Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike i računarstva

Ivan Miličić

Formalna specifikacija i Z jezik

Seminarski rad

Osijek, 2018.

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike i računarstva

Ivan Miličić

Formalna specifikacija i Z jezik

Seminarski rad

Kolegij: Softversko inženjerstvo
Voditelj kolegija: doc. dr. sc. Alfonzo Baumgartner

Osijek, 2018.

Sadržaj

Sažetak

Pojam „formalne metode" odnosi se na bilo koju aktivnost koja se oslanja na matematički zapis. Formalna specifikacija samo je jedna od aktivnosti formalnih metoda. Neke od koristi korištenja ovakve specifikacije su lakše uočavanje grešaka čak i na početku razvoja, smanjenje cijene sustava, bolje razumijevanje rada sustava, preciznost i jednoznačnost zapisa te mogućnost izrade dobre dokumentacije sustava. Još jedna od aktivnosti formalnih metoda je i dokaz koji u određenim situacijama garantira ispravnost rada sustava pa se koristi u sustavima koji su kritični za sigurnost. Jasno je da dokazi povećavaju cijenu sustava pa bi se trebali koristiti samo onda kada je to od iznimne važnosti.

Postoje različiti jezici za formalnu specifikaciju sustava. Jedan od njih je i Z jezik koji je razvijan na Sveučilištu u Oxfordu od 1970-tih godina od strane članova grupe za programsko istraživanje (Programming Research Group). Zasniva se na logici prvog reda i teoriji skupova. Glavno obilježje ovog jezika je da svaki podatak ima svoj tip. Treba napomenuti da postoje i različiti alati koji pomažu u izradi specifikacije, pa čak i komercijalni, a također i softveri za dokazivanje teorema.

Ključne riječi

Formalne metode, formalna specifikacija, dokaz, Z jezik za specifikaciju

Uvod

U ovom radu općenito ću opisati formalne metode te Z jezik za formalnu specifikaciju. U prvoj sekciji dan je općeniti tekst o formalnim metodama i prednostima matematičke notacije. U drugoj sekciji predstavljeni su neki mitovi koji nastoje opovrgnuti neka kriva mišljenja koja postoje o formalnim metodama. U trećoj sekciji dan je kratak pregled Z jezika zajedno s dva primjera: uređaj za računanje kvadratnog korijena i Display editora.

U općenitom dijelu ovog rada (prva i druga sekcija) korištene su knjige *Formal Specification and Documentation using Z: A Case Study Approach* (vidi [2]) i *Formal Specification Using Z* (vidi [1]). U trećoj sekciji primjeri su uzeti iz *Justification of Formal Methods for System Specification* (vidi [4]) i *Formal Specification of a Display-oriented Text Editor* (vidi [3]).

1 Formalne metode

Pojam „formalne metode" odnosi se na bilo koju aktivnost koja se oslanja na matematički zapis. Formalne metode uključuju formalnu specifikaciju sustava, analizu specifikacije i dokaz te razvoj i verifikaciju programa. Formalna specifikacija je opis sustava korištenjem matematičke notacije što je upravo i njena glavna mana. Stoga, da bi se koristila, treba najprije naučiti jezik za specifikaciju, a potom i steći određeno iskustvo prije nego se može izvući sva korist. S druge strane, prednosti matematike, pa samim tim i prednosti njezinog korištenja u speficikaciji sustava su sljedeće:

Preciznost: Matematički izrazi su precizni jer se oslanjaju samo na minimalne osnove i ne zahtjevaju kontekst. Sve potrebne informacije su matematički zapisane, a one nepotrebne se izostavljaju.

Sažetost: Matematički su izrazi često vrlo sažeti: značenje izraza prikazano je relativno malenim brojem simbola za razliku od opisa tog izraza prirodnim jezikom. To je jedna od prednosti kod opisivanja kompleksnih sustava.

Jasnoća: Matematičke forme su jasne jer ne ovise o kulturi pojedinog naroda. Za ispravnu se formulaciju može lako dokazati da je ispravna, a eventualne greške su lako uočljive te se isto tako brzo i ispravljaju.

Apstrakcija: Kod težih problema, prvo se promatraju samo osnovne stavke i njih se rješava, a kasnije se pridodaju i preostale pretpostavke.

Neovisnost o prirodnom jeziku: Matematiku može jednako dobro razumjeti bilo koji čovjek, neovisno o njegovom jeziku i kulturi.

Dokazi: Mogućnost dokazivanja tvrdnji je važan aspekt primjene matematike koji ima veliku primjenu u sustavima koji su kritični za sigurnost čovjeka.

Također, tu su i mnoge druge blagodati korištenja formalne specifikacije:

- povećava razumijevanje kako sustav radi, pogotovo u početku oblikovanja,
- vjerojatnost pogreške u oblikovanju je minimalna,
- povećava kvalitetu dokumentacije,
- smanjuje ukupnu cijenu softvera.

Unatoč svemu navedenom, formalne se metode ne koriste tako puno u izradi softvera zato što mnoge softverske tvrtke nisu htjele riskirati uvođenjem formalnih metoda u njihove razvojne procese jer ih nisu smatrali učinkovitima. No, kod organizacija koje jesu investirale u formalne metode se smanjio broj grešaka u isporučenom softveru bez da se povećala cijena razvoja tog softvera. Dakle, formalne metode mogu biti učinkovite ako se primjenjuju u glavnim dijelovima sustava i ako tvrtke žele uložiti u ovu tehnologiju.

2 Neki mitovi

Mit 1: Formalne metode garantiraju da je softver ispravan.

Kod svake tehnike izrade softvera može doći do greške, tj. niti jedna od njih nije nepogrešiva pa tako ni formalne metode. Čak i ako se ispravno dokaže neki matematički teorem o ispravnosti softvera, postoji mogućnost da je korišten krivi matematički model.

Mit 2: Formalne metode rade tako da dokažu da program radi ispravno.

Najveća korist formalnih metoda nije u dokazivanju, nego u formalnoj specifikaciji sustava jer se pri tom procesu uviđaju greške. Doduše, i dokazi teorema su korisni, ali ne moraju se izvoditi kod sustava koji nisu toliko kritični i za koje se hoće smanjiti ukupna cijena.

Mit 3: Samo najkritičniji sustavi imaju korist od korištenja formalnih metoda.

Formalne su se metode koristile u različitim vrstama sustava pri čemu su neki od njih bili kritični, a drugi ne.

Mit 4: Formalne metode koriste složenu matematiku.

Matematika koja je potrebna i poželjna za formalnu specifikaciju je na razini matematike koja bi se mogla učiti u školi jer njezin cilj je da bude što lakše razumljiva. Unatoč tome, mnogi nisu imali takvo obrazovanje u prošlosti.

Mit 5: Formalne metode povećavaju cijenu razvoja softvera.

Teoremi i njihovo dokazivanje može povećati cijenu, ali sâma formalna specifikacija ne zato što se korištenjem formalne specifikacije mogu vrlo rano otkriti greške koje je zbog toga lako otkloniti nego kada bi se te greške uočile kasnije te bi ih zbog toga bilo puno teže uočiti.

Mit 6: Klijent ne može razumjeti specifikaciju.

To je istina, no formalna specifikacija može pomoći u izradi puno bolje dokumentacije sustava koja je razumljiva i klijentu.

Mit 7: Nitko ih ne koristi u stvarnim projektima.

Postoji dosta stvarnih primjera u kojima su bile korištene formalne metode i u njima su se pokazale kao vrlo korisne. Dva primjera u kojima je bio korišten Z su Inmos Transputer Floating Point Unit microcode design i IBM CICS Transaction Processing System koji su oba osvojili UK Queen's Award.

Mit 8: Formalne metode usporavaju razvojni proces.

Previše korištenja formalnih metoda zaista može usporiti cijeli proces, a također i dokazivanje koje u nekim slučajevima nije od koristi. No, korištenje formalne specifikacije kao dio razvojnog procesa može biti od koristi te smanjiti vrijeme i cijenu softvera.

Mit 9: Nemaju alate.

Danas postoje određeni alati koji podržavaju formalne metode, a neki od njih se ozbiljno koriste u industriji, za različite jezike. Također, postoje i softveri za dokazivanje teorema, primjerice EVES, HOL, LP.

Mit 10: Korištenje formalnih metoda znači odbacivanje tradicionalnih metoda oblikovanja softvera.

Formalne metode ne bi se trebale koristiti kao zamjena za postojeće razvojne procese, nego bi se trebala uvesti i koristiti zajedno s drugim metodama. Također, način na koji će se određena formalna metoda primijeniti ovisi i o samom problemu.

Mit 11: Mogu se primijeniti jedino na razvoj softvera.

Postoje primjeri njihovog korištenja i u hardveru. Jedan od glavnih primjera je Inmos Transpuster. Z jezik se također koristio i u speficikaciji instrukcija za mikroprocesor, osciloskop i neke druge (viper microprocessor, cache coherence protocol, real-time kernel).

Mit 12: Nisu potrebne.

Standardi će zahtjevati, ili barem jako preporučiti, korištenje formalnih metoda za sustave s visokim integritetom, kao što su sustavi kritični za sigurnost.

Mit 13: Nisu podržane.

Postoje mnoge knjige o formalnim metodama, konferencije, tečajevi, a postoje i tvrtke koje se specijaliziraju u formalne metode (B-Core Limited, Computational Logic Inc., CRI, DST, IFAD,...) Postoji i niz komercijalnih alata (FDR model checker, LAMBDA toolset, ProofPower by ICL,...).

Mit 14: Formalni ljudi koriste formalne metode.

Iako je preporučljivo koristiti formalne metode, one nisu uvijek prikladne. Čak i oni koji ih jako dobro poznaju ne upotrebljavaju ih stalno.

3 Z jezik

Z je jezik u kojem podaci imaju tipove, a baziran je na teoriji skupova i predikatnoj logici prvog reda. Jedan od uvjeta Z jezika je da svako ime koje označava neku vrijednost mora biti deklarirano. U nastavku je dano osnovno o Z jeziku što je korišteno u ovom radu. Više o samom jeziku može se pogledati u [1]. Sâm Z jezik se može specificirati samim sobom (vidi [6]).

3.1 Osnove Z jezika

Od ugrađenih tipova, Z posjeduje isključivo cijele brojeve $\mathbb{Z}$. Za podatak n koji je cijeli broj pišemo $n : \mathbb{Z}$. Nadalje, prirodni brojevi se vrlo često koriste pa se definiraju kao cijeli brojevi uz uvjet da su nenegativni. Dakle, $\mathbb{N}$ nije pravi tip, nego podskup tipa $\mathbb{Z}$, pa je stoga $n : \mathbb{N}$ zapravo tipa $\mathbb{Z}$ uz uvjet $n \geq 0$. Što se tiče realnih brojeva, ako su potrebni u specifikaciji, može ih se definirati (vidi [5], stranice 9-36).

U Z-u je moguće definirati vlastite tipove. Takve tipove zovemo osnovni tipovi ili dani skupovi. Oni se deklariraju bez razmatranja o tome kakvi su zaista njegovi elementi. Primjerice, skup svih gradova na svijetu možemo definirati s $[Gradovi]$ i tada za neki grad pišemo $Osijek : Gradovi$.

Uz vlastite tipove, možemo koristiti i slobodne tipove. Njih koristimo onda kada možemo nanizati sve njegove elemente. Primjerice, $Operacije ::= plus \mid minus \mid puta \mid podijeljeno$.

U slučaju da su nam potrebni znakovi, možemo ih definirati kao osnovni tip ili vlastiti tip.

Specifikacijski dokument u Z-u sastoji se od opisa pisanog u prirodnom jeziku i notacije pisane u Z-u. Kako bi se jasno odijelio tekst od specifikacije, koristi se grafički format po imenu shema s kojima se mogu izvoditi i logičke operacije. Sheme također pomažu da specifikacija ostane jednostavna i čitljiva te da postigne određenu strukturu. Shema se zapisuje na sljedeći način:

S

$$a, b : \mathbb{N}$$

$$a < b$$

pri čemu prvi dio služi za deklaraciju, a drugi dio za predikat (logički izraz koji povezuje varijable). Ime sheme je S i ona deklarira dvije varijable a i b uz uvjet $a < b$. Shema se može zapisati i u linearnom obliku:

$$S \mathrel{\widehat=} [a, b : \mathbb{N} \mid a < b].$$

Sve varijable koje su deklarirane u nekoj shemi su lokalne varijable te sheme pa im se iz druge sheme može pristupiti jedino ako tu shemu uključimo. Primjerice, ako hoćemo u shemi *IncludeS* koristiti varijable sheme S, tada bismo to postigli na sljedeći način:

IncludeS

$$c : \mathbb{N}$$
$$S$$

$$c = a + b$$

što je ekvivalentno sljedećem zapisu:

IncludeS

$$c : \mathbb{N}$$
$$a, b : \mathbb{N}$$

$$c = a + b$$
$$a < b$$

Primijetimo da postoji mogućnost da smo u shemi *IncludeS* već imali definirane varijable istog imena kao i varijable u shemi S. U tom slučaju, one moraju pripadati istom tipu (i u jednoj, i u drugoj shemi) jer je u suprotnom ovakvo uključivanje sheme pogrešno.

Postoje i globalne varijable koje se uvode aksiomatskom definicijom. Primjerice,

$$n : \mathbb{Z}$$

$$n < 0$$

Na shemama možemo raditi različite logičke operacije. Za dvije sheme S i T definiramo $SandT \mathrel{\widehat=} S \wedge T$, $SorT \mathrel{\widehat=} S \vee T$, $SimplT \mathrel{\widehat=} S \Rightarrow T$, $SeqT \mathrel{\widehat=} S \Leftrightarrow T$, a značenje je sljedeće: deklaracija sheme *SandT* je unija deklaracija shema S i T, a predikat se dobije kao konjunkcija predikata shema S i T. Analogno za ostale.

Uz sve to, sheme koje definiraju neku operaciju koriste varijable označene s ? i !. Varijable označene s ? predstavljaju ulazne vrijednosti, a varijable označene s ! izlazne vrijednosti. Primjerice,

Add

$$a?, b? : \mathbb{Z}$$
$$sum! : \mathbb{Z}$$
$$sum! = a? + b?$$

Skupove u Z notaciji, ukoliko su konačni, možemo definirati tako da im nanižemo elemente: $Operacije = \{plus, minus, puta, podijeljeno\}$. Također je moguće koristiti predikat pomoću kojeg ćemo definirati skupove koji nisu nužno konačni. Primjerice, skup

$$P = \{n : \mathbb{N} \mid paran\, n \bullet n * n\}$$

je skup svih kvadrata parnih prirodnih brojeva. Općenito, takvi se skupovi definiraju na sljedeći način:

$$\{deklaracija \mid uvjet \bullet izraz\}$$

Često se $\bullet$ koristi i u značenju „takav da" kada se koriste kvantifikatori.

Brojevni niz $a \,..\, b$, gdje su $a, b : \mathbb{Z}$, je skup svih brojeva od a do b. Za konačan skup A s $\#A$ označavamo njegovu kardinalnost.

3.2 Relacije i funkcije

U Z-u se pojam relacija i funkcija definira kao i u teoriji skupova s nekim dodatnim oznakama i pojmovima. Dakle, relacija R između skupova P i Q je bilo koji podskup kartezijevog produkta tih skupova: $R \subseteq P \times Q$. Nadalje, $P \leftrightarrow Q$ je oznaka za skup svih relacija između skupova P i Q, tj. $P \leftrightarrow Q == \mathbb{P}(P \times Q)$ gdje je $\mathbb{P}$ oznaka za parcijalni skup skupa $P \times Q$. Za relaciju $R \in \mathbb{P}(P \times Q)$ pišemo $R : P \leftrightarrow Q$. Također, za neki element $(x, y) \in R$ pišemo i $x \mapsto y$ ili $x\ R\ y$.

Svaka relacija ima svoju domenu i sliku. Domena relacije $R : P \leftrightarrow Q$ je skup svih onih elemenata iz P koji su u relaciji s nekim elementom iz Q. Slično, slika relacije R je skup svih onih elemenata iz Q koji su u relaciji s nekim elementom iz P:

$$\mathrm{dom}\ R == \{x : P \mid (\exists\, y : Q \bullet (x, y) \in R)\}$$
$$\mathrm{ran}\ R == \{y : Q \mid (\exists\, x : P \bullet (x, y) \in R)\}$$

Slika skupa S pod djelovanjem relacije R, u oznaci $R(\!|\ S\ |\!)$, je skup svih elemenata iz slike te relacije u koje se preslika neki element iz S.

Nadalje, funkcije se definiraju kao poseban oblik relacija, no dijele se na parcijalne i totalne. Skup svih parcijalnih funkcija iz skupa X u skup Y označava se i definira kao:

$$X \pfun Y == \{f : X \leftrightarrow Y \mid (\forall\, x : X;\ y_1, y_2 : Y \bullet (x \mapsto y_1) \in f \land (x \mapsto y_2) \in f) \Rightarrow y_1 = y_2\}.$$

Parcijalne funkcije u Z-u su u matematici zapravo relacije sa svojstvom da se neki element iz domene preslika u točno jedan element kodomene. Dakle, nema uvjeta na to da se svaki element iz domene

mora preslikati. S druge strane, pojam totalne funkcije u Z-u odgovara matematičkoj definiciji funkcije. Stoga, skup svih totalnih funkcija $X \to Y$ je zapravo skup svih parcijalnih funkcija za koje vrijedi da je njihova domena jednaka skupu X.

$$X \to Y == \{f : X \pfun Y \mid \mathrm{dom}\ f = X\}$$

Uz parcijalne i totalne funkcije, koriste se i posebne oznake za injekcije, surjekcije i bijekcije te za konačne funkcije. U tablici 3.1 dane su sve vrste funkcija.

Vrsta funkcije	**Oznaka**
parcijalna funkcija	$\pfun$
parcijalna injekcija	$\pinj$
parcijalna surjekcija	$\psurj$
totalna funkcija	$\to$
totalna injekcija	$\inj$
totalna surjekcija	$\surj$
bijekcija	$\bij$
konačna parcijalna funkcija	$\ffun$
konačna parcijalna injekcija	$\finj$

Tablica 3.1: Popis svih vrsta funkcija i njihovih oznaka.

Posebna funkcija je identiteta na nekom skupu X, u oznaci $\mathrm{id}\ X$, koju definiramo kao

$$\mathrm{id}\ X == \{x : X \bullet x \mapsto x\}$$

U Z-u definiramo skup svih nizova čiji su elementi iz T kao:

$$\mathrm{seq}\ T == \{s : \mathbb{N} \ffun T \mid \mathrm{dom}\, s = 1 \mathinner{..} \#s\}$$

Primjerice,

$$s : \mathrm{seq}\ CHAR$$
$$s = \{1 \mapsto P, 2 \mapsto R, 3 \mapsto I, 4 \mapsto M, 5 \mapsto J, 6 \mapsto E, 7 \mapsto R\}.$$

pri čemu smatramo da je tip *CHAR* definiran kao znakovni tip. Koristimo i jednostavniju notaciju: $s = \langle P, R, I, M, J, E, R\rangle$. Za dva niza $s, t \in \mathrm{seq}\ T$ definiramo konkatenaciju tih nizova $s \frown t : 1 \mathinner{..} (\#s + \#t) \ffun T$ pri čemu je

$$j \mapsto \begin{cases} s(j) & 1 \le j \le \#s \\ t(j - \#s) & \#s < j \le (\#s + \#t) \end{cases}$$

Za nizove definiramo i operacije *head*, *last*, *tail*, *front*, *rev*. Primjerice, za niz $s : \mathrm{seq}\ CHAR$, $s = \langle J, E, D, N, O, S, T, A, V, N, O\rangle$ je

$$\begin{aligned} head\ s &= J \\ last\ s &= O \\ tail\ s &= \langle E, D, N, O, S, T, A, V, N, O\rangle \\ front\ s &= \langle J, E, D, N, O, S, T, A, V, N\rangle \\ rev\ s &= \langle O, N, V, A, T, S, O, N, D, E, J\rangle \end{aligned}$$

U nastavku ćemo pogledati dva primjera specifikacije pisana u Z jeziku.

3.3 Uređaj za računanje kvadratnog korijena

Pretpostavimo da kupac dolazi k nama sa sljedećim zahtjevom: „Izradi mi uređaj koji će računati kvadratni korijen." Samo na temelju ovoga ne bismo trebali odmah prihvatiti ugovor nego upitati kupca još neka pitanja. Primjerice: „Što je kvadratni korijen?"

Odgovor (od kupca): „To je broj koji pomnožen samim sobom daje ulaznu vrijednost."

Pitanje: „Koje skupove brojeva uzeti u obzir kao ulazne vrijednosti?"

Odgovor: „Pozitivne realne brojeve."

Na temelju ovih odgovora, možemo doći do sljedeće formalne specifikacije:

SQRT$_0$

$$
\begin{array}{l}
r? : \mathbb{R} \\
r! : \mathbb{R} \\
\hline
r? \geq 0 \land \\
(\exists\, t : \mathbb{R} \bullet \\
\quad r! = t \land \\
\quad t * t = r?)
\end{array}
$$

Pitanje: „Što ako se rezultat ne može konačno zapisati, kao u slučaju SQRT(3)?"

Odgovor: „U tom slučaju neka vrati dovoljno dobru aproksimaciju."

Pitanje: „Definiraj dovoljno dobru."

Odgovor: „Kvadrat rezultata se treba razlikovati od ulaznog podatka za manje od nekog malog nenegativnog broja."

SQRT$_1$

$$
\begin{array}{l}
r? : \mathbb{R} \\
\varepsilon? : \mathbb{R} \\
r! : \mathbb{R} \\
\hline
r? \geq 0 \land \\
\varepsilon? > 0 \land \\
(\exists\, t : \mathbb{R} \bullet \\
\quad r! = t \land \\
\quad r? - \varepsilon? \leq t * t \leq r? + \varepsilon?)
\end{array}
$$

Ni ovo nije dovoljno. Daljnja pitanja su cijena, propusnost, prezentacija... No, ova specifikacija sadrži neka specifična svojstva:

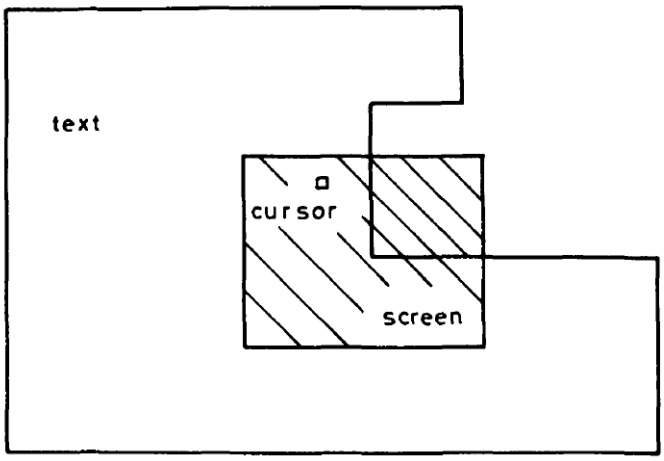
a) Definicija sadrži razumljive matematičke domene.

b) Razgovor s kupcem je zapisan u formalnoj notaciji iako ju kupac nije morao pročitati.

c) Iako je formalno opisana, specifikacija nije dovoljno uska, tj. program koji vraća negativan broj bi bio prihvatljiv kao i program koji vraća pozitivan, pa čak i program koji vraća i pozitivne i negativne vrijednosti.

d) Željeno ponašanje nije opisano algoritmom nego relacijom između ulaznih i izlaznih podataka.

e) Moguće je dokazati da algoritmi za računanje korijena zadovoljavaju ovu specifikaciju.



Slika 3.1: Neformalna specifikacija display editora. Operacije: MOVE, DELETE, INSERT, REPLACE... POČETAK/KRAJ RIJEČI, LINIJE, DOKUMENTA...

3.4 Display editor

Sada se okrećemo malo složenijem primjeru: display editor specificiran u Z-u Bernarda Sufrina. Neformalna specifikacija dana je na slici 3.1. Prekompleksno je da bismo vidjeli odakle početi. Tradicionalan pristup je tzv. top-down: započeti s jednostavnim opisom cijelog sustava, potom ga razdijeliti u međusobno povezane podsustave i pri tom dodavati detalje. Ovaj proces završava kada svaki list ovako konstruiranog stabla ne zahtjeva daljnu podijelu. Sufrinov pristup je drugačiji. On odbacuje većinu originalnog zahtjeva i traži jednostaniji, ali analogan sustav za koji se može konstruirati pripadna teorija. Potom se to poboljšaje kako bi se uzastopno dobile sve kompleksnije teorije sve dok se ne dospije do originalnog zahtjeva.

3.4.1 Sustav DOC

Najprije ćemo definirani jednostavan model editora u kojem ćemo trenutnu poziciju zvati pokazivač (kursor) i zabilježiti je kao par nizova znakova pri čemu prvi niz odgovara dijelu dokumenta koji je ispred pokazivača, a drugi koji je iza.

DOC

$$POS : \operatorname{seq} CHAR \times \operatorname{seq} CHAR$$

Nadalje, definirat ćemo funkcije za brisanje znaka lijevo od pokazivača, pomicanje pokazivača ulijevo te ubacivanje znaka lijevo od pokazivača.

$$
\begin{array}{|l}
\hline
del : POS \pfun POS \\
move : POS \pfun POS \\
ins : CHAR \to POS \to POS \\
content : POS \to \mathrm{seq}\, CHAR \\
\hline
\mathrm{dom}\, move = \mathrm{dom}\, del = \{l, r \mid l \neq \langle\rangle\} \\
(\forall (l, r) : POS;\ ch : CHAR) \\
move\ (l, r) = (front\ l, (last\ l) \frown r); \\
del\ (l, r) = (front\ l, r); \\
ins\ ch\ (l, r) = (l \frown \langle ch \rangle, r); \\
content\ (l, r) = l \frown r; \\
\hline
\end{array}
$$

Uočimo da su *del* i *move* parcijalne funkcije, tj. njihova domena je strogi podskup od *POS*.

Primjer 1. *Neka je trenutno stanje dokumenta: TRENUTNA pri čemu se pokazivač nalazi iza prvog slova N. Tada korištenjem gore definiranih operacija dobivamo:*

$$
\begin{gathered}
(\langle T, R, E, N \rangle, \langle U, T, N, A \rangle) \\
move \\
(\langle T, R, E \rangle, \langle N, U, T, N, A \rangle) \\
move \\
(\langle T, R \rangle, \langle E, N, U, T, N, A \rangle) \\
del \\
(\langle T \rangle, \langle E, N, U, T, N, A \rangle) \\
ins\ V \\
(\langle T, V \rangle, \langle E, N, U, T, N, A \rangle) \\
content \\
\langle T, V, E, N, U, T, N, A \rangle
\end{gathered}
$$

Vidimo da nedostaju akcije udesno. No, umjesto da definiramo funkcije *moveright*, *delright* i *insright*, možemo definirati metodu koja će definirati smjer.

$$
\begin{array}{|l}
\hline
mirror : POS \to POS \\
\hline
mirror\ (l, r) = (rev\ r, rev\ l) \\
\hline
\end{array}
$$

Primjer 2. *Promotrimo sljedeći niz akcija:*

$$(\langle P, R, I, \rangle, \langle M, J, E, R\rangle)$$
$$mirror$$
$$(\langle R, E, J, M\rangle, \langle I, R, P\rangle)$$
$$move$$
$$(\langle R, E, J\rangle, \langle M, I, R, P\rangle)$$
$$mirror$$
$$(\langle P, R, I, M\rangle, \langle J, E, R\rangle)$$

Uočavamo da smo postigli pomak udesno korištenjem samo mirror i move.

Lako se vidi da se na isti način mogu opisati brisanje i umetanje pa stoga definiramo sljedeće funkcije:

$$right, left : (POS \nrightarrow POS) \rightarrow (POS \nrightarrow POS)$$
$$right\ f = mirror \circ f \circ mirror$$
$$left\ f = f$$

Na taj način definirali smo funkcije

$$\begin{array}{llll} left\ move & left\ del & left\ (ins\ c) & \forall\, c : CHAR \\ right\ move & right\ del & right\ (ins\ c) & \forall\, c : CHAR \end{array}$$

koje imaju sljedeća svojstva:

1. Operacija ubacivanja znaka iza koje slijedi brisanje (s iste strane) ne mijenja dokument.
2. Dopušten pomak u jednom smjeru iza kojeg slijedi pomak u drugom smjeru ne mijenja dokument.
3. Dopušteno brisanje u jednom smjeru može se postići pomakom u tom smjeru iza kojeg slijedi brisanje u suprotnom smjeru.
4. Dopušten pomak nema utjecaja na sadržaj dokumenta.

Formalno ih zapisujemo na sljedeći način:

DOC PROPERTIES

$$\vdash (\forall\, c : CHAR)$$
$$\quad left\ del \circ left\ (ins\ c) = \mathrm{id}\ POS \land$$
$$\quad right\ del \circ right\ (ins\ c) = \mathrm{id}\ POS$$
$$\vdash left\ move \circ right\ move = \mathrm{id}\ \{l, r \mid r \neq \langle\rangle\} \land$$
$$right\ move \circ left\ move = \mathrm{id}\ \{l, r \mid l \neq \langle\rangle\}$$
$$\vdash left\ del \circ right\ move = right\ del \land$$
$$right\ del \circ left\ move = left\ del$$
$$\vdash (\forall (l, r) : POS \mid l \neq \langle\rangle)$$
$$\quad (content \circ left\ move)\ (l, r) = content\ (l, r)$$
$$\vdash (\forall (l, r) : POS \mid r \neq \langle\rangle)$$
$$\quad (content \circ right\ move)\ (l, r) = content\ (l, r)$$

Dokaz. Pokažimo $right\ del \circ right\ (ins\ c) = \mathrm{id}\ POS$. Neka je $(l, r) : POS$. Raspišimo najprije $right\ (ins\ c)\ (l, r)$:

$$\begin{aligned} right\ (ins\ c)\ (l, r) &= (mirror \circ ins\ c \circ mirror)\ (l, r) = mirror\ (ins\ c\ (mirror\ (l, r))) \\ &= mirror\ (ins\ c\ (rev\ r, rev\ l)) = mirror\ (rev\ r \frown \langle c\rangle, rev\ l) \\ &= (rev\ (rev\ l), rev\ (rev\ r \frown \langle c\rangle)). \end{aligned}$$

Vrijedi:

SEQ PROPERTIES

$$\vdash (\forall\, s, s_1, s_2 : \mathrm{seq}[X])$$
$$\quad rev\ (rev\ s) = s$$
$$\quad rev\ (s_1 \frown s_2) = rev\ s_2 \frown rev\ s_1$$

Dakle,

$$right\ (ins\ c)\ (l, r) = (l, \langle c\rangle \frown r).$$

Sada raspišemo i $right\ del\ (l, \langle c\rangle \frown r)$:

$$\begin{aligned} right\ del\ (l, \langle c\rangle \frown r) &= (mirror \circ del \circ mirror)\ (l, \langle c\rangle \frown r) = mirror\ (del\ (rev\ (\langle c\rangle \frown r), rev\ l)) \\ &= mirror\ (del\ (rev\ r \frown \langle c\rangle, rev\ l))) = mirror\ (rev\ r, rev\ l) \\ &= (l, r) = \mathrm{id}\ (l, r). \end{aligned}$$

Slično se pokažu i ostale tvrdnje. □

Sada prethodno definirane funkcije objedinjujemo sljedećim funkcijama:

$$ACTION : \mathbb{P}(POS \nrightarrow POS)$$
$$DIRECTION : \mathbb{P}((POS \nrightarrow POS) \to (POS \nrightarrow POS))$$

$$ACTION = \{del, move\}$$
$$DIRECTION = \{left, right\}$$

3.4.2 Sustav ED

Prethodno definirane funkcije *del* i *move* bile su parcijalne funkcije jer na rubovima dokumenta nije jasno što treba napraviti primjenom ovih funkcija. No, sada ćemo to nastojati popraviti. Naprije, definirajmo sljedeći model *ED*, koji se trenutno podudara s prethodnim modelom *DOC*:

ED

$$DOC$$

Nadalje, definirajmo funkciju koja će parcijalnu funkciju f preslikati u totalnu funkciju g tako da $g\ x = f\ x$ kad god je $f\ x$ definiran, a kada nije, onda vraća $\mathrm{id}\ \ x = x$.

$[X]$

$$try : (X \nrightarrow X) \rightarrow (X \rightarrow X)$$

$$try\ f = \mathrm{id}\ \ X \oplus f$$

Uočimo da je *try* funkcija definirana na općem skupu X. Stoga, za neki skup A s $try[A]$ označavamo funkciju tipa $(A \nrightarrow A) \rightarrow (A \rightarrow A)$.

Primjer 3.

$$try[POS]\ (move\ (\langle p \rangle, \langle o, c, e, t, a, k \rangle)) = (\langle\rangle, \langle p, o, c, e, t, a, k \rangle)$$
$$try[POS]\ (move\ (\langle\rangle, \langle p, o, c, e, t, a, k \rangle)) = (\langle\rangle, \langle p, o, c, e, t, a, k \rangle)$$

Stoga, sada možemo generalizirati prethodni model sljedećim funkcijama:

$$INSERT : CHAR \rightarrow POS \rightarrow POS$$
$$FUNCTION : (DIRECTION \times ACTION) \rightarrow POS \rightarrow POS$$

$$(\forall\ a : ACTION;\ d : DIRECTION;\ c : CHAR)$$
$$\quad INSERT\ c = ins\ c$$
$$\quad FUNCTION\ (d, a) = try\ (d\ a)$$

Želimo definirati funkcije koje će raditi na riječima, linijama i cijelom dokumentu. Najprije definiramo liniju kao skup svih *POS* čiji je lijevi niz prazan ili završava s *nl*. Ovakva čudna definicija je specifičnost formalne specifikacije. Umjesto da definiramo liniju teksta, mi definiramo predikat po domeni koji će biti istinit kad god se pokazivač nalazi na granicama linije, a to je ekvivalentno definiranju skupa svih dokumenata za koje je to ispunjeno.

$$nl : CHAR$$
$$line : \mathbb{P}\ POS$$

$$line = \{l, r \mid l = \langle\rangle \lor last\ l = nl\}$$

Primjer 4.

$$(\langle z, a, d, n, j, a, nl\rangle, \langle s, l, j, e, d, e, c, a\rangle) \in line$$

ali

$$(\langle z, a, d, n, j, a\rangle, \langle nl, s, l, j, e, d, e, c, a\rangle) \notin line$$

Dakle, skup line sadrži sve dokumente u kojima se pokazivač nalazi na početku linije. Nadalje, slika skupa line pod funkcijom mirror je

$$mirror(\!| \ line \ |\!) = \{l, r \mid r = \langle\rangle \vee first\ r = nl\}$$

pa je

$$(\langle z, a, d, n, j, a\rangle, \langle nl, s, l, j, e, d, e, c, a\rangle) \in mirror(\!| \ line \ |\!)$$

što znači da mirror(| line |) opisuje sve dokumente čiji se pokazivač nalazi na kraju linije.

Definirajmo i konstantu za razmak i riječ.

$$\begin{array}{|l} sp : CHAR \\ word : \mathbb{P}\ POS \\ \hline sp \neq nl \\ word = \{l, r \mid last\ l \in \{sp, nl\} \wedge first\ r \notin \{sp, nl\}\} \cup line \end{array}$$

Primjer 5.

$$(\langle t, r, i, v, i, j, a, l, a, n, sp, sp\rangle, \langle d, o, k, a, z\rangle) \in word$$
$$(\langle t, e, z, a, k\rangle, \langle sp, p, r, o, b, l, e, m\rangle) \in mirror(\!| \ word \ |\!)$$

Slično kao za liniju, uočimo da word definira sve dokumente u kojima se pokazivač nalazi na početku riječi (ili linije). Tada je mirror(| word |) skup svih dokumenata u kojima se pokazivač nalazi na kraju riječi (ili linije).

Na kraju uvodimo skup dokumenata čiji je pokazivač na početku dokumenta i skup čiji se pokazivači nalaze na granicama znakova (tj. skup svih mogućih dokumenata).

$$\begin{array}{|l} document : \mathbb{P}\ POS \\ character : \mathbb{P}\ POS \\ \hline document = \{l, r \mid l = \langle\rangle\} \\ character = POS \end{array}$$

Budući da smo uočili ponašanje funkcije *mirror* u prethodnim primjerima, dat ćemo funkciji *mirror* i identiteti posebna imena:

$$beginning, ending : POS \to POS$$

$$beginning = \mathrm{id}\, POS$$
$$ending = mirror$$

Uočimo: $beginning(\!|\ word\ |\!)$ je skup svih dokumenata u kojima je pokazivač na početku riječi, dok je $ending(\!|\ word\ |\!)$ skup dokumenata u kojima je pokazivač na kraju riječi. Na sličan način možemo gledati i druge skupove. Svi skupovi dokumenata u kojima se pokazivač nalazi na značajnijim granicama dani su u sljedećoj tablici.

$beginning(\!|\ character\ |\!)$	$ending(\!|\ character\ |\!)$
$beginning(\!|\ word\ |\!)$	$ending(\!|\ word\ |\!)$
$beginning(\!|\ line\ |\!)$	$ending(\!|\ line\ |\!)$
$beginning(\!|\ document\ |\!)$	$ending(\!|\ document\ |\!)$

Jasno je da je $beginning(\!|\ character\ |\!) = ending(\!|\ character\ |\!) = character$.

Sada obuhvatimo prethodno definirane funkcije te sve moguće granice u skupove *SIDE* i *PLACE*.

$$SIDE : \mathbb{P}(POS \to POS)$$
$$PLACE : \mathbb{P}(\mathbb{P}\, POS)$$

$$SIDE = \{beginning, ending\}$$
$$PLACE = \{character, word, line, document\}$$

Sljedeća funkcija za dani smjer, vrstu granice i dokument vraća udaljenost (strogo veću od 0) pokazivača u tom dokumentu do najbliže granice u danom smjeru.

$$dist : DIRECTION \to \mathbb{P}\, POS \to POS \nrightarrow \mathbb{N}$$

$$dist\ dir\ place = (\lambda\, doc \mid distances \neq \{\})(min\ distances)$$
$$\text{where } distances = \{d : \mathbb{N} \mid d > 0 \land dir\ move^d\ doc \in place\}$$

Primjer 6. *Odredimo udaljenost pokazivača u dokumentu* $(\langle h, e, r\rangle, \langle sp, h, a, n, d, sp, i, s\rangle)$ *do najbližeg kraja riječi. To formalno zapisujemo na sljedeći način:*

$$dist\ right\ ending(\!|\ word\ |\!)\ (\langle h, e, r\rangle, \langle sp, h, a, n, d, sp, i, s\rangle)$$

Uočimo da tražimo najmanji d za koji, kada se pomaknemo u smjeru right d-puta, dolazimo do kraja neke riječi. Uočavamo da je $d = 5$ *(jer se traži* $d > 0$*!)*

S druge strane, ako želimo pronaći udaljenost udesno u tom istom dokumentu do najbližeg početka linije, tada taj broj ne postoji. Upravo zbog toga, funkcija dist je parcijalna.

Sljedeća funkcija iterativno ponavlja neku akciju u zadanom smjeru sve do određene pozicije u dokumentu.

$$
\begin{array}{|l}
\hline
to : ((ACTION \times DIRECTION) \times \mathbb{P}\, POS) \rightarrow POS \nrightarrow POS \\
\hline
(action, dir)\ to\ place = \\
\quad (\lambda\ doc \mid doc \in \mathrm{dom}\ (dist\ dir\ place)\ (dir\ action^{n}\ doc)) \\
\quad where\ n = dist\ dir\ place\ doc \\
\hline
\end{array}
$$

Uočimo da dokument u kojem brišemo mora biti iz domene funkcije *dist dir place*, tj. broj n je dobro definiran. Razlog zašto je ova funkcija parcijalna je zato što primjenjujemo neku akciju na dokument, a akcija je parcijalna funkcija.

Primjer 7. *U dokumentu* $(\langle h, e, r\rangle, \langle sp, h, a, n, d, sp, i, s\rangle)$ *želimo od trenutne pozicije pokazivača brisati udesno sve do kraja sljedeće riječi. To postižemo sljedećom funkcijom:*

$$(del, right)\ to\ ending(\!|\ word\ |\!).$$

Dakle, primjenom ove funkcije na dokument dobivamo

$$(\langle h, e, r\rangle, \langle sp, i, s\rangle)$$

Uočimo da funkcije

$$
\begin{array}{l}
(move, left)\ to\ character \\
(move, right)\ to\ character \\
(del, left)\ to\ character \\
(del, right)\ to\ character
\end{array}
$$

odgovaraju funkcijama *left move*, *right move*, *left del* i *right del* iz DOC modela.

Ovaj model može se i dalje poboljšati, no to premašuje opseg ovog rada. Ostatak se može vidjeti u [3].

Zaključak

Formalne metode su sve više prihvaćene i u školstvu i u industriji kao jedan od mogućih načina za postizanje veće kvalitete hardverskih i softverskih sustava. No, treba istaknuti da se formalne metode ne bi trebale koristiti kao zamjena za postojeće razvojne procese, nego nadopunjavati druge metode. Unatoč koristi formalnih metoda, one se ne koriste tako puno u izradi softvera zato što mnoge softverske tvrtke nisu htjele riskirati uvođenjem formalnih metoda u njihove razvojne procese jer ih nisu smatrali učinkovitima. Problem je također i matematička notacija jer mnogi nisu imali takvo obrazovanje u prošlosti da bi mogli sve brzo shvatiti i steći određeno iskustvo. Ipak, koriste se u sustavima koji su kritični za sigurnost.

Z jezik za formalnu specifikaciju samo je jedan od mnoštva drugih jezika. Njegova korist proizlazi iz oslanjanja na matematičku notaciju, tj. na logiku i teoriju skupova. Zbog toga Z specifikacije nisu dvosmislene, nego precizne, sažete i jasne. One opisuju što sustav treba raditi, a ne način na koji se to može postići. Smatram da bi u razvoju sustava uvelike pomoglo uvođenje formalne specifikacije, ali ne nužno Z jezik, nego bilo koja druga varijanta koja će u određenoj situaciji najbolje odgovarati.

Literatura

[1] Lightfoot, David. 1991. *Formal Specification Using Z*. Macmillan Education Ltd. London.

[2] Bowen, Jonathan. 2003. *Formal Specification and Documentation using Z: A Case Study Approach*. International Thomson Computer Press. California.

[3] Sufrin, Bernard. 1982. *Formal Specification of a Display-oriented Text Editor*. North-Holland Publishing Company. North-Holland.

[4] Cohen, B. 1982. *Justification of Formal Methods for System Specification*. Software and Microsystem. Volume 1, number 5. Pages 119-127.

[5] Bowen, J.P.; Nicholls, J. E. 1993. *Z User Workshop, London 1992, Workshops in Computing*. Springer-Verlag. Berlin.

[6] Brien, S.M; Nicholls, J.E. 1992. *Z base standard*. Oxford University Computing Laboratory. Oxford.