

Parsifal:

Preuves Automatiques et Raisonnement
sur des SpécIFICATIONS Logiques

(Proof search and reasoning with logic specifications)

A project proposal to INRIA-Futurs

Location: LIX, École Polytechnique

presented by Dale Miller

CP meeting, 8 September 2005

Theme: Sym A - Symbolic systems: Reliability and safety of software

Personnel

Permanent Members:

Joëlle Despeyroux, CR, INRIA-Sophia

Dale Miller, DR, INRIA-Futurs (team leader)

Lutz Strassburger, CR, INRIA-Futurs (starting 01.12.05)

Doctorants: **Alexis Saurin**, ENS d'Ulm and

Axelle Ziegler, ENS d'Ulm (co-supervised with Palamidessi).

Training: **David Baelde** (MPRI/ENS) (spring/summer 2005).

Applying for grant for PhD studies with Miller.

Post Doc: **Tom Chothia**, CNRS post doc joint with Comète and

Kevin Watkins (PhD CMU with Pfenning), starting INRIA
postdoc 01.11.05.

Outline

- Short, non-technical overview (3)
- Background: relating logic and computing (8)
- Project specifics: short-term and long-term tasks (9)
- Publications, funding, collaborators, relationships. (4)

Slogan: Addressing program correctness in the global computing setting

Global computing focuses our attention on the following:

- *Diversity of hardware and software.* Logic is a universal language (no trademark). Useful for specifying types, interfaces, memory layouts, safety criterion, etc.
- *Trust and security are enforced at load/run time.* Showing that code matches its specification/safety requirements/etc.
- *Communication and mobility of code.* Needs to address concurrency, for example, the π -calculus, bisimulation, etc.
- *Rapid changes, new languages/protocols/APIs/...* Importance of executable specifications, prototyping, etc.

Of course, these topics have important applications outside of global computing.

Some application areas: Part I

Areas where a lot of *shallow theorems* must be done quickly.

Ex: Some third party code (such as a library, API, or applet) needs to be used securely. Can I determine any guarantees statically?

1. type inference, interface checking.
2. proof carrying code: proof checking on client side; proof generation on producer (compiler) side.
3. symbolic bisimulation, symbolic model checking: compare a possibly large space of communication behaviors.

Some application areas: Part II

Areas where it is important to have completely *formal proofs* of *simple properties* about large and complex objects, such as entire programming languages, protocols, specifications, etc.

Ex: We have statically determined that a program produces an object of type *wallet*. Is it the case that if evaluation terminates, the resulting value has type *wallet*?

1. Type preservation theorems (subject-reduction).
2. Determinacy, coverage checking.
3. Termination of simple loops and recursions.

These applications overlap the POPLmark challenge: more later...

Roles of Logic in the Specification of Computation

Computation-as-model: Computations are mathematical structures (nodes, transitions, states, etc). Logic is used externally to make statements *about* those structures. E.g. Hoare triples, modal logics.

Computation-as-deduction: Pieces of logic are used to model elements of computation directly.

Functional programming. Programs are proofs and computation is proof normalization (λ -conversion, cut-elimination).

Logic programming. Programs are theories and computation is the search for normal proofs. The dynamics of computation are encoded in the changes to sequents that occur during that search.

Operational Semantics: Example

Just an illustration: don't worry about details.

$$\frac{M \Downarrow (\lambda x.R) \quad N \Downarrow U \quad (R[U/x]) \Downarrow V}{(M N) \Downarrow V}$$

$$\frac{B \Downarrow \text{true} \quad M \Downarrow V}{(\text{if } B \text{ } M \text{ } N) \Downarrow V} \quad \frac{B \Downarrow \text{false} \quad N \Downarrow V}{(\text{if } B \text{ } M \text{ } N) \Downarrow V}$$

$$\frac{\Gamma \vdash M : \sigma' \rightarrow \sigma \quad \Gamma \vdash N : \sigma'}{\Gamma \vdash (M N) : \sigma} \quad \frac{\Gamma, x : \sigma' \vdash R : \sigma}{\Gamma \vdash (\lambda x.R) : \sigma' \rightarrow \sigma} \textit{ proviso...}$$

We wish to *animate* and *reason* directly with such specifications.

Operational Semantics

Operational semantics is a dominate form of specification.

- Standard ML provided a slim and (nearly) complete specification of an entire programming language in this style.
- See your average POPL paper.
- Logical core of Centaur (former INRIA project Croap).

When *denotational semantics* is given, it is usually a deep results about a language (eg, game semantics, D_∞).

... and Logic

OpSem looks a lot like logic, but is it?

OpSem often have many side-conditions, unformalized assumptions, etc. Lots of hacking, little communicating of high-level meaning.

Proof Search versus Logic Programming

- Proof theory is a nature place to model intensional aspects of computation: binders, binder mobility, and resource management.
- Proof search specifications can often be quickly implemented using unification, backtracking, resource-splitting, etc.
- Proof search has a focus on logically correct deduction: unsound features (Prolog has many!) have no role.
- Proof search design and theory focuses on the meaning of logical connectives and quantifiers: in contrast to constraint logic programming and concurrent logic programming language.

λ -tree syntax: an approach to HOAS

Higher-order abstract syntax (HOAS) uses meta-level binders to encode object-level binders.

In the functional programming (ML, Coq): binders are mapped into function spaces. Interesting and lively recent work. It is problematic (adequacy, induction, equality, ...).

In proof search: binders are mapped to λ -expressions modulo mild rules for equality: α , η and β (for simple types). Called the *λ -tree syntax* approach.

λ Prolog was the first and still only programming language that incorporates λ -tree syntax. Isabelle and Twelf also include it. Maybe alphaCaml, FreshOCaml?

The LINC meta-logic

Alwen Tiu's PhD thesis (2004) describes the LINC meta-logic.

LINC stands for “**lambda, induction, nabla, and co-induction**”. It is a sequent calculus presentation of a rich intuitionistic logic that includes *induction* and *co-induction*.

LINC includes the new quantifier ∇ for the generic judgments that arise from using λ -tree syntax in negative statements. This quantifier differs from the Pitts-Gabbay “new” quantifier.

Tiu proved cut-elimination for the full logic and demonstrated its strength via many examples.

LINC forms the core meta-logic for this proposal.

LINC also stands for “LINC is not CIC”.

Planned deployment for Parsifal

I. Applications: focus on operational semantics

π -calculus Idealize Algol
 Concurrent-ML ...
 ... λ -calculus

II. Object-logics: A small number of these

HC HH LL ...

III. Meta-logic: Just one of these

LINC

Typical deployment of Coq

I. Applications: Logical focuses on math problems

graphs (planar, colors)
data-flow analysis
prime number theorem
Cantor normal form
...

II. Libraries: Large number of these with many authors

III. Type system: Just one of these

CIC

Implementations of provers

An Interactive Prover. Building such a prover certainly seems useful for understanding the logic and its possible applications.

Special purpose, automatic provers. Intended to be part of a larger systems in which deduction is necessary. We call them *logic engines*.

- Identify application areas. For example, type checking, type inference, proof checking, model checking, bisimulation checking, deduction in narrow domains.
- Reuse components and approaches that have already been validated: code for explicit substitutions, unification, backtracking search, print/parsing, etc. Reuse helps ensure trustworthiness.
- New applications might require exploring new theories.

The POPLmark Challenge: Mechanized Metatheory for the Masses

Initiated by two groups at University of Pennsylvania (Pierce, et.al.) and University of Cambridge (Sewell, et.al.).

“How close are we to a world where programming language papers are routinely supported by machine-checked metatheory proofs, where full-scale language definitions are expressed in machine-processed mathematics, and where language implementations are directly tested against those definitions?”

“To clarify the current state of the art, and to motivate further research, we propose some concrete benchmarks for measuring progress.”

An interactive theorem prover meant to explore LINC’s abilities might be used to address this challenge.

Examples of some meta-theorems

Can one really formalize typical type preservation theorems? How hard are these theorems to prove automatically? McDowell and Miller did several in [ToCL 2002].

What about security protocols? These generally have simple operational semantics with significant amounts of abstraction (hiding): however, their formal properties are difficult to prove.

How does meta-theory here compare with meta-theory developed within, say, Coq? For example, the encoding of the π -calculus by Honsell, Miculan, and Scagnetto [TCS 2001]?

Proof carrying code (PCC)

How can a machine receiving a *binary executable* know it to be safe to run?

PCC extends the *infrastructure* so that client machines post their *safety criterion* and producer machines send both the executable and a *proof* that it satisfies the clients safety criterion.

PCC is one of the first non-theorem proving applications that actually needs proofs. Engineering proof objects for this kind of application is an interesting problem.

Obvious trade-offs: A proof with *all details* present is huge but easy to check; a proof with *fewer details* can be small but expensive to check (involving, for example, backtracking search).

Checking bisimulation

When can two concurrent, not-necessarily terminating, programs be considered equal?

Miller and Tiu have used LINC to specify the π -calculus (LICS'03, CONCUR'05). The specification was so clean that

- the fact that bisimulation is a congruence came for free [Ziegler's DEA internship, SOS'05].
- we learned something new: proving bisimulation in intuitionistic logic yields *open bisimulation*; proving it in classical logic yields *late-bisimulation*, and
- we had an immediate implementation of bisimulation for the finite π -calculus using a small logic engine incorporating SML code from Tiu/Nadathur/Miller.

Model checking

A long-term project is viewing *model checking* as proof search.

Possible benefits:

- The implementations of model checkers are quite advanced. Some of that technology should help to implement subsets of LINC. For example: *tabled deduction*.
- LINC's treatment of higher-orders and binders provides a way to extend model checking to software [CONCUR'05].
- Symmetries and abstractions (used to pruning search space) can be described using high-level logical statements. Seeing model checking as deduction should allow for rich incorporation of such abstractions into search.

Improving the meta-logic LINC

There are many possible enhancements to the logic's design and meta-theory.

Integrating better ∇ with induction and co-induction.

Cyclic proofs. One approach to doing fast deduction of shallow theorems using co-induction is to employ cyclic proofs.

Accounting for classical reasoning.

More avenues for understanding ∇ . A simple model theory would be useful. Gabbay and Cheney in LICS'04 have implemented ∇ using the new quantifier of Gabbay and Pitts.

Incorporation of rewriting. Much of the foundations of how rewriting, higher-order quantification, and unification interact has been developed.

Type theory counterparts to LINC

Sequent calculus provides a flexible arena for experimenting with logics and extensions. It addresses the notion of *provability* but not necessarily *proof*.

Type theory is less flexible for exploring extension to logic but provides a more systematic approach to identifying proofs and computations on them (via Curry-Howard).

Having identified finite failure, HOAS, and ∇ as important logical principles, how can we systematize proof structures for them?

Type theory counterparts to (parts of) LINC might follow from Despeyroux, Pfenning, Leleu, and Schürmann [MCSC'01, TCS'01], where type theories allowed recursion on λ -trees syntax using type modalities.

Recent publications and prototypes

Despeyroux: *Mathematical Structures in Computer Science, Theoretical Computer Science*

Miller: *Logic Colloquium* • *Theoretical Computer Science* • CSL04
• FCS03 (Foundations of Computer Security) • TPHOL03

Miller, Tiu: *ACM Trans. on Computational Logic*, • FGUC04
(Foundations of Global Ubiquitous Computing), • LICS'03

Miller, Saurin: MFPS05 (Mathematical Foundations of
Programming Semantics)

Saurin: LICS'05

Tiu: CONCUR05 • TYPES 03 • PhD thesis

Ziegler, Miller, Palamidessi: SOS05

Software prototypes: Tiu has built two prototypes: Level0/1
(SML) and BLinc (λ Prolog, Java).

Current Funded Activities

- **Slimmer**: “Equipes Associées” via INRIA. Current associates: Minnesota (with matching NSF funds). Planned associates: Udine, Edinburgh, McGill.
- **Vallauris**: France-Spain Integrated Action Program Picasso, with Universidad Politécnica de Madrid.
- **Rossignol**: ACI-Sécurité
- **GeoCal**: ACI-Nouvelles interfaces des mathématiques
- **Mobius**: IP6 FET Global Computing, coordinated by Gilles Barthe.

Collaborators

Collaborators who have or are applying for funded relations:

Gopalan Nadathur, Minnesota

Alwen Tiu, Loria

Brigitte Pientka, McGill

Alberto Momigliano, Edinburgh

Marino Miculan, Udine

Agata Ciabattoni, Vienna

Current and Pending Visitors to LIX

Claudia Faggian, Padova, Italy (spring & fall 2005)

Elaine Pimentel, Belo Horizonte, Brazil (spring & fall 2005)

Frank Pfenning, CMU, Pittsburgh (spring 2006).

Relationships with other INRIA projects

- Logical/Proval
- Comète
- SecSI
- Marelle (Lemme)
- Everest
- Calligramme
- Signes
- Contraintes
- Croap

Questions or Comments?

A word about linear logic

Intuitionistic logic was proposed as a *challenge* to the way mathematicians did business. Not so for *linear logic*.

For us, *linear logic* is another device for the direct specification of computation. To the list of formal calculi useful for encoding computation — the λ -calculus, the π -calculus, Petri nets, transition systems, automata, etc — add linear logic.

Linear logic represents a rich approach to the specification of and reasoning about computation. It does not lead us to consider doing a different form of mathematical reasoning.

While our object-logic might be linear (or π or λ or Petri), our formalized mathematical reasoning will be intuitionistic or classical.

What is our relationship with Twelf?

λProlog (UPenn, 86) implements hereditary Harrop formulas (hhf).

Elf (CMU, 89) implements the Edinburgh LF.

LF and hhf share the same logic but LF internalizes proofs via dependent-types. Linear logic extensions considered by both teams.

Parsifal: Uses two levels of logic. The meta-logic is a new and strong logic (LINC) for explicitly doing induction/co-induction. Object-logics can range over several (determined by application).

Twelf: No explicit meta-logic of LF specifications. In particular, no induction principles are formalized; instead SML-based tools do some aspects of induction (totality, determinacy, etc).

It seems that the bisimulation and model checking examples are not possible in Twelf.