Liet. matem. rink, **45***, spec. nr.*, 2005, 139–143

Komponentinių programų struktūrinės sintezės teorinės problemos

Vaidas GIEDRIMAS, Audronė LUPEIKIENĖ (MII)
el. paštas: vaigie@fm.su.lt, audronel@ktl.mii.lt

Reziumė.
Struktūrinė programų sintezė (SSP) yra deduktyvinis programų sintezės metodas sėkmingai pritaikytas struktūrinių ir objektinių programų kūrimui. Šiame straipsnyje nagrinėjamas struktūrinės programų sintezės metodo taikymas komponentinėms sistemoms kurti ir aptariamos teorinės šio proceso problemos: specifikacijos problema, komponento pakartotinio panaudojimo ir neapibrėžtųjų komponentų problemos. Pateikiami galimi šių problemų sprendimo būdai: plėsti tradiciškai SSP naudojamą formalizmą (t.y. intuicionistinį teiginių skaičiavimą) arba parinkti kitą, papildyti įrodymo paieškai naudojamų aksiomų aibę.

Raktiniai žodžiai: komponentinių programų sistemų inžinerija, struktūrinė sintezė, formalieji metodai.

1. Įvadas

Struktūrinė programų pintezė (SSP) yra deduktyvinis programų sintezės metodas sėkmingai pritaikytas PRIZ, XpertPriz [6, 10] ir NUT sistemose [11]. Pagrindinė metodo idėja – konstruoti programas iš atskirų modulių („juodųjų dėžių“), atsižvelgiant tik į jų įeities bei išeities taškus ir visiškai neatsižvelgiant į jų realizacijos detales. Kiekvieną tokį modulį atitinka viena intuicionistinio teiginių skaičiavimo (ITS) aksioma. Programos sintezė vyksta šiais etapais:

- sudaroma uždavinio specifikacija;
- ši specifikacija transformuojama į ITS teoremą;
- naudojantis žinių baze, kuriai priklauso ir anksčiau minėtos aksiomos, ieškoma konstruktyvaus teoremos įrodymo;
- jei įrodymas rastas, jis nusako programos struktūrą, kuri, keičiant panaudotas aksiomas jas atitinkančiais moduliais, užpildoma ir gaunama visa programa.

Detalesnis SSP metodo aprašymas pateikiamas [6, 8, 9] darbuose. Dauguma darbų nagrinėja SSP pritaikymą struktūrinio ar objektinio programavimo paradigmoms. Metodo taikymas komponentinei paradigmai reikalauja teorinių tyrimų. Lammerman ir Rao [4] nagrinėja SSP taikymą komponentinių sistemų kūrimui, tačiau jų siūlomi sprendimai yra tik daliniai, nagrinėjantys tik vieną komponento modelį – *Web servisus*.

Šio straipsnio autoriai teigia, jog Struktūrinė programų sintezė gali būti panaudota komponentinių programų automatizuotam kūrimui. Šią prielaidą paremia pati SSP metodo esmė – operavimas „juodosiomis dėžėmis“ [6, 8] ir tas faktas, kad komponentas taip pat yra „juodoji dėžė“ [7]. Komponentinės struktūrinės sintezės atveju pakartotinai panaudojamų modulių vaidmenį atlieka komponentai, o jų įeities ir išeities taškais vadinami reikalaujami ir realizuojami interfeisai atitinkamai. Pvz., komponentas, kuris

teikia paslaugą B, ir reikalauja paslaugos A, intuicionistiniame teiginių skaičiavime aprašomas tokia aksioma:

$$A \longrightarrow B. \tag{1}$$

Straipsnio tikslas – nustatyti komponentinių programų struktūrinės sintezės teorines problemas ir pasiūlyti galimus jų sprendimo būdus.

2. Specifikacijos problema

SSP metodo problemas tikslinga klasifikuoti ir nagrinėti pagal atskiras programos sintezės stadijas:

- uždavinio specifikavimas aukšto lygmens kalba;
- uždavinio specifikacijos transformavimas į ITS teoremą;
- įrodymo paieška;
- įrodymo transformavimas į programą.

Uždavinio specifikavimo aukšto lygmens kalba problemos nėra SSP-specifinės: specifikavimo kalbos išraiškingumas, patogumas vartotojui ir t.t. [1, 2]. Panašios problemos kyla ir naudojant kitus sintezės metodus, todėl jos čia nebus nagrinėjamos. Uždavinio specifikacijos transformavimo į ITS teoremą procesas yra apribotas vienu reikalavimu: turi egzistuoti vienareikšmė atitiktis tarp aukšto lygmens kalbos elementų ir naudojamo formalizmo (šiuo atveju – ITS) elementų [6, 11]. Tai reiškia, kad aukšto lygmens kalbos išraiškos priemonės yra ribojamos formaliosios teorijos. Šis ribojimas nebuvo esminis struktūrinei ir netgi objektinei programoms, tačiau komponentinei paradigmai jis yra viena iš esminių problemų. Klasikinis SSP metodas negali būti panaudojamas kitokio nei valdymo srautų ir duomenų srautų architektūros stilių programoms kurti. Lammerman ir Tyugu darbe [4] pasiūlytasis Išplėstasis SSP metodas (ESSP) įgalina kurti objektinio architektūros stiliaus programas, gebančias reaguoti į išimtis (*angl. exception*). Tačiau ir ESSP nesuteikia galimybės kurti įvykiais valdomo architektūros stiliaus programas. Nei viena SSP ar ESSP realizacija nenumato asinchroninių operacijų tarp modulių galimybės.

3. Komponento pakartotinio panaudojimo problema

Naudojamų loginių teorijų trūkumai pastebimi ir įrodymo paieškos stadijoje. SSP naudotas implikatyvusis intuicionistinio teiginių skaičiavimo fragmentas, numatė tik dvi logines operacijas: implikaciją ir konjukciją [10]. Kaip parodė Lammerman [4], tokia logika yra nepakankama specifikuoti objektams ir jų sąryšiams. ESSP metodas naudoja intuicionistinį teiginių skaičiavimą, kuriame įvesta disjunkcijos operacija ir klaidos (*falsity*) sąvoka.

Atlikti tyrimai parodė, kad klasikinis intuicionistinis teiginių skaičiavimas kaip formalizmas turi trūkumų. Šios problemos susijusios su pakartotiniu to paties komponento panaudojimu sistemoje kelis kartus. Kaip teigiama darbe [5], naudojant intuicionistinį teiginių skaičiavimą, kaip formalizmą, sistemų su použdavinias atveju rekursinėms programoms neįrodomas paieškos proceso baigtumas. Naudojant išvedimo taisyklę

$$\frac{A \longrightarrow A, A \longrightarrow A \ldots A \longrightarrow A}{A \longrightarrow A}, \tag{2}$$

ir neatsižvelgiama į tą faktą, kad vienos aksiomos daugkartinis taikymas nebūtinai duoda tą patį rezultatą, kaip ir vienkartinis.

Pavyzdys. Duotas komponentas K turintis įėjimo tašką A ir išėjimo tašką A. Komponento prasmė yra tokia: $A := A + 1$. Tarkime, kad specifikuota problema „Sukurti programą, kuri duotąjį skaičių padidina reikšme N“. Sintezės rezultatas turėtų būti penki nuosekliai interfeisais sujungti penki komponento egzemplioriai, tačiau nei SSP, nei ESSP tokio sprendinio nerastų. Šią problemą išspręsti galima dviem būdais:

- praplečiant intuicionistinį teiginių skaičiavimą įvedant naują loginę operaciją;
- parenkant kitą formalizmą, kuris turėtų priemones išreikšti pakartotinio aksiomos taikymo neinvariantiškumą.

4. Neapibrėžtųjų komponentų problema

Kita aktuali problema – SSP reikalavimas, kad visos būsimosios programos sudėtinės dalys (šiuo atveju, komponentai) būtų specifikuotos prieš sintezės procesą. Daroma prielaida, kad visi reikiami komponentai jau yra, tik reikia išspręsti jų pasirinkimo uždavinį. Jei nors vienas sistemai sukurti reikalingas komponentas neaprašytas aksioma, gali susidaryti situacija, kai teoremai įrodyti pritrūks būtent šios aksiomos.

Tuo tarpu vienas iš komponentinių programų kūrimo etapų yra komponentų paieška [7]. Be to, ši paieška turi būti vykdoma realizacijos lygmenyje, jau suprojektavus programų sistemą. Vienas iš galimų neapibrėžtųjų komponentų problemos sprendimo būdų – įvesti papildomas aksiomas.

Tarkime, kad M – sintezės uždavinio aksiomų aibė. Kiekviena aksioma $m \in M$ turi pavidalą:

$$u_1, u_2, \ldots u_k \longrightarrow v_1, v_2, \ldots v_l, \tag{3}$$

K – visų bent vienai aksiomai priklausančių propozicinių kintamųjų aibė:

$$\forall i, j u_i, v_j \in K \tag{4}$$

o A – algoritmas, analizuojantis aibes K ir M, ieškantis teoremos

$$p_1, p_2, \ldots p_n \longrightarrow r_1, r_2, \ldots r_m; \quad \forall i, j p_i, r_j \in K \tag{5}$$

įrodymo. Tarkime, kad A baigė darbą, tačiau teoremos įrodymas nebuvo rastas. Tada aibė M papildoma *virtuliomis aksiomomis*:

$$a\colon u_1, u_2, \ldots u_k \longrightarrow v_1, v_2, \ldots v_l, \quad k, l > 0, a \in M', \tag{6}$$

ir gautai aibei $M' = M \cup M''$ vėl taikomas algorimas A. Galima įrodyti, kad aibėje M' algoritmas A visada baigs darbą ir ras mažiausiai vieną sprendinį.

Virtuliaja aksioma (VA) šiame straipsnyje vadinama neturinti realizacijos (t.y. nesusieta nei su vienu komponentu) aksioma, teigianti, kad naudojant vienų propozicinių kintamųjų (PK) reikšmes, galima apskaičiuoti kitų PK reikšmes. Kitaip tariant, turimų komponentų aibėje nėra nei vieno tokio, kuris būtų susijęs su kokia nors VA ryšiu *“vienas-su-vienu”*.

Algoritmui A baigus darbą ir radus teoremos įrodymą, sintezės sistemos naudotojui tampa prieinama informacija apie tai, kokios VA turi būti panaudotos įrodyme, o tai savo ruožtu parodo, kokių komponentų trūksta gauti galutiniam rezultatui – komponentinei programai.

Metodas yra gana paprastas, tačiau turi ir trūkūmų. Įvertinkime papildomų aksiomų įvedimo įtaką teoremos įrodymo paieškos algoritmui. Tarkime, kad po papildymo aibėje M' yra visos aksiomos, kokias tik galima sudaryti iš K aibės propozicinių kintamųjų. Tada, bendras aibėje M' esančių aksiomų skaičius yra

$$n(M') = \sum_{i=1}^{k} C_k^i \cdot \sum_{j=1}^{k} C_k^j,$$

kur k – propozicinių kintamųjų aibėje K skaičius. Atmeskime iš šios aibės $A \longrightarrow A$ pavidalo aksiomas, nes, kaip minėta anksčiau, tokio tipo aksiomos laikomos trivialiomis ir dažniausiai nenaudojamos. Tokių aksiomų gali būti tiek, kiek yra skirtingų gretinių sudarytų iš aibės K elementų, todėl:

$$n(M') = \sum_{i=1}^{k} C_k^i \cdot \sum_{j=1}^{k} C_k^j - \sum_{i=1}^{k} C_k^i = \sum_{i=1}^{k} C_k^i (\sum_{j=1}^{k} C_k^j - 1). \tag{7}$$

Yra žinoma [3], kad

$$\sum_{i=0}^{k} C_k^i = 2^k,$$

todėl

$$n(M') = (2^k - 1)(2^k - 2) \approx 2^{2k}. \tag{8}$$

Palyginus gautus rezultatus su [8] pateiktais SSP algoritmų sudėtingumo įverčiais (1 lentelė) matyti, kad papildomų aksiomų įvedimas gali labai padidinti įrodymo paieškos laiką.

Kita vertus, didelis VA skaičius gali sukelti įrodymo nevienareikšmiškumo ir rezultatų nepatikimumo problemas.

Kuo daugiau virtualiųjų aksiomų yra įvesta, tuo daugiau skirtingų teoremos įrodymų gali būti rasta. Jei kiekviename iš šių įrodymų yra panaudota bent viena VA, tai sėkmingam sintezės procesui reikia rasti šią aksiomą atitinkantį komponentą. Jei atitinkamas komponentas neegzistuoja, sintezės sistemos naudotojas gali jį sukurti, arba atmesti nagrinėjamą teoremos įrodymą kaip nepagrįstą ir procesą tęsti ieškant kito įrodymo. Įvertinus komponentų paieškos ir skirtingų įrodymų gavimo laiką, galima teigti, kad sintezės procesas paailgėja.

Didelis virtualiųjų aksiomų skaičius leidžia įrodyti beveik bet kokią (5) pavidalo teoremą. Teorema neįrodoma tik tuo atveju, jei jos sąlygoje yra bent vienas propozicinis kintamasis nepriklausantis aibei K. Dėl šios priežasties gali būti surastas toks teoremos įrodymas, kad VA atitinkantys komponentai arba neegzistuoja, arba yra nesuderinami semantine prasme. Pvz., teoremos įrodyme panaudota aksioma: $PinigineSuma \longrightarrow TaskoEkraneSpalva$ neturi ir negali turėti ją realizuojančio komponento.

1 lentelė. SSP algoritmų sudėtingumas laiko atžvilgiu

Programos struktūra	Sudėtingumas
Tiesinė	$c_1 \cdot n^2$
Besišakojanti	$c_1 \cdot n^2 \cdot 2^n$
Su použdaviniais	$c_1 \cdot n^3 \cdot 2^n$

5. Išvados

Komponentinių programų struktūrinės sintezės teorinės problemos yra:

- specifikacijos problema;
- komponento pakartotinio panaudojimo problema;
- neapibrėžtųjų komponentų problema.

Komponento pakartotinio panaudojimo problema gali būti sprendžiama dviem būdais:

- praplečiant intuicionistinį teiginių skaičiavimą (pvz., įvedant naują loginę operaciją);
- parenkant kitą formalizmą (pvz., laiko logiką), kuris turėtų priemones išreikšti pakartotinio aksiomos taikymo neinvariantiškumą.

Vienas iš galimų neapibrėžtųjų komponentų problemos sprendimo būdų – papildyti aksiomų aibę virtualiomis aksiomomis. Tačiau, kaip rodo tyrimai, šis metodas gali būti taikomas tik tada, kai įvedamų aksiomų skaičius nedidelis.

Apibendrinus rezultatus galima teigti, kad aptartosios problemos gali būti išsprendžiamos ir struktūrinės programų sintezės metodą galima naudoti komponentinių sistemų kūrimui.

Literatūra

1. V. Giedrimas, Komponento modelis struktūrinės programų sintezės kontekste, *Informacijos mokslai*, **26**, 246–250 (2003).
2. V. Giedrimas, A. Lupeikienė, Komponento specifikacijos formalizavimas, *Liet. matem. rink.*, **44** (spec. nr.), 276–280 (2004).
3. J.H. Lint, R.M. Wilson, *A Course in Combinatorics*, Cambridge University Press (2002).
4. S. Lammermann, *Runtime Service Composition via Logic-based Program Synthesis*, PhD. thesis, KTH, Stockholm (2002).
5. M. Matskin, E. Tyugu, Strategies of structural synthesis of programs, in: *Automated Software Engineering Conference*, USA (1997), pp. 305–306.
6. G. Mints, E. Tyugu, Justification of the structural synthesis of programs, *Science of Computer Programming*, **2**(3), 215–240 (1982).
7. J.G. Schneider, *Components, Scripts, and Glue: a Conceptual Framework for Software Composition*, Bern, October (1999).
8. E. Tyugu, M. Harf, Algoritmy strukturnovo sinteza program, *Programirovanie*, **4**, 3–13 (1985).
9. E. Tyugu, A. Saabas, Problems of visual specification languages, in: *Proceedings of 35th International Conference on IT + SE* (2003).
10. E. Tyugu, Three new-generation software environments, *Comm. ACM*, **34**(6), 46–59 (1991).
11. T. Uustalu, T. Kopra, V. Kotkas, M. Matskin, E. Tyugu, *The NUT Language Report*, Technical Report TRITA-IT-R 94:14, CSLab, Dept. of Teleinformatics, Royal Institute of Technology (KTH), Stockholm (1994).

SUMMARY

V. Giedrimas, A. Lupeikienė. Theoretical problems of component-based structural synthesis

The Structural Synthesis of Programs (SSP) method is based on the idea that programs can be constructed taking into account only their structural properties. This method has been successfully used to synthesise structural and object-oriented programs. This paper presents the research on the SSP in the component-based environment. It discuses the possibility of SSP method application to software synthesis from components and shows the main problems: problem of the specification, problem of reusable components and problem of undefined components. The possible solutions to those problems are provided too.
Keywords: component-based software engineering, structural synthesis, formal methods.