Programa del Encuentro de Lógica y Métodos Formales
19 y 20 de Diciembre de 2013

Jueves 19 de diciembre - Salón de Posgrados InCo

<table>
<thead>
<tr>
<th>Time</th>
<th>Speaker(s)</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:00-9:30</td>
<td>Juan José Cabezas</td>
<td>Apertura</td>
</tr>
<tr>
<td>9:30-10:15</td>
<td>Marcos Viera</td>
<td>First Class Syntax, Semantics, and their Composition</td>
</tr>
<tr>
<td>10:15-10:45</td>
<td></td>
<td>*** CAFÉ ***</td>
</tr>
<tr>
<td>10:45-11:30</td>
<td>Alberto Pardo</td>
<td>Just Do It While Compiling! Fast Extensible Records In Haskell</td>
</tr>
<tr>
<td>11:30-12:15</td>
<td>Guillermo Calderón</td>
<td>Programación Genérica con Tipos Dependientes</td>
</tr>
<tr>
<td>12:15-13:00</td>
<td>Ernesto Copello</td>
<td>Aprender a Programar con Tipos Dependientes en Agda</td>
</tr>
<tr>
<td>13:00-14:30</td>
<td></td>
<td>*** ALMUERZO ***</td>
</tr>
<tr>
<td>14:30-16:00</td>
<td>Gustavo Betarte, Juan Diego Campo y Carlos Luna</td>
<td>A certified idealized model of virtualization</td>
</tr>
<tr>
<td>16:00-16:15</td>
<td></td>
<td>*** CAFÉ ***</td>
</tr>
<tr>
<td>16:15-17:00</td>
<td>Sergio Yovine</td>
<td>Correct and efficient parallel programs through formal transformations</td>
</tr>
<tr>
<td>Time</td>
<td>Speaker</td>
<td>Title</td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>9:00-9:45</td>
<td>Alvaro Tasistro</td>
<td>Advances in Nominal Abstract Syntax</td>
</tr>
<tr>
<td>9:45-10:30</td>
<td>Ana Bove</td>
<td>Combining Interactive and Automatic Reasoning about Functional Programs</td>
</tr>
<tr>
<td>10:30-11:00</td>
<td>*** CAFÉ ***</td>
<td></td>
</tr>
<tr>
<td>11:00-11:45</td>
<td>Daniel Fridlender</td>
<td>Chequeo de tipos basado en normalización por evaluación para la teoría de tipos de Martin-Löf con subtipos</td>
</tr>
<tr>
<td>11:45-12:30</td>
<td>Daniel Calegari</td>
<td>Métodos Formales e Ingeniería Dirigida por Modelos</td>
</tr>
<tr>
<td>12:30-14:00</td>
<td>*** ALMUERZO ***</td>
<td></td>
</tr>
<tr>
<td>14:00-14:45</td>
<td>Etienne Miquey</td>
<td>Título a confirmar</td>
</tr>
<tr>
<td>14:45-15:30</td>
<td>Mauricio Guillermo</td>
<td>Especificación de la Ley de Peirce en Realizabilidad Clásica</td>
</tr>
<tr>
<td>15:30-16:00</td>
<td>*** CAFÉ ***</td>
<td></td>
</tr>
<tr>
<td>16:00-16:45</td>
<td>Alexandre Miquel</td>
<td>Classical realizability and forcing</td>
</tr>
</tbody>
</table>
Resúmenes de las presentaciones

Jueves 19 de diciembre - Salón de Posgrados InCo

First Class Syntax, Semantics, and Their Composition

Marcos Viera
InCo, Facultad de Ingeniería, Universidad de la República

Ideally complexity is managed by composing a system out of quite a few, more or less independent, and much smaller descriptions of various aspects of the overall artefact. When describing (extensible) programming languages, attribute grammars have turned out to be an excellent tool for modular definition and integration of their different aspects. We show how to construct a programming language implementation by composing a collection of attribute grammar fragments describing separate aspects of the language. More specifically we describe a coherent set of libraries and tools which together make this possible in Haskell, where the correctness of the composition is enforced through the Haskell type systems ability to represent attribute grammars as plain Haskell values and their interfaces as Haskell types makes this possible. Semantic objects thus constructed can be combined with parsers which are constructed on the fly out of parser fragments and are also represented as typed Haskell values. Again the type checker prevents insane compositions. Using a very small example language and some simple extensions, we show how all our techniques fit together towards the construction of extensible compilers out of a collection of pre- compiled, statically type- checked language definition fragments.

Just Do It While Compiling! Fast Extensible Records In Haskell

Alberto Pardo
InCo, Facultad de Ingeniería, Universidad de la República

The library for strongly typed heterogeneous collections HList provides an implementation of extensible records in Haskell. In HList, records are represented as linked lists of label-value pairs with a lookup operation that is linear-time in the number of elds. Using type-level programming techniques we develop a more efficient representation of extensible records. We propose two encodings for extensible records that improve lookup at runtime without needing a total order on the labels. One of the encodings performs lookup in constant time but at a cost of linear time insertion. The other one performs lookup in logarithmic time while preserving the fast insertion of simple linked lists. Through staged compilation, the required slow search for a eld is moved to compile time in both cases. This is a joint work with Bruno Martinez and Marcos Viera.
Programación Genérica con Tipos Dependientes
Guillermo Calderón
InCo, Facultad de Ingeniería, Universidad de la República

Se presenta una visión general (estado del arte) sobre trabajos que investigan las relaciones e interacciones entre las áreas de Programación Genérica y Tipos Dependientes.
La Programación Genérica es una forma de programación en la cual una función recibe un tipo como argumento y su comportamiento depende de la estructura de este tipo.
Un lenguaje con Tipos Dependientes contiene tipos que dependen de valores. Típicamente se utilizan en asistentes de pruebas basados en teoría de tipos y/o en lenguajes funcionales con sistemas de tipos más expresivos que los habituales.
En esta charla se describen varios artículos publicados en los últimos años que vinculan ambas disciplinas. En algunos casos se trata de la representación y formalización de conceptos de programación genérica en un lenguaje con tipos dependientes; en otros se estudia la implementación de metodologías de la programación genérica para ser aplicadas en un lenguaje con tipos dependientes.

Aprender a Programar con Tipos Dependientes en Agda
Ernesto Copello
Facultad de Ingeniería, Universidad ORT Uruguay

Presentamos un caso interesante de programación con Tipos Dependientes, donde la información de tipos enriquecida dentro de los algoritmos, ayuda a obtener pruebas de corrección más directas. Presentamos así el ejemplo de corrección parcial del merge sort, presentado originalmente en Haskell con extensiones, y llevado de forma directa a tipos dependientes. Comenzamos con una primera versión ideal con Tipos Dependientes, para luego comparar con la efectivamente lograda en el asistente de pruebas de Agda. Finalmente, presentamos distintas líneas de trabajo que venimos desarrollando.

VirtualCert: A certified idealized model of virtualization
Gustavo Betarte, Juan Diego Campo, Carlos Luna
InCo, Facultad de Ingeniería, Universidad de la República
Joint work with Gilles Barthe, IMDEA Software, Madrid, Spain.

Virtualization platforms, and more specifically hypervisors, allow several operating systems to coexist on commodity hardware, and provide support for multiple applications to run seamlessly on the guest operating systems they manage. Moreover, hypervisors provide a means to guarantee that applications with different security policies can execute securely in parallel, by ensuring isolation between their guest operating systems. In effect, hypervisors
are increasingly used as a means to improve system flexibility and security, and their use has become ubiquitous in enterprise data centers and cloud computing. The increasingly important role of hypervisors in software systems makes them a prime target for formal verification. Indeed, several projects have set out to formally verify the correctness of hypervisor implementations.

VirtualCert is a machine-checked model of virtualization that can be used to reason about isolation between operating systems in presence of cache-based side-channels. VirtualCert abstracts away most implementations issues and specifies the effects of hypervisor actions axiomatically, in terms of preconditions and postconditions.

We have developed, in the first place, a minimalistic idealized model of a hypervisor, and have formally proved that the hypervisor correctly enforces isolation between guest operating systems. For tractability and concreteness, we concentrate on memory management in paravirtualization platforms. Our model abstracts away many specifics of the underlying hardware and runtime environment such as I/O devices. Instead, our model focuses on the aspects that are most relevant for isolation properties, namely read and write resources on machine addresses, and is sufficiently complete to allow us to reason about isolation properties. Overall, our verification effort shows that the model is adequate to reason about safety properties (read and write isolation), 2-safety properties (OS isolation), and liveness properties (availability). This work has been reported in [Barthe et al 2011].

Virtualization platforms, unfortunately, are often vulnerable to practical side-channel attacks. Cache attacks are a common class of side-channel attacks that use the cache as a side channel. Therefore, we have developed an extension of the idealized model in [Barthe et al 2011] with a formalization of the cache and Translation Lookaside Buffer (TLB), a specific cache used to map virtual addresses to physical addresses. Then, drawing inspiration from physically observable cryptography [Micali; Reyzin 2004], we consider an extended model of traces in which operating systems can draw observations on the history of the cache. The main contribution of this work is a machine-checked proof of isolation in an idealized model of virtualization where the cache and the TLB may leak information. We have also stated and machine-checked a proof of transparency. Transparency states that the virtualization platform is a correct abstraction of the underlying hardware, in the sense that a guest operating system cannot distinguish whether it executes alone or together with other systems. This work has been reported in [Barthe et al 2012].

The models and proofs described above have all been machine-checked in the Coq proof assistant [Coq manual] and the corresponding Coq code can be found in the VirtualCert project site [VirtualCert 2013]. In this talk we shall overview the VirtualCert project in further details and shall discuss ongoing research.

Referencias

Correct and efficient parallel programs through formal transformations

Sergio Yovine
Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Argentina y Facultad de Ingeniería, Universidad ORT Uruguay

One of the questions that the software engineering community faces now is how to develop in a cost-efficient way correct concurrent software that exploits the raw performance of parallel hardware. With the aim to tackling this issue, we have started developing a language, called FXML, and its associated compilation chain. FXML does not rely on any concrete execution model of concurrency, so it allows for multiple implementations of a program. To achieve this, the basic FXML can be extended with constructs for expressing more concrete mechanisms and execution models of concurrency. Then, an FXML compilation-chain is a sequence of program transformations which are only allowed to restrict the degree of parallelism. This approach allows generating different implementations of the same parallel program.
Advances in Nominal Abstract Syntax

Alvaro Tasistro
Facultad de Ingeniería, Universidad ORT Uruguay

Nominal Abstract Syntax is a framework for languages with bound names introduced by Jamie Gabbay and Andy Pitts some years ago. A recent account is to be found in \(^1\). In this setting, an expression with a binder is not interpreted as a certain constructor applied to a function in e.g. lambda notation. And, since the entities that are bound are actually names, they don’t disappear as in de Bruijn’s notation. Rather, the notion of bound name is taken as basic and can be given the interpretation of just local identifier.

The resulting language is purely first-order and has an interesting meta-theory whose most interesting result is that the framework can be used for providing a representation of inductive data types of abstract syntax up to \(\alpha\)-equivalence, thus admitting induction and recursion principles while at the same time identifying \(\alpha\)-equivalent expressions. I shall first try to explain how this works and explore some logic and programming technology that has or can be obtained by implementing the preceding idea. Also I will describe two other applications on which we have been working lately. One is proof assistants for First-Order Logic that work at the level of schematic formulæ, thus actually corresponding to so called One-and-a-Halfth-Order Logic. And the other is the typing of nominal terms (i.e. expressions in Nominal Abstract Syntax) with dependent types so as to build up a logical framework on a basis more elementary than the lambda calculus.

---

Combining Interactive and Automatic Reasoning about Functional Programs

A. Bove, P. Dybjer and A. Sicard-Ramírez
Department of Computer Science and Engineering, Chalmers University of Technology and Göteborg University, Sweden

We propose a new approach to the computer-assisted verification of functional programs. We work in first order theories of functional programs which are obtained by extending Aczel’s first order theory of combinatory formal arithmetic with positive inductive and coinductive predicates. Rather than building a special purpose system we implement our theories in Agda, a proof assistant for dependent type theory which can be used as a generic theorem prover. Agda provides support for interactive reasoning by encoding first order theories using the formulae-as-types principle. Further support is provided by off-the-shelf automatic

---

theorem provers for first order logic which can be called by a program which translates Agda representations of first order formulae into the TPTP language understood by the provers. We show some examples where we combine interactive and automatic reasoning, covering both proof by induction and coinduction.

Chequeo de tipos basado en normalización por evaluación para la teoría de tipos de Martin-Löf con subtipos

Daniel Fridlender
FAMAF, Universidad Nacional de Córdoba, Argentina
En colaboración con Miguel Pagano quien presentó este trabajo en TLCA 2013

Se presenta una teoría de tipos con subtipos: una jerarquía cumulativa de universos y la regla contravariante para productos dependientes. Se extiende a este cálculo la técnica de normalización por evaluación definida para teoría de tipos sin subtipos. Gracias a esta función de normalización, la relación de subtipado y el chequeo de tipos resultan decidibles. Segn nuestro conocimiento, es la primera vez que se aplica la técnica de normalización por evaluación en el contexto de subtipos.

Métodos Formales e Ingeniería Dirigida por Modelos

Daniel Calegari
InCo, Facultad de Ingeniería, Universidad de la República

La Ingeniería Dirigida por Modelos (MDE por sus siglas en inglés) es un paradigma de desarrollo de software que jerarquiza el modelado como principal actividad del desarrollo de un sistema. El paradigma propone la construcción de modelos de diferentes aspectos de un sistema y su transformación promoviendo la construcción (semi)automática del sistema. La factibilidad del enfoque depende de la disponibilidad de lenguajes y herramientas que permitan la definición de modelos y su transformación, así como del establecimiento de bases formales que sustenten la práctica. En esta presentación se mostrarán diversos trabajos relacionados a la aplicación de métodos formales en el contexto de MDE, particularmente relacionados con la verificación formal de transformaciones de modelos. Además, profundizaré en la definición de un entorno unificado que permite la verificación formal de modelos y transformaciones de modelos mediante el uso de métodos de verificación heterogéneos, producto de mi tesis de doctorado. Este entorno está basado en la Teoría de Instituciones, la cual proporciona una base sólida para la representación de los elementos de MDE (a través de instituciones) sin depender de ningún dominio lógico específico. También proporciona una forma de especificar traducciones (a través de comorfismos) que preservan la semántica entre estos elementos y otros dominios lógicos utilizados para la verificación formal.
Realizabilidad Clásica: una introducción
Etienne Miquey
Centro de Matemática, Facultad de Ciencias, Universidad de la República

Desde los años 60 se conoce la correspondencia entre pruebas (de la lógica intuicionista) y programas funcionales, por medio del λ-cálculo. En los años 90, Krivine introdujo una extensión del λ-cálculo para trabajar en lógica clásica, que ya tiene resultados prometedores. En esta charla presentaré las ideas y las definiciones básicas de aquel sistema, y explicaré que tipo de resultados se pueden conseguir así. A continuación, Mauricio Guillermo y Alexandre Miquel darán dos ejemplos más específicos de tales resultados.

Especificación de la Ley de Peirce en Realizabilidad Clásica
Mauricio Guillermo
Centro de Matemática, Facultad de Ciencias, Universidad de la República

En la Lógica Intuicionista (LJ), la Ley de Peirce \(((A \Rightarrow B) \Rightarrow A) \Rightarrow A\), es equivalente al tercero excluído \(A \lor \neg A\). El problema de definir una Realizabilidad Clásica se reduce entonces al problema de hallar un realizador para la Ley de Peirce.

Como habremos visto en la charla previa, \(cc\) realiza la Ley de Peirce y permite entonces obtener una semántica de realizabilidad para la Lógica Clásica (LK) –y más aún, para la aritmética clásica de segundo orden–, con su lema de adecuación.

Una vez hallada una instrucción que realiza Peirce, la siguiente es una pregunta natural:

¿Cuál es el comportamiento que debe tener un realizador cualquiera de la Ley de Peirce?

El problema de la especificación consiste en hallar el comportamiento algorítmico de todos los realizadores de una fórmula dada. Estudiaremos entonces el problema de la especificación para el caso concreto de la Ley de Peirce.

Classical Realizability and Forcing
Alexandre Miqué
Centro de Matemática, Facultad de Ciencias, Universidad de la República

In this talk, I will present some connections between Krivine’s theory of classical realizability and Cohen’s theory of forcing, that was introduced in the 60’s to prove the independence of the Continuum Hypothesis. Firstly, I will present a computational analysis of forcing through the proofs-as-programs correspondence, and show that this method consists to extend Krivine’s Abstract Machine with a protected mode that is reminiscent from the design of modern computer architectures. Secondly, I will explain why classical realizability is a generalization of the method of forcing, and present the perspectives in model theory.