15-399 Supplementary Notes: Propositions and Types

Robert Harper

March 22, 2004

1 Introduction

In constructive logic a proposition may be judged to be true only if there is evidence for it. Evidence for the truth of a proposition is a construction, or program, whose form is determined by the form of the proposition itself. The introductory rules for a proposition specify the fundamental, or canonical, forms of evidence for it. The eliminatory rules specify how to reason from the assumption of evidence for a proposition. The introduction and elimination forms must be mutually inverse to one another, ensuring that no more or no less evidence may be extracted from a proposition than was put in as evidence of its truth.

Type theory is a comprehensive theory of constructions. A construction is a step-by-step procedure for accomplishing a task specified by its type. In short, a construction is a well-typed program, and a type is a description of the behavior of a program.

The Curry-Howard isomorphism specifies that a proposition is identified with the type of its proofs — every proposition *is* a type specifying a construction that counts as evidence for it. Rather than treat propositions as types, we may instead associate with each proposition the type of its proofs. This makes the Curry-Howard interpretation explicit, and makes clear that propositions correspond only to certain forms of types.

2 Brief Review of Type Theory

The fundamental judgements of type theory are these:

	Formation	Def'l Equality
Types	$\Gamma \vdash \tau\ \mathit{type}$	$\Gamma \vdash \sigma \equiv \tau\ \mathit{type}$
Terms	$\Gamma \vdash t \in \tau$	$\Gamma \vdash t \equiv u \in \tau$

What types are there? There is no final answer to this question, but we can describe a rich collection of types that we have encountered thus far in the

course. These may be broadly classified into groups, but this grouping should not be taken too seriously!

- Simple types:
  - **1**: the type with precisely one element.
  - **0**: the empty type, with no elements.
  - $\tau_1 \times \tau_2$: the Cartesian product of τ_1 and τ_2.
  - $\tau_1 + \tau_2$: the disjoint union of the types τ_1 and τ_2.
  - $\tau_1 \to \tau_2$: the type of functions from τ_1 to τ_2.
- Dependent types:
  - $\Pi x{\in}\tau_1.\ \tau_2$: the dependent function space, or generalized Cartesian product, of the type τ_1 with the τ_1-indexed family of types τ_2.
  - $\Sigma x{\in}\tau_1.\ \tau_2$: the dependent product, or generalized sum, of the type τ_1 and the τ_1-indexed family of types τ_2.
- Data types:
  - **nat**: natural numbers.
  - τ **list**: lists of elements of type τ.
- Logical types:
  - **prop**: the type of propositions.
  - **pfs**(P): the type of proofs of proposition P.

3 Propositions and Types

The formation rules for propositions are recast in the framework of type theory as introductory rules for the type **prop**.

- Propositional logic:
  - $\top$: the trivially true proposition.
  - $\bot$: the trivially false proposition.
  - $P \wedge Q$: conjunction of P and Q.
  - $P \vee Q$: disjunction of P and Q.
  - $P \supset Q$: implication from P to Q.
- Quantification:
  - $\forall x{\in}\tau.\ P$: universal quantification over the type τ.
  - $\exists x{\in}\tau.\ P$: exitential quantification over the type τ.

- Primitive predicates:
  - $t =_N u$: equality of natural numbers.
  - $t =_L u$: equality of lists.

The semantics of propositions may now be expressed as equations that define the type of proofs of each form of proposition:

- Propositional logic:
  - $\mathbf{pfs}(\top) \equiv \mathbf{1}$.
  - $\mathbf{pfs}(\bot) \equiv \mathbf{0}$.
  - $\mathbf{pfs}(P \wedge Q) \equiv \mathbf{pfs}(P) \times \mathbf{pfs}(Q)$.
  - $\mathbf{pfs}(P \vee Q) \equiv \mathbf{pfs}(P) + \mathbf{pfs}(Q)$.
  - $\mathbf{pfs}(P \supset Q) \equiv \mathbf{pfs}(P) \to \mathbf{pfs}(Q)$.
- Quantifiers:
  - $\mathbf{pfs}(\forall x{\in}\tau.\ P) \equiv \Pi x{\in}\tau.\, \mathbf{pfs}(P)$.
  - $\mathbf{pfs}(\exists x{\in}\tau.\ P) \equiv \Sigma x{\in}\tau.\, \mathbf{pfs}(P)$.
- Predicates:
  - $\mathbf{pfs}(a =_N b)$ is a primitive type.
  - $\mathbf{pfs}(k =_L l)$ is a primitive type.

These equations express directly the semantics of constructive logic in terms of the constructions of type theory! In other words, once we have type theory in hand, we automatically have constructive logic "for free" through these defining equations.