Tattered type theories
(Type theories with holes)

Murdoch J. Gabbay
Joint work with Dominic Mulligan
Thanks to Conor McBride

November 24, 2010

Introduction

Nominal techniques were introduced by myself and Andrew Pitts in a 1999 LICS paper. The canonical journal reference is now "A New Approach to Abstract Syntax with Variable Binding", Formal Aspects of Computing, 13(3-5): 341-363 (2002).

An accessible introduction to nominal sets is Andrew Pitts's 2003 "Nominal Logic" paper in I&C (actually a first-order axiomatisation). Let me advertise my survey paper "Foundations of nominal techniques" to be published in the Bulletin of Symbolic Logic, available on my webpage.

These papers concern reasoning on abstract syntax with binding, e.g. inductive proofs. See Urban's brilliant implementation of these ideas in Isabelle, intended to make it much easier to quickly set up an inductive definition and reason on it, in Isabelle/HOL.

Languages for formal specification

But that's not what my talk will be about. Names and binding turn up in meta-mathematical frameworks.

Universal algebra, first-order logic, higher-order logic, and (dependent) type theories are examples of meta-mathematical frameworks (a formal or implemented logic for conducting theoretical computer science).

Implementations include $\mu CRL2$, Isabelle/FOL, HOL, COQ, and Matita.

Universal algebra and first-order logic do not have binding in terms. The others do, via λ-abstraction.

Languages for formal specification

In a series of papers I and my co-authors have constructed logics for meta-reasoning based on nominal sets semantics. These are first-order, but terms can bind names in their arguments. Instead of λ-abstraction, 'nominal' languages have α-abstraction, which is weaker.

These include:

- Nominal rewriting.
- Nominal algebra.
- Permissive nominal logic (PNL).

(I mention also αProlog. Cheney and Urban wrote that they considered it as strictly limited to syntax-with-binding, i.e. not as a general meta-mathematical framework. I wonder if it could be applied as such.)

The goal of this talk: tattered type theories

I want to tell you about ongoing research into adding 'holes' to type theories; loosely speaking, by this I mean meta-mathematical frameworks like HOL or dependent types.

This talk will contain few technical details.

Imagine you are designing a theorem-prover.

- You fix a logic. This is the meta-mathematical language of your tool.
- You fix a theory of derivations. These derivations are the proof-objects that your tool must construct.

But that was the easy part.

The hard part

You still need to deal with incomplete derivations, and tactics.

You need to manipulate derivations with holes, or as I like to call them, level 2 variables (also called meta-variables though I deprecate this).

This is a core interest of the system; it is a central design issue.

Let's fast-forward to Matita. This presents completed derivations as terms, but internally includes holes plus explicit substitutions and β-equivalence so it can decide equality of partially determined derivations. I don't believe this 'tattered' theory is entirely formal.

When last I heard, COQ has analogous constructs and did not have type safe instantiation of holes, so the type checker must be rerun after instantiation. Since type checking requires β-reduction, even if it is decidable this can be unboundedly expensive.

The hard part

Nominal rewriting, nominal algebra, and permissive-nominal logic have holes built in to them. These are nominal unknowns.

In principle, all we need to do is to add a type system. The unknowns represent 'unknown parts' of a derivation; the type system determines the propositions that derivations populate.

The outlines of this idea are developed with Mulligan in Curry-Howard for incomplete first-order logic derivations (I&C 2010), where we used nominal terms to give semantics to incomplete derivations in first-order logic.

This paper is certainly not 'the answer', but it's a step in a certain direction.

Let's take a look at some incomplete derivations in first-order logic.

Three simple examples

Here are three simple derivations (the third is complete):

$$(\mathbf{1})\ \begin{array}{c}\vdots X\\ \bot\Rightarrow A\end{array} \qquad (\mathbf{2})\ \dfrac{\begin{array}{c}[\bot]\\ \vdots X'\\ A\end{array}}{\bot\Rightarrow A}\,(\Rightarrow\mathbf{I}) \qquad (\mathbf{3})\ \dfrac{\dfrac{[\bot]}{A}\,(\bot\mathbf{E})}{\bot\Rightarrow A}\,(\Rightarrow\mathbf{I})$$

Here are suggested representations using typed nominal terms:

$(\mathbf{1})\ X : \bot\Rightarrow A$

$(\mathbf{2})\ \lambda a{:}\bot.X' : \bot\Rightarrow A$

$(\mathbf{3})\ \lambda a{:}\bot.\mathrm{xf(a)} : \bot\Rightarrow \mathrm{A}$

A fourth (harder)

$$\dfrac{\begin{array}{c}\vdots X\\ \forall c.\mathrm{P(c)}\end{array} \qquad (\forall c.\mathrm{P(c)}) \Rightarrow \mathrm{B}}{\mathrm{B}}\,(\Rightarrow\mathbf{E})$$

is represented as

$$\dfrac{\dfrac{\dfrac{}{X : \mathrm{P(c)}}\,(\mathbf{TX})}{\lambda c.X : \forall c.\mathrm{P(c)}}\,(\mathbf{T}\forall\mathbf{I}) \qquad \dfrac{}{q : (\forall c.\mathrm{P(c)})\Rightarrow \mathrm{B}}\,(\mathbf{Ta}\phi)}{q(\lambda c.X) : \mathrm{B}}\,(\mathbf{T}\Rightarrow\mathbf{E})$$

The next step

Merely by permitting λ-abstraction over nominal unknowns, we should obtain a programming language that can input an (incomplete) derivation and operate on it.

We have a name for this: tactics. So:

- a is an assumption. $\lambda a.\mathrm{xf(a)}$ is a complete derivation.
- X is a hole. $\lambda a.X$ is an incomplete derivation.
- $\lambda X.\lambda a.X$ is a tactic that takes an incomplete derivation and λ-abstracts over one of its assumptions.
- $\lambda X.\lambda a.\mathrm{if}\ (a{=}X)\ \mathrm{then}\ b\ \mathrm{else}\ c$ is full meta-programming, i.e. we deconstruct and build derivations (hopefully with some guarantee of type safety).

The next step

So here in a nutshell is our idea:

- A two-level λ-calculus where level 1 represents complete derivations, the 'one-and-a-halfth level' (with unknowns but without level 2 abstraction) represents incomplete derivations, and level 2 represents tactics.
- An accompanying type system.

The object-level derivation system and meta-level derivation-construction system become tightly integrated; they are two sides of a single coin.

(Proof-construction and tactics become meta-programming.)

Required properties

1. Confluence
2. Strong normalization
3. Subject reduction
4. Soundness and weak completeness / conservativity with respect to the original type theory
5. Type safety of level 2 (metavariable) instantiation

Conclusions

Nominal techniques offer a mechanism to treat computation, incompleteness, and binding as interacting but distinct components.

They have a well-understood and elementary semantics in nominal sets.

We can write a tactic using a level 2 language, handle incompleteness using variables, and reason about binding using level 1 abstraction. When in doubt, the nominal sets semantics tells us what this all means.

Concluding speculations

Tight integration of the entire system will—we speculate—mean that users need only learn one language. This will have excellent and certifiable mathematical properties; incompleteness will be 'level two' — rather than second class.

Not only derivations but also the means by which derivations are constructed can be certified correct by types within the system itself. A tactic would be just another term, and its type would describe its behaviour.

There is a market for solutions to these kinds of problems.

Thank you.