ALIAS CALCULUS FOR A SIMPLE IMPERATIVE LANGUAGE

N. V. Shilov[1,2]*

1) School of Science and Technology, Nazarbayev University, Astana, Kazakhstan; 2) A.P. Ershov Institute of Informatics Systems Russian Academy of Science, Novosibirsk, Russia; *nikolay.shilov@nu.edu.kz

Introduction. In programming the aliasing problem is to predict, detect and/or trace pointers to the same addresses in dynamic memory. Importance of the problem is due to mistakes and errors that may happen in program run-time due to improper alias handling. Below are two simple examples of errors of this type:

x = malloc(sizeof(int)); x = malloc(sizeof(int));

y = x; free x; free y;

The first example shows a loss of a link to a piece of memory allocated first (which can result in run out of memory, if iterated); the second example shows an attempt to free a deleted piece of memory (which can result in an abnormal program termination immediately). In spite of 40 years of study the aliasing problem the alias analysis is still very active and competitive field of research [1].

Method and Results. Alias calculus was proposed by Bertrand Meyer [2] for a toy programming language with single data type for abstract pointers. The original calculus is a set of syntax-driven rules to compute an upper approximation aft(S,P) for aliasing after execution of a program P for a given initial aliasing S; this calculus guarantees partial correctness of the assertion {S}P{aft(S,P)}.

The primary purpose of the present research is a variant of alias calculus for more realistic programming language MoRe with decidable pointer arithmetic. Context-free syntax of the language under study is given below:

P::= skip | var V=C | V:=T | V:=cons(T*) |

[V]:=T | V:=[V] | dispose(V) |

(P;P) | (if F then P else P) | (while F do P) where V, C and T are metavariables for program variables, integer constants, and for arithmetic expressions.

The new calculus is safe in the following sense.

Theorem. Let D be any alias distribution, a be any MoRe-program and s be a state that satisfies D. If s' is a state such that s<a>s' then s' satisfies aft(D, a). If a started in s has memory leak and/or invalid memory access then memory-leak and/or invalid-access warning(s) will be casted in aft(D, a).

Conclusion. The calculus is control flow insensitive and uses only stack variables for analysis. Topics for further research include a sensitive to control flow calculus that takes into account some heap information. Then it will make sense to put into research agenda prototyping of a tool on base of the calculus.

Acknowledgments. This research has been supported by Nazarbayev University Seed Grant KF-14/16 "Research of Formal Models for analysis of programs with Dynamic Memory".

References.

1. M. Sridharan, S. Chandra, J. Dolby, SJ. Fink, and E. Yahav, Alias analysis for object-oriented programs. In D. Clarke, T. Wrigstad, and J. Noble, editors, Aliasing in Object-Oriented Programming: types, analysis, and verification. Springer, 2013, Lecture Notes in Computer Science, Vol. 7850, pp. 196-232.
2. B. Meyer, Steps Towards a Theory and Calculus of Aliasing. International Journal of Software and Informatics, special issue (Festschrift in honor of Manfred Broy), 2011, pp. 77-115.