Freefinement

Stephan van Staden
ETH Zurich
Stephan.vanStaden@inf.ethz.ch

Cristiano Calcagno
ETH Zurich, Imperial College London and Monoidics Ltd
c.calcagno@imperial.ac.uk

Bertrand Meyer
ETH Zurich
Bertrand.Meyer@inf.ethz.ch

Abstract

Freefinement is an algorithm that constructs a sound refinement calculus from a verification system under certain conditions. In this paper, a verification system is any formal system for establishing whether an inductively defined term, typically a program, satisfies a specification. Examples of verification systems include Hoare logics and type systems. Freefinement first extends the term language to include specification terms, and builds a verification system for the extended language that is a sound and conservative extension of the original system. The extended system is then transformed into a sound refinement calculus. The resulting refinement calculus can interoperate closely with the verification system – it is even possible to reuse and translate proofs between them. Freefinement gives a semantics to refinement at an abstract level: it associates each term of the extended language with a set of terms from the original language, and refinement simply reduces this set. The paper applies freefinement to a simple type system for the lambda calculus and also to a Hoare logic.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification; D.3.1 [*Programming Languages*]: Formal Definitions and Theory; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs

General Terms Languages, Theory, Verification

Keywords Formal Systems, Proof Theory, Refinement

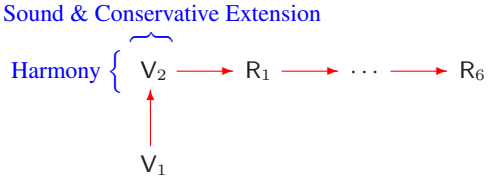
1. Introduction

Many theories in computer science are presented, or approximated, by compositional *verification systems*. In this paper, a verification system is any formal system for establishing whether an inductively defined term, typically a program, satisfies a specification. For example, Hoare logics and type systems can be viewed as verification systems. In the case of Hoare logics, the system proves that a statement satisfies certain specifications given as preconditions and postconditions. In the case of type systems, the system proves that a term has a certain type in a type context.

Refinement systems play a similar role to verification systems, the main difference being that they relate terms to other terms, instead of terms and specifications. Another difference is that they typically include so-called specification terms. Intuitively, a term refines another if it is 'better', i.e. if it satisfies more specifications. *Refinement calculi* are formal systems for establishing refinements. For example, the calculus of Morgan [9] derives refinements between statements based on total correctness specifications. Starting from an appropriate specification statement, one can derive a correct algorithm for computing the factorial of a number by applying Morgan's refinement rules.

This paper originates from the observation that a Hoare logic and a refinement calculus for a command language do not have to be independent entities: once the Hoare logic is extended with specification statements, the two systems can be accommodated in a single theory. Moreover, there is a strong relation between the two systems. The paper explains that this relation is not a coincidence: it is possible to analyze the structure of the inference rules defining a verification system, and automatically generate a related refinement calculus. Freefinement is an algorithm that implements this transformation. Surprisingly, freefinement is not limited to Hoare logics, but can be applied to any verification system whose inference rules satisfy certain conditions. Several refinement rules proposed in the literature in different contexts arise in this way.

The freefinement algorithm works as follows. Given a term language and an accompanying verification system V_1 that satisfies certain conditions, freefinement extends the term language with specification terms and builds a verification system V_2 for extended terms. The conditions on V_1 ensure that it is possible to extend the terms without breaking the inference rules; V_2 is consequently a sound and conservative extension of V_1. Moreover, freefinement proposes a sound refinement system R that is in harmony with V_2. Harmony means that the two formal systems can interoperate smoothly. It entails, for example, that a term satisfies a specification according to V_2 if and only if it is possible to refine the specification into the term with R. In fact, proof translation between V_2 and R becomes possible because harmony is demonstrated constructively. Freefinement internally constructs the refinement calculus by 'linearizing' V_2 in a series of steps. The conditions on V_1 ensure that successful linearization is possible. According to the presentation below, at most six steps are needed for this 'refinement of refinement systems'. The situation is summarized as follows:



Freefinement requires no human intervention. The conditions it imposes are fulfilled by many program logics and type systems: examples include Hoare logic, separation logic, the simply-typed

POPL'12, January 25–27, 2012, Philadelphia, PA, USA.

lambda calculus and System F. Freefinement defines the semantics of refinement at an abstract level: it associates each term of the extended language with a set of terms from the original language, and refinement simply reduces this set.

With freefinement, tools that are based on verification systems can readily include refinement as a complementary or alternative development style. Freefinement provides correctness by construction for free.

Outline. Section 2 describes the freefinement algorithm, which is applied in Section 3 to a simple type system for the lambda calculus and also to Hoare logic. Section 4 concludes with related work.

2. Freefinement

2.1 The Inputs

Freefinement requires four things as input:

1. A set of constructors $\mathbb{K}$. The constructors give rise to a term language T, where an arbitrary term t of T is defined by the grammar:

   $$t ::= C(t_1, \ldots, t_n)$$

   where $C \in \mathbb{K}$.
2. A set of specifications $\mathbb{S}$.
3. A binary relation $\models_{\mathsf{V}_1} \_ \mathit{Sat} \_$ between terms and specifications. Intuitively, $\models_{\mathsf{V}_1} t \mathit{Sat} S$ denotes that term $t \in T$ satisfies specification $S \in \mathbb{S}$.
4. A formal system $\mathsf{V}_1(\mathbb{K}, \mathbb{S}, \models_{\mathsf{V}_1} \_ \mathit{Sat} \_)$, which consists of a set of inference rules for proving sentences of the form t *Sat* S. Each rule of V_1 must have the form A_1 or B_1:

   $$A_1 \ \frac{t_1 \mathit{Sat} S_1 \quad \ldots \quad t_n \mathit{Sat} S_n}{C(t_1, \ldots, t_n) \mathit{Sat} S} \quad \text{provided Pred(C, } S_1, \ldots, S_n, \text{ S).}$$

   $$B_1 \ \frac{t \mathit{Sat} S_1 \quad \ldots \quad t \mathit{Sat} S_m}{t \mathit{Sat} S} \quad \text{provided Pred}(S_1, \ldots, S_m, \text{ S).}$$

   The t's, S's and C in the rule forms indicate where the rules of V_1 must use metavariables. Thus a rule of form A_1 has only the freedom to choose a concrete n and a definition for its proviso predicate Pred; the proviso predicate implements the side condition of the rule based on the arguments C, S_1, …, S_n and S. A rule of form B_1 is also a pair: a concrete m and a definition of a predicate with arguments S_1, …, S_m and S. Freefinement requires that the rules must be sound with respect to the following semantics:

   Definition 1 (Semantics of the Inference Rules)**.**

   1.1 For rules of the form A_1:
   $\text{Pred(C, } S_1, \ldots, S_n, \text{ S)} \Rightarrow [\forall t_1, \ldots, t_n \in T \cdot \models_{\mathsf{V}_1} t_1 \mathit{Sat} S_1 \wedge \ldots \wedge \models_{\mathsf{V}_1} t_n \mathit{Sat} S_n \Rightarrow \models_{\mathsf{V}_1} C(t_1, \ldots, t_n) \mathit{Sat} S]$

   1.2 For rules of the form B_1:
   $\text{Pred}(S_1, \ldots, S_m, \text{ S}) \Rightarrow [\forall t \in T \cdot \models_{\mathsf{V}_1} t \mathit{Sat} S_1 \wedge \ldots \wedge \models_{\mathsf{V}_1} t \mathit{Sat} S_m \Rightarrow \models_{\mathsf{V}_1} t \mathit{Sat} S]$

The rule forms stipulate that the rules of V_1 must be highly compositional – a requirement that freefinement will exploit. For example, rules cannot inspect or constrain the t's that appear in premises. This will allow freefinement to reuse the rules after specification terms are added to the term language.

Consider the following three rules over $\mathbb{K} = \{0, \text{succ}, \text{pred}\}$ and $\mathbb{S} = \{\mathbb{N}\}$, where n is a metavariable:

$$1 \ \frac{n : \mathbb{N}}{\text{succ(n)} : \mathbb{N}} \qquad 2 \ \frac{\text{succ(n)} : \mathbb{N}}{\text{pred(succ(n))} : \mathbb{N}} \qquad 3 \ \frac{n : \mathbb{N}}{\text{pred(n)} : \mathbb{N}} \quad \text{provided positive(n).}$$

Rule 1 can be written in form A_1 with $n = 1$ by defining the proviso Pred(C, S_1, S) as $C = \text{succ} \wedge S_1 = S = \mathbb{N}$. Rule 2 is unacceptable, because its premise inspects the term and requires it to match succ(n). Rule 3 is also unacceptable, because it constrains the term in its proviso.

It will become clear later that the 'structural' rules of Hoare logic, such as the rule of consequence, are examples of rules of form B_1. Other rules of Hoare logic, such as the assignment axiom and rule for sequential composition, have the form A_1.

Let $\vdash_{\mathsf{V}_1} t \mathit{Sat} S$ denote that t *Sat* S is derivable with V_1. The soundness of the rules with respect to the semantics of Definition 1 implies the soundness of V_1:

Theorem 1 (Soundness of V_1)**.** $\vdash_{\mathsf{V}_1} t \mathit{Sat} S \Rightarrow \models_{\mathsf{V}_1} t \mathit{Sat} S$

Proof. By induction on the derivation of t *Sat* S:

- A rule of the form A_1 was last applied. Assume Pred(C, S_1, …, S_n, S) and the induction hypothesis $\models_{\mathsf{V}_1} t_1 \mathit{Sat} S_1 \wedge \ldots \wedge \models_{\mathsf{V}_1} t_n \mathit{Sat} S_n$. Then $\models_{\mathsf{V}_1} C(t_1, \ldots, t_n) \mathit{Sat} S$ by Definition 1.1.
- A rule of the form B_1 was last applied. Assume Pred(S_1, …, S_m, S) and also the induction hypothesis $\models_{\mathsf{V}_1} t \mathit{Sat} S_1 \wedge \ldots \wedge \models_{\mathsf{V}_1} t \mathit{Sat} S_m$. From Definition 1.2 follows $\models_{\mathsf{V}_1} t \mathit{Sat} S$. □

Freefinement does not assume the completeness of V_1, i.e. it never assumes $\models_{\mathsf{V}_1} t \mathit{Sat} S \Rightarrow \vdash_{\mathsf{V}_1} t \mathit{Sat} S$.

2.2 The Extended Language and Formal System

This section extends the language T with specification terms that are useful for refinement. It gives a semantics to the resulting language U, and extends V_1 in a sound and conservative way to prove sentences of the form u *Sat* S where $u \in U$.

2.2.1 The Extended Language U

Suppose $\mathbb{K}$ and $\mathbb{S}$ are disjoint (if they are not, then they can always be decorated to become disjoint) and do not contain a symbol $\bigsqcup$. The extended set of constructors

$$\mathbb{K}' = \mathbb{K} \cup \mathbb{S} \cup \{\bigsqcup \text{ with arity } n \mid n \in \mathbb{N}\}$$

gives rise to an extended language U, which can also be written as:

$$u ::= C(u_1, \ldots, u_n) \mid S \mid \bigsqcup(u_1, \ldots, u_n)$$

A term of the form S is called a *spec* term, and a term of the form $\bigsqcup(u_1, \ldots, u_n)$ is called the *join* of u_1, …, u_n. Intuitively, S is a generic term that satisfies S, and $\bigsqcup(u_1, \ldots, u_n)$ is a generic term that satisfies any S that any of the u_1, …, u_n satisfy. Although the details will become clear later, the reasons for adding these terms are simple: the refinement system should be able to refine spec terms into other terms for top-down development, and join terms will be important for simplifying rules of the form B_1 where $m > 1$. If there are no rules of the form B_1 where $m > 1$, then join terms and their consequent treatment can be omitted.

A couple of constructs are used for giving a semantics to U. Let X denote a subset of T, and let Y denote a subset of $\mathbb{S}$. Specs(X) is the set of all specifications that all the terms in X satisfy, and Terms(Y) is the set of terms of T that satisfy all the specifications in Y:

Definition 2 (Specs and Terms)**.**

- $\text{Specs(X)} \stackrel{\text{def}}{=} \{S \mid \forall t \in X \cdot \models_{\mathsf{V}_1} t \mathit{Sat} S\}$
- $\text{Terms(Y)} \stackrel{\text{def}}{=} \{t \mid \forall S \in Y \cdot \models_{\mathsf{V}_1} t \mathit{Sat} S\}$

An antitone Galois connection[1] exists between Specs and Terms:

Lemma 1. $X \subseteq \text{Terms}(Y) \Leftrightarrow Y \subseteq \text{Specs}(X)$

Proof.
$$\begin{aligned} & X \subseteq \text{Terms}(Y) \\ \Leftrightarrow\ & \{\text{definition of Terms and } \subseteq\} \\ & \forall t \in X \cdot \forall S \in Y \cdot \models_{V_1} t\ \mathit{Sat}\ S \\ \Leftrightarrow\ & \{\text{predicate calculus}\} \\ & \forall S \in Y \cdot \forall t \in X \cdot \models_{V_1} t\ \mathit{Sat}\ S \\ \Leftrightarrow\ & \{\text{definition of Specs and } \subseteq\} \\ & Y \subseteq \text{Specs}(X) \end{aligned}$$ □

Antitone Galois connections have several well-known properties. For instance, (Terms ∘ Specs) and (Specs ∘ Terms) are extensive, increasing and idempotent and therefore closure operators. Freefinement relies on the following properties (their proofs appear in the Appendix):

Corollary 1.

1.1 $X \subseteq \text{Terms}(\text{Specs}(X))$
1.2 $\text{Terms}(\text{Specs}(\text{Terms}(Y))) = \text{Terms}(Y)$
1.3 $\text{Specs}(X) \subseteq \text{Specs}(X') \Leftrightarrow \text{Terms}(\text{Specs}(X)) \supseteq \text{Terms}(\text{Specs}(X'))$
1.4 $\text{Terms}(Y \cup Y') = \text{Terms}(Y) \cap \text{Terms}(Y')$

The following auxiliary definition provides a shorthand for the set of all terms of the form $C(t_1, \ldots, t_n)$ where $t_1 \in X_1, \ldots, t_n \in X_n$:

$$C(X_1, \ldots, X_n) \stackrel{\text{def}}{=} \{C(t_1, \ldots, t_n) \mid \textstyle\bigwedge_{i \in 1..n} t_i \in X_i\}$$

For example, it yields a singleton set for nullary constructors:

$$\{C() \mid \textstyle\bigwedge_{i \in 1..0} t_i \in X_i\} = \{C() \mid \mathit{True}\} = \{C()\}$$

The semantics of U is given by the function $[\![\_]\!]$ of type $U \to \mathcal{P}(T)$, i.e. every term in U denotes a set of terms from T:

Definition 3 (Semantics of U)**.**

$$\begin{aligned} [\![C(u_1, \ldots, u_n)]\!] &\stackrel{\text{def}}{=} \text{Terms}(\text{Specs}(C([\![u_1]\!], \ldots, [\![u_n]\!]))) \\ [\![S]\!] &\stackrel{\text{def}}{=} \text{Terms}(\{S\}) \\ [\![\textstyle\bigsqcup(u_1, \ldots, u_n)]\!] &\stackrel{\text{def}}{=} \textstyle\bigcap_{i \in 1..n} [\![u_i]\!] \end{aligned}$$

If the relation $\models_{V_1} \_\ \mathit{Sat}\ \_$ is well-behaved in a sense that will be made precise later, then $[\![u]\!]$ has a simple intuitive explanation: it denotes the set of all primitive terms, i.e. terms from T, that refine u. For a term $C(u_1, \ldots, u_n)$, first consider $C([\![u_1]\!], \ldots, [\![u_n]\!])$ – the set of terms of the form $C(t_1, \ldots, t_n)$ where $t_1 \in [\![u_1]\!]$ (i.e. t_1 refines u_1) and … and $t_n \in [\![u_n]\!]$. All the specifications that all these terms implement are then collected, and any primitive term that satisfies all such specifications refines $C(u_1, \ldots, u_n)$. The primitive terms that refine S are exactly those that satisfy S. Finally, $\bigsqcup(u_1, \ldots, u_n)$ is refined by any primitive term that refines all $u_1, \ldots, u_n$.

For all u, the set $[\![u]\!]$ is a fixpoint of Terms ∘ Specs and hence a closed element:

Lemma 2. $\text{Terms}(\text{Specs}([\![u]\!])) = [\![u]\!]$

Proof. By induction on the structure of u:

- If u has the form $C(u_1, \ldots, u_n)$ or S, then $[\![u]\!] = \text{Terms}(Y)$ for some Y and the result follows by Corollary 1.2.
- If u has the form $\bigsqcup(u_1, \ldots, u_n)$, assume $[\![u_i]\!] = \text{Terms}(\text{Specs}([\![u_i]\!]))$ for all $i \in 1..n$. So $[\![\bigsqcup(u_1, \ldots, u_n)]\!] = \bigcap_{i \in 1..n} \text{Terms}(\text{Specs}([\![u_i]\!]))$ $= \text{Terms}(\bigcup_{i \in 1..n} \text{Specs}([\![u_i]\!]))$ by Corollary 1.4, and Corollary 1.2 concludes the proof. □

[1] Also known as an *order-reversing* or *contravariant* Galois connection.

The rest of the paper introduces further properties of the semantics as needed.

2.2.2 Extending V_1: Preliminaries

The next section will extend V_1 to obtain a formal system V_2 for proving sentences of the form u *Sat* S. The aim is to construct a sound and conservative extension of V_1. Informally, a sound extension of V_1 must have equal or more power:

Definition 4 (Sound Extension)**.** $V_2(\mathbb{K}', \mathbb{S}', \models_{V_2} \_\ \mathit{Sat}\ \_)$ is a sound extension of $V_1(\mathbb{K}, \mathbb{S}, \models_{V_1} \_\ \mathit{Sat}\ \_)$ if and only if

1. V_2 uses richer terms and specifications:
   $\mathbb{K} \subseteq \mathbb{K}'$ and $\mathbb{S} \subseteq \mathbb{S}'$
2. V_2 can prove everything that V_1 can prove:
   $\forall t \in T, S \in \mathbb{S} \cdot \vdash_{V_1} t\ \mathit{Sat}\ S \Rightarrow \vdash_{V_2} t\ \mathit{Sat}\ S$
3. V_2 uses a richer semantics:
   $\forall t \in T, S \in \mathbb{S} \cdot \models_{V_2} t\ \mathit{Sat}\ S \Rightarrow \models_{V_1} t\ \mathit{Sat}\ S$
4. V_2 is sound:
   $\forall u \in U, S' \in \mathbb{S}' \cdot \vdash_{V_2} u\ \mathit{Sat}\ S' \Rightarrow \models_{V_2} u\ \mathit{Sat}\ S'$

As a consequence, $\forall t \in T, S \in \mathbb{S} \cdot \vdash_{V_2} t\ \mathit{Sat}\ S \Rightarrow \models_{V_1} t\ \mathit{Sat}\ S$, which intuitively means that V_2 restricted to $\mathbb{K}$ and $\mathbb{S}$ is sound with respect to the semantics of V_1.

In a sound and conservative extension, the converse of requirement 2 also holds:

Definition 5 (Sound and Conservative Extension)**.** A formal system $V_2(\mathbb{K}', \mathbb{S}', \models_{V_2} \_\ \mathit{Sat}\ \_)$ is a sound and conservative extension of $V_1(\mathbb{K}, \mathbb{S}, \models_{V_1} \_\ \mathit{Sat}\ \_)$ if and only if

1. V_2 is a sound extension of V_1.
2. V_1 and V_2 restricted to $\mathbb{K}$ and $\mathbb{S}$ have equal derivability:
   $\forall t \in T, S \in \mathbb{S} \cdot \vdash_{V_1} t\ \mathit{Sat}\ S \Leftrightarrow \vdash_{V_2} t\ \mathit{Sat}\ S$

Although a sound and conservative extension cannot prove more sentences of the form t *Sat* S, it is still useful for extending the term language and installing a richer semantics. It can also extend the specifications, but the V_2 of the next section will simply use $\mathbb{S}$.

2.2.3 The Extended Formal System V_2

The construction of V_2 starts with the empty set of rules and proceeds in two steps:

1. For each rule of V_1, replace t's by u's and add the resulting rule. This change of metavariables yields the rule forms A_2 and B_2 in V_2:

$$A_2\ \frac{u_1\ \mathit{Sat}\ S_1 \quad \ldots \quad u_n\ \mathit{Sat}\ S_n}{C(u_1, \ldots, u_n)\ \mathit{Sat}\ S}$$
provided $\text{Pred}(C, S_1, \ldots, S_n, S)$.

$$B_2\ \frac{u\ \mathit{Sat}\ S_1 \quad \ldots \quad u\ \mathit{Sat}\ S_m}{u\ \mathit{Sat}\ S}$$
provided $\text{Pred}(S_1, \ldots, S_m, S)$.

2. Add the following rules for spec terms and joins:

$$\textsc{Spec}\ \frac{}{S\ \mathit{Sat}\ S} \qquad \textsc{Join}\ \frac{u\ \mathit{Sat}\ S}{\textstyle\bigsqcup(\ldots, u, \ldots)\ \mathit{Sat}\ S}$$

By induction on the derivation, V_1 and V_2 are equivalent with respect to derivability on T, i.e. $\vdash_{V_1} t\ \mathit{Sat}\ S \Leftrightarrow \vdash_{V_2} t\ \mathit{Sat}\ S$. So for V_2 to be a sound and conservative extension of V_1, it will suffice to equip V_2 with a richer semantics and to prove it sound.

The *Sat* relation between U and $\mathbb{S}$ is defined as follows:

Definition 6 (Extended Satisfaction)**.**
$\models_{V_2} u \; \mathit{Sat}\; S \stackrel{\text{def}}{=} \forall t \in [\![u]\!] \cdot \models_{V_1} t \; \mathit{Sat}\; S$

Furthermore, the U-semantics of t contains t as an element:

Lemma 3 (Term Embedding)**.** $\forall t \in T \cdot t \in [\![t]\!]$

Proof. By induction on the structure of t. Suppose $t = C(t_1, \ldots, t_n)$ and assume $t_1 \in [\![t_1]\!], \ldots, t_n \in [\![t_n]\!]$. So $t \in C([\![t_1]\!], \ldots, [\![t_n]\!])$, which is a subset of $\mathrm{Terms}(\mathrm{Specs}(C([\![t_1]\!], \ldots, [\![t_n]\!])))$ by Corollary 1.1. □

Therefore $\models_{V_2} t \; \mathit{Sat}\; S \Rightarrow \models_{V_1} t \; \mathit{Sat}\; S$ holds, and the soundness proof of V_2 establishes that V_2 is a sound and conservative extension of V_1:

Theorem 2 (Soundness of V_2)**.** $\vdash_{V_2} u \; \mathit{Sat}\; S \Rightarrow \models_{V_2} u \; \mathit{Sat}\; S$

Proof. By induction on the structure of the derivation:

- For each rule of the form A_2, assume $\mathrm{Pred}(C, S_1, \ldots, S_n, S)$ and assume
$$\begin{array}{c} \forall t_1 \in [\![u_1]\!] \cdot \models_{V_1} t_1 \; \mathit{Sat}\; S_1 \\ \vdots \\ \forall t_n \in [\![u_n]\!] \cdot \models_{V_1} t_n \; \mathit{Sat}\; S_n \end{array}$$
So $\forall t_1 \in [\![u_1]\!], \ldots, t_n \in [\![u_n]\!] \cdot \models_{V_1} C(t_1, \ldots, t_n) \; \mathit{Sat}\; S$ because the corresponding rule of the form A_1 in V_1 is sound with respect to Definition 1.1. So $S \in \mathrm{Specs}(C([\![u_1]\!], \ldots, [\![u_n]\!]))$ and hence $\forall t \in \mathrm{Terms}(\mathrm{Specs}(C([\![u_1]\!], \ldots, [\![u_n]\!]))) \cdot \models_{V_1} t \; \mathit{Sat}\; S$.
- For each rule of the form B_2, assume $\mathrm{Pred}(S_1, \ldots, S_m, S)$ and assume $\forall t \in [\![u]\!] \cdot \models_{V_1} t \; \mathit{Sat}\; S_1 \;\wedge\; \ldots \;\wedge\; \models_{V_1} t \; \mathit{Sat}\; S_m$. Now $\forall t \in [\![u]\!] \cdot \models_{V_1} t \; \mathit{Sat}\; S$ because the corresponding rule of the form B_1 in V_1 is sound with respect to Definition 1.2.
- SPEC: $\forall t \in \mathrm{Terms}(\{S\}) \cdot \models_{V_1} t \; \mathit{Sat}\; S$ by definition.
- JOIN: Assume $\forall t \in [\![u]\!] \cdot \models_{V_1} t \; \mathit{Sat}\; S$. If $t \in [\![\bigsqcup(\ldots, u, \ldots)]\!]$ then $t \in [\![u]\!]$ and hence $\models_{V_1} t \; \mathit{Sat}\; S$. □

Extended satisfaction has an alternative characterization that freefinement will also use:

Lemma 4. $\models_{V_2} u \; \mathit{Sat}\; S \;\Leftrightarrow\; S \in \mathrm{Specs}([\![u]\!])$

Proof.
$$\begin{array}{ll} & \models_{V_2} u \; \mathit{Sat}\; S \\ \Leftrightarrow & \{\text{definition}\} \\ & \forall t \in [\![u]\!] \cdot \models_{V_1} t \; \mathit{Sat}\; S \\ \Leftrightarrow & \{\text{definition of Specs}\} \\ & S \in \mathrm{Specs}([\![u]\!]) \end{array}$$
□

2.3 System V_2 and Refinement

The next section will construct several refinement systems, or calculi, that are based on V_2. These refinement systems are formal systems for proving sentences of the form $u \sqsubseteq u'$. The definition of the refinement relation makes the semantics of refinement precise:

Definition 7 (Refinement)**.** $\models u \sqsubseteq u' \stackrel{\text{def}}{=} [\![u]\!] \supseteq [\![u']\!]$

This definition leads to simple proofs, and is equivalent to several other formulations. The following theorem states one such alternative, and its proof mentions others:

Lemma 5 (Equivalent Characterization of Refinement)**.**
$\models u \sqsubseteq u' \;\Leftrightarrow\; \forall S \cdot \models_{V_2} u \; \mathit{Sat}\; S \Rightarrow \models_{V_2} u' \; \mathit{Sat}\; S$

Proof.
$$\begin{array}{ll} & [\![u]\!] \supseteq [\![u']\!] \\ \Leftrightarrow & \{\text{Lemma 2}\} \\ & \mathrm{Terms}(\mathrm{Specs}([\![u]\!])) \supseteq \mathrm{Terms}(\mathrm{Specs}([\![u']\!])) \\ \Leftrightarrow & \{\text{Corollary 1.3}\} \\ & \mathrm{Specs}([\![u]\!]) \subseteq \mathrm{Specs}([\![u']\!]) \\ \Leftrightarrow & \{\text{definition of } \subseteq\} \\ & \forall S \cdot S \in \mathrm{Specs}([\![u]\!]) \Rightarrow S \in \mathrm{Specs}([\![u']\!]) \\ \Leftrightarrow & \{\text{Lemma 4}\} \\ & \forall S \cdot \models_{V_2} u \; \mathit{Sat}\; S \;\Rightarrow\; \models_{V_2} u' \; \mathit{Sat}\; S \end{array}$$
□

If $\models_{V_1} \_ \; \mathit{Sat} \; \_$ is well-behaved, then there is also another explanation for defining $\models u \sqsubseteq u'$ as $[\![u]\!] \supseteq [\![u']\!]$: u' refines u iff every primitive term that refines u' also refines u. Put differently, u' refines u iff u' constrains the set of eventual primitive terms that refinement can produce to the same or higher degree compared to u. So u can be seen as a placeholder for any of the primitive terms in $[\![u]\!]$, and the role of refinement is to reduce the uncertainty.

Many examples of refinements will follow later, so here is a small one: a join term implements the least upper bound (join) of its immediate subterms with respect to $\sqsubseteq$, hence the name. In particular:

1. $\forall i \in 1..n \cdot \models u_i \sqsubseteq \bigsqcup(u_1, \ldots, u_n)$
2. If $(\forall i \in 1..n \cdot \models u_i \sqsubseteq u)$, then $\models \bigsqcup(u_1, \ldots, u_n) \sqsubseteq u$.

The notation $u \equiv u'$ is a shorthand for $[\![u]\!] = [\![u']\!]$, which is equivalent to $\models u \sqsubseteq u' \wedge \models u' \sqsubseteq u$.

A refinement system R will be sound if and only if $\vdash_R u \sqsubseteq u'$ implies $\models u \sqsubseteq u'$. In the next section, freefinement will construct several sound refinement systems where each system R is related to V_2 by the properties Harmony 1 and 2 below.

Harmony 1. If $\vdash_{V_2} u \; \mathit{Sat}\; S$ and $\vdash_R u \sqsubseteq u'$, then $\vdash_{V_2} u' \; \mathit{Sat}\; S$.

Intuitively, Harmony 1 says that V_2 contains sufficient machinery to prove the same properties about u' that it could prove about u. In other words, R is not too powerful for V_2.

Harmony 2. If $\vdash_{V_2} u \; \mathit{Sat}\; S$, then $\vdash_R S \sqsubseteq u$.

Intuitively, Harmony 2 means that the refinement system R contains sufficient machinery to refine a specification into any term that satisfies it according to V_2. In other words, V_2 is embedded in R and hence R is not too weak.

Harmony 1 is stronger than the converse of Harmony 2:

Theorem 3. If V_2 and a refinement system R are related by Harmony 1, then $\vdash_R S \sqsubseteq u \;\Rightarrow\; \vdash_{V_2} u \; \mathit{Sat}\; S$.

Proof. Assume $\vdash_R S \sqsubseteq u$. Since $\vdash_{V_2} S \; \mathit{Sat}\; S$ by SPEC, it follows from Harmony 1 that $\vdash_{V_2} u \; \mathit{Sat}\; S$. □

A refinement system R is called *harmonic* iff it satisfies Harmony 1 and 2. Harmonic refinement systems interoperate nicely with V_2. In fact, the proofs of Harmony 1 and 2 in the next section are constructive in the sense that they enable proof translation. Given a V_2-proof of $u \; \mathit{Sat}\; S$ and an R-proof of $u \sqsubseteq u'$, they describe a V_2-proof of $u' \; \mathit{Sat}\; S$. Based on a V_2-proof of $u \; \mathit{Sat}\; S$, they show how to build an R-proof for $S \sqsubseteq u$. Since Harmony 1 is established constructively, given an R-proof of $S \sqsubseteq u$, the proof of Theorem 3 shows how to build a V_2-proof for $u \; \mathit{Sat}\; S$.

The final refinement system that freefinement produces will also have a specific desired form. This form guarantees that refinement proofs are 'linear' developments where terms can be refined in-place. Formally, a refinement system has the desired form if the rules with premises describe either the transitivity or the monotonicity of refinement. All the other rules must be axioms, i.e. without any premise.

2.4 The Refinement of Refinement Systems

V_2 can be linearized in a series of steps to obtain a sound and harmonic refinement system of the desired form. At most six steps are necessary according to this presentation – the exact number depends on V_1. The steps make it easy to prove and maintain

soundness and harmony, which would otherwise be more complex to establish for the final refinement calculus.

Many of the steps take a previously constructed refinement system and add or remove rules to obtain a new system. If a sound and harmonic refinement system is extended with a rule that is sound and respects Harmony 1, then the resulting system will be sound and harmonic. There is no need to prove Harmony 2 again, because the new refinement system can still derive all sentences that the old one could derive. If a rule is removed from a sound and harmonic refinement system, then the resulting system remains sound and will also be harmonic if it satisfies Harmony 2. A simple way of showing that Harmony 2 still holds is to show that any application of the old rule can be achieved by a combination of rules that remain in the system.

2.4.1 Getting Started: R_1

The first refinement system R_1 is obtained from V_2 by a simple syntactic transformation: each sentence u *Sat* S becomes $\mathrm{S} \sqsubseteq \mathrm{u}$. R_1 has rules of the form A_3 and B_3, a SPEC rule and also a JOIN rule if join terms were needed:

$$\mathrm{A}_3\ \frac{\mathrm{S}_1 \sqsubseteq \mathrm{u}_1 \qquad \dots \qquad \mathrm{S}_n \sqsubseteq \mathrm{u}_n}{\mathrm{S} \sqsubseteq \mathrm{C}(\mathrm{u}_1, \dots, \mathrm{u}_n)}$$
provided Pred(C, $\mathrm{S}_1, \dots, \mathrm{S}_n$, S).

$$\mathrm{B}_3\ \frac{\mathrm{S}_1 \sqsubseteq \mathrm{u} \qquad \dots \qquad \mathrm{S}_m \sqsubseteq \mathrm{u}}{\mathrm{S} \sqsubseteq \mathrm{u}}$$
provided Pred($\mathrm{S}_1, \dots, \mathrm{S}_m$, S).

$$\textsc{Spec}\ \frac{}{\mathrm{S} \sqsubseteq \mathrm{S}}$$

$$\textsc{Join}\ \frac{\mathrm{S} \sqsubseteq \mathrm{u}}{\mathrm{S} \sqsubseteq \bigsqcup(\dots, \mathrm{u}, \dots)}$$

V_2 and R_1 are isomorphic: a proof of u *Sat* S in V_2 corresponds to a proof of $\mathrm{S} \sqsubseteq \mathrm{u}$ in R_1 and vice versa, so $\vdash_{\mathsf{V}_2} \mathrm{u}\ \mathit{Sat}\ \mathrm{S} \Leftrightarrow \vdash_{\mathsf{R}_1} \mathrm{S} \sqsubseteq \mathrm{u}$. The soundness proof of R_1 relies on the following equivalence:

Lemma 6. $\models_{\mathsf{V}_2} \mathrm{u}\ \mathit{Sat}\ \mathrm{S} \Leftrightarrow \models \mathrm{S} \sqsubseteq \mathrm{u}$

Proof.
$\models_{\mathsf{V}_2} \mathrm{u}\ \mathit{Sat}\ \mathrm{S}$
$\Leftrightarrow$ {Lemma 4}
$\{\mathrm{S}\} \subseteq \mathrm{Specs}(\llbracket \mathrm{u} \rrbracket)$
$\Leftrightarrow$ {Lemma 1}
$\llbracket \mathrm{u} \rrbracket \subseteq \mathrm{Terms}(\{\mathrm{S}\})$ □

Theorem 4 (Soundness of R_1)**.** $\vdash_{\mathsf{R}_1} \mathrm{u} \sqsubseteq \mathrm{u}' \Rightarrow \models \mathrm{u} \sqsubseteq \mathrm{u}'$

Proof. If $\vdash_{\mathsf{R}_1} \mathrm{u} \sqsubseteq \mathrm{u}'$, then u has the form S and $\vdash_{\mathsf{V}_2} \mathrm{u}'\ \mathit{Sat}\ \mathrm{S}$. The soundness of V_2 implies $\models_{\mathsf{V}_2} \mathrm{u}'\ \mathit{Sat}\ \mathrm{S}$, and Lemma 6 in turn implies $\models \mathrm{S} \sqsubseteq \mathrm{u}'$. □

Theorem 5. R_1 is harmonic.

Proof. Harmony 2 holds by construction. For Harmony 1, assume $\vdash_{\mathsf{R}_1} \mathrm{u} \sqsubseteq \mathrm{u}'$. Then u has the form S″ and $\vdash_{\mathsf{V}_2} \mathrm{u}'\ \mathit{Sat}\ \mathrm{S}''$ by construction. That $\vdash_{\mathsf{V}_2} \mathrm{u}\ \mathit{Sat}\ \mathrm{S}'$ (i.e. $\vdash_{\mathsf{V}_2} \mathrm{S}''\ \mathit{Sat}\ \mathrm{S}'$) implies $\vdash_{\mathsf{V}_2} \mathrm{u}'\ \mathit{Sat}\ \mathrm{S}'$ for all S′ follows by induction on the derivation of S″ *Sat* S′:

- SPEC: S′ and S″ are the same. Since $\vdash_{\mathsf{V}_2} \mathrm{u}'\ \mathit{Sat}\ \mathrm{S}''$, it holds that $\vdash_{\mathsf{V}_2} \mathrm{u}'\ \mathit{Sat}\ \mathrm{S}'$.
- For each rule of the form B_2: S and S′ are the same. Assume Pred($\mathrm{S}_1, \dots, \mathrm{S}_m$, S), $\vdash_{\mathsf{V}_2} \mathrm{u}\ \mathit{Sat}\ \mathrm{S}_1, \dots, \vdash_{\mathsf{V}_2} \mathrm{u}\ \mathit{Sat}\ \mathrm{S}_m$, and by the induction hypothesis also $\vdash_{\mathsf{V}_2} \mathrm{u}'\ \mathit{Sat}\ \mathrm{S}_1, \dots, \vdash_{\mathsf{V}_2} \mathrm{u}'\ \mathit{Sat}\ \mathrm{S}_m$. So the rule being considered is applicable and $\vdash_{\mathsf{V}_2} \mathrm{u}'\ \mathit{Sat}\ \mathrm{S}$. Hence $\vdash_{\mathsf{V}_2} \mathrm{u}'\ \mathit{Sat}\ \mathrm{S}'$. □

Note: if V_2 has only rules of the form A_2 where $n = 0$ and/or rules of the form B_2 where $m = 0$, then R_1 is a refinement system of the desired form and freefinement stops.

2.4.2 Adding Transitivity: R_2

The refinement system R_2 extends R_1 with the rule TRANS which states that refinement is transitive:

$$\textsc{Trans}\ \frac{\mathrm{u}_1 \sqsubseteq \mathrm{u}_2 \qquad \mathrm{u}_2 \sqsubseteq \mathrm{u}_3}{\mathrm{u}_1 \sqsubseteq \mathrm{u}_3}$$

TRANS is sound because $\sqsupseteq$ is transitive, and it maintains Harmony 1 since implication is transitive. So R_2 is sound and harmonic.

2.4.3 Simplification: R_3

The presence of SPEC and TRANS in R_2 allows the simplification of rules of the form B_3 with $m = 1$:

$$\mathrm{B}_3\ \frac{\mathrm{S}_1 \sqsubseteq \mathrm{u}}{\mathrm{S} \sqsubseteq \mathrm{u}}$$
provided Pred(S_1, S).

For an arbitrary rule of this form, consider the derivation

$$\mathrm{B}_3\ \frac{\textsc{Spec}\ \dfrac{}{\mathrm{S}_1 \sqsubseteq \mathrm{S}_1}}{\mathrm{S} \sqsubseteq \mathrm{S}_1}$$
provided Pred(S_1, S).

By virtue of having been derived, the new rule

$$\mathrm{B}_3\ \frac{}{\mathrm{S} \sqsubseteq \mathrm{S}_1}$$
provided Pred(S_1, S).

is sound and maintains Harmony 1, and can therefore be added to R_2 to obtain a sound and harmonic refinement system. In fact, it can replace the old version without breaking Harmony 2, since removing the old version will not decrease the derivable set of sentences: every application of the old B_3 can be changed into:

$$\textsc{Trans}\ \frac{\mathrm{B}_3\ \dfrac{}{\mathrm{S} \sqsubseteq \mathrm{S}_1} \qquad \mathrm{S}_1 \sqsubseteq \mathrm{u}}{\mathrm{S} \sqsubseteq \mathrm{u}}$$

since Pred(S_1, S) is guaranteed.

The refinement system R_3 is the same as R_2, except that the rules of the form B_3 with $m = 1$ are replaced by their simplified versions. R_3 is sound and harmonic.

Note: if V_2 has only rules of the form A_2 where $n = 0$ and rules of the form B_2 where $m \leq 1$, then R_3 is a refinement system of the desired form and freefinement stops.

2.4.4 Adding Monotonicity: R_4

All the constructors of U are monotone with respect to $\sqsubseteq$, i.e. the following rules are sound:

$$\text{C-}i\ \frac{\mathrm{u}_i \sqsubseteq \mathrm{u}_i'}{\mathrm{C}(\mathrm{u}_1, \dots, \mathrm{u}_i, \dots, \mathrm{u}_n) \sqsubseteq \mathrm{C}(\mathrm{u}_1, \dots, \mathrm{u}_i', \dots, \mathrm{u}_n)}$$

$$\textsc{Join-}i\ \frac{\mathrm{u}_i \sqsubseteq \mathrm{u}_i'}{\bigsqcup(\mathrm{u}_1, \dots, \mathrm{u}_i, \dots, \mathrm{u}_n) \sqsubseteq \bigsqcup(\mathrm{u}_1, \dots, \mathrm{u}_i', \dots, \mathrm{u}_n)}$$

Moreover, these rules maintain harmony:

Lemma 7. C-i maintains Harmony 1.

Proof. Assume $\forall \mathrm{S}' \cdot \vdash_{\mathsf{V}_2} \mathrm{u}_i\ \mathit{Sat}\ \mathrm{S}' \Rightarrow \vdash_{\mathsf{V}_2} \mathrm{u}_i'\ \mathit{Sat}\ \mathrm{S}'$. That $\forall \mathrm{S} \cdot \vdash_{\mathsf{V}_2} \mathrm{C}(\mathrm{u}_1, \dots, \mathrm{u}_i, \dots, \mathrm{u}_n)\ \mathit{Sat}\ \mathrm{S} \Rightarrow \vdash_{\mathsf{V}_2} \mathrm{C}(\mathrm{u}_1, \dots, \mathrm{u}_i', \dots, \mathrm{u}_n)\ \mathit{Sat}\ \mathrm{S}$ follows by induction on the derivation of $\mathrm{C}(\mathrm{u}_1, \dots, \mathrm{u}_i, \dots, \mathrm{u}_n)$ *Sat* S:

- A_2: Suppose $\vdash_{\mathsf{V}_2} \mathrm{u}_j\ \mathit{Sat}\ \mathrm{S}_j$ for $j \in 1..n$, and also suppose Pred(C, $\mathrm{S}_1, \dots, \mathrm{S}_n$, S) holds. Since $\vdash_{\mathsf{V}_2} \mathrm{u}_i'\ \mathit{Sat}\ \mathrm{S}_i$, the same rule A_2 can be applied to derive $\mathrm{C}(\mathrm{u}_1, \dots, \mathrm{u}_i', \dots, \mathrm{u}_n)$ *Sat* S.

- B_2: Suppose $\vdash_{V_2} C(u_1, \ldots, u_i, \ldots, u_n)$ *Sat* S_j for $j \in 1..m$, and suppose $\text{Pred}(S_1, \ldots, S_m, S)$. The induction hypothesis is the assumption $\vdash_{V_2} C(u_1, \ldots, u_i', \ldots, u_n)$ *Sat* S_j for $j \in 1..m$. Since $\text{Pred}(S_1, \ldots, S_m, S)$, the same rule B_2 is applicable and hence $\vdash_{V_2} C(u_1, \ldots, u_i', \ldots, u_n)$ *Sat* S. □

Lemma 8. JOIN-i maintains Harmony 1.

Proof. Assume $\forall S' \cdot \vdash_{V_2} u_i$ *Sat* $S' \Rightarrow \vdash_{V_2} u_i'$ *Sat* S'. That $\forall S \cdot \vdash_{V_2} \bigsqcup(u_1, \ldots, u_i, \ldots, u_n)$ *Sat* $S \Rightarrow \vdash_{V_2} \bigsqcup(u_1, \ldots, u_i', \ldots, u_n)$ *Sat* S follows by induction on the derivation of $\bigsqcup(u_1, \ldots, u_i, \ldots, u_n)$ *Sat* S:

- JOIN: Suppose u_j *Sat* S was the premise for some $j \in 1..n$. If $j \neq i$, then apply JOIN to the premise u_j *Sat* S to derive the required $\bigsqcup(u_1, \ldots, u_i', \ldots, u_n)$ *Sat* S. If $j = i$, then by assumption $\vdash_{V_2} u_i'$ *Sat* S holds, and the result follows by JOIN.
- B_2: Suppose $\vdash_{V_2} \bigsqcup(u_1, \ldots, u_i, \ldots, u_n)$ *Sat* S_j for $j \in 1..m$, and suppose $\text{Pred}(S_1, \ldots, S_m, S)$. The induction hypothesis is the assumption $\vdash_{V_2} \bigsqcup(u_1, \ldots, u_i', \ldots, u_n)$ *Sat* S_j for $j \in 1..m$. Since $\text{Pred}(S_1, \ldots, S_m, S)$, the same rule B_2 is applicable and hence $\vdash_{V_2} \bigsqcup(u_1, \ldots, u_i', \ldots, u_n)$ *Sat* S. □

Let the notation v[u] denote a term in U whose parse tree is factored into two parts: a core tree v with a 'hole' where the subtree for u fits. The rule MONO packages C-i and JOIN-i in a single convenient form:

$$\textsc{Mono}\ \frac{u \sqsubseteq u'}{v[u] \sqsubseteq v[u']}$$

Informally, the rule MONO allows in-place refinement: if u_0 can be factored as v[u], and u′ refines u, then v[u′] refines u_0.

MONO is sound and maintains harmony because C-i and JOIN-i are sound and maintain harmony. The refinement system R_4 extends R_3 with MONO. It is sound and harmonic.

2.4.5 Simplification: R_5

The rule MONO makes it possible to simplify:

- The JOIN rule:

$$\textsc{Join}\ \frac{S \sqsubseteq u}{S \sqsubseteq \bigsqcup(\ldots, u, \ldots)}$$

- Rules of the form A_3 with $n \geq 1$:

$$A_3\ \frac{S_1 \sqsubseteq u_1 \quad \ldots \quad S_n \sqsubseteq u_n}{S \sqsubseteq C(u_1, \ldots, u_n)}$$
provided $\text{Pred}(C, S_1, \ldots, S_n, S)$.

Consider the derivation:

$$\textsc{Join}\ \frac{\textsc{Spec}\ \dfrac{}{S \sqsubseteq S}}{S \sqsubseteq \bigsqcup(\ldots, S, \ldots)}$$

By virtue of having been derived, the simplified rule

$$\textsc{Join}\ \frac{}{S \sqsubseteq \bigsqcup(\ldots, S, \ldots)}$$

is sound and respects Harmony 1. It can replace the old version of JOIN without decreasing derivability, because any application of the old version can be achieved by:

$$\textsc{Trans}\ \frac{\textsc{Join}\ \dfrac{}{S \sqsubseteq \bigsqcup(\ldots, S, \ldots)} \qquad \textsc{Mono}\ \dfrac{S \sqsubseteq u}{\bigsqcup(\ldots, S, \ldots) \sqsubseteq \bigsqcup(\ldots, u, \ldots)}}{S \sqsubseteq \bigsqcup(\ldots, u, \ldots)}$$

Likewise, for each rule of the form A_3, the derived rule

$$A_3\ \frac{}{S \sqsubseteq C(S_1, \ldots, S_n)}$$
provided $\text{Pred}(C, S_1, \ldots, S_n, S)$.

is sound and respects harmony. It makes the old version redundant, since any application of the old rule can be replaced by:

$$A_3\ \frac{}{S \sqsubseteq C(S_1, \ldots, S_n)}\ E_1 \quad \ddots \quad E_n$$

where E_i is given by:

$$\textsc{Trans}\ \frac{S \sqsubseteq C(u_1, \ldots, u_{i-1}, S_i, \ldots, S_n) \qquad P_i}{S \sqsubseteq C(u_1, \ldots, u_i, S_{i+1}, \ldots, S_n)}$$

and P_i is the proof tree:

$$\textsc{Mono}\ \frac{S_i \sqsubseteq u_i}{C(u_1, \ldots, u_{i-1}, S_i, \ldots, S_n) \sqsubseteq C(u_1, \ldots, u_i, S_{i+1}, \ldots, S_n)}$$

Apart from these simplifications, the refinement system R_5 is the same as R_4. It is sound and harmonic.

Note: if V_2 does not include rules of the form B_2 where $m > 1$, then R_5 has the desired form and freefinement stops.

2.4.6 Wrapping Up: R_6

It remains to simplify rules of the form B_3 with $m > 1$:

$$B_3\ \frac{S_1 \sqsubseteq u \quad \ldots \quad S_m \sqsubseteq u}{S \sqsubseteq u}$$
provided $\text{Pred}(S_1, \ldots, S_m, S)$.

If $\text{Pred}(S_1, \ldots, S_m, S)$, then R_5 can derive:

$$B_3\ \frac{\textsc{Join}\ \dfrac{}{S_1 \sqsubseteq \bigsqcup(S_1, \ldots, S_m)} \quad \cdots \quad \textsc{Join}\ \dfrac{}{S_m \sqsubseteq \bigsqcup(S_1, \ldots, S_m)}}{S \sqsubseteq \bigsqcup(S_1, \ldots, S_m)}$$

The derived rule

$$B_3\ \frac{}{S \sqsubseteq \bigsqcup(S_1, \ldots, S_m)}$$
provided $\text{Pred}(S_1, \ldots, S_m, S)$.

is therefore sound and respects Harmony 1. Together with the rule:

$$\textsc{Unjoin}\ \frac{}{\bigsqcup(u, \ldots, u) \sqsubseteq u}$$

which is trivially sound and respects Harmony 1, it can replace the old B_3 because any application of the old rule can be rewritten as:

$$\textsc{Trans}\ \frac{B_3\ \dfrac{}{S \sqsubseteq \bigsqcup(S_1, \ldots, S_m)}\ F_1 \quad \ddots \quad F_m \qquad G}{S \sqsubseteq u}$$

where G is UNJOIN, F_i is given by:

$$\textsc{Trans}\ \frac{S \sqsubseteq \bigsqcup(\overbrace{u, \ldots, u}^{i-1}, S_i, \ldots, S_m) \qquad Q_i}{S \sqsubseteq \bigsqcup(u, \ldots, u, S_{i+1}, \ldots, S_m)}$$

and Q_i is the proof tree:

$$\textsc{Mono}\ \frac{S_i \sqsubseteq u}{\bigsqcup(u, \ldots, u, S_i, \ldots, S_m) \sqsubseteq \bigsqcup(\underbrace{u, \ldots, u}_{i}, S_{i+1}, \ldots, S_m)}$$

R_6 is the same as R_5, except that it includes UNJOIN and replaces rules of the form B_3 where $m > 1$ with their simplified versions. R_6 is sound, harmonic and of the desired form.

2.5 Discussion

R_6 can be made more powerful in several ways. For example, the following generalization of JOIN is sound and preserves Harmony 1:

$$\textsc{Join}' \ \frac{}{u \sqsubseteq \bigsqcup(\ldots, u, \ldots)}$$

The same holds for the reflexivity of refinement, which generalizes SPEC, and other rules such as UNNEST:

$$\textsc{Unnest} \ \frac{}{\bigsqcup(u_1, \ldots, u_n) \sqsubseteq \bigsqcup(u_1, \ldots, u_{i-1}, u'_1, \ldots, u'_m, u_{i+1}, \ldots, u_n)}$$

provided $1 \leq i \leq n$ and $u_i = \bigsqcup(u'_1, \ldots, u'_m)$.

In specific applications of freefinement, it might also be useful to add derived rules to R_6. Examples of this will follow later.

Freefinement assumes as little as possible about $\models_{\mathsf{V}_1}$ *Sat* and is consequently very generic. As one might expect, additional assumptions can help to construct more powerful refinement systems. For example, suppose 'plus' is a constructor that is commutative in the sense that

$$\forall t_1, t_2 \in T, S \in \mathbb{S} \cdot \models_{\mathsf{V}_1} \mathrm{plus}(t_1, t_2) \mathit{\ Sat\ } S \ \Leftrightarrow\ \models_{\mathsf{V}_1} \mathrm{plus}(t_2, t_1) \mathit{\ Sat\ } S$$

Then $\mathrm{Specs}(\mathrm{plus}([\![u_1]\!], [\![u_2]\!])) = \mathrm{Specs}(\mathrm{plus}([\![u_2]\!], [\![u_1]\!]))$ because

$$\begin{array}{ll} & S \in \mathrm{Specs}(\mathrm{plus}([\![u_1]\!], [\![u_2]\!])) \\ \Leftrightarrow & \forall t_1 \in [\![u_1]\!], t_2 \in [\![u_2]\!] \cdot \models_{\mathsf{V}_1} \mathrm{plus}(t_1, t_2) \mathit{\ Sat\ } S \\ \Leftrightarrow & \forall t_1 \in [\![u_1]\!], t_2 \in [\![u_2]\!] \cdot \models_{\mathsf{V}_1} \mathrm{plus}(t_2, t_1) \mathit{\ Sat\ } S \\ \Leftrightarrow & S \in \mathrm{Specs}(\mathrm{plus}([\![u_2]\!], [\![u_1]\!])) \end{array}$$

So $[\![\mathrm{plus}(u_1, u_2)]\!] = [\![\mathrm{plus}(u_2, u_1)]\!]$ and therefore the refinement rule $\mathrm{plus}(u_1, u_2) \equiv \mathrm{plus}(u_2, u_1)$ is sound. Depending on the rules of V_1, it might also preserve harmony.

As mentioned before, the semantic function $[\![\_]\!]$ and the refinement order $\sqsubseteq$ have nice interpretations when $\models_{\mathsf{V}_1}$ *Sat* is well-behaved. Here is the definition:

Definition 8 (Well-behavedness)**.** $\models_{\mathsf{V}_1}$ *Sat* is well-behaved iff
$\forall C \in \mathbb{K}, t_1, \ldots, t_n \in T, S \in \mathbb{S} \cdot \models_{\mathsf{V}_1} C(t_1, \ldots, t_n) \mathit{\ Sat\ } S \Rightarrow$
$\forall t \in C(\mathrm{Terms}(\mathrm{Specs}(\{t_1\})), \ldots, \mathrm{Terms}(\mathrm{Specs}(\{t_n\}))) \cdot \models_{\mathsf{V}_1} t \mathit{\ Sat\ } S$

There is also an alternative characterization of well-behavedness:

Lemma 9. $\models_{\mathsf{V}_1}$ *Sat* is well-behaved iff
$\forall C \in \mathbb{K}, t_1, \ldots, t_n \in T \cdot \mathrm{Terms}(\mathrm{Specs}(\{C(t_1, \ldots, t_n)\})) =$
$\mathrm{Terms}(\mathrm{Specs}(C(\mathrm{Terms}(\mathrm{Specs}(\{t_1\})), \ldots, \mathrm{Terms}(\mathrm{Specs}(\{t_n\})))))$

Proof. $t_i \in \mathrm{Terms}(\mathrm{Specs}(\{t_i\}))$ for $i \in 1..n$ by Corollary 1.1, so $\{C(t_1, \ldots, t_n)\} \subseteq C(\mathrm{Terms}(\mathrm{Specs}(\{t_1\})), \ldots, \mathrm{Terms}(\mathrm{Specs}(\{t_n\})))$. Hence by Corollary 2.3 in the Appendix, $\mathrm{Specs}(\{C(t_1, \ldots, t_n)\}) \supseteq \mathrm{Specs}(C(\mathrm{Terms}(\mathrm{Specs}(\{t_1\})), \ldots, \mathrm{Terms}(\mathrm{Specs}(\{t_n\}))))$.

Therefore:
$\models_{\mathsf{V}_1}$ *Sat* is well-behaved
$\Leftrightarrow$
$\forall C \in \mathbb{K}, t_1, \ldots, t_n \in T, S \in \mathbb{S} \cdot \models_{\mathsf{V}_1} C(t_1, \ldots, t_n) \mathit{\ Sat\ } S \Rightarrow$
$\forall t \in C(\mathrm{Terms}(\mathrm{Specs}(\{t_1\})), \ldots, \mathrm{Terms}(\mathrm{Specs}(\{t_n\}))) \cdot \models_{\mathsf{V}_1} t \mathit{\ Sat\ } S$
$\Leftrightarrow$
$\forall C \in \mathbb{K}, t_1, \ldots, t_n \in T, S \in \mathbb{S} \cdot S \in \mathrm{Specs}(\{C(t_1, \ldots, t_n)\}) \Rightarrow$
$S \in \mathrm{Specs}(C(\mathrm{Terms}(\mathrm{Specs}(\{t_1\})), \ldots, \mathrm{Terms}(\mathrm{Specs}(\{t_n\}))))$
$\Leftrightarrow$
$\forall C \in \mathbb{K}, t_1, \ldots, t_n \in T \cdot \mathrm{Specs}(\{C(t_1, \ldots, t_n)\}) \subseteq$
$\mathrm{Specs}(C(\mathrm{Terms}(\mathrm{Specs}(\{t_1\})), \ldots, \mathrm{Terms}(\mathrm{Specs}(\{t_n\}))))$
$\Leftrightarrow$ {by the reasoning above}
$\forall C \in \mathbb{K}, t_1, \ldots, t_n \in T \cdot \mathrm{Specs}(\{C(t_1, \ldots, t_n)\}) =$
$\mathrm{Specs}(C(\mathrm{Terms}(\mathrm{Specs}(\{t_1\})), \ldots, \mathrm{Terms}(\mathrm{Specs}(\{t_n\}))))$
The result then follows by Corollary 1.3. □

Freefinement does not require well-behavedness of $\models_{\mathsf{V}_1}$ *Sat*, but the next theorem shows that the intuitions behind the definitions are simple when $\models_{\mathsf{V}_1}$ *Sat* is well-behaved. For example, Theorem 6.3 says that $[\![u]\!]$ is the set of all primitive terms that refine u.

Theorem 6. If $\models_{\mathsf{V}_1}$ *Sat* is well-behaved, then

6.1 $\forall t \in T \cdot [\![t]\!] = \mathrm{Terms}(\mathrm{Specs}(\{t\}))$
6.2 $\forall t \in T, S \in \mathbb{S} \cdot \models_{\mathsf{V}_1} t \mathit{\ Sat\ } S \ \Leftrightarrow\ \models_{\mathsf{V}_2} t \mathit{\ Sat\ } S$
6.3 $\forall t \in T, u \in U \cdot t \in [\![u]\!] \ \Leftrightarrow\ \models u \sqsubseteq t$

Proof.

6.1 By induction on the structure of t. Suppose $t = C(t_1, \ldots, t_n)$ and assume $[\![t_i]\!] = \mathrm{Terms}(\mathrm{Specs}(\{t_i\}))$ for $i \in 1..n$. Then:
$[\![t]\!] = \mathrm{Terms}(\mathrm{Specs}(C([\![t_1]\!], \ldots, [\![t_n]\!])))$
$=$ {induction hypothesis}
$\mathrm{Terms}(\mathrm{Specs}(C(\mathrm{Terms}(\mathrm{Specs}(\{t_1\})), \ldots, \mathrm{Terms}(\mathrm{Specs}(\{t_n\})))))$
$=$ {Lemma 9}
$\mathrm{Terms}(\mathrm{Specs}(\{C(t_1, \ldots, t_n)\})) = \mathrm{Terms}(\mathrm{Specs}(\{t\}))$.

6.2
$\models_{\mathsf{V}_1} t \mathit{\ Sat\ } S$
$\Leftrightarrow$ {definition of Specs}
$S \in \mathrm{Specs}(\{t\})$
$\Leftrightarrow$ {Corollary 2.7 in the Appendix}
$S \in \mathrm{Specs}(\mathrm{Terms}(\mathrm{Specs}(\{t\})))$
$\Leftrightarrow$ {Theorem 6.1}
$S \in \mathrm{Specs}([\![t]\!])$
$\Leftrightarrow$ {Lemma 4}
$\models_{\mathsf{V}_2} t \mathit{\ Sat\ } S$

6.3 The $\Leftarrow$ proof is trivial since $t \in [\![t]\!]$. For $\Rightarrow$, assume $\{t\} \subseteq [\![u]\!]$. Then $\mathrm{Terms}(\mathrm{Specs}(\{t\})) \subseteq \mathrm{Terms}(\mathrm{Specs}([\![u]\!]))$ by Corollary 2.5 in the Appendix, and $[\![t]\!] \subseteq [\![u]\!]$ by Theorem 6.1 and Lemma 2. □

Whether $\models_{\mathsf{V}_1}$ *Sat* is well-behaved depends partly on the expressivity of specifications. For example, suppose

$$\mathbb{K} = \{x := e \mid e \text{ is an arithmetic expression}\} \cup \{\_ \fatsemi \_\}$$

i.e. there is a nullary constructor $x := e$ for all arithmetic expressions e, and a binary constructor for sequential composition. Suppose $\mathbb{S} = \{\mathrm{Even\_x}\}$, and $\models_{\mathsf{V}_1} t \mathit{\ Sat\ } \mathrm{Even\_x}$ holds iff, if t is executed in any state where x is even, then x is even in every resulting state. So $\models_{\mathsf{V}_1} x := x + 1 \fatsemi x := x + 1 \mathit{\ Sat\ } \mathrm{Even\_x}$, but it is not the case that $\models_{\mathsf{V}_1} x := x + 1 \mathit{\ Sat\ } \mathrm{Even\_x}$. In fact, $x := x + 1$ does not satisfy any specification. This implies that $\mathrm{Terms}(\mathrm{Specs}(\{x := x + 1\})) = T$, so $x := 1 \fatsemi x := 1 \in \mathrm{Terms}(\mathrm{Specs}(\{x := x + 1\})) \fatsemi \mathrm{Terms}(\mathrm{Specs}(\{x := x + 1\}))$. But $\models_{\mathsf{V}_1} x := 1 \fatsemi x := 1 \mathit{\ Sat\ } \mathrm{Even\_x}$ does not hold, hence $\models_{\mathsf{V}_1}$ *Sat* is not well-behaved.

Even though $\models_{\mathsf{V}_1}$ *Sat* is not well-behaved, it is still possible to have inference rules that are amenable to freefinement, for example:

$$1\ \frac{}{x := e \mathit{\ Sat\ } \mathrm{Even\_x}} \qquad 2\ \frac{t \mathit{\ Sat\ } \mathrm{Even\_x} \qquad t' \mathit{\ Sat\ } \mathrm{Even\_x}}{t \fatsemi t' \mathit{\ Sat\ } \mathrm{Even\_x}}$$

provided $e \in \{\ldots, -2, 0, 2, \ldots\}$.

If $\mathbb{S}$ is instead a set of specifications of the form $[P, Q]$, where P is a precondition and Q a postcondition, and

$$\models_{\mathsf{V}_1} t \fatsemi t' \mathit{\ Sat\ } [P, Q] \ \Leftrightarrow\ \exists R \cdot \models_{\mathsf{V}_1} t \mathit{\ Sat\ } [P, R] \wedge \models_{\mathsf{V}_1} t' \mathit{\ Sat\ } [R, Q]$$

then it is easy to show that this $\models_{\mathsf{V}_1}$ *Sat* is well-behaved.

The completeness of V_1 is a sufficient condition for the well-behavedness of $\models_{\mathsf{V}_1}$ *Sat*:

Theorem 7. If V_1 is complete, then $\models_{\mathsf{V}_1}$ *Sat* is well-behaved.

Proof. If V_1 is complete, then $\models_{\mathsf{V}_1} C(t_1, \ldots, t_n) \mathit{\ Sat\ } S \ \Leftrightarrow\ \vdash_{\mathsf{V}_1} C(t_1, \ldots, t_n) \mathit{\ Sat\ } S$. The well-behavedness of $\models_{\mathsf{V}_1}$ *Sat* follows by induction on the derivation of $C(t_1, \ldots, t_n) \mathit{\ Sat\ } S$:

- For each rule of the form A_1, assume Pred(C, $S_1, \ldots, S_n$, S) and $S_i \in \text{Specs}(\{t_i\})$ for all $i \in 1..n$. So $\forall t_i' \in \text{Terms}(\text{Specs}(\{t_i\}))\cdot \models_{V_1} t_i'$ *Sat* S_i for all $i \in 1..n$. The rule is sound with respect to Definition 1.1, hence $\forall t \in C(\text{Terms}(\text{Specs}(\{t_1\})), \ldots, \text{Terms}(\text{Specs}(\{t_n\})))\cdot \models_{V_1} t$ *Sat* S.
- For each rule of the form B_1, assume Pred($S_1, \ldots, S_m$, S) and $\forall t \in C(\text{Terms}(\text{Specs}(\{t_1\})), \ldots, \text{Terms}(\text{Specs}(\{t_n\})))\cdot \models_{V_1} t$ *Sat* S_i for all $i \in 1..m$. The rule is sound w.r.t. Definition 1.2, so $\forall t \in C(\text{Terms}(\text{Specs}(\{t_1\})), \ldots, \text{Terms}(\text{Specs}(\{t_n\})))\cdot \models_{V_1} t$ *Sat* S. □

3. Applications

3.1 Lambda Calculus

The top left corner of Figure 1 contains a type system λ_1 for the lambda calculus. By considering pairs of the form (typing context, type) as specifications, it is possible to apply freefinement and obtain a refinement calculus for (extended) lambda terms in the spirit of Denney [4]. The inputs to freefinement are as follows:

1. $\mathbb{K} = \text{Var} \cup \{\lambda x.\ \_ \mid x \in \text{Var}\} \cup \{\_\ \_\}$
   Note that $\mathbb{K}$ defines the language T of lambda terms:
   $$e ::= x \mid \lambda x.\, e \mid e\ \ e'$$
   Here and in the following, x ranges over the set of variables Var, and e ranges over T.
2. $\mathbb{S} = \{[\Gamma; \tau] \mid \Gamma \in \text{Context} \wedge \tau \in \text{Type}\}$, where Context is the set of typing contexts and Type is the set of types that contains the type constructor $\_ \to \_$. The intended representation of a typing context Γ is a list of variable names paired with types. Variables may appear more than once in Γ, and variable lookup uses the rightmost occurrence. In the following, σ and τ range over Type, and Γ ranges over Context.
3. $\models_{V_1} \_$ *Sat* $\_$ is defined by:
   - $\models_{V_1} x$ *Sat* $[\Gamma; \tau] \Leftrightarrow x : \tau \in \Gamma$
   - $\models_{V_1} \lambda x.\, e$ *Sat* $[\Gamma; \tau] \Leftrightarrow$ $\exists \sigma, \tau' \cdot \tau = \sigma \to \tau' \wedge \models_{V_1} e$ *Sat* $[\Gamma, x : \sigma; \tau']$
   - $\models_{V_1} e\ \ e'$ *Sat* $[\Gamma; \tau] \Leftrightarrow$ $\exists \sigma \cdot \models_{V_1} e$ *Sat* $[\Gamma; \sigma \to \tau] \wedge \models_{V_1} e'$ *Sat* $[\Gamma; \sigma]$
4. V_1, shown in the top right corner of Figure 1, is obtained from λ_1 by replacing $\Gamma \vdash e : \tau$ with e *Sat* $[\Gamma; \tau]$. The rules VAR, ABS and APP are all of the form A_1 with $n = 0$, 1 and 2 respectively. For example, in the case of ABS, Pred(C, S_1, S) is defined as $\exists x, \Gamma, \sigma, \tau \cdot C = \lambda x.\ \_ \wedge S_1 = [\Gamma, x : \sigma; \tau] \wedge S = [\Gamma; \sigma \to \tau]$.

Since V_1 does not contain rules of the form B_1 where $m > 1$, freefinement does not add join terms to the lambda calculus. The system λ_2 in Figure 1 is V_2 where f *Sat* $[\Gamma; \tau]$ is written instead as $\Gamma \vdash f : \tau$. The system R_5, shown in the bottom right of Figure 1, is the final harmonic refinement calculus that freefinement produces.

Here is an example top-down typing derivation with R_5:

$[\Gamma; (\sigma \to \tau) \to (\sigma \to \tau)]$
$\sqsubseteq$ "ABS"
$\lambda x.\, [\Gamma, x : \sigma \to \tau; \sigma \to \tau]$
$\sqsubseteq$ "MONO with ABS"
$\lambda x.\, \lambda y.\, [\Gamma, x : \sigma \to \tau, y : \sigma; \tau]$
$\sqsubseteq$ "MONO with APP"
$\lambda x.\, \lambda y.\, ([\Gamma, x : \sigma \to \tau, y : \sigma; \sigma \to \tau]\ \ [\Gamma, x : \sigma \to \tau, y : \sigma; \sigma])$
$\sqsubseteq$ "Twice MONO with VAR"
$\lambda x.\, \lambda y.\, (x\ \ y)$

λ_1

$$\text{VAR}\ \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \qquad \text{ABS}\ \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x.\, e : \sigma \to \tau} \qquad \text{APP}\ \frac{\Gamma \vdash e : \sigma \to \tau \quad \Gamma \vdash e' : \sigma}{\Gamma \vdash e\ \ e' : \tau}$$

V_1

$$\text{VAR}\ \frac{}{x\ \ \textit{Sat}\ \ [\Gamma; \tau]} \quad \text{provided } x : \tau \in \Gamma. \qquad \text{ABS}\ \frac{e\ \ \textit{Sat}\ \ [\Gamma, x : \sigma; \tau]}{\lambda x.\, e\ \ \textit{Sat}\ \ [\Gamma; \sigma \to \tau]} \qquad \text{APP}\ \frac{e\ \ \textit{Sat}\ \ [\Gamma; \sigma \to \tau] \quad e'\ \ \textit{Sat}\ \ [\Gamma; \sigma]}{e\ \ e'\ \ \textit{Sat}\ \ [\Gamma; \tau]}$$

λ_2

$$\text{VAR}\ \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \qquad \text{ABS}\ \frac{\Gamma, x : \sigma \vdash f : \tau}{\Gamma \vdash \lambda x.\, f : \sigma \to \tau} \qquad \text{APP}\ \frac{\Gamma \vdash f : \sigma \to \tau \quad \Gamma \vdash f' : \sigma}{\Gamma \vdash f\ \ f' : \tau} \qquad \text{SPEC}\ \frac{}{\Gamma \vdash [\Gamma; \sigma] : \sigma}$$

R_5

$$\text{VAR}\ \frac{}{[\Gamma; \tau] \sqsubseteq x} \quad \text{provided } x : \tau \in \Gamma. \qquad \text{ABS}\ \frac{}{[\Gamma; \sigma \to \tau] \sqsubseteq \lambda x.\, [\Gamma, x : \sigma; \tau]} \qquad \text{APP}\ \frac{}{[\Gamma; \tau] \sqsubseteq [\Gamma; \sigma \to \tau]\ \ [\Gamma; \sigma]}$$

$$\text{SPEC}\ \frac{}{[\Gamma; \sigma] \sqsubseteq [\Gamma; \sigma]} \qquad \text{TRANS}\ \frac{f_1 \sqsubseteq f_2 \qquad f_2 \sqsubseteq f_3}{f_1 \sqsubseteq f_3} \qquad \text{MONO}\ \frac{f \sqsubseteq f'}{g[f] \sqsubseteq g[f']}$$

Figure 1. Freefinement and a typed lambda calculus

Since R_5 is harmonic and V_2 is a sound and conservative extension of V_1, it holds that $\vdash_{\lambda_1} \Gamma \vdash \lambda x.\, \lambda y.\, (x\ \ y) : (\sigma \to \tau) \to (\sigma \to \tau)$.

One might wish to extend R_5 using knowledge particular to lambda calculus typing. It is simple to show that V_1 is complete, so

$$\vdash_{\lambda_1} \Gamma \vdash e : \tau \Leftrightarrow \vdash_{V_1} e\ \textit{Sat}\ [\Gamma; \tau] \Leftrightarrow \models_{V_1} e\ \textit{Sat}\ [\Gamma; \tau]$$

Furthermore, by Theorems 7 and 6.2,

$$\models_{V_1} e\ \textit{Sat}\ [\Gamma; \tau] \Leftrightarrow \models_{V_2} e\ \textit{Sat}\ [\Gamma; \tau]$$

and because V_2 is a sound and conservative extension of V_1,

$$\vdash_{V_1} e\ \textit{Sat}\ [\Gamma; \tau] \Leftrightarrow \vdash_{V_2} e\ \textit{Sat}\ [\Gamma; \tau]$$

Consider the property of *preservation*:

Definition 9. A relation $\rightsquigarrow \subseteq T \times T$ satisfies preservation $\stackrel{\text{def}}{=}$ $\forall \Gamma, \tau, e, e' \cdot$ if $\vdash_{\lambda_1} \Gamma \vdash e : \tau$ and $e \rightsquigarrow e'$, then $\vdash_{\lambda_1} \Gamma \vdash e' : \tau$.

Theorem 8. If $\rightsquigarrow$ satisfies preservation, then:

8.1 If $e \rightsquigarrow e'$, then $\models e \sqsubseteq e'$.
8.2 If $\vdash_{V_2} e$ Sat $[\Gamma; \tau]$ and $e \rightsquigarrow e'$, then $\vdash_{V_2} e'$ *Sat* $[\Gamma; \tau]$.

Proof. The proof of 8.2 is trivial. For 8.1:
$\forall \Gamma, \tau, e, e' \cdot \vdash_{\lambda_1} \Gamma \vdash e : \tau \ \wedge\ e \rightsquigarrow e' \ \Rightarrow\ \vdash_{\lambda_1} \Gamma \vdash e' : \tau$
$\Leftrightarrow$ {predicate logic}
$\forall e, e' \cdot e \rightsquigarrow e' \ \Rightarrow\ (\forall \Gamma, \tau \cdot \vdash_{\lambda_1} \Gamma \vdash e : \tau \ \Rightarrow\ \vdash_{\lambda_1} \Gamma \vdash e' : \tau)$
$\Leftrightarrow$
$\forall e, e' \cdot e \rightsquigarrow e' \ \Rightarrow\ (\forall S \in \mathbb{S} \cdot \models_{V_2} e\ \textit{Sat}\ S \ \Rightarrow\ \models_{V_2} e'\ \textit{Sat}\ S)$
$\Leftrightarrow$ {Lemma 5}
$\forall e, e' \cdot e \rightsquigarrow e' \ \Rightarrow\ \models e \sqsubseteq e'$ □

So any relation that satisfies preservation contains only sound refinements that satisfy Harmony 1, and can augment R_5 to yield a sound and harmonic refinement system. Examples of relations that satisfy preservation include:

- The α-conversion relation.
- The β-reduction relation.
- The η-contraction relation. So $\lambda x.\,(e\ x) \sqsubseteq e$, provided x does not appear free in e.
- The relation $\leq$ on closed terms, where $e \leq e'$ exactly when e has fewer types than e'.

Here is a small example that uses the η-contraction extension:

$$\begin{array}{ll} & \lambda x.\,\lambda y.\,\lambda z.\,((x\ y)\ z) \\ \sqsubseteq & \{\textsc{Mono} \text{ with } \eta\text{-contraction}\} \\ & \lambda x.\,\lambda y.\,(x\ y) \\ \sqsubseteq & \{\textsc{Mono} \text{ with } \eta\text{-contraction}\} \\ & \lambda x.\,x \end{array}$$

3.2 Hoare Logic

The top left corner of Figure 2 contains system H, a Hoare logic for simple imperative programs. P is a precondition, Q a postcondition, and c a command in the Hoare triple $\{P\}c\{Q\}$, and $\models_{\mathsf{H}}\{P\}c\{Q\}$ is the usual partial correctness interpretation of $\{P\}c\{Q\}$. By interpreting a specification as a pre-post pair, the rules of H do not fit the rule forms A_1 and B_1, since the proviso of AUXVARELIM inspects the command c to determine the variables that it writes and reads. However, if specifications also keep track of written and read variables, then it becomes possible to apply freefinement to obtain a refinement calculus in the spirit of Morgan [9]. Here are the inputs:

1. There are constructors for assignments, sequential composition, conditionals and loops:

$$\begin{array}{rcl} \mathbb{K} & = & \{x := e \mid x \in \mathrm{Var} \ \wedge\ e \in \mathrm{IntExp}\} \\ & & \cup\ \{\_ \fatsemi \_\} \\ & & \cup\ \{\textbf{if}\ b\ \textbf{then}\ \_\ \textbf{else}\ \_ \mid b \in \mathrm{BoolExp}\} \\ & & \cup\ \{\textbf{while}\ b\ \textbf{do}\ \_ \mid b \in \mathrm{BoolExp}\} \end{array}$$

2. A specification consists of two sets of variables and two assertions, written in a notation resembling Morgan's specification statement [8]:

$$\mathbb{S} = \left\{\overline{x}; \overline{y} : \{P, Q\} \mid \overline{x}, \overline{y} \in \mathcal{P}(\mathrm{Var}) \ \wedge\ P, Q \in \mathrm{Assertion}\right\}$$

3. In the specification $\overline{x}; \overline{y} : \{P, Q\}$, the $\overline{x}$ and $\overline{y}$ are upper bounds on the sets of variables written and read by the command respectively, the P is a precondition and the Q a postcondition:

$$\models_{\mathsf{V}_1} c\ \mathit{Sat}\ \overline{x}; \overline{y} : \{P, Q\} \stackrel{\text{def}}{=} \mathit{writes}(c) \subseteq \overline{x} \ \wedge\ \mathit{reads}(c) \subseteq \overline{y} \ \wedge\ \models_{\mathsf{H}}\{P\}c\{Q\}$$

4. V_1, shown in the top right corner of Figure 2, has the following relationship with H:

$$\vdash_{\mathsf{V}_1} c\ \mathit{Sat}\ \overline{x}; \overline{y} : \{P, Q\} \ \Leftrightarrow\ \mathit{writes}(c) \subseteq \overline{x} \ \wedge\ \mathit{reads}(c) \subseteq \overline{y} \ \wedge\ \vdash_{\mathsf{H}}\{P\}c\{Q\}$$

   Note that:

   - The non-structural rules of H have counterparts in V_1 that embody the definitions of *writes* and *reads*. For example, the conclusion of COND reflects that

     $\mathit{writes}(\textbf{if}\ b\ \textbf{then}\ c\ \textbf{else}\ c') \stackrel{\text{def}}{=} \mathit{writes}(c) \cup \mathit{writes}(c')$ and $\mathit{reads}(\textbf{if}\ b\ \textbf{then}\ c\ \textbf{else}\ c') \stackrel{\text{def}}{=} \mathit{reads}(c) \cup \mathit{reads}(c') \cup \mathit{FV}(b)$.
   - The structural rules of H that inspect c for its write and/or read sets have counterparts in V_1 that consult the specification instead. See for example the proviso of AUXVARELIM.
   - CONSEQUENCE in V_1 allows the enlargement of write and read sets. This loosening of the bounds is useful in refinement developments, because then the resulting code is not forced to write and read all the variables that were originally available for writing and reading.

   The V_1-counterparts of the structural rules of H are all of the form B_1. For example, $m = 1$ in the case of CONSTANCY, and $m = 2$ for DISJ. The other rules are of the form A_1. For example, $n = 2$ in the case of COND, and $n = 1$ for LOOP.

The systems V_2 and R_6 that freefinement produces appear at the bottom of Figure 2. R_6 yields several derived rules that may be useful in practical refinement developments. For example, the rule:

$$\textsc{DerivedVarAssign}\ \frac{}{\overline{x}; \overline{y} : \{P, Q\} \sqsubseteq z := e}$$
provided $z \in \overline{x}$ and $\mathit{FV}(e) \subseteq \overline{y}$ and $P \Rightarrow Q[e/z]$.

can replace VARASSIGN, and is similar to the assignment law of Morgan (Law 1.3 on p. 8 of [9]). Likewise, the derived rule:

$$\textsc{FollowingVarAssign}\ \frac{}{\overline{x}; \overline{y} : \{P, Q\} \sqsubseteq \overline{x}; \overline{y} : \{P, Q[e/z]\} \fatsemi z := e}$$
provided $z \in \overline{x}$ and $\mathit{FV}(e) \subseteq \overline{y}$.

is similar to the *following assignment* law of Morgan (Law 3.5 on p. 32 of [9]).

Here is an example showing that R_6 can derive a correct factorial algorithm starting with its specification:

$$\begin{array}{ll} & y,z;\ x,y,z : \{\mathrm{true},\ y = x!\} \\ \sqsubseteq & \text{“SEQCOMP”} \\ & y,z;\ \emptyset : \{\mathrm{true},\ y = 1 \wedge z = 0\} \fatsemi y,z;\ x,y,z : \{y = 1 \wedge z = 0,\ y = x!\} \end{array}$$

The first spec statement is refined as follows:

$$\begin{array}{ll} & y,z;\ \emptyset : \{\mathrm{true},\ y = 1 \wedge z = 0\} \\ \sqsubseteq & \text{“SEQCOMP”} \\ & y;\ \emptyset : \{\mathrm{true},\ y = 1\} \fatsemi z;\ \emptyset : \{y = 1,\ y = 1 \wedge z = 0\} \\ \sqsubseteq & \text{“Twice MONO with CONSEQUENCE”} \\ & y;\ \emptyset : \{1 = 1,\ y = 1\} \fatsemi z;\ \emptyset : \{y = 1 \wedge 0 = 0,\ y = 1 \wedge z = 0\} \\ \sqsubseteq & \text{“Twice MONO with VARASSIGN”} \\ & y := 1 \fatsemi z := 0 \end{array}$$

And for the second spec statement:

$$\begin{array}{ll} & y,z;\ x,y,z : \{y = 1 \wedge z = 0,\ y = x!\} \\ \sqsubseteq & \text{“CONSEQUENCE”} \\ & y,z;\ x,y,z : \{y = z!,\ y = z! \wedge \neg z{\neq}x\} \\ \sqsubseteq & \text{“LOOP”} \\ & \textbf{while}\ z{\neq}x\ \textbf{do}\ y,z;\ y,z : \{y = z! \wedge z{\neq}x,\ y{\cdot}z = z!\} \\ \sqsubseteq & \text{“MONO with SEQCOMP”} \\ & \textbf{while}\ z{\neq}x\ \textbf{do}\ z;\ z : \{y = z! \wedge z{\neq}x,\ y{\cdot}z = z!\} \fatsemi y;\ y,z : \{y{\cdot}z = z!,\ y = z!\} \end{array}$$

H

$$\textsc{AuxVarElim}\ \frac{\{P\}c\{Q\}}{\{\exists v \cdot P\}c\{\exists v \cdot Q\}}$$
provided $v \notin \mathit{writes}(c) \cup \mathit{reads}(c)$.

$$\textsc{Consequence}\ \frac{\{P'\}c\{Q'\}}{\{P\}c\{Q\}}$$
provided $P \Rightarrow P'$ and $Q' \Rightarrow Q$.

$$\textsc{Constancy}\ \frac{\{P\}c\{Q\}}{\{P \wedge R\}c\{Q \wedge R\}}$$
provided $FV(R) \cap \mathit{writes}(c) = \emptyset$.

$$\textsc{Disj}\ \frac{\{P\}c\{Q\} \qquad \{P'\}c\{Q'\}}{\{P \vee P'\}c\{Q \vee Q'\}}$$

$$\textsc{VarAssign}\ \frac{}{\{P[e/x]\}x := e\{P\}}$$

$$\textsc{SeqComp}\ \frac{\{P\}c\{Q\} \qquad \{Q\}c'\{R\}}{\{P\}c \fatsemi c'\{R\}}$$

$$\textsc{Cond}\ \frac{\{P \wedge b\}c\{Q\} \qquad \{P \wedge \neg b\}c'\{Q\}}{\{P\}\textbf{if } b \textbf{ then } c \textbf{ else } c'\{Q\}}$$

$$\textsc{Loop}\ \frac{\{I \wedge b\}c\{I\}}{\{I\}\textbf{while } b \textbf{ do } c\{I \wedge \neg b\}}$$

V_1

$$\textsc{AuxVarElim}\ \frac{c \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P, Q\}}{c \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{\exists v \cdot P, \exists v \cdot Q\}}$$
provided $v \notin \overline{x} \cup \overline{y}$.

$$\textsc{Consequence}\ \frac{c \quad \mathit{Sat} \quad \overline{x'}; \overline{y'} : \{P', Q'\}}{c \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P, Q\}}$$
provided $\overline{x} \supseteq \overline{x'}$ and $\overline{y} \supseteq \overline{y'}$ and $P \Rightarrow P'$ and $Q' \Rightarrow Q$.

$$\textsc{Constancy}\ \frac{c \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P, Q\}}{c \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P \wedge R, Q \wedge R\}}$$
provided $FV(R) \cap \overline{x} = \emptyset$.

$$\textsc{Disj}\ \frac{c \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P, Q\} \qquad c \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P', Q'\}}{c \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P \vee P', Q \vee Q'\}}$$

$$\textsc{VarAssign}\ \frac{}{x := e \quad \mathit{Sat} \quad x; FV(e) : \{P[e/x], P\}}$$

$$\textsc{SeqComp}\ \frac{c \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P, Q\} \qquad c' \quad \mathit{Sat} \quad \overline{x'}; \overline{y'} : \{Q, R\}}{c \fatsemi c' \quad \mathit{Sat} \quad \overline{x} \cup \overline{x'}; \overline{x} \cup \overline{x'} : \{P, R\}}$$

$$\textsc{Cond}\ \frac{c \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P \wedge b, Q\} \qquad c' \quad \mathit{Sat} \quad \overline{x'}; \overline{y'} : \{P \wedge \neg b, Q\}}{\textbf{if } b \textbf{ then } c \textbf{ else } c' \quad \mathit{Sat} \quad \overline{x} \cup \overline{x'}; \overline{y} \cup \overline{y'} \cup FV(b) : \{P, Q\}}$$

$$\textsc{Loop}\ \frac{c \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{I \wedge b, I\}}{\textbf{while } b \textbf{ do } c \quad \mathit{Sat} \quad \overline{x}; \overline{y} \cup FV(b) : \{I, I \wedge \neg b\}}$$

V_2

$$\textsc{AuxVarElim}\ \frac{s \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P, Q\}}{s \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{\exists v \cdot P, \exists v \cdot Q\}}$$
provided $v \notin \overline{x} \cup \overline{y}$.

$$\textsc{Consequence}\ \frac{s \quad \mathit{Sat} \quad \overline{x'}; \overline{y'} : \{P', Q'\}}{s \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P, Q\}}$$
provided $\overline{x} \supseteq \overline{x'}$ and $\overline{y} \supseteq \overline{y'}$ and $P \Rightarrow P'$ and $Q' \Rightarrow Q$.

$$\textsc{Constancy}\ \frac{s \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P, Q\}}{s \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P \wedge R, Q \wedge R\}}$$
provided $FV(R) \cap \overline{x} = \emptyset$.

$$\textsc{Disj}\ \frac{s \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P, Q\} \qquad s \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P', Q'\}}{s \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P \vee P', Q \vee Q'\}}$$

$$\textsc{VarAssign}\ \frac{}{x := e \quad \mathit{Sat} \quad x; FV(e) : \{P[e/x], P\}}$$

$$\textsc{SeqComp}\ \frac{s \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P, Q\} \qquad s' \quad \mathit{Sat} \quad \overline{x'}; \overline{y'} : \{Q, R\}}{s \fatsemi s' \quad \mathit{Sat} \quad \overline{x} \cup \overline{x'}; \overline{x} \cup \overline{x'} : \{P, R\}}$$

$$\textsc{Cond}\ \frac{s \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P \wedge b, Q\} \qquad s' \quad \mathit{Sat} \quad \overline{x'}; \overline{y'} : \{P \wedge \neg b, Q\}}{\textbf{if } b \textbf{ then } s \textbf{ else } s' \quad \mathit{Sat} \quad \overline{x} \cup \overline{x'}; \overline{y} \cup \overline{y'} \cup FV(b) : \{P, Q\}}$$

$$\textsc{Loop}\ \frac{s \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{I \wedge b, I\}}{\textbf{while } b \textbf{ do } s \quad \mathit{Sat} \quad \overline{x}; \overline{y} \cup FV(b) : \{I, I \wedge \neg b\}}$$

$$\textsc{Spec}\ \frac{}{\overline{x}; \overline{y} : \{P, Q\} \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P, Q\}}$$

$$\textsc{Join}\ \frac{s \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P, Q\}}{\bigsqcup(\ldots, s, \ldots) \quad \mathit{Sat} \quad \overline{x}; \overline{y} : \{P, Q\}}$$

R_6

$$\textsc{AuxVarElim}\ \frac{}{\overline{x}; \overline{y} : \{\exists v \cdot P, \exists v \cdot Q\} \sqsubseteq \overline{x}; \overline{y} : \{P, Q\}}$$
provided $v \notin \overline{x} \cup \overline{y}$.

$$\textsc{Consequence}\ \frac{}{\overline{x}; \overline{y} : \{P, Q\} \sqsubseteq \overline{x'}; \overline{y'} : \{P', Q'\}}$$
provided $\overline{x} \supseteq \overline{x'}$ and $\overline{y} \supseteq \overline{y'}$ and $P \Rightarrow P'$ and $Q' \Rightarrow Q$.

$$\textsc{Constancy}\ \frac{}{\overline{x}; \overline{y} : \{P \wedge R, Q \wedge R\} \sqsubseteq \overline{x}; \overline{y} : \{P, Q\}}$$
provided $FV(R) \cap \overline{x} = \emptyset$.

$$\textsc{Disj}\ \frac{}{\overline{x}; \overline{y} : \{P \vee P', Q \vee Q'\} \sqsubseteq \bigsqcup(\overline{x}; \overline{y} : \{P, Q\}, \overline{x}; \overline{y} : \{P', Q'\})}$$

$$\textsc{VarAssign}\ \frac{}{x; FV(e) : \{P[e/x], P\} \sqsubseteq x := e}$$

$$\textsc{SeqComp}\ \frac{}{\overline{x} \cup \overline{x'}; \overline{x} \cup \overline{x'} : \{P, R\} \sqsubseteq \overline{x}; \overline{y} : \{P, Q\} \fatsemi \overline{x'}; \overline{y'} : \{Q, R\}}$$

$$\textsc{Cond}\ \frac{}{\begin{array}{l}\overline{x} \cup \overline{x'}; \overline{y} \cup \overline{y'} \cup FV(b) : \{P, Q\} \\ \sqsubseteq \textbf{if } b \textbf{ then } \overline{x}; \overline{y} : \{P \wedge b, Q\} \textbf{ else } \overline{x'}; \overline{y'} : \{P \wedge \neg b, Q\}\end{array}}$$

$$\textsc{Loop}\ \frac{}{\overline{x}; \overline{y} \cup FV(b) : \{I, I \wedge \neg b\} \sqsubseteq \textbf{while } b \textbf{ do } \overline{x}; \overline{y} : \{I \wedge b, I\}}$$

$$\textsc{Spec}\ \frac{}{\overline{x}; \overline{y} : \{P, Q\} \sqsubseteq \overline{x}; \overline{y} : \{P, Q\}}$$

$$\textsc{Join}\ \frac{}{\overline{x}; \overline{y} : \{P, Q\} \sqsubseteq \bigsqcup(\ldots, \overline{x}; \overline{y} : \{P, Q\}, \ldots)}$$

$$\textsc{Trans}\ \frac{s_1 \sqsubseteq s_2 \qquad s_2 \sqsubseteq s_3}{s_1 \sqsubseteq s_3}$$

$$\textsc{Mono}\ \frac{s \sqsubseteq s'}{t[s] \sqsubseteq t[s']}$$

$$\textsc{Unjoin}\ \frac{}{\bigsqcup(s, \ldots, s) \sqsubseteq s}$$

Figure 2. Freefinement and Hoare logic

$\sqsubseteq$ "MONO with VARASSIGN"

while z≠x **do** z; z : {y = z! ∧ z≠x, y·z = z!} ⨟ y := y·z

$\sqsubseteq$ "MONO with CONSEQUENCE"

while z≠x **do** z; z : {y·(z+1) = (z+1)!, y·z = z!} ⨟ y := y·z

$\sqsubseteq$ "MONO with VARASSIGN"

while z≠x **do** z := z+1 ⨟ y := y·z

Since $\vdash_{R_6}$ y,z; x,y,z : {true, y = x!} $\sqsubseteq$ y := 1 ⨟ z := 0 ⨟ **while** z≠x **do** z := z+1 ⨟ y := y·z, it is the case that $\vdash_{V_1}$ y := 1 ⨟ z := 0 ⨟ **while** z≠x **do** z := z+1 ⨟ y := y·z *Sat* y,z; x,y,z : {true, y = x!} and hence also $\vdash_H$ {true}y := 1 ⨟ z := 0 ⨟ **while** z≠x **do** z := z+1 ⨟ y := y·z{y = x!}.

Here is another example of using R_6; it involves join statements. The statement $\bigsqcup(\overline{x}; \overline{y} : \{P_1, Q_1\}, \overline{x}; \overline{y} : \{P_2, Q_2\})$ is the join of the specification statements $\overline{x}; \overline{y} : \{P_1, Q_1\}$ and $\overline{x}; \overline{y} : \{P_2, Q_2\}$. Expressing it as a spec statement is simple because

$$\bigsqcup(\overline{x}; \overline{y} : \{P_1, Q_1\}, \overline{x}; \overline{y} : \{P_2, Q_2\}) \equiv \overline{x}; \overline{y} : \{P_1, Q_1\} \textbf{ also } \{P_2, Q_2\}$$

where the definition of {P₁, Q₁} **also** {P₂, Q₂}, taken from [11], is: {(P₁ ∧ z=1) ∨ (P₂ ∧ z≠1), (Q₁ ∧ z=1) ∨ (Q₂ ∧ z≠1)} where z is fresh. R_6 can derive both directions of refinement. Firstly:

$\bigsqcup(\overline{x}; \overline{y} : \{P_1, Q_1\}, \overline{x}; \overline{y} : \{P_2, Q_2\})$

$\sqsubseteq$ "Twice MONO with CONSEQUENCE"

$\bigsqcup(\overline{x}; \overline{y} : \{\exists z\cdot (P_1 \wedge z{=}1 \vee P_2 \wedge z{\neq}1) \wedge z{=}1, \exists z\cdot (Q_1 \wedge z{=}1 \vee Q_2 \wedge z{\neq}1) \wedge z{=}1\},$
$\overline{x}; \overline{y} : \{\exists z\cdot (P_1 \wedge z{=}1 \vee P_2 \wedge z{\neq}1) \wedge z{\neq}1, \exists z\cdot (Q_1 \wedge z{=}1 \vee Q_2 \wedge z{\neq}1) \wedge z{\neq}1\})$

$\sqsubseteq$ "Twice MONO with AUXVARELIM"

$\bigsqcup(\overline{x}; \overline{y} : \{(P_1 \wedge z{=}1 \vee P_2 \wedge z{\neq}1) \wedge z{=}1, (Q_1 \wedge z{=}1 \vee Q_2 \wedge z{\neq}1) \wedge z{=}1\},$
$\overline{x}; \overline{y} : \{(P_1 \wedge z{=}1 \vee P_2 \wedge z{\neq}1) \wedge z{\neq}1, (Q_1 \wedge z{=}1 \vee Q_2 \wedge z{\neq}1) \wedge z{\neq}1\})$

$\sqsubseteq$ "Twice MONO with CONSTANCY"

$\bigsqcup(\overline{x}; \overline{y} : \{P_1 \wedge z{=}1 \vee P_2 \wedge z{\neq}1, Q_1 \wedge z{=}1 \vee Q_2 \wedge z{\neq}1\},$
$\overline{x}; \overline{y} : \{P_1 \wedge z{=}1 \vee P_2 \wedge z{\neq}1, Q_1 \wedge z{=}1 \vee Q_2 \wedge z{\neq}1\})$

$\sqsubseteq$ "UNJOIN"

$\overline{x}; \overline{y} : \{P_1, Q_1\}$ **also** $\{P_2, Q_2\}$

Secondly:

$\overline{x}; \overline{y} : \{(P_1 \wedge z{=}1) \vee (P_2 \wedge z{\neq}1), (Q_1 \wedge z{=}1) \vee (Q_2 \wedge z{\neq}1)\}$

$\sqsubseteq$ "DISJ"

$\bigsqcup(\overline{x}; \overline{y} : \{P_1 \wedge z{=}1, Q_1 \wedge z{=}1\}, \overline{x}; \overline{y} : \{P_2 \wedge z{\neq}1, Q_2 \wedge z{\neq}1\})$

$\sqsubseteq$ "Twice MONO with CONSTANCY"

$\bigsqcup(\overline{x}; \overline{y} : \{P_1, Q_1\}, \overline{x}; \overline{y} : \{P_2, Q_2\})$

Leino and Manohar [7] mention several uses of the join of spec-like statements.

3.3 Discussion

The type system λ_1 considered above is very simple. Freefinement also applies to System F and other more sophisticated type systems.

Although λ_1 had only rules of the form A_1, typing rules of the form B_1 are quite common – examples include rules for subtyping and intersection types:

$$\textsc{Sub}\ \frac{\Gamma \vdash e : \tau}{\Gamma \vdash e : \tau'} \quad \text{provided } \tau <: \tau'. \qquad \textsc{Inter}\ \frac{\Gamma \vdash e : \tau \qquad \Gamma \vdash e : \tau'}{\Gamma \vdash e : \tau \wedge \tau'}$$

There is no golden recipe for adapting proof systems to make them amenable to freefinement. However, enriching specifications and/or terms might help. The Hoare logic example used enriched specifications to keep track of write and read sets. Consider again the two problematic rules from before:

$$2\ \frac{\text{succ(n)} : \mathbb{N}}{\text{pred(succ(n))} : \mathbb{N}} \qquad 3\ \frac{\text{n} : \mathbb{N}}{\text{pred(n)} : \mathbb{N}} \quad \text{provided positive(n).}$$

Rule 2 can be accommodated by choosing $\mathbb{S} = \{\text{'z'}, \text{'s'}, \text{'p'}\} \times \{\mathbb{N}\}$. Intuitively, the specification ('s', $\mathbb{N}$) tracks the fact that the outermost constructor is 'succ'. The rule then becomes:

$$2\ \frac{\text{n} : (\text{'s'}, \mathbb{N})}{\text{pred(n)} : (\text{'p'}, \mathbb{N})}$$

Rule 3 can be accommodated by choosing $\mathbb{S} = \mathbb{N} \times \{\mathbb{N}\}$. Then the sentence n : $(i, \mathbb{N})$ tracks the fact that term n denotes the natural number i. The adapted rule is of the form A_1 with $n = 1$:

$$3\ \frac{\text{n} : (i, \mathbb{N})}{\text{pred(n)} : (i-1, \mathbb{N})} \quad \text{provided } i > 0.$$

In some cases it might be useful to enrich the term language. For example, consider the rule of concurrent separation logic [3] that removes auxiliary commands (ghost assignments):

$$\textsc{Auxiliary}\ \frac{\Gamma \vdash \{P\}c\{Q\}}{\Gamma \vdash \{P\}c\backslash a\{Q\}} \quad \text{provided a} \in \mathit{aux}(c) \text{ and a} \cap (\mathit{FV}(P) \cup \mathit{FV}(Q)) = \emptyset.$$

This rule is not of the form A_1 or B_1, because it contains a meta-operation in the conclusion. However, if the meta-operation is turned into an explicit constructor (and specifications track auxiliaries), then the rule is of the form B_1 with $m = 1$ and freefinement can handle it.

To get an approximate idea of what will happen when freefinement is applied to a separation logic, consider the frame and concurrency rules:

$$\textsc{Frame}\ \frac{\{P\}c\{Q\}}{\{P * R\}c\{Q * R\}}$$

$$\textsc{Concurrency}\ \frac{\{P_1\}c_1\{Q_1\} \qquad \{P_2\}c_2\{Q_2\}}{\{P_1 * P_2\}c_1 || c_2\{Q_1 * Q_2\}}$$

A concrete setting and system will typically make syntactic restrictions on the commands in the triples. So the specification statement {P, Q} might contain more components, but freefinement will yield refinement versions of the rules that look roughly as follows:

$$\textsc{Frame}\ \frac{}{\{P * R, Q * R\} \sqsubseteq \{P, Q\}}$$

$$\textsc{Concurrency}\ \frac{}{\{P_1 * P_2, Q_1 * Q_2\} \sqsubseteq \{P_1, Q_1\} \,||\, \{P_2, Q_2\}}$$

4. Related Work

In his work on refinement for the lambda calculus, Denney [4] treats types as rudimentary specifications and introduces a specification construct $?_\tau$ for each type τ. Conceptually, $?_\tau$ corresponds to $[\Gamma; \tau]$ where the context Γ is left implicit. For example, consider the term $\lambda x : \sigma.\, ?_\tau$ in the context Γ. The $?_\tau$ inside the term corresponds to $[\Gamma, x : \sigma; \tau]$. Denney also considers richer specifications for lambda terms in his PhD thesis [5]. This results in a more

powerful refinement calculus in which specification constructs can contain logical assumptions.

The specification statement $\overline{x} : [P, Q]$ of Morgan [8] is analogous to $\overline{x}; \mathrm{Var} : \{P, Q\}$, since there is no restriction on the variables that the statement may read. However, his specification statement is a total correctness specification, and the accompanying refinement calculus [9] establishes total correctness. Similar refinement calculi for total correctness were proposed by Back [1, 2], Morris [10] and Hehner [6]. The books [2, 6, 9] contain many examples of how correct algorithms can be constructed from their specifications via refinement.

Leino and Manohar [7] consider the join of Morgan's specification statements $\overline{x} : [P_1, Q_1]$ and $\overline{x} : [P_2, Q_2]$, and mention several of its uses. Freefinement adds explicit constructors for joins, and relies on the ability to join arbitrary terms from U in order to establish harmony.

There is a relationship between observational equivalence of terms and the function Specs, because $\models_{V_1} \_ \mathit{Sat} \_$ gives rise to a notion of observability from the specification point of view. In particular, two terms t and t' are observationally equivalent in this sense iff $t \sim t'$, where

$$t \sim t' \stackrel{\text{def}}{=} \mathrm{Specs}(\{t\}) = \mathrm{Specs}(\{t'\})$$

It is trivial to check that $\sim$ is an equivalence relation. If $\models_{V_1} \_ \mathit{Sat} \_$ is well-behaved, then $t \sim t' \Leftrightarrow [\![t]\!] = [\![t']\!]$ (i.e. $t \sim t' \Leftrightarrow t \equiv t'$) by Corollary 2.9 in the Appendix and Theorem 6.1.

Acknowledgments

Van Staden was supported by ETH Research Grant ETH-15 10-1. Calcagno was partially funded by EPSRC.

References

[1] R.-J. Back. Correctness preserving program refinements: Proof theory and applications. *Mathematical Centre Tracts*, 131, 1980.

[2] R.-J. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998. Graduate Texts in Computer Science.

[3] S. Brookes. A semantics for concurrent separation logic. *Theor. Comput. Sci.*, 375:227–270, April 2007.

[4] E. Denney. Simply-typed underdeterminism. *Journal of Computer Science and Technology*, 13:491–508, 1998.

[5] E. Denney. A theory of program refinement. Technical Report ECS-LFCS-99-412, University of Edinburgh, 1999.

[6] E. C. R. Hehner. *A practical theory of programming*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[7] K. R. M. Leino and R. Manohar. Joining specification statements. *Theor. Comput. Sci.*, 216(1-2):375–394, 1999.

[8] C. Morgan. The specification statement. *ACM Trans. Program. Lang. Syst.*, 10:403–419, July 1988.

[9] C. Morgan. *Programming from specifications (2nd ed.)*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, 1994.

[10] J. M. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Sci. Comput. Program.*, 9:287–306, December 1987.

[11] M. J. Parkinson and G. M. Bierman. Separation logic, abstraction and inheritance. In *POPL '08*, pages 75–86, New York, NY, USA, 2008. ACM.

A. Antitone Galois Connections

Lemma 1 established that an antitone Galois connection exists between the functions Specs and Terms:

$$X \subseteq \mathrm{Terms}(Y) \Leftrightarrow Y \subseteq \mathrm{Specs}(X) \qquad (*)$$

Theorems derived from this equivalence come in pairs because of the symmetry between Specs and Terms. Here are a few well-known ones together with their proofs:

Corollary 2.

2.1 $X \subseteq \mathrm{Terms}(\mathrm{Specs}(X))$
2.2 $Y \subseteq \mathrm{Specs}(\mathrm{Terms}(Y))$
2.3 $X \subseteq X' \Rightarrow \mathrm{Specs}(X) \supseteq \mathrm{Specs}(X')$
2.4 $Y \subseteq Y' \Rightarrow \mathrm{Terms}(Y) \supseteq \mathrm{Terms}(Y')$
2.5 $X \subseteq X' \Rightarrow \mathrm{Terms}(\mathrm{Specs}(X)) \subseteq \mathrm{Terms}(\mathrm{Specs}(X'))$
2.6 $Y \subseteq Y' \Rightarrow \mathrm{Specs}(\mathrm{Terms}(Y)) \subseteq \mathrm{Specs}(\mathrm{Terms}(Y'))$
2.7 $\mathrm{Specs}(\mathrm{Terms}(\mathrm{Specs}(X))) = \mathrm{Specs}(X)$
2.8 $\mathrm{Terms}(\mathrm{Specs}(\mathrm{Terms}(Y))) = \mathrm{Terms}(Y)$
2.9 $\mathrm{Specs}(X) \subseteq \mathrm{Specs}(X')$
$\Leftrightarrow \mathrm{Terms}(\mathrm{Specs}(X)) \supseteq \mathrm{Terms}(\mathrm{Specs}(X'))$
2.10 $\mathrm{Terms}(Y) \subseteq \mathrm{Terms}(Y')$
$\Leftrightarrow \mathrm{Specs}(\mathrm{Terms}(Y)) \supseteq \mathrm{Specs}(\mathrm{Terms}(Y'))$
2.11 $\mathrm{Specs}(X \cup X') = \mathrm{Specs}(X) \cap \mathrm{Specs}(X')$
2.12 $\mathrm{Terms}(Y \cup Y') = \mathrm{Terms}(Y) \cap \mathrm{Terms}(Y')$

Proof.

2.1 In $(*)$, instantiate Y with Specs(X).
2.3 $X \subseteq$ "Assumption" $X' \subseteq$ "2.1" $\mathrm{Terms}(\mathrm{Specs}(X'))$. In $(*)$, instantiate Y with $\mathrm{Specs}(X')$.
2.5 If $X \subseteq X'$, then $\mathrm{Specs}(X) \supseteq \mathrm{Specs}(X')$ holds by 2.3. The result follows from 2.4.
2.7 From 2.1 and 2.3 follows $\mathrm{Specs}(X) \supseteq \mathrm{Specs}(\mathrm{Terms}(\mathrm{Specs}(X)))$. Instantiating Y with Specs(X) in 2.2 yields $\mathrm{Specs}(X) \subseteq \mathrm{Specs}(\mathrm{Terms}(\mathrm{Specs}(X)))$.
2.9 $\Rightarrow$ holds by 2.4. From $\mathrm{Terms}(\mathrm{Specs}(X)) \supseteq \mathrm{Terms}(\mathrm{Specs}(X'))$ and 2.3, $\mathrm{Specs}(\mathrm{Terms}(\mathrm{Specs}(X))) \subseteq \mathrm{Specs}(\mathrm{Terms}(\mathrm{Specs}(X')))$. $\mathrm{Specs}(X) \subseteq \mathrm{Specs}(X')$ by 2.7.
2.11 Proof by indirect equality. For arbitrary Y:

$$\begin{array}{ll} & Y \subseteq \mathrm{Specs}(X \cup X') \\ \Leftrightarrow & \{\text{By } (*)\} \\ & X \cup X' \subseteq \mathrm{Terms}(Y) \\ \Leftrightarrow & \{\text{Set theory}\} \\ & X \subseteq \mathrm{Terms}(Y) \ \wedge \ X' \subseteq \mathrm{Terms}(Y) \\ \Leftrightarrow & \{\text{By } (*)\} \\ & Y \subseteq \mathrm{Specs}(X) \ \wedge \ Y \subseteq \mathrm{Specs}(X') \\ \Leftrightarrow & \{\text{Set theory}\} \\ & Y \subseteq \mathrm{Specs}(X) \cap \mathrm{Specs}(X') \end{array}$$

□