Curriculum Vitae

Kohei Suenaga

October 26, 2009

1 Personal Information

Name:	Kohei Suenaga
Citizenship:	Japan
Email Address:	`ksuenaga@gmail.com`

2 Education

2008 Ph.D in Computer Science from University of Tokyo, JAPAN
2005 M.S. in Computer Science from University of Tokyo, JAPAN
2003 B.S. in Computer Science from University of Tokyo, JAPAN

3 Employment History

1. 2007- : Research Fellow of the Japan Society for the Promotion of Science (DC2)
2. 2008- : Research Fellow of the Japan Society for the Promotion of Science (PD)
3. 2009- : Researcher in Tokyo Research Laboratory, IBM Japan

4 Professional Society Memberships

ACM (SIGPLAN); IEEE; Japan Society for Software Science and Technology.

5 Research Experience

1. 2008–Present: **Fractional Ownerships for Safe Memory Deallocation**

   We have proposed a type system for a programming language with memory allocation/deallocation primitives, which prevents memory-related errors

such as double-frees and memory leaks. The main idea is to aug- ment pointer types with fractional ownerships, which express both capabilities and obligations to access or deallocate memory cells. By assigning an ownership to each pointer type constructor (rather than to a variable), our type system can properly reason about list/tree-manipulating programs. Furthermore, thanks to the use of fractions as ownerships, the type system admits a polynomial-time type inference algorithm, which serves as an algorithm for automatic verification of lack of memory-related errors. A prototype verifier has been implemented and tested for C programs.

The current verifier requires users to explicitly pass ownerships among pointers by *assertions*. We are now developing a method to automatically insert assertions into programs.

References: [9]

2. 2006–Present: **Formal Verification for Concurrent Programs**

   The aim of this research is to establish a method for verification of certain critical properties (such as deadlock- and race-freedom) of concurrent programs. Since many real-world concurrent programs, such as operating system kernels, make heavy use of threads and interrupts, it is important that the method can properly deal with both of the two features.

   As a first step towards the goal, we have formalized a concurrent calculus equipped with primitives for threads and interrupts handling [8]. We have also proposed a type system that guarantees deadlock-freedom in the presence of interrupts in [8]. To our knowledge, ours has been the unique type system for deadlock-freedom that can deal with both thread and interrupt primitives.

   We have also designed a deadlock-freedom verification method for programs with non-block-structured lock primitives and mutable references [6]. Deadlock-freedom verification for programs with those two features had not been developed yet.

   In my Ph.D thesis [7], we have merged those two methods to a type-based verification for concurrent programs with (1) non-block-structured lock primitives (2) mutable references and (3) interrupts.

   References: [6, 8]

3. 2005–2006: **Resource Usage Analysis for the π-calculus**

   We have proposed a type-based resource usage analysis for the π-calculus extended with resource creation/access primitives. The goal of the resource usage analysis is to statically check that a program accesses resources such as files and memory in a valid manner. Our type system is an extension of previous behavioral type systems for the pi-calculus, and can guarantee the safety property that no invalid access is performed, as

well as the property that necessary accesses (such as the close operation for a file) are eventually performed unless the program diverges. A sound type inference algorithm for the type system is also developed to free the programmer from the burden of writing complex type annotations. Based on the algorithm, we have implemented a prototype resource usage analyzer for the π-calculus. To our knowledge, ours is the first type-based resource usage analysis that deals with an expressive concurrent language like the π-calculus.

References: [1, 2]

4. 2003–Present: **Translation of Tree-Processing Programs into Stream-Processing Programs Based on Ordered Linear Types**

   There are two ways to write a program for manipulating tree-structured data such as XML documents and S-expressions: One is to write a tree-processing program focusing on the logical structure of the data and the other is to write a stream-processing program focusing on the physical structure. While tree-processing programs are easier to write than stream-processing programs, tree-processing programs are less efficient in memory usage since they use trees as intermediate data.

   The goal of this study is to establish a method for automatically translating a tree-processing program to a stream-processing one in order to take the best of both worlds. To achieve the goal, we first introduce a statically-typed language that accepts only tree-processing programs that traverse input trees from left to right in the depth-first order, and show an algorithm for translating well-typed tree-processing programs into stream-processing programs [3, 4]. We then remove the restriction on the access order by extending the language with primitives for selectively buffering part of trees on memory.

   With the extended language, programmers can write arbitrary tree-processing, but inserting the buffering primitives manually is sometimes tedious. We therefore also develop a type-based algorithm that inputs arbitrary tree-processing programs and automatically inserts the buffering primitives [10].

   Though the extended framework enables every simply-typed tree-processing programs to be translated into a stream-processing program, the resulting programs sometimes introduce redundant buffering of input data. This is because, in real-world programs, many trees are accessed twice or more, so that they are buffered, but only a part of such trees are actually used. To solve this problem, we extend the framework above by introducing ordered, *non-linear* types in addition to ordered linear types [5]. A tree with an ordered, non-linear type is read lazily on memory, so that if a part of the tree is not used, it can be discarded without read on memory. The resulting transformation framework reduces the redundant buffering, generating more efficient stream-processing programs.

References: [3–5, 10]

5. 2002–2003: **The Interface Definition Language for Fail-Safe C**

   Fail-Safe C is a safe implementation of full ANSI-C. It uses its own internal data representations such as 2-word pointers and memory blocks with headers describing their contents. Because of this, calls to external functions compiled by conventional compilers require conversion of data representations. Moreover, for safety, many of those functions need additional checks on their arguments and return values. This paper presents a method of semi-automatically generating a wrapper doing such work. Our approach is to develop an Interface Definition Language to describe what the wrappers have to do before and after function calls. Our language is based on CamlIDL, which was developed for a similar purpose between Objective Caml and C. Our IDL processor generates code by using the types and *attributes* of functions. The attributes are additional information describing properties which cannot be expressed only by ordinary types, such as whether a pointer can be `NULL`, what range of memory can be safely accessed via a pointer, etc. We examined Linux system calls as test cases and designed a set of attributes required for generating their wrapper.

   References: [11]

6 Educational Experience

1. 2003: Teaching assistant of "Compiler Lab" in Department of Information Science, University of Tokyo.
2. 2004-2005: Teaching assistant of "ML Lab" in Department of Information Science, University of Tokyo.

Publications

[1] Naoki Kobayashi, Kohei Suenaga, and Lucian Wischik. Resource usage analysis for the π-calculus. In *Proceedings of 7th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2006), Charleston, SC, USA*, volume 3855 of *Lecture Notes in Computer Science*, pages 298–312, January 2006.

[2] Naoki Kobayashi, Kohei Suenaga, and Lucian Wischik. Resource usage analysis for the π-calculus. *Logical Methods in Computer Science*, 2(3:4), 2006.

[3] Koichi Kodama, Kohei Suenaga, and Naoki Kobayashi. Translation of tree-processing programs into stream-processing programs based on ordered

linear type. In *Proceedings of the 2nd Asian Symposium on Programming Languages and Systems (APLAS 2004), Taipei, Taiwan*, volume 3302 of *Lecture Notes in Computer Science*, pages 41–56, November 2004.

[4] Koichi Kodama, Kohei Suenaga, and Naoki Kobayashi. Translation of tree-processing programs into stream-processing programs based on ordered linear type. *Journal of Functional Programming*, 18(3):333–371, January 2007.

[5] Kohei Suenaga Ryosuke Sato and Naoki Kobayashi. Ordered types for stream processing of tree-structured data. In *Programming Language Techniques for XML (PLAN-X 2009)*, January 2009.

[6] Kohei Suenaga. Type-based deadlock-freedom verification for non-block-structured lock primitives and mutable references. In *Proceedings of the 6th ASIAN Symposium on Programming Languages and Systems (APLAS 2008), Bangalore, India*, volume 5356 of *Lecture Notes in Computer Science*, pages 155–170, December 2008.

[7] Kohei Suenaga. *Type Systems for Formal Verification of Concurrent Programs*. PhD thesis, University of Tokyo, March 2008.

[8] Kohei Suenaga and Naoki Kobayashi. Type-based analysis of deadlock for a concurrent calculus with interrupts. In *Proceedings of 16th European Symposium on Programming (ESOP 2007),Braga,Portugal*, volume 4421 of *Lecture Notes in Computer Science*, March 2007.

[9] Kohei Suenaga and Naoki Kobayashi. Fractional ownerships for safe memory deallocation. In *Proceedings of the 7th ASIAN Symposium on Programming Languages and Systems (APLAS 2009). To appear.*, December 2009.

[10] Kohei Suenaga, Naoki Kobayashi, and Akinori Yonezawa. Extension of type-based approach to generation of stream-processing programs by automatic insertion of buffering primitives. In *Proceedings of International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2005)*, number 3901 in Lecture Notes in Computer Science, pages 98–114, September 2005.

[11] Kohei Suenaga, Yutaka Oiwa, Eijiro Sumii, and Akinori Yonezawa. The interface definition language for Fail-Safe C. In *Proceedings of International Symposium on Software Security (ISSS 2003)*, volume 3855 of *Lecture Notes in Computer Science*, pages 192–208, November 2003.