УДК 004.517 DOI 10.29141/2949-477X-2023-2-4-1 EDN CDUMAE

K. E. Iklassova, R. N. Kozhakhmetova

Manash Kozybayev North Kazakhstan university, Petropavlovsk, Republic of Kazakhstan

Modeling the dynamic semantics of a computer program

Abstract. In this paper, two approaches are used to construct the dynamic semantics of computer programs: the first is the representation of mathematical models of computational processes generated by a computer program in the form of a discrete transformer; the second is the representation of the mathematical model of a computer program in terms of functional grammars. The study focuses on the computational process generated by a computer program. Application of the discrete transducers concept to describe computational processes allows us to obtain a mathematical model or dynamic semantics of a program, which is a composition of two discrete systems B and C. In this case, component A is a control component (a model of the program block diagram), and component B is a model of the program memory. The decomposition of a program into two components B and B is convenient when studying the process of modelling a computational process. This is due to the fact that the prospects of searching for invariants in critical nodes of a programme for mathematical proof of correctness of a computer programme are opened. The comparative analysis of the two approaches leads to the theorem on the regularity of the representation of a function generated by a programme within the framework of functional grammar.

Key words: modeling, mathematical model of the language, dynamical system, semantics, regularity, context-free grammar, loop operator, syntax.

Paper submitted: November 11, 2023.

For citation: Iklassova K. E., Kozhakhmetova R. N. Modeling the dynamic semantics of a computer program. Digital models and solutions. 2023. Vol. 2, no. 4. Pp. 5–12. DOI: 10.29141/2949-477X-2023-2-4-1. EDN: CDUMAE.

ISSN 2782-4934 (online)
ISSN 2949-477X (print)

Системный анализ
System analysis

К. Е. Икласова, Р. Н. Кожахметова

Северо-Казахстанский университет имени Манаша Козыбаева, г. Петропавловск, Республика Казахстан

Моделирование динамической семантики компьютерных программ

Аннотация. В данной статье для построения динамической семантики компьютерных программ используются два подхода: первый – представление математических моделей вычислительных процессов, формируемых компьютерной программой, в виде дискретного трансформатора; второй – представление математической модели компьютерной программы в терминах функциональных грамматик. В исследовании основное внимание сосредоточено на вычислительном процессе, генерируемом компьютерной программой. Применение концепции дискретных преобразователей для описания вычислительных процессов позволяет получить математическую модель или динамическую семантику программы, которая представляет собой композицию двух дискретных систем B и B. В этом случае компонент A – компонент управления (модель блок-схемы программы), а компонент B – модель памяти программы. Разложение программы на две компоненты B и B удобно при изучении процесса моделирования вычислительного процесса. Это связано с тем, что открываются перспективы поиска инвариантов в критических узлах программы для математического доказательства корректности компьютерной программы. Сравнительный анализ двух подходов приводит к теореме о регулярности представления функции, порождаемой программой в рамках функциональной грамматики.

Ключевые слова: моделирование; математическая модель языка; динамические системы; семантика.

Дата поступления статьи: 11 ноября 2023 г.

Для цитирования: Iklassova K. E., Kozhakhmetova R. N. Modeling the dynamic semantics of a computer program // Цифровые модели и решения. 2023. Т. 2, № 4. С. 5–12. DOI: 10.29141/2949-477X-2023-2-4-1. EDN: CDUMAE.

Introduction

Consider the loop operator of a programming language from two points of view:

– from the point of view of discrete converter [1–3];
– from the point of view of functional grammars [4; 5].

These two theories are conveniently used in computer modeling of various dynamical systems, such as a computational process generated by numerical algorithms in order to find the invariants in the critical nodes of the program. The found invariants can be used to prove the specificity of the program and can be represented in the language of positively formed formulas [5; 6].

The memory of program A denoted by $B \subseteq D^v$, where D – set of values, $V = \{v_1, v_2, \ldots\}$ – set of variables of a program A. In other words, B is a set of interpretations of variables from the set V, that is, if $b \in B$, then $b : V \to D$, and for any $v_i \in V b(v_i) = d$, where $d \in D$, d – is a value taken by the variable v_i. Sometimes authors use postfix notation $v_i b$ instead of $b(v_i)$. This work is limited to that class of computer programs A, the meaning of which can be represented as function $f_A : B \to 2^B$. In other words, the program A starts working at some initial state $b_H \in B$ of memory and finishes its work in one of the finishing states $b_K \in 2^B$, thus the program A is a multi-valued mapping. It is quite natural that the program can be considered as a discrete processor, consisting of two components: A and B, where B – control component, B – a set of program memory states. The control component A of the program A (designations match) has various representations:

1) in the form of a text, when the program A is presented in a programming language (it is said that A has a text metatype);

2) in the form of a control graph block diagram consisting of vertices denoted by symbols a_1, a_2, … ∈A , which are connected by arcs of transitions with the notations u_i/y_i , where $u_i \in \hat{U}$ is a logical expression composed of basic conditions $u \in U$ of program A using logical operations ∧ (conjunctions), ∨ and ¬. Each $y_i \in Y^*$ is a superposition of basic assignment operations $y \in Y$ of program A, having the form $y_i = (v_{i1} : t_{i1}(v), \ldots, v_{in} := t_{in}(\mathrm{v})) = (v := t(v))$, where t – is an expression, constructed from operations belonging to Ω from the data algebra operations signature $D = (D, \Omega, \prod)$ of program A , where D – a set of data values, $\prod$ – data predicate signature. Based on the foregoing, we give a mathematical definition of a discrete program converter A, called «$U - Y$– scheme of the program A», sometimes referred to simply as a component A, when it is clear from the context.

Materials and methods

Determination. $U - Y$ – Scheme of the program A, is a set $A = \{a_i \mid i = 1, \ldots, m\}$ states of the circuit along with set $T \subset A \times \hat{U} \times Y^* \times A$.

Graphically an element of the set T can be represented as: $(a_i \to u/y\, a_j) \in T$. If $y = e$ identical to the reference $e : B \to B$ so, that $\forall b \in B\, e(b) = b$, then the graph is $(a_i \xrightarrow{y} a_j)$. Authors note a formal logical circuit $(a_i \xrightarrow{u} a_j)$ can be turned into a function $u/e : B \to B$ narrowing of scope $e : B \to B$ on condition $\forall b \in B : u(b) = true \to e(b)$. Thus, we can construct the set $U/e = \{u/e \mid u \in U\}$, and correspondingly $\hat{U}/e = \{u/e \mid u \in\}$. If by q_i we denote words in the alphabet Y, that is $q_i = y_1 O y_2 O \ldots O y_m, y_i \in Y$ and $q_i \in Y^*$, then $q_i : B \to B$ we will call elementary memory state converters.

Transitions $a_1 \xrightarrow{u_1/q_1} a_2$ and $a_2 \xrightarrow{u_2/q_2} a_3$ we will call conjugate. A sequence of transitions in which any two transitions are conjugate is called the path l from a_i to a_j, where a_i is the beginning of the path l, and a_j is the end is the path. If the computing process is in a state $(a_j, b_j) \in A \times B$, then at the end of the path l the process will be in a condition $(a_j, b_j) \in A \times B$. If the superposition of functions q transitions of path l easy to compose $q \in \hat{U}/e \cup Y^*$, then $b_j = q(b)$.

Suppose $U - Y$ – scheme of program A has one initial a_H and one final state of the scheme A. Then we can state the following regularity theorem for the function $f_A : B \to 2^B$, generated by the program A.

Theorem. Display of $f_A : B \to 2^B$, performed by $U - Y$ –scheme of program A, is regular to the set $\hat{U} / e \cup Y^*$.

Proof. To prove this theorem, we construct an algebraic model of $U - Y$ – scheme of program A and in the constructed model we consider the loop operator. To do this, we fix the set B of memory states and consider binary relations $f \subset B \times B$. As a supporting set of the algebra, all subsets $B^2 \subset B \times B$ we denote by $M = \{f \subset B \times B$. Fix the set $Y \subset M$. We set on the set Y the following operations:

1) such superposition $fOf' = g$, that $g(b) = b' \Leftrightarrow b'' \in B : f(b) = b''$ and $f'(b'') = b'$; ¢;

2) such association $f \cup f^i = g$, that $g(b) = b' \Leftrightarrow b'$ or $f'(b) = b'$;

3) iteration $f^* = \bigcup_{n=0}^{\infty} f^n$, where $f^0 = e, f^{n+1} = fOf^{n+1} = fOf^n$, Y –is an identity relation. Smallest set of functions containing Y, as well e – is an identical display, an empty mapping and closed with respect to the operations of superposition, unification, and iteration will be an algebraic model $U - Y$ – scheme of program A. The theorem is proved.

Moving on to the consideration in this model the cycle operator of the programming language. Imagining the loop operator syntax rule

$$<cycle>: : \text{while } u \text{ do } \{f_{10}\} \text{ end } \{f_{10}\};$$

Function f_{10} is assigned to the terminal <*cycle*> and describes a syntax function or mapping generated by a loop operator. In this construction, u is *a < logical expression >*, P stands for nonterminal <*operators*>. Function f_{10} has two arguments u and P.

Lemma. In the constructed algebraic model $Y \cup \{e\} \cup \{\emptyset\}$, signature operations $\Omega = (O, \cup, *, /)$ function f_{10} is regular with respect to the alphabet $(Y, e, \emptyset, \Omega)$.

Proof. Function f_{10} can be represented as an algebraic expression.

$$\bigcup_{n=0}^{\infty} (u/ P \vee \overline{u} / e)^n, \text{ that is } f_{10} = \bigcup_{n=0}^{\infty} (u/ P \vee \overline{u} / e)^n = (u/ P \vee \overline{u} / e)^*.$$

From the lemma we can conclude that any syntactic construction of a programming language can be represented as a mapping, which is a semantic function construction. Since the loop operator is a more complex operator compared to the other ones, the remaining operators are also regular. In the function formula $f_{10} P$ must be of type func, that is, it must be represented as a superposition of basic functions. Recall that the original form P – is a text fragment of a programming language that has a text type and is a passive object. However, the passive object "*text P*" discrete transducer "is able to convert" into the active object func P, which is a functional type. The machine equivalent of a func P object is an exe file named "*P.exe*", if the object "*text P*" corresponds to the type of the file "*P.cpp*" (in case if the program A is written in the C++ programming language). Further, the value metatype appears quite naturally, since it is possible to reduce a functional object «*func P*» to value «*value P*». In other words execute «*func P*» and get the result «*value P*». In the theory of discrete converter concepts text, func, value present implicitly, but in functional grammars it is present explicitly. There are a number of fairly good reasons for this. One of the first is the desire in science of computer science to manipulate functions in both their active phase «func» and in their passive phase «*text*». Such manipulations are indisputably necessary for systems of automatic synthesis of computer programs, automation of programming and artificial intelligence. It will also be quite natural to move in the direction of data activation in programming, in other words, an attempt to present all passive data, starting

ISSN 2782-4934 (online)
ISSN 2949-477X (print)

with integers and real numbers and so on, in the form of functions. This tendency is observed in works [7; 8], where it is always emphasized that all data are presented in the form of data algebra $D=(D,\Omega,\prod)$, where D – support set, elements of which d, which are the values of variables, must also have the form of functions [9; 10], Ω – is operation signature, $\prod$ – is predicate signature. The authors of this point of view were largely influenced by the theory of functional programming, and therefore it is considered necessary to pay close attention to the functional programming Fort. In connection with the foregoing, we turn to the consideration of one of the examples of functional grammars. First of all, we note that in functional grammars it was possible to combine concepts-primitives: context-free grammars, an identification operation (a basic constructively constructed alphabetical mapping between grammars), an operation *eval*, alternative operation | (analog ∨), recursion [11].

So, let's get to the issue of operator <*cycle*> representation as a display within functional grammars. The purpose of this consideration is the question of regularity theorem of the program L in a given programming language L regarding basic operations and functional grammar concepts. The syntax is represented as usual:

$$<cycle> : : \underline{\text{while}}\ \text{u}\ \underline{\text{do}}\ P\ \underline{\text{end}}\ \{f_{10}\}.$$

Semantic function $\{f_{10}\}$ is written as follows:

$$f_{10}=(\underline{func}\ x, y)\ value : (u, P)\ r_1(u, u, P),$$

where $r_1=(\underline{value}\ x, \underline{func}\ x, y)\ value\ (:)\ (\underline{true}, u, P)\ f_4(P, r_1(u, u, P))\ |\ (\underline{false}, u, P)\ e)$,
where e – identical presentation. Note that the function r_1 is recursive, where a recursive call occurs through a though a function f_4. Function f_4 – is a two-argument semantic function that corresponds to the syntax:

$$<operators> : : = <operators> ; <operators> \{f_4\}.$$

If designated by <*operators*> through Q, and <*operators*> through R, then the indicated construction will take the form of $Q^{(1)}: : = Q; R\ \{f_4\}$, where $Q^{(1)}$ corresponds to the function f_4. Function f_4 has two arguments Q and R. Values of the arguments Q and R may have a metatype text, that is, they are syntactically textual arguments to the program, or they are a superposition of functions f_Q and f_R, which have func. However, functions f_Q and f_R can be executed and will get results that will have metatypes value. So the header of the function f_4 has the form $(\underline{value}\ x, y)$ $\underline{value}$:, where x and y are formal arguments, and the actual arguments will be f_0 (corresponds to x) and f_R (corresponds to y). Overall the function f_4 has the form.

$$f_4=(value\ x, y)\ \underline{value} : (VALUE, VALUE^{(1)})\ ALUE^{(1)};$$

Here *value* and $value^{(1)}$ are results of the execution of functions f_Q and f_R.

Function f_{10} has a header $(function\ x, y)\ value$:. Instead of arguments x and y actual arguments are substituted that are u and P. The result is the result of executing a recursive function r_1 with three arguments: u, u, P. First argument u called by value (type value), and the second u and the third argument are called as functions (type func). From the body of a recursive function r_1 further calculation scheme is obvious. From consideration of the loop operator <*cycle*> in the framework of functional grammars, the following conclusion can be drawn: for each syn-

ISSN 2782-4934 (online)
ISSN 2949-477X (print)

tactic construction, you can uniquely specify a function that describes the meaning of this rule. It follows that an arbitrary program is a superposition of functions. And the final conclusion will be the validity of the theorem on the regularity of an arbitrary program A regarding the basic functions of a programming language.

Theorem. For an arbitrary program A, defined in syntactic grammar G_0, function f_A, displaying input to output is regular with respect to basic functions.

Proof. Each syntactic construct from syntactic context-free grammar G_0 programming language has a syntactic function. Then the parsing algorithm of an arbitrary program A as a result, gives a superposition of basis functions, which represents a function f_A of program A. The theorem is proved.

References

1. Mohsin A., Janjua N.K., Islam M.S, Babar M.A. SAM-SoS: A stochastic software architecture modeling and verification approach for complex system-of-systems. IEEE Access. 2020. Vol. 8. Pp. 177580–177603. DOI: https://doi.org/10.1109/ACCESS.2020.3025934.
2. Zabotkina V.I., Boyarskaya E.L. K voprosu o dinamicheskoj konceptual'noj semantike: modelirovanie struktury koncepta preodolenie [The issue of dynamic conceptual semantics revisited: frame modelling of overcoming]. Kognitivnye issledovaniya yazyka. 2020. No. 3(42). Pp. 128–134. EDN: https://www.elibrary.ru/aafphd. (In Russ.)
3. Chen N., Geng S., Li Y. Modeling and verification of uncertain cyber-physical system based on decision processes. Mathematics. 2023. Vol. 11, iss. 19. Art. no. 4122. DOI: https://doi.org/10.3390/math11194122.
4. Granichin O., Uzhva D., Volkovich Z. Cluster flows and multiagent technology // Mathematics. 2021. Vol. 9, iss. 1. Art. no. 22. DOI: https://doi.org/10.3390/math9010022.
5. Kolesnikov A.V., Rumovskaya S.B., Yasinskij E.V. Intellektualizaciya operativno-tekhnologicheskogo upravleniya regional'noj elektroenergetikoj metodami kognitivnyh gibridnyh intellektual'nyh sistem. Chast' 4 [Intellectualization of operational and technological control of regional electric power by cognitive hybrid intelligent systems. Part 4]. Vestnik Baltijskogo federal'nogo universiteta im. I. Kanta. Seriya: Fiziko-matematicheskie i tekhnicheskie nauki. 2021. No. 4. pp. 49–75. EDN: OLYTHB. (In Russ.)
6. Kolyeva N. Organization of multi-access in databases // E3S Web of Conferences. 2021. Vol. 270. Art. no. 01006. DOI: https://doi.org/10.1051/e3sconf/202127001006.
7. Tsarev I.V. Sistema imitacionnogo modelirovaniya dinamicheskih avtomatnyh setej [A system for simulation modeling of dynamic automata networks]. Nauchnyj vestnik Novosibirskogo gosudarstvennogo tekhnicheskogo universiteta. 2018. No. 3(72). Pp. 107–120. DOI: https://doi.org/10.17212/1814-1196-2018-3-107-120. EDN: https://www.elibrary.ru/yowtfb. (In Russ.)
8. Kolyeva N., Gorodnichev V. Analysis of using queuing systems for estimating the performance of computing systems // AIP Conference Proceedings. 2023. Vol. 2812. Art. no. 020088. DOI: https://doi.org/10.1063/5.0161392.
9. Wang Y., Xiong W., Yang J. et al. A robust feedback path tracking control algorithm for an indoor carrier robot considering energy optimization // Energies. 2019. Vol. 12, iss. 10. Art. no. 2010. DOI: https://doi.org/10.3390/en12102010.

10. Grzybek H., Xu S., Gulliver S., Fillingham V. Considering the feasibility of semantic model design in the built-environment. Buildings. 2014. Vol. 4. P. 849–879. DOI: https://doi.org/10.3390/buildings4040849.
11. Shi Z., Lin J., Chen J. et al. Symmetry based material optimization. Symmetry. 2021. Vol. 13, iss. 2. Art. no. 315. DOI: https://doi.org/10.3390/sym13020315.

Источники

1. Mohsin A., Janjua N.K., Islam M.S, Babar M.A. SAM-SoS: A stochastic software architecture modeling and verification approach for complex system-of-systems // IEEE Access. 2020. Vol. 8. P. 177580–177603. DOI: https://doi.org/10.1109/ACCESS.2020.3025934.
2. Заботкина В.И., Боярская Е.Л. К вопросу о динамической концептуальной семантике: моделирование структуры концепта преодоление // Когнитивные исследования языка. 2020. № 3(42). С. 128–134. EDN: AAFPHD.
3. Chen N., Geng S., Li Y. Modeling and verification of uncertain cyber-physical system based on decision processes // Mathematics. 2023. Vol. 11, iss. 19. Art. no. 4122. DOI: https://doi.org/10.3390/math11194122.
4. Granichin O., Uzhva D., Volkovich Z. Cluster flows and multiagent technology // Mathematics. 2021. Vol. 9, iss. 1. Art. no. 22. DOI: https://doi.org/10.3390/math9010022.
5. Колесников А.В., Румовская С.Б., Ясинский Э.В. Интеллектуализация оперативно-технологического управления региональной электроэнергетикой методами когнитивных гибридных интеллектуальных систем. Часть 4 // Вестник Балтийского федерального университета им. И. Канта. Серия: Физико-математические и технические науки. 2021. № 4. С. 49–75. EDN: OLYTHB.
6. Kolyeva N. Organization of multi-access in databases // E3S Web of Conferences. 2021. Vol. 270. Art. no. 01006. DOI: https://doi.org/10.1051/e3sconf/202127001006.
7. Царев И.В. Система имитационного моделирования динамических автоматных сетей // Научный вестник Новосибирского государственного технического университета. 2018. № 3(72). С. 107–120. DOI: https://doi.org/10.17212/1814-1196-2018-3-107-120. EDN: YOWTFB.
8. Kolyeva N., Gorodnichev V. Analysis of using queuing systems for estimating the performance of computing systems // AIP Conference Proceedings. 2023. Vol. 2812. Art. no. 020088. DOI: https://doi.org/10.1063/5.0161392.
9. Wang Y., Xiong W., Yang J. et al. A robust feedback path tracking control algorithm for an indoor carrier robot considering energy optimization // Energies. 2019. Vol. 12, iss. 10. Art. no. 2010. DOI: https://doi.org/10.3390/en12102010.
10. Grzybek H., Xu S., Gulliver S., Fillingham V. Considering the feasibility of semantic model design in the built-environment // Buildings. 2014. Vol. 4. P. 849–879. DOI: https://doi.org/10.3390/buildings4040849.
11. Shi Z., Lin J., Chen J. et al. Symmetry based material optimization // Symmetry. 2021. Vol. 13, iss. 2. Art. no. 315. DOI: https://doi.org/10.3390/sym13020315.

Information about the authors

Kainizhamal E. Iklassova, PhD, Assosiate Professor of the Department "Information and communication". Manash Kozybayev North Kazakhstan university, Petropavlovsk, Republic of Kazakhstan, 150000, Pushkina str. 86. E-mail: kiklasova@mail.ru.

Raushangul N. Kozhakhmetova, Master Sc. (Technics), Senior Lecturer of the Department "Information and communication", Manash Kozybayev North Kazakhstan university, Petropavlovsk, Republic of Kazakhstan, 150000, Pushkina str. 86. E-mail: raushan@mail.ru.

Информация об авторах

Икласова Кайнижамал Есимсеитовна, доктор PhD, доцент кафедры «Информационно-коммуникационные технологии». Северо-Казахстанский университет имени Манаша Козыбаева, 150000, Республика Казахстан, г. Петропавловск, ул. Пушкина, 86. E-mail: kiklasova@mail.ru.

Кожахметова Раушангуль Назаровна, магистр технических наук, старший преподаватель кафедры «Информационно-коммуникационные технологии». Северо-Казахстанский университет имени Манаша Козыбаева, 150000, Республика Казахстан, г. Петропавловск, ул. Пушкина, 86. E-mail: raushan@mail.ru.