LIBERTAS MATHEMATICA , VOL. VI (1986)

SEMANTICS OF CONSTRUCTIVE TYPE THEORY

Adrian Rezus

1. INTRODUCTION

Since 1967, when Bishop's treatise on the Foundations of Constructive Analysis was published [Bishop 67], a great deal of work has been spent on the formalization of constructive mathematics . If built upon the Zermelo-Fraenkel set theory (with intuitionistic logic: H. Friedman and J. Myhill) or on first-order theories of effective operations and classes (S. Feferman and, more recently, M. Beeson) the proposed formal systems came, usually, with their own model-theoretic justification: a general concept of a model is, in principle, available for systems based on predicate calculus. It was not so for type-theoretic formalisms: these have appeared somewhat earlier (1971), contained a lot of involved and rather unfamiliar (syntactic) constructs and lacked a good model theory. The latter approach originates in work of Per Martin-Löf (see, e.g., [Martin-Löf 72,75,75a,80,82,84]) and, since 1971, has evolved in several slightly distinct variants. (For pioneering work in this field see, e. g., [Scott 70].)

The central concept occurring in Martin-Löf's theories is that of a constructive (data) type: a type is defined by prescribing what one has to do in order to construct a "canonical" object of that type. A type definition must thus contain a "law" concerning the "generation" of ("canonical") objects in the defined type and a "specification" of the equality relation for such objects. This is, roughly, Bishop's notion of a (constructive) set. "Canonical" objects in a given type are its "typical inhabitants", where the "inhabiting" relation "...has type..." or "...is of type..." ("...ἐστὶ...") is not to be confused with the ZF ε-relation "...is an element of..." (or with any other variant occuring in theories of classes). In Martin-Löf's formulations, an object is "non-canonical" (or better: "in non-canonical form") if it requires some further (outermost) evaluation. This way of

speaking hides some (deliberate) confusion of use and mention (Quine; the fact has been also noted by other authors, although in different terms; see, e.g., [Diller & Troelstra 84]). Anyway, the "forms of evaluation" are prescribed at the meta-level and can be retrieved from mere syntactic stipulations, so one may think of a harmless abus de langage. It however turns out that one has to speak about evaluation and term-reduction even at the level of syntax-description in order to make things very clear. This approach is rather strange, but has something to do with Martin-Löf's way of explaining the type theory in terms of a fundamental theory of meaning [Martin-Löf 75a,80,84]. (We are not concerned with the philosophical implications of this theory here).

Any object "inhabiting" a type has a value : this is obtained by evaluation. A "canonical" object has itself as a value, while "non-canonical" objects evaluate always to "canonical" ones: they have equal "canonical" objects as values. Actually, it is better to speak about expressions denoting ("standing for") objects in this context, although this won't probably make any difference from the point of view of Martin-Löf's doctrine of meaning.

The evaluation of expressions denoting objects is carried out by a meta-program; the latter cooperates closely with the lexical "parser" specified at the syntax-description level and is always of the outermost-innermost kind ("from without" or "lazy").

"Non-canonical" objects (expressions) are often called "programs": each form of evaluation is associated to a syntactically unique "program form". The evaluation (meta-)program recognizes the required "form of evaluation" reading it off from the "program form" it first encounters.

The originality (pace philosophical grounds) consists here of the fact that the evaluator acts on object-expressions (denoting "canonical" objects and "programs") as well as on data-expressions (standing for "data types") with the same resources. In fact, Martin-Löf's theories manipulate generalized type structures, where expressions for types may be parametrized and do often depend on other (data) types, "canonical" objects or even "programs". So, as regards evaluation, (data) types and "canonical" objects behave differently: like the expressions standing for "canonical inhabitants" the corresponding type expressions do not require outermost evaluation, but may contain parts that are subject to evaluation.

The stipulations concerning the evaluation of expressions for (data)

types, "canonical" objects and "programs" do already specify the operational semantics behind the type theory (a kind of linguistic theory of computation). Still there is no denotational content in such specifications.

A denotational approach to the semantics of Martin-Löf's type theories ("real" model theory thus) turns out to be particularly difficult, due to the presence of generalized (or "parametrized") types. For some subsystems one can cope with such matters reasonably, by translating the type distinctions into first-order (type-free) theories of combinators. This way of providing a model theory for (a fragment of) Martin-Löf's type theory has been first proposed by Peter Aczel [Aczel 77]. The theory concerned was essentially that of [Martin-Löf 75] and the details of Aczel's interpretation for a version of the type theory with convertibility qua equality have been worked out in [Smith 78]. However, Aczel's models do not feature the full type theory of [Martin-Löf 75], leaving outside considerations concerning Universes.

The presence of one or more Universes in Martin-Löf's type theories makes a denotational approach even harder. The distinction between a common type and a Universe matches, in a way, the category theoretic analogue ("small" versus "large" categories). "Small" types are collected, at a higher level, into a "large" type and a "small" type "inhabits" a "large" one somewhat in the same sense an object "inhabits" a ("small") type, being subject to analogous closure conditions as regards type-formation. In later formulations of the type theory there is even a hierarchy of Universes, each one "inhabiting" the next in the sequence. Such aspects of the type theory can not be suitably accounted for in a first-order type-free context, even if one works with appropriate extensions of (type-free) lambda-calculus.

One way to handle the new complications consists of translating the (operational) semantics given in [Martin-Löf 82,84] into a closely related logical theory. Such an interpretation has been proposed, apparently by Aczel, and worked out by J. Smith in [Smith 84]; it "may be viewed as a metamathematical version of the semantical explanation (present in [Martin-Löf 82,84], A.R.), formalized in the logical theory" [Smith 84]. The logical theory is then further interpreted into a Frege structure with reflection on proposition and true proposition and with a special object interpreting the class of natural numbers. Readers familiar with [Aczel 80] should know how to

construct a Frege structure from arbitrary models of type-free lambda-calculus. Accurately enough, one may also think of this interpretation as being an attempt to reduce the type theory to a (type-free) lambda-theory; in fact, it is an extended variant of Aczel's original interpretation [Aczel 77] of the theory without Universes. Alternatively, one could have probably used something similar to Scott's type-free approach to classes [Scott 75] in order to insure (some form of) consistency for the type theory with one Universe. However, Smith's new attempt to a denotational semantics for Martin-Löf's type theory is rather involved (it needs two intermediate stages to reach its objective, a logical theory and a Frege structure) : meanings can be hardly read off from the "source" models. On the other hand, it has the advantage of respecting (in some sense) the intending meaning of the original theory, which is "operational" in nature. (Smith's interpretation in [Smith 84] concerns only Martin-Löf's type theory with one Universe).

About at the same time as Smith (± 1980), Michael Beeson developed a semantics for the theory of extended effective operations (building on earlier work of Feferman, Kreisel, Troelstra and his own). A by-product of the enterprise was a kind of recursive realizability interpretation for Martin-Löf type theories. Beeson's models appear in [Beeson 82,85]. Unfortunately, they offer a less transparent access to meaning than Smith's approach. Generality is however a good point in [Beeson 82]: as a bonus, one can obtain (via results of [Aczel 78]; cf. also [Aczel 82], for extensions to choice principles) Beeson recursive models for the constructive set theories (= formalizations) of Friedman and Myhill.

Finally, category theorists seem to have been aware for a long time of the close connections subsisting between Martin-Löf's type theories and certain categories known to be sufficient for the development of (much of) the mathematics of toposes. The problem of a detailed characterization of this relationship has been raised (again) in the McGill Categorical Logic Seminar of J. Lambek (McGill University, Montréal), in 1981-1982, and eventually solved by Robert Seely, in [Seely 82] (see also [Seely 84], for details). Specifically, Seely has shown that the logic of so-called "locally cartesian closed categories" (that is, categories **C** whose "slice categories" **C**/A are cartesian closed) is given by a theory of types essentially equivalent to that of [Martin-Löf 75] (in fact, as simplified by J. Diller in [Diller 80]). It is

easily seen that the (type) theory concerned is a proper subsystem of Martin-Löf's theory with one Universe, the main purpose of Seely being "to characterize locally cartesian closed categories", rather than to supply a category theoretical semantics for the full system **CST** of [Martin-Löf 82]. (So far, we were unable to find evidence, in print, documenting rumors about a possible categorical interpretation of the full system.)

The present paper is an attempt to overcome the logical parochialization of constructive type theory by providing an alternative ("non-standard") interpretation for Martin-Löf's theory with one Universe: the approach is purely denotational and does not depend on the so-called "proposition-as-types" approach or on specific presuppositions taken from the constructivists' credo-library. As in the case of the Aczel-Smith interpretation, we translate the type-distinctions present in Martin-Löf's type theory into a type-free setting, specifically: into a model of the type-free lambda-calculus. The preferred model is the Graph Model ([Plotkin 72], [Scott 76]), although one could have worked as well with Scott D_∞-models.

The main idea behind the constructions following below can be traced back to [Scott 76] and consist of :

1) interpreting types as (doubly strict) closures operators in $P\omega$ and
2) reading the "inhabiting"-relation "... has type ... " as "... is a fixed point of...".

The success of the construction does also depend on finding a good interpretation for the (first) Universe (here it could be interpreted as a closure whose fixed points are closures, possibly only the doubly strict ones) and on simulating several general "closure conditions" in the language of Scott(-continuous) closures.

Unlike the Aczel-Smith approach, the present interpretation is not primarily concerned with the intended (operational) meaning of the original theory any more; in fact, we take the operational semantics as being a part of a special discipline: the pragmatics of type theory (somewhat in the sense of C.W. Morris, with special application to the theory and design of typed functional programming languages).

The motivations we had for this approach are rather straightforward; before attempting to attach some (intended) operational meaning to a theory one must be sure that it does not lead to (unintended) simple inconsistencies.

For a type theory, a consistency proof must be more informative than in the case of logic(s) and/or so called equational theories: equational (simple or Hilbert-Post) consistency does not automatically insure the consistency of typing, if (above all) the latter is too complex. We do not think that constructive type theory is self-justifying (as Martin-Löf would probably want); in particular, informal explanations of the type-theoretical primitives (of the kind supplied by Martin-Löf and others : [Diller 80], [Diller & Troelstra 84], [Schwichtenberg 83], [Beeson 82,85], etc.), while intuitively sound, do not guarantee as such the classical consistency of the resulting formalizations. Indeed, the intended type-theoretic concepts and their meta-theoretical properties are rather sensitive to the choice of some particular formalization. For instance, while there is a relative agreement on how to formulate rigorously the properties of a generalized product ("dependent" product of an indexed family) of types, one can produce several distinct formalizations of the corresponding generalized sum construct ("disjoint union of an indexed family of types"), with non-equivalent typings (under translation); this is apparently due to the fact that usual products (of types) can be given different ("degrees of") generalization(s), according to some intended purpose; cf., e.g., generalized sums in [Martin-Löf 82,84] vs their analogues in Zucker's version of AUTOMATH [Zucker 75] vs similar constructs ("dependent products", "generalized existential types") in programming languages (Russell [Demers & Donahue 79], or better, its "pure" version, Kernel Russell [Hook 84], Pebble [Burstall & Lampson 84] etc.; see also the suggestions of Plotkin and Mitchell in [Mitchell & Plotkin 84]).

Even if some "standard" formalism is presented (cf. [Martin-Löf 80,82,84], [Beeson,82,85]), "translating" the intended meaning in some plausible sense, the outcome is far of being an "interpreted theory" (the "semantics" is too operational): in particular, the intuitive explanations of the type-theoretical primitives do not furnish any argument against, say, the existence of a (closed) term (in the formalism) that could be assigned any type, by mere formal transformations (= using only the proposed rules of the system).

This paper shows (implicitly) that such formal manipulations are ultimately safe, for a class of (non-equivalent) formalizations of Martin-Löf's constructive type theory.

The approach taken here is thus classical in nature (as opposed to

constructivistic) : it is essentially meant to insure (on mathematical grounds) that a certain subsystem of the Constructive Type Theory is perfectly intelligible from a classical standpoint and actually independent of the philosophical and operational explanations (= "justification") which could be extracted from Martin-Löf's own theory of meaning.

It is, however, fair to say that the latter forms a coherent and quite original philosophy of mathematics, which, if correctly understood and accepted, makes the classical explanation dispensable, in a sense. Martin-Löf seems to derive the classical consistency of his system of rules from such explanations (he uses to call "semantical"), on purely philosophical grounds thus. Rigorously speaking, the problem does not even arise in the framework of his theory of meaning : "form" (syntactic construction) and "content" (operational semantics) can not and should not be separated while explaining the type theory; it is in this sense one should take the famous dictum that the formal rules of the system constitute the explanation of the type-theoretical primitives.

The only consistency problem which may be posed from within the standpoint of Martin-Löf's philosophy of mathematics is, apparently, the so-called constructivistic (or, if one prefers, intuitionistic) consistency problem, viz. "are all types inhabited ?". Moreover, one has to understand the exact meaning of such a question appropriately : the universal quantifier may only refer to "generable" or already generated collections; indeed, the (classically) natural and innocent phrase "arbitrary type" (or "arbitrary set", to use Martin-Löf's recent terminology) is meaningless (while, on the other hand, "arbitrary object of a given type" makes sense and is even basic for the explanation/understanding of the theory). This is the only kind of consistency ever discussed in some detail by Martin-Löf.

It turns out that certain formal extensions of (particular fragments of) the type theory can be shown to be both consistent in the classical sense (Hilbert-Post, as extended to the typing relation) and intuitionistically inconsistent. The best example is perhaps $\mathbf{CA}_{\tau}$, the pure Classical AUTOMATH system extended with the axiom

$$[\,] \quad |\!-\!- \quad U \;:\; U$$

(stating that the Universe of all small types is a small type), proved classically consistent in [Barendregt & Rezus 83]. This theory is exactly

Martin-Löf's system of [Martin-Löf 71], which allowed proving the so-called Girard paradox ([Girard 72], [Martin-Löf 72], [Beeson 85, Exercise XI.21.1]), whence intuitionistic inconsistency ("all types inhabited").

Another motivation for the present approach consists of the fact the (contemplated fragment of) Constructive Type Theory is viewed as an abstract counterpart of a class of functional programming languages. The class is implicitly "defined" by the system in a sense which can be made formally precise. The model-theoretical considerations following below show that the languages (systems of programming notations) thus "derived" admit of a "classical" denotational semantics à la Scott and Strachey. A more detailed discussion of these topics is planned for a future report.

Related work. A subsystem of Martin-Löf's type theory (essentially the Girard-Reynolds second-order typed lambda calculus, cf. [Girard 72], [Reynolds 74,84], [Fortune et al. 83]) has been interpreted in $P\omega$, along the pattern used here or nearly, in [McCracken 79] (independently, [Barendregt & Rezus 83] have also noticed the possibility of such an interpretation; see also [Bruce & Meyer 84]). This attempt has been revised in [McCracken 83], using finitary retracts in ω-algebraic cpo's ("finitary Scott domains", cf., e.g., [Scott 81,82,82a]) in place of closures in additive domains. It is important to mention that second-order typed lambda-calculus gives rise to an interesting class of typed (functional) programming languages (as, e.g. Plexp in [McCracken 79], or Sol in [Mitchell 84], see also [Reynolds 84]) which support parametric type-polymorphism in a sense closely related to Martin-Löf's type theory (although somewhat weaker: the second-order typed lambda-calculi can be interpreted in Martin-Löf's theories with one Universe, but not conversely).

More generally, (viz. while working with additive lattice-domains, cf. [Sanchis 77] for terminology), the same ideas have been fruitfully exploited in [Barendregt & Rezus 83], in order to obtain models for the Classical AUTOMATH system of N.G. de Bruijn (cf. [de Bruijn 80], [Rezus 83],) second-order typed lambda-calculus, pure LCF etc. From the present paper, it should be also clear how to interpret, along the same pattern, Zucker's AUT-Pi System ([Zucker 75]), as well as many related systems, as, e.g., the Gothenburg

versions of Martin-Löf's type theory, with List-types (discussed in [Martin-Löf 80,84], [Nordström 81], [Nordström & Petersson 83], [Nordström & Smith 84], [Petersson 82]), typed systems used for program verification and theorem-proving by R.L. Constable and his collaborators ([Constable 71,80,83,83a], [Constable & Zlatin 84]), or, even, existing typed functional programming languages, as, e.g., Kernel Russell, Pebble, HOPE, Standard ML, etc.

Working in a classical setting (no "propositions-as-types") we were unable to capture, model-theoretically, the behavior of Martin-Löf's "identity types". This is, in a way, as expected, because such types seem to be intrinsically tied to an operational reading of the type-theoretical primitives (agreeing with the "intended" constructive explanation of equality). Actually, these constructs do not have a firm status in various presentations of the theory and we think one can argue that their addition to the system described does not alter the consistency result established here. (In particular, the identity types are irrelevant if one takes Martin-Löf's type theory as a programming language, in which case one must adopt a different operational interpretation for equality.)

The type theory discussed here is thus, accurately, Martin-Löf's type theory with one Universe and without identity types (called $\mathbf{CST}_1$, for further reference). If extended with list-types, it gives $\mathbf{CST}_1$**-L** (our label!), a system which is somewhat more interesting as a programming language. "Tree-data-types" may be also added, adopting as such or suitably weakening Martin-Löf's rules for Well-orderings (intended to handle Brouwer ordinals).

Appropriate extensions of $\mathbf{CST}_1$ can be thus profitably customized in order to design a large class of typed (functional) programming languages (CONMATH-languages, for further reference; the label coming obviously from "Constructive Mathematics"). The proposal can be traced back to E. Bishop and arguments supporting the enterprise have been further advanced by P. Martin-Löf [Martin-Löf 80,82], the Gothenburg school [Nordström 81], [Petersson 82], Robert Cartwright [Cartwright 80], R.L. Constable [Constable 71,83], Michael Beeson (cf. [Beeson 85a] and the references given there to the Language of Data project), etc.

This paper will, in fact, present $\mathbf{CST}_1$ as a skeleton ("kernel language") for a programming language in the CONMATH family. The basic syntax has been,

however, kept abstract (as much as this has been feasible) in order to allow direct access to the proposed semantics.

Acknowledgements. This work has been partially supported first by the University of Nijmegen and currently by the Netherlands Organization for the Advancement of Pure Research (Z.W.O.), as part of the author's project "Constructive Type Theories and Functional Programming". Draft versions of material appearing here have been presented previously in lectures held at the University of Nijmegen (Department of Computer Science), in 1983, December, and during the Spring Semester 1985, opportunity for which I am grateful to my (sub-) department chairman Raymond T. Boute (Computer Architecture and Operating Systems). I am also indebted to Peter Aczel, for explaining to me some obscure points of Martin-Löf's theories and to N. G. de Bruijn and Henk Barendregt, for useful discussions on AUTOMATH and the Graph Model.

2. SCOTT CLOSURES IN THE GRAPH MODEL.

21. Basic facts.

Throughout in the sequel we assume that the reader has some familiarity with the Plotkin-Scott Graph Model $P\omega$ ([Plotkin 72], [Scott 76]) and list notation and concepts only as necessary to recover actual proofs or if contextually constrained so. A type-free language LAMBDA (slightly extending Scott's homonymous language) will be used qua language of the model (and, in fact, in a readable, "loose" way) in order to express properties of (Scott-continuous) closures in $P\omega$. In 22 through 24 we generalize results of Scott [Scott 76], in view of subsequent manipulations.

21.1. Algebraic lattices and closures.

This section mentions the minimal lattice-theoretic background and notations necessary to understand the main text. Details may be retrieved from

[Gierz et al. 80].

Scott continuity. If $D = (D, \leq_D)$ is a complete lattice, the corresponding lattice-theoretic operations will denoted as usual by: $\bigsqcup_D$ (finitary sup), $\bigsqcap_D$ (finitary inf), $\sup_D$ (supremum), $\inf_D$ (infimum), while $\bot_D = \sup_D\emptyset = \inf_D D$ and $\top_D = \sup_D D = \inf_D\emptyset$ stand for the bottom and the top of the lattice resp.

As any power set, $P\omega = 2^{\mathbb{N}}$ (where $\mathbb{N} = \{0, 1, 2, \ldots\}$) is a complete lattice under $\subseteq$ (set inclusion); in this case the lattice operations will be denoted by $\bigcup$, $\bigcap$, $\bigsqcup$, $\bigsqcap$, resp., while $\bot = \emptyset$ and $\top = \mathbb{N}$. Moreover, as a complete lattice, $P\omega$ is algebraic.

Roughly speaking, a complete lattice is algebraic if it is isomorphic to a "subalgebra" of some power set 2^X, i.e., a subfamily closed under arbitrary intersections and unions of towers (the isomorphism is a lattice-isomorphism). An element $e \in D$ of a complete lattice D is compact ("isolated from below", "finite") if, for any directed $X \subseteq D$, $e \leq_D \sup_D X$ implies that $e \leq_D d$, for some $d \in D$. (Here a subset $X \subseteq D$ is directed if every finite subset $Y \subseteq X$ has an upper bound in X.). For example, the bottom $\bot_D$ of a complete lattice D is always compact. Let K(D) be the set of compacts of a complete lattice D. Then D is said to be algebraic iff $\forall x \in D.\ x = \sup_D \{e \in K(D) : e \leq_D x\}$.

Algebraic lattices are omnipresent in logic, algebra, topology and theoretical computer science. For example, the power sets 2^X (X any set) are always algebraic lattices under set inclusion, where the compact elements are the finite subsets of X. Further, the lattice of ideals of a sup-semilattice with zero is algebraic, with principal ideals as compact elements. If X is a topological space then the lattice O_X of open sets in X is algebraic iff the space has a basis of quasicompact open sets (where "quasicompact" qualifies, à la Bourbaki, subsets with the Heine-Borel property). Finally, if it is a complete lattice, a Scott domain (cf. [Scott 81,82,82a]) is an algebraic lattice with countably many compacts (it is said to be "ω-algebraic" or "countably based") and a case in point is the lattice $P\omega$, where the compacts are the finite sets $e \subseteq \mathbb{N}$. What makes this lattice more interesting is the fact that $K(P\omega)$ is a r. e. set.

The more general concept of a continuous lattice is obtained by weakening ("relativizing") the concept of compactness (see [Scott 72], [Gierz et al. 80]

or [Mislove 82]). If D is complete lattice, we say that an element x is relatively compact in y (or that "x is way below y", notation $x \ll_D y$), if, for any directed set $X \subseteq D$, with $y \leq_D \sup_D X$, there is some $d \in X$, with $x \leq_D d$. So $e \in K(D)$ iff e is relatively compact in itself (i.e., $e \ll_D e$). A complete lattice D is then continuous if $\forall x \in D.\ x = \sup_D \{y \in D : y \ll_D x\}$. Algebraic lattices are always continuous since a compact of D is "way below" any element above it ("above" in the sense of the underlying order), the converse is not true, however: take any complete chain to illustrate this.

From among the various functions $f : D \to E$ on algebraic lattices we are interested in those preserving suprema of directed sets: they are usually said to be (Scott) continuous. The terminology has a topological flavor and is well motivated: there is a topology behind the scenes, the so-called Scott topology (this appears in its full generality in [Scott 72], but see [Gierz et al. 80], mainly II.1, for a self-contained exposition). In general, the definition of $\sigma(D)$, the Scott topology on D, can be given for any poset D and reasonably so for up-complete (= "Dedekind complete") posets. In the case of concern, one can exploit the fact that $P\omega$ is a rather special poset: $\sigma(P\omega)$, the Scott topology on $P\omega$ is then directly accessible in terms of $P\omega$-compacts (= finite subsets of $\mathbb{N}$), viz. a set $O \subseteq P\omega$ is Scott open iff it is a family of finite character, qua subfamily of $P\omega$. (Explicitly, $O \subseteq P\omega$ is Scott open iff, for all $x \in P\omega$,

$x \in O$ iff $e \in O$ and $e \subseteq x$, for some $e \in K(P\omega)$.)

One checks easily that the Scott topology $\sigma(P\omega)$ is T_0 and that the (Scott) topology on products $P\omega \times P\omega$ agrees with the product of Scott topologies (*reason*: $\sigma(P\omega)$ is a continuous lattice). Moreover, continuity in the order-theoretic sense (sup-preservation) coincides with topological continuity relative to the Scott topology.

This brings us back to the familiar concepts of approximation and approximating map in Scott domains [Scott 81,82,82a]; every element $x \in P\omega$ is " approximated" (in the sense of $\subseteq$) by the compacts "below" it, i.e., by the finite sets it contains: $x = \bigsqcup \{e \in K(P\omega) : e \subseteq x\}$. Hence the Scott continuous functions on $P\omega$ are exactly those maps $f : P\omega \to P\omega$ which preserve this "approximating behavior":

21.1.1. Fact. A map $f : P\omega \to P\omega$ is Scott continuous iff

$$\forall x \in P\omega.\ f(x) = \bigsqcup \{f(e) : (e \in K(P\omega)) \ \&\ (e \subseteq x)\}.$$

Proof. See, e.g., [Gierz et al. 80], II. 2.1. □

Since the $P\omega$-compacts are r. e., one can find a one-one enumeration

$$(e_n)_{n \in \mathbb{N}},$$

of this set allowing the following explicit characterization:

21.1.2. Fact. (Characterization Theorem for Scott Continuity). For any bijection $n \mapsto e_n$ ($: \mathbb{N} \to K(P\omega)$), a function $f : P\omega \to P\omega$ is Scott continuous iff, for all $m \in \mathbb{N}$ and all $x \in P\omega$, one has

$$e_m \subseteq f(x) \quad \Rightarrow \quad \exists n \in \mathbb{N}.\ (e_n \subseteq x) \ \&\ (e_m \subseteq f(e_n)).$$

Proof. Except for details of coding, this holds for algebraic lattices in general; cf. [Gierz et al. 80], II. 2.1. □

A particular bijection $n \mapsto e_n$ to work with be chosen later.

21.1.3. Proposition.

(1) Arbitrary maps $f : P\omega^n \to P\omega$ are Scott continuous iff they are so in each component separately.

(2) The finitary sup and inf operations are Scott continuous in $P\omega$.

Proof. (1) See [Scott 76] or [Gierz et al. 80], II.2.9.

(2) For suprema: cf. [Gierz et al. 80], II. 4.13. For infima: $P\omega$ is meet-continuous; see op. cit. O. 4.1, O. 4.1, II. 4.16 and top of page 134. □

Cartesian products and function spaces. For complete (continuous, algebraic) lattices D, D', ... the cartesian products $D \times D'$ consist of "tuples" (pairs, etc.) partially ordered "componentwise".

21.1.4. Fact. The finitary product operations make new complete (continuous, algebraic) lattices from old.

Proof. Well known. □

Alternatively, the products $D \times D'$, etc. may be viewed as topological

spaces, endowed with the corresponding Scott topology (and, in all cases of concern here, this is the product topology, since the contemplated lattices $\sigma(D)$, etc. of Scott-open sets are continuous).

Another standard construction: for D, D', ... as above we let $[D \to D']$ stand for the space of Scott continuous functions $f: D \to D'$; viewed as a lattice, it is taken always with the "pointwise" ordering.

21.1.5. Fact. For complete (continuous, algebraic) lattices D, D',
(1) $[D \to D']$ is again a complete (continuous, algebraic) lattice, while
(2) the corresponding "order-induced" topology is Scott.
Proof. (1) Easy. (2) See [Scott 72], [Gierz et al. 80], II. 2.4, II.2.8. □

We write FUN for $[P\omega \to P\omega]$, the continuous function space of $P\omega$. The notation introduced so far will help identifying the type of various functionals used later.

The fixed point functional. A trivial set-theoretic argument shows that Scott continuous self maps f on complete lattices D are monotone. By completeness, a well-known theorem of Tarski [Tarski 55] guarantees that such functions do always possess both least and greatest fixed points (relative to the underlying order), while the latter are resp. the bottom and the top of a well-defined lattice $FIX_D(f) = \{d \in D : d = f(d)\}$, which is complete in the induced order: this is the fixed point lattice of f (in D). Hereafter, we write FIX(f) for the fixed point set of $f: P\omega \to P\omega$.

A summary inspection of the corresponding proof in [Tarski 55] reveals the fact that the existence of the bottom $\bot_f$ resp. the top $\top_f$ of $FIX_D(f)$ with D complete, is not established constructively. In the context of Scott domains, the common way out from this inconvenient consists of contemplating the sub-lattices $FIX_D(f)$, for Scott continuous $f : D \to D$. The outcome is rewarding, at least inasfar least fixed points are concerned; viz., for $P\omega$:

21.1.6. Proposition. (Fixed Point Theorem). Every Scott continuous map $f \in FUN$ has a least fixed point. Moreover, there is a Scott continuous functional

$$\mathbf{fix}: [P\omega \to P\omega] \to P\omega,$$

such that, for every $f \in FUN$, **fix**(f) is the least fixed point of f, i.e. the bottom of the complete lattice $FIX(f)$.

Proof. This holds for Scott continuous self maps f on any up-complete poset D, with a bottom $\bot_D$ (that is: for any "cpo"). To see this, define:

$$\textbf{fix}(f) = \sup_D \{f^n(\bot_D) : n \in \mathbb{N}\}$$

where $f^0(d) = d$ and $f^{n+1}(d) = f(f^n(d))$, for all $d \in D$ and all $n \in \mathbb{N}$. □

Composition, evaluation and abstraction. A well-known fact about algebraic lattices (resp. Scott domains) is that the category **ALG** (resp. $\textbf{ALG}_\omega$) with algebraic (resp. ω-algebraic) lattices as objects and Scott continuous maps as arrows is cartesian closed. For present purposes it suffices to notice that function composition and the corresponding evaluation and abstraction functionals are easily available (and Scott continuous). Indeed with notation as earlier, define **ev**: $[P\omega \to P\omega] \times P\omega \to P\omega$ by:

$$\textbf{ev}(f, x) = f(x) \qquad (\forall f \in FUN, \forall x \in P\omega)$$

and if, for any $f \in [P\omega \times P\omega \to P\omega]$, a map $\hat{f}$: $P\omega \to FUN$ is defined by:

$$\hat{f}(x) = \lambda\, y \in P\omega.f(x,y),$$

then set

$$\textbf{abs} = \lambda\, f \in [P\omega \times P\omega \to P\omega].\hat{f}.$$

21.1.7. Proposition.

(1) The composition functional $\circ$ on FUN, $\circ$: FUN x FUN → FUN, given by:

$$f \circ g\,(x) = f(g(x)) \qquad \forall f, g \in FUN, \forall x \in P\omega,$$

is Scott continuous.

(2) The evaluation functional **ev** is Scott continuous and

(3) so is the abstraction functional **abs**.

Proof. Standard. □

Closures. If D is a complete lattice, a monotone self map $f : D \to D$ (relative to $\leq_D$) is said to be a projection (of D) if it is idempotent relative to function composition, i. e., $f = f \circ f$; somewhat stronger, f is a closure (of D) if f is a projection (of D) with, moreover, $id_D \leq f$ (where $\leq$ is "pointwise" and id_D is the identity on D). In other words, for closures f of D, one has $x \leq_D f(x)$, for all $x \in D$. In particular, Scott continuous projections will be called (Scott) retractions and Scott continuous closures will be also referred

to as Scott closures.

For complete D, a Scott retraction $f : D \to D$ "retracts continuously D onto f(D)"; its range $F(D) = \{f(x) : x \in D\}$ is then a (Scott) retract (of D). This way of speaking is frequent.

21.1.8. Fact. The fixed point set $FIX_D(f)$ of a projection f of D (D a complete lattice) coincides with its range: $FIX_D(f) = f(D)$.
Proof. Trivial. □

If D is a continuous lattice we know something more, viz.,

21.1.9. Fact. Where D is a continuous lattice and f is a (Scott-) retraction of D, $FIX_D(f)$ (= f (D)) is also a continuous lattice in the induced order.
Proof. [Gierz et al. 80], I.2.14. ◘

Where do closures come from? In general, if f is a self map on a complete lattice D, f is a closure of D iff

$$x \leqq_D f(y) \iff f(x) \leqq_D f(y), \qquad \forall x, y \in D.$$

Note that the closure maps of power sets are the usual "closure operators", while the Scott-closures of power sets are the so-called "algebraic closure operators". The corresponding fixed point sets of closure operators

$$f : 2^X \to 2^X \qquad (X \text{ any set})$$

are known as "closure systems" ("Moore systems"; see [Birkhoff 67]), and the Scott continuous closures have "algebraic closure systems" as fixed point sets. There is a well-known one-one correspondence between (Scott continuous) closure operators $f: 2^X \to 2^X$ and (algebraic) closure systems $\mathfrak{C} \subseteq 2^X$, where $\mathfrak{C}_X = f(2^X)$ and X is any set (see, e.g., [Scott 82a], [Erné 80].

In particular, an algebraic closure system $\mathfrak{C}_X \subseteq 2^X$ is a "subalgebra" of 2^X, i.e., a subfamily of 2^X closed under arbitrary intersections and unions of directed subfamilies $\subseteq 2^X$; alternatively, one can replace here the directed subfamilies by towers (= chains of sets) in 2^X.

The analogue of the preceeding Fact for Scott-closures is:

21.1.10. Fact.

(1) If D is an algebraic lattice and f is a Scott-closure of D then $FIX_D(f) = f(D)$ is, again, an algebraic lattice.

(2) Moreover, in these conditions, $f(K(D)) = K(f(D)) = K(FIX_D(f))$, i.e., the compacts of the fixed point lattice of f are all and only the f-images of D-compacts.

Proof. See [Scott 76,82a] or [Gierz et al. 80], I.4.9. □

One has also the following representation result for algebraic lattices.

21.1.11. Proposition. (Representation Theorem for Algebraic Lattices).

(1) Each algebraic lattice D is isomorphic to the fixed point set of some Scott continuous closure operator $f : 2^X \to 2^X$, for some set X.

Analogously,

(2) each algebraic lattice D is isomorphic to an algebraic closure system $\mathbb{C}_X \subseteq 2^X$, for some set X (i.e., D is isomorphic to a "subalgebra" of 2^X).

Proof. Standard. See [Gierz et al. 80], I.4.15. □

This result can be slightly improved for the countable case (i.e., for ω-algebraic lattices):

21.1.12. Proposition. (Representation Theorem for Scott domains). Each Scott domain (= algebraic lattice with countably many compacts) is isomorphic to the fixed point set of some Scott (continuous) closure $f : P\omega \to P\omega$.

Proof. [Scott 76], Theorem 5.2 or [Scott 82,82a]. □

21.2. Computing with the Graph Model.

The construction of the so-called Graph Model $P\omega$ originates with Gordon Plotkin [Plotkin 72]. Independently, Dana Scott advocated similar ideas since 1973 and worked out the full details in [Scott 75a,76]. For earlier sources see [Myhill & Sheperdson 55] and [Rogers 67] 9.7, 9.8.

In the original approach, the method of construction relies on the fact that the algebraic lattice FUN of Scott continuous functions on $P\omega$ (ordered "pointwise") can be embedded as a retract into the lattice $P\omega$. The retract-

embedding is established via a specific coding of the (graphs of the) Scott continuous functions as subsets of $\mathbb{N}$.

This makes $P\omega$ into a lambda-model, whose representable functions are exactly the Scott continuous maps of $P\omega$. Here are the (minimal) details of construction:

(1) Accessing compacts. An immediate consequence of the fact that $K(P\omega)$, the set of $P\omega$-compacts, is r.e. is that one can access its elements recursively: we can find an effective enumeration $(e_n)_{n\in\mathbb{N}}$ of $K(P\omega)$. The choice of such a map is however important and has to be made cum grano salis.

In what follows, we fix an effective enumeration

$$\mathbf{fin} = n \mapsto e_n : \mathbb{N} \to K(P\omega)$$

of the set $K(P\omega)$ by (see, e.g., [Rogers 76]):

$e_n = \{k_0, \ldots, k_{m-1}\}$, with $k_0 < k_1 < \ldots < k_{m-1}$

iff

$$n = \sum_{i<m} 2^{k_i} = 2^{k_0} + \ldots + 2^{k_{m-1}}.$$

Now **fin** is standard and one can compute it in many different manners: one way of doing it is by "subtree recursion" in B_0 the universal ("oriented") binary tree, with appended root 0.

To make the recursion explicit, define an auxiliary map

$$\mathbf{succ}_{\mathbf{fin}} : K(P\omega) \to K(P\omega)$$

by

$$\mathbf{succ}_{\mathbf{fin}}(\emptyset) = \emptyset$$
$$\mathbf{succ}_{\mathbf{fin}}(\{k_0, \ldots, k_{m-1}\}) = \{k_0+1, \ldots, k_{m-1}+1\}.$$

Write, for convenience, with $e \in K(P\omega)$ and $n \in \mathbb{N}$,

$$e \,\hat{+}\, n := \mathbf{succ}_{\mathbf{fin}}^{\,n}(e),$$

iterating as expected. Example: $\{0,1\} \,\hat{+}\, 2 = \{2,3\}$.

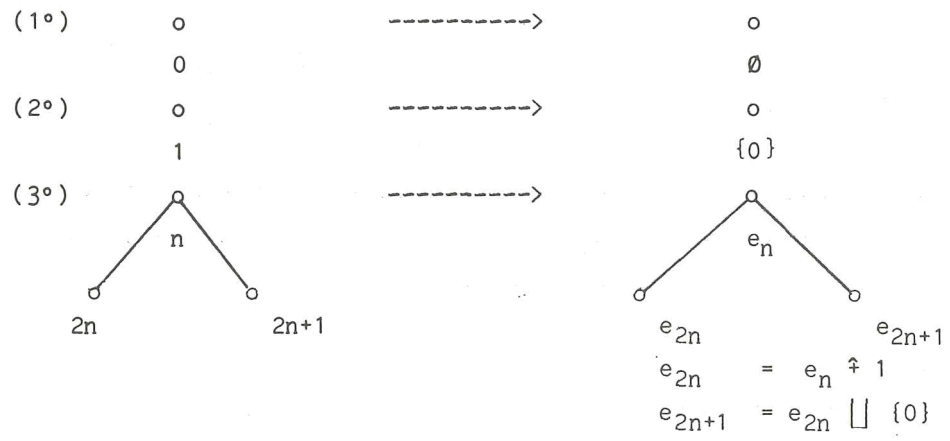
Somewhat redundantly, the recursion works as follows:

$$e_0 = \emptyset$$
$$e_1 = \{0\}$$
$$e_{2n} = e_n \hat{+} 1 \qquad (n \geq 0).$$
$$e_{2n+1} = e_{2n} \sqcup \{0\} \qquad (n \geq 0).$$

and the corresponding "program" computing **fin** is (à la Dijkstra):

if n=0 **then** $\emptyset$
else if n = 2k **then** $e_k \hat{+} 1$
$\square$ **if** n = 2k + 1 **then** $(e_k + 1) \sqcup \{0\}$.

This corresponds to a tree-like computation (in B_0), depicted graphically as follows:



One can also compute **fin** in a more "algebraic" manner, by noting that

(1°°) $(e_i \sqcup e_j) \hat{+} p = (e_i \hat{+} p) \sqcup (e_j \hat{+} p)$

(2°°) $(e_k \hat{+} p) \hat{+} q = e_k \hat{+} (p + q)$

etc., for e_i, e_j, $e_k \in K(P\omega)$ and $p, q \in \mathbb{N}$, the remaining rules for $\sqcup$ (as applied to finite sets) being unchanged.

We realize easily that **fin** is primitive recursive, one-one, onto $K(P\omega)$, with (primitive) recursive inverse:

$$\mathbf{fin}^{-1} = e_n \mapsto n : K(P\omega) \to \mathbb{N}$$

defined as expected.

(2) Pairing naturals. Recall that the standard Cantor coding

$$\mathbf{code} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

of the pairs of natural numbers onto $\mathbb{N}$ is given by

$$\mathbf{code}(m,n) = 1/2(m+n)(m+n+1) + n \qquad (m,n \in \mathbb{N}).$$

This is a primitive recursive map, one-one, onto $\mathbb{N}$, with primitive recursive "projection" (inverses) **left, right**: $\mathbb{N} \to \mathbb{N}$ (see e.g., [Mal'cev 70], II.3.3) and, as well known, the triple (**code, left, right**) forms a (computable) surjective pairing in the sense that, for all $n, n_0, n_1 \in \mathbb{N}$,

$$\mathbf{left}(\mathbf{code}(n_0, n_1)) = n_0$$
$$\mathbf{right}(\mathbf{code}(n_0, n_1)) = n_1$$
$$\mathbf{code}(\mathbf{left}(n), \mathbf{right}(n)) = n.$$

The "projections" **left, right** resp. (viewed as infinite sequences of naturals) are easily obtained by successively appending the initial segments of $\mathbb{N}$, with the reversed resp. the natural order:

left : 0 1 0 2 1 0 3 2 1 0 4 3 2 1 0 5 4 3 2 1 0 ... ...
right: 0 0 1 0 1 2 0 1 2 3 0 1 2 3 4 0 1 2 3 4 5 ... ...

As a bijection, **code** has an inverse $\mathbf{code}^{-1}$ which enumerates $\mathbb{N} \times \mathbb{N}$ (without repetitions). The enumeration takes place effectively, along the finite ("small") diagonals of the infinite matrix $\mathbb{N} \times \mathbb{N}$.

In what follows, we write conveniently

$$(m,n) := \mathbf{code}(m,n) \qquad (m,n \in \mathbb{N}).$$

It is useful to note that, for all $m,n \in \mathbb{N}$,

(1) $n \leq (m,n)$ and $m \leq (m,n)$

with, in particular,

(2) $0 = (0,0)$ and $1 = (1,0)$.

(3) Retracting. The remaining (and final) step in the construction of the Graph Model exploits the fact that $P\omega$ is a reflexive lattice (domain), that is: FUN is a retract of $P\omega$.

The retract construction is quite familiar in mathematical practice; in

the case of concern one has to "retract continuously" the full model onto its (continuous) FUNction space (where the continuity is Scott).

21.2.1. Proposition. (G. Plotkin, D. Scott). There is a pair (a "retraction pair")

$$(\mathbf{fun}, \mathbf{graph})$$

of Scott continuous maps

$$\mathbf{fun} : P\omega \to [P\omega \to P\omega] \qquad \text{("retraction")}$$
$$\mathbf{graph} : [P\omega \to P\omega] \to P\omega \qquad \text{("section")}$$

with

$$\mathbf{fun} \circ \mathbf{graph} = \mathbf{id}_{FUN}.$$

Proof. Explicitly, the "retraction pair" is given by

$$\mathbf{fun}(x)(y) = \{m \in \mathbb{N} : \exists\, n \in \mathbb{N}.\ (e_n \subseteq y)\ \&\ ((n,m) \in x)\} \qquad \forall x, y \in P\omega.$$
$$\mathbf{graph}(f) = \{(n,m) \in \mathbb{N} : m \in f(e_n)\} \qquad \forall f \in FUN. \ \square$$

(Here, **fun** is onto FUN and **graph** is into $P\omega$, since $\mathbf{id}_{FUN}$ is the identity on FUN.)

Some details of the semantic constructions below will drastically depend on the fact that $P\omega$ is - qua reflexive lattice - an additive lattice domain in the sense of [Sanchis 77] (cf. also [Barendregt & Rezus 83]). This means that the "retraction pair" (**fun**, **graph**) satisfies the "additivity condition":

$$\mathbf{id}_{P\omega} \subseteq \mathbf{graph} \circ \mathbf{fun}$$

(where $\subseteq$ is "pointwise").

Note that **fun** and **graph** are not bijections. That is: we do not have yet a (lattice-)isomorphism $P\omega \simeq FUN$.

Finally, from the preceeding Proposition one has (by standard techniques, see [Meyer 82], [Barendregt 84]) that $P\omega$ is a lambda-model, i.e., a model of the type-free lambda-calculus $\lambda\beta$.

It is not necessary to insist on the $P\omega$-semantics of $\lambda\beta$, since we use, in the sequel, a type-free language LAMBDA (similar to Scott's LAMBDA in [Scott 74,75a,76]), an enriched version of $\lambda\beta$, qua language of $P\omega$. Actually, there is much more "structure" in $P\omega$, as captured via LAMBDA, than expressible within $\lambda\beta$. Some relevant details of this "structure" may be also captured in "partial" formalisms, $p\lambda\beta$, (polyadic lambda-calculus, with pattern matching abstraction for "tuples" or sequences), $\lambda\beta SP$ (lambda-calculus with surjective

pairing), which contain $\lambda\beta$ properly [Rezus 85].

Finally, $P\omega$ is not a model of extensional lambda-calculus $\lambda\beta\eta$, but this is harmless for our purposes.

21.3. The language LAMBDA: syntax and semantics.

The type-free language LAMBDA used in what follows extends the language of pure type-free lambda-calculus $\lambda\beta$ [Barendregt 84], with a "surjective pairing" (**pair**, **fst**, **snd**), three "arithmetic" primitives **0** (zero), **succ** (successor), **pred** (predecessor) and the Scott conditional [Scott 74], denoted here by **scond**. Rigorously, our LAMBDA is a proper extension of Scott's type-free language LAMBDA [Scott 74,75a,76], viz. the latter lacks the surjective pairing (**pair**, **fst**, **snd**) from the primitive syntax. It is, however, very likely both languages have the same strength as regards definability, upon the given interpretation in $P\omega$. The extended LAMBDA is introduced here in order to economize on considerations regarding computability and definability of functions $f : P\omega \to P\omega$. Scott's LAMBDA will be not mentioned any further in the present text.

The syntax and semantics of LAMBDA will be first described in a formal setting; afterwards, LAMBDA will be used rather loosely (neglecting technical details concerning the interpretation of LAMBDA-terms, valuations in the Graph Model, etc.).

LAMBDA: syntax. Formally, the syntax of (LAMBDA-)terms is as follows; let first Var be a denumerably infinite set of variables, the one has recursively:

(1)	Any variable x in Var is a term.	(indeterminates)
(2)	**0** is a term.	(zero)
(3)	if a, b, c are terms then so are	
(31)	(ab)	(application terms)
(32)	(λx.a)	(abstraction terms)
(33)	(**pair** a b)	(pairing)
(34)	(**fst** c)	(first/left projection)
(35)	(**snd** c)	(second/right projection)
(36)	(**succ** a)	(successor of a)
(37)	(**pred** a)	(predecessor of a)
(38)	(**scond** c a b)	("if c then a else b")

Parentheses will be omitted whenever possible. Free and bound variables occurring in a (LAMBDA-)term, open and closed (LAMBDA-)terms etc. are supposed to be defined in the standard way. Also the alphabetic variance of LAMBDA-terms ("alpha conversion") must be handled as usual ("λ" is the only abstractor here). The substitution operator ("one step substitution") is denoted by

$$a[x:=b]$$

(read : "x becomes b in a"; where a, b are terms and x is a variable, occurring - even fictiously - in a). Equality (=) and inclusion ($\subseteq$) are available qua items of language: the only formulas of the language are of the form a = b or $a \subseteq b$ (a, b terms).

LAMBDA: semantics. The interpretation of LAMBDA-terms in Pω is given in the familiar first-order (Tarski) style.

A valuation is a map $\rho : Var \to P\omega$. The set of valuations in Pω is denoted by Env ("environments").

For $\rho \in Env$, the interpretation ("value") of a (LAMBDA-)term a at/under ρ in Pω (notation : $[\![a]\!]_\rho$) is defined recursively, on the structure of a, by:

(1°) $[\![x]\!]_\rho = \rho(x)$ for $x \in Var$

(2°) $[\![\mathbf{0}]\!]_\rho = \{0\}$ (this is the singleton {0})

(3°1) $[\![ab]\!]_\rho = \mathbf{fun}([\![a]\!]_\rho)([\![b]\!]_\rho)$

(3°2) $[\![\lambda x.a]\!]_\rho = \mathbf{graph}(f)$

where $f = \lambda d.[\![a]\!]_{\rho[x:=d]}$

with $\rho[x:=d] \in Env$ given by: for all $y \in Var$,

$$\rho[x:=d](y) = \begin{cases} \rho(y) & \text{if } y \neq x \\ d & \text{if } y = x, \end{cases}$$

(3°3) $[\![\mathbf{pair}\ a\ b]\!]_\rho = \{2n : n \in [\![a]\!]_\rho\} \sqcup \{2m+1 : m \in [\![b]\!]_\rho\}$

(3°4) $[\![\mathbf{fst}\ a]\!]_\rho = \{n : 2n \in [\![a]\!]_\rho\}$

(3°5) $[\![\mathbf{snd}\ a]\!]_\rho = \{m : 2m+1 \in [\![a]\!]_\rho\}$

(3°6) $[\![\mathbf{succ}\ a]\!]_\rho = \{n+1 : n \in [\![a]\!]_\rho\}$

(3°7) $[\![\mathbf{pred}\ a]\!]_\rho = \{n : n+1 \in [\![a]\!]_\rho\}$

(3°8) $[\![\textbf{scond}\ c\ a\ b]\!]_\rho = H_{ac} \sqcup H_{bc}$

where

$H_{ac} = \{n : (n \in [\![a]\!]_\rho) \ \&\ (0 \in [\![c]\!]_\rho)\}$

and

$H_{bc} = \{m : (m \in [\![b]\!]_\rho) \ \&\ (\exists k.\ k+1 \in [\![c]\!]_\rho)\}$

The semantic predicates can be defined, as expected, by

$$\vdash a \subseteq b \quad <=> \quad \forall_\rho \in Env.\ [\![a]\!]_\rho \subseteq [\![b]\!]_\rho$$

and

$$\vdash a = b \quad <=> \quad \forall \rho \in Env.\ [\![a]\!]_\rho = [\![b]\!]_\rho .$$

Actually, these notions will be mostly used here as relativized to specific first-order contexts.

One checks, by standard methods, that the interpretation map $[\![...]\!]_\rho$ is well-defined (see, e.g., [Meyer 82], [Barendregt 84]), this is not trivial, since **graph** makes sense only if applied to continuous maps.

An element **a** of $P\omega$ is LAMBDA-definable if it is the value of a LAMBDA-term a under the preceding semantics. Clearly, definable elements of the model are always denoted by closed terms ("LAMBDA-combinators"). Moreover, for closed terms, the interpretation does not depend on valuations.

From now on we shall loosely use the LAMBDA-notation as an interpreted language (along the semantics above): we write, e.g., $\lambda x.xa$ for the "official" $[\![\lambda x.xa]\!]_\rho$, etc. This practice will be extended, in the familiar way, to first-order formulas.

Basic LAMBDA-"combinators". The following are well-known or expected so:

21.3.1. Theorem (Continuity Theorem). The LAMBDA-definable functions $f : P\omega \to P\omega$ are Scott continuous.

Proof Adapt Theorem 2.1. in [Scott 76] to the present case, realizing that **pair**, **fst**, **snd** are Scott continuous. □

21.3.2. Theorem (Lambda Calculus Theorem). For all terms a, b, c, d one has

(r)	$a \sqsubseteq a$	and $a = a$
(t$\sqsubseteq$)	$(a \sqsubseteq b) \,\&\, (b \sqsubseteq c) \Rightarrow (a \sqsubseteq c)$	
(t=)	$(a = b) \,\&\, (b = c) \Rightarrow (a = c)$	
($\cong$)	$(a \sqsubseteq b) \,\&\, (b \sqsubseteq a) \Rightarrow (a = b)$	
(s)	$(a = b) \Rightarrow (b = a)$	
(m$\sqsubseteq$)	$(a \sqsubseteq b) \,\&\, (c \sqsubseteq d) \Rightarrow (ac \sqsubseteq bd)$	
(m=)	$(a = b) \,\&\, (c = d) \Rightarrow (ac = bd)$	
(ext$\sqsubseteq$)	$\forall x.a \sqsubseteq b \Rightarrow \lambda x.a \sqsubseteq \lambda x.b$	
(ext=)	$\forall x.a = b \Rightarrow \lambda x.a = \lambda x.b$	
(β)	$(\lambda x.a)b = a_{[x:=b]}$	
(η^-)	$a \sqsubseteq \lambda x.ax$, provided x is not free in a.	

Proof. Standard. □

From the above, it follows explicitly that $P\omega$ is a lambda-model. As the pure lambda-calculus primitives (application and functional abstraction) are readily available in LAMBDA, one can give explicit definitions for the familiar combinators:

I := $\lambda x.x$

K := $\lambda xy.x$ **K'** := $\lambda xy.y$

1_{ch} := $\lambda xy.xy$ (Church's numeral 1)

2_{ch} := $\lambda xy.x(xy)$ (Church's numeral 2)

and, in general, for all $n \in \mathbb{N}$,

$\mathbf{n}_{ch}$:= $\lambda xy.x^n y$,

iterating as expected.

B := $\lambda xyz.x(yz)$

S := $\lambda xyz.xz(yz)$

Y := $\lambda x.(\lambda y.x(yy))(\lambda z.x(zz))$ (Curry's paradoxical combinator)

These are standard names; the underlying notational conventions are as in lambda-calculus.

A new pure combinator, particularly productive in a later section is the **G**-combinator, introduced in [Barendregt & Rezus 83]; it is renamed here **gen**:

gen := $\lambda uvxy.v(uy)(x(uy))$.

We also use the familiar infix notation:

$f \circ g := \mathbf{B}\, f\, g = \lambda x.f(gx)$

(where x is not free in f, g). So,

$$2_{ch} = \lambda x.x \circ x.$$

For reasons appearing below, 2_{ch} will be relabelled **ret** later.

Another familiar result :

21.3.3. Theorem (First Recursion Theorem : D. Park). Where u is the graph of a Scott continuous function f : Pω → Pω, one has

$$Yu = \mathbf{fix}\ f \quad (= \text{the least fixed point of } f).$$

Proof. Mutatis mutandis, as in [Park 76] or [Scott 76]. □

(In other words, one has **Y** = **fix** in the model: Tarski's fixed point functional is thus LAMBDA-definable and coincides with the interpretation of the Curry "paradoxical combinator" **Y**.)

With the following notation (Scott uses ⇀ differently!):

$$x + 1 := \mathbf{succ}\ x,$$

$$x - 1 := \mathbf{pred}\ x$$

and

$$z \rightharpoonup x,y := \mathbf{scond}\ z\ x\ y$$

one has

$$\mathbf{succ} = \lambda x.x+1$$

$$\mathbf{pred} = \lambda x.x-1$$

$$\mathbf{scond} = \lambda zxy.z \rightharpoonup x,y.$$

That is : **succ**, **pred**, **scond** are LAMBDA-definable elements of the model. Since **0** is a term, the singleton {0} is also definable. Easy calculations show that so must be the "finitary" ingredients of the lattice Pω, as, e.g., bottom, top, the singletons {n} ⊆ ℕ, etc. Specifically, the following elements (note the relabelling!) of Pω are definable:

$$\bot := \emptyset \qquad \text{and} \qquad \top := \mathbb{N}$$

$$\mathbf{n} := \{n\} \qquad (\text{for all } n \in \mathbb{N}),$$

$$\text{any finite set } e_n \qquad (\text{for } n \in \mathbb{N}).$$

It is important to keep in mind the following idiosyncrasies of Pω (qua lambda-model):

$$\bot = \lambda x.\bot \qquad \text{and } \top = \lambda x.\top$$

with also, for all a ∈ Pω,

$$\mathbf{0}\ a = \mathbf{0}.$$

Finally, the finitary sup and inf of the lattice $P\omega$ are definable. (For detailed proofs see [Scott 76].)

The explicit behavior of the Scott conditional **scond** is, however somewhat less comfortable. Indeed, one has, by cases, that, for all (LAMBDA-definable) x, y, z in $P\omega$,

$$z \mathbin{\mathrm{o}\!\!\rightarrow} x,y = \begin{cases} \bot & \text{if } z = \bot \\ x & \text{if } z = \mathbf{0} \\ y & \text{if } 0 \notin z \neq \bot \\ x \sqcup y & \text{if } 0 \in z \neq \mathbf{0}, \end{cases}$$

and the values at z, for $0 \in z, z \neq \mathbf{0}$ are rather ad hoc.
The main use of **scond** is readily suggested by its action on singletons, as values of z, viz.,

$$\mathbf{0} \mathbin{\mathrm{o}\!\!\rightarrow} x,y = x$$

$$\mathbf{n+1} \mathbin{\mathrm{o}\!\!\rightarrow} x,y = y \qquad (x,y \in P\omega,\ n \in \mathbb{N}).$$

it is equally easy to check that

$$\bot \mathbin{\mathrm{o}\!\!\rightarrow} x,y = \bot$$

and

$$\top \mathbin{\mathrm{o}\!\!\rightarrow} x,y = x \sqcup y \qquad (x,y \in P\omega)$$

also

$$\mathbf{scond}\ \bot = \mathbf{scond}\ \bot\ \bot = \mathbf{scond}\ \bot\ \bot\ \bot = \bot$$

while

$$\text{if either } x \neq \bot \text{ or } y \neq \bot \text{ then } (\lambda z.z \mathbin{\mathrm{o}\!\!\rightarrow} x,y) \neq \bot$$

whence, for all $z \in P\omega$, and x,y as above, if $z \neq \bot$ then

$$z \mathbin{\mathrm{o}\!\!\rightarrow} x,y \neq \bot.$$

finally, for all $x,y \in P\omega$,

$$0 \notin \lambda z.\mathbf{scond}\ z\ x\ y.$$

We shall often use a nicer operator: the doubly strict conditional given by

$$\mathbf{cond} := \lambda zxy.\mathbf{scond}\ z\ (\mathbf{scond}\ z\ x\ \top)(\mathbf{scond}\ x\ \top\ y).$$

In analogy with **scond**, one has:

$$z \twoheadrightarrow x,y := \mathbf{scond}\ z\ x\ y \qquad (x,y,z \in P\omega)$$

and, clearly, **cond** is definable, Scott continuous, etc. The semantics of **cond** is given by the following

21.3.4. Proposition. For all $x,y,z \in P\omega$,

(1) $z \twoheadrightarrow x,y = \{n \in x : 0 \in z\} \bigsqcup \{m \in y : \exists k.\ k+1 \in z\} \bigsqcup \{p \in \mathbb{N} : (0 \in z) \,\&\, (\exists k.\ k+1 \in z)\}$

whence

(2) $(z \twoheadrightarrow x,y) = (z \twoheadrightarrow_{\sigma} x,y) \bigsqcup \{p \in \mathbb{N} : (0 \in z) \,\&\, (\exists k.\ k+1 \in z)\}$

and thus

(3) $(z \twoheadrightarrow_{\sigma} x,y) \subseteq (z \twoheadrightarrow x,y)$.

Proof. Trivial. □

The explicit behavior of **cond** is readily transparent: for all $x,y,z \in P\omega$,

$$z \twoheadrightarrow x,y = \begin{cases} \bot & \text{if } z = \bot \\ x & \text{if } z = \mathbf{0} \\ y & \text{if } 0 \notin z \neq \bot \\ \top & \text{else (i.e., } 0 \in z \neq \mathbf{0}). \end{cases}$$

Thus, unlike for **scond** one would rather have, for all $x,y \in P\omega$,

$$\top \twoheadrightarrow x,y = \top$$

and clearly, **cond** is "doubly strict", giving

$$\mathbf{cond}\ \bot = \bot \text{ and } \mathbf{cond}\ \top = \top$$

(while, although $\mathbf{scond}\ \bot = \bot$, we had, in general, $\mathbf{scond}\ \top = \lambda xy.x \bigsqcup y \neq \top$).

An important feature of $P\omega$ consists of the fact it admits of a definable and thus Scott continuous surjective pairing. (This is also the case with other specific lambda-models, as e.g., Scott's inverse limit construction(s) D_∞, although the very reasons are different! Cf. [Scott 72,73], etc.).

It has been already noted that the functions **pair**, **fst**, **snd** are Scott continuous. In the sequel we use the following notation:

$$\langle x,y \rangle := (\mathbf{pair}\ x\ y)$$
$$(u)_0 := (\mathbf{fst}\ u)$$
$$(u)_1 := (\mathbf{snd}\ u)$$

sparing on parentheses, whenever possible. These maps behave, indeed, as a pairing, for one has a

21.3.5. Lemma. For all $x, x', y, y', u, u' \in P\omega$,

(1) $\langle x,y \rangle = \langle x',y' \rangle \Leftrightarrow (x = x') \,\&\, (y = y')$

(2) $u = u' \Leftrightarrow ((u)_0 = (u')_0) \,\&\, ((u)_1 = (u')_1)$.

(3) However, one can have $u \neq u'$ and still $(u)_1 = (u')_i$, for $i = 0$ or $i = 1$, but not both (just in case $(u)_j \neq (u')_j$, for that $j \in \{0,1\}$ with $i \neq j$).

(4) Analogously for $\sqsubseteq$.

Proof. Exercise. ■

Moreover, the triple (**pair**, **fst**, **snd**) forms a surjective pairing:

21.3.6. Theorem. (Surjective Pairing Theorem). For all $x, y, u \in P\omega$,

(1) $\langle x,y \rangle_0 = x$ and $\langle x,y \rangle = y$

while

(2) $\langle u_0,u_1 \rangle = u$.

Proof. From definitions. □

These pairs distribute well over finitary sups and infs, viz.

21.3.7. Corollary. For all $x, x', y, y' \in P\omega$,

(1) $\langle x,y \rangle \sqcup \langle x',y' \rangle = \langle x \sqcup x', y \sqcup y' \rangle$

(2) Similarly for $\sqcap$.

Proof. Trivial. □

Note that this pairing exploits the specific behavior of $P\omega$, qua power set of $\mathbb{N}$.

It is now relatively easy to imagine various "test functions" for appropriate properties of elements in arbitrary sets $x \in P\omega$. We have some use, later, for the following parity test.

$$\mathbf{par} := \lambda x.\ (x_0 \mapsto 0,\ 0) \sqcup (x_1 \mapsto 1,\ 1).$$

Obviously, **par** is LAMBDA-definable, Scott continuous and so on. Its behavior is completely described by the following semantics:

21.3.8. Lemma. (The Parity Test). For all $x \in P\omega$,

$$\mathbf{par}\ x = \begin{cases} e_0 = \bot, & \text{if } x = \bot \\ e_1 = 0, & \text{if } x \neq \bot \text{ and } \forall p \in x.\ p \text{ even} \\ e_2 = 1, & \text{if } x \neq \bot \text{ and } \forall p \in x.\ p \text{ odd} \\ e_3 = \{0,1\}, & \text{else.} \end{cases}$$

Proof. Straightforward, from definition. □

The preceeding considerations can be easily extended to *sequence-forming* operators (with the corresponding "projections"), either by iterating the **pair**-construction or by generalizing LAMBDA in the obvious way, but such extensions are unnecessary.

22. A Universe for Scott closures.

In this section we go into a deeper analysis of the structure of Scott closures in Pω.

22.1. Scott retractions and closures.

As expected, the concepts of a Scott retraction and closure (21.1) can be easily captured in LAMBDA. We revise the corresponding terminology as appropriate.

22.1.1. Definition. Let $a \in FUN$. Then

(1) a is a (Scott) retraction (map of Pω) if $a = a \circ a$.

(2) a is a (Scott) closure (map of Pω) if it is a retraction with $I \subseteq a$.

(3) $a^\circ = \{ ax : x \in P\omega \}$ is the range of a (in Pω).

(4) $FIX(a) = \{ x \in P\omega : ax = x \}$ is the fixed point set of a (in Pω).

22.1.2. Notation.

$x\ \mathbf{E}\ a \iff x \in FIX(a)$.

$RET = \{ a : a \text{ is a retraction} \}$.

CLOS = { a : a is a closure }.

22.1.3. Remarks.

(1) $\forall a \in RET.\ \forall x \in P\omega.\ ax\ \mathbf{E}\ a.$

(2) $CLOS \subsetneq RET.$

Proof. (1) Clear, from definition. (2) The inclusion follows from definitions. It is strict, for one can take $\bot \in RET$. Now $I \not\sqsubseteq \bot$, since otherwise $I = \bot = \emptyset$, while one can check easily that $\mathbf{I} \neq \emptyset$. □

The following distinguo will be also relevant.

22.1.4. Definition. A (Scott continuous) function $f \in FUN$ is

(1) strict below if $\bot \in f$,

(2) strict above if $\top \in f$ and

(3) doubly strict if it is strict below and above.

22.1.5. Remarks.

(1) **K** is doubly strict and $\bot$ and $\top$ are the only fixed points of **K**.

(2) Note also that $\bot, \top \in FUN$; $\bot$ is strict below but not above and $\top$ is strict above but not below. That is : $\bot\ \mathbf{E}\ \bot$ and $\top\ \mathbf{E}\ \top$.

We collect now the relevant facts about FUN :

22.1.6. Theorem (Continuous Function Theorem).

(1) **fun** = 1_{ch} (thus **fun** $\in$ FUN); so **fun** is definable.

(2) $\mathbf{I} \sqsubseteq$ **fun** (this is the "additivity property" of $P\omega$).

(3) **fun** $\in$ RET (that is : **fun** o **fun** = **fun**) and, moreover,

(4) **fun** $\in$ CLOS (**fun** is a closure map).

(5) FUN = FIX(**fun**) and

(6) **fun** is doubly strict ($\bot, \top\ \mathbf{E}$ **fun**).

Proof. Trivial. □

22.1.7. Corollary. FIX(**fun**) is a complete lattice under set inclusion.

Proof. For FUN = FIX(**fun**), by **22.1.6.** The result then follows by Tarski's Fixed Point Theorem. □

One may also realize that, with our notation, one has

I, K, K', S E fun

Analogously, the basic facts on Scott retractions are :

22.1.8. Theorem (Retraction Theorem). Let a ε RET. Then

(1) a ε FUN but RET $\subsetneq$ FUN. So, in the end, one has the (strict) inclusions CLOS $\subsetneq$ RET $\subsetneq$ FUN.

(2) a° = FIX(a).

(3) Where **ret** := 2_{ch}, one has RET = FIX(**ret**),

$\forall x \in P\omega.\ x \in RET \iff x\ \mathbf{E}\ \mathbf{ret}$,

(4) **ret** is not a retraction.

Proof. (1) The inclusion is clear. It is strict, for **K** is not in RET, but one has **K E fun**. (2) See **21.1.8.** (3) If a ε RET then a = a o a = **ret** a and conversely. (4) One has now **ret** o **ret** = 2_{ch} o 2_{ch} = $4_{ch} \neq 2_{ch}$ = **ret**. □

22.1.9. Terminology. A set D $\subseteq$ Pω is E-representable (in Pω) if there is an element $\mathbf{d}_D$ ε Pω such that

$\forall x \in P\omega.\ x \in D \iff x\ \mathbf{E}\ \mathbf{d}_D$.

In this case, we say that $\mathbf{d}_D$ is a representing element (or a universe) for D in Pω. This way of speaking will be used rather freely (modulo **graph** say). Clearly, Pω itself as well as the sets FUN, RET are **ret**-representable : take **I**, 1_{ch}, 2_{ch}, resp., as representing elements. An element d ε Pω is (**E**)-reflexive if d **E** d. So are **I**, 1_{ch} = **fun**, while **ret** is not **E**-reflexive. For a given set D $\subseteq$ Pω (resp. D $\subseteq$ FUN), a reflexive universe for D will be said to be a universal element for D; universal elements for FUN, RET, CLOS, resp., are called universal functions, universal retractions, resp., universal closures. Sets D $\subseteq$ Pω which are representable by universal elements are self-representable. Thus **I** is a universal element for the full model, whereas **fun** is a universal function. Note that if d and d′ are both universes for some subset D of Pω (resp. FUN) it may be not the case that d = d′ in Pω.

The following Theorem, due to Yu. Ershov and C. Hosono and M. Sato, independently, shows, ultimately, that RET is not **E**-self-representable in Pω.

22.1.10. Theorem. RET is not a continuous lattice (under set inclusion).
Proof. See [Scott 76] for a sketch of Ershov's proof or [Hosono & Sato 77]. ▣

22.1.11. Corollary. There is no universal retraction in $P\omega$.
Proof. Suppose otherwise and let **r** be a universe for RET with **r** ε RET. Then RET = **fix(r)** and, by a standard result, mentioned in **21.1**, RET must be a continuous lattice (under set inclusion), thereby contradicting **22.1.10.** ▣

22.1.12. Examples. One has $\bot$, $\top$, **I**, **K'** $\mathbf{E}$ **ret**, but **K** and, as shown above, **ret** itself are not in RET.

22.1.13. Remarks. Define, for a ε RET, $\bot_a := a\bot$ and similarly for $\top_a$. Recall that a° = FIX(a) is a complete (in fact, continuous) lattice. Then

(1) $\bot_a$ is the bottom of a° and $\top_a$ is the top of this lattice.

(2) If, moreover, a is strict below then $\bot_a = \bot$ and if it is strict above then $\top_a = \top$, whence for a doubly strict retract a, $P\omega$ and the (continuous) lattice a° = FIX(a) have the same bottom and top elements.

22.2. Universal closure.

A nice feature of the Graph Model consists of the fact that CLOS is the range of a Scott continuous closure : this follows from an earlier observation of Per Martin-Löf and Peter Hancock, cf. also [Scott 76]. Actually, the result holds for any additive lattice domain, as noted in [Barendregt & Rezus 83] (see also [McCracken 79]). The basic construction can be obtained as follows.

22.2.1. Definition. $\mathbf{V} := \lambda xy.\mathbf{Y}(\lambda z.y \sqcup xz)$.

22.2.2. Lemma. For all x, y ε $P\omega$,

(1) $\mathbf{V}x(\mathbf{V}xy) = \mathbf{V}xy$,

(2) $y \sqsubseteq \mathbf{V}xy$

That is, ultimately : $\forall x \varepsilon P\omega$. $\mathbf{V}x$ is a closure.

Proof. Easy. □

22.2.3. Theorem (Universe Theorem for Closures).

(1) $\forall x \in P\omega.\ x \in CLOS \iff x \in \mathbf{V}$.

(2) Moreover, **V** is a closure.

Proof. Cf. [Barendregt & Rezus 83]. □

22.2.4. Remarks.

(1) $\top$ is a closure (i.e.,a fixed-point of **V**). However, by Scott induction, one has that $\mathbf{V}\bot = \mathbf{V}\mathbf{I} = \mathbf{I}$, so $\bot$ is not a closure, while **I** is one.

(2) **I** has the largest fixed-point set (the full space) and is doubly strict (as a closure).

(3) On the other hand, $\top$ is the only fixed point of $\top$ (and, mutatis mutandis, this is also true of $\bot$).

(4) Note also that $\top = \lambda x.\mathbf{V}x\top$ and

(5) $\mathbf{I} = \lambda x.x \twoheadrightarrow x,\ x$.

(6) Finally, $\forall a \in CLOS.\ a^\circ$ is an algebraic lattice, while the representation theorem for Scott domains says that

(7) every Scott domain D is such that $a^\circ \simeq D$, for some closure $a \in CLOS$ (lattice-isomorphism).

23. "Structuring" the Universe.

In this section we establish several closure conditions in $P\omega$, matching the generalized product, generalized sum, and disjoint sum constructs of Martin-Löf's Constructive Type Theory. One wants, of course, these conditions be expressible within LAMBDA, possibly with the help of the auxiliary "fixed point predicate" **E**.

23.1. Generalized products of closures.

A straightforward consequence of a result in [Barendregt & Rezus 83] (see also [McCracken 79]) is that CLOS is "closed under" arbitrary cartesian products of families of Scott continuous closures. For completeness, we repeat the construction here, as relativized to the Graph Model.

Recall first that, in LAMBDA, we had **gen** := $\lambda uvxy.v(uy)(x(uy))$, the **G**-combinator of [Barendregt & Rezus 83].

23.1.1. Definition.

(1) $\Pi x{:}a.b_{[x]} := \mathbf{gen}\ a\ (\lambda x.b_{[x]})$

(2) $\lambda x{:}a.b_{[x]} := (\lambda x.b_{[x]})\ o\ a$

(3) $a \to b := \mathbf{gen}\ a\ (\mathbf{K}\ b) = (\lambda x.b\ o\ x\ o\ a)$.

Note. Here and everywhere in the sequel,"$\lambda x{:}a.b_{[x]}$" is meant to stand for "typed lambda-abstraction" (whenever "a" represents a type), i.e., for a function with domain restricted to a^o. The function is, again, a "typed object". For appropriate "b" 's, possibly depending uniformly on x, the behavior of the corresponding "type" can be easily read off from the following

23.1.2. Theorem (Generalized Product Theorem). Let $a \mathbf{E} \mathbf{V}$. Then

(1) $\forall x \mathbf{E} a.\ b_{[x]} \mathbf{E} \mathbf{V} \iff \Pi x{:}a.b_{[x]} \mathbf{E} \mathbf{V}$.

(2) $\forall x \mathbf{E} a.\ c_{[x]} \mathbf{E} b_{[x]} \iff \lambda x{:}a.c_{[x]} \mathbf{E} \Pi x{:}a.b_{[x]}$.

(3) $f \mathbf{E} \Pi x{:}a.b_{[x]} \Rightarrow f = \lambda x{:}a.fx$.

In fact,

(4) $f \mathbf{E} \Pi x{:}a.b_{[x]} \iff (\forall x \mathbf{E} a.\ fx \mathbf{E} b_{[x]})\ \&\ (f = \lambda x{:}a.fx)$,

(for all f, not containing x free).

Proof. As in [Barendregt & Rezus 83], 1.14. □

Note. Theorem **23.1.2** generalizes in the obvious way Scott's Function Space Theorem in [Scott 76] (cf. Theorem 5.3, and our Corollary **23.1.6** below).

Moreover, **gen** respects "strictness" in the following sense :

23.1.3. Theorem. Let a, $b_{[x]}$ be LAMBDA-terms such that

$$a \mathbf{E} \mathbf{V},$$

$$\forall x \mathbf{E} a.\ \bot, \top \mathbf{E} b_{[x]} \mathbf{E} \mathbf{V}.$$

Then also

$$\bot, \top \mathbf{E} \Pi x{:}a.b_{[x]} \mathbf{E} \mathbf{V}.$$

Proof. Easy. □

23.1.4. Corollary (Evaluation : typed β-rule). Let $a \mathbf{E} \mathbf{V}$ and assume that $b_{[x]}$ is such that

$$\forall x.\ \mathbf{E}\ a.\ b_{[x]}\ \mathbf{E}\ \mathbf{V}.$$

If

$$\lambda x{:}a.c_{[x]}\ \mathbf{E}\ \Pi x{:}a.b_{[x]}\ (\mathbf{E}\ \mathbf{V})$$

and

$$d\ \mathbf{E}\ a\ (\mathbf{E}\ \mathbf{V})$$

then

$$(\lambda x{:}a.c_{[x]})d \quad = \quad c_{[x:=d]}\ \mathbf{E}\ b_{[x:=d]}\ \mathbf{E}\ \mathbf{V}.$$

Proof. By **23.1.2**, (1), (2). □

23.1.5. Corollary (Functionality : typed η-rule). Let $a\ \mathbf{E}\ \mathbf{V}$ and assume that $b_{[x]}$ is such that

$$\forall x\ \mathbf{E}\ a.\ b_{[x]}\ \mathbf{E}\ \mathbf{V}$$

Then, where $f\ \mathbf{E}\ \Pi x{:}a.b_{[x]}$ (and f does not contain x free), one has also

$$f = \lambda x{:}a.fx \quad = \quad (\lambda x.fx) \circ a = f \circ a.$$

Proof. By **23.1.2**, (3) and the definition of typed lambda-abstraction. □

So full type-free extensionality is not necessary (in the model) in order to interpret "extensional typed lambda-calculus".

In particular, the generalized products of closures relativize as intended to the usual function space constructs of [Scott 76].

23.1.6. Corollary (Function Space Theorem, [Scott 76]). Let $a\ \mathbf{E}\ \mathbf{V}$. Then

(1) $b\ \mathbf{E}\ \mathbf{V} \iff (a \to b)\ \mathbf{E}\ \mathbf{V}$,

(2) $\forall x\ \mathbf{E}\ a.\ c_{[x]}\ \mathbf{E}\ b_{[x]}\ \mathbf{E}\ \mathbf{V} \iff \lambda x{:}a.c_{[x]}\ \mathbf{E}\ (a \to b)\ \mathbf{E}\ \mathbf{V}$,

(where x does not occur free in b), and, moreover,

(3) $f\ \mathbf{E}\ (a \to b) \iff (\forall x\ \mathbf{E}\ a.\ fx\ \mathbf{E}\ b)\ \&\ (f = \lambda x{:}a.fx)$.

Proof. From Definition **23.1.1** and Theorem **23.1.2**. □.

In the strict case :

23.1.8. Corollary ([Scott 76]). If $a\ \mathbf{E}\ \mathbf{V}$ and $\bot, \top\ \mathbf{E}\ b\ \mathbf{E}\ \mathbf{V}$ then also

$$\bot, \top\ \mathbf{E}\ (a \to b)\ \mathbf{E}\ \mathbf{V}.$$

Proof. Direct calculation. □

23.1.8. Remark. As noted, independently, in [McCracken 79] and [Barendregt & Rezus 83], the construction behind **23.1.2** can be repeated in every additive lattice domain D. Apparently, it is not so for the "strict" version of this construction, appearing in **23.1.3**. (Note that $\bot, \top$ **E K** is an idiosyncrasy of the Graph Model! That is : the bottom of the lattice D must be equal to the "pointwise" bottom of [D → D] and similarly for top elements.)

23.2. Generalized sums of closures.

The fact that $P\omega$ admits of a "representable" surjective pairing allows the following construction :

23.2.1. Definition.

(1) **spec** := $\lambda ufx.< ux_0, f(ux_0)x_1 >$

(2) $\Sigma x{:}a.b_{[x]}$:= **spec** a $(\lambda x.b)_{[x]}$ (= $\lambda z.< az_0, b_{[x:=az_0]}z_1 >$)

(3) a × b := **spec** a (**K** b) (= $\lambda z.< az_0, bz_1 >$)

(4) **split** := $\lambda xy.yx_0x_1$.

23.2.2. Theorem (Generalized Sum Theorem). Let a **E V**. Then

(1) $\forall x$ **E** a. $b_{[x]}$ **E V** <=> $\Sigma x{:}a.b_{[x]}$ **E V**.

(2) Assume also that

$$\forall x \text{ E } a.b_{[x]} \text{ E V}$$

(or, alternatively, that

$$\Sigma x{:}a.b_{[x]} \text{ E V}).$$

Then

(21) $\forall u,v \in P\omega.$ (u **E** a) & (v **E** $b_{[x:=u]}$) => < u,v > **E** $\Sigma x{:}a.b_{[x]}$

(22) $\forall z \in P\omega.$ z **E** $\Sigma x{:}a.b_{[x]}$ => (z_0 **E** a) & (z_1 **E** $b_{[x:=z_0]}$)

(23) whence also

$\forall z \in P\omega.$ z **E** $\Sigma x{:}a.b_{[x]}$ <=> (z = < z_0, z_1 >) & (z_0 **E** a) & (z_1 **E** $b_{[x:=z_0]}$)

Proof. (1), (21) and (22) : routine computations in $P\omega$. (23) follows from (21), (22) by **21.3.6** (surjectivity of pairing). □

23.2.3. Remark. Note that, for i = 0,1, one has $\bot_i = \bot$ and $\top_i = \top$. Hence

$$< \bot, \bot > = \bot \quad \text{and} \quad < \top, \top > = \top.$$

23.2.4. Theorem (Strictness Theorem for Generalized Sums). If

$$\bot, \top \mathbf{E}\ a\ \mathbf{E}\ \mathbf{V}$$

and

$$\forall x\ \mathbf{E}\ a.\ \bot, \top \mathbf{E}\ b_{[x]}\ \mathbf{E}\ \mathbf{V}$$

then also

$$\bot, \top \mathbf{E}\ \Sigma x{:}a.b_{[x]}\ (\mathbf{E}\ \mathbf{V}).$$

Proof. Direct calculation. □

23.2.5. Theorem (Splitting Theorem). Let $a\ \mathbf{E}\ \mathbf{V}$ and assume that

$$\forall x\ \mathbf{E}\ a.\ b_{[x]}\ \mathbf{E}\ \mathbf{V}$$

$$\forall z\ \mathbf{E}\ \Sigma x{:}a.b_{[x]}.\ h_{[z]}\ \mathbf{E}\ \mathbf{V}$$

with, moreover,

$$\forall x\ \mathbf{E}\ a.\ (\forall y\ \mathbf{E}\ b_{[y]}.\ d_{[x,y]}\ \mathbf{E}\ h_{[z:=\langle x,y\rangle]})$$

Then, one has the following conclusions :

(1) If $c\ \mathbf{E}\ \Sigma x{:}a.b_{[x]}$ then

$$c_0\ \mathbf{E}\ a\ (\mathbf{E}\ \mathbf{V})$$

$$c_1\ \mathbf{E}\ b_{[x:=c_0]}\ (\mathbf{E}\ \mathbf{V})$$

$$\mathbf{split}\ c\ (\lambda x{:}a.\lambda y{:}b_{[x]}.d_{[x,y]}) = d_{[x:=c_0,y:=c_1]}\ \mathbf{E}\ h_{[z:=c]}.$$

(2) If $c\ \mathbf{E}\ a\ \mathbf{V}$ and $e\ \mathbf{E}\ b_{[x:=c]}\ \mathbf{E}\ \mathbf{V}$, ceteribus paribus, then also

$$\mathbf{split}\ \langle c, e\rangle\ (\lambda x{:}a.\lambda y{:}b_{[x]}.d_{[x,y]}) = f(c,e)$$

where

$$f(c,e) = d_{[x:=c,y:=e]}\ \mathbf{E}\ h_{[z:=\langle c,e\rangle]}.$$

Proof. (1) By the definition of **split**, Theorem **23.2.1**, (2) and surjectivity of pairing. (2) Direct calculation. □.

From **23.2.2** and **23.2.4** one has the following results, present already in [Scott 76].

23.2.6. Corollary (Product Theorem, [Scott 76]). Let $a\ \mathbf{E}\ \mathbf{V}$. Then

$$b\ \mathbf{E}\ \mathbf{V} \iff a \times b\ \mathbf{E}\ \mathbf{V}.$$

Assume also that $b\ \mathbf{E}\ \mathbf{V}$. Then, for all $u,v \in P\omega$,

$$(u\ \mathbf{E}\ a)\ \&\ (v\ \mathbf{E}\ b) \Rightarrow \langle u,v\rangle\ \mathbf{E}\ a \times b$$

$$u\ \mathbf{E}\ (a \times b) \Rightarrow (u_0\ \mathbf{E}\ a)\ \&\ (u_1\ \mathbf{E}\ b)$$

whence

$$u\ \mathbf{E}\ (a \times b) \iff (u = \langle u_0, u_1\rangle)\ \&\ (u_0\ \mathbf{E}\ a)\ \&\ (u_1\ \mathbf{E}\ b).$$

Proof. By **23.2.2** and the definition of $\times$. □

In the strict case :

23.3.7. Corollary (Strictness for Products, [Scott 76]). If, $\bot$, $\top$ **E** a **E V** and $\bot$, $\top$ **E** b **E V** then also $\bot$, $\top$ **E** (a × b) (**E V**).

Proof. By 23.2.4 and the definition of ×. □

23.3. Disjoint sums of closures.

The following constructions establish closure under disjoint sums. Except for **when**,they are obliquely referred to in [Scott 76], (Theorem 5.4).

Note first that the semantics of the parity test **par** (21.3.8) gives :

$$\bot \textbf{ E par}$$
$$\textbf{par } x = \bot \implies x = \bot$$
$$\textbf{par } \top = \{0,1\}.$$

23.3.1.Definition.

(11) **sh** = λux.**0** ⨆ (u(x-1) + 1) ("shift")

(12) **sc** = λx.**0** ⨆ (x+1)

(2) **sum** = λxyz.**par** z ↠ < **sh** x z_0, $\bot$ >, < $\bot$, **sh** y z_1 >

(31) **inl** = λu.< **sc** u, $\bot$ >

(32) **inr** = λu.< $\bot$, **sc** u >

(4) **when** = λuxy.**par** u ↠ $x(u_0-1)$, $y(u_1-1)$.

23.3.2. Notation. For all a,b ε Pω, we write a + b := **sum** a b.

23.3.3. Remarks. For all a, x ε Pω,

(1) **sc** a = **0** ⨆ **succ** a

(2) **sh** a (**succ** a) = **sh** a (**sc** a) = **sc** (ax) = **0** ⨆ **sc** (ax) = **0** ⨆ (ax + 1)

(3) **inl** = < **0** ⨆ (x+1), $\bot$ >

(4) **inr** = < $\bot$, **0** ⨆ (x+1) >

whence

(5) **sc** = $(\textbf{inl})_0 = (\textbf{inr})_1$

Proof. From definitions. □

23.3.4. Lemma. For all u ε Pω,

(1) $u = \langle u_0, \bot \rangle \iff u = \{ 2n : n \varepsilon a' \}$, for some $a' \varepsilon P\omega$ and if,

moreover, $u \neq \bot$, then $a' \neq \bot$ (and u contains only even numbers).
Analogously,

(2) $u = \langle \bot, u_1 \rangle \iff u = \{ 2m+1 : m \in a'' \}$, for some $a'' \in P\omega$ and if, moreover, $u = \neq \bot$ then $a'' \neq \bot$ (and u contains only odd numbers).

Proof. Immediate, using the definition of the surjective pairing. □

23.3.5. Corollary. For all $a, x \in P\omega$,

(1) $x \mathbf{E} a \iff \mathbf{sh}\ a\ (\mathbf{sc}\ x) = \mathbf{sc}\ x$,

that is, we can "shift" the fixed point relation to appropriate "successors" :

$$x \mathbf{E} a \iff (\mathbf{sc}\ x) \mathbf{E} (\mathbf{sh}\ a).$$

(2) In particular, for all $a, b, u \in P\omega$,

$$u \mathbf{E} a \iff (\mathbf{inl})_0 \mathbf{E} (\mathbf{sh}\ a)$$
$$u \mathbf{E} b \iff (\mathbf{inr})_1 \mathbf{E} (\mathbf{sh}\ b).$$

Proof. (1) In one direction (=>) : from **23.2.3**, (2). Conversely, note that 0 is not an element of e+1, for $e \in P\omega$. So $ax + 1 = x + 1$, whence $ax = x$. (2) From (1) and **23.3.3**, (4). □

We obtain now the main theorem :

23.3.6. Theorem (Disjoint Sum Theorem). Let $a, b \in P\omega$. Then, for all $u \in P\omega$,

(1) $\bot, \top \mathbf{E} (a + b)$

(2) $u \mathbf{E} a \implies (\mathbf{inl}\ u) \mathbf{E} (a + b)$

(3) $u \mathbf{E} b \implies (\mathbf{inr}\ u) \mathbf{E} (a + b)$

(4) $u \mathbf{E} (a + b) \iff$
either $u = \bot$
or $u = \top$
or $\exists x \in P\omega.\ (u = \mathbf{inl}\ x) \,\&\, (x \mathbf{E} a)$
or $\exists y \in P\omega.\ (u = \mathbf{inr}\ y) \,\&\, (y \mathbf{E} b)$
with, in particular,
$x = u_0 - 1$ and $y = u_1 - 1$.

Proof. (1) By properties of **cond**, (see **21.3**). (2) Use Lemma **23.3.4** and properties of **par**. (3) Similarly. (4) Half of this (<=) follows from (1) - (3). Conversely, if $u \mathbf{E} (a + b)$, then the definition of **par** (and its explicit characterization in **21.3.8**), gives the necessary case-analysis, viz.,

(a) $u = \bot$, then $\mathbf{par} = \bot$ and $u \mathbf{E} (a + b)$, by the definition of +

(b) $u \neq \bot$, u contains only even numbers and $\mathbf{par}\ u = 0$.

In this case,

(i) $u = \langle 0 \sqcup a(u_0-1)+1 \perp \rangle$,

so

(ii) $u_0 = 0 \sqcup a(u_0-1)+1$

and,in particular,

$$0 \in u_0$$

whence

(iii) $u_0 = 0 \sqcup u_0$.

Now, (ii) gives

$$u_0 = 0 \sqcup \mathbf{sh}\ a\ u_0 = \mathbf{sh}\ a\ u_0$$

since $0 \in \mathbf{sh}\ a\ x$, for all $a, x \in P\omega$.

Thus

(iv) $u_0\ \mathbf{E}\ \mathbf{sh}\ a$.

With $x = u_0 - 1$, one has, from (iv),

(v) $(\mathbf{succ}\ x)_0\ \mathbf{E}\ \mathbf{sh}\ a$.

Rewriting (iii) gives

$$\mathbf{succ}\ x = 0 \sqcup x = \mathbf{sc}\ x,$$

so, from (v), one has

(vi) $(\mathbf{sc}\ x)_0\ \mathbf{E}\ \mathbf{sh}\ a$,

whence

(vii) $(\mathbf{inl}\ x)_0\ \mathbf{E}\ \mathbf{sh}\ a$

by 23.3.3, (5), and

(viii) $x\ \mathbf{E}\ a$

follows from Corollary 23.3.4, (21).

Obviously, then

$$u = \mathbf{inl}\ (ax) = \mathbf{inl}\ x$$

by (i) and (viii),resp., completing the case.

(c) $u \neq \perp$, u contains only odd numbers and $\mathbf{par}\ u = 1$; one argues as under (b);

(d) else $\mathbf{par}\ u = \{0,1\}$, u is a "mixed" subset of $\mathbb{N}$ and this is only possible for $u = \top$. □

Note that, in 23.3.6, nothing has been said about CLOS (a and b were arbitrary elements of the Graph Model). The Theorem says that + behaves, indeed, as a disjoint sum operation, with "injections" **inl**, **inr**. The

corresponding closure condition is given by the following

23.3.7. Theorem (Closure for Disjoint Sums). For all $a,b \in P\omega$,

(1) $a, b \mathbf{E} \mathbf{V} \Rightarrow (a + b) \mathbf{E} \mathbf{V}$

Moreover,

(2) $a, b \mathbf{E} \mathbf{V} \Rightarrow \bot, \top \mathbf{E} (a + b)$.

Proof. Routine case-analysis, using the properties of **par** (21.3.8) and the fact that

$$a \mathbf{E} \mathbf{ret} \Rightarrow (\mathbf{sh}\ a) \mathbf{E} \mathbf{ret},$$

etc. □

23.3.8. Theorem (Case Analysis). Let $a,b \mathbf{E} \mathbf{V}$. Assume that

$$\forall z \mathbf{E} (a + b).\ h_{[z]} \mathbf{E} \mathbf{V}$$
$$\bot \mathbf{E} h_{[z:=\bot]}$$
$$\top \mathbf{E} h_{[z:=\top]}$$

with also

$$\forall x \mathbf{E} a.\ d_{[x]} \mathbf{E} h_{[z:=(\mathbf{inl}\ x)]}$$
$$\forall y \mathbf{E} b.\ e_{[y]} \mathbf{E} h_{[z:=(\mathbf{inr}\ y)]}.$$

Then, one has the following conclusions

(1) if $c \mathbf{E} (a + b)$ then

$$\mathbf{when}\ c\ (\lambda x{:}a.d_{[x]})\ (\lambda y{:}b.e_{[y]}) \mathbf{E} h_{[z:=c]};$$

(21) if $p \mathbf{E} a$ then

$$\mathbf{when}\ (\mathbf{inl}\ p)\ (\lambda x{:}a.d_{[x]})\ (\lambda y{:}b.e_{[y]}) = d_{[x:=p]}$$

where

$$d_{[x:=p]} \mathbf{E} h_{[z:=(\mathbf{inl}\ p)]};$$

Analogously,

(22) if $q \mathbf{E} b$ then

$$\mathbf{when}\ (\mathbf{inr}\ q)\ (\lambda x{:}a.d_{[x]})\ (\lambda y{:}b.e_{[y]}) = e_{[x:=q]}$$

where

$$e_{[x:=q]} \mathbf{E} h_{[z:=(\mathbf{inr}\ q)]}.$$

Proof. Routine calculations. □

23.3.9. Remarks. One may replace the first three hypotheses of 23.3.8 by

$$\forall z \mathbf{E} (a + b).\ \bot, \top \mathbf{E} h_{[z]} \mathbf{E} \mathbf{V}.$$

Alternatively, the theorem holds under the following (stronger) hypotheses :

$$\bot, \top \;\mathbf{E}\; a \;\mathbf{V}$$
$$\bot, \top \;\mathbf{E}\; b \;\mathbf{V}$$
$$\forall z \;\mathbf{E}\; (a + b).\ \bot, \top \;\mathbf{E}\; h_{[z]} \;\mathbf{E}\; \mathbf{V}$$
$$\forall x \;\mathbf{E}\; a.\ d_{[x]} \;\mathbf{E}\; h_{[z:=(\mathbf{inl}\ x)]}$$
$$\forall y \;\mathbf{E}\; b.\ e_{[y]} \;\mathbf{E}\; h_{[z:=(\mathbf{inr}\ y)]}.$$

24. "Inhabiting" the Universe.

In this section we are going to diversify the landscape of our Universe (of closures) by identifying possible candidates for the representation of commonly accepted "ground data types" as,e.g., the finite types, the boolean type and the integer type. Actually, we shall bring "before the scenes" only a minimal amount of entities,in the spirit of a rather parcimonious Occam-razor policy. Thus, for instance, the boolean type **bool** = {**true**, **false**} will be conveniently identified with the two-element fixed-point set ("data type") $\mathbf{fin}_2 = \{0, 1\}$, while the elements of the finite (data-) types $\mathbf{fin}_n$, ($n \in \mathbb{N}$) will be among the elements of the integer type **nat** (relabelled here **int**) and will, in fact, coincide with the "official" (non-negative) integers of the Graph Model (modulo the preceeding convention $\mathbf{n} = \{n\}$).

24.1. Finite types as closures.

The finite (data-) types $\mathbf{int}_n$ ($n \in \mathbb{N}$) are defined by :

24.1.1. Definition.

$\mathbf{int}_0$:= $\lambda x.\mathbf{cond}\ x\ \top\ \top$
$\mathbf{int}_1$:= $\lambda x.\mathbf{cond}\ x\ 0\ \top$
$\mathbf{int}_2$:= $\lambda x.\mathbf{cond}\ x\ 0\ (\mathbf{cond}\ (x\text{-}1)\ x\ \top)$
$\mathbf{int}_3$:= $\lambda x.\mathbf{cond}\ x\ 0\ (\mathbf{cond}\ (x\text{-}1)\ 1\ (\mathbf{cond}\ (x\text{-}2)\ x\ \top))$
...
$\mathbf{int}_{n+1}$:= $\lambda x.\mathbf{cond}\ x\ 0\ (\mathbf{cond}\ (x\text{-}1)\ 1\ (\ \ldots\ (\mathbf{cond}\ (x\text{-}n)\ \ x\ \top)\ \ldots)$

Clearly, in the preceeding definition,"**cond** (x-k) **k**" iterates "in depth", for k := 0,1,...,n and the "innermost rightmost" subterm is "**cond** (x-n) x $\top$ ".

24.1.2. Lemma.

(1) $\bot, \top$ **E** $\mathbf{int}_0$

(2) For all $n \in \mathbb{N}$,

$\bot, \mathbf{0}, \ldots, \mathbf{n}, \top$ **E** $\mathbf{int}_{n+1}$.

(3) There are no more fixed points of $\mathbf{int}_n$ ($n \in \mathbb{N}$), except as indicated.

Proof. Straightforward, from the properties of **cond**. □

24.1.3. Theorem. For all $n \in \mathbb{N}$, $\mathbf{int}_n$ **E V**.

Proof. Easy inductions, using Lemma 24.1.2 and extensionality for closures. □

24.1.4. Remark. For every $n \in \mathbb{N}$, $\mathbf{int}_n$ is a strict closure, as already established in 24.1.2.

24.1.5. Definition.

(1) $\mathbf{case}_0 := \bot$

(2) $\mathbf{case}_1 := \lambda x y_0.\mathbf{cond}\ x\ y_0\ \bot$

(3) $\mathbf{case}_{n+1} := \lambda x y_0 \ldots y_n.\mathbf{cond}\ x\ y_0\ (\mathbf{case}_n\ (\mathbf{pred}\ x)\ y_1 \ldots y_n)$

(Note : subscripts are not projections here!)

24.1.6. Remark. Note that

$$\mathbf{case}_1 = \lambda x y.\mathbf{cond}\ x\ y\ (\mathbf{case}_0\ (\mathbf{pred}\ x))$$

(by extensionality for functions in Pω!).

24.1.7. Theorem (Finite Type Recursion). Let $n \in \mathbb{N}$ and assume that

$$\forall x\ \mathbf{E}\ \mathbf{int}_n.\ h_{[x]}\ \mathbf{E}\ \mathbf{V}$$

and

$$\bot\ \mathbf{E}\ h_{[x:=\bot]}$$
$$\top\ \mathbf{E}\ h_{[x:=\top]}$$

Then

(1) whenever c **E** $\mathbf{int}_n$ one has also

$$\mathbf{case}_0\ c\ \mathbf{E}\ h_{[x:=c]}, \text{ for } n = 0$$
$$\mathbf{case}_n\ c\ d_0 \ldots d_{n-1}\ \mathbf{E}\ h_{[x:=c]}, \text{ for } n \geq 1.$$

(2) If, moreover, $n \geq 1$ then, for all k, with $0 \leq k \leq n-1$,

$$\mathbf{case}_n\ \mathbf{k}\ d_0 \ldots d_{n-1} = d_k\ \mathbf{E}\ h_{[x:=\mathbf{k}]}.$$

Proof. Easy. □

24.1.8. Remark. For n = 0, Theorem **24.1.7** is a trivial statement. For n = 1, $\mathbf{int}_1$ is the unit type. In this case, if, for all x **E** $\mathbf{int}_1$, $h_{[x]}$ is a closure (i.e.,an "extensional family of types" over the unit type) and if c **E** $\mathbf{int}_1$ then $\mathbf{case}_1$ c acts as an identity and **E** $h_{[x:=c]}$. In particular,

$$\forall x \ \mathbf{E}\ h_{[x:=0]}\ \mathbf{E}\ \mathbf{V}.\ \mathbf{case}_1\ 0\ x = x,$$

whence, for $h_{[x:=0]}$, as above,

$$(\lambda x.\mathbf{case}_1\ 0\ x) \circ h_{[x:=0]} = h_{[x:=0]}.$$

The case n = 2 deserves a separate treatment.

24.1.9. Definition.

(1) **bool** := $\mathbf{int}_2$, **true** := 1, **false** := 0

(2) **if** x **then** a **else** b := $\mathbf{case}_2$ x a b

24.1.10. Remarks.

(1) **bool** **E** **V**

(2) $\bot$, **true**, **false**, $\top$ **E** **bool**

(3) Let $h_{[x]}$ be such that

$$\bot\ \mathbf{E}\ h_{[x:=\bot]}\ \mathbf{E}\ \mathbf{V}$$
$$d_0\ \mathbf{E}\ h_{[x:=\mathbf{true}]}\ \mathbf{E}\ \mathbf{V}$$
$$d_1\ \mathbf{E}\ h_{[x:=\mathbf{false}]}\ \mathbf{E}\ \mathbf{V}$$
$$\top\ \mathbf{E}\ h_{[x:=\top]}\ \mathbf{E}\ \mathbf{V}.$$

Then, whenever c **E** **bool**, one has

$$(\mathbf{if}\ c\ \mathbf{then}\ d_0\ \mathbf{else}\ d_1)\ \mathbf{E}\ h_{[x:=c]}$$

with, in particular,

$$(\mathbf{if\ true\ then}\ d_0\ \mathbf{else}\ d_1) = d_0$$
$$(\mathbf{if\ false\ then}\ d_0\ \mathbf{else}\ d_1) = d_1,$$

while, for "fictious" values given to c ($\bot$, $\top$), the conditional returns the corresponding "fictious" values. (Here, again, subscripts are not projections!)

24.1.11. Remark. Theorem **24.1.7** holds, a fortiori, for $h_{[x]}$ such that (for all n ε $\mathbb{N}$),

$$\forall x\ \mathbf{E}\ \mathbf{int}_n.\ \bot, \top\ \mathbf{E}\ h_{[x]}\ \mathbf{E}\ \mathbf{V}.$$

24.2. The integer type as a closure.

We may "collect" the Pω-integers into a closure, as in [Scott 76] :

24.2.1. Definition.

(1) **int** := **fix** (λfx.**cond** x **0** (**cond** (f(**pred** x)) x x))

(2) **0** := {0}, **n** := $\mathbf{succ}^{n}$ **0**,

iterating as usual, (n > 0).

24.2.2. Remark. For all $x \in P\omega$,

$$\mathbf{int}\, x = (x \mapsto \mathbf{0},\ (\mathbf{int}\ (x-1) \mapsto x, x))$$

24.2.3. Lemma ([Scott 76]). For all $x \in P\omega$, $(n \in \mathbb{N})$,

$$\mathbf{int}\ x = \begin{cases} \bot & \text{if } x = \bot \\ \mathbf{n} & \text{if } x = \mathbf{n} \\ \top & \text{else} \end{cases}$$

and, in particular, **int** $\top = \top$.

Proof. By Scott induction (on **fix**, $x \in P\omega$, if any). □

24.2.4. Theorem ([Scott 76]). **int E V**, and, finally, **int** is a strict closure.

Proof. Easy case-inspection, using Lemma **24.2.3**. □

24.2.5. Remark. Thus the "official" integers of Pω are adopted here qua formal version of the naturals in LAMBDA. One can also define, in the same spirit of economy, the successor and predecessor as restricted to **int**.

A "test for zero" is quickly available :

24.2.6. Definition. **zero** := λx.**cond**(**int** x) **K K'**.

24.2.7. Lemma. For all $x \in P\omega$, $n \in \mathbb{N}$,

$$\mathbf{zero}\ x = \begin{cases} \bot & \text{if } x = \bot \\ \mathbf{K} & \text{if } x = \mathbf{0} \\ \mathbf{K'} & \text{if } x = \mathbf{n+1} \\ \top & \text{else} \end{cases}$$

We have thus an "adequate numeral system" in LAMBDA (see [Rezus 81], Appendix or [Barendregt 84], for representations of **nat** in pure type-free lambda-calculi); it should be then clear that one can "do" at least (primitive) recursion on **int** (accurately : $\mathbf{int}^{\circ}$), preserving the familiar type-free lambda-calculus pattern. To show that the corresponding type-free techniques do actually work in the present (typed) setting, we give here the explicit definitions.

24.2.8. Definition.

$$\mathbf{rec} := \mathbf{fix}\ (\lambda rmde.\mathbf{cond}\ m\ d(e\ (\mathbf{pred}\ m)(r\ (\mathbf{pred}\ m)\ d\ e)).$$

24.2.9. Remark. So, for $m, d, e \in P\omega$,

$$\mathbf{rec}\ m\ d\ e = m \twoheadrightarrow d,e(m-1)(\mathbf{rec}\ (m-1)\ d\ e)$$

and the type-free behavior of **rec** is as expected, viz.,

24.2.10. Lemma (Primitive Recursion). For all $d, e \in P\omega$ and all $n \in \mathbb{N}$,

$$\mathbf{rec}\ \bot\ d\ e = \bot$$
$$\mathbf{rec}\ \mathbf{0}\ d\ e = d$$
$$\mathbf{rec}\ \mathbf{n+1}\ d\ e = e\ n\ (\mathbf{rec}\ \mathbf{n}\ d\ e)$$
$$\mathbf{rec}\ \top\ d\ e = \top.$$

Proof. Straightforward, from **24.2.8.** □

The corresponding "typed" version of **24.2.10** is now as follows :

24.2.11. Theorem (Primitive Recursion Theorem). Assume

$$\forall z\ \varepsilon\ \mathbf{int}.\ h_{[z]}\ \mathbf{E}\ \mathbf{V}$$
$$\bot\ \mathbf{E}\ h_{[z:=\bot]}$$
$$\top\ \mathbf{E}\ h_{[z:=\top]}$$
$$d\ \mathbf{E}\ h_{[z:=\mathbf{0}]}$$
$$\forall x\ \mathbf{E}\ \mathbf{int}.\ (\forall y\ \mathbf{E}\ h_{[x]}.\ e_{[x,y]}\ \mathbf{E}\ h_{[z:=\mathbf{succ}\ x]}).$$

Then one has the following conclusions :

(1) if $c\ \mathbf{E}\ \mathbf{int}$ then

$$\mathbf{rec}\ c\ d\ (\lambda x{:}\mathbf{int}.\lambda y{:}h_{[x]}.e_{[x,y]})\ \mathbf{E}\ h_{[x:=c]}.$$

Specifically,

$$\mathbf{rec}\ 0\ d\ (\lambda x{:}\mathbf{int}.\lambda y{:}h_{[x]}.e_{[x,y]})\ \mathbf{E}\ h_{[x:=0]}$$

(2) and, whenever $c\ \mathbf{E}\ \mathbf{int}$,

$$\mathbf{rec}\ (\mathbf{succ}\ c)\ d\ (\lambda x{:}\mathbf{int}.\lambda y{:}h_{[x]}.e_{[x,y]} = e'$$

where

$$e' = e_{[x:=c,\ y:=\mathbf{rec}\ c\ d\ (\lambda x:\mathbf{int}.\lambda y:h_{[x]}.e_{[x,y]})]}.$$

Proof. Use Lemma 24.2.10. □

24.2.11. Remark. It is easy to see that the preceeding Theorem holds also under the following hypothesis, (replacing the first three, stated earlier) :

$$\forall x\ \ \mathbf{E}\ \ \mathbf{int}.\ \bot, \top\ \mathbf{E}\ h_{[x]}\ \mathbf{E}\ \mathbf{V}.$$

In this form, one has a "strict" typed version of the Recursion Theorem for the "official" integers of $P\omega$.

3. MARTIN-LOF'S TYPE THEORY: SYNTAX AND SEMANTICS.

31. Constructive Type Theory as a polymorphic typed language.

If considered as a formal theory, a type(d) system admits of a uniform syntactic description, according to a general plan based on a few "top-down" design principles. The method generates a class of polymorphic typed languages or generalized typed lambda-calculi (ptl's, for short) and has been first applied, in the form described here, to the description of the AUT(OMATH)-family of languages ([Rezus 83], cf. also [Barendregt & Rezus 83]). Here "syntax" is taken to mean grammar + proof theory, the grammar being a parsing module (an abstract parser), concerned exclusively with the well-formedness of various forms of expressions admitted in the language , while the "proof"-part is meant to supply formal grounds for the correct manipulations of such expressions in the intended typed context. In particular, any one of Martin-Löf's systems formalizing Constructive Type Theory is a polymorphic typed language (on a syntactic level), viz., it can be generated along the proposed scheme. $\mathbf{CST}_1$, Martin-Löf's Constructive Type Theory with one Universe, has been first described in this way in [Rezus 83a]. This section gives a short account of the method.

31.1. Syntactic structures.

A polymorphic typed language L can be generated by a two-level syntax which will be conveniently described first in an abstract setting.

(1) Parsing. If L is a ptl, the first level of its syntax gives, roughly, a parser (a "correctness-free" grammar, the "well-formedness" description stage) and consists of displaying the alphabet A^L of L and its syntactic categories (where A^L is countable and each syntactic category of L is a recursive set of words over A^L). It turns out that, in all interesting cases, the "pure" part of a ptl can be already generated with only three distinct syntactic categories: L-terms, L-formulas (or L-sentences) and L-contexts.

Full specification at this description stage amounts, essentially, to the specification of the particular abstraction mechanism (viz. the structural behavior of the "abstraction forms") of L, underlying the construction of L-terms (see below).

(2) Type-inference and proof-theory. The second stage (level) of syntactic description specifies the "correctness grammar" (or the "correctness categories") of L. This is achieved by displaying a recursive relation $|--^L$ on the ("correctness free") syntactic categories of L, called L-entailment (L-type-assignment, or type-inference relation for L). The L-entailment relation $|--^L$ is generated by a recursive set of "correctness rules"; their rôle is to single out those syntactic units of L whose components admit of a "correct typing". Hence the proof-theoretic aspect of this description level. Actually, most of the known ptl's, among which Martin-Löf's type theories are, essentially formal systems for the correct manipulation of (constructive) data-types. Thus logic - first-order etc. - is not necessarily a part of a ptl. Rather, a specific form of logic, viz., that presupposed by the development of constructive mathematics [Bishop 67], [Martin-Löf 82,84], may be seen to be available in sufficiently complex ptl's, under some preferred interpretation of its "data-forms". In general, one is not committed to such interpretations. (The set of additional assumptions that are necessary in order to interpret $\mathbf{CST}_1$ say as a first-order intuitionistic logic is part

of a special purpose pragmatics for CST_1.). Last, an important distinguo: if L is interpreted as a programming language, $|--^L$ gives a proof-system for L and determines, up to a certain point, its operational semantics.

31.2. Abstract syntax.

In detail, the abstract syntax of a ptl L is as follows:

(1) **Parser** ("correctness-free" grammar):

(11) **Alphabet:** A^L

(11.1) **identifiers** ; these are variables (a denumerably infinite set $Var = \{v_n : n \in \mathbb{N}\}$, ranged over by x, y, z, ... possibly with sub and/or superscript decorations) or constants (to be specified as required, but see 31.3.3);

(11.2) **abstraction forms** (or abstractors): a denumerable non-empty set Abs^L, ranged over by A, to be specified, for each L, separately (the abstraction forms of CST_1 will be listed in 32);

(11.3) **predicate symbols** (or relation symbols): at least three predicate symbols whose intended meaning and associated arity are as follows:

T	"... is a type"	1
E	"... has type ..."	2
Q	"... equals ..."	2;

(11.4) **auxiliary symbols** ("punctuation"): any appropriate list; in general, its choice depends on the size and complexity of Abs^L and its elements are supposed to facilitate an unambiguous parsing of L-terms; our preferred list contains parentheses, brackets, a colon (possibly a semi-colon), a comma and a dot.

(12) **Syntactic categories** : sets of words over A^L

(12.1) $Term^L$ (L-terms, or terms generated by Abs^L), ranged over by a,b,c,d,e,f,g,h,... possibly with sub- and/or superscript decorations (for an explicit abstract construction, see 31.1);

(12.2) Form ^{L} (L-sentences of L-formulas), ranged over by lower-case Greek letters, possibly with decorations; in general, Form ^{L} admits of the following sub-categorization, respecting the primitive predicate symbols of L:

(12.21) Form_{T}^{L} (T-sentences or T-formulas in L): words of the form Ta (a ε Term ^{L}); intended meaning: "a is a type in L"; notation: a :: type;

(12.22) Form_{E}^{L} (E-sentences or E-formulas in L): words of the form Eab (a,b ε Term^{L}); intended meaning: "a has type b in L"; notation: a : b;

(12.23) Form_{Q}^{L} (Q-formulas in L): words of the form Qab (a,b ε Term^{L}); intended meaning: "a equals b in L"; notation: a = b.

Note. If a,b ε Term ^{L} and ϕ: = Ta or ϕ:= Eab then ϕ is a typing L-sentence and a is the subject of ϕ.

(12.3) Ctx^{L} (L- contexts); words of the form

$$[x_1:a_1, \ldots, x_n:a_n] \qquad n \varepsilon \mathbb{N}$$

where the $[x_i:a_i]$, $(1 \leq i \leq n)$, are typing E-sentences whose subjects are pairwise distinct variables in Var. Here n is the length of the L-context.

Meta-notation: the L-contexts are ranged over by Δ, possibly with sub- and/or super-scripts. For n = 0, the empty L-context is denoted by []. Intended meaning: an L-context declares a finite sequence of variables and specifies their types; it is thus an assumption list, viz. L-contexts are used only as hypotheses of statements (in proof).

Note. Condition 12.3 requires, in fact, minimal consistency for typing: a variable cannot have a priori syntactically distinct types.

Notation. To help intuition and readability the following notation is recommended (and will be adopted):

a = b :: type stands for (Ta) & (Tb) & (Qab),
read : "a and b are equal types",
a = b : c stands for (Eac) & (Ebc) & (Qab),
read : "a and b are equal objects of type c",
a : b :: type stands for (Eab) & (Tb),
a : b : c stands for (Eab) & (Ebc),

(where "&" is the meta-linguistic "and").

Accurately, the resulting "syntactic categories" would be "derived" syntactic categories of L. The former two, together with what we called above "L-formulas" correspond to Martin-Löf's "forms of judgment".

Remark. The derived forms may also occur as primitive notation in the description of the "correctness grammar" of L. (This policy has a philosophical justification in [Martin-Löf 75,75a,80,82,84].)

(2) Type-inference (proof theory):

A recursive relation $|--^L \subseteq Ctx^L \times Sent^L$, called L-entailment or L-type-inference relation is specified by a r. e. list of correctness rules for L. The L-entailment relation generates the correctness categories of L.

Notation. Where $\Delta \in Ctx^L$, $\phi \in Form^L$,

$$\Delta \ |--^L \phi$$

stands for $\langle \Delta, \phi \rangle \in |--^L$ and reads: "the L-context Δ yields the L-sentence ϕ in L" or "the L-sentence ϕ is L-correct relative to Δ", with lexical variants easy to identify. In principle, the L-entailment can be specified such as to generate the L-correctness categories by simultaneous induction; this amounts to the specification of the correctness predicate for L.

Remarks. The L-correctness categories are, according to this generation scheme, either primitive or derived: the primitive correctness categories of L are subsets of $|--^L$, while the derived (correctness) categories are constructed by explicit definition from the primitive ones (they are not subsets of the L-entailment relation). For details see mutatis mutandis, [Rezus 83].

Note. This method is applicable to polymorphic typed languages sharing the following features:

(1) the Q-predicate "...equals ..." is reflexive in L-correct contents.

(2) L contains at least some designated abstraction form U ε Abs^L, called universe symbol; intended meaning: the universe symbol U denotes the type of all (small) types. For syntactic purposes, it is immaterial whether U itself is supposed to stand for a "small" type or for a "large" one: the distinguo may be even not intended. In order to generate "correct contexts" say it is enough to observe that U itself is a privileged abstraction form in L and that any correct context in L should yield some closed L-sentence involving U, e.g. something like U = U or U :: type.

31.3. Term syntax

The syntactic category $Term^L$ of L-terms can be also described in abstracto, for arbitrary ptl's L. What is more interesting in this approach is the fact that - once the abstract setting is fixed - the actual specification of $Term^L$, for given L, depends only on the specification of the "abstraction mechanism" of L, i. e., on its abstraction forms. The definition scheme displayed here relies on suggestions from Peter Aczel.

We need first some auxiliary concepts. Hereafter, L is any ptl.

31.3.1 Definition. A signature ("complete arity") is a finite, possibly empty, sequence $\langle k_1, \ldots, k_n \rangle$, $(n \geq 0)$, of non-negative integers. Here n is the length of the signature.

31.3.2 Convention. Each abstraction form A in Abs^L is supposed to have a (unique) associated signature.

31.3.3 Definition. In particular, an atomic constant (L-constant) is an abstractor A whose signature has length 0.

We assume conveniently that there is a unique signature of length 0, denoted by $\langle\ \rangle$.

31.3.4 Definition. An abstraction form A is improper (simple) if its signature is of the form <0,...,0>, (n times "0", n > 0), otherwise it is proper.

As mentioned earlier, a given set Abs^L of abstraction forms generates the recursive set $Term^L$ and, according to the structural properties of the L-terms they generate, the abstraction forms in Abs^L admit of the following sub-categorization:

(1) constructors (or "data-forms"), which are type-constructors or ob(ject)-constructors and

(2) selectors ("program forms"). The first sub-category is supposed to cooperate (exclusively) to the construction of "canonical L-terms" ("L-data", i. e., L-terms which do not admit of any further "external" evaluation). These can be either L- data-types or ob(ject)s inhabiting L- data-types . The L-data need not be always irreducible (so to speak, "in normal form"). That is : particular L-data may still require some form of evaluation; the latter, if possible at all, must not be of the "outermost" kind. In other words, L-data are always subject to zero or more "internal evaluation steps", but, by definition, they do not admit of "external" evaluation (head-reduction, say). A useful consequence of this distinguo consists of the fact that L-data can always be recognized/identified, among other L-constructs, by L-data-forms, i. e., by a mere inspection of their (head)-constructors.

The second sub-category (the "selectors") contribute to the construction of the so-called non-canonical L-terms ("evaluable L-programs", L-terms which do require "outermost" evaluation) and, thereby, to the specification of the forms of evaluation admissible for L.

In particular, once Abs^L is given ("fully specified"), one can define inductively $Term^L$, the set of L-terms, as the Aczel-closure of Var in Abs^L.

31.1.5. Definition (scheme).

(1) Any variable in Var is an L-term.

(2) If $\underline{A} \in Abs^L$, with signature $< k_1,\ldots,k_n >$, $(n \geq 0,\ k_i \in \mathbb{N})$ then

$$\underline{A}([\mathbf{x}_1 : a_1]b_1;\ \ldots\ ;\ [\mathbf{x}_n : a_n]b_n) \qquad (*)$$

is an L-term, provided that, for $1 \leq i \leq n$,

$$\mathbf{x}_i := x_{i,1}, \ldots, x_{i,k_i}$$

are sequences of pairwise distinct variables in Var,

$$\mathbf{a}_i := a_{i,1}, \ldots, a_{i,k_i}$$

are sequences of L-terms and, moreover, $b_1, \ldots, b_n$ are L-terms

31.3.6. Terminology. Given an L-term a which is not an L-atom (i.e., a variable or an atomic constant, with arity < >), it is always of the form (*) above and, for $1 \leq i \leq n$,

(1) b_i is its i-th body,

(2) the i-th abstraction prefix of a is

$$[x_{i,1} : a_{i,1}, \ldots, x_{i,k_i} : a_{i,k_i}] := [\mathbf{x}_i : \mathbf{a}_i]$$

and has k_i domain parts $a_{i,j}$ $(1 \leq j \leq k_i)$, while, finally, $[\mathbf{x}_i : \mathbf{a}_i]b_i$ is the i-th immediate subterm of a.

Free and bound variables in an L-term, subterms of an L-term are to be defined in the expected way : occurrences of a variable at place $j \leq k_i$ in a sequence $\mathbf{x}_i$ that are free in the i-th body b_i and in any domain part

$$a_{i,j+1}, \ldots, a_{i,k_i}$$

of a become bound in a and so on. Similarly for closed and open L-terms.

Simultaneous substitution is denoted by

$$b_{[\mathbf{x}:=\mathbf{a}]},$$

where b, $\mathbf{a} := a_1, \ldots, a_n$ are L-terms and $\mathbf{x} := x_1, \ldots, x_n$ are pairwise distinct variables.

The alphabetic variants of L-terms are identified, by "alpha-conversion";

32. An abstract parser for CST_1-terms.

Once we think of CST_1 as being a polymorphic typed language (in the technical sense just described), it is sufficient to specify the set of CST_1-abstraction forms in order to get a full specification of the CST_1-parser. The abstract generation pattern described in 31 will then take care of the remaining ingredients. In particular, the structure of the set of abstraction forms of CST_1 is quite regular (this is motivated theoretically in [Martin-Löf 75a,80,82]) : for each type-constructor in the set, there are zero or more ob-constructors and, with one exception, exactly one selector.

32.1. CST_1 : the abstraction mechanism.

The alphabet of CST_1 is as follows :

I Variables : a set Var = $\{v_n : n \in \mathbb{N}\}$

II Abstraction forms (see table)

III Predicate symbols: T, E, Q (see 31.2).

IV Auxiliary symbols: ( ) [ ] , :

II Abstraction forms

	Constructors ("data-forms")				Selectors ("program forms")	
	Type-constructors		Ob-constructors			
0	U	< >				
1	Π	< 1 >	λ	< 1 >	F	< 0, 0 >
2	Σ	< 1 >	p	< 0, 0 >	E	< 0, 2 >
3	+	< 0, 0 >	i	< 0 >	D	< 0, 1, 1 >
			j	< 0 >		
4	N	< >	0	< >	R	< 0, 0, 2 >
			s	< 0 >		
5_0	N_0	< >			R_0	< >
5_m	N_m	< >	0_m	< >	R_m	< 0, ..., 0 >
			...			
			$m-1_m$	< >		

Note. The corresponding signatures are specified on the r.h.s. of each form.

Remark. The following meta-notation for the abstraction forms of CST_1 anticipates the semantics discussed below; it may also serve as mnemonics :

univ	(U)				
gen	(Π)	fun	(λ)	app	(F)
spec	(Σ)	pair	(p)	split	(E)
sum	(+)	inl,inr	(i,j)	when	(D)
int	(N)	0,succ	(0,s)	rec	(R)
int_0	(N_0)			$case_0$	(R_0)
int_m	(N_m)	k_m	(k_m)	$case_m$	(R_m)

32.2. CST_1 : the terms.

The CST_1-terms are defined recursively by : if a, b, c, d, d_m, e, f, g, h are CST_1-terms ($m \geq 1$) then so are the following ones,

(0)	U	can (type)	
(11)	Π([x:f]g)	can type	Πx:f.g
(12)	λ([x:f]a)	can ob	λx:f.a
(13)	F(c,d)	non-can eval	(cd)
(21)	Σ([x:f]g)	can type	Σx:f.g
(22)	p(a,b)	can ob	< a, b >
(23)	E(c,[x:f][y:g]d)	non-can eval	
(31)	+(f,g)	can type	f + g
(321)	i(a)	can ob	
(322)	j(b)	can ob	
(33)	D(c,[x:f]d,[y:g]e)	non-can eval	
(41)	N	can type	
(421)	0	can ob	
(422)	s(a)	can ob	
(43)	R(c,d,[x:N][y:h]e)	non-can eval	
($5_0$1)	N_0	can type	
($5_0$3)	R_0(c)	non-can eval	
(5_m1)	N_m	can type	
(5_m2)	$0_m,\ldots,m-1_m$	can ob	
(5_m3)	$R_m(c,d_0,\ldots,d_{m-1})$	non-can eval	

In the above, a practical notation ("syntactic sugar") is displayed, whenever applicable, on the r.h.s. of each (kind of) term, while the middle column gives the corresponding syntactic sub-category : canonical type term, canonical ob(ject)-term or non-canonical term ("form of evaluation"). The tree-like classification occurring on the l.h.s. matches the table of CST_1-abstraction forms from **32.1.**

Remarks. The type-information occurring in ob-terms (specifically, in (12),(23),(33) and (43) above) is erased in Martin-Löf's presentations of **CST.** The syntax adopted here has been appropriately disambiguated in order to facilitate the model-theoretical approach. As is easily seen, it agrees,

essentially, with the syntax of abstract AUTOMATH ([Rezus 83],[Barendregt & Rezus 83]). (The concrete syntax of the languages in the AUT-family is, actually, more involved; see [de Bruijn 80] and the bibliography listed there for details.)

If $\mathbf{CST}_1$ is considered as a programming language then the definition above gives only the core of the internal syntax of the language, as seen by a possible interpreter. In practice, one has to supply a more flexible notation for constructs which are supposed to be externally visible (to a potential user say) : there is a large spectrum of notational variants capable of expressing (equivalently) the abstract parsing scheme described here, but we cannot afford the space for a discussion (consult, however, [Reynolds 84], [Mitchell 84], [McCracken 79,83], [Nordström 81], [Petersson 82], [Cardelli & MacQueen 83] for closely related proposals).

33. Correctness rules for $\mathbf{CST}_1$.

The correctness rules of $\mathbf{CST}_1$ are classified as follows:

(1) structural rules,

(2) type assignment and evaluation rules,

(3) equivalence rules and

(4) congruence rules.

Remarks. For each type-constructor A of $\mathbf{CST}_1$, a "type-assignment rule" may appear as:

- a rule of A-formation (A-i)
- a rule of A-abstraction (A-abs)
- a rule of A-application (A-app),

(the latter situation may occur only if the type-constructor A has an associated selector; here, in all cases, except for U). One can also add appropriate rules of A-analysis ("elimination"), (A-e), as opposed to A-formation, for any type-constructor A of $\mathbf{CST}_1$, provided A is not a $\mathbf{CST}_1$-constant (i.e., its signature is not $< >$; here, in all cases, except for U,N and the N_m's). Such extensions will be ignored, since they are not considered by Martin-Löf (in fact, these rules must be "admissible" in the system, in the sense they do not increase the stock of "provable entailments" of $\mathbf{CST}_1$). Note that each type-constructor A, which has an associated selector, requires

corresponding "evaluation" rules (A-eval), (here : all except U). As it stands, the list of rules following below is somewhat redundant.

Nota bene. Martin-Löf has "introduction" and "elimination" for our "abstraction" and "application" rules, resp. Also, in his own formulations [Martin-Löf 82,84], he used a (somewhat elliptic) "natural deduction"-style presentation of the rules; e.g., contexts ("assumption lists") were displayed only as necessary, while "derivability" (in our terms : occurrences of the entailment symbol) is indicated by suggestive notational expedients. Except for such details, our formulations agree rigorously with Martin-Löf's presentation. It is a trivial exercise to transform the present formalism into a formal version of Aczel's explanation of the rules ([Aczel 81], [Beeson 85], XI.14).

Notation. (See also 31.2.)

(1) If $\Delta := [x_1:f_1, \ldots, x_n:f_1]$ then $\Delta[x:f] := [x_1:f_1, \ldots, x_1:f_1, x:f]$

(2) $\Delta \mid\!\!-\!\!-\ a$ is shorthand for $\Delta \mid\!\!-\!\!-\ a = a$

(3) $\Delta \mid\!\!-\!\!-\ a, b$ is shorthand for $(\Delta \mid\!\!-\!\!-\ a) \ \&\ (\Delta \mid\!\!-\!\!-\ b)$

The abbreviations adopted in 31.2 for Martin-Löf's "forms of judgment" are always used in particular contexts. That is, for example :

$$\Delta \mid\!\!-\!\!-\ a : b :: type$$

is shorthand for

$$(\Delta \mid\!\!-\!\!-\ a : b) \ \&\ (\Delta \mid\!\!-\!\!-\ b :: type)$$

etc. Here "&" is meta-linguistic "and", as expected. Rigorously, a rule with n occurrences of "&" in the conclusion ($n \geq 1$) is a bundle of n distinct rules with a single conclusion!

33.1. Structural rules.

Note. Not all these rules appear in Martin-Löf's own presentation of **CST**, but they are, obviously, implicit in assumptions concerning the manipulation of contexts, substitution, etc. (Although contexts are mentioned explicitly, they are left unformalized in [Martin-Löf 82,84].). The meaning of the first six rules, appearing here before (sub-E), is "packed into" the so-called "assumption rule" of [Martin-Löf 82,84], which is, actually, an informal scheme. Here, (sub-E) and (sub-Q) have, each, n premisses of the form

$$\Delta' \text{ |-- } \phi_i \qquad (1 \leq i \leq n)$$

where the ϕ_i's are as indicated. Accurately, these are rules intended to allow the "change of the current context", (that is, passing from Δ of the form below to a suitably chosen "correct" Δ').

Let $\Delta := [x_1:f_1, \ldots, x_n:f_n]$, everywhere. Then x is <u>fresh for</u> Δ if $x \neq x_i$ and x does not occur in f_i, $1 \leq i \leq n$.

Initialization :

(r-U) $\qquad [] \text{ |-- } U$

Context recursion : if x is fresh for Δ then

(c-1) $\quad \Delta \text{ |-- } U \quad ===> \quad \Delta[x:U] \text{ |-- } U$

(c-2) $\quad \Delta \text{ |-- } a :: type \quad ===> \quad \Delta[x:a] \text{ |-- } U$

Context projection ($n \geq 1$, $1 \leq i \leq n$) :

(cp) $\quad \Delta \text{ |-- } U \quad ===> \quad \Delta \text{ |-- } x_i : f_i :: type$

Sentence projection :

(r-s-E) $\quad \Delta \text{ |-- } a : b :: type \quad ===> \quad \Delta \text{ |-- } a$

(r-p-E) $\quad \Delta \text{ |-- } a : b :: type \quad ===> \quad \Delta \text{ |-- } b$

(r-Q) $\quad \Delta \text{ |-- } a = b \quad ===> \quad \Delta \text{ |-- } a, b$

Substitution ($n \geq 1$); $\mathbf{x} := x_1, \ldots, x_n$; $\mathbf{a} := a_1, \ldots, a_n$.

(sub-E) :

$\Delta \text{ |-- } g : h :: type$; $\Delta' \text{ |-- } a_i : f_{i[\mathbf{x}:=\mathbf{a}]} ===> \Delta' \text{ |-- } g_{[\mathbf{x}:=\mathbf{a}]} : h_{[\mathbf{x}:=\mathbf{a}]} :: type$

(sub-Q) :

$\Delta \text{ |-- } g = h$; $\Delta' \text{ |-- } a_i : f_{i[\mathbf{x}:=\mathbf{a}]} ===> \Delta' \text{ |-- } g_{[\mathbf{x}:=\mathbf{a}]} = h_{[\mathbf{x}:=\mathbf{a}]}$

33.2. Type-assignment and evaluation rules.

Rules for the Universe :

U-formation (U-i) :

$$[] \text{ |-- } U :: type$$

U-analysis (U-e) :

$$\Delta \text{ |-- } f : U \quad ===> \quad \Delta \text{ |-- } f :: type$$

U-Π-abstraction (U-Π-abs) :

$$\Delta \text{ |-- } f : U;\ \Delta[x:f] \text{ |-- } g_{[x]} : U \ ===> \ \Delta \text{ |-- } \Pi x:f.g_{[x]} : U$$

U-Σ-abstraction (U-Σ-abs) :

$$\Delta \text{ |-- } f : U;\ \Delta[x:f] \text{ |-- } g_{[x]} : U \ ===> \ \Delta \text{ |-- } \Sigma x:f.g_{[x]} : U$$

U-+-abstraction (U-+-abs) :

$$\Delta \text{ |-- } f : U;\ \Delta \text{ |-- } g : U \ ===> \ \Delta \text{ |-- } (f + g) : U$$

U-N-abstraction (U-N-abs) :

$$[] \text{ |-- } N : U$$

U-N_m-abstraction (U-N_m-abs), ($m \geq 0$) :

$$[] \text{ |-- } N_m : U$$

Rules for generalized products :

Π-formation (Π-i) :

$$\Delta \text{ |-- } f :: type;\ \Delta[x:f] \text{ |-- } g_{[x]} :: type \ ===> \ \Delta \text{ |-- } \Pi x:f.g_{[x]} :: type$$

Π-abstraction (Π-abs) :

$$\Delta \text{ |-- } f :: type;\ \Delta[x:f] \text{ |-- } a_{[x]} : g_{[x]} :: type \ ===> \ \Delta \text{ |-- } \lambda x:f.a_{[x]} : \Pi x:f.g_{[x]}$$

Π-application (Π-app) :

$$\Delta \text{ |-- } d : f :: type;\ \Delta \text{ |-- } c : \Pi x:f.g_{[x]} :: type \ ===> \ \Delta \text{ |-- } cd : g_{[x:=d]}$$

Π-evaluation (Π-eval-β) :

$$\Delta \text{ |-- } d : f :: type;\ \Delta' \text{ |-- } a_{[x]} : g_{[x]} :: type \ ===> \ \Delta \text{ |-- } (\lambda x:f.a_{[x]})d = a'$$

where $\Delta' := \Delta[x:f]$ and $a' := a_{[x:=d]} : g_{[x:=d]}$

Π-evaluation (Π-eval-η); if x is not free in c then

$$\Delta \text{ |-- } c : \Pi x:f.g_{[x]} :: type \ ===> \ \Delta \text{ |-- } \lambda x:f.cx = c$$

Rules for generalized sums :

Σ-formation (Σ-i) :

Δ |-- f :: type; $\Delta[x:f]$ |-- $g_{[x]}$ :: type ===> Δ |-- $\Sigma x:f.g_{[x]}$ :: type

Σ-abstraction (Σ-abs) :

$$\frac{\begin{array}{r}\Delta \text{ |-- } a : f :: \text{type} \\ \Delta[x:f] \text{ |-- } g_{[x]} :: \text{type} \\ \Delta \text{ |-- } b : g_{[x:=a]}\end{array}}{\Delta \text{ |-- } \langle a,b \rangle : \Sigma x:f.g_{[x]}}$$

Σ-application (Σ-app) :

$$\frac{\begin{array}{rll}\Delta \text{ |-- } f :: \text{type} & & (1) \\ \Delta[x:f] \text{ |-- } g_{[x]} :: \text{type} & & (2) \\ \Delta[z : \Sigma x:f.g_{[x]}] \text{ |-- } h_{[z]} :: \text{type} & & (3) \\ \Delta[x:f][y:g_{[x]}] \text{ |-- } d_{[x,y]} : h_{[z:=\langle x,y\rangle]} & & (4) \\ \Delta \text{ |-- } c : \Sigma x:f.g_{[x]} & &\end{array}}{\Delta \text{ |-- } E(c,[x:f][y:g_{[x]}]d_{[x,y]}) : h_{[z:=c]}}$$

Σ-evaluation (Σ-eval) :

$$\frac{\begin{array}{rl}(1), (2), (3), (4) & \text{(as above)} \\ \Delta \text{ |-- } a : f & \\ \Delta \text{ |-- } b : g_{[x:=a]} &\end{array}}{\Delta \text{ |-- } E(\langle a,b \rangle,[x:f][y:g_{[x]}]d_{[x,y]}) = d_{[x:=a,y:=b]} : h_{[z:=\langle a,b\rangle]}}$$

Rules for disjoint unions :

+-formation (+-i) :

Δ |-- f :: type; Δ |-- g :: type ===> Δ |-- (f + g) :: type.

+-i-abstraction (+-i-abs) :

Δ |-- a : f :: type; Δ |-- b : g :: type ===> Δ |-- i(a) : (f + g)

+-j-abstraction (+-j-abs) :

Δ |-- a : f :: type; Δ |-- b : g :: type ===> Δ |-- j(b) : (f + g)

+-application (+-app) :

$$\begin{array}{rll} \Delta \text{ |-- } f :: \text{type} & & (1) \\ \Delta \text{ |-- } g :: \text{type} & & (2) \\ \Delta[z : (f+g)] \text{ |-- } h_{[z]} :: \text{type} & & (3) \\ \Delta[x:f] \text{ |-- } d_{[x]} : h_{[z:=i(x)]} & & (4) \\ \Delta[y:g] \text{ |-- } e_{[y]} : h_{[z:=j(y)]} & & (5) \\ \Delta \text{ |-- } c : (f + g) & & \end{array}$$

Δ |-- $D(c,[x:f]d_{[x]},[y:g]e_{[y]}) : h_{[x:=c]}$

+-i-evaluation (+-i-eval) :

(1), (2), (3), (4), (5) (as above)

Δ |-- a : f

Δ |-- $D(i(a),[x:f]d_{[x]},[y:g]e_{[y]}) = d_{[x:=a]} : h_{[z:=i(a)]}$

+-j-evaluation (+-j-eval) :

(1), (2), (3), (4), (5) (as above)

Δ |-- b : g

Δ |-- $D(j(b),[x:f]d_{[x]},[y:g]e_{[y]}) = e_{[y:=b]} : h_{[z:=j(b)]}$

Rules for natural numbers :

N-formation (N-i) :

[] |-- N :: type

N-0-abstraction (N-0-abs) :

[] |-- 0 : N

N-s-abstraction (N-s-abs) :

Δ |-- a : N ===> Δ |-- s(a) : N

N-application (N-app) :

$$\frac{\begin{array}{r}\Delta[z{:}N] \;|\!-\!-\; h_{[z]} :: \text{type} \quad (1)\\ \Delta \;|\!-\!-\; d : h_{[z:=0]} \quad (2)\\ \Delta[x{:}N][y{:}h_{[x]}] \;|\!-\!-\; e_{[x,y]} : h_{[z:=s(x)]} \quad (3)\\ \Delta \;|\!-\!-\; c : N\end{array}}{\Delta \;|\!-\!-\; R(c,d,[x{:}N][y{:}h_{[x]}]e_{[x,y]}) : h_{[z:=c]}}$$

N-0-evaluation (N-0-eval) :

$$\frac{(1),\ (2),\ (3) \qquad \text{(as above)}}{\Delta \;|\!-\!-\; R(0,d,[x{:}N][y{:}h_{[x]}]e_{[x,y]}) = d : h_{[z:=0]}}$$

N-s-evaluation (N-s-eval) :

$$\frac{\begin{array}{r}(1),\ (2),\ (3) \qquad \text{(as above)}\\ \Delta \;|\!-\!-\; a : N\end{array}}{\begin{array}{l}\Delta \;|\!-\!-\; R(s(a),\ d,\ [x{:}N][y{:}h_{[x]}]e_{[x,y]}) = e' : h_{[z:=s(a)]}\\ \text{where } e' = e_{[x:=a,y:=R(a,d,[x:N][y:h_{[x]}]e_{[x,y]})]}\end{array}}$$

Rules for finite types ($m \geq 0$) :

N_m-formation (N_m-i) :

$$[] \;|\!-\!-\; N_m :: \text{type}$$

N_m-abstraction (N_m-abs), for $0 \leq k \leq m-1$, if $m \geq 1$:

$$[] \;|\!-\!-\; k_m : N_m$$

N_m-application (N_m-app), $m \geq 1$:

$$\frac{\begin{array}{r}\Delta[z{:}N_m] \;|\!-\!-\; h_{[z]} :: \text{type}\\ \Delta \;|\!-\!-\; d_0 : h_{[z:=0_m]}\\ \ldots\\ \Delta \;|\!-\!-\; d_k : h_{[z:=k_m]}\\ \ldots\\ \Delta \;|\!-\!-\; d_{m-1}\ h_{[z:=m-1_m]}\\ \Delta \;|\!-\!-\; c : N_m\end{array}}{}$$

$$\Delta \;|\!-\!-\; R_m(c,\ d_0,\ \ldots, d_{m-1}) : h_{[z:=c]}$$

N_m-evaluation (N_m-eval), $m \geq 1$, $0 \leq k \leq m-1$:

$$\Delta[z{:}N_m] \;|\!-\!-\; h_{[z]} :: \text{type}$$
$$\Delta \;|\!-\!-\; d_0 : h_{[z:=0_m]}$$
$$\ldots$$
$$\Delta \;|\!-\!-\; d_k : h_{[z:=k_m]}$$
$$\ldots$$
$$\Delta \;|\!-\!-\; d_{m-1} : h_{[z:=m-1_m]}$$

$$\Delta \;|\!-\!-\; R_m(k_m,\ d_0,\ \ldots, d_{m-1}) = d_k : h_{[z:=k_m]}$$

Note. In absence of (Π-eval-η) one has a β-system. With (Π-eval-η) one has a Π-extensional system. Appropriate Σ- and +-extensionality (or, better, "functionality") rules may be also added (see [Rezus 85]). These are not derivable in the Π-extensional system, unless one assumes Martin-Löf's rules for "identity types".

33.3. Equivalence rules.

Reflexivity : see (r-U), (r-s-E), (r-p-E) and (r-Q) in the list of structural rules (**33.1**).

Symmetry :

(s-T) :

$$\Delta \;|\!-\!-\; f = g :: \text{type} \;\Longrightarrow\; \Delta \;|\!-\!-\; g = f :: \text{type}$$

(s) :

$$\Delta \;|\!-\!-\; f :: \text{type};\ \Delta \;|\!-\!-\; a = b : f \;\Longrightarrow\; \Delta \;|\!-\!-\; b = a : f$$

Transitivity :

(t-T) :

$$\Delta \;|\!-\!-\; f = g :: \text{type};\ \Delta \;|\!-\!-\; g = h :: \text{type} \;\Longrightarrow\; \Delta \;|\!-\!-\; f = h :: \text{type}$$

(t) :

$$\Delta \;|\!-\!-\; f :: \text{type};\ \Delta \;|\!-\!-\; a = b : f;\ \Delta \;|\!-\!-\; b = c : f \;\Longrightarrow\; \Delta \;|\!-\!-\; a = c : f$$

Type-conversion :

(eq-p) :

$$\Delta \;|\!-\!-\; f = g :: \text{type};\ \Delta \;|\!-\!-\; a : f \;\Longrightarrow\; \Delta \;|\!-\!-\; a : g$$

33.4. Congruence rules.

The congruence rules of CST_1 state, roughly speaking, the fact that the equality predicate (denoted by Q, here) is a congruence relative to the syntactic "operations" induced by the primitive abstraction forms of the language, i. e., functional application, functional abstraction, product formation, splitting - with E -, pairing, sum formation, etc.

Here are some typical examples:

(mon-λ) :

$$\frac{\Delta \;|\!-\!-\; f = g :: \text{type} \qquad \Delta[x:f] \;|\!-\!-\; a = b : h :: \text{type}}{\Delta \;|\!-\!-\; \lambda x{:}f.a = \lambda x{:}g.b \;:\; \Pi x{:}f.h}$$

(mon-Π) :

$$\frac{\Delta \;|\!-\!-\; f = g :: \text{type} \qquad \Delta[x:f] \;|\!-\!-\; h_{[x]} = h'_{[x]} :: \text{type}}{\Delta \;|\!-\!-\; \Pi x{:}f.h = \Pi x{:}g.h' :: \text{type}}$$

Once we have understood the behavior of contexts in the present setting, it should be clear how to obtain a complete list of congruence rules from Martin-Löf's analogous "natural deduction" presentation [Martin-Löf 82,84]. Writing them down in full is a tedious (although almost mechanical) task; we skip details, leaving them to the reader.

34. Strict closure semantics for CST_1.

We can now give a "fixed-point strict closure semantics" for CST_1 (cf. also [Barendregt & Rezus 83]). Given results obtained in 22.2 through 24.2 above, one has the following interpretation of CST_1-terms in Pω (via LAMBDA). In what follows, LAMBDA is used in the loose way, as earlier.

34.1. Definition.

(1) A valuation (in Pω) is a map

$$\rho : Var \to P\omega.$$

(2) The value (interpretation) of a CST_1-term f relative to /at ρ (in $P\omega$),(notation: $[\![f]\!]_\rho$) is defined by induction on the structure of f, as follows :

(210) $[\![x]\!]_\rho = \rho(x)$, for $x \in Var$

(220) $[\![U]\!]_\rho = \mathbf{V}$

(231) $[\![ab]\!]_\rho = ([\![a]\!]_\rho)([\![b]\!]_\rho) = \mathbf{fun}([\![a]\!]_\rho)([\![b]\!]_\rho)$

(232) $[\![\lambda x{:}a.b]\!]_\rho = (\mathbf{graph}(\lambda d.[\![b]\!]_{\rho(x:=d)})) \circ ([\![a]\!]_\rho)$

(233) $[\![\Pi x{:}a.b]\!]_\rho = \mathbf{gen}(\mathbf{graph}(\lambda d.[\![b]\!]_{\rho(x:=d)}))([\![a]\!]_\rho)$

(241) $[\![E(c,[x{:}f][y{:}g]d]\!]_\rho = \mathbf{split}([\![c]\!]_\rho)([\![h]\!]_\rho)$
where $h := \lambda x{:}f.\lambda y{:}g.d$

(242) $[\![< a,b >]\!]_\rho = \mathbf{pair}([\![a]\!]_\rho)([\![b]\!]_\rho)$

(243) $[\![\Sigma x{:}a.b]\!]_\rho = \mathbf{spec}(\mathbf{graph}(\lambda d.[\![b]\!]_{\rho(x:=d)}))([\![a]\!]_\rho)$

(251) $[\![D(c,[x{:}f]d,[y{:}g]e)]\!]_\rho = \mathbf{when}([\![c]\!]_\rho)([\![h']\!]_\rho)([\![h'']\!]_\rho)$
where $h' := \lambda x{:}f.d$ and $h'' := \lambda y{:}g.e$

(2521) $[\![i(a)]\!]_\rho = \mathbf{inl}([\![a]\!]_\rho)$

(2522) $[\![j(b)]\!]_\rho = \mathbf{inr}([\![b]\!]_\rho)$

(253) $[\![a + b]\!]_\rho = \mathbf{sum}([\![a]\!]_\rho)([\![b]\!]_\rho)$

(261) $[\![R(c,d,[x{:}N][y{:}h]e)]\!]_\rho = rec([\![c]\!]_\rho)([\![d]\!]_\rho)([\![h']\!]_\rho)$
where $h' := \lambda x{:}N.\lambda y{:}h.e$

(2621) $[\![0]\!]_\rho = \mathbf{0} = \{0\}$

(2622) $[\![s(a)]\!]_\rho = \mathbf{succ}([\![a]\!]_\rho)$

(263) $[\![N]\!]_\rho = \mathbf{int}$

$(271)_0$ $[\![R_0(c)]\!]_\rho = \mathbf{case}_0([\![c]\!]_\rho)$

$(271)_m$ $[\![R_m(c,d_0,\ldots,d_{m-1})]\!]_\rho =$
$= \mathbf{case}_m([\![c]\!]_\rho)([\![d_0]\!]_\rho)\ldots([\![d_{m-1}]\!]_\rho)$, for $m \geq 1$,

$(272)_m$ $[\![k_m]\!]_\rho = \mathbf{k} = \{k\}$, for $k \in \mathbb{N}$, $m \geq 0$

$(273)_m$ $[\![N_m]\!]_\rho = \mathbf{int}_m$, for $m \geq 0$.

Note. In the above, λ specifies a convenient meta-notation for (Scott continuous) functions in FUN (see, e. g., [Barendregt 84]), while, for $d \in P\omega$, $\rho(x:=d)$ "updates" ρ "at" $x \in Var$, as usual. One could have written (241), (251) and (261) explicitly, with **graph** as in (233) and (243); anyway, (232) takes care of the inserted "where"-clauses and this way of putting things somewhat helps reading the full list! Occurrences of **fun**, (which should have

appeared on the r.h.s. of =), unessential for the understanding of (2), have been tacitly omitted.

34.2. Definition. Let ρ be a valuation in $P\omega$.

(1) A T-sentence a :: type is *true at* ρ *in* $P\omega$ (notation: $\rho \models a :: type$) if

$$[\![a]\!]_\rho \ \mathbf{E}\ \mathbf{V}.$$

(21) Let now introduce the following "fixed-point predicate" over $P\omega$: for $\mathbf{a}, \mathbf{b} \in P\omega$,

$$\mathbf{a}\ \mathbf{E}_*\ \mathbf{b} \iff (\mathbf{a}\ \mathbf{E}\ \mathbf{b}\ \mathbf{E}\ \mathbf{V})\ \&\ \mathbf{dis(a,b)},$$

where

$$\mathbf{dis(a,b)} \iff \begin{cases} \bot, \top\ \mathbf{E}\ \mathbf{a}, & \text{if } \mathbf{b} = \mathbf{V} \\ \bot, \top\ \mathbf{E}\ \mathbf{b}, & \text{if } \mathbf{b} \neq \mathbf{V} \end{cases}$$

(thus $\mathbf{a}\ \mathbf{E}_*\ \mathbf{b}$ iff **a** is a fixed-point of a closure equal to **b** and either **a** is doubly strict, whenever it is a closure, or **b** is doubly strict). The main definition is now:

(22) an E-sentence a : b is *true at* ρ *in* $P\omega$ (notation: $\rho \models a : b$) if

$$[\![a]\!]_\rho \ \mathbf{E}_*\ [\![b]\!]_\rho$$

(3) A Q-sentence a = b is *true at* ρ *in* $P\omega$ (notation: $\rho \models a = b$) if

$$[\![a]\!]_\rho = [\![b]\!]_\rho$$

(41) If $\Delta = [x_1:a_1, \ldots, x_n:a_n]$ is a context, one defines

$$\rho \models \Delta \iff (\rho \models x_1:a_1)\ \&\ \ldots\ \&\ (\rho \models x_n:a_n)$$

where $n \geq 1$; (for $n = 0$, one can take $\rho \models []$ to be a true proposition).

(42) For all contexts Δ, as above, and all $\mathbf{CST}_1$-sentences ϕ, one defines

$$\rho \models^\Delta \phi \iff (\rho \models \Delta \Rightarrow \rho \models \phi)$$

(5) Finally, *validity* for $\mathbf{CST}_1$-sentences ϕ is defined by:

$$\models^\Delta \phi \iff \forall\rho.\ \rho \models^\Delta \phi.$$

The main result is now as follows :

34.3. Theorem (Soundness for $\mathbf{CST}_1$). Let ϕ be a (T-, E- or Q-) sentence of $\mathbf{CST}_1$. Then

$$\Delta \vdash \phi \Rightarrow \models^\Delta \phi.$$

Proof. Induction on the generation of $\vdash$ in $\mathbf{CST}_1$, using results obtained in section 2. □

34.4. Remarks. Define, as usual, [Martin-Löf 82,84],

$$a \dashrightarrow b \quad := \quad \Pi x{:}a.b$$

(in $\mathbf{CST}_1$), where x is not free in b. Then, using **E** instead of E_* in 34.2, (22), gives validity for

(U) $\quad [] \ |\!-\!-\ U : U$

and

(UU) $\quad [] \ |\!-\!-\ U : (U \dashrightarrow U)$

in the model, while (U) generates the well known Girard paradox when added to some elementary subsystem of $\mathbf{CST}_1$ (viz., pure Classical AUTOMATH; cf. [Girard 72], [Martin-Löf 72], [Barendregt & Rezus 83]).

It is, however, obvious that the present interpretation forbids both (U) and (UU) (as well as many other oddities derivable from the behavior of **V**). Indeed, in the case of (U), it is not true that $\mathbf{V}\ E_*\ \mathbf{V}$, since

$$\mathbf{V}\bot = \mathbf{I} \neq \bot,$$

while, for (UU), one can check easily that

$$(\mathbf{V} \to \mathbf{V})\bot = \mathbf{K}' \neq \bot$$

That is: **V** and $(\mathbf{V} \to \mathbf{V})$ are not doubly strict closures in $P\omega$.

Although apparently more general, the analogue fixed-point semantics which would use finitary retracts in ω-algebraic cpo's ([Scott 81,82,82a], cf. [McCracken 83]) in place of $P\omega$-closures offers no essential improvement over the present one; viz. the corresponding "space" of finitary retracts is already **E**-self-representable (as a finitary retract), sharing thus the inconvenients of CLOS (and **V**). (A possible gain - mainly, in elegance and, to some extent, technically - might consist of the absence of "fictious" top elements, that are rather boring in the lattice-theoretical approach.)

If, however, one is concerned only with the "first-order" fragment of $\mathbf{CST}_1$ (forgetting thus the "axiom"

$$[] \ |\!-\!-\ U :: \text{type}$$

and the U-rules), without the "ground types" N, N_m ($m \in \mathbb{N}$), then there is a closely related "idempotent splitting" map which carries the underlying type structure into appropriate type-free λ-theories, based on polyadic lambda-calculus $\mathbf{p\lambda\beta(\eta)}$ or on $\mathbf{\lambda\beta(\eta)SP}$ (lambda-calculus with surjective pairing). The corresponding type theory thus interpretable is, in fact, a fragment of a Generalized Typed Lambda Calculus, which extends the "pure" part of Zucker's

AUT-Pi system in the natural way. The details of this, more general, interpretation of the Constructive Type Theory as well as the category theoretical aspects therein involved are deferred and will be discussed elsewhere.

A philosophical aside: the strict closure semantics of $\mathbf{CST}_1$ is neither meant nor claimed to be intuitionistically acceptable. Indeed, the way we have introduced the basic "predicates" =, **E** and $\mathbf{E}_*$ is essentially non-constructive; our interpretation obtains "by way of anything which happens (to be true) in the model". It is then hardly surprising that the proposed semantics is unable to capture the meaning of Martin-Löf's identity types. In the original system(s), the latter are meant to allow the formalization of the concept of a constructive set; as mentioned in the Introduction, if taken in this sense, a set comes equipped with its own equality relation (also requiring a proof that this relation is an equivalence). In the present setting, one might certainly simulate model-theoretically a sort of Leibniz identity relation, that could be thereafter conveniently relativized to any type (closure). Set, e.g.,

$$\mathbf{Id} := \lambda f{:}\mathbf{V}.\lambda x{:}f.\lambda y{:}f.\Pi z{:}f \dashrightarrow \mathbf{V}.zx \dashrightarrow zy,$$

in LAMBDA. Then $\mathbf{Id}_f(a,b) := \mathbf{Id}\ f\ a\ b$ is a closure if so is f and if $a,b\ \mathbf{E}\ f$, and, moreover, $\mathbf{Id}_f(a,b)$ has "typical inhabitants" of the form

$$\mathbf{r}_f(a) := \lambda z{:}f \dashrightarrow \mathbf{V}.\lambda u{:}za.u,$$

provided a = b. It is immediate that the "identity types" thus defined do not respect "strictness" in the sense intended here, neither do they interpret Martin-Löf's "I-elimination rules" [Martin-Löf 80,82,84] faithfully (the corresponding fixed point sets should have been one-element algebraic lattices, looking like $\mathbf{int}_1$!). Without identity types (or "equality types"), $\mathbf{CST}_1$ is, in some sense, non-productive. It can certainly generate new types from old, given the primitives N and N_m ($m \in \mathbb{N}$). However, it is easy to establish that, in absence of the "equality types " I(f,a,b), one cannot use the full strength of the rules involving extensional families of types. In the latest formulations of the type theory ([Martin-Löf 80,82,84]), the identity types are the main source for extensional families in the system. (So, if one thinks of $\mathbf{CST}_1$ as a programming language, one has to insure that the user has the facility of generating families of types, under the régime of an appropriate intensional equality; for a possible solution see [Constable 82] or [Constable & Zlatin 84].)

The fact that the intuitionistic interpretation of the type-theoretical primitives is in conflict with counter-intuitive assumptions of the form (U) or (UU), as shown in [Girard 72], while the latter are also ruled out by the strict closure interpretation is only a happy accident. The present semantics makes all types inhabited (even "fictiously" so, with the Pω's bottom and top elements as only inhabitants, in the case of $\mathbf{int}_0$) and provides them with a complex internal structure: any type (inhabiting the Universe, a "small" type thus) is actually a rather special algebraic lattice (modulo °, see 22.1.1), viz. one that is already a Scott domain, where the "fictious" bottom and top elements of the domain coincide (under appropriate identifications) with the bottom and top of the full model (which is also a Scott domain, namely a "universal" one, cf. [Scott 82,82a]). As is well-known, there is already a standard way of "doing" computation theory in this setting [Scott 76,81,82]: ignoring all motivations, the approach remains classical in nature (as opposed to "constructivistic", in any, philosophically-motivated, plausible sense). We regard the classical interpretation(s) of the type-theoretical primitives as being - technically - more fundamental than the logical (say intuitionistic) approaches: the latter are also somewhat special purpose oriented and perhaps too philosophically committed. (See the "Discussions" appearing in [Beeson 85], for a - incomplete - survey of the possible philosophical standpoints to Martin-Löf's Constructive Type Theories, referred to there as $\mathbf{ML}_0$ and $\mathbf{ML}_1$.) A formal pragmatics of the type theory would probably offer a neutral frame to discuss such matters properly and on a rigorous mathematical basis, but, so far, such a meta-discipline lacks a good conceptual development and it is certainly premature to advance precise statements.

34.5. Problem. Let $\mathrm{CLOS}_* = \{ a \in P\omega : a \text{ is a doubly strict closure} \}$. Is there any $\mathbf{V}_* \in P\omega$ with

(1) $\mathbf{V}_* \ \mathbf{E} \ \mathbf{V}$,

(2) $a \in \mathrm{CLOS}_* \iff a \ \mathbf{E} \ \mathbf{V}_*$,

for all $a \in P\omega$, and preferably such that

(3) $\mathbf{V}_* \in \mathrm{CLOS}_*$

is not the case ? The question amounts to the E-representability of CLOS_* in Pω, possibly by non-E-reflexive elements of the model. Such a $\mathbf{V}_*$ would somewhat simplify the definition of satisfaction in 34.2 (allowing E in place

of $\mathbf{E}_*$ and being also more natural an interpretation for U, the Universe of "small" types, along the present semantics). However, answering negatively the question won't do any harm, leaving things as they are.

34.6. Remarks (The "context-closure" semantics). The approach taken in [Barendregt & Rezus 83] transfers, mutatis mutandis, to $\mathbf{CST}_1$, using results of section 2. This possibility has been obliquely noticed in that paper, but the authors suspected that a precise claim would eventually require too much detail and deferred a discussion. Indeed, define satisfaction for $\mathbf{CST}_1$-sentences (in $P\omega$) as in **34.2** above, changing first (22) into

(22c) an E-sentence a : b is true at ρ (in $P\omega$), (notation: $\rho \models\!\!= a : b$) if

$$[\![a]\!]_\rho \ \mathbf{E} \ [\![b]\!]_\rho$$

with, further, (41c), (42c), and (5c) for (41), (42), and (5) resp.:

(41c) if $\Delta = [x_1{:}a_1, \ldots, x_n{:}a_n]$ is a context ($n \geq 1$) and b is a term then let

$$\Delta_\lambda b := \lambda x_1{:}a_1 \ldots \lambda x_n{:}a_n.b$$
$$\Delta_\Pi b := \Pi x_1{:}a_1 \ldots \Pi x_n{:}a_n.b.$$

Such terms are called context closures. Then

(42c) for all contexts Δ, as above, and all $\mathbf{CST}_1$-sentences ϕ, one defines

(42.1c) $\rho \models\!\!=^\Delta a :: \text{type} \iff [\![\Delta_\lambda a]\!]_\rho \ \mathbf{E} \ [\![\Delta_\Pi U]\!]_\rho$
if ϕ is a :: type,

(42.2c) $\rho \models\!\!=^\Delta a : b \iff [\![\Delta_\lambda a]\!]_\rho \ \mathbf{E} \ [\![\Delta_\Pi b]\!]_\rho$,
if ϕ is a : b, and

(42.3c) $\rho \models\!\!=^\Delta a = b \iff [\![\Delta_\lambda a]\!]_\rho = [\![\Delta_\lambda b]\!]_\rho$,
if ϕ is a = b,

whence

(5c) c-validity ("context validity") for $\mathbf{CST}_1$-sentences ϕ is defined by

$$\models\!\!=^\Delta \phi \iff \forall\rho.\ \rho \models\!\!=^\Delta \phi.$$

Then Theorem **34.3** remains true, with $\models\!\!=^\Delta$ in place of $\models^\Delta$, but this semantics (interpreting terms by "context-closures") makes also (U) and (UU) valid in the model, with all implied disadvantages. Mutatis mutandis, the "context-closure"-semantics should also work for the finitary retract model(s), proceeding somewhat as in [McCracken 83]. Note, however, that, for the purposes of [McCracken 83], one does not need a Universe expressed as an element of the model, nor "context-closure". This is due to the fact that the

type-system of the second-order typed lambda-calculus can be given "beforehand": the types are not allowed to depend on "object-parameters", but only on "type-parameters". In fact, the Girard-Reynolds calculi do not really contain so-called "dependent types", in the genuine sense of Constructive Type Theory (or AUTOMATH); it is easy to see (and well-known from [Martin-Löf 72], say) that a second-order system is a proper fragment of $\mathbf{CST}_1$. Thus the preceeding considerations would equally apply (although somewhat uneconomically) to the second-order systems of [McCracken 79,83], [Mitchell 84] or [Reynolds 84] as well as to some of the formalisms studied in [Girard 71,72].

REFERENCES

Aczel, P. 77

The strength of Martin-Löf's intuitionistic type theory with one universe,

in: **Proc. Symp. Math. Logic**, Oulu, 1974 and Helsinki, 1975, (S. Miettinen and S. Väänänen, eds.), Report No 2, University of Helsinki, Department of Philosophy, 1977, pp. 1-32.

Aczel, P. 78

On the type theoretic interpretation of constructive set theory,

in: **Logic Colloquium '77**, (A. Macintyre, L. Pacholski and J. Paris, eds.), North Holland, Amsterdam, etc., pp. 55-66.

Aczel, P. 80

Frege structures and the notion of proposition, truth and set,

in: **The Kleene Symposium**, (J. Barwise, H. J. Keisler and K. Kunen,eds.), North Holland, Amsterdam, etc., pp. 31-60.

Aczel,P. 81

Martin-Löf's Language for Constructive Mathematics, Lectures given at the University of Amsterdam, November, 1981, accompanying **The Beth Lectures 1981**.

Aczel, P. 82

The type theoretic interpretation of constructive set theory: choice principles,

in: **The L. E. J. Brouwer Centenary Symposium**, (A. S. Troelstra and D. van Dalen, eds.), North Holland, Amsterdam, etc., pp. 1-40.

Barendregt, H. P. 84

The Lambda Calculus, Its Syntax and Semantics, North Holland, Amsterdam, etc. (second revised edition)

Barendregt, H. P. and Rezus, A. 83

Semantics for classical AUTOMATH and related systems, **Information and Control, 59**, 1983, pp. 127-147.

Beeson,M. 82

Recursive models for constructive set theories, **Annals of Mathematical Logic, 23**, 1982, pp. 127-178, (available since 1980, as Preprint 80-179, University of Utrecht, Department of Mathematics).

Beeson, M. 85

Foundations of Constructive Mathematics: Metamathematical Studies, Springer Verlag, Berlin, etc., (Ergebnisse der Mathematik und ihrer Grenzgebiete, 3. Folge, Band 6), XXIII + 466 pp.

Beeson, M. 85a

Proving programs and programming proofs, **Logic, Methodology, and Philosophy of Science, VII**, North Holland, Amsterdam, etc. [to appear]

Birkhoff, G. 67

Lattice Theory, AMS, Providence, Rhode Island.

Bishop, E. 67

Foundations of Constructive Analysis, McGraw Hill, New York, etc.

Bruce, K. B. and Meyer, A. R. 84

A semantics of second order polymorphic lambda calculus, in: [Kahn et alii 84], pp. 131-144

de Bruijn, N. G. 80

A survey of the project AUTOMATH, in: [Seldin & Hindley 80], pp. 579-607.

Burstall, R. and Lampson, B. 84

A kernel language for abstract data types and modules, in: [Kahn et alii 84], pp. 1-50, (also revised as Research Report Digital Systems Research Center, Palo Alto, CA, September 1984).

Cardelli, L. and MacQueen, D. 83 [-85,continued], (eds.)

Polymorphism, The ML/LCF/Hope Newsletter, (non-periodical, privately circulated from Bell Laboratories, Murray Hill, NJ 07974).

Cartwright, R. 80
A constructive alternative to axiomatic data type definitions, in: **Proc. 1980 LISP Conference**, Stanford (also available as TR 80-427, Cornell University, Department of Computer Science, Ithaca, NY), 13 pp.

Constable, R. L. 71
Constructive mathematics and automatic proof writers, in: **Proc. IFIP Congress '71**, Ljubljana, 1971, pp. 229-233.

Constable, R.L. 80
Programs and types, in: **Proc. 21st Symp. Foundations Comp. Sci.**, IEEE, New York, pp. 118-128.

Constable R. L. 82
Intensional analysis of functions and types, Preprint, University of Edinburgh, Department of Computer Science, 77 pp.

Constable, R. L. 83
Constructive mathematics as a programming logic, I. Some principles of theory, in: **Foundations of Computer Science**, International Conf., Borgholm, Sweden, August, 1983, (M. Karpinksi, ed.), Springer Verlag, Berlin, etc., pp. 64-77, (LNCS 158).

Constable, R. L. 83a
Programs as proofs, **Inf. Process. Letters, 16**, 1983, pp. 105-112

Constable, R. L. and Zlatin, D. R. 84
The type theory of PL/CV3, **ACM Transactions Programming Languages Systems, 6**, 1984, pp. 94-117.

Demers, A. and Donahue, J. 79
Revised report on Russell, TR 79-389, Cornell University, Department of Computer Science, Ithaca, NY, (September 1979), 41 pp.

Diller, J. 80
Modified realizability and the formulae-as-types notion, in: [Seldin & Hindley 80], pp. 491-502.

Diller, J. and Troelstra, A. S. 84
Realizability and intuitionistic logic, **Synthese, 60**, 1984, pp. 253-282, (available since 1982 as Report 82-12, University of Amsterdam, Mathematical Institute).

Erné, M. 80

Veralgemeinerungen der Verbandstheorie I, Report No 109, University of Hannover, Department of Mathematics (see also **Order, Topology and Closure**, Springer Verlag, Berlin, etc. [to appear]).

Fortune, S., Leivant, D. and O'Donnell, M. 83

The expressiveness of simple and second-order type structure, **Journal ACM, 30**, 1983, pp. 151-185, (available since 1980 as Research Report IBM, RC 8542, Computer Science, IBM Thomas J. Watson Research Center, Yorktown Heights, NY).

Gierz, G. et alii 80

A Compendium of Continuous Lattices, Springer Verlag, Berlin, etc.

Girard, J.-Y. 71

Une extension de l'interpretation de Gödel à l'analyse, et son application a l'élimination des coupures dans l'analyse et la théorie des types,
in: **Proc. Second Scandinavian Logic Symp.**, (J. E. Fenstad, ed.), North Holland, Amsterdam, etc., pp. 63-92.

Girard, J.-Y. 72

Interpretation fonctionelle et élimination des coupures de l'arithmetique d'ordre superieur, Ph. D. Dissertation (Doctorat d'Etat), University of Paris 7.

Hook, M. 84

Understanding Russell - a first attempt, in [Kahn et alii 84], pp. 69-86.

Hosono, C. and Sato, M. 77

The retracts of Pω do not form a continuous lattice - A solution to Scott's problem,
Theoretical Computer Science, 4, 1977, pp. 137-142.

Kahn, G. et alii 84

Semantics of Data types, International Symp., Sophia-Antipolis, France, June 1984, Springer Verlag, Berlin, etc. (LNCS 173).

Mal'cev, A. I. 70

Algorithms and Recursive Functions, Wolters and Noordhoff, Groningen, (English translation of the Russian original: "Nauka", Moscow, 1965, by L. F. Boron, L. E. Sanchis, J. Stillwell and K. Iséki).

Martin-Löf, P. 71

A theory of types, Report No 71-3, University of Stockholm, Mathematical Institute (February 1971, revised October 1971), 57 pp.

Martin-Löf, P. 72

An intuitionistic theory of types, Report, University of Stockholm, Mathematical Institute, 86 pp.

Martin-Löf, P. 75

An intuitionistic theory of types : predicative part, in: **Logic Colloquium '73**, (H. Rose and J. Sheperdson, eds.), North Holland, Amsterdam, etc., (written in 1972-1973), pp. 73-118.

Martin-Löf, P. 75a

Syntax and Semantics of Mathematical Language, Lectures given at Oxford University, Michaelmas Term, 1975; Lecture Notes by Peter Hancock, partial, 56 pp.

Martin-Löf, P. 80

Lectures on Constructive Set Theory, Lectures delivered at the Symposium "Konstruktive Mengenlehre und Typentheorien", Munich, 29 September - 3 October 1980 (organized by. Prof. Dr. Helmuth Schwichtenberg, University of Munich, Institute of Mathematics). The content of these lectures is essentially reported in [Beeson 82] and [Beeson 85,VI].

Martin-Löf, P. 82

Constructive mathematics and computer programming, in: **Logic, Methodology, and Philosophy of Science, VI** , (L. J. Cohen,J. Łos,H. Pfeiffer and K. D. Podewski, eds.), North Holland, Amsterdam and Polish Scientific Publishers, Warsaw, (available since 1979 as Report 79-11, University of Stockholm, Mathematical Institute), pp. 153-179.

Martin-Löf, P. 84

Intuitionistic Type Theory, Lectures given at the Laboratorio per Ricerche di Dinamica dei Sistemi e di Eletronica Biomedica, CNR, Padua, Italy, June 1980; Lecture Notes by Giovanni Sambin, Bibliopolis, Edizioni di Filosofia e Scienze, ("Studies in Proof Theory", Lecture Notes I), Naples (via Arangio Ruiz 83, Naples, Italy), 92 pp.

McCracken, N. J. 79

An Investigation of a Programming Language with a Polymorphic Type Structure, Ph. D. Dissertation, Syracuse University, School of Computer and Information Science, IV + 125 pp.

McCracken, N. J. 83

A finitary retract model for the polymorphic lambda-calculus, TR 83-2, Syracuse University, School of Computer and Information Science, 52 pp.

Information and Control, [to appear].

Meyer, A. R. 82

What is a model of the lambda calculus ?,

Information and Control, 52 1982, pp. 87-121.

Mislove, M. W. 82

An introduction to the theory of continuous lattices,

in: **Ordered Sets**, (I. Rival, ed.), Reidel, Dordrecht, etc., pp. 379-406.

Mitchell, J. C. 84

Lambda Calculus Models of Typed Programming Languages, Ph. D. Dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambrige, MA, (August, 1984).

Mitchell, J. C. and Plotkin, G. 84

Abstract types have existential type,

in: **Conf. Record 12th ACM Symp. Principles of Programming Languages**, [extended abstract], pp. 37-51, (= Chapter II of [Mitchell 84]).

Myhill, J. and Sheperdson, J. C. 55

Effective operations on partial recursive functions, in:

Zeitschrift math. Logik Grundlagen Math., 1, 1955, pp. 310-317.

Nordström, B. 81

Programming in constructive set theory: some examples,

in: **Proc. 1981 ACM Conf. Functional Languages Computer Architecture**, Wenthworth-by-the-Sea, Portsmouth, NH, pp. 141-153.

Nordström, B. and Petersson, K. 83

Types and specifications,

in: **Proc. IFIP Congress '83**, Paris, (R. E. A. Mason, ed.), Elsevier Science Publishers, North Holland, Amsterdam, pp. 915-920.

Nordström, B. and Smith, J. 82

Why type theory for programming ? A short introduction, LPM Report, Department of Computer Science, University of Gothenburg and Chalmers Institute of Technology, Gothenburg, (see [Nordström & Smith 84]).

Nordström, B. and Smith, J. 84

Propositions, types and specifications of programs in Martin-Löf's type theory,
in: **BIT, 24**, 1984, pp. 288-301.

Park, D. 76
The Y combinator in Scott's lambda-calculus models, Theory of Computation Report 76-13, University of Warwick, Department of Computer Science, [revised version].

Petersson, K. 82
A programming system for type theory, LPM Memo 21, Department of Computer Science, University of Gothenburg and Chalmers Institute of Technology, Gothenburg, (March 1982), IV + 90 pp.

Plokin, G. 72
A set theoretical definition of application, Memo MIP-R-95 University of Edinburgh, Scotland, School of Artificial Intelligence, 32 pp.

Rezus, A. 81
Lambda-conversion and Logic, Ph. D. Dissertation, University of Utrecht, XII + 198 pp.

Rezus, A. 83
Abstract AUTOMATH, Mathematical Centre, Amsterdam, ("Mathematical Centre Tracts" 160).

Rezus, A. 83a
Constructive set theory and functional programming, Paper presented to the "Algemeen Informatica Colloquium", University of Nijmegen, Department of Computer Science, December 1983.

Rezus, A. 85
Generalized Polymorphism [in preparation]

Reynolds, J. 74
Towards a theory of type structure,
in: **Proc. Programming Symp.**, Paris, April, 1974, (B. Robinet, ed.), Springer Verlag, Berlin, etc., pp. 408-425, (LNCS 19).

Reynolds, J. 84
Three approaches to type structure,
in: **Mathematical Foundations of Software Development**, International Joint Conf. TAPSOFT, vol. 1 ("Colloquium on Trees in Algebra and Programming"), (H. Ehrig, C. Floyd, M. Nivat and J. Thatcher, eds.), Springer Verlag,

Berlin, etc., pp. 97-138, (LNCS 185).

Rogers, H. 67

Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York, 'etc.

Sanchis, L. E. 77

Data types as lattices: retractions, closures and projections, in: **R A I R O Informatique Theorique, 11**, 1977, pp. 329-344.

Schwichtenberg, H. 83

On Martin-Löf's theory of types, Preprint, University of Munich, Department of Mathematics, 27 pp.

Scott, D. S. 70

Constructive validity, in: **Symposium on Automatic Demonstration**, IRIA, Versailles, (December 1968), Springer Verlag, Berlin, etc. (LNM 125), pp. 237-275.

Scott, D. S. 72

Continuous lattices, in: **Toposes, Algebraic Geometry and Logic**, (F. W. Lawvere, ed.), Springer Verlag, Berlin, etc., pp. 97-136.

Scott, D. S. 73

Models for various type-free calculi, in: **Logic, Methodology, and Philosophy of Science, IV**, (P. Suppes, A. Joja and Gr. C. Moisil, eds.), North Holland, Amsterdam, etc., pp. 157-187.

Scott, D. S. 74

The language LAMBDA [abstract], in: **The Journal of Symbolic Logic, 39**, 1974, pp. 425-427.

Scott, D. S. 75

Combinators and classes, in: **λ -calculus and Computer Science Theory**, Proc. Rome Symp., March 1975, (C. Böhm, ed.), Springer Verlag, Berlin, etc., pp. 1-26.

Scott, D. S. 75a

Lambda calculus and recursion theory, in: **Proc. Third Scandinavian Logic Symposium**, (S. Kanger, ed.), North Holland, Amsterdam, etc., pp. 154-193.

Scott, D. S. 76

Data types as lattices,
in: **S I A M Journal of Computing**, 5, 1976, pp. 522-587.

Scott, D. S. 81
Lectures on a Mathematical Theory of Computation,
in: **Theoretical Foundations of Programming Methodology**, International Summer School, Marktoberdorf, 1981, (M. Broy and G. Schmidt, eds.), Reidel, Dordrecht, etc., pp. 145-292, Lectures given in Oxford University, (Merton College), Marktoberdorf, and elsewhere, available since 1979.

Scott, D. S. 82
Some ordered sets in computer science,
in: **Ordered Sets**, (I. Rival, ed.), Reidel, Dordrecht, etc., pp. 677-718.

Scott, D. S. 82a
Domains for denotational semantics, (a corrected and expanded version of a paper prepared for ICALP '82, Aarhus, Denmark, July 1982),
Information and Control, [to appear]

Seely, R. A. G. 82
Locally cartesian closed categories and type theory,
Comptes Rendus Mathematiques Acad. Sci., The Royal Society of Canada, 4, 1982, pp. 271-275.

Seely, R. A. G. 84
Locally cartesian categories and type theory,
Math. Proc. Cambrige. Phil. Soc, 95, 1984, pp. 33-48.

Seldin, J. P. and Hindley, J. R. 80 (eds.)
To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press, London, etc.

Smith, J. 78
On the relation between a type theoretic and a logical formulation of the theory of constructions, Ph. D. Dissertation, University of Gothenburg.

Smith, J. 83
The identification of propositions and types in Martin-Löf's type theory: a programming example,
in: **Foundations of Computation Theory**, International Conf., Borgholm,

Sweden, August, 1983, (M. Karpinski, ed.), Springer Verlag, Berlin, etc., pp. 445-456, (LNCS 158).

Smith, J. 84

An interpretation of Martin-Löf's type theory in a type free theory of propositions,

The Journal of Symbolic Logic, 49, 1984, pp.730-753, (available since 1980, as Report No 80-20, Chalmers Institute of Technology and the University of Gothenburg).

Tarski, A. 55

A lattice theoretical fixpoint theorem and its applications,

Pacific Journal of Mathematics, 5, 1955, pp. 285-309.

Zucker, J. 75

Formalization of classical mathematics in AUTOMATH,

in: **Colloque international de logique**, Clermont-Ferrand, France, July, 1975, Editions du Centre National de Recherche, Paris, pp. 135-145.

Added in proof (May 1986).Peter de Bruin has noticed that the "strict closure semantics" does not yield soundness for the system with one Universe.This implies a correction of some statements in this paper,particularly in Sections 34.2,34.4 and 34.6. Details will be included in a fortcoming paper.