Kripke Models and the (In)equational Logic of the Second-Order Lambda-Calculus

Jean Gallier

University of Pennsylvania
3401 Walnut Street, Suite 400C
Philadelphia, PA 19104-6228

September 1995

IRCS Report 95-25

Kripke models and the (in)equational logic of the second-order λ-calculus

Jean Gallier[*]
Department of Computer and Information Science
University of Pennsylvania
200 South 33rd St.
Philadelphia, PA 19104, USA
e-mail: `jean@saul.cis.upenn.edu`

August 23, 1995

Abstract. We define a new class of Kripke structures for the second-order λ-calculus, and investigate the soundness and completeness of some proof systems for proving inequalities (rewrite rules) as well as equations. The Kripke structures under consideration are equipped with preorders that correspond to an abstract form of reduction, and they are not necessarily extensional. A novelty of our approach is that we define these structures directly as functors $A: \mathcal{W} \to \mathtt{Preor}$ equipped with certain natural transformations corresponding to application and abstraction (where $\mathcal{W}$ is a preorder, the set of worlds, and $\mathtt{Preor}$ is the category of preorders). We make use of an explicit construction of the exponential of functors in the Cartesian-closed category $\mathtt{Preor}^{\mathcal{W}}$, and we also define a kind of exponential $\prod_{\Phi}(A^s)_{s\in T}$ to take care of type abstraction. However, we strive for simplicity, and we only use very elementary categorical concepts. Consequently, we believe that the models described in this paper are more palatable than abstract categorical models which require much more sophisticated machinery (and are not models of rewrite rules anyway). We obtain soundness and completeness theorems that generalize some results of Mitchell and Moggi to the second-order λ-calculus, and to sets of inequalities (rewrite rules).

*This research was partially supported by ONR Grant NOOO14-93-1-1217.

1 Introduction

In order to have a deeper and hopefully more intuitive understanding of various typed λ-calculi and their logical properties, it is useful to define and study classes of models for these calculi. Typically, given some typed λ-calculus, we are interested in reduction or conversion properties of this calculus, and the crucial properties of reduction and conversion are axiomatized by a proof system for deriving equations or rewrite rules (for example, β-conversion). Models will be useful only if they are sound with respect to the given proof system, in the sense that provable equations (or rewrite rules) must be valid. Then, models can be helpful for showing that a certain equation $M \doteq N$ is not derivable from a given set E of equations: it is sufficient to exhibit a model in which all equations in E are valid and in which $M \doteq N$ is falsified. Conversely, we can better calibrate the strength of a proof system if we can prove a completeness theorem. For example, we say that we have *strong completeness* if we can show that for any set E of equations and any equation $M \doteq N$, if $M \doteq N$ is valid in every model of the equations in E, then $M \doteq N$ is provable from E. Then, we know that if $M \doteq N$ is not a consequence of E, then there is a model of E that falsifies $M \doteq N$. One can also consider refinements of strong completeness theorems where completeness is shown for classes of models with certain required properties.

For the simply-typed λ-calculus, models inspired by Henkin models [7] were defined by Friedman [2], who proved a strong completeness theorem, as well as another interesting completeness theorem. Plotkin [14] and Statman [17], [18], also proved some refinements of the strong completeness theorem for the simply-typed λ-calculus.

So far, we have assumed that the models under consideration have nonempty carriers for all types. However, in computer science applications, the assumption that carriers are nonempty may be unreasonable, because too restrictive. This fact was first observed by Goguen and Meseguer [5] in the framework of many-sorted algebras, and later on, by Meyer, Mitchell, Moggi, and Statman [10], for the second-order λ-calculus. The example of the polymorphic boolean type *polybool* is particularly illuminating. Consider the type

$$polybool := \forall X.\, (X \to (X \to X)),$$

of *polymorphic booleans*, and define the terms $True$, $False$, and $Cond$, as

$$True := \lambda X.\, \lambda x{:}X.\, \lambda y{:}X.\, x,$$

$$False := \lambda X.\, \lambda x{:}X.\, \lambda y{:}X.\, y,$$

$$Cond := \lambda b{:}polybool.\, b.$$

The terms $True$ and $False$ are the only (pure) closed terms of type *polybool*, and it is easy to verify that the equations

$$Cond\, True\, X x y \doteq x \qquad Cond\, False\, X x y \doteq y$$

are provable, for any term X.

For any $b{:}polybool$, what about the equation

$$Cond\, True\, bX yy \doteq y \tag{1}$$

In fact, it can be shown that this equation does not follow from the previous one. This is because there are models where (1) fails, e.g. when there are elements in *polybool* other than $True$, $False$, for instance $b = \bot_{polybool}$ (the least element of a cpo) as in the usual cpo-based model. The previous example suggets the following question:

Question: Is it consistent to assume that $True$ and $False$ are the only elements of *polybool*?

Ingenious contructions of Moggi and Coquand show that the answer is **yes**. Indeed, it can be shown that there is a model of the polymorphic λ-calculus in which *polybool* consists exactly of two elements. In this model, (1) is valid. But, these models contain *empty types*. In fact, Meyer, Mitchell, Moggi, and Statman [10] showed that

In any (nontrivial) model of the polymorphic λ-calculus with all types nonempty, equation (1) is not valid. In particular, there must be at least three elements of type *polybool* in such a model.

Breazu-Tannen and Coquand [1] showed that these results can be extended to types of the form $\sigma = \forall X_1 \ldots \forall X_n.\, \tau$, where τ is a quantifier-free type (in the sense that there is a model in which elements of the type σ are precisely those definable by the pure closed terms of type σ iff models have empty types).

Thus, models with empty types are **indispensable**. Unfortunately, empty types cause trouble w.r.t. soundness and completeness! The *"generic" model* property also fails for models with empty carriers. For example, consider the set E consiting of a the single equation

$$E = \{ \rhd \lambda x\colon \sigma.\, \lambda y\colon \tau.\, True \doteq \lambda x\colon \sigma.\, \lambda y\colon \tau.\, False\}.$$

Meyer, Mitchell, Moggi, and Statman [10] proved that the theory of the class $\mathcal{C}$ of all models of E (with empty carriers) is not equal to the theory of any single model.

In turn, the absense of the generic model property causes problems for completeness proofs. In the traditional proof system w.r.t. models *without* empty types, we need the rule:

$$\frac{\Gamma, x\colon \sigma \rhd M_1 \doteq M_2\colon \sigma}{\Gamma \rhd M_1 \doteq M_2\colon \sigma} \quad (nonempty)$$

provided that $x \notin FV(M_1) \cup FV(M_2)$.

But rule (*nonempty*) is not sound w.r.t. models with empty carriers! So, we can try to *delete* rule (*nonempty*) from the traditional proof system. But then, we *loose completeness*!

Let π_1 and π_2 be the simply-typed terms

$$\pi_1 = \lambda x\colon \sigma.\, \lambda y\colon \sigma.\, x, \quad \pi_2 = \lambda x\colon \sigma.\, \lambda y\colon \sigma.\, y,$$

and let $f\colon (\sigma \to \sigma \to \sigma) \to \sigma$. Then,

$$\rhd \lambda x\colon \sigma.\, (f\pi_1) \doteq \lambda x\colon \sigma.\, (f\pi_2)\colon (\sigma \to \sigma) \tag{2}$$

semantically implies

$$\rhd f\pi_1 \doteq f\pi_2\colon \sigma. \tag{3}$$

However, the above implication cannot be derived in the traditional proof system without rule (*nonempty*).

Meyer, Mitchell, Moggi, and Statman [10], gave a complete proof system w.r.t. models with empty carriers. However, reasoning in such a system is rather complicated, since it is necessary to add new axioms

$$\mathit{empty}(\sigma), x{:}\,\sigma \triangleright \mathit{True} \doteq \mathit{False}{:}\,\mathit{polybool}$$

and a new rule to reason by cases:

$$\frac{\Gamma, x{:}\,\sigma \triangleright M \doteq N{:}\,\tau \quad \Gamma, \mathit{empty}(\sigma) \triangleright M \doteq N{:}\,\tau}{\Gamma \triangleright M \doteq N{:}\,\tau} \quad (\text{cases})$$

where $x \notin FV(M) \cup FV(N)$.

Also, to the best of our knowledge, a detailed completeness proof has not been published. Thus, it appears that dealing with models with empty types is not such a simple matter, and that classical models do not seem well suited.

Mitchell and Moggi [12] observed that after all, proof systems for typed λ-calculi are intuitionistic (in most cases), and that the semantics in terms of Henkin-like models with possibly empty carriers is just too classical in nature, in the sense that arguments where we assume that a carrier is either empty or nonempty, may be used freely. Thus, Mitchell and Moggi suggested to consider intuitionistic semantics such as Kripke-style semantics. Indeed, a Kripke-style semantics forces an intuitionistic interpretation of the connectives, and extended completeness holds again for the usual proof system, regardless of the fact that carriers may be empty. Also, in the Kripke semantics, for any set E of equations, there is a Kripke model $\mathcal{A}$ such that, an equation $M \doteq N$ is valid in $\mathcal{A}$ iff $M \doteq N$ is provable from E. Besides having the virtue that these desirable completeness properties are regained in the Kripke semantics, from a categorical point of view, Kripke models are essentially equivalent to arbitrary CCC's, as sketched in Mitchell and Moggi [12]. However, this relationship will not be considered in the present paper.

In this paper, we define a new class of Kripke structures for the second-order λ-calculus, and investigate the soundness and completeness of some proof systems for proving inequalities (rewrite rules) or equations. Actually, we consider a more general class of structures. Traditionally, only models of *conversion* have been considered. However, we believe that models can also be used to prove properties of the *reduction* relation. Thus, the Kripke structures considered in this paper are equipped with preorders that correspond to an abstract form of reduction, and they are not necessarily extensional. This approach allows us to consider models of sets of rewrite rules, as well as sets of equations. We obtain soundness and completeness theorems that generalize some results of Mitchell and Moggi [12] to the second-order λ-calculus, and to sets of inequalities (rewrite rules).

Since the paper is quite technical, in order to help the reader sort out what is really new, which difficulties had to be overcome, and where are the most important results of this paper, we provide the following summary.

The new contributions are:

(1) A construction of Kripke models of the second-order λ-calculus, extending that of Mitchell and Moggi for the simply-typed λ-calculus.

(2) The fact that these Kripke models are models of the *reduction* relation, and not just of the *conversion* relation.

(3) A clarification of the nature of extensionality.

(4) Proof systems for rewrite rules as well as equations, and proofs of soundness and completeness with respect to the new class of Kripke models (also, the generic model property).

Not surprisingly, the greatest difficulties were encountered in looking for an interpretation of second-order types. Inspired by Breazu-Tannen and Coquand's notion of a type algebra [1] and a model constuction in Gunter [6], we eventually came up with the idea of the dependent product $\mathcal{D}\Pi_{\Phi}(A^s)_{s\in T}$. We were stuck for quite a while, not having realized that $\mathcal{D}\Pi_{\Phi}(A^s)_{s\in T}$ is really an exponential. Once we realized that a functorial contruction was necessary, everything got unlocked. We believe that our construction is quite elegant (although hard-core category scientists might have preferred an invocation of the Yoneda lemma). The construction of a generic model is not that different from that of Mitchell and Moggi, except that checking the details regarding polymorphic types is quite involved. Similarly, the soundness proof is very tedious, but fairly standard.

Another point that gave us quite a bit of trouble is extensionality. It took us a long time to realize that extensionality corresponds to the injectivity of some of the primitive operators involved in the definition of models. Again, we believe that our solution is quite elegant, and sheds some new light on the nature of extensionality.

Finding the proof systems for rewrite rules was fairly straightforward, but tuning the extensionality rules was a bit tricky. Contrary to proof systems for equations, extensionality rules are not equivalent to η-like rules. We also observed that the substitution rule cannot always be dispended with (in the nonextensional case).

The most important sections of this paper are section 4, where Kripke structures are defined, section 6, where the proof systems are defined, and section 7, where the soundness and completeness results are proved (lemma 7.1, lemma 7.2, theorem 7.3).

Although we were not expecting to use any category theory in this paper, we realized that this was almost unvoidable in order to come up with the "right" concepts. In particular, we don't believe that we would have come up with the right notion of dependent product for interpreting typed λ-abstraction, if we had not known that categories of presheaves are Cartesian-closed. Thus, we found it convenient to define these structures directly as functors $A\colon \mathcal{W} \to \mathtt{Preor}$ equipped with certain natural transformations corresponding to application and abstraction (where $\mathcal{W}$ is a preorder, the set of worlds, and $\mathtt{Preor}$ is the category of preorders). We make use of an explicit construction of the exponential of functors in the Cartesian-closed category $\mathtt{Preor}^{\mathcal{W}}$, and we also define a kind of exponential $\prod_{\Phi}(A^s)_{s\in T}$ to take care of type abstraction. However, we only use elementary categorical concepts, and we do not appeal to any fancy machinery.

Actually, categorical models of polymorphic λ-calculi have been investigated by Seely [16] and Pitts [13]. Seely works with so-called PL categories, and obtains a soundness and completeness theorem for the equational $\beta\eta$-theory of a version of the ω-order λ-calculus. The completeness theorem is a consequence of an equivalence of categories. We have no idea how to construct a counter-example model, or whether this can be done at all, but we also have to admit that the categorical machinery is well beyond our level of sophistication. Pitts gives a construction for embedding a so-called $2T\Lambda C$-hyperdoctrine into a topos model. This is achieved in two steps, the first one beeing a Grothendieck fibration construction, and the second one a Yoneda embedding. Pitts does obtain a soundness and completeness theorem for the the equational $\beta\eta$-theory of the second-order λ-calculus. Again, we have to confess that the categorical machinery is well beyond our level of sophistication. Nevertheless, in view of these two rather abstract constructions, we do not see how explicit counter-example models could be obtained easily. With our class of models, such counter-examples can be obtained rather easily by a quotient construction. Furthermore, we can also handle nonextensional models, and rewrite rules. Considering the level of sophistication required to handle equations with categorical models, we worry that constructing categorical models of reduction could be really complicated. We view our work as a necessary preliminary step in investigating models of reduction for the second-order λ-calculus, more in a proof-theoretic spirit than a categorical spirit, and we leave the more sophisticated categorical constructions as a challenge to categorists.

In order to understand what motivated our definition of a Kripke structure for the second-order λ-calculus, it is useful to review the usual definition of an applicative structure for the simply-typed λ-calculus (for example, as presented in Gunter [6]). For simplicity, we are restricting our attention to arrow types. Let $\mathcal{T}$ be the set of simple types built up from some base types using the constructor $\rightarrow$. Given a signature Σ of function symbols, where each symbol in Σ is assigned some type in $\mathcal{T}$, an *applicative structure* $\mathcal{A}$ is defined as a triple

$$\langle (A^{\sigma})_{\sigma\in\mathcal{T}},\ (\mathtt{app}^{\sigma,\tau})_{\sigma,\tau\in\mathcal{T}},\ \mathit{Const}\rangle,$$

where

$(A^{\sigma})_{\sigma\in\mathcal{T}}$ is a family of nonempty sets called *carriers*,

$(\mathtt{app}^{\sigma,\tau})_{\sigma,\tau\in\mathcal{T}}$ is a family of *application operators*, where each $\mathtt{app}^{\sigma,\tau}$ is a total function $\mathtt{app}^{\sigma,\tau}\colon A^{\sigma\rightarrow\tau}\times A^{\sigma}\rightarrow A^{\tau}$;

and *Const* is a function assigning a member of A^{σ} to every symbol in Σ of type σ.

The meaning of simply-typed λ-terms is usually defined using the notion of an *environment*, or *valuation*. A valuation is a function $\rho\colon\mathcal{X}\rightarrow\bigcup(A^{\sigma})_{\sigma\in\mathcal{T}}$, where $\mathcal{X}$ is the set of term variables. Although when nonempty carriers are considered (which is the case right now), it is not really necessary to consider judgements for interpreting λ-terms, since we are going to consider more general applicative structures, we define the semantics of terms using judgements. Recall that a judgement is an expression of the form $\Gamma\rhd M\colon\sigma$, where Γ, called a context, is a set of variable declarations of the form $x_1\colon\sigma_1,\ldots,x_n\colon\sigma_n$, where the x_i are pairwise distinct and the σ_i are types, M is a simply-typed λ-term, and σ is a type. There is a standard proof system that allows to type-check terms. A term M type-checks with type σ in the context Γ (where Γ contains an assignment of types to all the variables in M) iff the judgement $\Gamma\rhd M\colon\sigma$ is derivable in this proof system. Given a context Γ, we say that a valuation ρ *satisfies* Γ iff $\rho(x)\in A^{\sigma}$ for every $x\colon\sigma\in\Gamma$ (in other words, ρ respects the typing of the variables declared in Γ). Then given a context Γ and a valuation ρ satisfying Γ, the meaning $[\![\Gamma\rhd M\colon\sigma]\!]\rho$ of a judgement $\Gamma\rhd M\colon\sigma$ is defined by induction on the derivation of $\Gamma\rhd M\colon\sigma$, according to the following clauses:

$[\![\Gamma\rhd x\colon\sigma]\!]\rho=\rho(x)$, if x is a variable;

$[\![\Gamma\rhd c\colon\sigma]\!]\rho=\mathit{Const}(c)$, if c is a constant;

$[\![\Gamma\rhd MN\colon\tau]\!]\rho=\mathtt{app}^{\sigma,\tau}([\![\Gamma\rhd M\colon(\sigma\rightarrow\tau)]\!]\rho,\ [\![\Gamma\rhd N\colon\sigma]\!]\rho)$,

$[\![\Gamma \triangleright \lambda x\colon \sigma.\, M\colon (\sigma \to \tau)]\!]\rho = f$, where f is the unique element of $A^{\sigma\to\tau}$ such that $\mathtt{app}^{\sigma,\tau}(f,\, a) = [\![\Gamma, x\colon \sigma \triangleright M\colon \tau]\!]\rho[x\colon = a]$, for every $a \in A^{\sigma}$.

Note that in order for the element $f \in A^{\sigma\to\tau}$ to be uniquely defined in the last clause, we need to make certain additional assumptions. First, we assume that we are considering *extensional* applicative structures, which means that for all $f, g \in A^{\sigma\to\tau}$, if $\mathtt{app}(f,\, a) = \mathtt{app}(g,\, a)$ for all $a \in A^{\sigma}$, then $f = g$. This condition garantees the uniqueness of f if it exists. The second condition is more technical, and asserts that each A^{σ} contains enough elements so that there is an element $f \in A^{\sigma\to\tau}$ such that $\mathtt{app}^{\sigma,\tau}(f,\, a) = [\![\Gamma, x\colon \sigma \triangleright M\colon \tau]\!]\rho[x\colon = a]$, for every $a \in A^{\sigma}$.

Note that each operator $\mathtt{app}^{\sigma,\tau}\colon A^{\sigma\to\tau} \times A^{\sigma} \to A^{\tau}$ induces a function $\mathtt{fun}^{\sigma,\tau}\colon A^{\sigma\to\tau} \to [A^{\sigma} \Rightarrow A^{\tau}]$, where $[A^{\sigma} \Rightarrow A^{\tau}]$ denotes the exponential of A^{σ} and A^{τ} (in this case, since we are in the category of sets, the set of functions from A^{σ} to A^{τ}), defined such that

$$\mathtt{fun}^{\sigma,\tau}(f)(a) = \mathtt{app}^{\sigma,\tau}(f,\, a),$$

for all $f \in A^{\sigma\to\tau}$, and all $a \in A^{\sigma}$. Then, extensionality is equivalent to the fact that each $\mathtt{fun}^{\sigma,\tau}$ is injective. Note that $\mathtt{fun}^{\sigma,\tau}\colon A^{\sigma\to\tau} \to [A^{\sigma} \Rightarrow A^{\tau}]$ is the "curried" version of $\mathtt{app}^{\sigma,\tau}\colon A^{\sigma\to\tau} \times A^{\sigma} \to A^{\tau}$, and it exists because the category of sets is Cartesian-closed. For the category of sets, the fact that $[A^{\sigma} \Rightarrow A^{\tau}]$ is an exponential object is a triviality, but for more general categories, as this will be the case when we define Kripke structures (categories of presheaves), the existence of exponentials is no longer a trivial fact (but not a difficult one).

The clause defining $[\![\Gamma \triangleright \lambda x\colon \sigma.\, M\colon (\sigma \to \tau)]\!]\rho$ suggests that a partial map $\mathtt{abst}^{\sigma,\tau}\colon [A^{\sigma} \Rightarrow A^{\tau}] \to A^{\sigma\to\tau}$, "abstracting" a function $\varphi \in [A^{\sigma} \Rightarrow A^{\tau}]$ into an element $\mathtt{abst}^{\sigma,\tau}(\varphi) \in A^{\sigma\to\tau}$, can be defined. For example, the function φ defined such that $\varphi(a) = [\![\Gamma, x\colon \sigma \triangleright M\colon \tau]\!]\rho[x\colon = a]$ would be mapped to $[\![\Gamma \triangleright \lambda x\colon \sigma.\, M\colon (\sigma \to \tau)]\!]\rho$. In order for the resulting structure to be a model of β-reduction, we just have to require that $\mathtt{fun}^{\sigma,\tau}$ and $\mathtt{abst}^{\sigma,\tau}$ satisfy the axiom

$$\mathtt{fun}^{\sigma,\tau}(\mathtt{abst}^{\sigma,\tau}(\varphi)) = \varphi,$$

whenever $\varphi \in [A^{\sigma} \Rightarrow A^{\tau}]$ is in the domain of $\mathtt{abst}^{\sigma,\tau}$. But now, observe that if pairs of operators $\mathtt{fun}^{\sigma,\tau}, \mathtt{abst}^{\sigma,\tau}$ satisfying the above axiom are defined, the injectivity of $\mathtt{fun}^{\sigma,\tau}$ is superfluous for defining $[\![\Gamma \triangleright \lambda x\colon \sigma.\, M\colon (\sigma \to \tau)]\!]\rho$.

Thus, by defining a more general kind of applicative structure using the operators $\mathtt{fun}^{\sigma,\tau}$ and $\mathtt{abst}^{\sigma,\tau}$, we can still give meanings to λ-terms, even when these structures are nonextensional. In particular, our approach is an alternative to the method where one considers applicative structures with meaning functions, as for example in Mitchell [11]. In particular, the term structure together with the meaning function defined using substitution can be seen to be an applicative structure according to our definition. In fact, this approach allows us to go further. We can assume that each carrier A^{σ} is equipped with a preorder $\preceq^{\sigma}$, and rather than considering the equality

$$\mathtt{fun}^{\sigma,\tau}(\mathtt{abst}^{\sigma,\tau}(\varphi)) = \varphi,$$

we can consider inequalities

$$\mathtt{fun}^{\sigma,\tau}(\mathtt{abst}^{\sigma,\tau}(\varphi)) \succeq \varphi.$$

This way, we can deal with intentional (nonapplicative) structures that model reduction rather than conversion. We learned from Gordon Plotkin that models of β-reduction (or $\beta\eta$-reduction) have been considered before, in particular by Girard [4], Jacobs, Margaria, and Zacchi [8], and Plotkin [15]. However, except for Girard who studies qualitative domains for system F, the other authors consider models of the untyped λ-calculus. In [4], definition 1.12, Girard defines a *λ-structure* as a triple $D = \langle X, H, K\rangle$ consisting of

(i) a qualitative domain X,

(ii) a stable function H from X to $X \Rightarrow X$, and

(iii) a stable function K from $X \Rightarrow X$ to X,

where $X \Rightarrow X$ is the set of all traces of stable functions from X to X. Girard then shows that a λ-structure D models β-reduction if $H \circ K \subset Id_{X \Rightarrow X}$, and that D models η-reduction if $K \circ H \subset Id_X$ (note that the partial order $\subset$ corresponds to the opposite of our ordering $\preceq$). Girard also states that such structures have nice features, in particular because they can be approximated by finite λ-structures.

The major difference with our approach is that the above models are intended for the untyped λ-calculus.

In [15], section 3, Plotkin introduces a notion of model of β-reduction that he calls an *ordered λ-interpretation*. After Mitchell [11], Plotkin defines such a structure as a triple $\mathcal{P} = \langle P, \cdot, [\![\cdot]\!](\cdot)\rangle$, where P is a partial order, $\cdot$ is a monotonic application operation $\cdot \colon P \times P \to P$, and $[\![\cdot]\!](\cdot)$ is a *meaning function*, that maps terms and environments to P, and such that some obvious conditions on $[\![]\!](\cdot)$ hold. If the condition

$$[\![\lambda x.\, M]\!](\rho) \cdot a \preceq [\![M]\!](\rho[x := a]),$$

holds, we say that $\mathcal{P}$ is a model of β-reduction. Plotkin then proceeds to show that such models are sound and complete with respect to Curry-style type inference systems (also know as systems for F-deducibility), for various type disciplines. The main difference with our approach is that Plotkin's structures are models of the untyped λ-calculus, and that meaning functions are an intrinsic part of their definition. In our definition, the meaning function is not part of the definition, but it is uniquely defined. For our purposes, this is a much more suitable approach.

We now show how to construct Kripke structures along the ideas sketched above. First, we review Mitchell and Moggi's definition [12]. The main new ingredient is that we have a preordered set $\langle \mathcal{W}, \sqsubseteq\rangle$, intuitively, a set of worlds. Then, a *Kripke applicative structure* is defined as a tuple

$$\langle \mathcal{W}, \sqsubseteq, (A_w^\sigma)_{\sigma \in \mathcal{T}, w \in \mathcal{W}}, (\mathsf{app}_w^{\sigma,\tau})_{\sigma,\tau \in \mathcal{T}, w \in \mathcal{W}}, (i_{w_1,w_2}^\sigma)_{\sigma \in \mathcal{T}, w_1, w_2 \in \mathcal{W}}\rangle,$$

where,

$\mathcal{W}$ is a set of worlds preordered by $\sqsubseteq$,

$(A_w^\sigma)_{\sigma \in \mathcal{T}, w \in \mathcal{W}}$ is a family of (possibly empty) sets called *carriers*,

$(\mathsf{app}_w^{\sigma,\tau})_{\sigma,\tau \in \mathcal{T}, w \in \mathcal{W}}$ is a family of *application operators*, where each $\mathsf{app}_w^{\sigma,\tau}$ is a total function $\mathsf{app}_w^{\sigma,\tau} \colon A_w^{\sigma \to \tau} \times A_w^\sigma \to A_w^\tau$;

$i_{w_1,w_2}^\sigma \colon A_{w_1}^\sigma \to A_{w_2}^\sigma$ is a *transition function*, whenever $w_1 \sqsubseteq w_2$.

Furthermore, certain conditions hold, making each A^σ into a functor from $\mathcal{W}$ to `Sets`, and each $\mathtt{app}^{\sigma,\tau}$ into a natural transformation between the functors $A^{\sigma\to\tau}\times A^\sigma$ and A^τ. For example, we have

$$i^\tau_{w_1,w_2}(\mathtt{app}^{\sigma,\tau}_{w_1}(f,\, a)) = \mathtt{app}^{\sigma,\tau}_{w_2}(i^{\sigma\to\tau}_{w_1,w_2}(f),\, i^\sigma_{w_1,w_2}(a)),$$

for all $f\in A^{\sigma\to\tau}_{w_1}$ and all $a\in A^\sigma_{w_1}$.[1]

If we want to adapt this definition to give a more general definition in terms of the operators $\mathtt{fun}^{\sigma,\tau}$ and $\mathtt{abst}^{\sigma,\tau}$, we need to define $\mathtt{fun}^{\sigma,\tau}$ as the "curried" version of the natural transformation $\mathtt{app}^{\sigma,\tau}$ between the functors $A^{\sigma\to\tau}\times A^\sigma$ and A^τ. This is where we use a bit of category theory. Each A^σ can be viewed as a functor $A^\sigma\colon\mathcal{W}\to\mathtt{Sets}$ from the preorder $\mathcal{W}$ viewed as a category, and the category of sets, and these functors together with the natural transformations between them form a category, a *presheaf category*, which is known to be Cartesian-closed (see Mac Lane and Moerdijk [9]). Furthermore, it is possible to give an explicit construction of the exponential $[A^\sigma\Rightarrow A^\tau]$ (see definition 3.5) between two functors A^σ and A^τ, and to define `fun` as `curry(app)`. Then, it is easy to define a Kripke applicative structure in terms of the natural transformations $\mathtt{fun}^{\sigma,\tau}$ and $\mathtt{abst}^{\sigma,\tau}$.

In order to deal with second-order types, first, we need to provide an interpretation of the type variables. Thus, as in Breazu-Tannen and Coquand [1], we assume that we have an *algebra of types* T, which consists of a quadruple

$$\langle T, \to, [T\Rightarrow T], \forall\rangle,$$

where T is a nonempty set of types, $\to\colon T\times T\to T$ is a binary operation on T, $[T\Rightarrow T]$ is a nonempty set of functions from T to T, and $\forall$ is a function $\forall\colon[T\Rightarrow T]\to T$.

We hope that readers will forgive us for using the same letter T to denote an algebra of types and its carrier. Intuitively, given a valuation $\theta\colon\mathcal{V}\to T$ (where $\mathcal{V}$ is the set of type variables), a type $\sigma\in\mathcal{T}$ will be interpreted as an element $[\![\sigma]\!]\theta$ of T. Then, a *second-order applicative structure* is defined as a tuple

$$\langle T,\, (A^s)_{s\in T},\, (\mathtt{app}^{s,t})_{s,t\in T},\, (\mathtt{tapp}^\Phi)_{\Phi\in[T\to T]}\rangle,$$

where

T is an algebra of types;

$(A^s)_{s\in T}$ is a family of nonempty sets called *carriers*,

$(\mathtt{app}^{s,t})_{s,t\in T}$ is a family of *application operators*, where each $\mathtt{app}^{s,t}$ is a total function $\mathtt{app}^{s,t}\colon A^{s\to t}\times A^s\to A^t$;

$(\mathtt{tapp}^\Phi)_{\Phi\in[T\to T]}$ is a family of *type-application operators*, where each $\mathtt{tapp}^\Phi$ is a total function $\mathtt{tapp}^\Phi\colon A^{\forall(\Phi)}\times T\to\coprod(A^{\Phi(s)})_{s\in T}$, such that $\mathtt{tapp}^\Phi(f,\, t)\in A^{\Phi(t)}$, for every $f\in A^{\forall(\Phi)}$, and every $t\in T$.

In order to define second-order applicative structures using operators like `fun` and `abst`, we need to define the curried version $\mathtt{tfun}^\Phi$ of $\mathtt{tapp}^\Phi\colon A^{\forall(\Phi)}\times T\to\coprod(A^{\Phi(s)})_{s\in T}$. For this, we define a kind of *dependent product* $\mathcal{D}\Pi_\Phi(A^s)_{s\in T}$ (see definition 3.8). Then, we have families of operators $\mathtt{tfun}^\Phi\colon A^{\forall(\Phi)}\to\mathcal{D}\Pi_\Phi(A^s)_{s\in T}$, and $\mathtt{tabst}^\Phi\colon\mathcal{D}\Pi_\Phi(A^s)_{s\in T}\to A^{\forall(\Phi)}$, for every $\Phi\in[T\Rightarrow T]$.

Now, if we want to adapt the above definition to define Kripke applicative structures, we have to view $A^{\forall(\Phi)}\times T$ and $\coprod(A^{\Phi(s)})_{s\in T}$ as functors, and $\mathtt{tapp}^\Phi\colon A^{\forall(\Phi)}\times T\to\coprod(A^{\Phi(s)})_{s\in T}$ as

[1]Constants can be handled too, but for simplicity, they are dropped.

a natural transformation between them. Then, we need to define some form of exponential of T and $\coprod(A^{\Phi(s)})_{s\in T}$. Such an exponential can indeed be constructed as a functor $\prod_\Phi(A^s)_{s\in T}$ defined in terms of the dependent products $\mathcal{D}\Pi_\Phi(A^s_w)_{s\in T}$ (see definition 3.8). We also need to show that the functor $\prod_\Phi(A^s)_{s\in T}$ satisfies a universal property analogous to the property satisfied by the functor $[A^s \Rightarrow A^t]$. For this, we define the set $\mathtt{Nat}_\Phi(H\times T,\ \coprod(A^{\Phi(s)})_{s\in T})$ as the set of natural transformations $\eta\colon H\times T \to \coprod(A^{\Phi(s)})_{s\in T}$, such that, $\eta_u(a,t)\in A_u^{\Phi(t)}$, for every $a\in H_u$ and every $t\in T$ (see definition 3.9). Then, we can prove a lemma (lemma 3.11) that shows that $\prod_\Phi(A^s)_{s\in T}$ is indeed a certain kind of exponential. Thus, at the level of presheaf categories, we have the usual maps `curry` and `uncurry` that set up a (natural) bijection between $\mathtt{Nat}(H\times F,\ G)$ and $\mathtt{Nat}(H,\ [F\Rightarrow G])$, but also some maps $\mathtt{curry}_\Phi$ and $\mathtt{uncurry}_\Phi$ that set up a (natural) bijection between the sets of natural transformations $\mathtt{Nat}_\Phi(H\times T,\ \coprod(A^{\Phi(s)})_{s\in T})$ and $\mathtt{Nat}(H,\ \prod_\Phi(A^s)_{s\in T})$.

Armed with the definition of the functors $[A^s\Rightarrow A^t]$ and $\prod_\Phi(A^s)_{s\in T}$, and the natural transformations `fun`, `abst`, `tfun`, and `tabst`, we can define Kripke applicative structures (see definition 4.1). In fact, the definition also applies to the product and sum types, and to carriers A^s_w equipped with preorders. This way, we can define models of sets of rewrite rules, as well as models of sets of equations.

The paper is organized as follows. Section 2 is a review of the syntax of the second-order typed λ-calculus $\lambda^{\to,\times,+,\forall^2}$. Section 3 contains a review of some elementary notions of category theory. An explicit construction of the exponential of functors $F,G\colon\mathcal{W}\to\mathtt{Preor}$, where $\mathcal{W}$ is a preorder, and $\mathtt{Preor}$ is the category of preorders, is given. The dependent product $\prod_\Phi(A^s)_{s\in T}$ is also defined. Kripke pre-applicative structures are defined in section 4. In section 5, we show how to interpet second-order λ-terms using Kripke applicative structures. A number of proof systems for proving inequalities (rewrite rules) and equations are defined in section 6. Satisfaction and validity (in a Kripke structure) is also defined. Some soundness and completeness results are proved in section 7. The results of section 7 are adapted to equations in section 8. Section 9 contains the conclusion and some suggestions for further research.

2 Syntax of the Second-Order Typed λ-Calculus $\lambda^{\to,\times,+,\forall^2}$

In this section, we review quickly the syntax of the second-order typed λ-calculus $\lambda^{\to,\times,+,\forall^2}$. This includes a definition of the second-order types under consideration, of raw terms, or the type-checking rules for judgements, and of the reduction rules. For more details (on the subsystem $\lambda^{\to,\forall^2}$), the reader should consult Breazu-Tannen and Coquand [1].

Let $\mathcal{T}$ denote the set of second-order types. This set comprises type variables X, type constants k, and compound types $(\sigma\to\tau)$, $(\sigma\times\tau)$, $(\sigma+\tau)$, and $\forall X.\ \sigma$. It is assumed that we have a set TC of *type constants* (also called *base types of kind* $\star$). We have a countably infinite set $\mathcal{V}$ of type variables (denoted as upper case letters X,Y,Z), and a countably infinite set $\mathcal{X}$ of term variables (denoted as lower case letters x,y,z). We denote the set of free type variables occurring in a type σ as $FTV(\sigma)$. We use the notation $\star$ for the kind of types. Since we are only considering second-order quantification over predicate symbols (of kind $\star$) of arity 0, this is superfluous. However, it will occasionally be useful to consider contexts Γ in which type variables are explicitly present, since this makes the type-checking rules more uniform in the case of λ-abstraction and typed λ-abstraction. Thus, officially, a context Γ is a set $\{x_1\colon\sigma_1,\ \ldots,\ x_n\colon\sigma_n\}$, where $x_1,\ldots,x_n$ are term variables, and

$\sigma_1, \ldots, \sigma_n$ are types. We let $dom(\Gamma) = \{x_1, \ldots, x_n\}$. As usual, we assume that the variables x_j are pairwise distinct. We also assume that $x \notin dom(\Gamma)$ in a context $\Gamma, x\colon \sigma$. Informally, we will also consider contexts $\{X_1\colon \star, \ldots, X_m\colon \star, x_1\colon \sigma_1, \ldots, x_n\colon \sigma_n\}$, where $X_1, \ldots, X_m$ are type variables, and $x_1, \ldots, x_n$ are term variables, with the two sets $\{X_1, \ldots, X_m\}$ and $\{x_1, \ldots, x_n\}$ disjoint, the variables X_i pairwise distinct, and the variables x_j pairwise distinct. We assume that $X \notin dom(\Gamma)$ in a context $\Gamma, X\colon \star$. For the sake of brevity, rather than writing typed λ-abstraction as $\lambda X\colon \star.\, M$, it will be written as $\lambda X.\, M$.

It is assumed that we have a set *Const* of constants, together with a function *Type*: $Const \to \mathcal{T}$, such that every constant c is assigned a *closed* type $Type(c)$ in $\mathcal{T}$. The set *TC* of type constants, together with the set *Const* of constants, and the function *Type*, constitute a *signature* Σ. Let us review the definition of raw terms.

Definition 2.1 The set of *raw terms* is defined inductively as follows: every variable $x \in \mathcal{X}$ is a raw term, every constant $c \in Const$ is a raw terms, and if M, N are raw terms and σ, τ are types, then (MN), $(M\tau)$, $\lambda x\colon \sigma.\, M$, $\lambda X.\, M$, $\pi_1(M)$, $\pi_2(M)$, $\langle M,\ N\rangle$, $\mathtt{inl}(M)$, $\mathtt{inr}(M)$, and $[M,\ N]$, are raw terms.

We let $FV(M)$ denote the set of free term-variables in M. Raw terms may contain free variables and may not type-check (for example, (xx)). In order to define which raw terms type-check, we consider expressions of the form $\Gamma \rhd M\colon \sigma$, called *judgements*, where Γ is a context in which all the free term variables in M are declared. A term M type-checks with type σ in the context Γ iff the judgement $\Gamma \rhd M\colon \sigma$ is provable using axioms and rules summarized in the following definition.

Definition 2.2 The judgements of the polymorphic typed λ-calculus $\lambda^{\to,\times,+,\forall^2}$ are defined by the following rules.

$$\Gamma \rhd x\colon \sigma, \quad \text{when } x\colon \sigma \in \Gamma,$$

$$\Gamma \rhd c\colon Type(c), \quad \text{when } c \text{ is a constant},$$

$$\frac{\Gamma, x\colon \sigma \rhd M\colon \tau}{\Gamma \rhd (\lambda x\colon \sigma.\, M)\colon (\sigma \to \tau)} \quad (abstraction)$$

$$\frac{\Gamma \rhd M\colon (\sigma \to \tau) \quad \Gamma \rhd N\colon \sigma}{\Gamma \rhd (MN)\colon \tau} \quad (application)$$

$$\frac{\Gamma \rhd M\colon \sigma \quad \Gamma \rhd N\colon \tau}{\Gamma \rhd \langle M,\ N\rangle\colon \sigma \times \tau} \quad (pairing)$$

$$\frac{\Gamma \rhd M\colon \sigma \times \tau}{\Gamma \rhd \pi_1(M)\colon \sigma} \quad (projection) \qquad \frac{\Gamma \rhd M\colon \sigma \times \tau}{\Gamma \rhd \pi_2(M)\colon \tau} \quad (projection)$$

$$\frac{\Gamma \rhd M\colon \sigma}{\Gamma \rhd \mathtt{inl}(M)\colon \sigma + \tau} \quad (injection) \qquad \frac{\Gamma \rhd M\colon \tau}{\Gamma \rhd \mathtt{inr}(M)\colon \sigma + \tau} \quad (injection)$$

$$\frac{\Gamma \rhd M\colon (\sigma \to \delta) \quad \Gamma \rhd N\colon (\tau \to \delta)}{\Gamma \rhd [M,\ N]\colon (\sigma + \tau) \to \delta} \quad (co\text{-}pairing)$$

$$\frac{\Gamma, X\colon\star \rhd M\colon\sigma}{\Gamma \rhd (\lambda X.\, M)\colon \forall X.\, \sigma} \quad (\forall\text{-}intro)$$

provided that $X \notin \bigcup_{x:\tau\in\Gamma} FTV(\tau)$;

$$\frac{\Gamma \rhd M\colon \forall X.\, \sigma}{\Gamma \rhd (M\tau)\colon \sigma[\tau/X]} \quad (\forall\text{-}elim)$$

The reason why we do not officially consider that a context contains type variables, is that in the rule ($\forall$-*elim*), the type τ could contain type variables not declared in Γ, and it would be necessary to have a weakening rule to add new type variables to a context (or some other mechanism to add new type variables to a context). As long as we do not deal with dependent types, this technical annoyance is most simply circumvented by assuming that type variables are not included in contexts.

Instead of using the construct $\texttt{case}\ P\ \texttt{of}\ \texttt{inl}(x\colon\sigma) \Rightarrow M \mid \texttt{inr}(y\colon\tau) \Rightarrow N$, we found it more convenient and simpler to use the slightly more general construct $[M,\ N]$, where M is of type $\sigma \to \delta$ and N is of type $\tau \to \delta$, even when M and N are not λ-abstractions. This will be especially advantageous for the semantic treatment to follow. Then, we can define the conditional construct $\texttt{case}\ P\ \texttt{of}\ \texttt{inl}(x\colon\sigma) \Rightarrow M \mid \texttt{inr}(y\colon\tau) \Rightarrow N$, where P is of type $\sigma + \tau$, as $[\lambda x\colon\sigma.\, M,\ \lambda y\colon\tau.\, N]P$.

Definition 2.3 The reduction rules of the system $\lambda^{\to,\times,+,\forall}$ are listed below:

$$\begin{aligned}
(\lambda x\colon\sigma.\, M)N &\longrightarrow M[N/x],\\
\pi_1(\langle M, N\rangle) &\longrightarrow M,\\
\pi_2(\langle M, N\rangle) &\longrightarrow N,\\
[M,\ N]\texttt{inl}(P) &\longrightarrow MP,\\
[M,\ N]\texttt{inr}(P) &\longrightarrow NP,\\
(\lambda X.\, M)\tau &\longrightarrow M[\tau/X].
\end{aligned}$$

The reduction relation defined by the rules of definition 2.3 is denoted as $\longrightarrow_\beta$ (even though there are reductions other than β-reduction). From now on, when we refer to a λ-term, we mean a λ-term that type-checks. In order to define Kripke models for $\lambda^{\to,\times,+,\forall^2}$, we need to review a few concepts from category theory.

3 Exponentials and Dependent Products in the Category $\texttt{Preor}^{\mathcal{W}}$

In this section, we define an algebra of polymorphic types, and review some elementary notions of category theory. We give an explicit construction of the exponential of functors $F, G\colon \mathcal{W} \to \texttt{Preor}$, where $\mathcal{W}$ is a preorder, and $\texttt{Preor}$ is the category of preorders. We also define the dependent product $\prod_\Phi (A^s)_{s\in T}$, and show that this functor is a certain kind of exponential, if the right set of natural transformations is considered.

Definition 3.1 An *algebra of (polymorphic) types* is a tuple

$$\langle T, \to, \times, +, [T \Rightarrow T], \forall\rangle,$$

where T is a nonempty set of *types*, $\rightarrow, \times, +: T \times T \rightarrow T$ are binary operations on T, $[T \Rightarrow T]$ is a nonempty set of functions from T to T, and $\forall$ is a function $\forall: [T \Rightarrow T] \rightarrow T$.

We hope that readers will forgive us for using the same letter T to denote an algebra of types and its carrier. Intuitively, given a valuation $\theta: \mathcal{V} \rightarrow T$, a type $\sigma \in \mathcal{T}$ will be interpreted as an element $[\![\sigma]\!]\theta$ of T.

We need to define two categories of preorders.

Definition 3.2 The category `Preor` is the category whose objects are preordered sets $\langle W, \sqsubseteq\rangle$, and whose arrows $f: W_1 \rightarrow W_2$ are monotonic functions (with respect to $\sqsubseteq_1$ and $\sqsubseteq_2$). The category $\mathtt{Preor}_p$ is the category whose objects are preordered sets $\langle W, \sqsubseteq\rangle$, and whose arrows $f: W_1 \rightarrow W_2$ are monotonic partial functions (with respect to $\sqsubseteq_1$ and $\sqsubseteq_2$).

It is obvious that `Preor` and $\mathtt{Preor}_p$ are categories. Given a monotonic function $f: W_1 \rightarrow W_2$, where W_1 and W_2 are preorders, we say that f is *isotone* iff $f(w_1) \sqsubseteq f(w_2)$ implies that $w_1 \sqsubseteq w_2$, for all $w_1, w_2 \in W_1$.

Any preordered set $\langle \mathcal{W}, \sqsubseteq\rangle$ can be viewed as the category whose objects are the elements of $\mathcal{W}$, and such that there is a single arrow denoted $w_1 \rightarrow w_2$ from w_1 to w_2 iff $w_1 \sqsubseteq w_2$. We will be interested in functors $F: \mathcal{W} \rightarrow \mathtt{Preor}$. Such a functor assigns a preorder $F(w)$ to every $w \in \mathcal{W}$, and an arrow $F(w_1 \rightarrow w_2): F(w_1) \rightarrow F(w_2)$ to every pair such that $w_1 \sqsubseteq w_2$. The preorder $F(w)$ is also denoted as $\langle F_w, \preceq^F_w\rangle$, and the arrow $F(w_1 \rightarrow w_2)$ is a monotonic function denoted as $i^F_{w_1,w_2}: F_{w_1} \rightarrow F_{w_2}$. The fact that F is a functor means that $i^F_{w,w} = \mathtt{id}$, and that $i^F_{w_1,w_3} = i^F_{w_2,w_3} \circ i^F_{w_1,w_2}$, whenever $w_1 \sqsubseteq w_2 \sqsubseteq w_3$.

Recall that a natural transformation $\eta: F \rightarrow G$ between two functors $F, G: \mathcal{W} \rightarrow \mathtt{Preor}$ is a family $\eta = (\eta$