, teoria multimilor, logica predicatelor, specificatii formale pentru produse software.*

1. Prezentarea limbajului Z

Folosirea limbajului natural pentru descrierea specificațiilor de produs este inadecvată, datorită lipsei de precizie și ambiguității. Limbajele de programare, deși permit o descriere precisă și neambiguă a unei probleme, realizează acest lucru dependent de o platformă de lucru.
Pentru o descriere independentă a specificațiilor este nevoie de o notație matematică, bazată pe teoria multimilor, logica predicatelor, relații și funcții.
O specificatie scrisă în limbajul Z este o combinație între enunțuri matematice formale și text explicativ. Ambele aspecte sunt importante: partea formală conferă o descriere precisă a sistemului (fundamentată matematic), în timp ce partea explicativă face descrierea mai lizibilă și ușor de înțeles [2][3][4].
O specificație de produs scrisă in limbajul Z este bazată pe stări. Sistemul (produsul) este descris pe baza unor stări interne. Funcționarea lui este specificată prin tranziții între diferite stări, în condiții bine definite. Structura unei specificații scrise in Z este, în linii mari, următoarea:

- definirea entităților globale,
- definirea elementelor de stare (spațiul stărilor),
- definirea operațiilor în cadrul sistemului.

1. Definirea entităților globale

Entitățile globale stabilesc toate tipurile ce vor fi folosite în specificație, pe baza cărora vor fi declarate variabile. Stabilirea de tipuri se poate face:

- prin declarare de tipuri de bază (basic types);
- prin declarare de tipuri libere (free types).

Tipurile de bază se definesc prin enumerare, fără a se specifica structura internă a acestora. De exemplu, în specificația formală a unui sistem de gestiune a unei biblioteci, tipurile de bază ar putea fi:

[Book, Copy, Person, Author, Subject]

Tipurile libere sunt folosite în general pentru definirea tipurilor cu un număr mic de elemente nominale. În cazul specificației de mai sus, definirea mesajelor (de execuție, de eroare etc) se poate face folosind un tip liber:

```
Report ::= Okay
         | NotAMember
         | NotRegistered
         | UnknownBook
         etc.
```

Tot în secțiunea de entități globale pot fi definite variabile globale definite axiomatic, adică specificând un domeniu de valori restrictiv.

De exemplu, | MaxCopyAllowed : N specifică faptul că variabila MaxCopyAllowed va avea doar valori numere naturale. În mod implicit, tipul variabilei MaxCopyAllowed este tipul N.

2. Definirea spațiului stărilor

Spațiul stărilor este dat de totalitatea variabilelor de stare, cu alte cuvinte de mulțimea variabilelor globale pentru care orice schimbare semnifică o modificare a stării sistemului. Dacă specificația descrie un sistem de dimensiuni reduse, spațiul stărilor poate fi definit într-o singură schemă. Pentru sisteme complexe, spațiul stărilor este descris prin mai multe scheme, care vor putea fi combinate folosind calculul cu scheme.
Toate acțiunile sistemului vor determina tranziții de stare. Pentru ca specificația să fie consistentă, trebuie definită și o stare inițială, care determină valorile de plecare ale variabilelor de stare.

3. Definirea operațiilor

Operațiile reprezintă acțiunile ce pot să apară în cadrul sistemului, producând tranziția acestuia dintr-o stare în alta. În cazuri particulare, operațiile pot lăsa sistemul în aceeasi stare (de ex. operațiile de raportare mesaje). Din punct de vedere al tipului de operație descrisă, schemele sunt:
- scheme ce definesc operații propriu-zise (cu pre-condiție și post-condiție),
- scheme ce definesc tratarea erorilor (pre-conditie nesatisfăcută),
- scheme ce definesc operații robuste (operații propriu-zise plus tratarea erorilor).

4. Consistența specificației și verificarea proprietăților

După cum am arătat, limbajul Z conține elemente de teoria mulțimilor și de logică matematică. Cu ajutorul calculului cu predicate se verifică consistența specificației și se demonstrează existența unor proprietăți ale sistemului. O specificație este consistentă dacă definițiile din cadrul acesteia nu sunt contradictorii. Verificarea proprietăților specificației se face pe baza definițiilor date în specificație și a teoremelor de derivare și/sau echivalența.

5. Specificații formale asistate de instrumente software

În prezent există preocupări pentru automatizarea procesului de scriere de specificații formale, încurajate și de standardul comercial privind notația Z. Scrierea specificațiilor formale este abordată în general cu extensii ale editoarelor tip LATEX. Consistența și verificarea proprietăților intră în domeniul inteligenței artificiale, existând în prezent module implementate în Prolog.

II. Specificație formală a algoritmului de criptare DES

Vom demonstra utilitatea si eleganta descrierii Z pentru algoritmul criptografic DES [1][5]. O descriere formală a DES-ului permite evidențierea proprietăților mesajului cifrat relativ la mesajul clar și la cheia de cifrare. Enunțarea acestor proprietăți pe baza tranzițiilor de stare permite amânarea detaliilor de proiectare pâna la faza de implementare.

1. Specificație de nivel înalt

La acest nivel de abstractizare, descrierea algoritmului DES se face prin evidențierea relațiilor dintre mesajul clar, cheia de cifrare și mesajul cifrat.

[MESSAGE, KEY]

CIPHER

encipher : (MESSAGE x KEY) → MESSAGE
decipher : (MESSAGE x KEY) → MESSAGE

SIMMETRY CIPHER

$\forall$ m : MESSAGE; k : KEY o
decipher (encipher (m, k), k) = m

SECURITY CIPHER

$\forall$ m : MESSAGE; k, j : KEY o
$\neg$(decipher (encipher(m, k), j) = m$\circledcirc$(j$\neq$k))
m $\neq$ encipher(m, k)

Schema CIPHER declară funcțiile de criptare și decriptare, cu domeniul obținut prin produsul cartezian între mulțimea mesajelor și mulțimea cheilor de cifrare, și codomeniul mulțimea mesajelor. La acest nivel nu se fac specificații asupra structurii funcțiilor.
Formularea completă a DES la acest nivel este dată de conjuncția schemelor:
DES≡CIPHER$\wedge$SIMMETRY$\wedge$SECURITY.

Tipurile folosite au la bază definiția polimorfă BITSEQ[n].

BITSEQ[n] == {f : seq[Bool] | #f = n}
BLOCK == BITSEQ[64]
KEY == BITSEQ[64]
HALFBLOCK== BITSEQ[32]
SUBKEY == BITSEQ[48]
USEKEY == BITSEQ[56]
MESSAGE == seq[Bool]

2. Specificație detaliată

La acest nivel se vor face specificații asupra structurii funcțiilor de cifrare și decifrare.

Schema PERM formalizează funcția de permutare. Această schemă stă la baza descrierii diferitelor operații de permutare din cadrul algoritmului.

PERM

p : seq[N]
n : N

p = n
ran p = 1 .. n

Implementarea funcțiilor de permutare inițială și finală este dată de definiția și schema următoare:

iperm : PERM

iperm.n = 64
iperm.p = <58,50,...,7>

INITIAL_PERMUTATION

IP : BLOCK $\rightarrow$ BLOCK
IP^{-1} : BLOCK $\rightarrow$ BLOCK

$\forall$ b : BLOCK
IP(b) = iperm.p ; b
$IP^{-1}(b) = (iperm.p)^{-1}$; b

Permutarea PC1 transpune cheia de 64 biți în cheia de 56 biți. Permutarea PC2 concatenează cele două secvențe de 28 biți și le transpune în chei de câte 48 biți, necesare în cele 16 runde.

pc1perm : PERM
pc2perm : PERM
---
pc1perm.n = 56
pc2perm.n = 48
pc1perm.p = <57,49,...,4>
pc2perm.p = <14,17,...,32>

KEY PERMUTE

PC1 : KEY → USEKEY
PC2 : USEKEY → SUBKEY

∀ k : KEY o PC1(k) = pc1perm.p ; k
∀ k : USEKEY o PC2(k) = pc2perm.p ; k

Pe baza schemei KEY_PERMUTE şi a definiţiei de mai sus, este descrisă schema de formare a cheilor de cifrare şi decifrare, pentru cele 16 runde ale algoritmului.

KEYS

KEY_PERMUTE
encipherkeys : seq[SUBKEY]
decipherkeys : seq[SUBKEY]
KS : (N x seq[Bool]) → seq[Bool]
PK : N → USEKEY
k : KEY

∀ n : N; s : seq[Bool] o
1 ≤ n ≤ 16
KS(n, s) = LS(left(s), keyschedule(n)) ^ LS(right(s), keyschedule(n))
PK(1) = KS(1, PC1(k))
∀ n : 2..16 o PK(n) = KS(n, PK(n-1))
# enciphers = 16
encipherkeys = PK ; PC2
decipherkeys = rev(encipherkeys)

Definirea completă a schemelor care implementeaza restul funcţiilor folosite în algoritmul DES se poate face în aceeaşi maniera. Formalizarea finală a DES-ului, în varianta detaliată, este dată de schemele următoare:

BLOCK_DES

ROUNDS
KEYS
INITIAL_PERMUTATION
encrypt : BLOCK → BLOCK
decrypt : BLOCK → BLOCK
---
encrypt = IP ; P(encipherkeys) ; R^{15} ; F ; IP^{-1}
decrypt = IP ; P(decipherkeys) ; R^{15} ; F ; IP^{-1}

DES

BLOCK_DES
encipher : (MESSAGE x KEY) → MESSAGE
decipher : (MESSAGE x KEY) → MESSAGE

∀ m : MESSAGE o
∃ blocks : seq[BLOCK] o^/ blocks = m
encipher(m, k) = ^/ (map encrypt blocks)
decipher(m, k) = ^/ (map decript blocks)

3. Consistența și demonstrarea proprietăților

Vom considera proprietatea : *permutarea finală este inversul permutării inițiale*, cu transcrierea matematică: $IP; IP^{-1} = id_{BLOCK}$ Raționamentul următor este bazat pe calculul cu predicate și pe definițiile din cadrul specificației algoritmului DES.

∀ b : BLOCK

1. $IP^{-1}(IP(b)) = b$	def lui ; si id
2. $IP^{-1}(iperm.p ; b) = b$	1, def lui IP
3. $(iperm.p)^{-1} ; (iperm.p ; b) = b$	2, def IP^{-1}
4. $((iperm.p)^{-1} ; iperm.p) ; b = b$	3, asoc lui ;
5. $(id(ran\ iperm.p)) ; b = b$	4, legea ;
6. $id_{dom\ b} ; b = b$	5, dom PERM = ran PERM
7. b = b	6, legea ;
8. True	7, legea =

q.e.d.

III. Concluzii

Descrierea corectă și neambiguă a unui proiect software se poate face cu ajutorul unui limbaj de specificație formală. Limbajul Z, descris ca standard, permite realizarea de specificații bazate pe tranziția stărilor. Specificația unui produs se poate face la diferite nivele de abstractizare, independent de implementarea specifică. Asupra specificației formale se pot face verificări privind consistența și demonstrații de proprietăți. Abia după încheierea etapei de verificare a corectei specificații a algoritmilor, se poate trece la implementarea secvențială, paralelă sau distribuită a produsului.

Bibliografie

1. Philip Venables, *Formal Specifications and Parallel Implementation of the Data Encryption Standard*, Oxford University Computing Lab., Sept 1990
2. B. Potter, J. Sinclair, D. Till, *An Introduction to Formal Specification and Z*, Prentice Hall, 1991
3. ISO Panel JTC1/SC22/WG19 (Rapporteur Group for Z), *Z Standard - Part 1 & 2*, Dec 1996
4. D. Rann, J. Turner, J. Whitworth, *Z - a beginner's guide*, Chapman & Hall, 1994
5. Federal Information Processing Standards, *The Data Encryption Standard*, Publication 46, 1977