
Monotonicity Inference for Higher-Order Formulas

Jasmin Christian Blanchette · Alexander Krauss

the date of receipt and acceptance should be inserted later

Abstract Formulas are often monotonic in the sense that satisfiability for a given domain of discourse entails satisfiability for all larger domains. Monotonicity is undecidable in general, but we devised three calculi that infer it in many cases for higher-order logic. The third calculus has been implemented in Isabelle’s model finder Nitpick, where it is used both to prune the search space and to soundly interpret infinite types with finite sets, leading to dramatic speed and precision improvements.

1 Introduction

Formulas occurring in logical specifications often exhibit monotonicity in the sense that if the formula is satisfiable when the types are interpreted with sets of given (positive) cardinalities, it is still satisfiable when these sets become larger. Consider the following formulas, in which superscripts indicate types and \simeq denotes equality:

1. $\exists x^\alpha y. x \neq y$
2. $f x^\alpha \simeq x \wedge f y \neq y$
3. $(\forall x^\alpha. f x \simeq x) \wedge f y \neq y$
4. $\{y^\alpha\} \simeq \{z\}$
5. $\exists x^\alpha y. x \neq y \wedge \forall z. z \simeq x \vee z \simeq y$
6. $\forall x^\alpha y. x \simeq y$

It is easy to see that formulas 1 and 2 are satisfiable iff $|\alpha| > 1$, formula 3 is unsatisfiable, formula 4 is satisfiable for any cardinality of α , formula 5 is satisfiable iff $|\alpha| = 2$, and formula 6 is satisfiable iff $|\alpha| = 1$. Formulas 1 to 4 are monotonic, whereas 5 and 6 are not.

Monotonicity can be exploited in model finders to prune the search space. Model finders are automatic tools that generate finite set-theoretic models of formulas. They are useful for exploring specifications and producing counterexamples. Notable model finders include Paradox [6], MACE [12], and SEM [24] for first-order logic, Alloy [10] and Kodkod [21] for first-order relational logic, and Nitpick [4] and Refute [23] for higher-order logic.

Model finders for many-sorted or typed logics typically work by systematically enumerating the domain cardinalities for the atomic types (type variables and other uninterpreted

Research partially supported by the DFG grants Ni 491/11-1 and Ni 491/11-2.

Institut für Informatik, Technische Universität München, Germany
E-mail: {blanchette,krauss}@in.tum.de

types) occurring in the formula. To exhaust all models up to a given cardinality bound k for a formula involving n atomic types, a model finder iterates through k^n combinations of cardinalities and must consider all models for each of these combinations. In general, this exponential behavior is necessary for completeness, since the formula may dictate a model with specific cardinalities. However, if the formula is monotonic, it is sufficient to consider only the models in which all types have cardinality k .

Another use of monotonicity is to find finite fragments of infinite models. A formal specification of a programming language might represent variables by strings, natural numbers, or values of some other infinite type. Typically, the exact nature of these types is irrelevant; they are merely seen as inexhaustible name stores and used monotonically. If we weaken the specification to allow finite models, we can apply model finders and have the guarantee that any satisfying finite models correspond to satisfying infinite models.

Monotonicity occurs surprisingly often in practice. Consider the specification of a hotel key card system with recordable locks [9, pp. 299–306; 16]. Such a specification involves rooms, guests, and keys, modeled as distinct atomic types. A desirable property of the system is that only the occupant of a room may unlock it. Unsurprisingly, a counterexample requiring one room, two guests, and four keys will still be a counterexample if more rooms, guests, or keys are available. Indeed, it should remain a counterexample if infinitely many keys are available, as would be the case if keys are modeled by integers or strings.

In this article, we present three calculi for detecting monotonicity of higher-order logic (HOL) formulas. The first calculus (Section 5) simply tracks the use of equality and quantifiers. Although useful on its own, it mainly serves as a stepping stone for a second, refined calculus (Section 6), which uses a type system to detect the ubiquitous “sets as predicates” idiom and treats it specially. The third calculus (Section 7) develops this idea further. While some of the difficulties we face are specific to HOL, the calculi can be adapted to any logic that provides unbounded quantification, such as many-sorted first-order logic with equality.

The calculi are constructive: Whenever they infer monotonicity, they also yield a recipe for transforming smaller models into larger (possibly infinite) models. They are also readily extended to handle constant definitions (Section 8.1) and inductive datatypes (Section 8.2), which pervade HOL formalizations. Our evaluation (Section 8.3) is done in the context of Nitpick, a counterexample generator for Isabelle/HOL [17]. On a corpus of 1183 monotonic formulas from six theories, the strongest calculus infers monotonicity for 85% of them.

We presented an earlier version of this article at IJCAR 2010 [3]. The main additions here are the introduction of the third calculus and the treatment of constant definitions. We also elaborated some of the existing proofs and added explanations throughout.

2 Related Work

In plain first-order logic without equality, every formula is monotonic, since it is impossible to express an upper bound on the cardinality of the models and hence any model can be extended to a model of arbitrarily larger cardinality. This property is essentially a weak form of the upward Löwenheim–Skolem theorem. When equality is added, nonmonotonicity follows suit. For example, the formula $\forall xy. x \simeq y$ is satisfied only by singleton models.

Moving to higher-order logic introduces new complications. Since HOL is typed, we are interested in monotonicity with respect to a given type variable or some other uninterpreted type α . Moreover, our calculi must cope with occurrences of α in nested function types such as $(\alpha \rightarrow \beta) \rightarrow \beta$ and in datatypes such as α list. We are not aware of any previous work on inferring or proving monotonicity for HOL.

In the first-order world, Alloy constitutes an interesting case in point. Although Alloy’s logic is unsorted, models must give a semantics to “primitive types,” which are sets of uninterpreted atoms. Early versions of the logic ensured monotonicity with respect to the primitive types by providing only bounded quantification and disallowing explicit references to the sets that denote the types [10]. Monotonicity has been lost in more recent versions of Alloy, which allow such references [9, p. 165]. Nonetheless, many Alloy formulas are monotonic, notably those in existential–bounded–universal form [11].

The satisfiability-modulo-theory (SMT) community has also shown interest in monotonicity. The original Nelson–Oppen method allows the combination of decision procedures for first-order theories satisfying certain restrictions, notably that the theories are stably infinite (or finitely unsatisfiable) [15]. This criterion has since been weakened, and one of the remaining requirements on the theories to be composed is that they be “smooth,” that is, every quantifier-free formula must be monotonic modulo the theory [20]. Smoothness is usually proved with pen and paper for the theories that need to be combined.

For some logics, small model theorems provide an upper bound on the cardinality of a sort [5], primitive type [14], or variable’s domain [18]. If no model exists below that bound, no larger models exist. Paradox and Alloy exploit such theorems to speed up the search. Our approach is complementary and could be called a *large* model theorem.

3 Higher-Order Logic

Our presentation of HOL is very similar to that of Andrews [1], but instead of a single type ι of individuals, we use type variables α, β, γ to denote uninterpreted types.

Definition 3.1 (Syntax) The *types* and *terms* of HOL are that of the simply-typed λ -calculus, augmented with constants and a special type \mathbf{o} of Booleans:

Types:	Terms:
$\sigma ::= \mathbf{o}$ (Boolean type)	$t ::= x^\sigma$ (variable)
α (type variable)	c^σ (constant)
$\sigma \rightarrow \sigma$ (function type)	$t t$ (application)
	$\lambda x^\sigma. t$ (abstraction)

The function arrow associates to the right, reflecting the left-associativity of application. We assume throughout that terms are well-typed using the standard typing rules and write x and c rather than x^σ and c^σ when σ is irrelevant or clear from the context; inversely, we write t^σ to indicate that an arbitrary term t has type σ . A formula is a term of type \mathbf{o} . Constants express the logical primitives, whose interpretation is fixed a priori. We only take equality ($\simeq^{\sigma \rightarrow \sigma \rightarrow \mathbf{o}}$ for any σ) and implication ($\longrightarrow^{\mathbf{o} \rightarrow \mathbf{o} \rightarrow \mathbf{o}}$) as primitive constants.

Unlike in Gordon’s version of HOL [7], on which several popular proof assistants are based [8, 17, 19], we treat polymorphism in the metalanguage: Polymorphic constants such as equality are expressed as collections of constants, one for each type.

Definition 3.2 (Scope) A *scope* S is a function from type variables to nonempty sets (domains). We write $S \leq_\alpha S'$ to mean that $S(\alpha) \subseteq S'(\alpha)$ and $S(\beta) = S'(\beta)$ for all $\beta \neq \alpha$.

Types and terms are interpreted in the standard set-theoretic way, relative to a scope that fixes the interpretation of type variables. The set $S(\alpha)$ can be finite or infinite, although for model finding we usually have finite domains in mind. In contexts where S is clear, the cardinality of $S(\alpha)$ is written $|\alpha|$ and the elements of $S(\alpha)$ are denoted by $0, 1, 2$, etc. Scopes are also called “type environments”; our terminology here is consistent with Jackson [10].

Definition 3.3 (Interpretation of Types) The *interpretation* $\llbracket \sigma \rrbracket_S$ of a type σ in a scope S is defined recursively by the equations

$$\llbracket \circ \rrbracket_S = \{\perp, \top\} \quad \llbracket \alpha \rrbracket_S = S(\alpha) \quad \llbracket \sigma \rightarrow \tau \rrbracket_S = \llbracket \sigma \rrbracket_S \rightarrow \llbracket \tau \rrbracket_S$$

where $A \rightarrow B$ denotes the set of (total) functions from A to B .

Definition 3.4 (Model) A *constant model* is a scope-indexed family of functions M_S that map each constant c^σ to a value $M_S(c) \in \llbracket \sigma \rrbracket_S$. A *variable assignment* V for a scope S is a function that maps each variable x^σ to a value $V(x) \in \llbracket \sigma \rrbracket_S$. A *model* for S is a triple $\mathcal{M} = (S, V, M)$, where V is a variable assignment for S and M is a constant model.

Definition 3.5 (Interpretation of Terms) Let $\mathcal{M} = (S, V, M)$ be a model. The *interpretation* $\llbracket t \rrbracket_{\mathcal{M}}$ of a term t in \mathcal{M} is defined recursively by the equations

$$\begin{aligned} \llbracket x \rrbracket_{(S, V, M)} &= V(x) & \llbracket t u \rrbracket_{(S, V, M)} &= \llbracket t \rrbracket_{(S, V, M)} (\llbracket u \rrbracket_{(S, V, M)}) \\ \llbracket c \rrbracket_{(S, V, M)} &= M_S(c) & \llbracket \lambda x^\sigma. t \rrbracket_{(S, V, M)} &= a \in \llbracket \sigma \rrbracket_S \mapsto \llbracket t \rrbracket_{(S, V[x \mapsto a], M)}. \end{aligned}$$

If t is a formula and $\llbracket t \rrbracket_{\mathcal{M}} = \top$, we say that \mathcal{M} is a *model* of t , written $\mathcal{M} \models t$. A formula is *satisfiable* for scope S if it has a model for S .

Convention In the sequel, we always use the standard constant model \widehat{M}_S , which interprets \rightarrow and \simeq in the standard way, allowing us to omit the third component of (S, V, \widehat{M}) .

The remaining connectives and quantifiers are defined as abbreviations in terms of implication and equality. Abbreviations also cater for set-theoretic notations.

Notation 3.6 (Logical Abbreviations)

$$\begin{aligned} \text{True} &\equiv (\lambda x^\circ. x) \simeq (\lambda x. x) & p \wedge q &\equiv \neg(p \rightarrow \neg q) \\ \text{False} &\equiv (\lambda x^\circ. x) \simeq (\lambda x. \text{True}) & p \vee q &\equiv \neg p \rightarrow q \\ \neg p &\equiv p \rightarrow \text{False} & \forall x^\sigma. p &\equiv (\lambda x. p) \simeq (\lambda x. \text{True}) \\ p \not\equiv q &\equiv \neg p \simeq q & \exists x^\sigma. p &\equiv \neg \forall x. \neg p. \end{aligned}$$

Notation 3.7 (Set Abbreviations)

$$\begin{aligned} \emptyset &\equiv \lambda x. \text{False} & s \cap t &\equiv \lambda x. s \wedge t \ x & x \in s &\equiv s \ x \\ \mathcal{U} &\equiv \lambda x. \text{True} & s \cup t &\equiv \lambda x. s \vee t \ x & \text{insert } x \ s &\equiv (\lambda y. y \simeq x) \cup s. \\ & & s - t &\equiv \lambda x. s \wedge \neg t \ x & & \end{aligned}$$

The constants \emptyset and *insert* can be seen as (non-free) constructors for finite sets. Following tradition, we write $\{x_1, \dots, x_m\}$ instead of *insert* x_1 (\dots (*insert* x_m \emptyset) \dots).

4 Monotonicity

Definition 4.1 (Monotonicity) A formula t is *monotonic* with respect to a type variable α if for all scopes S, S' such that $S \leq_\alpha S'$, if t is satisfiable for S , it is also satisfiable for S' . It is *antimonotonic* with respect to α if its negation is monotonic with respect to α .

Example 4.2 If you have five Finnish friends and all five are blond, the existential statement “at least one of your Finnish friends is dark-haired” is monotonic—it will either stay false or become true as you expand your circle of Nordic friends. Inversely, the universal statement “all your Finnish friends are blond” is antimonotonic.

Since monotonicity is a semantic property, it is not surprising that it is undecidable, and the best we can do is approximate it.

Theorem 4.3 (Undecidability) *Monotonicity with respect to α is undecidable.*

Proof (reduction) For any closed HOL formula t , let $t^* \equiv t \vee \forall x^\alpha y. x \simeq y$, where α does not occur in t . Clearly, t^* must be monotonic if t is valid, since the second disjunct becomes irrelevant in this case. If t is not valid, then t^* cannot be monotonic, since it is true for $|\alpha| = 1$ due to the second disjunct but false for some larger scopes. Thus, validity in HOL (which is undecidable) can be reduced to monotonicity. \square

Convention In the sequel, we denote by $\tilde{\alpha}$ the type variable with respect to which we consider monotonicity when not specified otherwise.

5 First Calculus: Tracking Equality and Quantifiers

This section presents the simple calculus \mathfrak{M}_1 for inferring monotonicity, which serves as a stepping stone toward the more general calculi \mathfrak{M}_2 and \mathfrak{M}_3 of Sections 6 and 7. Since the results are fairly intuitive and subsumed by those of the next sections, we omit the proofs.

5.1 Extension Relation and Constancy

We first introduce a concept that is similar to monotonicity but that applies not only to formulas but also to terms of any type—the notion of *constancy*. Informally, a term is constant if it denotes essentially the same value before and after we enlarge the scope. What it means to denote “essentially the same value” can be formalized using an extension relation \sqsubseteq , which relates elements of the smaller scope to elements of the larger scope.

For types such as \circ and $\tilde{\alpha}$, this is easy: Any element of the smaller scope is also present in the larger scope and can serve as an extension. For functions, we expect that the extended function coincides with the original one where applicable; elements not present in the smaller scope may be mapped to any value. For example, when going from $|\tilde{\alpha}| = 1$ to $|\tilde{\alpha}| = 2$, the function $f^{\tilde{\alpha} \rightarrow \circ} = [0 \mapsto \top]$ can be extended to $g = [0 \mapsto \top, 1 \mapsto \perp]$ or $g' = [0 \mapsto \top, 1 \mapsto \top]$. In other words, we take the liberal view that both g and g' are “essentially the same value” as f , and we write $f \sqsubseteq^{\tilde{\alpha} \rightarrow \circ} g$ and $f \sqsubseteq^{\tilde{\alpha} \rightarrow \circ} g'$. We reconsider this decision in Section 6.

Definition 5.1 (Extension) Let σ be a type, and let S, S' be scopes such that $S \leq_{\tilde{\alpha}} S'$. The *extension* relation $\sqsubseteq^\sigma \subseteq \llbracket \sigma \rrbracket_S \times \llbracket \sigma \rrbracket_{S'}$ for S and S' is defined by the following equivalences:

$$\begin{aligned} a \sqsubseteq^\sigma b & \text{ iff } a = b & \text{ if } \sigma \text{ is } \circ \text{ or a type variable} \\ f \sqsubseteq^{\sigma \rightarrow \tau} g & \text{ iff } \forall a b. a \sqsubseteq^\sigma b \longrightarrow f(a) \sqsubseteq^\tau g(b). \end{aligned}$$

The expression $a \sqsubseteq^\sigma b$ is read “ a is extended by b ” or “ b extends a .” The element a is b ’s *restriction* to S , and b is a ’s *extension* to S' . In addition, we will refer to elements $b \in \llbracket \sigma \rrbracket_{S'}$ as being *old* if they admit a restriction to S and *new* if they do not admit any.

Figure 5.1 illustrates \sqsubseteq^σ for various types. We represent a function from σ to τ by a $|\sigma|$ -tuple such that the n th element for σ (according to the lexicographic order, with $\perp < \top$ and $n < n + 1$) is mapped to the n th tuple component. Observe that \sqsubseteq^σ is always left-total (i.e., total: $\forall a. \exists b. a \sqsubseteq^\sigma b$) and left-unique (i.e., injective: $\forall a a' b. a \sqsubseteq^\sigma b \wedge a' \sqsubseteq^\sigma b \longrightarrow a = a'$).

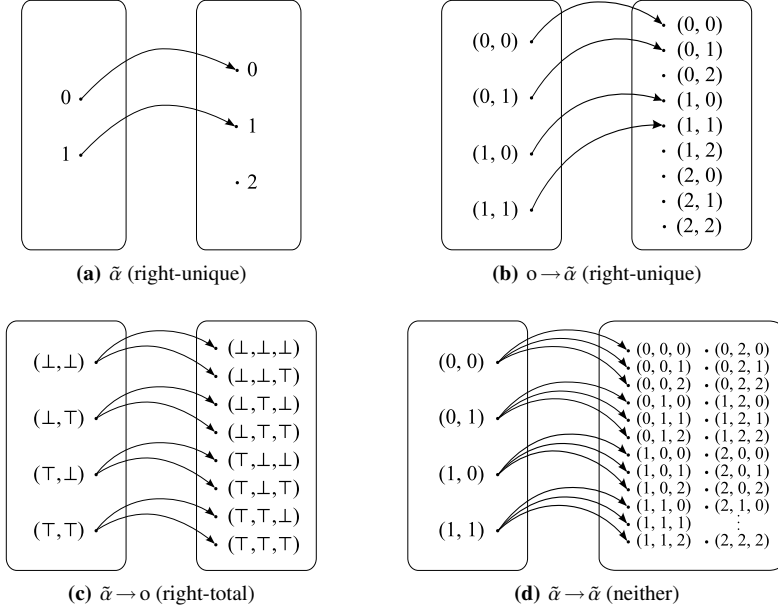


Figure 5.1. \sqsubseteq^σ for various types σ , with $|S(\tilde{\alpha})| = 2$ and $|S'(\tilde{\alpha})| = 3$

It is also right-total (i.e., surjective: $\forall b. \exists a. a \sqsubseteq^\sigma b$) if $\tilde{\alpha}$ does not occur positively in σ (e.g., $\sigma = \tilde{\alpha} \rightarrow o$), and right-unique (i.e., functional: $\forall a b b'. a \sqsubseteq^\sigma b \wedge a \sqsubseteq^\sigma b' \longrightarrow b = b'$) if $\tilde{\alpha}$ does not occur negatively (e.g., $\sigma = o \rightarrow \tilde{\alpha}$). These properties are crucial to the correctness of our calculus, which restricts where $\tilde{\alpha}$ may occur. They are proved in Section 6.

Definition 5.2 (Model Extension) Let $\mathcal{M} = (S, V)$ and $\mathcal{M}' = (S', V')$ be models. The model \mathcal{M}' extends \mathcal{M} , written $\mathcal{M} \sqsubseteq \mathcal{M}'$, if $S \leq_{\tilde{\alpha}} S'$ and $V(x) \sqsubseteq^\sigma V'(x)$ for all x^σ .

The relation \sqsubseteq on models gives us a recipe to transform a smaller model \mathcal{M} into a larger model \mathcal{M}' . Because \sqsubseteq^σ is left-total, the recipe always works.

Definition 5.3 (Constancy) A term t^σ is constant if $\llbracket t \rrbracket_{\mathcal{M}} \sqsubseteq^\sigma \llbracket t \rrbracket_{\mathcal{M}'}$ for all models $\mathcal{M}, \mathcal{M}'$ such that $\mathcal{M} \sqsubseteq \mathcal{M}'$.

Example 5.4 $f^{\tilde{\alpha} \rightarrow \tilde{\alpha}} x$ is constant. Proof: Let $V(x) = a_1$ and $V(f)(a_1) = a_2$. For any $\mathcal{M}' = (S', V')$ that extends $\mathcal{M} = (S, V)$, we have $V(x) \sqsubseteq^{\tilde{\alpha}} V'(x)$ and $V(f) \sqsubseteq^{\tilde{\alpha} \rightarrow \tilde{\alpha}} V'(f)$. By definition of \sqsubseteq^σ , $V'(x) = a_1$ and $V'(f)(a_1) = a_2$. Thus, $\llbracket f x \rrbracket_{\mathcal{M}} = \llbracket f x \rrbracket_{\mathcal{M}'} = a_2$.

Example 5.5 $f^{o \rightarrow \tilde{\alpha}} \simeq g$ is constant. Proof: For any $\mathcal{M}' = (S', V')$ that extends $\mathcal{M} = (S, V)$, we have $V(f) \sqsubseteq^{o \rightarrow \tilde{\alpha}} V'(f)$ and $V(g) \sqsubseteq^{o \rightarrow \tilde{\alpha}} V'(g)$. By definition of \sqsubseteq^σ , $V'(f) = V(f)$ and $V'(g) = V(g)$. Hence, $\llbracket f \simeq g \rrbracket_{\mathcal{M}} = \llbracket f \simeq g \rrbracket_{\mathcal{M}'}$.

Example 5.6 $p^{\tilde{\alpha} \rightarrow o} \simeq q$ is not constant. Counterexample: $|S(\tilde{\alpha})| = 1$, $V(p) = V(q) = (\top)$, $|S'(\tilde{\alpha})| = 2$, $V'(p) = (\top, \perp)$, $V'(q) = (\top, \top)$. Then $\llbracket p \simeq q \rrbracket_{(S, V)} = \top$ but $\llbracket p \simeq q \rrbracket_{(S', V')} = \perp$.

More generally, we note that variables are always constant, and constancy is preserved by λ -abstraction and application. On the other hand, the equality symbol $\simeq^{\sigma \rightarrow \sigma \rightarrow o}$ is constant only if $\tilde{\alpha}$ does not occur negatively in σ . Moreover, since \sqsubseteq^o is the identity relation, constant formulas are both monotonic and antimonotonic.

Remark Relations between models of the λ -calculus that are preserved under abstraction and application are called *logical relations* [13] and are widely used in semantics and model theory. If we had no equality, \sqsubseteq would be a logical relation, and constancy of all terms would follow from the “Basic Lemma,” which states that the interpretations of any term are related by \sim if \sim is a logical relation. This property is spoiled by equality since in general $\llbracket \simeq \rrbracket_{\mathcal{M}} \not\sqsubseteq \llbracket \simeq \rrbracket_{\mathcal{M}'}$. Our calculus effectively carves out a sublanguage for which \sqsubseteq is a logical relation.

5.2 Syntactic Criteria

We syntactically approximate constancy, monotonicity, and antimonotonicity with the predicates $K(t)$, $M^+(t)$, and $M^-(t)$, respectively. The goal is to derive $M^+(t)$ for the formula t we wish to prove monotonic. If $M^+(t)$ and the model \mathcal{M} satisfies t , we can apply our recipe \sqsubseteq to obtain arbitrarily larger models \mathcal{M}' that also satisfy t . The predicates depend on the auxiliary functions $TV^+(\sigma)$ and $TV^-(\sigma)$, which collect the positive and negative type variables of σ .

Definition 5.7 (Positive and Negative Type Variables) The set of *positive type variables* $TV^+(\sigma)$ and the set of *negative type variables* $TV^-(\sigma)$ of a type σ are defined as follows:

$$\begin{array}{l} TV^+(\alpha) = \{\alpha\} \quad TV^s(o) = \emptyset \\ TV^-(\alpha) = \emptyset \quad TV^s(\sigma \rightarrow \tau) = TV^{\sim s}(\sigma) \cup TV^s(\tau) \end{array} \quad \text{where } \sim s = \begin{cases} + & \text{if } s = - \\ - & \text{if } s = +. \end{cases}$$

Definition 5.8 (Constancy and Monotonicity Rules) The predicates $K(t)$, $M^+(t)$, and $M^-(t)$ are inductively defined by the rules

$$\begin{array}{c} \frac{}{K(x)} \quad \frac{}{K(\longrightarrow)} \quad \frac{\tilde{\alpha} \notin TV^-(\sigma)}{K(\simeq^{\sigma \rightarrow \sigma \rightarrow o})} \quad \frac{K(t)}{K(\lambda x. t)} \quad \frac{K(t^{\sigma \rightarrow \tau}) \quad K(u^\sigma)}{K(t u)} \\ \frac{K(t)}{M^s(t^o)} \quad \frac{M^{\sim s}(t) \quad M^s(u)}{M^s(t \longrightarrow u)} \quad \frac{K(t^\sigma) \quad K(u^\sigma)}{M^-(t \simeq u)} \quad \frac{M^-(t)}{M^-(\forall x. t)} \quad \frac{M^+(t) \quad \tilde{\alpha} \notin TV^+(\sigma)}{M^+(\forall x^\sigma. t)}. \end{array}$$

The rules for K simply traverse the term structure and ensure that equality is not used on types in which $\tilde{\alpha}$ occurs negatively. The first two rules for M^s are easy to justify semantically. The other three are more subtle:

- The $M^-(t \simeq u)$ rule is sound because the extensions of distinct elements are always distinct (since \sqsubseteq^σ is left-unique).
- The $M^-(\forall x. t)$ rule is sound because if enlarging the scope makes x range over new elements, these cannot make $\forall x. t$ become true if it was false in the smaller scope.
- The $M^+(\forall x. t)$ rule is the most difficult one. If $\tilde{\alpha}$ does not occur at all in σ , then monotonicity is preserved. Otherwise, there is the danger that the formula t is true for all values $a \in \llbracket \sigma \rrbracket_S$ but not for some $b \in \llbracket \sigma \rrbracket_{S'}$. However, in Section 6 we show that this can only happen for b 's that do not extend any a , which can only exist if $\tilde{\alpha} \in TV^+(\sigma)$.

Definition 5.9 (Derived Monotonicity Rules) For logical abbreviations, we can derive the following rules using Notation 3.6 and Definition 5.8:

$$\begin{array}{c} \frac{}{M^s(False)} \quad \frac{}{M^s(True)} \quad \frac{M^{\sim s}(t)}{M^s(\neg t)} \quad \frac{M^s(t) \quad M^s(u)}{M^s(t \wedge u)} \\ \frac{M^s(t) \quad M^s(u)}{M^s(t \vee u)} \quad \frac{M^+(t)}{M^+(\exists x. t)} \quad \frac{M^-(t) \quad \tilde{\alpha} \notin TV^+(\sigma)}{M^-(\exists x^\sigma. t)}. \end{array}$$

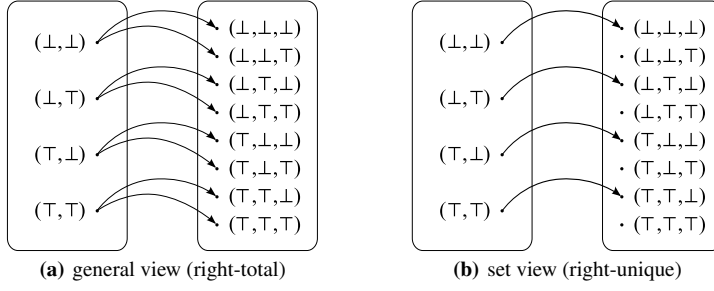


Figure 6.1. $\sqsubseteq^{\bar{\alpha} \rightarrow \circ}$ with $S(\bar{\alpha}) = 2$ and $S'(\bar{\alpha}) = 3$

Definition 6.1 (Annotated Type) An *annotated type* is a HOL type in which each function arrow carries an *annotation* $A \in \{G, F\}$.

The annotations specify how \sqsubseteq should extend function values to larger scopes: While G-functions are extended as in the previous section, the extension of an F-function must map all new values to \perp . The annotations have no influence on the interpretation of types and terms, which is unchanged. For notational convenience, we sometimes use annotated types in contexts where plain types are expected; in such cases, the annotations are simply ignored.

6.1 Extension Relation

Definition 6.2 (Extension) Let σ be an annotated type, and let S, S' be scopes such that $S \leq_{\bar{\alpha}} S'$. The *extension relation* $\sqsubseteq^{\sigma} \subseteq \llbracket \sigma \rrbracket_S \times \llbracket \sigma \rrbracket_{S'}$ for S and S' is defined by the following equivalences:

$$\begin{aligned}
a \sqsubseteq^{\sigma} b & \text{ iff } a = b & \text{ if } \sigma \text{ is } \circ \text{ or a type variable} \\
f \sqsubseteq^{\sigma \rightarrow G \tau} g & \text{ iff } \forall ab. a \sqsubseteq^{\sigma} b \longrightarrow f(a) \sqsubseteq^{\tau} g(b) \\
f \sqsubseteq^{\sigma \rightarrow F \tau} g & \text{ iff } \forall ab. a \sqsubseteq^{\sigma} b \longrightarrow f(a) \sqsubseteq^{\tau} g(b) \text{ and } \forall b. (\nexists a. a \sqsubseteq^{\sigma} b) \longrightarrow g(b) = \langle \tau \rangle
\end{aligned}$$

where $\langle \circ \rangle = \perp$, $\langle \sigma \rightarrow \tau \rangle = a \in \llbracket \sigma \rrbracket_{S'} \mapsto \langle \tau \rangle$, and $\langle \alpha \rangle$ is any element of $S(\alpha)$.

The extension relation \sqsubseteq^{σ} distinguishes between the two kinds of arrows. The G case coincides with Definition 5.1. Although F is tailored to predicates, the annotated type $\sigma \rightarrow_F \tau$ is legal for any type τ . The value $\langle \tau \rangle$ then takes the place of \perp as the default extension.

We now prove the crucial properties of \sqsubseteq^{σ} , which we stated in Section 5 for the G case.

Lemma 6.3 *The relation \sqsubseteq^{σ} is left-total (total) and left-unique (injective).*

Proof (structural induction on σ) For \circ and α , both properties are obvious. For $\sigma \rightarrow_A \tau$, \sqsubseteq^{σ} and \sqsubseteq^{τ} are left-unique and left-total by induction hypothesis. Since $\sqsubseteq^{\sigma \rightarrow F \tau} \subseteq \sqsubseteq^{\sigma \rightarrow G \tau}$ by definition, it suffices to show that $\sqsubseteq^{\sigma \rightarrow F \tau}$ is left-total and $\sqsubseteq^{\sigma \rightarrow G \tau}$ is left-unique.

LEFT-TOTALITY: For $f \in \llbracket \sigma \rightarrow \tau \rrbracket_S$, we find an extension g as follows: Let $b \in \llbracket \sigma \rrbracket_{S'}$. If b extends an a , that a is unique by left-uniqueness of \sqsubseteq^{σ} . Since \sqsubseteq^{τ} is left-total, there exists a y such that $f(a) \sqsubseteq^{\tau} y$, and we let $g(b) = y$. If b does not extend any a , then we set $g(b) = \langle \tau \rangle$. By construction, $f \sqsubseteq^{\sigma \rightarrow F \tau} g$.

LEFT-UNIQUENESS: We assume $f, f' \sqsubseteq^{\sigma \rightarrow G \tau} g$ and show that $f = f'$. For every $a \in \llbracket \sigma \rrbracket_S$, left-totality of \sqsubseteq^{σ} yields an extension $b \sqsubseteq^{\sigma} a$. Then $f(a) \sqsubseteq^{\tau} g(b)$ and $f'(a) \sqsubseteq^{\tau} g(b)$, and since \sqsubseteq^{τ} is left-unique, $f(a) = f'(a)$. \square

Definition 6.4 (Positive and Negative Type Variables) The set of *positive type variables* $\text{TV}^+(\sigma)$ and the set of *negative type variables* $\text{TV}^-(\sigma)$ of an annotated type σ are defined as follows:

$$\begin{aligned} \text{TV}^+(\mathfrak{o}) &= \emptyset & \text{TV}^-(\mathfrak{o}) &= \emptyset \\ \text{TV}^+(\alpha) &= \{\alpha\} & \text{TV}^-(\alpha) &= \emptyset \\ \text{TV}^+(\sigma \rightarrow_{\mathfrak{G}} \tau) &= \text{TV}^+(\tau) \cup \text{TV}^-(\sigma) & \text{TV}^-(\sigma \rightarrow_{\mathfrak{G}} \tau) &= \text{TV}^-(\tau) \cup \text{TV}^+(\sigma) \\ \text{TV}^+(\sigma \rightarrow_{\mathfrak{F}} \tau) &= \text{TV}^+(\tau) \cup \text{TV}^-(\sigma) \cup \text{TV}^+(\sigma) & \text{TV}^-(\sigma \rightarrow_{\mathfrak{F}} \tau) &= \text{TV}^-(\tau). \end{aligned}$$

This rather unusual generalization of Definition 5.7 reflects our wish to treat occurrences of $\tilde{\alpha}$ differently for sets and ensures that the following key lemma holds uniformly.

Lemma 6.5 *If $\tilde{\alpha} \notin \text{TV}^+(\sigma)$, then \sqsubseteq^σ is right-total (surjective). If $\tilde{\alpha} \notin \text{TV}^-(\sigma)$, then \sqsubseteq^σ is right-unique (functional).*

Proof (structural induction on σ) For \mathfrak{o} and α , both properties are obvious.

RIGHT-TOTALITY OF $\sqsubseteq^{\sigma \rightarrow_{\mathfrak{G}} \tau}$: If $\tilde{\alpha} \notin \text{TV}^+(\sigma \rightarrow_{\mathfrak{G}} \tau) = \text{TV}^+(\tau) \cup \text{TV}^-(\sigma)$, then by induction hypothesis \sqsubseteq^τ is right-total and \sqsubseteq^σ is right-unique. For $g \in \llbracket \sigma \rightarrow \tau \rrbracket_{S'}$, we find a restriction f as follows: Let $a \in \llbracket \sigma \rrbracket_S$. Since \sqsubseteq^σ is both left-total (Lemma 6.3) and right-unique, there is a unique b such that $a \sqsubseteq^\sigma b$. By right-totality of \sqsubseteq^τ , we obtain an $x \sqsubseteq^\tau b$, and we set $f(a) = x$. By construction, $f \sqsubseteq^{\sigma \rightarrow_{\mathfrak{G}} \tau} g$.

RIGHT-TOTALITY OF $\sqsubseteq^{\sigma \rightarrow_{\mathfrak{F}} \tau}$: If $\tilde{\alpha} \notin \text{TV}^+(\sigma \rightarrow_{\mathfrak{F}} \tau) = \text{TV}^+(\tau) \cup \text{TV}^-(\sigma) \cup \text{TV}^+(\sigma)$, then by induction hypothesis \sqsubseteq^τ is right-total, and \sqsubseteq^σ is both right-total and right-unique. By right-totality of \sqsubseteq^σ , the second condition in the definition of $\sqsubseteq^{\sigma \rightarrow_{\mathfrak{F}} \tau}$ becomes vacuous, and $\sqsubseteq^{\sigma \rightarrow_{\mathfrak{F}} \tau} = \sqsubseteq^{\sigma \rightarrow_{\mathfrak{G}} \tau}$, whose right-totality was shown above.

RIGHT-UNIQUENESS OF $\sqsubseteq^{\sigma \rightarrow_{\mathfrak{G}} \tau}$: If $\tilde{\alpha} \notin \text{TV}^-(\sigma \rightarrow_{\mathfrak{G}} \tau) = \text{TV}^-(\tau) \cup \text{TV}^+(\sigma)$, then by induction hypothesis \sqsubseteq^τ is right-unique and \sqsubseteq^σ is right-total. We consider g, g' such that $f \sqsubseteq^{\sigma \rightarrow_{\mathfrak{G}} \tau} g$ and $f \sqsubseteq^{\sigma \rightarrow_{\mathfrak{G}} \tau} g'$, and show that $g = g'$. For every $b \in \llbracket \sigma \rrbracket_{S'}$, right-totality of \sqsubseteq^σ yields a restriction $a \sqsubseteq^\sigma b$. Then $f(a) \sqsubseteq^\tau g(b)$ and $f(a) \sqsubseteq^\tau g'(b)$, and since \sqsubseteq^τ is right-unique, $g(b) = g'(b)$.

RIGHT-UNIQUENESS OF $\sqsubseteq^{\sigma \rightarrow_{\mathfrak{F}} \tau}$: If $\tilde{\alpha} \notin \text{TV}^-(\sigma \rightarrow_{\mathfrak{F}} \tau) = \text{TV}^-(\tau)$, then by induction hypothesis \sqsubseteq^τ is right-unique. We consider g, g' such that $f \sqsubseteq^{\sigma \rightarrow_{\mathfrak{F}} \tau} g$ and $f \sqsubseteq^{\sigma \rightarrow_{\mathfrak{F}} \tau} g'$, and show that $g = g'$. For any $b \in \llbracket \sigma \rrbracket_{S'}$, if there exists no restriction $a \sqsubseteq^\sigma b$, then by definition $g(b) = g'(b) = \langle \tau \rangle$. Otherwise, we assume $a \sqsubseteq^\sigma b$. Then $f(a) \sqsubseteq^\tau g(b)$ and $f(a) \sqsubseteq^\tau g'(b)$, and since \sqsubseteq^τ is right-unique, $g(b) = g'(b)$. \square

The new definition of TV^+ and TV^- solves the problem raised by $\{y\} \simeq \{z\}$ (Example 5.11), since the $\tilde{\alpha}$ in $\tilde{\alpha} \rightarrow_{\mathfrak{F}} \mathfrak{o}$ counts as a positive occurrence. However, we ensure that types are consistently annotated; otherwise, we could easily overconstrain the free variables and end up in a situation where there exists no model \mathcal{M}' such that $\mathcal{M} \sqsubseteq \mathcal{M}'$ for two scopes $S \leq_{\tilde{\alpha}} S'$.

6.2 Type Checking

Checking constancy can be seen as a type checking problem involving annotated types. The basic idea is to derive typing judgments $\Gamma \vdash t : \sigma$, whose intuitive meaning is that the denotations of t in a smaller and a larger scope are related by \sqsubseteq^σ (i.e., that t is constant in a sense given by σ). Despite this new interpretation, the typing rules are similar to those of the simply-typed λ -calculus, extended with a particular form of subtyping.

Definition 6.6 (Context) A *context* is a pair of mappings $\Gamma = (\Gamma_c, \Gamma_v)$, where Γ_c maps constant symbols to sets of annotated types, and Γ_v maps variables to annotated types.

Allowing constants to have multiple annotated types gives us a form of polymorphism on the annotations, which is sometimes useful.

Definition 6.7 (Compatibility) A constant context Γ_c is *compatible* with a constant model M if $\sigma \in \Gamma_c(c)$ implies $M_S(c) \sqsubseteq^\sigma M_{S'}(c)$ for all scopes S, S' with $S \leq_{\tilde{\alpha}} S'$ and for all constants c and annotated types σ .

Convention In the sequel, we always use a fixed constant context Γ_c compatible with the standard constant model \widehat{M} , allowing us to omit the first component of $\Gamma = (\Gamma_c, \Gamma_v)$.

Definitions 5.2 and 5.3 and the K part of Definition 5.8 are generalized as follows.

Definition 6.8 (Model Extension) Let $\mathcal{M} = (S, V)$ and $\mathcal{M}' = (S', V')$ be models. The model \mathcal{M}' *extends* \mathcal{M} in a context Γ , written $\mathcal{M} \sqsubseteq_\Gamma \mathcal{M}'$, if $S \leq_{\tilde{\alpha}} S'$ and $\Gamma(x) = \sigma$ implies $V(x) \sqsubseteq^\sigma V'(x)$ for all x .

Definition 6.9 (Constancy) Let σ be an annotated type. A term t is σ -*constant* in a context Γ if $\llbracket t \rrbracket_{\mathcal{M}} \sqsubseteq^\sigma \llbracket t \rrbracket_{\mathcal{M}'}$ for all models $\mathcal{M}, \mathcal{M}'$ such that $\mathcal{M} \sqsubseteq_\Gamma \mathcal{M}'$.

Definition 6.10 (Typing Rules) The *typing* relation $\Gamma \vdash t : \sigma$ is given by the rules

$$\begin{array}{c} \frac{\Gamma(x) = \sigma}{\Gamma \vdash x : \sigma} \text{VAR} \quad \frac{\sigma \in \Gamma_c(c)}{\Gamma \vdash c : \sigma} \text{CONST} \quad \frac{\Gamma \vdash t : \sigma \quad \sigma \leq \sigma'}{\Gamma \vdash t : \sigma'} \text{SUB} \\ \\ \frac{\Gamma[x \mapsto \sigma] \vdash t : \tau}{\Gamma \vdash \lambda x. t : \sigma \rightarrow_G \tau} \text{LAM} \quad \frac{\Gamma \vdash t : \sigma \rightarrow_A \tau \quad \Gamma \vdash u : \sigma}{\Gamma \vdash t u : \tau} \text{APP} \end{array}$$

where the *subtype* relation $\sigma \leq \tau$ is defined by the rules

$$\frac{}{\sigma \leq \sigma} \text{REFL} \quad \frac{\sigma' \leq \sigma \quad \tau \leq \tau'}{\sigma \rightarrow_A \tau \leq \sigma' \rightarrow_G \tau'} \text{GEN} \quad \frac{\sigma' \leq \sigma \quad \sigma \leq \sigma' \quad \tau \leq \tau'}{\sigma \rightarrow_F \tau \leq \sigma' \rightarrow_F \tau'} \text{FALSE.}$$

Lemma 6.11 *If $\sigma \leq \sigma'$, then $\sqsubseteq^\sigma \subseteq \sqsubseteq^{\sigma'}$.*

Proof (induction on the derivation of $\sigma \leq \sigma'$) The REFL case is trivial.

GEN: We may assume $\sqsubseteq^{\sigma'} \subseteq \sqsubseteq^\sigma$ and $\sqsubseteq^\tau \subseteq \sqsubseteq^{\tau'}$. Since $\sqsubseteq^{\sigma \rightarrow_A \tau} \subseteq \sqsubseteq^{\sigma \rightarrow_G \tau}$ (by Definition 6.2), it is sufficient to consider the case $A = F$. If $f \sqsubseteq^{\sigma \rightarrow_G \tau} g$, we conclude $f \sqsubseteq^{\sigma' \rightarrow_G \tau'} g$ because $a \sqsubseteq^{\sigma'} b \implies a \sqsubseteq^\sigma b \implies f(a) \sqsubseteq^\tau g(b) \implies f(a) \sqsubseteq^{\tau'} g(b)$.

FALSE: We may assume $\sqsubseteq^\sigma = \sqsubseteq^{\sigma'}$ and $\sqsubseteq^\tau \subseteq \sqsubseteq^{\tau'}$. If $f \sqsubseteq^{\sigma \rightarrow_F \tau} g$, the first condition for $f \sqsubseteq^{\sigma' \rightarrow_F \tau'} g$ follows with the same reasoning as above. The second condition holds since $\sqsubseteq^\sigma = \sqsubseteq^{\sigma'}$. \square

As a consequence of Lemma 6.11, \leq is a partial quasiorder. Observe that $\sigma \rightarrow_F 0 \leq \sigma \rightarrow_G 0$ for any σ , and if $\tilde{\alpha}$ does not occur in σ we also have $\sigma \rightarrow_G 0 \leq \sigma \rightarrow_F 0$. On the other hand, $(\tilde{\alpha} \rightarrow_G 0) \rightarrow_G 0$ and $(\tilde{\alpha} \rightarrow_F 0) \rightarrow_F 0$ are not related, so the quasiorder \leq is not total (linear).

Theorem 6.12 (Soundness of Typing) *If $\Gamma \vdash t : \sigma$, then t is σ -constant in Γ .*

Proof (induction on the derivation of $\Gamma \vdash t : \sigma$)

VAR: Because $V(x) \sqsubseteq^\sigma V'(x)$ by assumption for $\sigma = \Gamma(x)$.

CONST: By compatibility of Γ_c , $\widehat{M}_S(c) \sqsubseteq^\sigma \widehat{M}_{S'}(c)$ for all $\sigma \in \Gamma_c(c)$.

SUB: By Lemma 6.11 and Definition 6.9.

LAM: Let $a \in \llbracket \sigma \rrbracket_S$ and $b \in \llbracket \sigma \rrbracket_{S'}$ such that $a \sqsubseteq^\sigma b$. Then we have the extended models $\mathcal{M}_a = (S, V[x \mapsto a])$ and $\mathcal{M}'_b = (S', V'[x \mapsto b])$. Thus, $\mathcal{M}_a \sqsubseteq_{\Gamma[x \mapsto \sigma]} \mathcal{M}'_b$, and by induction hypothesis $\llbracket \lambda x. t \rrbracket_{\mathcal{M}_a}(a) = \llbracket t \rrbracket_{\mathcal{M}_a} \sqsubseteq^\tau \llbracket t \rrbracket_{\mathcal{M}'_b} = \llbracket \lambda x. t \rrbracket_{\mathcal{M}'_b}(b)$. Hence $\llbracket \lambda x. t \rrbracket_{\mathcal{M}} \sqsubseteq^{\sigma \rightarrow \mathbf{G}^\tau} \llbracket \lambda x. t \rrbracket_{\mathcal{M}'}$.

APP: By induction hypothesis, and since $\sqsubseteq^{\sigma \rightarrow \mathbf{F}^\tau} \subseteq \sqsubseteq^{\sigma \rightarrow \mathbf{G}^\tau}$, we have $\llbracket t \rrbracket_{\mathcal{M}} \sqsubseteq^{\sigma \rightarrow \mathbf{G}^\tau} \llbracket t \rrbracket_{\mathcal{M}'}$ and $\llbracket u \rrbracket_{\mathcal{M}} \sqsubseteq^\sigma \llbracket u \rrbracket_{\mathcal{M}'}$. By Definition 6.2, we know that $\llbracket t u \rrbracket_{\mathcal{M}} = \llbracket t \rrbracket_{\mathcal{M}}(\llbracket u \rrbracket_{\mathcal{M}}) \sqsubseteq^\tau \llbracket t \rrbracket_{\mathcal{M}'}(\llbracket u \rrbracket_{\mathcal{M}'}) = \llbracket t u \rrbracket_{\mathcal{M}'}$, which shows that $t u$ is τ -constant in Γ . \square

While our typing rules propagate F-annotations nicely, they cannot derive them, since the LAM rule annotates all arrows with G. In particular, the basic set operations \emptyset , \cup , \cap , and $-$ cannot be typed appropriately from their definitions (Notation 3.7). We solve this issue pragmatically by treating common set operations as primitive constants along with implication and equality. The typing rules then propagate type information through expressions such as $A \cup (B \cap C)$. We address this limitation more generally in Section 7.

Definition 6.13 (Standard Constant Context) The *standard constant context* $\widehat{\Gamma}_c$ is the following mapping:

\longrightarrow	\mapsto	$\{\mathbf{o} \rightarrow_{\mathbf{G}} \mathbf{o} \rightarrow_{\mathbf{G}} \mathbf{o}\}$	
\simeq	\mapsto	$\{\sigma \rightarrow_{\mathbf{G}} \sigma \rightarrow_{\mathbf{F}} \mathbf{o}\}$	$ \ \tilde{\alpha} \notin \text{TV}^-(\sigma)\}$
\emptyset	\mapsto	$\{\sigma \rightarrow_{\mathbf{F}} \mathbf{o}\}$	
\mathcal{U}	\mapsto	$\{\sigma \rightarrow_{\mathbf{G}} \mathbf{o}\}$	
\cup	\mapsto	$\{(\sigma \rightarrow_{\mathbf{A}} \mathbf{o}) \rightarrow_{\mathbf{G}} (\sigma \rightarrow_{\mathbf{A}} \mathbf{o}) \rightarrow_{\mathbf{G}} \sigma \rightarrow_{\mathbf{A}} \mathbf{o}\}$	$ \ A \in \{\mathbf{G}, \mathbf{F}\}\}$
\cap	\mapsto	$\{(\sigma \rightarrow_{\mathbf{A}} \mathbf{o}) \rightarrow_{\mathbf{G}} (\sigma \rightarrow_{\mathbf{A}} \mathbf{o}) \rightarrow_{\mathbf{G}} \sigma \rightarrow_{\mathbf{A}} \mathbf{o}\}$	$ \ A \in \{\mathbf{G}, \mathbf{F}\}\}$
$-$	\mapsto	$\{(\sigma \rightarrow_{\mathbf{A}} \mathbf{o}) \rightarrow_{\mathbf{G}} (\sigma \rightarrow_{\mathbf{G}} \mathbf{o}) \rightarrow_{\mathbf{G}} \sigma \rightarrow_{\mathbf{A}} \mathbf{o}\}$	$ \ A \in \{\mathbf{G}, \mathbf{F}\}\}$
\in	\mapsto	$\{\sigma \rightarrow_{\mathbf{G}} (\sigma \rightarrow_{\mathbf{A}} \mathbf{o}) \rightarrow_{\mathbf{G}} \mathbf{o}\}$	$ \ A \in \{\mathbf{G}, \mathbf{F}\}\}$
<i>insert</i>	\mapsto	$\{\sigma \rightarrow_{\mathbf{G}} (\sigma \rightarrow_{\mathbf{A}} \mathbf{o}) \rightarrow_{\mathbf{G}} \sigma \rightarrow_{\mathbf{A}} \mathbf{o}\}$	$ \ A \in \{\mathbf{G}, \mathbf{F}\}, \tilde{\alpha} \notin \text{TV}^-(\sigma)\}$.

Notice how the lack of a specific annotation for “true-leaning sets” prevents us from giving precise typings to the complement of an F-annotated function; for example, \emptyset is captured precisely by $\sigma \rightarrow_{\mathbf{F}} \mathbf{o}$, but \mathcal{U} can be typed only as $\sigma \rightarrow_{\mathbf{G}} \mathbf{o}$. In Section 7, we introduce a T-annotation similar to F but with \top instead of \perp as the default extension.

Lemma 6.14 *The standard constant context $\widehat{\Gamma}_c$ is compatible with the standard constant model \widehat{M} .*

Proof For space reasons, we only show the proof for equality.

CASE \simeq : Since $\tilde{\alpha} \notin \text{TV}^-(\sigma)$, \sqsubseteq^σ is right-unique (Lemma 6.5). Unfolding the definition of \sqsubseteq , we assume $a \sqsubseteq^\sigma b$ and show that if $a' \sqsubseteq^\sigma b'$, then $(a = a') = (b = b')$, and that if there exists no restriction a' such that $a' \sqsubseteq^\sigma b'$, then $(b = b') = \perp$. The first part follows from the left-uniqueness and right-uniqueness of \sqsubseteq^σ . For the second part, $b \neq b'$ because b extends a while b' admits no restriction. \square

Example 6.15 Let $\sigma = \alpha \rightarrow_{\mathbf{F}} \mathbf{o}$ and $\Gamma_v = [x \mapsto \sigma, y \mapsto \sigma]$. The following derivation shows that $x^{\alpha \rightarrow \mathbf{o}} \simeq y$ is constant with respect to α :

$$\frac{\frac{\frac{\sigma \rightarrow_{\mathbf{G}} \sigma \rightarrow_{\mathbf{F}} \mathbf{o} \in \Gamma_c(\simeq)}{\Gamma \vdash (\simeq) : \sigma \rightarrow_{\mathbf{G}} \sigma \rightarrow_{\mathbf{F}} \mathbf{o}} \text{CONST} \quad \frac{\Gamma(x) = \sigma}{\Gamma \vdash x : \sigma} \text{VAR}}{\Gamma \vdash (\simeq) x : \sigma \rightarrow_{\mathbf{G}} \mathbf{o}} \text{APP} \quad \frac{\Gamma(y) = \sigma}{\Gamma \vdash y : \sigma} \text{VAR}}{\Gamma \vdash x \simeq y : \mathbf{o}} \text{APP}$$

6.3 Monotonicity Checking

The rules for checking monotonicity and antimonotonicity are almost the same as in Section 5, except that they now extend the context when moving under a quantifier.

Definition 6.16 (Monotonicity Rules) The predicates $\Gamma \vdash M^+(t)$ and $\Gamma \vdash M^-(t)$ are given by the rules

$$\frac{\Gamma \vdash t : \circ}{\Gamma \vdash M^s(t)} \text{TERM} \quad \frac{\Gamma \vdash M^{\sim s}(t) \quad \Gamma \vdash M^s(u)}{\Gamma \vdash M^s(t \longrightarrow u)} \text{IMP} \quad \frac{\Gamma \vdash t : \sigma \quad \Gamma \vdash u : \sigma}{\Gamma \vdash M^-(t \simeq u)} \text{EQ}^-$$

$$\frac{\Gamma[x \mapsto \sigma] \vdash M^-(t)}{\Gamma \vdash M^-(\forall x. t)} \text{ALL}^- \quad \frac{\Gamma[x \mapsto \sigma] \vdash M^+(t) \quad \tilde{\alpha} \notin \text{TV}^+(\sigma)}{\Gamma \vdash M^+(\forall x. t)} \text{ALL}^+.$$

Definition 6.17 (Derived Monotonicity Rules) From Notation 3.6 and Definitions 6.10 and 6.16, we derive these rules for logical abbreviations:

$$\frac{}{\Gamma \vdash M^s(\text{False})} \text{FALSE} \quad \frac{}{\Gamma \vdash M^s(\text{True})} \text{TRUE} \quad \frac{\Gamma \vdash M^{\sim s}(t)}{\Gamma \vdash M^s(\neg t)} \text{NOT}$$

$$\frac{\Gamma \vdash M^s(t) \quad \Gamma \vdash M^s(u)}{\Gamma \vdash M^s(t \wedge u)} \text{AND} \quad \frac{\Gamma \vdash M^s(t) \quad \Gamma \vdash M^s(u)}{\Gamma \vdash M^s(t \vee u)} \text{OR}$$

$$\frac{\Gamma[x \mapsto \sigma] \vdash M^+(t)}{\Gamma \vdash M^+(\exists x^\sigma. t)} \text{EX}^+ \quad \frac{\Gamma[x \mapsto \sigma] \vdash M^-(t) \quad \tilde{\alpha} \notin \text{TV}^+(\sigma)}{\Gamma \vdash M^-(\exists x^\sigma. t)} \text{EX}^-.$$

Theorem 6.18 (Soundness of M^s) Let \mathcal{M} and \mathcal{M}' be models such that $\mathcal{M} \sqsubseteq_\Gamma \mathcal{M}'$. If $\Gamma \vdash M^+(t)$, then $\mathcal{M} \models t$ implies $\mathcal{M}' \models t$. If $\Gamma \vdash M^-(t)$, then $\mathcal{M} \not\models t$ implies $\mathcal{M}' \not\models t$.

Proof (induction on the derivation of $\Gamma \vdash M^s(t)$) The TERM and IMP cases are obvious. Let $\mathcal{M} = (S, V)$ and $\mathcal{M}' = (S', V')$.

EQ⁻: Assume that $\Gamma \vdash t : \sigma$, $\Gamma \vdash u : \sigma$, and $\mathcal{M} \not\models t \simeq u$. Since \mathcal{M} is a standard model, we know that $\llbracket t \rrbracket_{\mathcal{M}} \neq \llbracket u \rrbracket_{\mathcal{M}}$. By Theorem 6.12, we have $\llbracket t \rrbracket_{\mathcal{M}} \sqsubseteq^\sigma \llbracket t \rrbracket_{\mathcal{M}'}$ and $\llbracket u \rrbracket_{\mathcal{M}} \sqsubseteq^\sigma \llbracket u \rrbracket_{\mathcal{M}'}$. By the left-uniqueness of \sqsubseteq^σ , the extensions cannot be equal, and thus $\mathcal{M}' \not\models t \simeq u$.

ALL⁻: Assume that $\Gamma[x \mapsto \sigma] \vdash M^-(t)$ and $\mathcal{M} \not\models \forall x^\sigma. t$. Then there exists $a \in \llbracket \sigma \rrbracket_S$ such that $(S, V[x \mapsto a]) \not\models t$. Since \sqsubseteq^σ is left-total, there exists an extension $b \sqsubseteq^\sigma a$. Since $(S, V[x \mapsto a]) \sqsubseteq_\Gamma (S', V'[x \mapsto b])$, we conclude $(S', V'[x \mapsto b]) \not\models t$ by induction hypothesis. Thus $\mathcal{M}' \not\models \forall x^\sigma. t$.

ALL⁺: Assume that $\Gamma[x \mapsto \sigma] \vdash M^+(t)$, $\tilde{\alpha} \notin \text{TV}^+(\sigma)$, and $\mathcal{M} \models \forall x^\sigma. t$. We show that $\mathcal{M}' \models \forall x^\sigma. t$. Let $b \in \llbracket \sigma \rrbracket_{S'}$. Since \sqsubseteq^σ is right-total (Lemma 6.5), there exists a restriction $a \in \llbracket \sigma \rrbracket_S$ with $a \sqsubseteq^\sigma b$. By assumption, $(S, V[x \mapsto a]) \models t$. Since $(S, V[x \mapsto a]) \sqsubseteq_\Gamma (S', V'[x \mapsto b])$, we conclude $(S', V'[x \mapsto b]) \models t$ by induction hypothesis. \square

Theorem 6.19 (Soundness of the Calculus) If $\Gamma \vdash M^+(t)$ can be derived in some arbitrary context Γ , then t is monotonic. If $\Gamma \vdash M^-(t)$ can be derived in some arbitrary context Γ , then t is antimonotonic.

Proof The definition of monotonicity requires showing the existence of a model $\mathcal{M}' = (S', V')$ for any scope S' such that $S \leq_{\tilde{\alpha}} S'$. By Theorem 6.18, we can take any model \mathcal{M}' for which $\mathcal{M} \sqsubseteq_\Gamma \mathcal{M}'$. Such a model exists because \sqsubseteq_Γ is left-total (by Lemma 6.3 and Definition 6.8). \square

Example 6.20 The following table lists some example formulas, including all those from the introduction. For each formula, we indicate whether it is monotonic or antimonotonic with respect to α according to the calculi \mathfrak{M}_1 and \mathfrak{M}_2 and to the semantic definitions.

FORMULA	MONOTONIC			ANTIMONOTONIC		
	\mathfrak{M}_1	\mathfrak{M}_2	SEM.	\mathfrak{M}_1	\mathfrak{M}_2	SEM.
$\exists x^\alpha y. x \not\approx y$	✓	✓	✓	·	·	·
$f x^\alpha \simeq x \wedge f y \not\approx y$	✓	✓	✓	✓	✓	✓
$x^{0 \rightarrow \alpha} \simeq y$	✓	✓	✓	✓	✓	✓
$s^{\alpha \rightarrow 0} \simeq t$	·	✓	✓	✓	✓	✓
$\{y^\alpha\} \simeq \{z\}$	·	✓	✓	✓	✓	✓
$(\lambda x^\alpha. x \simeq y) \simeq (\lambda x. x \simeq z)$	·	·	✓	✓	✓	✓
$(\forall x^\alpha. f x \simeq x) \wedge f y \not\approx y$	·	·	✓	✓	✓	✓
$\forall x^\alpha y. x \simeq y$	·	·	·	✓	✓	✓
$\exists x^\alpha y. x \not\approx y \wedge \forall z. z \simeq x \vee z \simeq y$	·	·	·	·	·	·

Remark In Definition 6.4, both the positive and the negative type variables in σ count as positive occurrences in $\sigma \rightarrow_F \tau$. This raises the question of whether a fully covariant behavior, with $\text{TV}^s(\sigma \rightarrow_F \tau) = \text{TV}^s(\sigma) \cup \text{TV}^s(\tau)$, could also be achieved, presumably with a different definition of $\sqsubseteq^{\sigma \rightarrow_F \tau}$. Although such a behavior looks more regular, it would make the calculus unsound, as the following counterexample shows:

$$\forall F^{(\tilde{\alpha} \rightarrow 0) \rightarrow 0} f^{\tilde{\alpha} \rightarrow 0} g h. f \in F \wedge g \in F \wedge f a \not\approx g a \longrightarrow h \in F.$$

The formula is not monotonic: Regardless of the value of the free variable a , it is true for $|\tilde{\alpha}| = 1$, since the assumptions imply that $f \not\approx g$, and as there are only two functions of type $\tilde{\alpha} \rightarrow 0$, h can only be one of them, so it must be in F . This argument breaks down for larger scopes, so the formula is not monotonic. However, with a fully covariant F-arrow, we could type F as $F^{(\tilde{\alpha} \rightarrow G^0) \rightarrow F^0}$ and the rule ALL^+ would apply, since there are no positive occurrences of $\tilde{\alpha}$ in the types of F , f , g , and h .

6.4 Type Inference

Expecting all types to be fully annotated with G and F is unrealistic, so we now face the problem of computing annotations such that a given term is typable—a type inference problem. We follow a standard approach to type inference: We start by annotating all types with *annotation variables* ranging over $\{G, F\}$. Then we construct a typing derivation by backward chaining, collecting a set of constraints over the annotations. Finally, we look for an instantiation of the annotation variables that satisfies all the constraints.

Definition 6.21 (Annotation Constraint) An *annotation constraint* over a set of annotation variables X is an expression of the form $\sigma \leq \tau$, $\tilde{\alpha} \notin \text{TV}^+(\sigma)$, or $\tilde{\alpha} \notin \text{TV}^-(\sigma)$, where the types σ and τ may contain annotation variables in V . Given a valuation $\rho : X \rightarrow \{G, F\}$, the meaning of a constraint is given by Definitions 6.4 and 6.10.

A straightforward way of solving such constraints is to encode them in propositional logic, following Definitions 6.4 and 6.10, and give them to a SAT solver. Annotation variables, which may take two values, are mapped directly to propositional variables. Only one rule, SUB, is not syntax-directed, but it is sufficient to apply it to the second argument of an application and to variables and constants before invoking VAR or CONST. This approach proved very efficient on the problems that we encountered in our experiments.

We have so far been unable to prove that the satisfiability problem for this constraint language is NP-complete. We suspect that it is not, but we have not found a polynomial-time algorithm. Thus, it is unclear if our use of a SAT solver is fully appropriate from a theoretical point of view, even though it works perfectly well in practice.

Remark Similarly to the simply-typed λ -calculus, our type system admits principal types if we promote annotation variables to first-class citizens. When performing type inference, we would then keep the constraints as part of the type, instead of computing an arbitrary solution for the collected constraints. More precisely, a *type schema* would have the form $\forall \bar{A}. \forall \bar{a}. \sigma \langle C \rangle$, where σ is an annotated type containing annotation variables \bar{A} and type variables \bar{a} , and C is a list of constraints of the form given in Definition 6.21. For example, equality would have the principal type schema $\forall \alpha. \alpha \rightarrow_G \alpha \rightarrow_A \circ \langle \tilde{\alpha} \notin \text{TV}^-(\alpha) \rangle$. This approach nicely extends ML-style polymorphism.

7 Third Calculus: Inferring Set Comprehensions

An obvious deficiency of the calculus \mathfrak{M}_2 from the previous section is that the rule LAM always types λ -abstractions with G-arrows. The only way to construct a term whose type contains F-annotations is to build it from primitives whose types are justified semantically. In other words, we cannot type set comprehensions precisely.

This solution is far from optimal. To take one example, consider the term $\lambda R S x z. \exists y. R x y \wedge S y z$, which composes two binary relations R and S . Semantically, composition is constant for type $(\alpha \rightarrow_F \beta \rightarrow_F \circ) \rightarrow_G (\beta \rightarrow_F \gamma \rightarrow_F \circ) \rightarrow_G \alpha \rightarrow_F \gamma \rightarrow_F \circ$, but \mathfrak{M}_2 cannot infer this. As a result, it cannot infer the monotonicity of any of the four type variables occurring in the associativity law for composition, unless composition is considered a primitive and added to the constant context Γ_c . Another annoyance is that the η -expansion $\lambda x. t x$ of a term $t^{\sigma \rightarrow_F \tau}$ can only be typed with a G-arrow.

The calculus \mathfrak{M}_3 introduced in this section is designed to address this. The underlying intuition is that a term $\lambda x^\alpha. t$ can be typed as $\alpha \rightarrow_F \circ$ if we can show that the body t evaluates to \perp whenever x is new in the larger scope. The key is to track what happens to a term when one or several of its free variables are assigned a new value. When abstracting over a variable, we can then use this information to annotate the function arrow precisely. Such a scheme covers bounded set comprehensions such as $\lambda x. x \in A^{\alpha \rightarrow_F \circ} \wedge p x$ and, by way of consequence, bounded quantifications such as $\forall x. x \in A^{\alpha \rightarrow_F \circ} \longrightarrow q x$.

Definition 7.1 (Annotated Type) An *annotated type* is a HOL type in which each function arrow carries an *annotation* $A \in \{G, N, F, T\}$.

The G- and F-annotations have the same meaning as in Section 6. The T-annotation is similar to F, but with \top instead of \perp as its default extension value of type \circ ; the universal set $\mathcal{U}^{\alpha \rightarrow \circ}$ can be typed precisely as $\alpha \rightarrow_T \circ$. Finally, the N-annotation indicates that if the argument to the function is a new value, so is its result.

7.1 Extension Relation

The extension relation \sqsubseteq^σ distinguishes the four kinds of arrows. The G and F cases coincide with Definition 6.2.

Definition 7.2 (Extension) Let σ be an annotated type, and let S, S' be scopes such that $S \leq_{\tilde{\alpha}} S'$. The *extension relation* $\sqsubseteq^{\sigma} \subseteq \llbracket \sigma \rrbracket_S \times \llbracket \sigma \rrbracket_{S'}$ for S and S' is defined by

$$\begin{aligned} a \sqsubseteq^{\sigma} b & \text{ iff } a = b & \text{ if } \sigma \text{ is o or a type variable} \\ f \sqsubseteq^{\sigma \rightarrow A^{\tau}} g & \text{ iff } \forall a b. a \sqsubseteq^{\sigma} b \longrightarrow f(a) \sqsubseteq^{\tau} g(b) \text{ and } \forall b. N^{\sigma}(b) \longrightarrow A^{\tau}(g(b)) \end{aligned}$$

and

$$\begin{aligned} G^{\sigma}(b) & \text{ iff } \top & F^{\sigma}(b) & \text{ iff } b = \llbracket \sigma \rrbracket^F \\ N^{\sigma}(b) & \text{ iff } \nexists a. a \sqsubseteq^{\sigma} b & T^{\sigma}(b) & \text{ iff } b = \llbracket \sigma \rrbracket^T \end{aligned}$$

where $\llbracket \text{o} \rrbracket^F = \perp$, $\llbracket \text{o} \rrbracket^T = \top$, $\llbracket \alpha \rrbracket^A \in S(\alpha)$, and $\llbracket \sigma \rightarrow \tau \rrbracket^A = a \in \llbracket \sigma \rrbracket_{S'} \mapsto \llbracket \tau \rrbracket^A$.

We cannot require $\llbracket \alpha \rrbracket^F \neq \llbracket \alpha \rrbracket^T$ because this would be impossible for $|\alpha| = 1$. Hence, we will be careful to assume $\llbracket \sigma \rrbracket^F \neq \llbracket \sigma \rrbracket^T$ only if σ is of the form $\sigma_1 \rightarrow \dots \rightarrow \sigma_n \rightarrow \text{o}$.

Definition 7.3 (Positive and Negative Type Variables) The set of *positive type variables* $\text{TV}^+(\sigma)$ and the set of *negative type variables* $\text{TV}^-(\sigma)$ of an annotated type σ are defined as follows:

$$\begin{aligned} \text{TV}^+(\text{o}) &= \emptyset & \text{TV}^+(\sigma \rightarrow_A \tau) &= \begin{cases} \text{TV}^+(\tau) \cup \text{TV}^-(\sigma) & \text{if } A = G \\ \text{TV}^+(\tau) \cup \text{TV}^-(\sigma) \cup \text{TV}^+(\sigma) & \text{otherwise} \end{cases} \\ \text{TV}^+(\alpha) &= \{\alpha\} \\ \text{TV}^-(\text{o}) &= \emptyset & \text{TV}^-(\sigma \rightarrow_A \tau) &= \begin{cases} \text{TV}^-(\tau) \cup \text{TV}^+(\sigma) & \text{if } A \in \{G, N\} \\ \text{TV}^-(\tau) & \text{otherwise.} \end{cases} \\ \text{TV}^-(\alpha) &= \emptyset \end{aligned}$$

With the introduction of an N-annotation, not all annotated types are legitimate. For example, $\tilde{\alpha} \rightarrow_N \text{o}$ would mean that new values of type $\tilde{\alpha}$ are mapped to new Booleans, but there is no such thing as a new Boolean, since $\llbracket \text{o} \rrbracket$ is fixed to $\{\perp, \top\}$.

Definition 7.4 (Well-Annotated Type) An annotated type σ is *well-annotated* if $\text{WA}(\sigma)$ holds, where $\text{WA}(\sigma)$ is defined by the following equivalences:

$$\begin{aligned} \text{WA}(\sigma) & \text{ iff } \top & \text{ if } \sigma \text{ is o or a type variable} \\ \text{WA}(\sigma \rightarrow_A \tau) & \text{ iff } \text{WA}(\sigma) \text{ and } \text{WA}(\tau) \text{ and } A = N \longrightarrow \text{body}(\tau) = \tilde{\alpha} \end{aligned}$$

where $\text{body}(\text{o}) = \text{o}$, $\text{body}(\alpha) = \alpha$, and $\text{body}(\sigma \rightarrow \tau) = \text{body}(\tau)$.

Lemma 7.5 *If σ is well-annotated, then \sqsubseteq^{σ} is left-total (total) and left-unique (injective).*

Proof (structural induction on σ) The proof is similar to that of Lemma 6.3, except that we must propagate the WA assumption. There is one genuinely new case.

LEFT-TOTALITY OF $\sqsubseteq^{\sigma \rightarrow N^{\tau}}$: For $f \in \llbracket \sigma \rightarrow \tau \rrbracket_S$, we find an extension g as follows: Let $b \in \llbracket \sigma \rrbracket_{S'}$. If b extends an a , that a is unique by left-uniqueness of \sqsubseteq^{σ} . Since \sqsubseteq^{τ} is left-total, there exists a y such that $f(a) \sqsubseteq^{\tau} y$, and we let $g(b) = y$. If b does not extend any a , then we set $g(b)$ to a new element y constructed as follows: Since τ is well-annotated, it must be of the form $\tau_1 \rightarrow_{A_1} \dots \rightarrow_{A_{n-1}} \tau_n \rightarrow_{A_n} \alpha$. As value for y , we simply take $x_1 \in \llbracket \tau_1 \rrbracket_{S'} \mapsto \dots \mapsto x_m \in \llbracket \tau_n \rrbracket_{S'} \mapsto y'$, where y' is not the extension of any x' . Such an y' exists, because otherwise $|S(\tilde{\alpha})| = |S'(\tilde{\alpha})|$, which is inconsistent with the existence of an element b that does not extend any a . \square

Lemma 7.6 *Let σ be a well-annotated type. If $\tilde{\alpha} \notin \text{TV}^+(\sigma)$, then \sqsubseteq^{σ} is right-total (surjective). If $\tilde{\alpha} \notin \text{TV}^-(\sigma)$, then \sqsubseteq^{σ} is right-unique (functional).*

Proof The proof is similar to that of Lemma 6.5. Lemma 7.5, which replaces Lemma 6.3, requires σ to be well-annotated. \square

7.2 Type Checking

As in Section 6, constancy checking is treated as a type checking problem involving annotated types. The judgments still have the form $\Gamma \vdash t : \sigma$, but the context now carries more information about t 's value when its free variables are assigned new values.

Definition 7.7 (Context) A *context* is a pair $\Gamma = (\Gamma_c, \Gamma_v)$, where Γ_c maps constant symbols to sets of annotated types, and Γ_v is a list $[x_1 :^{A_1} \sigma_1, \dots, x_m :^{A_m} \sigma_m]$ associating m distinct variables x_i with annotations A_i and annotated types σ_i . A context is *well-annotated* if all the types σ_i are well-annotated.

We assume as before that Γ_c is fixed and compatible with the standard constant model and write simply Γ for Γ_v . We also let $\langle x_m :^{A_m} \sigma_m \rangle$ abbreviate $[x_1 :^{A_1} \sigma_1, \dots, x_m :^{A_m} \sigma_m]$.

The intuitive meaning of a typing judgment $\Gamma \vdash t : \sigma$ with $\Gamma = \langle x_m :^{A_m} \sigma_m \rangle$ is that if x_1 is new, then $A_1^{\sigma_1}(t)$ holds; if $V(x_1) \sqsubseteq^{\sigma_1} V'(x_1)$ but x_2 is new, then $A_2^{\sigma_2}(t)$ holds; and so on. Furthermore, if $V(x_i) \sqsubseteq^{\sigma_i} V'(x_i)$ for all $i \in \{1, \dots, m\}$, then $\llbracket t \rrbracket_{\mathcal{M}} \sqsubseteq^{\sigma} \llbracket t \rrbracket_{\mathcal{M}'}$. It may help to think of a judgment $\langle x_m :^{A_m} \sigma_m \rangle \vdash t : \sigma$ as meaning roughly the same as $\square \vdash \lambda x_1 \dots x_m. t : \sigma_1 \rightarrow_{A_1} \dots \rightarrow_{A_{m-1}} \sigma_m \rightarrow_{A_m} \sigma$.

Example 7.8 Given a constant r such that $\tilde{a} \rightarrow_{\top} \tilde{a} \rightarrow_{\text{F}} \circ \in \Gamma_c(c)$, the new typing rules will let us derive the following judgments:

$$[x :^{\top} \tilde{a}, y :^{\text{F}} \tilde{a}] \vdash r x y : \circ \quad [x :^{\top} \tilde{a}] \vdash \lambda y. r x y : \tilde{a} \rightarrow_{\text{F}} \circ \quad \square \vdash \lambda x y. r x y : \tilde{a} \rightarrow_{\top} \tilde{a} \rightarrow_{\text{F}} \circ.$$

Notice that the η -expanded form $\lambda x y. r x y$ can be typed in the same way as r .

Definition 7.9 (Model Extension) Let $\mathcal{M} = (S, V)$ and $\mathcal{M}' = (S', V')$ be models, and let Γ, Δ be two contexts with disjoint sets of variables. The model \mathcal{M}' *extends* \mathcal{M} strongly in Γ and weakly in Δ , written $\mathcal{M} \sqsubseteq_{\Gamma}^{\Delta} \mathcal{M}'$, if $S \leq_{\tilde{a}} S'$, $x :^A \sigma \in \Gamma$ implies $V(x) \sqsubseteq^{\sigma} V'(x)$, and $x :^A \sigma \in \Delta$ implies either $V(x) \sqsubseteq^{\sigma} V'(x)$ or $N^{\sigma}(V'(x))$ for all x . If $\Delta = \square$, we write $\mathcal{M} \sqsubseteq_{\Gamma} \mathcal{M}'$.

Definition 7.10 (Constancy) Let σ be an annotated type. A term t is σ -*constant* in a context Γ if $\llbracket t \rrbracket_{\mathcal{M}} \sqsubseteq^{\sigma} \llbracket t \rrbracket_{\mathcal{M}'}$ for all models $\mathcal{M}, \mathcal{M}'$ such that $\mathcal{M} \sqsubseteq_{\Gamma} \mathcal{M}'$.

Definition 7.11 (Conformity) Let σ be an annotated type, and let Γ be a context. A term t is σ -*conform* to Γ if for all decompositions $\Gamma = \Delta, [x :^A \tau], E$ and for all models $\mathcal{M}, \mathcal{M}'$ such that $\mathcal{M} \sqsubseteq_{\Delta}^E \mathcal{M}'$, we have that $N^{\tau}(\llbracket x \rrbracket_{\mathcal{M}'})$ implies $A^{\sigma}(\llbracket t \rrbracket_{\mathcal{M}'})$.

Equipped with these semantic definitions, we are ready to examine the inference rules of \mathfrak{M}_3 relating to constancy and monotonicity.

Definition 7.12 (Typing Rules) The *typing* relation $\Gamma \vdash t : \sigma$ is given by the rules below.

Context rules:

$$\frac{\Gamma, \Delta \vdash t : \tau}{\Gamma, [x :^G \sigma], \Delta \vdash t : \tau} \text{ADD} \quad \frac{\Gamma, [x :^A \sigma], \Delta \vdash t : \tau}{\Gamma, [x :^G \sigma], \Delta \vdash t : \tau} \text{ANN}$$

$$\frac{\Gamma, [y :^B \tau, x :^A \sigma], \Delta \vdash t : \nu}{\Gamma, [x :^A \sigma, y :^B \tau], \Delta \vdash t : \nu} \text{SWAP} \quad \text{where } A \in \{B, G\}.$$

Nonlogical rules:

$$\frac{}{[x :^N \sigma] \vdash x : \sigma} \text{VAR} \quad \frac{\sigma \in \Gamma_c(c)}{\square \vdash c : \sigma} \text{CONST}$$

$$\begin{array}{c}
\frac{\Gamma \vdash t : \sigma \quad \sigma \leq \sigma'}{\Gamma \vdash t : \sigma'} \text{ SUB} \qquad \frac{\Gamma, [x :^A \sigma] \vdash t : \tau}{\Gamma \vdash \lambda x. t : \sigma \rightarrow_A \tau} \text{ LAM} \\
\frac{\langle x_m :^{A_m} \sigma_m \rangle \vdash t : \sigma \rightarrow_B \tau \quad \langle x_m :^G \sigma_m \rangle, \langle y_n :^N \tau_n \rangle \vdash u : \sigma}{\langle x_m :^{A_m} \sigma_m \rangle, \langle y_n :^B \tau_n \rangle \vdash t u : \tau} \text{ APP} \quad \text{where } A_i \in \{G, F, T\}.
\end{array}$$

Logical rules:

$$\begin{array}{c}
\frac{}{\langle x_m :^F \sigma_m \rangle \vdash \text{False} : \circ} \text{ FALSE} \qquad \frac{}{\langle x_m :^T \sigma_m \rangle \vdash \text{True} : \circ} \text{ TRUE} \\
\frac{\langle x_m :^{A_m} \sigma_m \rangle \vdash t : \circ \quad \langle x_m :^{B_m} \sigma_m \rangle \vdash u : \circ}{\langle x_m :^{A_m \rightsquigarrow B_m} \sigma_m \rangle \vdash t \longrightarrow u : \circ} \text{ IMP} \quad \text{where } A \rightsquigarrow B = \begin{cases} T & \text{if } A = F \text{ or } B = T \\ F & \text{if } A = T \text{ and } B = F \\ G & \text{otherwise.} \end{cases}
\end{array}$$

The *subtype* relation $\sigma \leq \tau$ is defined by the rules

$$\frac{}{\sigma \leq \sigma} \text{ REFL} \qquad \frac{\sigma' \leq \sigma \quad \tau \leq \tau'}{\sigma \rightarrow_A \tau \leq \sigma' \rightarrow_G \tau'} \text{ GEN} \qquad \frac{\sigma' \leq \sigma \quad \sigma \leq \sigma' \quad \tau \leq \tau'}{\sigma \rightarrow_A \tau \leq \sigma' \rightarrow_A \tau'} \text{ ANY.}$$

The nonlogical rules are similar to the rules of Definition 6.10, but the LAM rule now allows arbitrary annotations, and the other rules impose various restrictions on the annotations in contexts. These rules are complemented by logical rules that support rudimentary propositional reasoning within terms.

Another noteworthy aspect of the new typing rules are the context rules. In the previous calculus, the context was a set and we could dispense with explicit context rules. The context now being a list, we need weakening rules to add and permute variables and to change the annotations in a controlled way. The new typing rules form a substructural type system [22].

Lemma 7.13 *If $\sigma \leq \sigma'$, then $\sqsubseteq^\sigma \subseteq \sqsubseteq^{\sigma'}$.*

Proof Similar to the proof of Lemma 6.11. \square

The proof of the soundness theorem relies on two closure properties of functional abstraction and application.

Lemma 7.14 *Let $g \in \llbracket \sigma \rightarrow \tau \rrbracket_{S'}$.*

- (a) *If $A^\tau(g(b))$ for all $b \in \llbracket \sigma \rrbracket_{S'}$, then $A^{\sigma \rightarrow B^\tau}(g)$.*
- (b) *If $A \in \{G, F, T\}$ and $A^{\sigma \rightarrow B^\tau}(g)$, then $A^\tau(g(b))$ for all $b \in \llbracket \sigma \rrbracket_{S'}$.*

Proof Immediate from Definition 7.2. \square

It is regrettable that Lemma 7.14(b) does not hold uniformly for all annotation types and, as a result, that the APP rule has a side condition $A_i \in \{G, F, T\}$. The crux is that while a function that maps old values to new values is necessarily new, the converse does not hold: A function may be new even if it maps old values to old values. Given the type $\tilde{\alpha} \rightarrow_F \circ$, the function (\perp, \perp, \top) depicted in Figure 6.1(b) is such an example.

Theorem 7.15 (Soundness of Typing) *If $\Gamma \vdash t : \sigma$, then t is both σ -constant in Γ and σ -conform to Γ .*

Proof (induction on the derivation of $\Gamma \vdash t : \sigma$) The cases ADD, ANN, VAR, CONST, FALSE, TRUE, and IMP are easy to prove using the definitions of \sqsubseteq^σ , constancy, and conformity. The remaining cases are proved below.

SWAP: The case $A = G$ is easy. And in the remaining case, the only subtlety occurs when both x and y are new, i.e., $N^\sigma(\llbracket x \rrbracket_{\mathcal{M}'})$ and $N^\tau(\llbracket y \rrbracket_{\mathcal{M}'})$; but since $A = B$, the behaviors dictated by x and y agree and we can exchange them.

SUB: By Lemma 7.13 and Definition 7.10.

LAM: The $(\sigma \rightarrow_A \tau)$ -conformity of $\lambda x. t$ follows from Lemma 7.14(a) and the induction hypothesis; constancy is easy to prove from the induction hypothesis and the definition of $\sqsubseteq^{\sigma \rightarrow A \tau}$. We can omit $[x :^A \sigma]$ in the conclusion because x does not occur free in $\lambda x. t$.

APP: Let $\Gamma = \langle x_m :^{A_m} \sigma_m \rangle, \langle y_n :^B \tau_n \rangle$. The constancy proof is as for Theorem 6.12. It remains to show that $t u$ is τ -conform to Γ . Let $\Gamma = \Delta, [z :^C v], E$ and assume $N^v(\llbracket z \rrbracket_{\mathcal{M}'})$. If z is one of the x_i 's, we have $C^{\sigma \rightarrow B \tau}(\llbracket t \rrbracket_{\mathcal{M}'})$ by the first induction hypothesis and hence $C^\tau(\llbracket t u \rrbracket_{\mathcal{M}'})$ by Lemma 7.14(b). If z is among the y_j 's (in which case $C = B$), the second induction hypothesis tells us that $\llbracket u \rrbracket_{\mathcal{M}'}$ is new, and since $\llbracket t \rrbracket_{\mathcal{M}'} \sqsubseteq^{\sigma \rightarrow B \tau} \llbracket t \rrbracket_{\mathcal{M}'}$ by the first induction hypothesis, we have $B^\tau(\llbracket t u \rrbracket_{\mathcal{M}'})$ by the definition of $\sqsubseteq^{\sigma \rightarrow B \tau}$. \square

From Definition 7.12, it is straightforward to derive typing rules for $\neg, \wedge, \vee, \forall$, and \exists in the style of Definition 6.17. To save space, we omit to do so.

Now that we have powerful typing rules at our disposal, we no longer need to reason semantically about set constructs. This leaves us with a much reduced constant context.

Definition 7.16 (Standard Constant Context) The *standard constant context* $\widehat{\Gamma}_c$ is the following mapping:

$$\begin{array}{lll} \longrightarrow & \mapsto & \{\circ \rightarrow_G \circ \rightarrow_G \circ\} \\ \simeq & \mapsto & \{\sigma \rightarrow_G \sigma \rightarrow_F \circ \mid \tilde{\alpha} \notin \text{TV}^-(\sigma)\}. \end{array}$$

The examples below exploit the new calculus to type set constructs precisely.

Example 7.17 The empty set $\emptyset^{\sigma \rightarrow \circ}$ and the universal set $\mathcal{U}^{\sigma \rightarrow \circ}$ get their natural typings:

$$\frac{\overline{[x :^F \sigma] \vdash \text{False} : \circ} \text{ FALSE}}{\boxed{\vdash} \lambda x. \text{False} : \sigma \rightarrow_F \circ} \text{ LAM} \qquad \frac{\overline{[x :^T \sigma] \vdash \text{True} : \circ} \text{ TRUE}}{\boxed{\vdash} \lambda x. \text{True} : \sigma \rightarrow_T \circ} \text{ LAM}$$

Example 7.18 The complement \bar{A} of a set A is the set $\mathcal{U} - A$. Using the rule

$$\frac{\langle x_m :^{A_m} \sigma_m \rangle \vdash t : \circ}{\langle x_m :^{\sim A_m} \sigma_m \rangle \vdash \neg t : \circ} \text{ NOT} \quad \text{where } \sim A = \begin{cases} T & \text{if } A = F \\ F & \text{if } A = T \\ G & \text{otherwise} \end{cases}$$

derived from IMP and FALSE in the obvious way, set complementation can be typed as follows for $A \in \{G, F, T\}$:

$$\frac{\overline{[s :^N \sigma \rightarrow_A \circ] \vdash s : \sigma \rightarrow_A \circ} \text{ VAR} \quad \overline{[x :^N \sigma] \vdash x : \sigma} \text{ VAR}}{\overline{[s :^G \sigma \rightarrow_A \circ] \vdash s : \sigma \rightarrow_A \circ} \text{ ANN} \quad \overline{[s :^G \sigma \rightarrow_A \circ, x :^N \sigma] \vdash x : \sigma} \text{ ADD}} \text{ APP} \\ \frac{\overline{[s :^G \sigma \rightarrow_A \circ, x :^A \sigma] \vdash s x : \circ} \text{ NOT}}{\overline{[s :^G \sigma \rightarrow_A \circ, x :^{\sim A} \sigma] \vdash \neg s x : \circ} \text{ LAM}} \text{ LAM} \\ \frac{\overline{[s :^G \sigma \rightarrow_A \circ] \vdash \lambda x. \neg s x : \sigma \rightarrow_{\sim A} \circ} \text{ LAM}}{\boxed{\vdash} \lambda s x. \neg s x : (\sigma \rightarrow_A \circ) \rightarrow_G \sigma \rightarrow_{\sim A} \circ} \text{ LAM}$$

7.3 Monotonicity Checking

The rules for checking monotonicity and antimonotonicity are similar to those given in Section 6. The only new rule is ALL^\ddagger ; it exploits the context to avoid the restriction on $\tilde{\alpha}$.

Definition 7.19 (Monotonicity Rules) The predicates $\Gamma \vdash M^+(t)$ and $\Gamma \vdash M^-(t)$ are given by the rules

$$\begin{array}{c} \frac{\Gamma \vdash t : \circ}{\Gamma \vdash M^s(t)} \text{TERM} \quad \frac{\langle x_m :^{A_m} \sigma_m \rangle \vdash M^{\sim s}(t) \quad \langle x_m :^{B_m} \sigma_m \rangle \vdash M^s(u)}{\langle x_m :^{A_m \rightsquigarrow B_m} \sigma_m \rangle \vdash M^s(t \longrightarrow u)} \text{IMP} \\ \\ \frac{\langle x_m :^G \sigma_m \rangle \vdash t : \sigma \quad \langle x_m :^G \sigma_m \rangle \vdash u : \sigma}{\langle x_m :^G \sigma_m \rangle \vdash M^-(t \simeq u)} \text{EQ}^- \quad \frac{\Gamma, [x :^G \sigma] \vdash M^-(t)}{\Gamma \vdash M^-(\forall x. t)} \text{ALL}^- \\ \\ \frac{\Gamma, [x :^G \sigma] \vdash M^+(t) \quad \tilde{\alpha} \notin \text{TV}^+(\sigma)}{\Gamma \vdash M^+(\forall x. t)} \text{ALL}^+ \quad \frac{\Gamma, [x :^T \sigma] \vdash M^+(t)}{\Gamma \vdash M^+(\forall x. t)} \text{ALL}^\ddagger. \end{array}$$

From Definition 7.19, it is straightforward to derive monotonicity rules for *False*, *True*, \neg , \wedge , \vee , and \exists .

Theorem 7.20 (Soundness of M^s) *Let \mathcal{M} and \mathcal{M}' be models such that $\mathcal{M} \sqsubseteq_\Gamma \mathcal{M}'$. If $\Gamma \vdash M^+(t)$, then t is monotonic and o-conform to Γ . If $\Gamma \vdash M^-(t)$, then t is antimonotonic and o-conform to Γ .*

Proof (induction on the derivation of $\Gamma \vdash M^s(t)$) The TERM, EQ⁻, ALL⁻, and ALL⁺ cases are similar to the corresponding cases in the proof of Theorem 6.18. The IMP case is analogous to the IMP case of Theorem 7.15. Finally, the rule ALL[‡] can be derived by considering $\forall x. t$ an abbreviation for $(\lambda x. t) \simeq (\lambda x. \text{True})$. \square

Theorem 7.21 (Soundness of the Calculus) *If $\Gamma \vdash M^+(t)$ can be derived in some arbitrary well-annotated context Γ , then t is monotonic. If $\Gamma \vdash M^-(t)$ can be derived in some arbitrary well-annotated context Γ , then t is antimonotonic.*

Proof By Theorem 7.20, we can take any model \mathcal{M}' such that $\mathcal{M} \sqsubseteq_\Gamma \mathcal{M}'$ as witness for monotonicity. Such a model exists because \sqsubseteq_Γ is left-total for well-annotated contexts Γ (by Lemma 6.3 and Definition 6.8). \square

7.4 Type Inference

At the cost of some inelegant technicalities, the approach sketched in Section 6.4 for inferring types can be adapted to the new setting.

The nonlogical rules VAR, CONST, and LAM as well as all the logical rules are syntax-directed and pose no problem when constructing a typing derivation by backward chaining. The SUB rule is unproblematic for the same reasons as for \mathfrak{M}_2 . Similarly, the context rules ADD and ANN can be delayed to the leaves of the derivation. The SWAP rule is useful only in conjunction with APP, the only other rule that examines the order of the context variables; other occurrences of SWAP can be either postponed or omitted altogether.

The challenge is to handle APP and SWAP, because it is not obvious where to split the context in two parts and whether variables should be exchanged beforehand. Since the context is finite, we could enumerate all permutations and splittings, but this could lead to an exponential explosion in the number of constraints.

Fortunately, there is a general approach to determine which variables to permute and where to split the context, based on the rule

$$\frac{\langle x_m :^{A_m} \sigma_m \rangle \vdash t : \sigma \rightarrow_B \tau \quad \langle x_m :^G \sigma_m \rangle, \langle y_n :^G \tau_n \rangle, \langle z_p :^N \nu_p \rangle \vdash u : \sigma}{\langle x_m :^{A_m} \sigma_m \rangle, \langle y_n :^G \tau_n \rangle, \langle z_p :^B \nu_p \rangle \vdash t u : \tau} \text{XAPP}$$

where $A_i \in \{G, F, T\}$, $x_i \in \text{FV}(t)$, and $y_j, z_k \notin \text{FV}(t)$. The rule is easy to derive from APP and ADD. Before applying it, we invoke SWAP repeatedly to separate the variables occurring free in t (the x_i 's) from the others (the y_j 's and z_k 's).

The remaining hurdle is to determine where to split between the y_j 's and the z_k 's. This can be coded as polynomial-size constraints. If $B = G$, we can ignore the z_k 's and list all variables $\notin \text{FV}(t)$ as part of the y_j 's; otherwise, the right-hand part of the context must have the form $\langle y_n :^G \tau_n \rangle, \langle z_p :^B \nu_p \rangle$ already—the SWAP rule cannot bring it into that form if it is not already in that form. So we keep the variables $\notin \text{FV}(t)$ in the order in which they appear without distinguishing statically between the y_j and z_k variables, and generate constraints to ensure that the annotations for the y_j 's and z_k 's in the conclusion have the desired form G, \dots, G, B, \dots, B and correspondingly in the second hypothesis but with N instead of B .

The completeness proof rests on two ideas:

1. Any APP instance in which some of the variables x_i do not occur free in t can only carry a G -annotation and will eventually be eliminated using ADD.¹
2. The variable exchanges we perform to bring the x_i 's to the left are necessary to meet the general invariant $\text{FV}(t) \subseteq \{x_1, \dots, x_m\}$ on all derivations $\langle x_m :^{A_m} \sigma_m \rangle \vdash t : \sigma$, and any additional SWAPS can be postponed.

Type inference is in NP because our procedure generates a polynomial-sized SAT problem. It is also NP-hard because we can reduce SAT to it using the following simple scheme: Translate propositional variables to HOL variables of type $\tilde{\alpha} \rightarrow \text{o}$ and connectives to the corresponding set operations; for example, $(A \wedge \neg B) \vee B$ becomes $(A \cap \bar{B}) \cup B$. The propositional formula is satisfiable iff the corresponding term is typable as $\tilde{\alpha} \rightarrow_{\top} \text{o}$ in some context.

8 Practical Considerations

To make monotonicity checking useful in practice, we must add support for user-defined constants and types, which we have ignored so far. In this section, we briefly sketch these extensions before we evaluate our approach empirically on existing Isabelle/HOL theories.

8.1 Constant Definitions

In principle, user-defined constant symbols are easy to handle: We can simply build a conjunction from the definitions and the negated conjecture (where the user-defined symbols appear as variables) and hand it to the monotonicity checker. Unfortunately, this has disastrous effects on the precision of our method: Even a simple definition of $f^{\alpha \rightarrow \beta}$ by an equality leads to a formula of the form $(\forall x^\alpha. f x \simeq t) \wedge \neg u$, which does not pass the monotonicity check because the universal quantifier requires $\alpha \notin \text{TV}^+(\alpha)$. Rewriting the definition to the form $f \simeq (\lambda x. t)$ does not help. We must thus treat definitions specially.

¹ In some pathological cases, such as the application $(\lambda y. \text{False}) x$, the variables not occurring in t can be annotated with F or T and eliminated using FALSE or TRUE. We can avoid these cases by simply requiring that terms are β -reduced.

Definition 8.1 Let x^σ be a variable. We say that a formula t is a *specification* for x if t is satisfiable in each scope and $FV(t) = \{x\}$. Then the (nonempty) set of values in $\llbracket \sigma \rrbracket_S$ that satisfy t is denoted by $Spec_S^t$.

In our application, specifications arise from Isabelle/HOL; we can assume that they are explicitly marked as such and do not have to recognize them syntactically.

Specifications are trivially monotonic because they are satisfiable in each scope, and we can safely drop the monotonicity check for them. However, we must assign an annotated type to the defined symbol x , which we can use for checking the rest of the formula.

Definition 8.2 Given an annotated type σ , we say that a specification t for x *respects* σ if $S \leq_{\bar{a}} S'$ implies that for each $a \in Spec_S^t$ there exists $b \in Spec_{S'}^t$ such that $a \sqsubseteq^\sigma b$.

It is easy to see that if a specification respects σ , we may assume that the defined value is σ -constant and augment our context with $[x :^G \sigma]$ while checking the rest of the formula.

For specification formats occurring in practice, we can check this property. For brevity, we only consider the case of a recursive function definition $\forall x. f^{\sigma \rightarrow \tau} x \simeq F f x$ where the recursion is known to be terminating (as is required by Isabelle). Termination gives us a well-founded relation $R^{\sigma \rightarrow \sigma \rightarrow o}$ such that

$$\models \forall f g x. (\forall y. R y x \longrightarrow f y \simeq g y) \longrightarrow F f x \simeq F g x. \quad (*)$$

Then it suffices to type-check the functional F , circumventing the quantifiers and equality.

Lemma 8.3 Let σ, τ be annotated types and $s = (\forall x. f^{\sigma \rightarrow \tau} x \simeq F f x)$ a specification for f such that the property $(*)$ holds for some well-founded relation R . If $\square \vdash F : (\sigma \rightarrow_G \tau) \rightarrow_G \tau$, then s respects $\sigma \rightarrow_G \tau$.

Proof Let S, S' be scopes such that $S \leq_{\bar{a}} S'$ and $\mathcal{M}, \mathcal{M}'$ be models for S and S' that both satisfy s . Let $\hat{f} \in Spec_S^s$ and $\hat{g} \in Spec_{S'}^s$. We show $\hat{f} \sqsubseteq^{\sigma \rightarrow_G \tau} \hat{g}$.

Let $\prec = \llbracket R \rrbracket_{\mathcal{M}'}$ and $F_{\mathcal{M}}(f, x) = \llbracket F \rrbracket_{\mathcal{M}}(f)(x)$. By well-founded induction on $b \in \llbracket \sigma \rrbracket_{S'}$ using the relation \prec , we show that $\forall a. a \sqsubseteq^\sigma b \longrightarrow \hat{f}(a) \sqsubseteq^\tau \hat{g}(b)$. As induction hypothesis we have $\forall b' \prec b. \forall a'. a' \sqsubseteq^\sigma b' \longrightarrow \hat{f}(a') \sqsubseteq^\tau \hat{g}(b')$. We pick an arbitrary extension $g \sqsupseteq^{\sigma \rightarrow_G \tau} \hat{f}$, and define the modified function g' as follows:

$$g'(x) = \begin{cases} \hat{g}(x) & \text{if } x \prec b \\ g(x) & \text{otherwise.} \end{cases}$$

From the induction hypothesis, we know that $f \sqsubseteq^{\sigma \rightarrow_G \tau} g'$ and can thus use the typing for F (and Theorem 7.15) to conclude $F_S(\hat{f}, a) \sqsubseteq^\tau F_{S'}(g', b)$. Moreover, the condition $(*)$ implies that $F_{S'}(g', b) = F_{S'}(\hat{g}, b)$, since g' and \hat{g} behave the same on \prec -smaller elements. Thus, unfolding the fixpoint equation (which holds in \mathcal{M} and \mathcal{M}'), we finally get

$$\hat{f}(a) = F_S(\hat{f}, a) \sqsubseteq^\tau F_{S'}(g', b) = F_{S'}(\hat{g}, b) = \hat{g}(b). \quad \square$$

8.2 Inductive Datatypes

The most important way of introducing new types in Isabelle/HOL is to declare an inductive datatype using the command

$$\text{datatype } \bar{a} \ \kappa = C_1 \ \sigma_{11} \dots \sigma_{1k_1} \mid \dots \mid C_n \ \sigma_{n1} \dots \sigma_{nk_n}$$

Inductive datatypes are a derived concept in HOL [2]. However, our analysis benefits from treating them specially as opposed to unfolding the underlying construction.

The datatype declaration introduces the type constructor κ , together with the term constructors C_i of type $\sigma_{i1} \rightarrow_{\mathcal{G}} \dots \rightarrow_{\mathcal{G}} \sigma_{ik_i} \rightarrow_{\mathcal{G}} \bar{\alpha} \kappa$. The type $\bar{\alpha} \kappa$ may occur recursively in the σ_{ij} 's, but only in positive positions. For simplicity, we assume that any arrows in the σ_{ij} 's already carry annotations. (In the implementation, annotation variables are used to infer them.) The interpretation $\llbracket \bar{\alpha} \kappa \rrbracket_S$ is given by the corresponding free term algebra.

We must now extend the basic definitions of \sqsubseteq , \leq , and TV^s to this new construct. For Definition 7.2, we add the following case:

$$C_i(a_1, \dots, a_{k_i}) \sqsubseteq^{\bar{\tau} \kappa} C_i(b_1, \dots, b_{k_i}) \quad \text{iff } \forall j \in \{1, \dots, k_i\}. a_j \sqsubseteq^{\sigma_{ij}[\bar{\alpha} \mapsto \bar{\tau}]} b_j.$$

Similarly, Definition 7.3 is extended with

$$\text{TV}^s(\bar{\tau} \kappa) = \bigcup_{\substack{1 \leq i \leq n \\ 1 \leq j \leq k_i}} \text{TV}^s(\sigma_{ij}[\bar{\alpha} \mapsto \bar{\tau}])$$

and Definition 7.12 with

$$\frac{\sigma_{ij}[\bar{\alpha} \mapsto \bar{\tau}] \leq \sigma_{ij}[\bar{\alpha} \mapsto \bar{\tau}'] \quad \text{for all } 1 \leq i \leq n, 1 \leq j \leq k_i}{\bar{\tau} \kappa \leq \bar{\tau}' \kappa}.$$

To extend our soundness result, we must show that Lemmas 7.5 to 7.14 still hold. The proofs are straightforward and omitted from this article. Constancy of the datatype constructors also follows directly from the above definitions.

Example 8.4 Consider the type α list of lists over α equipped with the constructors $\llbracket \cdot \rrbracket^{\alpha \text{ list}}$ and $\cdot^{\alpha \rightarrow \alpha \text{ list} \rightarrow \alpha \text{ list}}$. A theory of lists could comprise the following definitions:

$$\begin{array}{ll} \llbracket \cdot \rrbracket^{\alpha \text{ list}} ys \simeq ys & (x \cdot xs) \frown ys \simeq x \cdot (xs \frown ys) \\ \text{set } \llbracket \cdot \rrbracket^{\alpha \text{ list}} \simeq \emptyset & \text{set } (x \cdot xs) \simeq \{x\} \cup \text{set } xs \\ \text{dist } \llbracket \cdot \rrbracket^{\alpha \text{ list}} \simeq \top & \text{dist } (x \cdot xs) \simeq (x \notin \text{set } xs \wedge \text{dist } xs). \end{array}$$

The table below presents the results of our analyses on three theorems about lists adapted from Isabelle's *List* theory.

FORMULA	MONOTONIC			ANTIMONO.		
	\mathfrak{M}_1	\mathfrak{M}_2	\mathfrak{M}_3	\mathfrak{M}_1	\mathfrak{M}_2	\mathfrak{M}_3
$\text{set } (xs \frown ys) \simeq \text{set } xs \cup \text{set } ys$.	✓	✓	✓	✓	✓
$\text{dist } (xs \frown ys) \longrightarrow \text{dist } xs \wedge \text{dist } ys$	✓	✓	✓	✓	✓	✓
$\text{dist } (xs \frown ys) \longrightarrow \text{set } xs \cap \text{set } ys \simeq \emptyset$.	✓	✓	✓	✓	✓

8.3 Evaluation

What proportion of monotonic formulas are detected as such by our calculi? We applied Nitpick's implementations of \mathfrak{M}_1 , \mathfrak{M}_2 , and \mathfrak{M}_3 on the user-supplied theorems from six highly polymorphic Isabelle theories. In the spirit of counterexample generation, we conjoined the negated theorems with the relevant axioms. The results are given below.

THEORY	FORMULAS				SUCCESS RATE		
	\mathfrak{M}_1	\mathfrak{M}_2	\mathfrak{M}_3	TOTAL	\mathfrak{M}_1	\mathfrak{M}_2	\mathfrak{M}_3
<i>AVL2</i>	29	33	33	33	88%	100%	100%
<i>Fun</i>	71	94	116	118	60%	80%	98%
<i>Huffman</i>	46	91	90	99	46%	92%	91%
<i>List</i>	441	510	545	659	67%	77%	83%
<i>Map</i>	113	117	117	119	95%	98%	98%
<i>Relation</i>	64	87	100	155	41%	56%	65%

The table indicates how many formulas were found to involve at least one monotonic type variable using \mathfrak{M}_1 , \mathfrak{M}_2 , and \mathfrak{M}_3 , over the total number of formulas involving type variables in the six theories. Since the formulas are all negated theorems, they are all semantically monotonic (no models exist for any scope).

An ideal way to assess the calculi would have been to try them on a representative database including (negated) non-theorems, but we lack such a database. Nonetheless, our experience suggests that the calculi perform as well on non-theorems as on theorems, because realistic non-theorems tend to use equality and quantifiers in essentially the same way as theorems. Interestingly, non-theorems that are derived from theorems by omitting an assumption or mistyping a variable name are even more likely to pass the monotonicity check than the corresponding theorems.

Although the study of monotonicity is interesting in its own right and leads to an elegant theory, our main motivation—speeding up model finders—is resolutely pragmatic. For Nitpick, which uses a default upper bound of 8 on the cardinality of the atomic types, we observed a speed increase factor of about 5 per inferred monotonic type. Since each monotonic type reduces the number of scopes to consider by a factor of 8, we could perhaps expect an 8-fold speed increase; however, the scopes that can be omitted by exploiting monotonicity are smaller and faster to check than those that are actually checked. The time spent performing the monotonicity analysis (i.e., generating the annotation constraints and solving the resulting SAT problem) for \mathfrak{M}_2 is negligible; for \mathfrak{M}_3 , the SAT solver occasionally reached the time limit of one second, which explains why \mathfrak{M}_2 beat \mathfrak{M}_3 on the *Huffman* theory.

9 Conclusion

In model finders that work by enumerating scopes (domain cardinalities specifications), the choice of the scopes and their order is critical to obtain good performance, especially for formulas involving many atomic types. Yet, little work has been done on this problem beyond the discovery of small model theorems.

We presented a solution for HOL that prunes the search space by inferring monotonicity with respect to atomic types. Monotonicity is in general undecidable, so we approximate it with syntactic criteria. The main difficulty occurs in conjunction with common set idioms, which we detect using a suitable type system. Our approach also handles datatypes defined in terms of the atomic types.

Our measurements show that monotonic formulas are pervasive in HOL formalizations and that syntactic criteria can usually detect them. Our calculus \mathfrak{M}_3 has been implemented as part of Isabelle’s SAT-based counterexample generator Nitpick, with dramatic speed gains. It will be interesting to see whether this success can be repeated in the context of other model finders.

Acknowledgment. We would like to thank Lukas Bulwahn, Ann Lillieström, Tobias Nipkow, Andrei Popescu, Mark Summerfield, and the anonymous reviewers for suggesting several textual improvements, as well as Chad Brown and Pascal Fontaine, whose feedback on the IJCAR 2010 talk helped enrich the article.

References

1. P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof (2nd Ed.)*, volume 27 of *Applied Logic*. Springer, 2002.
2. S. Berghofer and M. Wenzel. Inductive datatypes in HOL—lessons learned in formal-logic engineering. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *TPHOLs '99*, volume 1690 of *LNCS*, pages 19–36, 1999.
3. J. C. Blanchette and A. Krauss. Monotonicity inference for higher-order formulas. In J. Giesl and R. Hähnle, editors, *IJCAR 2010*, volume 6173 of *LNCS*, pages 91–106. Springer, 2010.
4. J. C. Blanchette and T. Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In M. Kaufmann and L. Paulson, editors, *ITP 2010*, volume 6172 of *LNCS*, pages 131–146. Springer, 2010.
5. K. Claessen. Private communication, 2009.
6. K. Claessen and N. Sörensson. New techniques that improve MACE-style model finding. In *MODEL*, 2003.
7. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.
8. J. Harrison. HOL Light: A tutorial introduction. In *FMCAD '96*, volume 1166 of *LNCS*, pages 265–269. Springer, 1996.
9. D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.
10. D. Jackson, I. Shlyakhter, and M. Sridharan. A micromodularity mechanism. In *FSE/ESEC 2001*, pages 62–73, 2001.
11. V. Kuncak and D. Jackson. Relational analysis of algebraic datatypes. In H. C. Gall, editor, *ESEC/FSE 2005*, 2005.
12. W. McCune. A Davis–Putnam program and its application to finite first-order model search: Quasigroup existence problems. Technical report, ANL, 1994.
13. J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.
14. L. Momtahan. Towards a small model theorem for data independent systems in Alloy. *Electr. Notes Theor. Comput. Sci.*, 128(6):37–52, 2005.
15. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Prog. Lang. Sys.*, 1(2):245–257, 1979.
16. T. Nipkow. Verifying a hotel key card system. In K. Barkaoui, A. Cavalcanti, and A. Cerone, editors, *ICTAC 2006*, volume 4281 of *LNCS*, pages 1–14. Springer, 2006.
17. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
18. A. Pnueli, Y. Rodeh, O. Strichman, and M. Siegel. The small model property: How small can it be? *Inf. Comput.*, 178(1):279–293, 2002.
19. K. Slind and M. Norrish. A brief overview of HOL4. In O. A. Mohamed, C. Muñoz, and S. Tahar, editors, *TPHOLs 2008*, volume 5170 of *LNCS*, pages 28–32, 2008.
20. C. Tinelli and C. Zarba. Combining decision procedures for sorted theories. In J. Alferes and J. Leite, editors, *JELIA 2004*, volume 3229 of *LNCS*, pages 641–653. Springer, 2004.
21. E. Torlak and D. Jackson. Kodkod: A relational model finder. In O. Grumberg and M. Huth, editors, *TACAS 2007*, volume 4424 of *LNCS*, pages 632–647. Springer, 2007.
22. D. Walker. Substructural type systems. In B. Pierce, editor, *Advanced Topics in Types and Programming Languages*, pages 3–44. MIT Press, 2005.
23. T. Weber. *SAT-Based Finite Model Generation for Higher-Order Logic*. Ph.D. thesis, Dept. of Informatics, T.U. München, 2008.
24. J. Zhang and H. Zhang. SEM: A system for enumerating models. In C. S. Mellish, editor, *IJCAI 95*, volume 1, pages 298–303. Morgan Kaufmann, 1995.