

Building new constructive logical systems^{*}

John N. Crossley

Faculty of Information Technology,
Monash University, Clayton, Victoria, Australia 3800
`John.Crossley@infotech.monash.edu.au`

Abstract. Curry and Howard observed that ordinary propositional logic can also be viewed as a functional (programming) language. Thus programs are contained, in a certain sense, in proofs in mathematical logic. This has led to the viewing of proofs (originally, just in formal logic) as computer programs. The advantage of these techniques is that the task of programming a function is reduced to reasoning with domain knowledge.

In this paper we look at extensions of the Curry-Howard correspondence into an application: algebraic specifications using the language *Casl*. The full details may be found in our forthcoming book with Iman Poernomo and Martin Wirsing.

We also take a more abstract view of the process in which we are engaged and how one can incorporate other ideas, such as those of context and state, into the logic.

1 Introduction

What is logic? I believe I am following the ancient Greek philosopher Aristotle when I say that logic is the (correct) rearranging of facts to find the information that we want.

Here I shall restrict my attention to constructive or intuitionist logic.

In this short paper I wish to give an idea of how one can incorporate other ideas, such as those of context and state, into the logic. I also want to show how one can keep a record of what is going on by using Curry-Howard terms (defined below).

If we can prove $\forall x \exists y A(x, y)$ then we can actually get a program such that, given an x , it will compute a corresponding y . This is the idea of *program extraction*. (See [7], [5] or [8] for further details.) Moreover, we have a proof of $A(x, y)$ for this x and y so the program is “correct” in the sense that it meets its specification.¹

^{*} This paper was presented at the CLASE Workshop at ETAPS, 9 April 2005.

¹ Intuitively speaking, the specification can be regarded as a statement about the result of the program.

2 Structured specifications

In writing large specifications it is convenient to design specifications in a structural and modular fashion by combining and modifying smaller specifications. This helps us to master the complexity arising from a large number of function symbols and axioms.

Casl: The Common Algebraic Specification Language provides a standard for writing specifications. See [3] and [1] as well as the web site [2].²

We employ three specification-building operations from *Casl* [2]. A basic specification is of the form $\langle \Sigma, Ax \rangle$, where Σ is a signature consisting of a set of sorts, a set F of $(S^* \rightarrow S)$ -sorted function symbols and a set P of $(S^* \times S)$ -sorted predicate symbols. Ax is a set of Σ -formulae. Each such formula is a Harrop formula³ sometimes of the form $f(x_1, \dots, x_n) = e$ where e is a λ -expression. The specification-building operations for constructing specifications from basic ones are: *translation*, *union* and *hiding*. In *Casl* *translation* is written **SP with** ρ , where ρ is a symbol mapping, *union* is written **SP_1 & SP_2** and *hiding* is written **SP hide** Σ , where Σ is a symbol list. Note that many of the other common specification operators (extension, revealing, and local specifications) used in *Casl* can be constructed from these three operators.

Examples of specification built using *Casl* may be found in Fig. 1.

3 Building new rules

The most familiar logical rule is perhaps

$$\frac{A \quad (A \rightarrow B)}{B}$$

otherwise known as *modus ponens* or *detachment*: the A is ‘detached’ from the formula $(A \rightarrow B)$ leaving B . This is just one example of a logical rule.

When we wish to extract programs from proofs we use *Curry-Howard terms*: the Curry-Howard term carries *all* the information as to how we have constructed the proof so far.⁴

If a is the Curry-Howard term for the proof of A and d is the Curry-Howard term for the proof of $(A \rightarrow B)$, then the rule of *modus ponens* becomes:

$$\frac{a : A \quad d : (A \rightarrow B)}{(da) : B} \quad (1)$$

When we wish to extract programs from proofs from algebraic specifications the Curry-Howard terms that we use are now more complicated for two reasons.

² Not to be confused with the PDA language CASL.

³ A formula is a *Harrop formula* if it is 1. an atomic formula (including \perp), 2. of the form $(A \wedge B)$ where A, B are Harrop, 3. of the form $(Z \rightarrow A)$ where A (but not necessarily Z) is Harrop or 4. of the form $\forall x : s \bullet A$ where A is Harrop. Algebraic specifications will very often have only Harrop formulae for their axioms.

⁴ We do not give all the details here. They may be found in e.g. [5] or [8].

```

spec NAT_0 =
sorts
  Nat
ops 0 : Nat; s : Nat → Nat; + : Nat × Nat → Nat
preds
  ≥ : Nat × Nat
axioms
  Nat_0.1 : ∀x : Nat • x + 0 = x Nat_0.2 : ∀x : Nat; ∀y : Nat • x + s(y) = s(x + y)
  Nat_0.3 : ∀x : Nat • x ≥ 0 Nat_0.4 : ∀x : Nat; ∀y : Nat • x + y = y + x
  Nat_0.5 : ∀x : Nat • s(x) ≥ x
  Nat_0.6 : ∀x : Nat; ∀y : Nat; ∀v : Nat; ∀w : Nat • x ≥ v ∧ y ≥ w → x + y ≥ v + w
end

spec NAT_A =      spec NAT_B =      spec NAT_C =
  NAT_0 then      NAT_0 then      NAT_0 then
ops a : Nat      ops b : Nat      ops c : Nat
axioms          axioms          axioms
  A : a ≥ s(s(0))  B : b ≥ s(0)      C : c ≥ s(s(s(0)))
end              end              end

```

Fig. 1. Examples of specifications NAT_A, NAT_B and NAT_C. Note that 1. the axioms are Harrop formulae see footnote 3 and 2. we shall frequently write NAT_ALL for NAT_A & NAT_B & NAT_C.

In addition to the information from, for example, the logical rule being used, the Curry-Howard term also has to “remember” the specification. We have a similar situation for the structural rules. However, the message is as before: the Curry-Howard term carries *all* the information as to how we have constructed the proof so far.⁵

Thus the full rule for *modus ponens* we have is

$$\frac{\Gamma_1 \vdash a.SP : A \quad \Gamma_2 \vdash d.SP : (A \rightarrow B)}{\Gamma_1 \cup \Gamma_2 \vdash (da).SP : B} (\rightarrow \text{E})$$

Let me explain what is going on in this rule for implication elimination (\rightarrow -E). We are working within a single specification SP. We have Curry-Howard terms d for the proof of $(A \rightarrow B)$ and a for the proof of A . Therefore we have, just as in (1), (da) as the Curry-Howard term for the resulting proof of B . As usual the contexts Γ_1 and Γ_2 are added together. The specification SP is unchanged throughout.

The full set of rules is included in Figure 2.

⁵ Note that these Curry-Howard terms are quite different from the earlier ones although we use the same letters A and d to stress the generic similarity.

Logical Rules

Initial Rules

$$\frac{\{x : A\} \vdash_{\langle \text{sig}(A), \emptyset \rangle} \text{ass}(A, x) : A}{\emptyset \vdash_{\langle \Sigma, Ax \rangle} \text{ax}(\langle \Sigma, Ax \rangle, x) : A} \text{ (Ass I)} \quad \text{(Ax I)}$$

for $\{x : A\} \in Ax$

Introduction Rules

$$\frac{\Gamma \bar{\cup} \{x : A\} \vdash_{\text{SP}} d : B}{\Gamma \vdash_{\text{SP}} \lambda x : A. d : (A \rightarrow B)} (\rightarrow \text{I}) \quad \frac{\Gamma_1 \vdash_{\text{SP}_1} d : A \quad \Gamma_2 \vdash_{\text{SP}_2} e : B}{\Gamma_1 \cup \Gamma_2 \vdash_{\text{SP}_1 \ \& \ \text{SP}_2} \langle d, e \rangle : (A \wedge B)} (\wedge \text{I})$$

$$\frac{\Gamma \vdash_{\text{SP}} d : A}{\Gamma \vdash_{\text{SP}} \langle \pi_1, d \rangle : (A \vee B)} (\vee_1 \text{I}) \quad \frac{\Gamma \vdash_{\text{SP}} e : B}{\Gamma \vdash_{\text{SP}} \langle \pi_2, e \rangle : (A \vee B)} (\vee_2 \text{I})$$

$$\frac{\Gamma \vdash_{\text{SP}} d : A}{\Gamma \vdash_{\text{SP}} \lambda x : s. d : \forall x : s \bullet A} (\forall \text{I}) \quad \frac{\Gamma \vdash_{\text{SP}} d : A[t/x]}{\Gamma \vdash_{\text{SP}} (t, d) : \exists x : s \bullet A} (\exists \text{I})$$

Elimination Rules

$$\frac{\Gamma_1 \vdash_{\text{SP}_1} d : (A \rightarrow B) \quad \Gamma_2 \vdash_{\text{SP}_2} r : A}{\Gamma_1 \cup \Gamma_2 \vdash_{\text{SP}_1 \ \& \ \text{SP}_2} (dr) : B} (\rightarrow \text{E}) \quad \frac{\Gamma \vdash_{\text{SP}} d : (A_1 \wedge A_2)}{\Gamma \vdash_{\text{SP}} \pi_i(d) : A_i} (\wedge \text{E})$$

$$\frac{\Gamma \vdash_{\text{SP}} d : \forall x : s \bullet A}{\Gamma \vdash_{\text{SP}} dt : A[t/x]} (\forall \text{E}) \quad \frac{\Gamma \vdash_{\text{SP}} d : \perp}{\Gamma \vdash_{\text{SP}} dA : A} (\perp \text{E})$$

provided A is Harrop

$$\frac{\Gamma_1 \bar{\cup} \{x : A\} \vdash_{\text{SP}_1} d : C \quad \Gamma_2 \bar{\cup} \{y : B\} \vdash_{\text{SP}_2} e : C \quad \Gamma \vdash_{\text{SP}} f : (A \vee B)}{\Gamma^* \vdash_{\text{SP}^*} \text{case}(x : A. d : C, y : B. e : C, f : (A \vee B)) : C} (\vee \text{E})$$

where $\Gamma^* = (\Gamma_1 \cup \Gamma_2 \cup \Gamma) \bar{\cup} \{x : A\} \bar{\cup} \{y : B\}$ and $\text{SP}^* = \text{SP}_1 \ \& \ \text{SP}_2 \ \& \ \text{SP}$

$$\frac{\Gamma_1 \vdash_{\text{SP}_1} d : \exists x : s \bullet A \quad \Gamma_2 \bar{\cup} \{y : A[z/x]\} \vdash_{\text{SP}_2} e : C}{(\Gamma_1 \cup \Gamma_2) \vdash_{\text{SP}_1 \ \& \ \text{SP}_2} \text{select}(z : s. y : A[z/x]. e : C, d : \exists x : s \bullet A) : C} (\exists \text{E})$$

Structural Rules

$$\frac{\Gamma \vdash_{\text{SP}} d : A}{\rho'(\Gamma) \vdash_{\text{SP}} \text{with } \rho \bullet d : \rho \bullet (A)} \text{ (trans)} \quad \frac{\Gamma \vdash_{\text{SP}} d : A}{\Gamma \vdash_{\text{SP}} \text{hide } \Sigma \bullet d \text{ hide } \Sigma : A} \text{ (hide)}$$

if Γ is a $\text{sig}(\text{SP} - \Sigma)$ -context
and A is a $\text{sig}(\text{SP} - \Sigma)$ -formula

$$\frac{\Gamma \vdash_{\text{SP}_1} d : A}{\Gamma \vdash_{\text{SP}_1 \ \& \ \text{SP}_2} \text{union}_1(d, \text{SP}_2) : A} \text{ (union}_1\text{)} \quad \frac{\Gamma \vdash_{\text{SP}_2} d : A}{\Gamma \vdash_{\text{SP}_1 \ \& \ \text{SP}_2} \text{union}_2(d, \text{SP}_1) : A} \text{ (union}_2\text{)}$$

Fig. 2. Logical and Structural Rules.

4 Conclusion

The techniques we have presented here are based on a variant of Gabbay's labelled deductive systems [6]. Our logical rules are of the form

$$\frac{\text{Logical context, State, Curry-Howard term } \vdash \text{ Formula}}{\text{New Logical context, New State, New Curry-Howard term } \vdash \text{ New Formula}}$$

although the actual order may vary. Further, each of the items on the lower line may depend on, that is to say, be functions of, any or all of those on the top line, and of course there may be two or more sequences on the top line.

The semantics of these rules will depend on the structures that we are using. Also the interpretation of the informal terms: Logical context, State, etc. will also vary.

What seems to be most important is that we have extended the notion of logic in two ways. First of all we now have programs or other constructions (for example, specifications) interacting with the standard logical connectives. Secondly, the context of the logic may change in the course of a proof. This certainly happens in the context of algebraic specifications. Thirdly, we are now discussing logics (plural) and we arrive at such logics by an analysis of a technical setting. This seems to me to be following Aristotle's approach of looking at the real world, or a small part of it, and then abstracting the logical principles that work in that arena.

References

1. Michel Bidoit and Peter D. Mosses. *CASL User Manual*. LNCS 2900 (IFIP Series). Springer, 2004. With chapters by T. Mossakowski, D. Sannella, and A. Tarlecki.
2. CoFI Language Design Task Group on Language Design. *Casl*, November 2004. Available at <http://www.cofi.info/CASL.html> (accessed 31.03.2005).
3. CoFI (The Common Framework Initiative). *CASL Reference Manual*. LNCS 2960 (IFIP Series). Springer, 2004.
4. John Newsome Crossley. What is the difference between proofs and programs? Presented at the First Indian Conference on Logic and its relationship with other disciplines, Mumbai, January 2005..
5. John Newsome Crossley and John Cedric Shepherdson. Extracting programs from proofs by an extension of the Curry-Howard process. In John Newsome Crossley, Jeffrey B. Remmel, Richard A. Shore, and Moss Eisenberg Sweedler, editors, *Logical Methods: In honor of Anil Nerode's Sixtieth Birthday*, pages 222–288. Birkhäuser, Boston, MA, 1993.
6. Dov Gabbay. *Labelled deductive systems*. Oxford University Press, 1996.
7. Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and types*. Cambridge University Press, Cambridge, 1989.
8. Iman Hafiz Poernomo, John Newsome Crossley, and Martin Wirsing. *Proofs-as-Programs: The Curry Howard Protocol*. Springer, New York, 2005 (forthcoming).