Curriculum Vitae

Kohei Suenaga

October 1, 2013

1 Personal Information

Name:	Kohei Suenaga
Nationality:	Japan
Address	Bldg. Eng. 10, Yoshida Honmachi, Sakyo-ku, Kyoto, Kyoto, 606-8501, Japan
Phone	+81-75-753-5968
Email Address:	`ksuenaga@kuis.kyoto-u.ac.jp`
Homepage:	`http://www.fos.kuis.kyoto-u.ac.jp/~ksuenaga/`

2 Education

2008 Ph.D in Information Science and Technology from the University of Tokyo, JAPAN
2005 M.S. in Information Science and Technology from the University of Tokyo, JAPAN
2003 B.S. from the University of Tokyo, JAPAN

3 Employment History

1. April 2003 – September 2005 : Research Assistant of Prof. Akinori Yonezawa in the University of Tokyo
2. October 2005 – March 2007 : Research Assistant of Prof. Naoki Kobayashi in Tohoku University
3. April 2007 – March 2008 : Research Fellow of the Japan Society for the Promotion of Science (DC2)
4. April 2008 – March 2009 : Research Fellow of the Japan Society for the Promotion of Science (PD)
5. April 2009 – March 2010 : Researcher in Tokyo Research Laboratory, IBM Japan
6. April 2010 – January 2011 : Post doctoral researcher at Department of Informatics, Faculdade de Ciências da Universidade de Lisboa

7. February 2011 – March 2011 : Post doctoral researcher at Graduate School of Informatics, Kyoto University

8. April 2011 – March 2012 : Research Fellow of the Japan Society for the Promotion of Science (PD)

9. April 2012 – September 2013: Assistant Professor at Hakubi Center for Advanced Research, Kyoto University

10. October 2013 – : Associate Professor at Graduate School of Informatics, Kyoto University

4 Research Experience

1. 2011–Present: **Hybrid System Modeling and Verification Based on *Infinitesimal Programming***

   We add, to the common combination of a WHILE-language and a Hoare-style program logic, a constant `dt` that represents an infinitesimal (i.e. infinitely small) value. The outcome is a framework for modeling and verification of *hybrid systems*: hybrid systems exhibit both continuous and discrete dynamics and getting them right is a pressing challenge. We rigorously define the semantics of programs in the language of *nonstandard analysis*, on the basis of which the program logic is shown to be sound and relatively complete.

   We have also implemented a prototype verifier based on our methodology and confirmed that it can verify several relevant examples including one with Zeno behaviors. We are currently extending the prototype so that it can deal with more examples.

   As another direction, we have exploited the apparent similarity between (discrete-time) stream processing and (continuous-time) signal processing and transfer a deductive verification framework for the former to the latter. Our development is based on rigorous semantics that relies on nonstandard analysis (NSA). Specifically, we start with a discrete framework consisting of a Lustre-like stream processing language, its Kahn-style fixed point semantics, and a program logic (in the form of a type system) for partial correctness guarantees. This stream framework is transferred as it is to one for *hyperstreams*streams of streams, that typically arise from sampling (continuous-time) signals with progressively small intervalsvia the logical infrastructure of NSA. Under a certain continuity assumption we identify hyperstreams with signals; our final outcome thus obtained is a deductive verification framework of signals. In it one verifies properties of signals using the (conventionally discrete) proof principles, like fixed point induction.

   References: Joint work with Ichiro Hasuo and Hiroyoshi Sekine [**?**, **?**, **?**].

2. 2008–Present: **Fractional Ownerships for Safe Resource 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 augment 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 FreeSafeTy has been implemented and tested for C programs.

   We have also extended the framework to concurrency. The resulting framework now guarantees safe *resource* deallocation. Here, by resource, we mean locks and thread IDs as well as memory cells. We use the same idea as the previous type system for safe memory deallocation: fractional ownerships for pointer-type constructors. However, we need to deal with ownership transfer by synchronization with locks and fork/join in the world with concurrency. To this end, we augment lock types and thread ID types with *procurable type environments*. A procurable type environment describes the ownerships granted by using locks and thread IDs. The type system guarantees that ownership transfer by synchronization is done appropriately and guarantees that resource leak does not occur. We also designed type inference algorithm. As a side-product of the use of fractional ownerships, the new type system also guarantees race-freedom.

   The current verifier requires a user to explicitly pass ownerships around pointers by *assertions* if necessary. We are now developing a method to automatically insert appropriate assertions into programs.

   References: Joint work with Naoki Kobayashi, Atsushi Igarashi and Ryota Fukuda [**?**, **?**].

3. 2006–2009: **Type-Based Deadlock-Freedom Verification for Concurrent Programs with Interrupts**

   The aim of this research is to establish a method for verification of certain critical properties (such as deadlock- and race-freedom) of real-world software that is written as concurrent programs. As a first step towards the goal, we have formalized a concurrent calculus equipped with primitives for concurrency and interrupts and proposed a type system that guarantees deadlock-freedom in the presence of interrupts [**?**]. To our knowledge, this had been the unique type system for deadlock-freedom that can deal with both concurrency and interruption. We have also designed a deadlock-freedom verification method for programs with non-block-structured lock

primitives and mutable references [**?**]. Type-based deadlock-freedom verification for programs with those two features had not been developed yet until [**?**]. In my Ph.D thesis [**?**], 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.

We have concurrently investigated methods for user-friendly presentation of type errors based on type error slicing. The main idea is to associate locations in an input program with each type constraint. If a constraint set is unsatisfiable, our tool calculates an unsatisfiable core (a subset of the original constraint set that is unsatisfiable) and extracts the locations that are involved in the unsatsfiable core. Then, the tool slices the input program using the extracted locations. We applied this idea to the type system for deadlock and race detection for π-calculus [**?**, **?**].

References: Joint work with Naoki Kobayashi and Eri Iimura [**?**, **?**, **?**, **?**].

4. 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: Joint work with Naoki Kobayashi and Lucian Wischik [**?**, **?**].

5. 2003–2010 : **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 this goal, we first introduced 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 [?, ?, ?]. 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 [?].

Though the extended framework enables every simply-typed tree-processing program to be translated into a stream-processing program, the resulting programs sometimes do not become efficient on memory consumption as they should be. This is mainly because, in actual programs, many trees that are accessed twice or more (thus are buffered in the type system in [?, ?, ?, ?]) 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 [?, ?]. A tree with an ordered, non-linear type is constructed lazily on memory, thus if a part of the tree is not used, it can be discarded without consuming memory space. The resulting transformation framework reduces the redundant buffering, generating more efficient stream-processing programs.

References: Joint work with Naoki Kobayashi, Koichi Kodama, Ryosuke Sato and Akinori Yonezawa [?, ?, ?, ?, ?, ?, ?].

6. 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: Joint work with Yutaka Oiwa, Eijiro Sumii and Akinori Yonezawa [**?**, **?**].

5 Awards

1. 2011 Journal of Information Processing Outstanding Paper Award [**?**]
2. Presentation Award in the 14th Workshop on Programming and Programming Languages (PPL 2012)
3. Presentation Award in the 11th Workshop on Programming and Programming Languages (PPL 2009)
4. Presentation Award in the 9th Workshop on Programming and Programming Languages (PPL 2007)

6 Educational Experience

1. 2003: Teaching assistant of "Compiler Lab" in Department of Information Science, University of Tokyo.

   This lab is for construction of compilers for functional languages. I gave lecture once a week, answered questions from students and grading submitted reports. The topics covered in this lab include

   - Generating a lexer/parser using lex/yacc
   - Optimization (e.g., constant folding, removing redundant variable bindings, inlining etc.)
   - Closure conversion
   - Generating assembly code
   - Type system of ML and implementation of the type inference algorithm and
   - Some advanced topics about type-based program analysis.

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

3. 2012: Teaching assistant of "Functional Programming and Program Verification Lab" in Department of Information Science, Kyoto University.

   These labs are for learning Objective Caml and type checking/inference. I gave lecture once a week, answered questions from students and grading submitted reports. The topics covered in this lab include

- Data types of OCaml
- Type system of OCaml
- Module system of OCaml
- Evaluation strategies
- Implementing an interpreter and the type inference algorithm of MiniML.

7 Services

- General chair of PPL 2014.
- Poster and Demo chair of APLAS 2012.
- Program committee member of SACSIS 2013 and PEPM 2012.
- Local organizer of NII Syonan Meeting: Hybrid Systems: Theory and Practice, Seriously.
- External reviewer of ESOP 2013, FoSSaCS 2013, COB 2012, APLAS 2012, ECOOP 2012, LICS 2012, TACAS 2011, FoSSaCS 2011, ICFEM 2010, INFORUM 2010, APSEC2010, CONCUR 2010, ICALP 2010, LICS 2010, FoSSaCS 2009, POPL 2009, APLAS 2008, CiE 2008, CONCUR 2008, SAS 2007 and SAS 2006.

8 Professional Society Memberships

ACM (SIGPLAN); Japan Society for Software Science and Technology; Information Processing Society of Japan.

Publications

[1] Ichiro Hasuo and Kohei Suenaga. Exercises in nonstandard static analysis of hybrid systems. In P. Madhusudan and Sanjit A. Seshia, editors, *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 462–478. Springer, 2012.

[2] Eri Iimura, Naoki Kobayashi, and Kohei Suenaga. Slicing for type-based race analysis and presentation of the result (in japanese). In *Proceedings of the 9th Workshop on Programming and Programming Languages (PPL 2007)*, March 2007.

[3] Eri Iimura, Naoki Kobayashi, and Kohei Suenaga. Identifying deadlock errors by type error slicing (in japanese). *Transactions of Information Processing Society of Japan (Programming)*, 1(2):71–84, 2008.

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

[5] Naoki Kobayashi, Kohei Suenaga, and Lucian Wischik. Resource usage analysis for the *pi*-calculus. In E. Allen Emerson and Kedar S. Namjoshi, editors, *VMCAI*, volume 3855 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 2006.

[6] Koichi Kodama, Naoki Kobayashi, and Kohei Suenaga. Translation of tree-processing programs into stream-processing programs based on ordered linear types (in japanese). In *Proceedings of the 6th Workshop on Programming and Programming Languages*, March 2004.

[7] Koichi Kodama, Kohei Suenaga, and Naoki Kobayashi. Translation of tree-processing programs into stream-processing programs based on ordered linear type. In Wei-Ngan Chin, editor, *APLAS*, volume 3302 of *Lecture Notes in Computer Science*, pages 41–56. Springer, 2004.

[8] Koichi Kodama, Kohei Suenaga, and Naoki Kobayashi. Translation of tree-processing programs into stream-processing programs based on ordered linear type. *J. Funct. Program.*, 18(3):333–371, 2008.

[9] 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.

[10] Ryosuke Sato, Kohei Suenaga, and Naoki Kobayashi. Ordered types for stream processing of tree-structured data. In *Proceedings of Forum on Information Technology 2009*, September 2009.

[11] Ryosuke Sato, Kohei Suenaga, and Naoki Kobayashi. Ordered types for stream processing of tree-structured data. *JIP*, 19:74–87, 2011.

[12] Kohei Suenaga. Type-based deadlock-freedom verification for non-block-structured lock primitives and mutable references. In G. Ramalingam, editor, *APLAS*, volume 5356 of *Lecture Notes in Computer Science*, pages 155–170. Springer, 2008.

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

[14] Kohei Suenaga, Ryota Fukuda, and Atsushi Igarashi. Type-based safe resource deallocation for shared-memory concurrency. In Gary T. Leavens and Matthew B. Dwyer, editors, *OOPSLA*, pages 1–20. ACM, 2012.

[15] Kohei Suenaga and Ichiro Hasuo. Programming with infinitesimals: A while-language for hybrid system modeling. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP (2)*, volume 6756 of *Lecture Notes in Computer Science*, pages 392–403. Springer, 2011.

[16] Kohei Suenaga and Naoki Kobayashi. Type-based analysis of deadlock for a concurrent calculus with interrupts. In Rocco De Nicola, editor, *ESOP*, volume 4421 of *Lecture Notes in Computer Science*, pages 490–504. Springer, 2007.

[17] Kohei Suenaga and Naoki Kobayashi. Fractional ownerships for safe memory deallocation. In Zhenjiang Hu, editor, *APLAS*, volume 5904 of *Lecture Notes in Computer Science*, pages 128–143. Springer, 2009.

[18] 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 Patricia M. Hill, editor, *LOPSTR*, volume 3901 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2005.

[19] Kohei Suenaga, Yutaka Oiwa, Eijiro Sumii, and Akinori Yonezawa. The interface definition language for fail-safe c. In *Proceedings of the 5th Workshop on Programming and Programming Languages (PPL 2003)*, March 2003.

[20] Kohei Suenaga, Yutaka Oiwa, Eijiro Sumii, and Akinori Yonezawa. The interface definition language for fail-safe c. In Kokichi Futatsugi, Fumio Mizoguchi, and Naoki Yonezaki, editors, *ISSS*, volume 3233 of *Lecture Notes in Computer Science*, pages 192–208. Springer, 2003.

[21] Kohei Suenaga, Hiroyoshi Sekine, and Ichiro Hasuo. Hyperstream processing systems: nonstandard modeling of continuous-time signals. In Roberto Giacobazzi and Radhia Cousot, editors, *POPL*, pages 417–430. ACM, 2013.