Book Review

Type Theory and Functional Programming, Simon Thompson (Addison-Wesley, 1991)

Constructive Foundations for Functional Languages, Raymond Turner (McGraw-Hill, 1991)

Constructive mathematics is founded in a critique of classical methods, in particular the conception of infinite sets. In the constructive view, infinite sets cannot be treated as completed wholes, but as collections whose elements can be constructed as necessary. Similarly we do not have a set of all truths, but may prove individual propositions and thereby recognize them as true.

The consequences are well-known. The constructivist does not say "ϕ is true" but rather, "p is a witness of ϕ." A witness of $\phi \vee \psi$ contains a witness of ϕ or a witness of ψ, with an indication of which. A witness of $(\exists x \in A)\phi(x)$ is a pair $\langle a, p\rangle$, where $a \in A$ and p is a witness of $\phi(a)$. Unless we know which of ϕ and $\neg\phi$ is true, $\phi \vee \neg\phi$ need not hold. Unless we know which $a \in A$ makes $\phi(a)$ false, $\neg(\forall x \in A)\phi(x)$ need not imply $(\exists x \in A)\neg\phi(x)$.

Bishop's work on analysis has elaborated the constructive approach. Equality becomes a typed relation: the meaning of $a = b$ is not absolute, but depends upon the type of a and b. Each claimed property must be justified by a witness. The subset $\{x \in A \mid \phi(x)\}$ is regarded as consisting of pairs $\langle a, p\rangle$, where $a \in A$ and p is a witness of $\phi(a)$. The logarithm function requires two arguments, a real number x and a witness of $x > 0$.

Martin-Löf's Constructive Type Theory is an attempt to formalize Bishop's principles. By identifying propositions with the types of their witnesses, it can express much with a few primitives. From the second paragraph above, if ϕ and ψ are types, then $\phi\vee\psi$ behaves like their disjoint sum type. Furthermore, $(\exists x \in A)\phi(x)$ and $\{x \in A \mid \phi(x)\}$ are essentially the same: they consist of pairs, where the type of the second component may depend upon the value of the first. This is the general sum type $(\Sigma x \in A)\phi(x)$. As a special case it includes the binary product $A \times B$, which represents conjunctions. The general product type $(\Pi x \in A)\phi(x)$ represents $(\forall x \in A)\phi(x)$; as a special case it includes the function type $A \to B$, which represents implications.

Thus we have all the logical connectives and a rich suite of data structures. The theory also contains the elements of a functional programming language. Because its types can express propositions, they can express complete specifications; they include, for instance, the type of sorting functions. Proving a theorem can synthesize a functional program satisfying a specification.

It is hardly surprising that Martin-Löf's Type Theory has attracted so much attention. And yet, progress has been slow. The rules are difficult to use, and there are many of them. Many research groups have suggested modifying the theory, sometimes at variance to its spirit. Even Martin-Löf has found problems in his theories and has modified them.

Type Theory and Functional Programming, by Simon Thompson, is an excellent introduction to this work. It covers background material, then explores the theory's properties, applications, variations and foundations. Precise and formal, the book is yet readable, covering just the right topics. I have found it a pleasure.

Chapter 1 introduces formal logic, with a particularly nice description of quantifiers. Chapter 2 introduces the λ-calculus, with a clear presentation of Tait's strong normalization proof. Chapter 3 briefly discusses constructive mathematics. Chapter 4 is a methodical introduction to Martin-Löf's Type Theory. Chapter 5 comments and enlarges on the previous chapter, generalizing some rules and justifying others. Chapter 6 presents examples, including quicksort, vectors, program development, and program transformation. It contains a good discussion of abstract types and type classes. Chapter 7 discusses extensions such as quotient types, inductive types and well-founded recursion. Looking a bit hurried, Chapter 8 discusses foundations in model theory and proof theory, while Chapter 9 covers related work. Next comes a complete listing of the rules. The Index is minimal; it lists no authors.

Ignoring quibbles about notation and terminology, and some trivial errors, I have only a few complaints. Like many authors, Thompson takes a heterodox view of the theory. The orthodox view may be found in *Programming in Martin-Löf's Type Theory* by Nordström et al., written in close collaboration with Martin-Löf. Thompson's independence makes his book all the more informative, but perhaps he strays too far. Here are two instances. He replaces the typed equality judgement $a = b \in A$ by $a \leftrightarrow^* b$, an extralogical, untyped notion of convertibility. He uses the internal notion of function for the purpose of variable binding, which should be external to the theory.

The book's virtues outweigh its drawbacks. It is self-contained; it has lots of examples; it includes an excellent literature survey. I most enjoyed its comprehensive demolition of the case for subset types: they are at odds with the theory's spirit; they do not work well; they are not needed anyway. If you are only getting one book on Martin-Löf's Type Theory, let it be Thompson. For serious work, get Nordström et al. too; it is shorter and rather dry, but it meticulously presents the theory's main variants.

Although Martin-Löf's work is the most widely known, there exist other constructive type theories. *Constructive Foundations for Functional Languages*, by Raymond Turner, presents a theory based largely on the work of Feferman. It is generally well organized and well written, but technically heavy. Most of the proofs are unreadable, consisting of a monolithic paragraph covering the better part of a page, unrelieved even by formula displays. Thompson is immensely clearer and presents better examples, while Turner presents fascinating, important and hitherto inaccessible material.

The first part of the book covers background topics. Chapter 1 is a thoughtful introduction to functional languages and constructive theories. Chapter 2 introduces intuitionistic logic. Chapter 3 is a brief overview of the λ-calculus. Chapter 4 goes into more depth: the lazy λ-calculus, applicative bisimulation and free logics. Chapter 5 presents a simple functional language.

The next part develops the Feferman-style constructive type theory. Chapter 6 defines sum, product and function types. Chapter 7 covers comprehension and general sum and product types. Chapters 8 and 9 cover inductively defined types. Chapter 10 introduces polymorphism and abstract types.

The last part of the book is concerned with program extraction. Chapters 11 and 12 present several forms of realizability, for extracting programs from proofs. Chapter 13 demonstrates program extraction by deriving three sorting algorithms from a single specification of sorting. Chapter 14 demonstrates a novel approach to program transformation, putting it on a sound footing. Finally, Chapter 15 offers brief conclusions.

Through tremendous efforts, Raymond Turner derives a theory that looks remarkably similar to Martin-Löf's. He never attempts a comparison, but his theory does seem to have some advantages. It does not identify propositions with types, but takes intuitionistic logic as primitive. It supports a notion of absolute comprehension, yielding Martin-Löf's types and many others. Particularly impressive is its treatment of inductive definitions.

Both theories can be used to extract functional program from proofs of specifications, given in the form

$$\forall x\, [\phi(x) \to \exists y\, \psi(x, y)].$$

The extraction works by identifying propositions with types, or by realizability. Unfortunately, the resulting function may require two arguments: x and a witness for $\phi(x)$. It may deliver a pair of results: y and a witness for $\psi(x, y)$. Many authors have wanted to get rid of the witnesses. Turner's approach is complex: negative realizability. Thompson's approach is simply to change the specification to

$$\exists f\, [\forall x\, [\phi(x) \to \psi(x, f(x))]].$$

A witness for this proof consists of a pair whose first component is the function f and whose second component is a witnessing function relating ϕ and ψ, which may be thrown away.

I am sceptical about extracting programs from constructive proofs. As the examples make clear, finding the proof requires detailed knowledge of the desired program. And if we discard the witnesses, we might as well work in classical logic!

Both Thompson and Turner ought to discuss the constructive philosophy in greater depth. Formal rules are impossible to understand without motivation. I find Turner particularly hard to follow. His constructions hinge on principles that are never stated explicitly. He seems to identify types with collections of terms, and regards a term is meaningful if it reduces to a normal form. This form of constructive mathematics has more the flavour of Curry (extreme formalism) than intuitionism.

Constructive mathematics has much to offer in formal methods. Serious students and professionals will find both books rewarding. However, the field is treacherous and turbulent. Readers will encounter more questions than answers.

Lawrence C. Paulson
Computer Laboratory
University of Cambridge