Zaynah DARGAYE
85, rue Emile Bollaert
75019, Paris

Phone : 33 6 64 29 39 46
Email : zaynah.dargaye@inria.fr
Website : gallium.inria.fr/∼dargaye/

Date of birth 08/17/1980
Citizenship French

Education

2008–2009	**ATER** *teaching assistant* projet Cedric-CPR, at ENSIIE (French engineering school)
2005–	**Ph. D. candidate in Computer Science** *Mechanized verification of functional language optimizing compilation (supervisor :Xavier Leroy* projet Gallium, INRIA Rocquencourt *(funding :regional Ile de France)*

Education

2004–2005	**Master of Computer Science research, MPRI** *grade B* Paris 7 University
2003–2004	**M1 in Computer Science** *grade B* Paris 7 University
2002–2003	**Licence(3) in Computer Sciences** *grade B* Paris 7 University
2000–2002	**Deug MIAS(L2)** *grade B* Paris 7 University

Publications and developments

2007	**Mechanized Verification of CPS transformation** ,*LPAR'07 Logic for Programming Artificial Intelligence and Reasoning with X. Leroy LNCS 4790, pages 211-225. Springer, 2007.*
2007	**Décurryfication certifée** , *JFLA'07 Journées francophones des langages applicatifs (only in French), pages 119-133.*
2006	**Formal verification of a C compiler front-end** , *FM'06 Formal Methods with S. Blazy and X. Leroy. LNCS 4085, pages 460-475. Springer, 2006.*
2004	**Hyperlog Software** , http ://membres.lycos.fr/hyperlog/

Teaching

Jan-Feb 2009	**Formal specification** *6h tutorial* M1, ENSIIE *Resp. Sandrine Blazy*
Jan 2009	**Imperative Programming in C** *14h tutorial* L3, ENSIIE *Resp. Julien Forest*
Falls 2008	**Imperative Programming introduction to C** *21h tutorial* L3, ENSIIE *Resp. Renaud Rioboo*
Falls 2008	**Compilation** *14h tutorial* M1, ENSIIE *Resp. Sandrine Blazy*
Falls 2008	**Functional Programming Introduction to OCaml** *13h tutorial* L3, ENSIIE *Resp. Catherine Dubois*
April 2007	**Church $\lambda-$calculus** *2h lecture, Mechanized Proofs* M1 Computer Science, Université paris 7 *Resp. Alexandre Miquel*
April 2007	**Lists in Coq** *2h tutorial,Mechanized Proofs* M1 Computer Science, Université Paris 7 *Resp. Alexandre Miquel*

Scientific Visit

26/04 to 16/05/08	**Visiting PhD Fellows training Site at BRICS** Aarhus, Danemark *Inviting by Olivier Danvy*

Language

Native French and Fluent English

Ph.D. subject

As part of formal verification of critical software, preserving properties established on the source code in the executable code seems to be crucial. To have this preservation, the compiler has to be verified itself. A compiler is formally verified if it is joined with a proof of semantic preservation : *the behavior of the compiled code preserves the source code behavior, if the compilation succeeds.*
The CompCert project (http ://compcert.inria.fr) investigates the formal verification of realistic compilers usable for critical embedded software. The project designs, develops and mechanically verifies compilers within the Coq Proof Assistant. By this method, a C compiler producing PowerPC assembly code has already be developed and verified. Using the extraction mechanism of Coq, the compiler is automatically extracted into OCaml code, which is compiled by the Objective Caml system. Actually, the production of the executable compiler uses (or used to use) two unverified processes : the extraction mechanism and the Objective Caml compiler. In fact, this is true for any specified development in the Coq Proof Assistant when the target is to obtain an executable.
My thesis deals with the design, development and mechanized verification, in the Coq Proof Assistant, of a compiler for the purely functional fragment of ML, which is the language of extracted from Coq extraction. Concretely, a front-end from miniML (λ-calculus, let, letrec, pattern-matching) to Cminor has been developed. Cminor is a low-level C-like language, that is the first intermediate language of the CompCert back-end.

Such as the source language is expressive, the compiler is realistic. Classical functional language compilation optimizations are done : uncurrying (the same optimization as in OCaml), uniform data structure representation (as numbering constructor and closure conversion) and an optimizing CPS translation. As in modern compiler for high-level languages, the miniML compiler can interact with a memory manager. This interaction has been mechanized verified.