Monotonicity Inference for Higher-Order Formulas

Jasmin Christian Blanchette* and Alexander Krauss

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

Abstract. Formulas are often monotonic in the sense that if the formula is satisfiable for given domains of discourse, it is also satisfiable for all larger domains. Monotonicity is undecidable in general, but we devised two calculi that infer it in many cases for higher-order logic. The stronger calculus has been implemented in Isabelle's model finder Nitpick, where it is used to prune the search space, leading to dramatic speed improvements for formulas involving many atomic types.

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:

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

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 a specification (e.g., to check if a set of axioms is satisfiable) and for producing counterexamples. Notable model finders include Paradox [5], MACE [11], and SEM [19] for first-order logic (FOL), Alloy [9] and Kodkod [17] for first-order relational logic, and Nitpick [3] and Refute [18] for higher-order logic (HOL).

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 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 require 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.

* Research supported by the DFG grant Ni 491/11-1.

Monotonicity occurs surprisingly often in practice. As a real-world example, consider the specification of a hotel key card system with recordable locks [8, pp. 299–306; 13]. 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. Clearly, a counterexample requiring one room, two guests, and four keys will still be a counterexample if more rooms, guests, or keys are available.

In this paper, we present two calculi for detecting monotonicity of HOL formulas. The first calculus (Sect. 5) is based on the idea of tracking the use of equality and quantifiers. Although useful on its own, it mainly serves as a stepping stone for a second, refined calculus (Sect. 6), which uses a type system to detect the ubiquitous "sets as predicates" idiom and treats it specially. The soundness proof of the refined calculus explicitly relates models of different sizes. Both calculi are readily adapted to handle inductive datatypes (Sect. 7), which are pervasive in HOL formalizations. Our evaluation is done in the context of Nitpick (Sect. 8), a counterexample generator for Isabelle/HOL [14]. Although the focus is on HOL, the approach could be adapted to any logic that provides unbounded quantification, such as many-sorted FOL with equality.

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 monotonicity property is essentially a weak form of the upward Löwenheim–Skolem theorem.

When equality is added, nonmonotonic formulas follow suit. For example, the formula $\forall x\, y.\ x = y$ is satisfied only by singleton models. Nonetheless, $\forall x\, y.\ x = y \longrightarrow P(x,y)$ is monotonic, because equality occurs only negatively. Distinguishing between positive and negative occurrences of equality is a natural syntactic criterion for detecting monotonicity, and our approach is based on this idea.

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 $(\alpha \to \beta) \to \beta$ and in datatypes such as $\alpha\ \mathit{list}$. We are not aware of any previous work on detecting 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 Alloy language ensured monotonicity with respect to the primitive types by providing only bounded quantification and disallowing explicit references to the sets that denote the types [9]. Monotonicity has been lost in more recent versions of the language, which allow such references [8, p. 165]. Nonetheless, many Alloy formulas are monotonic, notably the existential–bounded-universal class of formulas studied by Kuncak and Jackson [10].

For some logics, small model theorems give an upper bound on the cardinality of a sort [4], primitive type [12], or variable's domain [15]. 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 o of Booleans:*

Types:		*Terms:*	
$\sigma ::= o$	(*Boolean type*)	$t ::= x^\sigma$	(*variable*)
$\mid \alpha$	(*type variable*)	$\mid c^\sigma$	(*constant*)
$\mid \sigma \to \sigma$	(*function type*)	$\mid t\ t$	(*application*)
		$\mid \lambda x^\sigma.\, t$	(*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 often omit the type superscripts. A formula is a term of type o.

Unlike in Gordon's version of HOL [6], on which several popular proof assistants are based [7, 14, 16], we treat polymorphism in the metalanguage: Polymorphic constants such as equality are expressed as collections of constants, one for each type.

Types and terms are interpreted in the standard set-theoretic way, relative to a type environment that fixes the interpretation of type variables.

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

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. Often, scopes are also called "type environments"; our terminology here is consistent with Jackson [9].

Definition 3.3 (Interpretation of Types). *The* interpretation $[\![\sigma]\!]_S$ *of type σ in scope S is defined recursively by the equations*

$$[\![o]\!]_S = \{\bot, \top\} \qquad [\![\alpha]\!]_S = S(\alpha) \qquad [\![\sigma \to \tau]\!]_S = [\![\sigma]\!]_S \to [\![\tau]\!]_S$$

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

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

Definition 3.5 (Interpretation of Terms). *Let $\mathcal{M} = (S, A, M)$ be a model. The* interpretation $[\![t]\!]_\mathcal{M}$ *of a term t is defined recursively by the equations*

$$[\![x]\!]_{(S,A,M)} = A(x) \qquad [\![t\ u]\!]_{(S,A,M)} = [\![t]\!]_{(S,A,M)}\left([\![u]\!]_{(S,A,M)}\right)$$
$$[\![c]\!]_{(S,A,M)} = M_S(c) \qquad [\![\lambda x^\sigma.\, t]\!]_{(S,A,M)} = a \in [\![\sigma]\!]_S \mapsto [\![t]\!]_{(S, A[x \mapsto a], M)}.$$

If t is a formula and $[\![t]\!]_\mathcal{M} = \top$, 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.*

We use constants to express the logical primitives, whose interpretation is fixed a priori for each scope. Our definition of HOL is fairly minimalistic, with equality ($=^{\sigma\to\sigma\to o}$ for any σ) and implication ($\longrightarrow^{o\to o\to o}$) as primitive constants. In the sequel, we always use the standard constant model $\widehat{M}$, which interprets implication and equality in the standard way, and we omit the last component of $(S, A, \widehat{M})$.

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

Notation 3.1 (Logical Abbreviations).

$$\begin{aligned} \mathit{True} &\equiv (\lambda x^o.\ x) = (\lambda x.\ x) & p \wedge q &\equiv \neg (p \longrightarrow \neg q)\\ \mathit{False} &\equiv (\lambda x^o.\ x) = (\lambda x.\ \mathit{True}) & p \vee q &\equiv \neg p \longrightarrow q\\ \neg p &\equiv p \longrightarrow \mathit{False} & \forall x^\sigma.\ p &\equiv (\lambda x.\ p) = (\lambda x.\ \mathit{True})\\ p \neq q &\equiv \neg p = q & \exists x^\sigma.\ p &\equiv \neg \forall x.\ \neg p. \end{aligned}$$

Notation 3.2 (Set Abbreviations).

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

The constants $\emptyset$ and *insert* can be seen as constructors for finite sets. Following tradition, we write $\{x_1, \ldots, x_n\}$ rather than $\mathit{insert}\ x_1\ (\ldots(\mathit{insert}\ x_n\ \emptyset)\ldots)$.

4 Monotonicity

The introduction gave an informal definition of monotonicity. A more rigorous definition follows.

Definition 4.1 (Monotonicity). *A formula t is* monotonic *w.r.t. 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 *w.r.t. α if its negation is monotonic w.r.t. α.*

Example 4.1. If you have five Swedish friends and all five are blond, the existential statement "at least one of your Swedish 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 Swedish friends are blond" is antimonotonic—it will either stay true or become false as you make new friends. ■

Theorem 4.1 (Undecidability). *Monotonicity w.r.t. α is undecidable.*

Proof (reduction). For any closed HOL formula t, let $t^\star \equiv t \vee \forall x^\alpha\, y.\ x = y$, where α does not occur in t. Clearly, $t^\star$ must be monotonic if t is valid, since the second disjunct becomes irrelevant in this case. If t is not valid, then $t^\star$ 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. □

The best we can do is approximate monotonicity.

Convention. In the rest of this paper, we denote by $\tilde{\alpha}$ the type variable w.r.t. which we consider monotonicity.

5 A Simple Calculus

This section presents the simple calculus $\mathfrak{M}_{\mathsf{F}}$ for inferring monotonicity. This simple calculus serves as a stepping stone toward the more general calculus $\mathfrak{M}_{\mathsf{FS}}$ of Sect. 6. (The 'F' in $\mathfrak{M}_{\mathsf{F}}$ and $\mathfrak{M}_{\mathsf{FS}}$ stands for "function," whereas 'S' stands for "set.") Since the results in this section are subsumed by those of the next section, we omit the proofs.

5.1 Extension Relation and Constancy

We first introduce a concept that is similar to monotonicity but that applies 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 o 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. In the case of 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 $f^{\tilde{\alpha}\to o} = [0 \mapsto \top]$ can be extended to either $g = [0 \mapsto \top, 1 \mapsto \bot]$ or $g' = [0 \mapsto \top, 1 \mapsto \top]$. For now, we take the liberal view that both g and g' are "essentially the same value" as f, which we write $f \sqsubseteq^{\tilde{\alpha}\to o} g$ and $f \sqsubseteq^{\tilde{\alpha}\to o} g'$. We will reconsider this decision in Sect. 6.

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

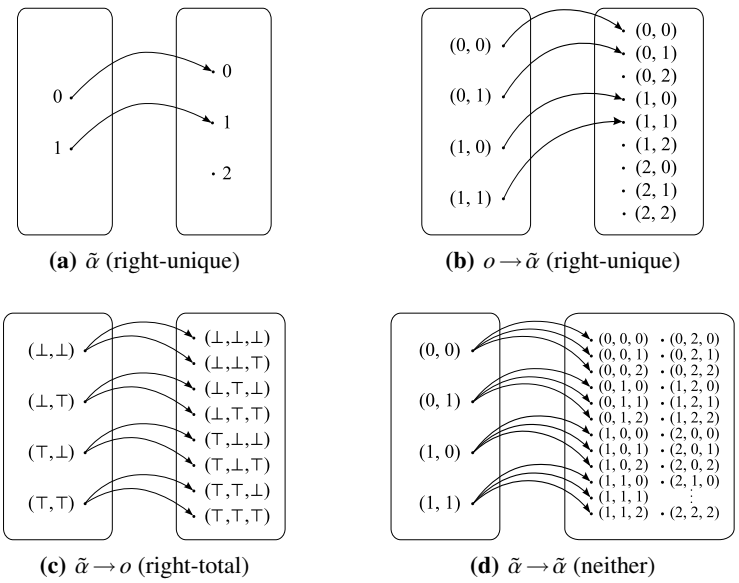
$$\begin{aligned} a \sqsubseteq^{\sigma} b \quad &\mathit{iff} \quad a = b && \mathit{if}\ \sigma\ \mathit{is}\ o\ \mathit{or a type variable} \\ f \sqsubseteq^{\sigma\to\tau} g \quad &\mathit{iff} \quad \forall a\, b.\, a \sqsubseteq^{\sigma} b \longrightarrow f(a) \sqsubseteq^{\tau} g(b). \end{aligned}$$

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

The symbol $\sqsubseteq^{\sigma}$ is read "is extended by." Fig. 5.1 illustrates $\sqsubseteq^{\sigma}$ for various types. We represent a function from σ to τ by a $|\sigma|$-tuple such that the nth element for σ (according to the lexicographic order, with $\bot < \top$ and $n < n+1$) is mapped to the nth tuple component. Observe that $\sqsubseteq^{\sigma}$ is always left-total ($\forall a.\, \exists b.\, a \sqsubseteq^{\sigma} b$) and left-unique (i.e., injective: $\forall a\, a'\, b.\, a \sqsubseteq^{\sigma} b \wedge a' \sqsubseteq^{\sigma} b \longrightarrow a = a'$). 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., $\sigma = \tilde{\alpha} \to o$), 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., $\sigma = o \to \tilde{\alpha}$). These properties are crucial to the correctness of our calculus.

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

Example 5.1. $f^{\tilde{\alpha}\to\tilde{\alpha}}\, x$ is constant. Proof: Let $A(x) = a_1$ and $A(f)(a_1) = a_2$. For any $\mathcal{M}' = (S', A')$ that extends $\mathcal{M} = (S, A)$, we have $A(x) \sqsubseteq^{\tilde{\alpha}} A'(x)$ and $A(f) \sqsubseteq^{\tilde{\alpha}\to\tilde{\alpha}} A'(f)$. By definition of $\sqsubseteq^{\sigma}$, $A'(x) = a_1$ and $A'(f)(a_1) = a_2$. Thus, $[\![f\, x]\!]_{\mathcal{M}} = [\![f\, x]\!]_{\mathcal{M}'} = a_2$. ■



(a) $\tilde{\alpha}$ (right-unique) **(b)** $o \to \tilde{\alpha}$ (right-unique)

(c) $\tilde{\alpha} \to o$ (right-total) **(d)** $\tilde{\alpha} \to \tilde{\alpha}$ (neither)

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

Example 5.2. $x^{o \to \tilde{\alpha}} = y$ is constant. Proof: For any $\mathcal{M}' = (S', A')$ that extends $\mathcal{M} = (S, A)$, we have $A(x) \sqsubseteq^{o \to \tilde{\alpha}} A'(x)$ and $A(y) \sqsubseteq^{o \to \tilde{\alpha}} A'(y)$. By definition of $\sqsubseteq^{\sigma}$, $A'(x) = A(x)$ and $A'(y) = A(y)$. Hence, $[\![x = y]\!]_{\mathcal{M}} = [\![x = y]\!]_{\mathcal{M}'}$. ■

Example 5.3. $f^{\tilde{\alpha} \to o} = g$ is not constant. Counterexample: $|S(\tilde{\alpha})| = 1$, $A(f) = A(g) = (\top)$, $|S'(\tilde{\alpha})| = 2$, $A'(f) = (\top, \bot)$, $A'(g) = (\top, \top)$. Then $[\![f = g]\!]_{(S,A)} = \top$ but $[\![f = g]\!]_{(S',A')} = \bot$. ■

More generally, we note that variables are always constant, and constancy is preserved by application and λ-abstraction. On the other hand, the equality symbol $=^{\sigma \to \sigma \to o}$ 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.

5.2 Syntactic Criteria

We syntactically approximate constancy, monotonicity, and antimonotonicity with the predicates $\mathbf{K}(t)$, $\mathbf{M}^{+}(t)$, and $\mathbf{M}^{-}(t)$, respectively. The goal is to derive $\mathbf{M}^{+}(t)$ for the formula t we wish to prove monotonic. The predicates depend on $\mathbf{TV}^{+}(\sigma)$ and $\mathbf{TV}^{-}(\sigma)$, which collect the positive and negative type variables of σ.

Definition 5.4 (Positive and Negative Type Variables). *The sets of* positive type variables $\mathbf{TV}^{+}(\sigma)$ *and of* negative type variables $\mathbf{TV}^{-}(\sigma)$ *of a type σ are defined as follows:*

$$\mathbf{TV}^{+}(\alpha) = \{\alpha\} \qquad \mathbf{TV}^{s}(o) = \emptyset$$
$$\mathbf{TV}^{-}(\alpha) = \emptyset \qquad \mathbf{TV}^{s}(\sigma \to \tau) = \mathbf{TV}^{\bar{s}}(\sigma) \cup \mathbf{TV}^{s}(\tau).$$

If $s = +$, then $\bar{s}$ denotes $-$; otherwise, $\bar{s}$ denotes $+$.

Definition 5.5 (Constancy and Monotonicity Rules). *The predicates* $\mathbf{K}(t)$, $\mathbf{M}^{+}(t)$, *and* $\mathbf{M}^{-}(t)$ *are inductively defined by the rules*

$$\frac{}{\mathbf{K}(x)} \qquad \frac{}{\mathbf{K}(\longrightarrow)} \qquad \frac{\tilde{\alpha} \notin \mathbf{TV}^{-}(\sigma)}{\mathbf{K}(=^{\sigma\to\sigma\to o})} \qquad \frac{\mathbf{K}(t^{\sigma\to\tau}) \quad \mathbf{K}(u^{\sigma})}{\mathbf{K}(t\,u)} \qquad \frac{\mathbf{K}(t)}{\mathbf{K}(\lambda x.\, t)}$$

$$\frac{\mathbf{K}(t)}{\mathbf{M}^{s}(t)} \qquad \frac{\mathbf{M}^{\bar{s}}(t) \quad \mathbf{M}^{s}(u)}{\mathbf{M}^{s}(t \longrightarrow u)} \qquad \frac{\mathbf{K}(t^{\sigma}) \quad \mathbf{K}(u^{\sigma})}{\mathbf{M}^{-}(t = u)} \qquad \frac{\mathbf{M}^{-}(t)}{\mathbf{M}^{-}(\forall x.\, t)} \qquad \frac{\mathbf{M}^{+}(t) \quad \tilde{\alpha} \notin \mathbf{TV}^{+}(\sigma)}{\mathbf{M}^{+}(\forall x^{\sigma}.\, t)}.$$

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

- The $\mathbf{M}^{-}(t = u)$ rule is sound because the extensions of distinct elements are always distinct (since $\sqsubseteq^{\sigma}$ is left-unique).
- The $\mathbf{M}^{-}(\forall x.\, t)$ 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 $\mathbf{M}^{+}(\forall x.\, t)$ 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 $a \in [\![\sigma]\!]_S$ but not for some $b \in [\![\sigma]\!]_{S'}$. However, in Sect. 6 we will show that this can only happen for b's that do not extend any a, which can only exist if $\alpha \in \mathbf{TV}^{+}(\sigma)$.

Example 5.4. The following derivation shows that $\forall x^{\alpha\to o}.\, P\,x$ is monotonic w.r.t. α:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{K}(P)} \quad \dfrac{}{\mathbf{K}(x)}}{\mathbf{K}(P\,x)}}{\mathbf{M}^{+}(P\,x)} \qquad \alpha \notin \mathbf{TV}^{+}(\alpha \to o)}{\mathbf{M}^{+}(\forall x^{\alpha\to o}.\, P\,x)}$$

■

Example 5.5. Formula 4 from Sect. 1 is monotonic, but $\mathbf{M}^{+}$ fails on it:

$$\dfrac{\dfrac{\dfrac{\alpha \notin \mathbf{TV}^{-}(\alpha \to o)}{\mathbf{K}(=^{(\alpha\to o)\to(\alpha\to o)\to o})} \quad \dfrac{\vdots}{\mathbf{K}(\{y\})}}{\mathbf{K}((=)\ \{y\})} \quad \dfrac{\vdots}{\mathbf{K}(\{z\})}}{\dfrac{\mathbf{K}(\{y\} = \{z\})}{\mathbf{M}^{+}(\{y\} = \{z\})}}$$

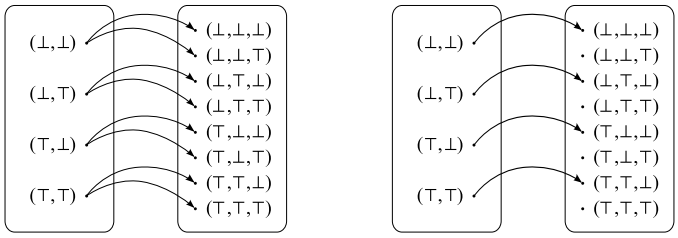
The assumption $\alpha \notin \mathbf{TV}^{-}(\alpha \to o)$ cannot be discharged, since $\mathbf{TV}^{-}(\alpha \to o) = \{\alpha\}$. ■

The last example exhibits a significant weakness of the calculus $\mathfrak{M}_{\mathsf{F}}$. HOL identifies sets with predicates, yet $\mathbf{M}^{+}$ prevents us from comparing terms of type $\tilde{\alpha} \to o$ for equality. This happens because the extension of a function of this type is not unique (cf. Fig. 5.1(c)), and thus equality is generally not preserved as we enlarge the scope.

This behavior of $\sqsubseteq^{\tilde{\alpha}\to o}$ is imprecise for sets, as it puts distinct sets in relation; for example, $\{0\} \sqsubseteq^{\tilde{\alpha}\to o} \{0,2\}$ if $S(\tilde{\alpha}) = 2$ and $S'(\tilde{\alpha}) = 3$. We would normally prefer each set to admit a unique extension, namely the set itself. This would make set equality constant. The next section introduces a refined calculus that formalizes this idea.

6 A Refined Calculus

To solve the problem sketched above, we introduce an alternative version of $\sqsubseteq^\sigma$ such that the extension of a set is always the set itself. Rephrased in terms of functions, this means that the extended function must return $\bot$ for all elements that are "new" in the larger scope. Fig. 6.1 compares this more conservative "set" approach to the liberal "functional" approach of Sect. 5; in subfigure (b), it may help to think of $(\bot,\bot)$ and $(\bot,\bot,\bot)$ as $\emptyset$, $(\top,\bot)$ and $(\top,\bot,\bot)$ as $\{0\}$, and so on.



(a) functional view (right-total) **(b)** set view (right-unique)

Figure 6.1. $\sqsubseteq^{\tilde{\alpha}\to o}$ with $S(\tilde{\alpha}) = 2$ and $S'(\tilde{\alpha}) = 3$

With this approach, we could easily infer that $\{y\} = \{z\}$ is constant. However, the wholesale application of this principle would have pernicious consequences on constancy: Semantically, the universal set $\mathcal{U}^{\tilde{\alpha}\to o}$, among others, would no longer be constant; syntactically, the introduction rule for $\mathbf{K}(\lambda x.\, t)$ would no longer be sound.

What we propose instead is a hybrid approach that supports both forms of extensions in various combinations. The required bookkeeping is conveniently expressed as a type system, in which each function arrow is annotated with F ("function") or S ("set"):

Definition 6.1 (Annotated Types). *An* annotated type *is a HOL type in which each function arrow carries an* annotation $X \in \{\mathsf{F},\mathsf{S}\}$.

The annotations have no influence on the interpretation of types as sets of values, which is unchanged. Instead, they specify how $\sqsubseteq$ should extend functional values to larger scopes. While F-functions are extended as in the previous section, the extension of an S-function must map all new values to $\bot$.

6.1 Refined Extension Relation

The extension relation $\sqsubseteq^\sigma$ distinguishes between the two kinds of arrows. The F case coincides with Def. 5.1.

Definition 6.2 (Extension Relation). *Let σ be an annotated type, and let S, S' be scopes such that $S \leq_{\tilde{\alpha}} S'$. The* extension relation $\sqsubseteq^\sigma \subseteq [\![\sigma]\!]_S \times [\![\sigma]\!]_{S'}$ *for S and S' is defined by the following equivalences:*

$$
\begin{array}{rll}
a \sqsubseteq^{\sigma} b & \textit{iff} & a = b \qquad\qquad \textit{if } \sigma \textit{ is } o \textit{ or a type variable} \\
f \sqsubseteq^{\sigma \to \mathsf{F} \tau} g & \textit{iff} & \forall a\, b.\, a \sqsubseteq^{\sigma} b \longrightarrow f(a) \sqsubseteq^{\tau} g(b) \\
f \sqsubseteq^{\sigma \to \mathsf{S} \tau} g & \textit{iff} & \forall a\, b.\, a \sqsubseteq^{\sigma} b \longrightarrow f(a) \sqsubseteq^{\tau} g(b) \textit{ and } \forall b.\, (\exists a.\, a \sqsubseteq^{\sigma} b) \vee g(b) = (\!|\tau|\!)_{S'}
\end{array}
$$

where $(\!|o|\!)_S = \bot$, $(\!|\sigma \to \tau|\!)_S = a \in [\![\sigma]\!]_S \mapsto (\!|\tau|\!)_S$, *and* $(\!|\alpha|\!)_S$ *is any element of* $S(\alpha)$.

Although the S annotation is tailored to predicates, the annotated type $\sigma \to_{\mathsf{S}} \tau$ is legal for any type τ. The value $(\!|\tau|\!)_S$ then takes the place of $\bot$ as the default extension.

We now prove the crucial properties of $\sqsubseteq^{\sigma}$, to which we alluded in Sect. 5. The unusual definitions of $\mathbf{TV}^{+}$ and $\mathbf{TV}^{-}$ in the S case ensure that Lem. 6.2 holds uniformly.

Lemma 6.1. *The relation* $\sqsubseteq^{\sigma}$ *is left-total and left-unique* (*injective*).

Proof (structural induction on σ). For o and α, both properties are obvious. For $\sigma \to_{\mathsf{X}} \tau$, we assume by induction that $\sqsubseteq^{\sigma}$ and $\sqsubseteq^{\tau}$ are left-unique and left-total. Since $\sqsubseteq^{\sigma \to \mathsf{S} \tau} \subseteq \sqsubseteq^{\sigma \to \mathsf{F} \tau}$, it suffices to show that $\sqsubseteq^{\sigma \to \mathsf{F} \tau}$ is left-unique and $\sqsubseteq^{\sigma \to \mathsf{S} \tau}$ is left-total.

LEFT-UNIQUENESS: We assume f, $f' \sqsubseteq^{\sigma \to \mathsf{F} \tau} g$ and show that $f = f'$. For every $a \in [\![\sigma]\!]_S$, left-totality of $\sqsubseteq^{\sigma}$ yields an extension b with $a \sqsubseteq^{\sigma} b$. Then $f(a) \sqsubseteq^{\tau} g(b)$ and $f'(a) \sqsubseteq^{\tau} g(b)$, and since $\sqsubseteq^{\tau}$ is left-unique, $f(a) = f'(a)$.

LEFT-TOTALITY: For $f \in [\![\sigma \to \tau]\!]_S$, we find an extension g as follows: Let $b \in [\![\sigma]\!]_{S'}$. If b extends an a, that a is unique by left-uniqueness of $\sqsubseteq^{\sigma}$. Since $\sqsubseteq^{\tau}$ 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 $g(b) = (\!|\tau|\!)_{S'}$. By construction, $f \sqsubseteq^{\sigma \to \mathsf{S} \tau} g$. □

Definition 6.3 (Positive and Negative Type Variables). *The sets of* positive type variables $\mathbf{TV}^{+}(\sigma)$ *and of* negative type variables $\mathbf{TV}^{-}(\sigma)$ *of an annotated type* σ *are defined as follows:*

$$
\begin{array}{rr}
\mathbf{TV}^{+}(o) = \emptyset & \mathbf{TV}^{-}(o) = \emptyset \\
\mathbf{TV}^{+}(\alpha) = \{\alpha\} & \mathbf{TV}^{-}(\alpha) = \emptyset \\
\mathbf{TV}^{+}(\sigma \to_{\mathsf{F}} \tau) = \mathbf{TV}^{-}(\sigma) \cup \mathbf{TV}^{+}(\tau) & \mathbf{TV}^{-}(\sigma \to_{\mathsf{F}} \tau) = \mathbf{TV}^{+}(\sigma) \cup \mathbf{TV}^{-}(\tau) \\
\mathbf{TV}^{+}(\sigma \to_{\mathsf{S}} \tau) = \mathbf{TV}^{+}(\sigma) \cup \mathbf{TV}^{-}(\sigma) \cup \mathbf{TV}^{+}(\tau) & \mathbf{TV}^{-}(\sigma \to_{\mathsf{S}} \tau) = \mathbf{TV}^{-}(\tau).
\end{array}
$$

Lemma 6.2. *If* $\tilde{\alpha} \notin \mathbf{TV}^{+}(\sigma)$, *then* $\sqsubseteq^{\sigma}$ *is right-total* (*surjective*). *If* $\tilde{\alpha} \notin \mathbf{TV}^{-}(\sigma)$, *then* $\sqsubseteq^{\sigma}$ *is right-unique* (*functional*).

Proof (structural induction on σ). For o and α, both properties are obvious. For $\sigma \to_{\mathsf{X}} \tau$, we assume by induction that the implications hold for $\sqsubseteq^{\sigma}$ and $\sqsubseteq^{\tau}$.

RIGHT-UNIQUENESS OF $\sqsubseteq^{\sigma \to \mathsf{F} \tau}$: If $\tilde{\alpha} \notin \mathbf{TV}^{-}(\sigma \to_{\mathsf{F}} \tau) = \mathbf{TV}^{+}(\sigma) \cup \mathbf{TV}^{-}(\tau)$, then by induction hypothesis $\sqsubseteq^{\sigma}$ is right-total and $\sqsubseteq^{\tau}$ is right-unique. We consider g, g' such that $f \sqsubseteq^{\sigma \to \mathsf{F} \tau} g$ and $f \sqsubseteq^{\sigma \to \mathsf{F} \tau} g'$, and show that $g = g'$. For every $b \in [\![\sigma]\!]_{S'}$, right-totality of $\sqsubseteq^{\sigma}$ yields a restriction $a \sqsubseteq^{\sigma} b$. Then $f(a) \sqsubseteq^{\tau} g(b)$ and $f(a) \sqsubseteq^{\tau} g'(b)$, and since $\sqsubseteq^{\tau}$ is right-unique, $g(b) = g'(b)$.

RIGHT-UNIQUENESS OF $\sqsubseteq^{\sigma \to \mathsf{S} \tau}$ AND RIGHT-TOTALITY OF $\sqsubseteq^{\sigma \to \mathsf{X} \tau}$: Omitted. □

Notice how the new definition of $\mathbf{TV}^{+}$ and $\mathbf{TV}^{-}$ solves the problem exhibited by the formula $\{y\} = \{z\}$ (Ex. 5.5), since the $\tilde{\alpha}$ in $\tilde{\alpha} \to_{\mathsf{S}} o$ counts as a positive occurrence. However, we must now ensure that types are consistently annotated.

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 $\sqsubseteq^\sigma$ (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.

Regrettably, our rules cannot derive the desired types for the basic set operations $\cup$, $\cap$, and $-$ when they are defined as abbreviations (cf. Notat. 3.2). This problem is solved by treating set constants as primitive along with implication and equality.

Definition 6.4 (Context). *A* context *is a pair of mappings $\Gamma = (\Gamma_\mathrm{c}, \Gamma_\mathrm{v})$, where Γ_c maps constant symbols to sets of annotated types, and Γ_v maps variables to annotated types. A constant context Γ_c is* compatible *with a constant model M if $\sigma \in \Gamma_\mathrm{c}(c)$ 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 types σ.*

Definition 6.5 (Standard Constant Context). *The* standard constant context $\widehat{\Gamma}_\mathrm{c}$ *is the following mapping:*

$$
\begin{array}{rcll}
\longrightarrow & \mapsto & \{o \to_\mathsf{F} o \to_\mathsf{F} o\} & \\
= & \mapsto & \{\sigma \to_\mathsf{F} \sigma \to_X o & \mid X \in \{\mathsf{F},\mathsf{S}\},\ \tilde{\alpha} \notin \mathbf{TV}^-(\sigma)\} \\
\emptyset & \mapsto & \{\sigma \to_X o & \mid X \in \{\mathsf{F},\mathsf{S}\}\} \\
\mathcal{U} & \mapsto & \{\sigma \to_\mathsf{F} o\} & \\
\cup & \mapsto & \{(\sigma \to_X o) \to_\mathsf{F} (\sigma \to_X o) \to_\mathsf{F} \sigma \to_X o & \mid X \in \{\mathsf{F},\mathsf{S}\}\} \\
\cap & \mapsto & \{(\sigma \to_X o) \to_\mathsf{F} (\sigma \to_X o) \to_\mathsf{F} \sigma \to_X o & \mid X \in \{\mathsf{F},\mathsf{S}\}\} \\
- & \mapsto & \{(\sigma \to_X o) \to_\mathsf{F} (\sigma \to_\mathsf{F} o) \to_\mathsf{F} \sigma \to_X o & \mid X \in \{\mathsf{F},\mathsf{S}\}\} \\
\in & \mapsto & \{\sigma \to_\mathsf{F} (\sigma \to_X o) \to_\mathsf{F} o & \mid X \in \{\mathsf{F},\mathsf{S}\}\} \\
\mathit{insert} & \mapsto & \{\sigma \to_\mathsf{F} (\sigma \to_X o) \to_\mathsf{F} \sigma \to_X o & \mid X \in \{\mathsf{F},\mathsf{S}\},\ \tilde{\alpha} \notin \mathbf{TV}^-(\sigma)\}.
\end{array}
$$

Allowing constants to have multiple annotated types gives us a form of polymorphism on the annotations. We treat $\widehat{\Gamma}_\mathrm{c}$ as a global table of annotated types for constants.

Lemma 6.3. *The standard constant context is compatible with the standard constant model.*

Proof. CASE $=$: Since $\tilde{\alpha} \notin \mathbf{TV}^-(\sigma)$, $\sqsubseteq^\sigma$ is right-unique (Lem. 6.2). 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 $(b = b') = \bot$. The first part follows from the left-uniqueness and right-uniqueness of $\sqsubseteq^\sigma$. For the second part, $b \neq b'$ because a restricts b but b' admits no restriction.

OTHER CASES: Omitted. $\square$

Defs. 5.2 and 5.3 and the **K** part of Def. 5.5 from Sect. 5 are generalized as follows.

Definition 6.6 (Model Extension). *Let $\mathcal{M} = (S, A)$ and $\mathcal{M}' = (S', A')$ 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 $\Gamma_\mathrm{v}(x) = \sigma$ implies $A(x) \sqsubseteq^\sigma A'(x)$ for all x.*

Definition 6.7 (Constancy). *Let σ be an annotated type. A term t is* σ-constant *in a context Γ if $[\![t]\!]_{\mathcal{M}} \sqsubseteq^{\sigma} [\![t]\!]_{\mathcal{M}'}$ for all models $\mathcal{M}$, $\mathcal{M}'$ such that $\mathcal{M} \sqsubseteq_{\Gamma} \mathcal{M}'$.*

Definition 6.8 (Typing Rules). *The typing relation $\Gamma \vdash t : \sigma$ is given by the rules*

$$\frac{\Gamma_{\mathrm{v}}(x) = \sigma}{\Gamma \vdash x : \sigma}\ \textsc{Var} \qquad \frac{\sigma \in \Gamma_{\mathrm{c}}(c)}{\Gamma \vdash c : \sigma}\ \textsc{Const}$$

$$\frac{\Gamma \vdash t : \sigma' \to_{X} \tau \quad \Gamma \vdash u : \sigma \quad \sigma \leq \sigma'}{\Gamma \vdash t\ u : \tau}\ \textsc{App} \qquad \frac{\Gamma[x \mapsto \sigma] \vdash t : \tau}{\Gamma \vdash \lambda x.\, t : \sigma \to_{\mathsf{F}} \tau}\ \textsc{Lam}$$

where $X \in \{\mathsf{F}, \mathsf{S}\}$ and the subtype relation $\sigma \leq \tau$ is defined by the rules

$$\frac{}{o \leq o} \qquad \frac{}{\alpha \leq \alpha} \qquad \frac{\sigma' \leq \sigma \quad \tau \leq \tau'}{\sigma \to_{X} \tau \leq \sigma' \to_{\mathsf{F}} \tau'} \qquad \frac{\tau \leq \tau'}{\sigma \to_{\mathsf{S}} \tau \leq \sigma \to_{\mathsf{S}} \tau'}.$$

In the above definition, $\Gamma[x \mapsto \sigma]$ abbreviates $(\Gamma_{\mathrm{c}}, \Gamma_{\mathrm{v}}[x \mapsto \sigma])$.

Lemma 6.4. *If $\sigma \leq \sigma'$, then $\sqsubseteq^{\sigma} \subseteq \sqsubseteq^{\sigma'}$.*

Proof. By straightforward induction on the derivation of $\sigma \leq \sigma'$. □

Theorem 6.1 (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: Obvious, since $A(x) \sqsubseteq^{\sigma} A'(x)$ by assumption for $\sigma = \Gamma_{\mathrm{v}}(x)$.

Const: Obvious, since $\widehat{M}_S(c) \sqsubseteq^{\sigma} \widehat{M}_{S'}(c)$ by assumption for all $\sigma \in \Gamma_{\mathrm{c}}(c)$.

App: By induction hypothesis, and since $\sqsubseteq^{\sigma' \to_{\mathsf{S}} \tau} \subseteq \sqsubseteq^{\sigma' \to_{\mathsf{F}} \tau}$, we have $[\![t]\!]_{\mathcal{M}} \sqsubseteq^{\sigma' \to_{\mathsf{F}} \tau} [\![t]\!]_{\mathcal{M}'}$ and $[\![u]\!]_{\mathcal{M}} \sqsubseteq^{\sigma} [\![u]\!]_{\mathcal{M}'}$. Lem. 6.4 and the condition $\sigma \leq \sigma'$ imply that $[\![u]\!]_{\mathcal{M}} \sqsubseteq^{\sigma'} [\![u]\!]_{\mathcal{M}'}$. Then by Def. 6.2, we know that $[\![t\ u]\!]_{\mathcal{M}} = [\![t]\!]_{\mathcal{M}}([\![u]\!]_{\mathcal{M}}) \sqsubseteq^{\tau} [\![t]\!]_{\mathcal{M}'}([\![u]\!]_{\mathcal{M}'}) = [\![t\ u]\!]_{\mathcal{M}'}$, which shows that $t\ u$ is τ-constant in Γ.

Lam: Let $a \in [\![\sigma]\!]_S$ and $b \in [\![\sigma]\!]_{S'}$ such that $a \sqsubseteq^{\sigma} b$. Then we have the extended models $\mathcal{M}_a = (S, A[x \mapsto a])$ and $\mathcal{M}'_b = (S', A'[x \mapsto b])$. Thus, $\mathcal{M}_a \sqsubseteq_{\Gamma[x \mapsto \sigma]} \mathcal{M}'_b$, and by induction hypothesis $[\![\lambda x.\, t]\!]_{\mathcal{M}}(a) = [\![t]\!]_{\mathcal{M}_a} \sqsubseteq^{\tau} [\![t]\!]_{\mathcal{M}'_b} = [\![\lambda x.\, t]\!]_{\mathcal{M}'}(b)$. This implies $[\![\lambda x.\, t]\!]_{\mathcal{M}} \sqsubseteq^{\sigma \to_{\mathsf{F}} \tau} [\![\lambda x.\, t]\!]_{\mathcal{M}'}$. □

6.3 Monotonicity Checking

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

Definition 6.9 (Monotonicity Rules). *The predicates $\Gamma \vdash \mathbf{M}^{+}(t)$ and $\Gamma \vdash \mathbf{M}^{-}(t)$ are given by the rules*

$$\frac{\Gamma \vdash t : o}{\Gamma \vdash \mathbf{M}^{s}(t)}\ \textsc{Term} \qquad \frac{\Gamma \vdash \mathbf{M}^{\bar{s}}(t) \quad \Gamma \vdash \mathbf{M}^{s}(u)}{\Gamma \vdash \mathbf{M}^{s}(t \longrightarrow u)}\ \textsc{Imp} \qquad \frac{\Gamma \vdash t : \sigma \quad \Gamma \vdash u : \sigma}{\mathbf{M}^{-}(t = u)}\ \textsc{Eq}^{-}$$

$$\frac{\Gamma[x \mapsto \sigma] \vdash \mathbf{M}^{-}(t)}{\Gamma \vdash \mathbf{M}^{-}(\forall x.\, t)}\ \textsc{All}^{-} \qquad \frac{\Gamma[x \mapsto \sigma] \vdash \mathbf{M}^{+}(t) \quad \tilde{\alpha} \notin \mathbf{TV}^{+}(\sigma)}{\Gamma \vdash \mathbf{M}^{+}(\forall x.\, t)}\ \textsc{All}^{+}.$$

Theorem 6.2 (Soundness of $\mathbf{M}^s$). *Let $\mathcal{M}$ and $\mathcal{M}'$ be models such that $\mathcal{M} \sqsubseteq_\Gamma \mathcal{M}'$. If $\Gamma \vdash \mathbf{M}^+(t)$, then $\mathcal{M} \vDash t$ implies $\mathcal{M}' \vDash t$. If $\Gamma \vdash \mathbf{M}^-(t)$, then $\mathcal{M} \nvDash t$ implies $\mathcal{M}' \nvDash t$.*

Proof (induction on the derivation of $\Gamma \vdash \mathbf{M}(t)$). Let $\mathcal{M} = (S, A)$ and $\mathcal{M}' = (S', A')$.

TERM: Because constancy implies (anti)monotonicity for type o.

IMP: Obvious.

EQ$^-$: Assume that $\Gamma \vdash s : \sigma$, $\Gamma \vdash t : \sigma$, and $\mathcal{M} \nvDash s = t$. Since $\mathcal{M}$ is a standard model, we know that $[\![s]\!]_\mathcal{M} \neq [\![t]\!]_\mathcal{M}$. By Thm. 6.1, we have $[\![s]\!]_\mathcal{M} \sqsubseteq^\sigma [\![s]\!]_{\mathcal{M}'}$ and $[\![t]\!]_\mathcal{M} \sqsubseteq^\sigma [\![t]\!]_{\mathcal{M}'}$. By the left-uniqueness of $\sqsubseteq^\sigma$, the extensions cannot be equal, and thus $\mathcal{M}' \nvDash s = t$.

ALL$^-$: Assume that $\Gamma[x \mapsto \sigma] \vdash \mathbf{M}^-(t)$ and $\mathcal{M} \nvDash \forall x^\sigma.\, t$. Then there exists $a \in [\![\sigma]\!]_S$ such that $(S, A[x \mapsto a]) \nvDash t$. Since $\sqsubseteq^\sigma$ is left-total, there exists an extension $b \in [\![\sigma]\!]_{S'}$ with $a \sqsubseteq^\sigma b$. Since $(S, A[x \mapsto a]) \sqsubseteq_\Gamma (S', A'[x \mapsto b])$, we can conclude $(S', A'[x \mapsto b]) \nvDash t$ by induction hypothesis. Thus $\mathcal{M}' \nvDash \forall x^\sigma.\, t$.

ALL$^+$: Assume that $\Gamma[x \mapsto \sigma] \vdash \mathbf{M}^+(t)$, $\tilde{\alpha} \notin \mathbf{TV}^+(\sigma)$, and $\mathcal{M} \vDash \forall x^\sigma.\, t$. We show that $\mathcal{M}' \vDash \forall x^\sigma.\, t$. Let $b \in [\![\sigma]\!]_{S'}$. Since $\sqsubseteq^\sigma$ is right-total (Lem. 6.2), there exists a restriction $a \in [\![\sigma]\!]_S$ with $a \sqsubseteq^\sigma b$. By assumption, $(S, A[x \mapsto a]) \vDash t$. Since $(S, A[x \mapsto a]) \sqsubseteq_\Gamma (S', A'[x \mapsto b])$, we can conclude $(S', A'[x \mapsto b]) \vDash t$ by induction hypothesis. □

Corollary 6.1 (Soundness of $\mathfrak{M}_{\mathsf{FS}}$). *If $\Gamma \vdash \mathbf{M}^+(t)$ can be derived in some arbitrary context Γ, then t is monotonic. If $\Gamma \vdash \mathbf{M}^-(t)$ can be derived in some arbitrary context Γ, then t is antimonotonic.*

Example 6.1. Let $\{\alpha\}$ stand for $\alpha \to_{\mathsf{S}} o$, and let $\Gamma_{\mathrm{v}} = [x \mapsto \{\alpha\}, y \mapsto \{\alpha\}]$. The following derivation shows that $x^{\alpha \to o} = y$ is monotonic w.r.t. α:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma(=) = \{\alpha\} \to_{\mathsf{F}} \{\alpha\} \to_{\mathsf{F}} o}{\Gamma \vdash (=) : \{\alpha\} \to_{\mathsf{F}} \{\alpha\} \to_{\mathsf{F}} o} \quad \dfrac{\Gamma(x) = \{\alpha\}}{\Gamma \vdash x : \{\alpha\}} \quad \dfrac{}{\{\alpha\} \leq \{\alpha\}}}{\Gamma \vdash (=)\, x : \{\alpha\} \to_{\mathsf{F}} o} \quad \dfrac{\Gamma(y) = \{\alpha\}}{\Gamma \vdash y : \{\alpha\}} \quad \dfrac{}{\{\alpha\} \leq \{\alpha\}}}{\Gamma \vdash x = y : o}}{\Gamma \vdash \mathbf{M}^+(x = y)}$$

■

Example 6.2. The following table lists some example formulas, including those from Sect. 1. For each formula, we indicate whether it is monotonic or antimononic w.r.t. α according to the calculi $\mathfrak{M}_{\mathsf{F}}$ and $\mathfrak{M}_{\mathsf{FS}}$ and to the semantic definitions.

	MONOTONIC			ANTIMONOTONIC		
FORMULA	$\mathfrak{M}_{\mathsf{F}}$	$\mathfrak{M}_{\mathsf{FS}}$	SEM.	$\mathfrak{M}_{\mathsf{F}}$	$\mathfrak{M}_{\mathsf{FS}}$	SEM.
$\exists x^\alpha\, y.\, x \neq y$	✓	✓	✓	·	·	·
$f\, x^\alpha = x \wedge f\, y \neq y$	✓	✓	✓	✓	✓	✓
$x^{o \to \alpha} = y$	✓	✓	✓	✓	✓	✓
$s^{\alpha \to o} = t$	·	✓	✓	✓	✓	✓
$\{y^\alpha\} = \{z\}$	·	✓	✓	✓	✓	✓
$(\lambda x^\alpha.\, x = y) = (\lambda x.\, x = z)$	·	·	✓	✓	✓	✓
$(\forall x^\alpha.\, f\, x = x) \wedge f\, y \neq y$	·	·	✓	✓	✓	✓
$\forall x^\alpha\, y.\, x = y$	·	·	·	✓	✓	✓
$\exists x^\alpha\, y.\, x \neq y \wedge \forall z.\, z = x \vee z = y$	·	·	·	·	·	·

■

6.4 Type Inference

Expecting all types to be fully annotated with F and S 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 $\{\mathsf{F},\mathsf{S}\}$. Then we construct a typing derivation, collecting a set of constraints over the annotations. Finally, we look for an instantiation for the annotation variables that satisfies all the constraints.

Definition 6.10 (Annotation Constraints). *An* annotation constraint *over a set of annotation variables V is an expression of the form $\sigma \leq \tau$, $\tilde{\alpha} \notin \mathbf{TV}^{+}(\sigma)$, or $\tilde{\alpha} \notin \mathbf{TV}^{-}(\sigma)$, where the types σ and τ may contain annotation variables in V. Given a valuation $\rho : V \to \{\mathsf{F},\mathsf{S}\}$, the meaning of a constraint is defined as in Defs. 6.3 and 6.8.*

A straightforward way of solving such constraints is to encode them in propositional logic, following Defs. 6.3 and 6.8, and give them to a SAT solver. Annotation variables, which may take two values, are mapped directly to propositional variables. This approach proved very efficient on the problems that we encountered in our experiments.

So far, we have 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.

7 Inductive Datatypes

To make monotonicity checking useful in practice, we must support user-defined types, which we have ignored so far. The most important way of introducing new types in Isabelle/HOL is to declare an inductive datatype using the command

$$\textbf{datatype}\ \bar{\alpha}\ \kappa\ =\ C_1\ \sigma_{11}\ \ldots\ \sigma_{1k_1}\ \mid\ \cdots\ \mid\ C_n\ \sigma_{n1}\ \ldots\ \sigma_{nk_n}$$

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 $\sigma_{i1} \to_{\mathsf{F}} \cdots \to_{\mathsf{F}} \sigma_{ik_i} \to_{\mathsf{F}} \bar{\alpha}\ \kappa$. 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 are annotated with F or S. (In the implementation, annotation variables are used to infer the annotations.) The interpretation $[\![\bar{\alpha}\ \kappa]\!]_S$ of the new type is given by the corresponding free term algebra.

We must now extend the basic definitions of $\sqsubseteq$, $\leq$, and $\mathbf{TV}^s$ to this new construct. For Def. 6.2, we add the following case:

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

Similarly, Def. 6.3 is extended with

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

and Def. 6.8 with

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

To extend our soundness result, we must show that Lems. 6.1 to 6.4 still hold. The proofs are straightforward and omitted from this paper. Constancy of the datatype constructors also follows directly from the above definitions.

8 Evaluation

What proportion of monotonic formulas are detected as such by our calculi? We applied Nitpick's implementations of $\mathfrak{M}_{\mathsf{F}}$ and (a large fragment of) $\mathfrak{M}_{\mathsf{FS}}$ 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.

	FORMULAS		SUCCESS RATE	
THEORY	$\mathfrak{M}_{\mathsf{F}}$	$\mathfrak{M}_{\mathsf{FS}}$	$\mathfrak{M}_{\mathsf{F}}$	$\mathfrak{M}_{\mathsf{FS}}$
AVL2	$18_{/24}$	$22_{/24}$	75.0%	91.7%
Fun	$49_{/87}$	$71_{/87}$	56.3%	81.6%
Huffman	$41_{/94}$	$86_{/94}$	43.6%	91.5%
List	$266_{/524}$	$402_{/524}$	50.8%	76.7%
Map	$94_{/97}$	$97_{/97}$	96.9%	100.0%
Relation	$59_{/144}$	$94_{/144}$	41.0%	65.3%

The table indicates how many formulas were found to involve at least one monotonic type variable using $\mathfrak{M}_{\mathsf{F}}$ and $\mathfrak{M}_{\mathsf{FS}}$, respectively, 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 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) is negligible.

9 Discussion

Fully covariant arrow. In Def. 6.3, both the positive and the negative type variables in σ count as positive occurrences in $\sigma \to_{\mathsf{S}} \tau$. This raises the question of whether a fully covariant behavior, with $\mathbf{TV}^s(\sigma \to_{\mathsf{S}} \tau) = \mathbf{TV}^s(\sigma) \cup \mathbf{TV}^s(\tau)$, can also be achieved, possibly with a different definition of $\sqsubseteq^{\sigma \to_{\mathsf{S}} \tau}$. Although such a behavior looks more regular, it would make the calculus unsound, as the following counterexample shows.

Consider the formula $t \equiv \forall F^{(\alpha \to o) \to o}\, f^{\alpha \to o}\, g\, h.\, f \in F \wedge g \in F \wedge f\, a \neq g\, a \longrightarrow h \in F$. The formula is not monotonic w.r.t. α: Regardless of the value of the free variable a, it is true for $|\alpha| = 1$, since the assumptions imply that $f \neq g$, and as there are only two functions of type $\alpha \to o$, 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 S-arrow, we could type F as $F^{(\alpha \to_{\mathsf{F}} o) \to_{\mathsf{S}} o}$ and the rule ALL$^+$ would apply, since there is no positive occurrence of α in the types of F, f, g, and h.

Principal types. Similar 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* has the form $\forall \overline{X}.\, \forall \overline{\alpha}.\, \sigma\, \langle C \rangle$, where σ is an annotated type containing annotation variables $\overline{X}$ and type variables $\overline{\alpha}$, and C is a list of constraints of the form given in Def. 6.10. As an example, equality has the principal type schema $\forall \alpha.\, \alpha \to_{\mathsf{F}} \alpha \to_{X} o\ \langle \tilde{\alpha} \notin \mathbf{TV}^-(\alpha) \rangle$. This approach nicely extends ML-style polymorphism.

Set comprehensions. An obvious weakness of our type system is that the rule LAM always types λ-abstractions with F-arrows. The only way to construct terms whose type contains S annotations is by building them from a set of primitives whose types are justified semantically. This solution is far from optimal. To take just one example, consider the term $\lambda x\, z.\, \exists y.\, R\, x\, y \wedge S\, y\, z$, which composes two binary relations R and S. Semantically, composition is constant for type $(\alpha \to_{\mathsf{S}} \beta \to_{\mathsf{S}} o) \to_{\mathsf{F}} (\beta \to_{\mathsf{S}} \gamma \to_{\mathsf{S}} o) \to_{\mathsf{F}} \alpha \to_{\mathsf{S}} \gamma \to_{\mathsf{S}} o$, but our analysis cannot infer this. As a consequence, our analysis cannot infer the monotonicity of any of the four type variables occurring in the associativity law for composition, unless composition is added to the constant context Γ_{c}.

10 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. 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. A direction for future research would be to extend the type system to handle more syntactic idioms (e.g., set comprehensions and "almost full" sets such as $\mathcal{U} - \{x\}$), thereby strengthening the analysis.

Our measurements show that monotonic formulas are pervasive in HOL formalizations and that syntactic criteria can detect them more often than not. Our more powerful calculus $\mathfrak{M}_{\mathsf{FS}}$ 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, Tobias Nipkow, Mark Summerfield, and the anonymous reviewers for suggesting several textual improvements.

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 T. Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In M. Kaufmann and L. Paulson, editors, *ITP-10*, LNCS. Springer, 2010. To appear.
4. K. Claessen. Private communication, 2009.
5. K. Claessen and N. Sörensson. New techniques that improve MACE-style model finding. In *MODEL*, 2003.
6. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.
7. J. Harrison. HOL Light: A tutorial introduction. In *FMCAD '96*, volume 1166 of *LNCS*, pages 265–269. Springer, 1996.
8. D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.
9. D. Jackson, I. Shlyakhter, and M. Sridharan. A micromodularity mechanism. In *FSE/ESEC 2001*, pages 62–73, 2001.
10. V. Kuncak and D. Jackson. Relational analysis of algebraic datatypes. In H. C. Gall, editor, *ESEC/FSE 2005*, 2005.
11. W. McCune. A Davis–Putnam program and its application to finite first-order model search: Quasigroup existence problems. Technical report, ANL, 1994.
12. L. Momtahan. Towards a small model theorem for data independent systems in Alloy. *ENTCS*, 128(6):37–52, 2005.
13. T. Nipkow. Verifying a hotel key card system. In K. Barkaoui, A. Cavalcanti, and A. Cerone, editors, *ICTAC 2006*, volume 4281 of *LNCS*. Springer, 2006.
14. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
15. 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.
16. K. Slind and M. Norrish. A brief overview of HOL4. In O. A. Mohamed, C. M. noz, and S. Tahar, editors, *TPHOLs 2008*, volume 5170 of *LNCS*, pages 28–32, 2008.
17. 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.
18. T. Weber. *SAT-Based Finite Model Generation for Higher-Order Logic*. Ph.D. thesis, Dept. of Informatics, T.U. München, 2008.
19. J. Zhang and H. Zhang. SEM: A system for enumerating models. In M. Kaufmann, editor, *IJCAI 95*, volume 1, pages 298–303, 1995.