Liet. matem. rink., T. 43, spec. nr., 228–232

Programų sistemų automatizuoto surinkimo iš gatavų komponentų uždavinys

Vaidas GIEDRIMAS (MII)
el. paštas: vaigie@fm.su.lt

Įvadas

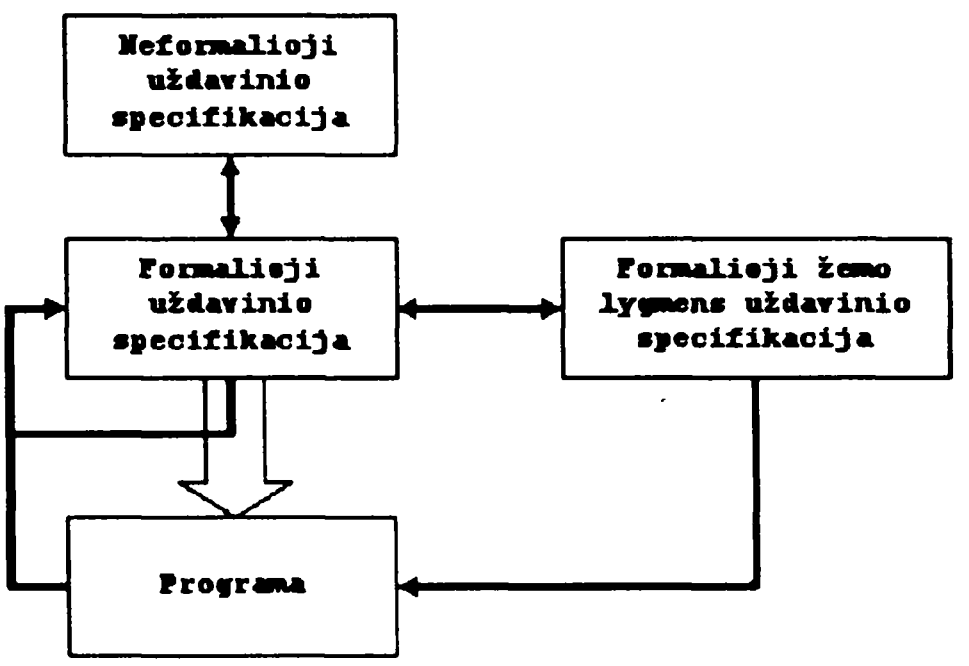
Apie automatizuoto programų kūrimo idėją imta kalbėti 70-aisiais metais, programavimo krizės metu [4, 8]. Ši krizė pasireiškė tuo, kad programų kūrėjai negalėjo patenkinti vartotojų poreikių dėl mažo programavimo proceso produktyvumo. Laikui bėgant atsirado naujos programavimo paradigmos, buvo pasiūlyta įvairių dalinių šios problemos sprendimo būdų. Vienas iš jų – automatizuotas programavimas.

Šio darbo tikslas – išanalizuoti programų sistemų automatizuoto surinkimo iš gatavų komponentų uždavinį ir apžvelgti galimus jo sprendimo būdus.

Automatizuoto programų surinkimo uždavinio samprata

Kaip teigia R. Balzer [2], visiškai automatinio programų kūrimo proceso, kuriame nedalyvautų žmogus negaii būti. Pagrindiniai automatizuoto programų sistemų kūrimo etapai yra šie (1 pav.):

- neformaliosios uždavinio specifikacijos sudarymas;
- formaliosios uždavinio specifikacijos sudarymas;
- neformaliosios specifikacijos peržiūra ir korekcija, jei nustatoma, kad tam tikrų elementų formalizuoti neįmanoma;
- formaliosios specifikacijos transformacija į formalų uždavinio aprašą tam tikroje teorijoje (šis aprašas dar vadinamas *žemo lygmens specifikacija*);



1 pav. Supaprastinta R. Balzer pasiūlyta automatizuoto programų kūrimo schema.

- programos sintezė, remiantis formaliąja žemo lygmens specifikacija.

Nagrinėjamo uždavinio kontekste sąvokas „sintezė“ ir „surinkimas“ galima sutapatinti. Yra numatyta ir formaliosios specifikacijos koregavimo galimybė. Koregavimas negali būti visiškai automatinis, todėl formuojant aukšto lygio specifikaciją ir ją transformuojant į žemo lygio specifikaciją būtinas žmogaus dalyvavimas [2, 3].

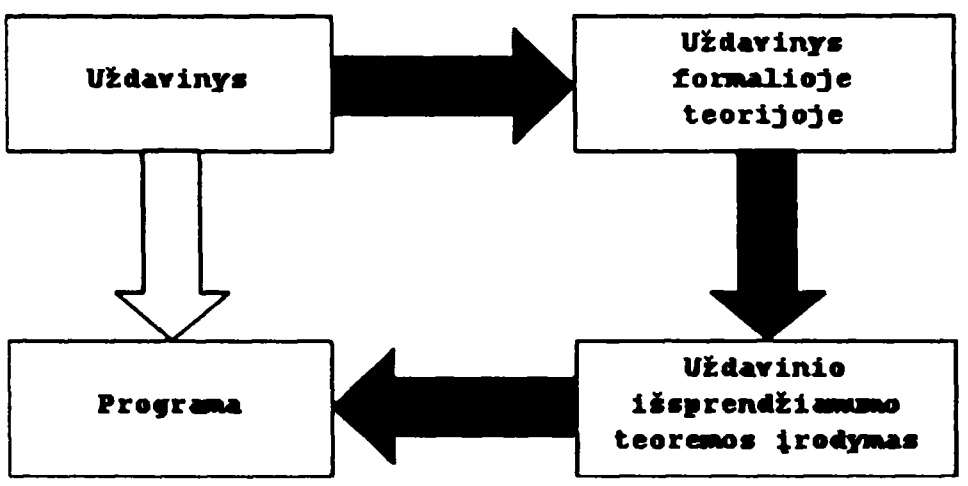
Egzistuoja ir kita programų sistemų automatizuoto surinkimo iš gatavų komponentų uždavinio sprendimo schema [10]. Joje kalbama tik apie *formaliąją uždavinio specifikaciją*, kuri dar vadinama tiesiog *uždaviniu*. Be to, priklausomai nuo to,kuris generavimo metodas naudojamas, tarp formaliosios žemo lygio specifikacijos ir galutinės programos dar gali būti viena ar daugiau tarpinių grandžių (2 pav.).

Pirmiausia uždavinys aprašomas, naudojant kurią nors deklaratyviąją kalbą (formalioji specifikacija). Iš šio aprašo gaunama aibė formulių tam tikroje pasirinktoje teorijoje. Šioje teorijoje uždavinys formuluojamas kaip teorema, kurią reikia įrodyti. Jei uždavinio išsprendžiamumo teorema įrodoma, tuomet atliekama transformacija *teoremos įrodymas-programa*.

Ši schema (2 pav.) vaizdžiai parodo pagrindines programų sistemos surinkimo problemas:

- uždavinio specifikavimo kalbos problema;
- transformacijos į formalų uždavinio aprašą tam tikroje teorijoje problema;
- teoremos formulavimo problema;
- teoremos įrodymo kelio paieškos problema;
- programos gavimo iš teoremos įrodymo problema.

Uždavinio specifikavimo kalbos problema akivaizdi. Neįmanoma formalizuoti natūraliaja kalba pateiktos neformaliosios specifikacijos. Kita vertus, reikalavimas vartotojui mokėti specifikuoti uždavinį formaliąja kalba, mažina jo motyvaciją naudoti automatizuotą programų kūrimo sistemą. Uždavinio specifikavimo kalba turi būti formali, tačiau šiek tiek suprantamesnė žmogui, nei formalioji teorija, kurioje atliekami skaičiavimai. Turint formaliąją uždavinio specifikaciją, ją galima užrašyti kokioje nors teorijoje, pvz., pirmos eilės predikatų logikoje. To paties uždavinio aprašas gali egzistuoti ir kitoje teorijoje. Kurią teoriją pasirinkti, spendžiama atsižvelgiant į tai, kokius veiksmus norima atlikti, kokius metodus ketinama taikyti. Generavimo metodą galima charakterizuoti jo taikymo zonos pločiu. Taikymo zonos pločiu vadinamas teorijų, kuriose pasirinktas me-



2 pav. Automatizuoto programų kūrimo schema.

todas gali būti taikomas, skaičius

$$\omega(x)_{TZ}, \tag{1}$$

čia x – generavimo metodas. Lyginant kelis programų sistemų generavimo metodus, nagrinėjamo uždavinio sprendimo požiūriu geresniu laikomas tas metodas, kurio taikymo zonos plotis didesnis. Kuo didesnis metodo taikymo zonos plotis, tuo daugiau uždavinių šiuo metodu galima išspręsti.

Teoremos formulavimo ir jos įrodymo kelio paieškos problemos taip pat priklauso ir nuo pasirinktos formaliosios teorijos ir nuo naudojamo generavimo metodo. Pažymėtina, kad nuo pasirinktos įrodymo kelio paieškos sistemos [14] priklauso galutinės programos surinkimo iš struktūrinių dalių greitis.

Programų sistemų surinkimo iš gatavų komponentų uždavinyje nėra konkretizuojamas termino *programa* turinys. Tai gali būti ir imperatyvinė programa, ir funkcinė programa. Iš dalies jau pačios formaliosios teorijos pasirinkimas labai priklauso nuo to, kokią programą tikimasi gauti. Pvz., funkcinėms programoms modeliuoti naudojamas 1930-aisiais metais A. Church pasiūlytas λ-skaičiavimas (angl. *λ-calculus*). Kitas pavyzdys – loginės programos [15]. Išskiriami du programavimo ir logikos bendro naudojimo („sankirtos“) atvejai: *programavimas logikoje* (angl. *programming in logic*) ir *loginis programavimas* (angl. *logic programing*).

Programos gavimo iš teoremos įrodymo problemos požiūriu programų sistemų generavimo metodą gali charakterizuoti gaunamos programos tipų aibė. Pvz., jei pasirinktas metodas skirtas tik funkcinėms programoms kurti, tai šią aibę sudarys vienas elementas F, jei tik imperatyvinėms – taip pat tik vienas elemetas I. Jei galima kurti abiejų tipų programas, aibė sudarys du elemetai: $\{F, I\}$. Universalsniu laikomas tas metodas, kurio gaunamos programos tipų aibės elementų skaičius yra didesnis.

Struktūrinės sintezės metodas

Nagrinėjamo uždavinio kontekste galima išskirti tokius programų sistemų [15] sintezės metodus: induktyvinė sintezė, konstruktyvioji sintezė, deduktyvinė sintezė, struktūrinė sintezė.

Induktyvinė sintezė plačiai taikoma loginių programų kūrimui. Egzistuoja net atskira loginio programavimo šaka – indukcinis loginis programavimas (ILP). Tačiau nagrinėjamo uždavinio kontekste šis metodas turi trūkumų. Uždavinio sąlygoje pasakyta, kad jau yra paruošti komponentai iš kurių reikia surinkti sistemą. Tuo tarpu induktyvinė sintezė dažniau naudojama naujiems objektams kurti, remiantis teigiamais ir neigiamais pavyzdžiais.

Konstruktyviosios sintezės metodas buvo sukurtas funkcinėms programoms kurti. Šiuo metu jis yra naudojamas keliuose programų sintezės sistemose. Metodas remiasi Curry-Howard kostruktyviosios tipų teorijos izomorfizmu. Tai reiškia, kad tarp konstruktyviojo egzistencijos teoremos įrodymo ir programos egzistuoja vienareikšmė atitiktis [6].

Struktūrinė sintezė – dedukcijos principu grindžiamas programų sintezės metodas. Daugiau kaip prieš dvidešimt metų [11] pasirodžiusi idėja buvo išvystyta. Sukurtos veikiančios NUT ir PRIZ sistemos.

Struktūrinė sintezė grindžiama teiginiu, kad konstruoti programas galima remiantis vien jų struktūrinėmis savybėmis. Tokiu atveju uždavinio sąlygą galima nusakyti aprašant turimų komponentų ir būsimos programų sistemos struktūrines savybes. Tokia specifikacija suprantama vartotojui, be to jos pagrindu galima lengvai sudaryti uždavinio sprendimo žemo lygio specifikaciją.

Struktūrinės sintezės metodas leidžia atlikti operacijas vienoje iš trijų teorijų [1]:

- λ-skaičiavimas;
- konstruktyvioji tipų teorija (angl. *constructive theory of types*);
- intuicionistinė logika (angl. *intuitionistic logic*).

Tipizuotų λ-skaičiavimų, paprastųjų tipų konstruktyviojoje tipų teorijoje ir intuicionistės logikos elementų ekvivalentumą galima įrodyti remiantis Curry-Howard darbais [6].

Struktūrinės sintezės metodas buvo sukurtas programoms sintezuoti iš modulių, o vėliau pritaikytas objektinio programavimo paradigmai. Jo panaudojimo galimybės programų sistemų surinkimui iš komponentų kol kas dar tiriamos [5, 9]. Vieni autoriai siūlo komponentinėms programų sistemoms aprašyti naudoti λ-skaičiavimą, nes jis tinka inkapsuliacijai, kompozicijai ir su tipais susijusiesiems reiškiniams modeliuoti.

Kiti autoriai atkreipia dėmesį į tai, jog λ-skaičiavimas negali būti taikomas konkurenciniams ir bendradarbiavimo procesams modeliuoti. Kaip alternatyva siūlomas π-skaičiavimas [7, 12, 13, 16].

Struktūrinės sintezės metodą galima taikyti mažiausiai trijose teorijose:

$$\omega(SS)_{TZ} \geqslant 3. \tag{2}$$

Struktūrinės sintezės *gaunamos programos tipų aibės* elementų skaičius taip pat yra didesnis už vienetą: šis metodas, iš pradžių buvo skirtas *loginiam programavimui*, tačiau dabar naudojamas ir *programavimui logikoje* [15].

Išvados

Apžvelgus programų sistemų surinkimo iš gatavų komponentų uždavinį ir keletą generavimo metodų, galima daryti išvadą, kad struktūrinės sintezės metodas gali būti taikomas šiam uždaviniui spręsti. Struktūrinės sintezės privalumai yra šie:

- konstruoti programas galima remiantis vien jų struktūrinėmis savybėmis;
- metodą galima taikyti keliose teorijose;
- *gaunamos programos tipų aibės* elementų skaičius didesnis už vienetą.

Sprendžiant programų sistemų surinkimo iš gatavų komponentų uždavinį, kartu su struktūrinės sintezės metodu galima taikyti ir kitus programų sistemų generavimo metodus.

Literatūra

[1] M. Addibpour, E. Tyugu, Structural synthesis of programs from refined user requirements, in: *Proc. of Dagstuhl Seminar on Methods for Semantics and Specification*, Schloss Dagstuhl, Germany, June 5–9 (1995).
[2] R. Bazler, A 15 year perspective on automatic programming, *IEEE Transactions On Software Engineering*, **SE-11**(11), November (1985).
[3] Y. Deville, K.-K. Lau, Logic program synthesis, *The Journal of Logic Programming*, **12**, 1–199 (1993).
[4] E.W. Dijkstra, The huble programmer, *Comunications of ACM*, **15**(10), 861 (1972).
[5] V. Giedrimas, Komponento modelis struktūrinės programų sintezės kontekste, *Informacijos mokslai*, **26**, 246–250 (2003).
[6] W.A. Howard, The formulae-as-types notion of constructions, in: H.B. Curry, *Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic press (1980), pp. 479–490.
[7] C.B. Jones, A pi-calculus semantics for an object-based design notation, in: E. Best (Ed.), *Proceedings CONCUR'93*, LNCS 715, Springer (1993), pp. 158–172.
[8] E.A. Karlsson, S. Sorumgard, E. Tryggeseth, *Classification of Object-Oriented Components for Reuse* (1992).
[9] M. Matskin, E. Tyugu, Strategies of structural synthesis of programs, in: *Automated Software Engineering Conference*, USA (1997), pp. 305–306.
[10] M. Matskin, J. Komarowski, Partial structural synthesis of programs, *Fundamenta Informaticae*, **30**, 23–41 (1997).
[11] G. Mints, E. Tyugu, Justification of the structural synthesis of programs, *Science of Computer Programming*, **2**(3), 215–240 (1982).
[12] D. Sangiorgi, *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*, PhD thesis, Computer Science Department, University of Edinburgh, UK, May (1993).
[13] J.G. Schneider, *Components, Scripts, and Glue: a Conceptual Framework for Software Composition*, Bern, den 28, October (1999).
[14] T. Uustalu, Extensions of structural synthesis of programs, in: U. Engberg *et al.* (Eds.), *Proc. 6th Nordic Workshop on Programming Theory*, BRICS Notes Series NS-94-6, Univ. of Aarhus (1994), pp. 416–428.
[15] T. Uustalu, *Structural Synthesis of Programs as Programming in Logic*, Technical Report TRITA-IT R 95:08, Department of Teleinformatics, Royal Institute of Technology, Stockholm, March (1995).
[16] D.J. Walker, Objects in the Pi-calculus, *Information and Computation*, **116**(2), 253–271 (1995).

Problem of software synthesis from made-up components

V. Giedrimas

The paper analyze software synthesis from components problem, overviews possible solutions. This article shows that synthesis of programs (SSP) is suitable method to solve this problem.