Hilbert-Post completeness for the state and the exception effects

Jean-Guillaume Dumas, Dominique Duval, Burak Ekici, Damien Pous, Jean-Claude Reynaud

Hilbert-Post completeness for the state and the exception effects

Jean-Guillaume Dumas* Dominique Duval* Burak Ekici*
Damien Pous† Jean-Claude Reynaud‡

March 2, 2015

Abstract

In this paper, we present a novel framework for studying the syntactic completeness of computational effects and we apply it to the exception effect. When applied to the states effect, our framework can be seen as a generalization of Pretnar's work on this subject. We first introduce a relative notion of Hilbert-Post completeness, well-suited to the composition of effects. Then we prove that the exception effect is relatively Hilbert-Post complete, as well as the "core" language which may be used for implementing it; these proofs have been formalized and checked with the proof assistant Coq.

1 Introduction

In order to add reasoning capabilities to computer algebra systems one has to be able to deal with various programming languages, including languages which involve computational effects. For instance at CICM Workshops 2014 we presented a method for certified proofs in programs involving exceptions, with its implementation in Coq and an application to exact linear algebra. A major difficulty for reasoning about programs involving computational effects is that their syntax does not look like their interpretation: typically, a piece of program with arguments in X that returns a value in Y is not interpreted as a function from X to Y, because of the effects. The best-known algebraic approach of this problem has been initiated by Moggi and implemented in Haskell; it focuses on the case where the interpretation is a function from X to T(Y) for a monad T [10]. Monads have then been extended to Lawvere theories and

*Laboratoire J. Kuntzmann, Université de Grenoble. 51, rue des Mathématiques, umr CNRS 5224, bp 53X, F38041 Grenoble, France, {Jean-Guillaume.Dumas,Dominique.Duval,Burak.Ekici}@imag.fr.

†Plume team, CNRS, ENS Lyon, Université de Lyon, INRIA, UMR 5668, France, Damien.Pous@ens-lyon.fr.

‡Reynaud Consulting (RC), Jean-Claude.Reynaud@imag.fr.

algebraic handlers [12, 13] while other approaches include effect systems [9], or Hoare logic [1]. Following this line, completeness results have been obtained for (global) states [11] and for local states [14]. We instead mix effect systems and algebraic theories by adding *decorations* to terms and equations for staying close to the syntax while reasoning with effects. Such decorated logical systems have been designed for the state (variables and memory) and exception (throwing and handling) effects and recently implemented in Coq [2, 5, 6]. They have been built so as to be sound with respect to their intended interpretation, but little is known about their completeness.

Then, Hilbert-Post completeness (also called Post completeness) is a syntactic notion of completeness which does not use any notion of negation, so that it is well-suited for equational logic. In a given logic L, we call *theory* a set of sentences which is deductively closed: everything you can derive from it (using the rules of L) is already in it. A theory is *(Hilbert-Post) consistent* if it does not contain all sentences, and it is *(Hilbert-Post) complete* if it is consistent and if any sentence which is added to it generates an inconsistent theory [15, Def. 4].

In this paper, we first introduce a *relative* notion of Hilbert-Post completeness in a logic L with respect to a sublogic L_0; it is still a syntactic notion of completeness using no negation, but it allows to express more flexible properties than the usual (or *absolute*) Hilbert-Post completeness. It can be proved that the decorated theory for global states is relatively Hilbert-Post complete. This proof is presented in Appendix A. It follows the same lines as the Hilbert-Post completeness proof for global states in [11]. In this paper, we prove a novel result: the decorated theory for exceptions is relatively Hilbert-Post complete, and moreover this property holds also for the *core* theory which may be used for implementing exceptions, as described in [5]. Our completeness proofs are mainly based on the approach given in [11]: first canonical forms are highlighted for syntactical terms and, second, theories are restricted to these forms so that the proofs follow. All the completeness proofs have been verified with the Coq proof assistant, using the framework introduced in [6] for the state effect.

Thus, in Section 2 we define the relative Hilbert-Post completeness. Then, using the relevant decorated logics, we prove the relative Hilbert-Post completeness of the theory of exceptions in Section 3 and of the corresponding core theory in Section 4. We conclude with a short description of the implementation of the proofs in Coq. The relative Hilbert-Post completeness of the decorated theory of states is proved in Appendix A.

2 Relative Hilbert-Post completeness

Each logic in this paper comes with a *language*, which is a set of *formulas*, and with *deduction rules*. Deduction rules are used for deriving (or generating) *theorems*, which are some formulas, from some chosen formulas called *axioms*. A *theory* T is a set of theorems which is *deductively closed*, in the sense that every theorem which can be derived from T using the rules of the logic is already in

T. We describe a set-theoretic *intended model* for each logic we introduce; the rules of the logic are designed so as to be *sound* with respect to this intended model. Given a logic L, the theories of L are partially ordered by inclusion. There is a maximal theory T_{max}, where all formulas are theorems. There is a minimal theory T_{min}, which is generated by the empty set of axioms. For all theories T and T', we denote by $T + T'$ the theory generated from T and T'.

Example 2.1. With this point of view there are many different *equational logics*, with the same deduction rules but with different languages, depending on the definition of *terms*. In an equational logic, formulas are *pairs of parallel terms* $(f, g) : X \to Y$ and theorems are *equations* $f \equiv g : X \to Y$. Typically, the language of an equational logic may be defined from a *signature* (made of sorts and operations). The deduction rules are such that the equations in a theory form a *congruence*, i.e., an equivalence relation compatible with the structure of the terms. For instance, we may consider the logic "of naturals" L_{nat}, with its language generated from the signature made of a sort N, a constant $0 : \mathbb{1} \to N$ and an operation $s : N \to N$. For this logic, the minimal theory is the theory "of naturals" T_{nat}, the maximal theory is such that $s^k \equiv s^\ell$ and $s^k \circ 0 \equiv s^\ell \circ 0$ for all natural numbers k and ℓ, and (for instance) the theory "of naturals modulo 6" T_{mod6} can be generated from the equation $s^6 \equiv id_N$. We consider models of equational logics in sets: each type X is interpreted as a set (still denoted X), which is a singleton when X is $\mathbb{1}$, each term $f : X \to Y$ as a function from X to Y (still denoted $f : X \to Y$), and each equation as an equality of functions.

Definition 2.2. Given a logic L and its maximal theory T_{max}, a theory T is *consistent* if $T \neq T_{max}$, and it is *Hilbert-Post complete* if it is consistent and if any theory which contains T coincides with T_{max} or with T.

Example 2.3. In Example 2.1 we considered two theories for the logic L_{nat}: the theory "of naturals" T_{nat} and the theory "of naturals modulo 6" T_{mod6}. Since both are consistent and T_{mod6} contains T_{nat}, the theory T_{nat} is not Hilbert-Post complete. The unique Hilbert-Post complete theory for L_{nat} is made of all equations but $s \equiv id_N$, it can be generated from the axioms $s \circ 0 \equiv 0$ and $s \circ s \equiv s$.

If a logic L is an extension of a sublogic L_0, each theory T_0 of L_0 generates a theory $F(T_0)$ of L. Conversely, each theory T of L determines a theory $G(T)$ of L_0, made of the theorems of T which are formulas of L_0, so that $G(T_{max}) = T_{max,0}$. The functions F and G are monotone and they form a *Galois connection*, denoted $F \dashv G$: for each theory T of L and each theory T_0 of L_0 we have $F(T_0) \subseteq T$ if and only if $T_0 \subseteq G(T)$. It follows that $T_0 \subseteq G(F(T_0))$ and $F(G(T)) \subseteq T$.

Definition 2.4. Given a logic L_0, an extension L of L_0 and the associated Galois connection $F \dashv G$, a theory T' of L is L_0-*derivable* from a theory T of L if $T' = T + F(T'_0)$ for some theory T'_0 of L_0, and it is *relatively Hilbert-Post complete with respect to* L_0 if it is consistent and if any theory of L which contains T is L_0-derivable from T.

Each theory T is L_0-derivable from itself, because $T = T + F(T_{min,0})$, where $T_{min,0}$ is the minimal theory of L_0. In addition, Theorem 2.6 shows that relative completeness lifts the usual "absolute" completeness from L_0 to L.

Lemma 2.5. *Let us consider a logic L_0, an extension L of L_0 and the associated Galois connection $F \dashv G$. For each theory T of L, a theory T' of L is L_0-derivable from T if and only if $T' = T + F(G(T'))$. As a special case, T_{max} is L_0-derivable from T if and only if $T_{max} = T + F(T_{max,0})$. A theory T of L is relatively Hilbert-Post complete with respect to L_0 if and only if it is consistent and every theory T' of L which contains T is such that $T' = T + F(G(T'))$.*

Proof. Clearly, if $T' = T + F(G(T'))$ then T' is L_0-derivable from T. So, let T'_0 be a theory of L_0 such that $T' = T + F(T'_0)$, and let us prove that $T' = T + F(G(T'))$. For each theory T' we know that $F(G(T')) \subseteq T'$; since here $T \subseteq T'$ we get $T + F(G(T')) \subseteq T'$. Conversely, for each theory T'_0 we know that $T'_0 \subseteq G(F(T'_0))$ and that $G(F(T'_0)) \subseteq G(T) + G(F(T'_0)) \subseteq G(T + F(T'_0))$, so that $T'_0 \subseteq G(T + F(T'_0))$; since here $T' = T + F(T'_0)$ we get first $T'_0 \subseteq G(T')$ and then $T' \subseteq T + F(G(T'))$. Then, the result for T_{max} comes from the fact that $G(T_{max}) = T_{max,0}$. The last point follows immediately. $\square$

Theorem 2.6. *Let us consider a logic L_0, an extension L of L_0 and the associated Galois connection $F \dashv G$. Let T_0 be a theory of L_0 and $T = F(T_0)$. If T_0 is Hilbert-Post complete (in L_0) and T is relatively Hilbert-Post complete with respect to L_0, then T is Hilbert-Post complete (in L).*

Proof. Since T is relatively complete with respect to L_0, it is consistent. Since $T = F(T_0)$ we have $T_0 \subseteq G(T)$. Let T' be a theory such that $T \subseteq T'$. Since T is relatively complete with respect to L_0, by Lemma 2.5 we have $T' = T + F(T'_0)$ where $T'_0 = G(T')$. Since $T \subseteq T'$, $T_0 \subseteq G(T)$ and $T'_0 = G(T')$, we get $T_0 \subseteq T'_0$. Thus, since T_0 is complete, either $T'_0 = T_0$ or $T'_0 = T_{max,0}$; let us check that then either $T' = T$ or $T' = T_{max}$. If $T'_0 = T_0$ then $F(T'_0) = F(T_0) = T$, so that $T' = T + F(T'_0) = T$. If $T'_0 = T_{max,0}$ then $F(T'_0) = F(T_{max,0})$; since T is relatively complete with respect to L_0, the theory T_{max} is L_0-derivable from T, which implies (by Lemma 2.5) that $T_{max} = T + F(T_{max,0}) = T'$. $\square$

Proposition 2.7 provides a characterization of relative Hilbert-Post completeness which will be used in the next Sections. In a given logic L, we denote $Th(E)$ the theory generated by a set E of formulas and we say that two sets of formulas E_1, E_2 are *equivalent with respect to* a theory T when $T + Th(E_1) = T + Th(E_2)$.

Proposition 2.7. *Let us consider a logic L_0, an extension L of L_0 and the associated Galois connection $F \dashv G$. Let T be a theory of L such that for each formula e in L there is a set E_0 of formulas of L_0 equivalent to $\{e\}$ with respect to T. Then each theory T' of L which contains T is L_0-derivable from T. Thus, if T is consistent then it is relatively Hilbert-Post complete with respect to L_0.*

Proof. Let T' be a theory which contains T. Let $T'' = T + F(G(T'))$, so that $T \subseteq T'' \subseteq T'$ (because $F(G(T')) \subseteq T'$ for any T'). Let us consider an arbitrary formula e in T', by assumption there is a set E_0 of formulas of L_0 such that $T + Th(\{e\}) = T + Th(E_0)$. Since e is in T' and $T \subseteq T'$ we have $T + Th(\{e\}) \subseteq T'$, so that $T + Th(E_0) \subseteq T'$. It follows that E_0 is a set of theorems of T' which are formulas of L_0, which means that $E_0 \subseteq G(T')$, and consequently $Th(E_0) \subseteq F(G(T'))$, so that $T + Th(E_0) \subseteq T''$. Since $T + Th(\{e\}) = T + Th(E_0)$ we get $e \in T''$. We have proved that $T' = T''$, so that T' is L_0-derivable from T. $\square$

3 Completeness for exceptions

Exception handling is provided by most modern programming languages. It allows to deal with anomalous or exceptional events which require special processing. E.g., one can easily and simultaneously compute dynamic evaluation in exact linear algebra using exceptions [5]. There, we proposed to deal with exceptions as a decorated effect, in order to prove properties of such programs: a term $f : X \to Y$ is not interpreted as a function $f : X \to Y$ unless it is pure. A term which may raise an exception is instead interpreted as a function $f : X \to Y + E$ where '+' is disjoint union operator and E is the set of exceptions. In this section, first following [5] we present a formalization of exceptions in a decorated setting, then we prove its relative Hilbert-Post completeness in Theorem 3.5.

As in [5], decorated logics for exceptions are obtained from equational logics by classifying terms. Terms are classified as *pure* terms or *propagators*, which is expressed by adding a *decoration* or superscript, respectively (0) or (1); decoration and type information about terms may be omitted when they are clear from the context or when they do not matter. All terms must propagate exceptions, and propagators are allowed to raise an exception while pure terms are not. The fact of catching exceptions is hidden: it is embedded into the `try/catch` construction, as explained below. In Section 4 we will consider an implementation of exceptions by a more basic language, where some terms are *catchers*, which means that they may recover form an exception, i.e., they do not have to propagate exceptions.

Let us describe informally a decorated theory for exceptions and its intended model. Each type X is interpreted as a set, still denoted X. The intended model is described with respect to a set E called the *set of exceptions*, which does not appear in the syntax. A pure term $u^{(0)} : X \to Y$ is interpreted as a function $u : X \to Y$ and a propagator $a^{(1)} : X \to Y$ as a function $a : X \to Y + E$; equations are interpreted as equalities of functions. There is an obvious conversion from pure terms to propagators, which allow to consider all terms as propagators whenever needed; if a propagator $a^{(1)} : X \to Y$ "is" a pure term, in the sense that it has been obtained by conversion from a pure term, then the function $a : X \to Y + E$ is such that $a(x) \in Y$ for each $x \in X$. The composition of propagators is the Kleisli composition associated to the monad $X + E$, which simply means that exceptions are always propagated: the interpretation of $(b \circ a)^{(1)} : X \to Z$ where $a^{(1)} : X \to Y$ and $b^{(1)} : Y \to Z$ is such that $(b \circ a)(x) = b(a(x))$ when $a(x)$ is not an exception and $(b \circ a)(x) = e$ when $a(x)$ is the exception e. Exceptions may be classified according to their *name*, as in [5]. Here, in order to focus on the main features of the proof of completeness, we assume that there is only one exception name. Each exception is built by *encapsulating* a parameter. Let P denote the type of parameters for exceptions. The fundamental operations for raising exceptions are the propagators $\mathtt{throw}_Y^{(1)} : P \to Y$ for each type Y: this operation throws an exception with a parameter p of type P and pretends that this exception has type Y. The interpretation of the term $\mathtt{throw}_Y^{(1)} : P \to Y$ is a function $\mathtt{throw}_Y : P \to Y + E$ such that $\mathtt{throw}_Y(p) \in E$ for each $p \in P$. The fundamental operations for handling exceptions are the propagators $(\mathtt{try}(a)\mathtt{catch}(b))^{(1)} : X \to Y$ for each terms $a : X \to Y$ and $b : P \to Y$: this operation first runs a until an exception with parameter p is raised (if any), then, if such an exception has been raised, it runs $b(p)$. The interpretation of the term $(\mathtt{try}(a)\mathtt{catch}(b))^{(1)} : X \to Y$ is a function $\mathtt{try}(a)\mathtt{catch}(b) : X \to Y + E$ such that $(\mathtt{try}(a)\mathtt{catch}(b))(x) = a(x)$ when a is pure and $(\mathtt{try}(a)\mathtt{catch}(b))(x) = b(p)$ when $a(x)$ throws an exception with parameter p.

More precisely, the *decorated logic for exceptions* L_{exc} is defined in Fig. 1 (next page). The *pure sublogic* $L_{exc}^{(0)}$, for dealing with pure terms, may be any logic which extends a *monadic equational logic* L_{eq}. A monadic equational logic is made of types, terms and operations, where all operations are unary and terms are simply paths. For instance, $L_{exc}^{(0)}$ may be an equational logic, with n-ary operations for arbitrary n. However, the rules for L_{exc} do not allow to form tuples of decorated terms, so that the term $\mathtt{op}(f, g)$ (where $\mathtt{op}$ is a pure operation of arity 2) is not well-formed, unless f and g are pure. It is well known that there is no "canonical" interpretation for such terms; however, the interpretation where f is runned before g can be formalized thanks to strong monads [10] or sequential products [4]. In this paper, in order to focus on completeness issues, we avoid such situations.

This pure sublogic $L_{exc}^{(0)}$ is extended to form the corresponding *decorated logic for exceptions* L_{exc} by applying the rules in Fig. 1, with the following intended meanings:

- (propagate) for each $a^{(1)} : X \to Y$, $a \circ \mathtt{throw}_X \equiv \mathtt{throw}_Y$: exceptions are always propagated.
- (recover) $\mathtt{throw}_Y$ is a monomorphism with respect to pure terms, for each Y: the parameter used for throwing an exception may be recovered.
- (try_0) for each $u^{(0)} : X \to Y$ and $b^{(1)} : P \to Y$, $\mathtt{try}(u)\mathtt{catch}(b) \equiv u$: pure code inside the `try` part never triggers the code inside the `catch` part.
- (try_1) for each $u^{(0)} : X \to P$ and $b^{(1)} : P \to Y$, $\mathtt{try}(\mathtt{throw}_Y \circ u)\mathtt{catch}(b) \equiv b \circ u$: code inside the `catch` part is executed as soon as an exception is thrown inside the `try` part.

The *theory of exceptions* T_{exc} is the theory of L_{exc} generated from some chosen theory $T^{(0)}$ of $L_{exc}^{(0)}$; with the notations of Section 2, $T_{exc} = F(T^{(0)})$. The soundness of the intended model follows, see, e.g., [5, §5.1] and [3], with the description of the handling of exceptions in Java, see for instance [8, Ch. 14], or in C++ [7, §15]. Now, in order to prove the completeness of the decorated

Monadic equational logic L_{eq}:

Types: $X, Y, \ldots$. Terms: $f : X \to Y, \ldots$ closed by composition:

$f_k \circ \cdots \circ f_1 : X_0 \to X_k$ for each $(f_i : X_{i-1} \to X_i)_{1 \le i \le k}$

with the empty path (when $k = 0$) denoted $id_X : X \to X$ for each X

Rules: (equiv) $\dfrac{f}{f \equiv f} \quad \dfrac{f \equiv g}{g \equiv f} \quad \dfrac{f \equiv g \quad g \equiv h}{f \equiv h}$

(subs) $\dfrac{f : X \to Y \quad g_1 \equiv g_2 : Y \to Z}{g_1 \circ f \equiv g_2 \circ f}$ (repl) $\dfrac{g_1 \equiv g_2 : X \to Y \quad h : Y \to Z}{h \circ g_1 \equiv h \circ g_2}$

Decorated logic for exceptions L_{exc}:

Pure part: some logic $L_{exc}^{(0)}$ extending L_{eq}, with a distinguished type P

Decorated terms: $\texttt{throw}_Y^{(1)} : P \to Y$ for each type Y,

$(\texttt{try}(a)\texttt{catch}(b))^{(1)} : X \to Y$ for each $a^{(1)} : X \to Y$ and $b^{(1)} : P \to Y$, and

$(f_k \circ \cdots \circ f_1)^{(\max(d_1, \ldots, d_k))} : X_0 \to X_k$ for each $(f_i^{(d_i)} : X_{i-1} \to X_i)_{1 \le i \le k}$

conversions from $f^{(0)} : X \to Y$ to $f^{(1)} : X \to Y$

Rules:

(equiv), (subs), (repl) for all decorations

(propagate) $\dfrac{a^{(1)} : X \to Y}{a \circ \texttt{throw}_X \equiv \texttt{throw}_Y}$

(recover) $\dfrac{u_1^{(0)}, u_2^{(0)} : X \to P \quad \texttt{throw}_Y \circ u_1 \equiv \texttt{throw}_Y \circ u_2}{u_1 \equiv u_2}$

(try_0) $\dfrac{u^{(0)} : X \to Y \quad b^{(1)} : P \to Y}{\texttt{try}(u)\texttt{catch}(b) \equiv u}$

(try_1) $\dfrac{u^{(0)} : X \to P \quad b^{(1)} : P \to Y}{\texttt{try}(\texttt{throw}_Y \circ u)\texttt{catch}(b) \equiv b \circ u}$

Figure 1: Logic for exceptions

theory for exceptions under suitable assumptions, we first determine canonical forms and then we study the equations between terms in canonical forms.

Proposition 3.1. *For each term $a^{(1)} : X \to Y$, either there exists some pure term $u^{(0)} : X \to Y$ such that $a \equiv u$ or there exists some pure term $u^{(0)} : X \to P$ such that $a \equiv \texttt{throw}_Y \circ u$.*

Proof. The proof proceeds by structural induction. If a is pure the result is obvious, otherwise a can be written in a unique way as $a = b \circ \texttt{op} \circ v$ where v is pure, $\texttt{op}$ is either $\texttt{throw}_Z$ for some Z or $\texttt{try}(c)\texttt{catch}(d)$ for some c and d, and b is the remaining part of a.

- If $a = b^{(1)} \circ \texttt{throw}_Z \circ v^{(0)}$, then by (propagate) $a \equiv \texttt{throw}_Y \circ v^{(0)}$.

- If $a = b^{(1)} \circ (\mathtt{try}(c^{(1)})\mathtt{catch}(d^{(1)})) \circ v^{(0)}$, then by induction we consider two subcases.
  - If $c \equiv w^{(0)}$ then by (try_0) $a \equiv b^{(1)} \circ w^{(0)} \circ v^{(0)}$ and by induction we consider two subcases: if $b \equiv t^{(0)}$ then $a \equiv (t \circ w \circ v)^{(0)}$ and if $b \equiv \mathtt{throw}_Y \circ t^{(0)}$ then $a \equiv \mathtt{throw}_Y \circ (t \circ w \circ v)^{(0)}$.
  - If $c \equiv \mathtt{throw}_Z \circ w^{(0)}$ then by (try_1) $a \equiv b^{(1)} \circ d^{(1)} \circ w^{(0)} \circ v^{(0)}$ and by induction we consider two subcases: if $b \circ d \equiv t^{(0)}$ then $a \equiv (t \circ w \circ v)^{(0)}$ and if $b \circ d \equiv \mathtt{throw}_Y \circ t^{(0)}$ then $a \equiv \mathtt{throw}_Y \circ (t \circ w \circ v)^{(0)}$.

□

Thanks to Proposition 3.1, in order to study equations in the logic L_{exc} we may restrict our study to pure terms and to propagators of the form $\mathtt{throw}_Y \circ v$ where v is pure. In order to express the distinction between exceptions and non-exceptions we need some kind of "booleans". In this equational setting without negations, this is obtained by introducing a type $\mathbb{B}$ with two constants `true` and `false` such that the equation $\mathtt{true} \equiv \mathtt{false}$ corresponds to the logical contradiction '$\perp$', in the sense that it makes everything collapse: the theory generated by the equation $\mathtt{true} \equiv \mathtt{false}$ is the maximal theory.

Definition 3.2. A type $\mathbb{1}$ is a *unit* if for each type X there is a pure term $\langle\ \rangle_X^{(0)} : X \to \mathbb{1}$ and every pure term $u^{(0)} : X \to \mathbb{1}$ is such that $u \equiv \langle\ \rangle_X$. If there is a unit type $\mathbb{1}$, a type $\mathbb{B}$ is a *boolean* type if there are pure terms $\mathtt{true}^{(0)}, \mathtt{false}^{(0)} : \mathbb{1} \to \mathbb{B}$ such that whenever $\mathtt{true} \equiv \mathtt{false}$ we have $a_1 \equiv a_2$ for each pair of parallel terms (a_1, a_2).

Proposition 3.3. *1. For all $v_1^{(0)}, v_2^{(0)} : X \to P$ let $a_1^{(1)} = \mathtt{throw}_Y \circ v_1 : X \to Y$ and $a_2^{(1)} = \mathtt{throw}_Y \circ v_2 : X \to Y$. Then $a_1^{(1)} \equiv a_2^{(1)} \iff v_1^{(0)} \equiv v_2^{(0)}$.*

2. Let us assume that there is a unit type $\mathbb{1}$ and a boolean type $\mathbb{B}$ in the sense of Definition 3.2 and that $\langle\ \rangle_X^{(0)}$ is an epimorphism with respect to pure terms. For all $v_1^{(0)} : X \to P$ and $v_2^{(0)} : X \to Y$, let $a_1^{(1)} = \mathtt{throw}_Y \circ v_1 : X \to Y$. Then $a_1^{(1)} \equiv v_2^{(0)} \iff \mathtt{true}^{(0)} \equiv \mathtt{false}^{(0)}$.

Proof. 1. Clearly, if $v_1 \equiv v_2$ then $a_1 \equiv a_2$. Conversely, if $a_1 \equiv a_2$, i.e., if $\mathtt{throw}_Y \circ v_1 \equiv \mathtt{throw}_Y \circ v_2$, then by rule (recover) it follows that $v_1 \equiv v_2$.

2. If $\mathtt{true} \equiv \mathtt{false}$ then according to the definition of a boolean type we have $a_1 \equiv v_2$. Conversely, if $a_1 \equiv v_2$, then $\mathtt{true} \circ \langle\ \rangle_Y \circ a_1 \equiv \mathtt{true} \circ \langle\ \rangle_Y \circ v_2 : X \to \mathbb{B}$, where $\mathtt{true} \circ \langle\ \rangle_Y \circ a_1 = \mathtt{true} \circ \langle\ \rangle_Y \circ \mathtt{throw}_Y \circ v_1 \equiv \mathtt{throw}_{\mathbb{B}} \circ v_1$ by rule (propagate) and $\mathtt{true} \circ \langle\ \rangle_Y \circ v_2 \equiv \mathtt{true} \circ \langle\ \rangle_X$, so that we get $\mathtt{throw}_{\mathbb{B}} \circ v_1 \equiv \mathtt{true} \circ \langle\ \rangle_X$. Let $b = \mathtt{false} \circ \langle\ \rangle_P : P \to \mathbb{B}$ then we get $\mathtt{try}(\mathtt{throw}_{\mathbb{B}} \circ v_1)\mathtt{catch}(b) \equiv \mathtt{try}(\mathtt{true} \circ \langle\ \rangle_X)\mathtt{catch}(b)$, where $\mathtt{try}(\mathtt{throw}_{\mathbb{B}} \circ v_1)\mathtt{catch}(b) \equiv b \circ v_1 \equiv \mathtt{false} \circ \langle\ \rangle_P \circ v_1 \equiv \mathtt{false} \circ \langle\ \rangle_X$ by (try_1) and $\mathtt{try}(\mathtt{true} \circ \langle\ \rangle_X)\mathtt{catch}(b) \equiv \mathtt{true} \circ \langle\ \rangle_X$ by (try_0). Thus, we obtain $\mathtt{false} \circ \langle\ \rangle_X \equiv \mathtt{true} \circ \langle\ \rangle_X$, and since $\langle\ \rangle_X$ is an epimorphism with respect to pure terms this implies $\mathtt{true} \equiv \mathtt{false}$.

$\square$

Remark 3.4. The assumption that $\langle\ \rangle_X^{(0)}$ is an epimorphism with respect to pure terms in Point 2 of Proposition 3.3 cannot be satisfied when the interpretation of X is the empty set. Thus, we have to handle the empty type in a specific way. In the decorated logic for exceptions, an *empty type* is defined as a type $\mathbb{0}$ such that for each Y there is a pure term $[\ ]_Y^{(0)} : \mathbb{0} \to Y$ such that $f \equiv [\ ]_Y$ for each term $f : \mathbb{0} \to Y$ (which may be a propagator). This definition is sound with respect to the intended model: it means that $\mathbb{0}$ is interpreted as the empty set.

Theorem 3.5. *If there is a unit type $\mathbb{1}$ and a boolean type $\mathbb{B}$ in the sense of Definition 3.2 and if $\langle\ \rangle_X^{(0)}$ is an epimorphism with respect to pure terms for each non-empty type X, the theory of exceptions T_{exc} is relatively Hilbert-Post complete with respect to the pure sublogic $L_{exc}^{(0)}$ of L_{exc}.*

Proof. The proof relies upon Propositions 3.1, 3.3 and 2.7. The theory T_{exc} is consistent: it cannot be proved that $\mathtt{throw}_P^{(1)} \equiv id_P^{(0)}$ because the logic L_{exc} is sound with respect to its intended model and the interpretation of this equation in the intended model is false: indeed, $\mathtt{throw}_P(p) \in E$ for each $p \in P$, and since $P + E$ is a disjoint union we have $\mathtt{throw}_P(p) \neq p$. Now, let us consider an equation between terms with domain X and let us prove that it is T_{exc}-equivalent to a set of pure equations (i.e., equations between pure terms). We distinguish two cases, whether X is empty or not. When X is non-empty, then $\langle\ \rangle_X$ is an epimorphism with respect to pure terms. Thus, Propositions 3.1 and 3.3 prove that the given equation is T_{exc}-equivalent to a set of pure equations. When X is empty, then all terms from X to Y are equivalent to $[\ ]_Y$ (see Remark 3.4), so that the given equation is T_{exc}-equivalent to the empty set of pure equations. Thus, in both cases the result follows from Proposition 2.7. $\square$

4 Completeness of the core language for exceptions

In this section, first following [5] we describe an implementation of the language for exceptions from Section 3 using a *core* language, then we prove the relative Hilbert-Post completeness of this core language in Theorem 4.5.

Let us call the usual language for exceptions with `throw` and `try/catch`, as described in Section 3, the *programmers' language* for exceptions. The documentation on the behaviour of exceptions in many languages (for instance in java [8]) makes use of a *core language* for exceptions which is studied in [5]. In this language, the empty type plays an important role and the fundamental operations for dealing with exceptions are $\mathtt{tag}^{(1)} : P \to \mathbb{0}$ for encapsulating a parameter inside an exception and $\mathtt{untag}^{(2)} : \mathbb{0} \to P$ for recovering its parameter from any given exception. The new decoration (2) corresponds to *catchers*: a catcher may recover from an exception, it does not have to propagate it. Moreover, the equations also are decorated: in addition to the equations '$\equiv$' as in Section 3, now called *strong equations*, there are *weak equations* denoted '$\sim$'.

As in Section 3, a set E of exceptions is chosen; the intended model interprets each type X as a set X, each pure term $u^{(0)} : X \to Y$ as a function $u : X \to Y$, each propagator $a^{(1)} : X \to Y$ as a function $a : X \to Y + E$ and each catcher $f^{(2)} : X \to Y$ as a function $f : X + E \to Y + E$. There is an obvious conversion from propagators to catchers; the interpretation of the composition of catchers is straightforward, and it is compatible with the Kleisli composition for the composition of propagators. Weak and strong equations coincide on propagators, where they are interpreted as equalities, but they differ on catchers: $f^{(2)} \sim g^{(2)} : X \to Y$ means that the functions $f, g : X+E \to Y+E$ coincide on X, but maybe not on E. The interpretation of $\mathtt{tag}^{(1)} : P \to \mathbb{0}$ is a function $\mathtt{tag} : P \to E$ and the interpretation of $\mathtt{untag}^{(2)} : \mathbb{0} \to P$ is the function $\mathtt{untag} : E \to P + E$ such that $\mathtt{untag}(\mathtt{tag}(p)) = p$ for each parameter p. Thus, the fundamental axiom relating $\mathtt{tag}^{(1)}$ and $\mathtt{untag}^{(2)}$ is the weak equation $\mathtt{untag} \circ \mathtt{tag} \sim id_P$.

More precisely, the *decorated logic for the core langage for exceptions* $L_{exc\text{-}core}$ is defined in Fig. 2. Its *pure sublogic* $L^{(0)}_{exc\text{-}core}$ may be any logic which extends a monadic equational logic with an empty type $L_{eq,\mathbb{0}}$. There is an obvious conversion from strong to weak equations ($\equiv$-to-$\sim$), and in addition strong and weak equations coincide on propagators by rule (eq$_1$). Two catchers $f_1^{(2)}, f_2^{(2)} : X \to Y$ behave in the same way on exceptions if and only if $f_1 \circ [\,]_X \equiv f_2 \circ [\,]_X : \mathbb{0} \to Y$, where $[\,]_X : \mathbb{0} \to X$ builds a term of type X from any exception. Then rule (eq$_2$) expresses the fact that weak and strong equations are related by the property that $f_1 \equiv f_2$ if and only if $f_1 \sim f_2$ and $f_1 \circ [\,]_X \equiv f_2 \circ [\,]_X$. This can also be expressed as a pair of weak equations: $f_1 \equiv f_2$ if and only if $f_1 \sim f_2$ and $f_1 \circ [\,]_X \circ \mathtt{tag} \equiv f_2 \circ [\,]_X \circ \mathtt{tag}$ by rule (eq$_3$). The *core theory of exceptions* $T_{exc\text{-}core}$ is the theory of $L_{exc\text{-}core}$ generated from some chosen theory $T^{(0)}$ of $L^{(0)}_{exc\text{-}core}$. Some easily derived properties are stated in Lemma 4.1; Point 1 will be used repeatedly.

Lemma 4.1.

1. *each $f^{(2)} : \mathbb{0} \to \mathbb{0}$ is such that $f \sim id_{\mathbb{0}}$, each $f^{(1)} : \mathbb{0} \to Y$ is such that $f \equiv [\,]_Y$, and each $f^{(1)} : \mathbb{0} \to \mathbb{0}$ is such that $f \equiv id_{\mathbb{0}}$.*

2. *The fundamental strong equation for exceptions is $\mathtt{tag} \circ \mathtt{untag} \equiv id_{\mathbb{0}}$.*

3. *For all pure terms $u_1^{(0)}, u_2^{(0)} : X \to P$, one has: $u_1 \equiv u_2 \iff \mathtt{tag} \circ u_1 \equiv \mathtt{tag} \circ u_2 \iff \mathtt{untag} \circ \mathtt{tag} \circ u_1 \equiv \mathtt{untag} \circ \mathtt{tag} \circ u_2$.*

4. *For all pure terms $u^{(0)} : X \to P$, $v^{(0)} : X \to \mathbb{0}$, one has: $u \equiv [\,]_P \circ v \iff \mathtt{tag} \circ u \equiv v$.*

Proof.

1. Clear.

2. By replacement in the axiom (ax) we get $\mathtt{tag} \circ \mathtt{untag} \circ \mathtt{tag} \sim \mathtt{tag}$; then by rule (eq$_3$) $\mathtt{tag} \circ \mathtt{untag} \sim id_{\mathbb{0}}$.

Monadic equational logic with empty type $L_{eq,\mathbb{0}}$:
Types and terms: as for monadic equational logic, plus
an empty type $\mathbb{0}$ and a term $[\,]_Y : \mathbb{0} \to Y$ for each Y

Rules: as for monadic equational logic, plus (empty) $\dfrac{f : \mathbb{0} \to Y}{f \equiv [\,]_Y}$

Decorated logic for the core language for exceptions L_{exc}:
Pure part: some logic $L^{(0)}_{exc\text{-}core}$ extending $L_{eq,\mathbb{0}}$, with a distinguished type P
Decorated terms: $\mathtt{tag}^{(1)} : P \to \mathbb{0}$, $\mathtt{untag}^{(2)} : \mathbb{0} \to P$, and
$(f_k \circ \cdots \circ f_1)^{(\max(d_1,\dots,d_k))} : X_0 \to X_k$ for each $(f_i^{(d_i)} : X_{i-1} \to X_i)_{1\le i\le k}$
with conversions from $f^{(0)}$ to $f^{(1)}$ and from $f^{(1)}$ to $f^{(2)}$
Rules:
(equiv$_\equiv$), (subs$_\equiv$), (repl$_\equiv$) for all decorations
(equiv$_\sim$), (repl$_\sim$) for all decorations, (subs$_\sim$) only when h is pure

(empty$_\sim$) $\dfrac{f : \mathbb{0} \to X}{f \sim [\,]_X}$ ($\equiv$-to-$\sim$) $\dfrac{f \equiv g}{f \sim g}$ (ax) $\dfrac{}{\mathtt{untag} \circ \mathtt{tag} \sim id_P}$

(eq$_1$) $\dfrac{f_1^{(d_1)} \sim f_2^{(d_2)}}{f_1 \equiv f_2}$ only when $d_1 \le 1$ and $d_2 \le 1$

(eq$_2$) $\dfrac{f_1, f_2 : X \to Y \quad f_1 \sim f_2 \quad f_1 \circ [\,]_X \equiv f_2 \circ [\,]_X}{f_1 \equiv f_2}$

(eq$_3$) $\dfrac{f_1, f_2 : \mathbb{0} \to X \quad f_1 \circ \mathtt{tag} \sim f_2 \circ \mathtt{tag}}{f_1 \equiv f_2}$

Figure 2: Logic for the core language for exceptions

3. Implications from left to right are clear. Conversely, if $\mathtt{untag} \circ \mathtt{tag} \circ u_1 \equiv \mathtt{untag} \circ \mathtt{tag} \circ u_2$, then using the axiom (ax) and the rule (subs$_\sim$) we get $u_1 \sim u_2$. Since u_1 and u_2 are pure this means that $u_1 \equiv u_2$.

4. First, since $\mathtt{tag} \circ [\,]_P : \mathbb{0} \to \mathbb{0}$ is a propagator we have $\mathtt{tag} \circ [\,]_P \equiv id_{\mathbb{0}}$. Now, if $u \equiv [\,]_P \circ v$ then $\mathtt{tag} \circ u \equiv \mathtt{tag} \circ [\,]_P \circ v \equiv v$. Conversely, if $\mathtt{tag} \circ u \equiv v$ then $\mathtt{tag} \circ u \equiv \mathtt{tag} \circ [\,]_P \circ v$, and by Point 3 this means that $u \equiv [\,]_P \circ v$.

□

The operation `untag` in the core language can be used for decomposing the `try/catch` construction in the programmer's language in two steps: a step for catching the exception, which is nested into a second step inside the `try/catch` block: this corresponds to an implementation of the programmer's language by the core language, as in [5], which is reminded below; then Proposition 4.2 proves the correction of this implementation. In view of this implementation we extend the core language with:

- for each $b^{(1)} : P \to Y$, a catcher $(\mathtt{CATCH}(b))^{(2)} : Y \to Y$ such that $\mathtt{CATCH}(b) \sim id_Y$ and $\mathtt{CATCH}(b) \circ [\,]_Y \equiv b \circ \mathtt{untag}$: if the argument of

$\mathtt{CATCH}(b)$ is non-exceptional then nothing is done, otherwise the parameter p of the exception is recovered and $b(p)$ is runned.

- for each $a^{(1)} : X \to Y$ and $k^{(2)} : Y \to Y$, a propagator $(\mathtt{TRY}(a,k))^{(1)} : X \to Y$ such that $\mathtt{TRY}(a,k) \sim k \circ a$: thus $\mathtt{TRY}(a,k)$ behaves as $k \circ a$ on non-exceptional arguments, but it does always propagate exceptions.

Then, an implementation of the programmer's language of exceptions by the core language is easily obtained:

- for each type Y: $\mathtt{throw}_Y^{(1)} = [\,]_Y \circ \mathtt{tag} : P \to Y$.
- for each $a^{(1)} : X \to Y$, $b^{(1)} : P \to Y$: $(\mathtt{try}(a)\mathtt{catch}(b))^{(1)} = \mathtt{TRY}(a, \mathtt{CATCH}(b))$.

Proposition 4.2. *If the pure term $[\,]_Y : \mathbb{0} \to Y$ is a monomorphism with respect to propagators for each type Y, the above implementation of the programmers' language for exceptions by the core language is correct.*

Proof. We have to prove that the images of the four basic properties of `throw` and `try`/`catch` are satisfied.

- (propagate) For each $a^{(1)} : X \to Y$, the rules of $L_{exc\text{-}core}$ imply that $a \circ [\,]_X \equiv [\,]_Y$, so that $a \circ [\,]_X \circ \mathtt{tag} \equiv [\,]_Y \circ \mathtt{tag}$.
- (recover) For each $u_1^{(0)}, u_2^{(0)} : X \to P$, if $[\,]_Y \circ \mathtt{tag} \circ u_1 \equiv [\,]_Y \circ \mathtt{tag} \circ u_2$ since $[\,]_Y$ is a monomorphism with respect to propagators we have $\mathtt{tag} \circ u_1 \equiv \mathtt{tag} \circ u_2$, so that, by Point 3 in Lemma 4.1, we get $u_1 \equiv u_2$.
- (try_0) For each $u^{(0)} : X \to Y$ and $b^{(1)} : P \to Y$, we have $\mathtt{TRY}(u, \mathtt{CATCH}(b)) \sim \mathtt{CATCH}(b) \circ u$ and $\mathtt{CATCH}(b) \circ u \sim u$ (because $\mathtt{CATCH}(b) \sim id$ and u is pure), so that $\mathtt{TRY}(u, \mathtt{CATCH}(b)) \sim u$; since both sides are propagators, we get $\mathtt{TRY}(u, \mathtt{CATCH}(b)) \equiv u$.
- (try_1) For each $u^{(0)} : X \to P$ and $b^{(1)} : P \to Y$, we have $\mathtt{TRY}([\,]_Y \circ \mathtt{tag} \circ u, \mathtt{CATCH}(b)) \sim \mathtt{CATCH}(b) \circ [\,]_Y \circ \mathtt{tag} \circ u$ and $\mathtt{CATCH}(b) \circ [\,]_Y \equiv b \circ \mathtt{untag}$ so that $\mathtt{TRY}([\,]_Y \circ \mathtt{tag} \circ u, \mathtt{CATCH}(b)) \sim b \circ \mathtt{untag} \circ \mathtt{tag} \circ u$. We have also $\mathtt{untag} \circ \mathtt{tag} \circ u \sim u$ (because $\mathtt{untag} \circ \mathtt{tag} \sim id$ and u is pure), so that $\mathtt{TRY}([\,]_Y \circ \mathtt{tag} \circ u, \mathtt{CATCH}(b)) \sim b \circ u$; since both sides are propagators, we get $\mathtt{TRY}([\,]_Y \circ \mathtt{tag} \circ u, \mathtt{CATCH}(b)) \equiv b \circ u$.

□

Now let us check that the core decorated theory for exceptions is also relatively Hilbert-Post complete, under suitable assumptions.

Proposition 4.3. *1. For each propagator $a^{(1)} : X \to Y$, either a is pure or there is a pure term $v^{(0)} : X \to P$ such that $a^{(1)} \equiv [\,]_Y^{(0)} \circ \mathtt{tag}^{(1)} \circ v^{(0)}$. And for each propagator $a^{(1)} : X \to \mathbb{0}$ (either pure or not), there is a pure term $v^{(0)} : X \to P$ such that $a^{(1)} \equiv \mathtt{tag}^{(1)} \circ v^{(0)}$.*

2. *For each catcher $f^{(2)} : X \to Y$, either f is a propagator or there is an propagator $a^{(1)} : P \to Y$ and a pure term $u^{(0)} : X \to P$ such that $f^{(2)} \equiv a^{(1)} \circ \mathtt{untag}^{(2)} \circ \mathtt{tag}^{(1)} \circ u^{(0)}$.*

Proof. 1. If the propagator $a^{(1)} : X \to Y$ is not pure then it contains at least one occurrence of $\mathtt{tag}^{(1)}$. Thus, it can be written in a unique way as $a = b \circ \mathtt{tag} \circ v$ for some propagator $b^{(1)} : \mathbb{0} \to Y$ and some pure term $v^{(0)} : X \to P$. Since $b^{(1)} : \mathbb{0} \to Y$ we have $b^{(1)} \equiv [\,]_Y^{(0)}$, and the first result follows. When $X = \mathbb{0}$, it follows that $a^{(1)} \equiv \mathtt{tag}^{(1)} \circ v^{(0)}$. When $a : X \to \mathbb{0}$ is pure, one has $a \equiv \mathtt{tag}^{(1)} \circ ([\,]_P \circ a)^{(0)}$.

2. The proof proceeds by structural induction. If f is pure the result is obvious, otherwise f can be written in a unique way as $f = g \circ \mathtt{op} \circ u$ where u is pure, $\mathtt{op}$ is either $\mathtt{tag}$ or $\mathtt{untag}$ and g is the remaining part of f. By induction, either g is a propagator or $g \equiv b \circ \mathtt{untag} \circ \mathtt{tag} \circ v$ for some pure term v and some propagator b. So, there are four cases to consider. (1) If $\mathtt{op} = \mathtt{tag}$ and g is a propagator then f is a propagator. (2) If $\mathtt{op} = \mathtt{untag}$ and g is a propagator then by Point 1 there is a pure term w such that $u \equiv \mathtt{tag} \circ w$, so that $f \equiv g^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ w^{(0)}$. (3) If $\mathtt{op} = \mathtt{tag}$ and $g \equiv b^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v^{(0)}$ then $f \equiv b \circ \mathtt{untag} \circ \mathtt{tag} \circ v \circ \mathtt{tag} \circ u$. Since $v : \mathbb{0} \to P$ is pure we have $\mathtt{tag} \circ v \equiv id_{\mathbb{0}}$, so that $f \equiv b^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ u^{(0)}$. (4) If $\mathtt{op} = \mathtt{untag}$ and $g \equiv b^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ v^{(0)}$ then $f \equiv b \circ \mathtt{untag} \circ \mathtt{tag} \circ v \circ \mathtt{untag} \circ u$. Since v is pure, by (ax) and (subs$_\sim$) we have $\mathtt{untag} \circ \mathtt{tag} \circ v \sim v$. Besides, by (ax) and (repl$_\sim$) we have $v \circ \mathtt{untag} \circ \mathtt{tag} \sim v$ and $\mathtt{untag} \circ \mathtt{tag} \circ v \circ \mathtt{untag} \circ \mathtt{tag} \sim \mathtt{untag} \circ \mathtt{tag} \circ v$. Since $\sim$ is an equivalence relation these three weak equations imply $\mathtt{untag} \circ \mathtt{tag} \circ v \circ \mathtt{untag} \circ \mathtt{tag} \sim v \circ \mathtt{untag} \circ \mathtt{tag}$. By rule (eq$_3$) we get $\mathtt{untag} \circ \mathtt{tag} \circ v \circ \mathtt{untag} \equiv v \circ \mathtt{untag}$, and by Point 1 there is a pure term w such that $u \equiv \mathtt{tag} \circ w$, so that $f \equiv (b \circ v)^{(1)} \circ \mathtt{untag} \circ \mathtt{tag} \circ w^{(0)}$.

□

Thanks to Proposition 4.3, in order to study equations in the logic $L_{exc\text{-}core}$ we may restrict our study to pure terms, propagators of the form $[\,]_Y^{(0)} \circ \mathtt{tag}^{(1)} \circ v^{(0)}$ and catchers of the form $a^{(1)} \circ \mathtt{untag}^{(2)} \circ \mathtt{tag}^{(1)} \circ u^{(0)}$.

Proposition 4.4. 1. *For all $a_1^{(1)}, a_2^{(1)} : P \to Y$ and $u_1^{(0)}, u_2^{(0)} : X \to P$, let $f_1^{(2)} = a_1 \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1 : X \to Y$ and $f_2^{(2)} = a_2 \circ \mathtt{untag} \circ \mathtt{tag} \circ u_2 : X \to Y$. Then $f_1 \sim f_2 \iff a_1 \circ u_1 \equiv a_2 \circ u_2$ and $f_1 \equiv f_2 \iff (a_1 \equiv a_2$ and $a_1 \circ u_1 \equiv a_2 \circ u_2)$.*

2. *For all $a_1^{(1)} : P \to Y$, $u_1^{(0)} : X \to P$ and $a_2^{(1)} : X \to Y$, let $f_1^{(2)} = a_1 \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1 : X \to Y$. Then $f_1 \sim a_2 \iff a_1 \circ u_1 \equiv a_2$ and $f_1 \equiv a_2 \iff (a_1 \circ u_1 \equiv a_2$ and $a_1 \equiv [\,]_Y \circ \mathtt{tag})$.*

3. *Let us assume that $[\,]_Y^{(0)}$ is a monomorphism with respect to propagators. For all $v_1^{(0)}, v_2^{(0)} : X \to P$, let $a_1^{(1)} = [\,]_Y \circ \mathtt{tag} \circ v_1 : X \to Y$ and $a_2^{(1)} = [\,]_Y \circ \mathtt{tag} \circ v_2 : X \to Y$. Then $a_1 \equiv a_2 \iff v_1 \equiv v_2$.*

4. *Let us assume that there is a unit type* $\mathbb{1}$ *and a boolean type* $\mathbb{B}$, *in the sense of Definition 3.2 and that* $\langle\ \rangle_Y^{(0)}$ *is an epimorphism with respect to pure terms. For all* $v_1^{(0)} : X \to P$ *and* $v_2^{(0)} : X \to Y$, *let* $a_1^{(1)} = [\ ]_Y \circ \texttt{tag} \circ v_1 : X \to Y$. *Then* $a_1 \equiv v_2 \iff \texttt{true} \equiv \texttt{false}$.

Proof. 1. Rule (eq$_2$) implies that $f_1 \equiv f_2$ if and only if $f_1 \sim f2$ and $f_1 \circ [\ ]_X \equiv f_2 \circ [\ ]_X$. On the one hand, $f_1 \sim f_2$ if and only if $a_1 \circ u_1 \equiv a_2 \circ u_2$: indeed, for each $i \in \{1, 2\}$, by (ax) and (subs$_\sim$), since u_i is pure we have $f_i \sim a_i \circ u_i$. On the other hand, let us prove that $f_1 \circ [\ ]_X \equiv f_2 \circ [\ ]_X$ if and only if $a_1 \equiv a_2$. For each $i \in \{1, 2\}$, the propagator $\texttt{tag} \circ u_i \circ [\ ]_X : \mathbb{0} \to \mathbb{0}$ satisfies $\texttt{tag} \circ u_i \circ [\ ]_X \equiv id_{\mathbb{0}}$, so that $f_i \circ [\ ]_X \equiv a_i \circ \texttt{untag}$. Thus, $f_1 \circ [\ ]_X \equiv f_2 \circ [\ ]_X$ if and only if $a_1 \circ \texttt{untag} \equiv a_2 \circ \texttt{untag}$. Clearly, if $a_1 \equiv a_2$ then $a_1 \circ \texttt{untag} \equiv a_2 \circ \texttt{untag}$. Conversely, if $a_1 \circ \texttt{untag} \equiv a_2 \circ \texttt{untag}$ then $a_1 \circ \texttt{untag} \circ \texttt{tag} \equiv a_2 \circ \texttt{untag} \circ \texttt{tag}$, so that by (ax) and (repl$_\sim$) we get $a_1 \sim a_2$, which means that $a_1 \equiv a_2$ because a_1 and a_2 are propagators.

2. Rule (eq$_2$) implies that $f_1 \equiv a_2$ if and only if $f_1 \sim a_2$ and $f_1 \circ [\ ]_X \equiv a_2 \circ [\ ]_X$. On the one hand, $f_1 \sim a_2$ if and only if $a_1 \circ u_1 \equiv a_2$: indeed, by (ax) and (subs$_\sim$), since u_1 is pure we have $f_1 \sim a_1 \circ u_1$. On the other hand, let us prove that $f_1 \circ [\ ]_X \equiv a_2 \circ [\ ]_X$ if and only if $a_1 \equiv [\ ]_Y \circ \texttt{tag}$, in two steps. Since $a_2 \circ [\ ]_X : \mathbb{0} \to Y$ is a propagator, we have $a_2 \circ [\ ]_X \equiv [\ ]_Y$. Since $f_1 \circ [\ ]_X = a_1 \circ \texttt{untag} \circ \texttt{tag} \circ u_1 \circ [\ ]_X$ with $\texttt{tag} \circ u_1 \circ [\ ]_X : \mathbb{0} \to \mathbb{0}$ a propagator, we have $\texttt{tag} \circ u_1 \circ [\ ]_X \equiv id_{\mathbb{0}}$ and thus we get $f_1 \circ [\ ]_X \equiv a_1 \circ \texttt{untag}$. Thus, $f_1 \circ [\ ]_X \equiv a_2 \circ [\ ]_X$ if and only if $a_1 \circ \texttt{untag} \equiv [\ ]_Y$. If $a_1 \circ \texttt{untag} \equiv [\ ]_Y$ then $a_1 \circ \texttt{untag} \circ \texttt{tag} \equiv [\ ]_Y \circ \texttt{tag}$, by (ax) and (repl$_\sim$) this implies $a_1 \sim [\ ]_Y \circ \texttt{tag}$, which is a strong equality because both members are propagators. Conversely, if $a_1 \equiv [\ ]_Y \circ \texttt{tag}$ then $a_1 \circ \texttt{untag} \equiv [\ ]_Y \circ \texttt{tag} \circ \texttt{untag}$, by Point 2 in Lemma 4.1 this implies $a_1 \circ \texttt{untag} \equiv [\ ]_Y$. Thus, $a_1 \circ \texttt{untag} \equiv [\ ]_Y$ if and only if $a_1 \equiv [\ ]_Y \circ \texttt{tag}$.

3. Clearly, if $v_1 \equiv v_2$ then $a_1 \equiv a_2$. Conversely, if $a_1 \equiv a_2$, i.e., if $[\ ]_Y \circ \texttt{tag} \circ v_1 \equiv [\ ]_Y \circ \texttt{tag} \circ v_2$, since $[\ ]_Y$ is a monomorphism with respect to propagators we get $\texttt{tag} \circ v_1 \equiv \texttt{tag} \circ v_2$. By Point 3 in Lemma 4.1, this means that $v_1 \equiv v_2$.

4. If $\texttt{true} \equiv \texttt{false}$ then according to the definition of a boolean type we have $a_1 \equiv v_2$. Conversely if $a_1 \equiv v_2$, let $a'_1 = \texttt{true} \circ \langle\ \rangle_Y \circ a_1 : X \to \mathbb{B}$ and $a'_2 = \texttt{true} \circ \langle\ \rangle_Y \circ v_2 : X \to \mathbb{B}$ and $b = \texttt{false} \circ \langle\ \rangle_P : P \to \mathbb{B}$, then $\texttt{TRY}(a'_1, \texttt{CATCH}(b)) \equiv \texttt{TRY}(a'_2, \texttt{CATCH}(b))$. Let us prove that this implies $\texttt{true} \circ \langle\ \rangle_X \equiv \texttt{false} \circ \langle\ \rangle_X$. On the right hand side, since a'_2 is pure we can use the substitution rule for weak equations, so that we get $\texttt{TRY}(a'_2, \texttt{CATCH}(b)) \sim \texttt{CATCH}(b) \circ a'_2 \sim id_{\mathbb{B}} \circ a'_2 \sim a'_2$. Since both $\texttt{TRY}(a'_2, \texttt{CATCH}(b))$ and a'_2 are propagators we get $\texttt{TRY}(a'_2, \texttt{CATCH}(b)) \equiv a'_2$. And since $a'_2 = \texttt{true} \circ \langle\ \rangle_Y \circ v_2 \equiv \texttt{true} \circ \langle\ \rangle_X$ we get $\texttt{TRY}(a'_2, \texttt{CATCH}(b)) \equiv \texttt{true} \circ \langle\ \rangle_X$. On the left hand side we get $\texttt{TRY}(a'_1, \texttt{CATCH}(b)) \sim \texttt{CATCH}(b) \circ a'_1$ where $a'_1 = \texttt{true} \circ \langle\ \rangle_Y \circ a_1 = \texttt{true} \circ \langle\ \rangle_Y \circ [\ ]_Y \circ \texttt{tag} \circ v_1$. Since $\texttt{true} \circ \langle\ \rangle_Y \circ [\ ]_Y : \mathbb{0} \to \mathbb{B}$ is pure we have $\texttt{true} \circ \langle\ \rangle_Y \circ [\ ]_Y \equiv [\ ]_B$, thus

$a'_1 \equiv [\;]_B \circ \mathtt{tag} \circ v_1$. It follows that $\mathtt{CATCH}(b) \circ a'_1 \equiv \mathtt{CATCH}(b) \circ [\;]_B \circ \mathtt{tag} \circ v_1 \equiv b \circ \mathtt{untag} \circ \mathtt{tag} \circ v_1$. Since $\mathtt{untag} \circ \mathtt{tag} \sim id_P$ and v_1 is pure we get $\mathtt{CATCH}(b) \circ a'_1 \sim b \circ v_1$, where $b \circ v_1 = \mathtt{false} \circ \langle\;\rangle_P \circ v_1 \equiv \mathtt{false} \circ \langle\;\rangle_X$. Altogether, we have $\mathtt{TRY}(a'_1, \mathtt{CATCH}(b)) \equiv \mathtt{false} \circ \langle\;\rangle_X$. Thus, we have proved that if $a_1 \equiv v_2$ then $\mathtt{true} \circ \langle\;\rangle_X \equiv \mathtt{false} \circ \langle\;\rangle_X$. Since $\langle\;\rangle_X$ is an epimorphism with respect to pure terms, we obtain $\mathtt{true} \equiv \mathtt{false}$.

□

Theorem 4.5. *If there is a unit type $\mathbb{1}$ and a boolean type $\mathbb{B}$ in the sense of Definition 3.2, if $[\;]_Y^{(0)}$ is a monomorphism with respect to propagators and if $\langle\;\rangle_Y^{(0)}$ is an epimorphism with respect to pure terms for each non-empty type Y, the core theory of exceptions $T_{exc\text{-}core}$ is relatively Hilbert-Post complete with respect to the pure sublogic $L_{exc\text{-}core}^{(0)}$ of $L_{exc\text{-}core}$.*

Proof. The proof is based upon Propositions 4.3, 4.4 and 2.7. It follows the same lines as the proof of Theorem 3.5, except when X is empty: because of catchers the proof here is slightly more subtle. First, the theory $T_{exc\text{-}core}$ is consistent: it cannot be proved that $\mathtt{untag}^{(2)} \equiv [\;]_P^{(0)}$ because because the logic $L_{exc\text{-}core}$ is sound with respect to its intended model and the interpretation of this equation in the intended model is false: indeed, the function $\mathtt{untag} : E \to P + E$ is such that $\mathtt{untag}(\mathtt{tag}(p)) = p \in P$ for each $p \in P$ while $[\;]_P(e) = e \in E$ for each $e \in E$, which includes $e = \mathtt{tag}(p)$; since $P + E$ is a disjoint union we have $\mathtt{untag}(e) \neq [\;]_P(e)$ when $e = \mathtt{tag}(p)$. Now, let us consider an equation between two terms f_1 and f_2 with domain X; we distinguish two cases, whether X is empty or not. When X is non-empty, then $\langle\;\rangle_X$ is an epimorphism with respect to pure terms. Thus, Propositions 4.3 and 4.4 prove that the given equation is $T_{exc\text{-}core}$-equivalent to a finite set of equations between pure terms. When X is empty, then all terms from X to Y are only *weakly* equivalent to $[\;]_Y$, so that we cannot conclude yet for any given equation. Let us consider two cases. First, if the given equation is an equation between propagators then both f_1 and f_2 are strongly equivalent to $[\;]_Y$ so that the given equation is $T_{exc\text{-}core}$-equivalent to the empty set of equations between pure terms. Otherwise, at least one of f_1 and f_2 is a catcher, and there are two subcases to consider, whether the given equation is weak or strong. If the equation is $f_1 \sim f_2$ then since $f_1 \sim [\;]_Y$ and $f_2 \sim [\;]_Y$ it is still $T_{exc\text{-}core}$-equivalent to the empty set of equations between pure terms. Now, if the equation is $f_1 \equiv f_2$ then by Point 1 or 2 of Proposition 4.4, the equation $f_1 \equiv f_2$ is $T_{exc\text{-}core}$-equivalent to a set of equations between propagators. We have seen that each equation between propagators (whether X is empty or not) is $T_{exc\text{-}core}$-equivalent to a set of equations between pure terms, so that $f_1 \equiv f_2$ is $T_{exc\text{-}core}$-equivalent to the union of the corresponding sets of pure equations. Finally, the result follows from Proposition 2.7. □

5 Verification of Hilbert-Post Completeness in Coq

All the statements of Sections 3 and 4 have been checked in Coq. The proofs can be found in `https://forge.imag.fr/frs/download.php/645/hp-0.2.tar.gz`, as well as an almost dual proof for the completeness of the state. They share the same framework, defined in [6]:

1. the terms of each logic are inductively defined through the dependent type named `term` which builds a new `Type` out of two input `Type`s. For instance, `term Y X` is the `Type` of all terms of the form `f`: `X` $\to$ `Y`;
2. the decorations are enumerated: `pure` and `propagator` for both languages, and `catcher` for the core language;
3. decorations are inductively assigned to the terms via the dependent type called `is`. The latter builds a proposition (a `Prop` instance in Coq) out of a `term` and a decoration. Accordingly, `is pure (id X)` is a `Prop` instance;
4. for the core language, we state the rules with respect to weak and strong equalities by defining them in a mutually inductive way.

For instance, the completeness proof for the exceptions core language is 950 SLOC in Coq where it is 460 SLOC in LaTeX. Full certification runs in 6.745s on a Intel i7-3630QM @2.40GHz using the Coq Proof Assistant, v. 8.4pl3.

Below table details the proof lengths and timings for each library.

Proof lengths & Benchmarks				
package	source	length in Coq	length in LaTeX	Coq cert. time
exc_cl-hp	HPCompleteCoq.v	40 KB	15 KB	6.745 sec.
exc_pl-hp	HPCompleteCoq.v	8 KB	6 KB	1.704 sec.
exc_impl	Proofs.v	4 KB	2 KB	1.696 sec.
st-hp	HPCompleteCoq.v	48 KB	15 KB	7.183 sec.

References

[1] Viviana Bono and Manfred Kerber. Extending Hoare Calculus to Deal with Crash. The University of Birmingham, School of Computer Science. Research Reports, CSR-06-08.

[2] Jean-Guillaume Dumas, Dominique Duval, Laurent Fousse, Jean-Claude Reynaud. Decorated proofs for computational effects: States. ACCAT 2012. Electronic Proceedings in Theoretical Computer Science 93, p. 45-59 (2012).

[3] Jean-Guillaume Dumas, Dominique Duval, Laurent Fousse, Jean-Claude Reynaud. A duality between exceptions and states. Mathematical Structures in Computer Science 22, p. 719-722 (2012).

[4] Jean-Guillaume Dumas, Dominique Duval, Jean-Claude Reynaud. Cartesian effect categories are Freyd-categories. J. of Symb. Comput. 46, p. 272-293 (2011).

[5] Jean-Guillaume Dumas, Dominique Duval, Burak Ekici and Jean-Claude Reynaud. Certified proofs in programs involving exceptions. CICM 2014, Coimbra, Portugal, 7–11 July 2014. CEUR Workshop Proceedings, n^o 1186, paper 20.

[6] Jean-Guillaume Dumas, Dominique Duval, Burak Ekici, Damien Pous. Formal verification in Coq of program properties involving the global state. JFLA 2014, pages 1–15, Frejus, France, 8–11 January 2014.

[7] Working Draft, Standard for Programming Language C++. ISO/IEC JTC1/SC22/WG21 standard 14882:2011.

[8] James Gosling, Bill Joy, Guy Steele, Gilad Bracha. The Java Language Specification, Third Edition. Addison-Wesley Longman (2005).

[9] John M. Lucassen, David K. Gifford. Polymorphic effect systems. POPL 1988. ACM Press, p. 47-57.

[10] Eugenio Moggi. Notions of Computation and Monads. Information and Computation 93(1), p. 55-92 (1991).

[11] Matija Pretnar. The Logic and Handling of Algebraic Effects. PhD. University of Edinburgh 2010.

[12] Gordon D. Plotkin, John Power. Notions of Computation Determine Monads. FoSSaCS 2002. LNCS, Vol. 2620, p. 342-356, Springer (2002).

[13] Gordon D. Plotkin, Matija Pretnar. Handlers of Algebraic Effects. ESOP 2009. LNCS, Vol. 5502, p. 80-94, Mpringer (2009).

[14] Sam Staton. Completeness for Algebraic Theories of Local State. FoSSaCS 2010. LNCS, Vol. 6014, p. 48-63, Springer (2010).

[15] Alfred Tarski. III On some fundamental concepts in mathematics. In Logic, Semantics, Metamathematics: Papers from 1923 to 1938 by Alfred Tarski, p. 30-37. Oxford University Press (1956).

A Completeness for states

Most programming languages such as C/C++ and Java support the usage and manipulation of the state (memory) structure. Even though the state structure is never syntactically mentioned, the commands are allowed to use or manipulate it, for instance looking up or updating the value of variables. This provides a great flexibility in programming, but in order to prove the correctness of programs, one usually has to revert to an explicit manipulation of the state. Therefore, any access to the state, regardless of usage or manipulation, is treated as a computational effect: a syntactical term $f : X \to Y$ is not interpreted as $f : X \to Y$ unless it is *pure*, that is unless it does not use the variables in any manner. Indeed, a term which updates the state has instead the following interpretation: $f : X \times S \to Y \times S$ where '$\times$' is the product operator and S is the set of possible states. In [6], we proposed a proof system to prove program properties involving states effect, while keeping the memory manipulations implicit. We summarize this system next and prove its Hilbert-Post completeness in Theorem A.6, as a variant of Pretnar's result [11].

As noticed in [5], the logic $L_{exc\text{-}core}$ is exactly dual to the logic L_{st} for states (as reminded below). Thus, the dual of all results in Section 4 are valid, with the dual proof. This holds for the completeness Theorem 4.5. However, the intended models for exceptions and for states rely on the category of sets, which is not self-dual, and the additional assumptions in Theorem 4.5, like the existence of a boolean type, cannot be dualized without loosing the soundness of the logic with respect to its intended interpretation. It follows that the completeness Theorem A.6 for the theory for states is not exactly the dual of Theorem 4.5. In this Appendix, for the sake of readability, we give all the details of the proof of Theorem A.6; we will mention which parts are *not* the dual of the corresponding parts in the proof of Theorem 4.5.

As in [2], decorated logics for states are obtained from equational logics by classifying terms and equations. Terms are classified as *pure* terms, *accessors* or *modifiers*, which is expressed by adding a *decoration* or superscript, respectively (0), (1) and (2); decoration and type information about terms may be omitted when they are clear from the context or when they do not matter. Equations are classified as *strong* or *weak* equations, denoted respectively by the symbols $\equiv$ and $\sim$. Weak equations relates to the values returned by programs, while strong equations relates to both values and side effects. In order to observe the state, accessors may use the values stored in *locations*, and modifiers may update these values. In order to focus on the main features of the proof of completeness, let us assume that only one location can be observed and modified; the general case, with an arbitrary number of locations, is considered in Remark A.7. The logic for dealing with pure terms may be any logic which extends a monadic equational logic with constants $L_{eq,\mathbb{1}}$; its terms are decorated as pure and its equations are strong. This *pure sublogic* $L_{st}^{(0)}$ is extended to form the corresponding *decorated logic for states* L_{st}. The rules for L_{st} are given in Fig. 3. A theory $T^{(0)}$ of $L_{st}^{(0)}$ is chosen, then the *theory of states* T_{st} is the theory of L_{st} generated from $T^{(0)}$. Let us now discuss the logic L_{st} and its intended interpretation in sets; it is assumed that some model of the pure subtheory $T^{(0)}$ in sets has been chosen; the names of the rules refer to Fig. 3.

Each type X is interpreted as a set, denoted X. The intended model is described with respect to a set S called the *set of states*, which does not appear in the syntax. A pure term $u^{(0)} : X \to Y$ is interpreted as a function $u : X \to Y$, an accessor $a^{(1)} : X \to Y$ as a function $a : S \times X \to Y$, and a modifier $f^{(2)} : X \to Y$ as a function $f : S \times X \to S \times Y$. There are obvious conversions from pure terms to accessors and from accessors to modifiers, which allow to consider all terms as modifiers whenever needed; for instance, this allows to interpret the composition of terms without mentioning Kleisli composition; the complete characterization is given in [2].

Here, for the sake of simplicity, we consider a single variable (as done, e.g., in [11] and [14]), and dually to the choice of a unique exception name in Section 4. See Remark A.7 for the generalization to an arbitrary number of variables. The values of the unique location have type V. The fundamental operations for dealing with the state are the accessor $\mathtt{lookup}^{(1)} : \mathbb{1} \to V$ for reading the value of the location and the modifier $\mathtt{update}^{(2)} : V \to \mathbb{1}$ for updating this value. According to their decorations, they are interpreted respectively as functions $\mathtt{lookup} : S \to V$ and $\mathtt{update} : S \times V \to S$. Since there is only one location, it might be assumed that $\mathtt{lookup} : S \to V$ is a bijection and that $\mathtt{update} : S \times V \to S$ maps each $(s, v) \in S \times V$ to the unique $s' \in S$ such that $\mathtt{lookup}(s') = v$: this is expressed by a weak equation, as explained below.

A strong equation $f \equiv g$ means that f and g return the same result and modify the state in "the same way", which means that no difference can be observed between the side-effects performed by f and by g. Whenever $\mathtt{lookup} : S \to V$ is a bijection, a strong equation $f^{(2)} \equiv g^{(2)} : X \to Y$ is interpreted as the equality $f = g : S \times X \to S \times Y$: for each $(s, x) \in S \times X$, let $f(s, x) = (s', y')$ and $g(s, x) = (s'', y'')$, then $f \equiv g$ means that $y' = y''$ and $s' = s''$ for all (s, x). Strong equations form a congruence. A weak equation $f \sim g$ means that f and g return the same result although they may modify the state in different ways. Thus, a weak equation $f^{(2)} \sim g^{(2)} : X \to Y$ is interpreted as the equality $pr_Y \circ f = pr_Y \circ g : S \times X \to Y$, where $pr_Y : S \times Y \to Y$ is the projection; with the same notations as above, this means that $y' = y''$ for all (s, x). Weak equations do not form a congruence: the replacement rule holds only when the replaced term is pure. The fundamental equation for states is provided by rule (ax): $\mathtt{lookup}^{(1)} \circ \mathtt{update}^{(2)} \sim id_V$. This means that updating the location with a value v and then observing the value of the location does return v. Clearly this is only a weak equation: its right-hand side does not modify the state while its left-hand side usually does. There is an obvious conversion from strong to weak equations ($\equiv$-to-$\sim$), and in addition strong and weak equations coincide on accessors by rule (eq_1). Two modifiers $f_1^{(2)}, f_2^{(2)} : X \to Y$ modify the state in the same way if and only if $\langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ f_2 : X \to \mathbb{1}$, where $\langle\ \rangle_Y : Y \to \mathbb{1}$ throws out the returned value. Then weak and strong equations are related by the property that $f_1 \equiv f_2$ if and only if $f_1 \sim f_2$ and $\langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ f_2$, by rule ($\mathrm{eq}_2$). This can be expressed as a pair of weak equations $f_1 \sim f_2$ and $\mathtt{lookup} \circ \langle\ \rangle_Y \circ f_1 \sim \mathtt{lookup} \circ \langle\ \rangle_Y \circ f_2$, by rule ($\mathrm{eq}_3$). Some easily derived properties are stated in Lemma A.1; Point 1 will be used repeatedly.

Lemma A.1. *1. each $f^{(2)} : \mathbb{1} \to \mathbb{1}$ is such that $f \sim id_{\mathbb{1}}$, each $f^{(1)} : X \to \mathbb{1}$ is such that $f \equiv \langle\ \rangle_X$, and each $f^{(1)} : \mathbb{1} \to \mathbb{1}$ is such that $f \equiv id_{\mathbb{1}}$.*

2. The fundamental strong equation for states is $\mathtt{update} \circ \mathtt{lookup} \equiv id_{\mathbb{1}}$.

3. For all pure terms $u_1^{(0)}, u_2^{(0)} : V \to Y$, one has: $u_1 \equiv u_2 \iff u_1 \circ \mathtt{lookup} \equiv u_2 \circ \mathtt{lookup} \iff u_1 \circ \mathtt{lookup} \circ \mathtt{update} \equiv u_2 \circ \mathtt{lookup} \circ \mathtt{update}$.

Monadic equational logic with constants $L_{eq,\mathbb{1}}$:
Types and terms: as for monadic equational logic, plus
a unit type $\mathbb{1}$ and a term $\langle\ \rangle_X : X \to \mathbb{1}$ for each X

Rules: as for monadic equational logic, plus (unit) $\dfrac{f\colon X \to \mathbb{1}}{f \equiv \langle\ \rangle_X}$

Decorated logic for states L_{st}:
Pure part: some logic $L_{st}^{(0)}$ extending $L_{eq,\mathbb{1}}$, with a distinguished type V
Decorated terms: $\texttt{lookup}^{(1)}\colon \mathbb{1} \to V$, $\texttt{update}^{(2)}\colon V \to \mathbb{1}$, and
$(f_k \circ \cdots \circ f_1)^{(\max(d_1,\ldots,d_k))} : X_0 \to X_k$ for each $(f_i^{(d_i)} : X_{i-1} \to X_i)_{1\le i\le k}$
with conversions from $f^{(0)}$ to $f^{(1)}$ and from $f^{(1)}$ to $f^{(2)}$
Rules:
(equiv$_\equiv$), (subs$_\equiv$), (repl$_\equiv$) for all decorations
(equiv$_\sim$), (subs$_\sim$) for all decorations, (repl$_\sim$) only when h is pure

(unit$_\sim$) $\dfrac{f\colon X \to \mathbb{1}}{f \sim \langle\ \rangle_X}$ (≡-to-∼) $\dfrac{f \equiv g}{f \sim g}$ (ax) $\dfrac{}{\texttt{lookup} \circ \texttt{update} \sim id_V}$

(eq$_1$) $\dfrac{f_1^{(d_1)} \sim f_2^{(d_2)}}{f_1 \equiv f_2}$ only when $d_1 \le 1$ and $d_2 \le 1$

(eq$_2$) $\dfrac{f_1, f_2\colon X \to Y \quad f_1 \sim f_2 \quad \langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ f_2}{f_1 \equiv f_2}$

(eq$_3$) $\dfrac{f_1, f_2\colon X \to \mathbb{1} \quad \texttt{lookup} \circ f_1 \sim \texttt{lookup} \circ f_2}{f_1 \equiv f_2}$

Figure 3: Logic for states (dual to Fig. 2)

4. *For all pure terms $u^{(0)} : V \to Y$, $v^{(0)} : \mathbb{1} \to Y$, one has: $u \equiv v \circ \langle\ \rangle_V \iff u \circ \texttt{lookup} \equiv v$.*

Proof. 1. Clear.

2. By substitution in the axiom (ax) we get $\texttt{lookup} \circ \texttt{update} \circ \texttt{lookup} \sim \texttt{lookup}$; then by rule (eq$_3$) $\texttt{update} \circ \texttt{lookup} \equiv id_{\mathbb{1}}$.

3. Implications from left to right are clear. Conversely, if $u_1 \circ \texttt{lookup} \circ \texttt{update} \equiv u_2 \circ \texttt{lookup} \circ \texttt{update}$, then using the axiom (ax) and the rule (repl$_\sim$) we get $u_1 \sim u_2$. Since u_1 and u_2 are pure this means that $u_1 \equiv u_2$.

4. First, since $\langle\ \rangle_V \circ \texttt{lookup} : \mathbb{1} \to \mathbb{1}$ is an accessor we have $\langle\ \rangle_V \circ \texttt{lookup} \equiv id_{\mathbb{1}}$. Now, if $u \equiv v \circ \langle\ \rangle_V$ then $u \circ \texttt{lookup} \equiv v \circ \langle\ \rangle_V \circ \texttt{lookup}$, so that $u \circ \texttt{lookup} \equiv v$. Conversely, if $u \circ \texttt{lookup} \equiv v$ then $u \circ \texttt{lookup} \equiv v \circ \langle\ \rangle_V \circ \texttt{lookup}$, and by Point (3) this means that $u \equiv v \circ \langle\ \rangle_V$.

□

Our main result is Theorem A.6 about the relative Hilbert-Post completeness of the decorated theory of states under suitable assumptions.

Proposition A.2. *1. For each accessor $a^{(1)} : X \to Y$, either a is pure or there is a pure term $v^{(0)} : V \to Y$ such that $a^{(1)} \equiv v^{(0)} \circ \mathtt{lookup}^{(1)} \circ \langle\,\rangle_X^{(0)}$. For each accessor $a^{(1)} : \mathbb{1} \to Y$ (either pure or not), there is a pure term $v^{(0)} : V \to Y$ such that $a^{(1)} \equiv v^{(0)} \circ \mathtt{lookup}^{(1)}$.*

2. For each modifier $f^{(2)} : X \to Y$, either f is an accessor or there is an accessor $a^{(1)} : X \to V$ and a pure term $u^{(0)} : V \to Y$ such that $f^{(2)} \equiv u^{(0)} \circ \mathtt{lookup}^{(1)} \circ \mathtt{update}^{(2)} \circ a^{(1)}$.

Proof. 1. If the accessor $a^{(1)} : X \to Y$ is not pure then it contains at least one occurrence of $\mathtt{lookup}^{(1)}$. Thus, it can be written in a unique way as $a = v \circ \mathtt{lookup} \circ b$ for some pure term $v^{(0)} : V \to Y$ and some accessor $b^{(1)} : X \to \mathbb{1}$. Since $b^{(1)} : X \to \mathbb{1}$ we have $b^{(1)} \equiv \langle\,\rangle_X^{(0)}$, and the first result follows. When $X = \mathbb{1}$, it follows that $a^{(1)} \equiv v^{(0)} \circ \mathtt{lookup}^{(1)}$. When $a : \mathbb{1} \to Y$ is pure, one has $a \equiv (a \circ \langle\,\rangle_V)^{(0)} \circ \mathtt{lookup}^{(1)}$.

2. The proof proceeds by structural induction. If f is pure the result is obvious, otherwise f can be written in a unique way as $f = u \circ \mathtt{op} \circ g$ where u is pure, $\mathtt{op}$ is either $\mathtt{lookup}$ or $\mathtt{update}$ and g is the remaining part of f. By induction, either g is an accessor or $g \equiv v \circ \mathtt{lookup} \circ \mathtt{update} \circ b$ for some pure term v and some accessor b. So, there are four cases to consider.

- If $\mathtt{op} = \mathtt{lookup}$ and g is an accessor then f is an accessor.
- If $\mathtt{op} = \mathtt{update}$ and g is an accessor then by Point 1 there is a pure term w such that $u \equiv w \circ \mathtt{lookup}$, so that $f \equiv w^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \circ g^{(1)}$.
- If $\mathtt{op} = \mathtt{lookup}$ and $g \equiv v^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \circ b^{(1)}$ then $f \equiv u \circ \mathtt{lookup} \circ v \circ \mathtt{lookup} \circ \mathtt{update} \circ b$. Since $v : V \to \mathbb{1}$ is pure we have $v \circ \mathtt{lookup} \equiv id_{\mathbb{1}}$, so that $f \equiv u^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \circ b^{(1)}$.
- If $\mathtt{op} = \mathtt{update}$ and $g \equiv v^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \circ b^{(1)}$ then $f \equiv u^{(0)} \circ \mathtt{update} \circ v^{(0)} \circ \mathtt{lookup} \circ \mathtt{update} \circ b^{(1)}$. Since v is pure, by (ax) and (repl$_\sim$) we have $v \circ \mathtt{lookup} \circ \mathtt{update} \sim v$. Besides, by (ax) and (subs$_\sim$) we have $\mathtt{lookup} \circ \mathtt{update} \circ v \sim v$ and $\mathtt{lookup} \circ \mathtt{update} \circ v \circ \mathtt{lookup} \circ \mathtt{update} \sim v \circ \mathtt{lookup} \circ \mathtt{update}$. Since $\sim$ is an equivalence relation these three weak equations imply $\mathtt{lookup} \circ \mathtt{update} \circ v \circ \mathtt{lookup} \circ \mathtt{update} \sim \mathtt{lookup} \circ \mathtt{update} \circ v$. By rule (eq$_3$) we get $\mathtt{update} \circ v \circ \mathtt{lookup} \circ \mathtt{update} \equiv \mathtt{update} \circ v$, so that $f \equiv u^{(0)} \circ \mathtt{update} \circ (v \circ b)^{(1)}$.

□

Thanks to Proposition A.2, in order to study equations in the logic L_{st} we may restrict our study to pure terms, accessors of the form $v^{(0)} \circ \mathtt{lookup}^{(1)} \circ \langle\,\rangle_X^{(0)}$ and modifiers of the form $u^{(0)} \circ \mathtt{lookup}^{(1)} \circ \mathtt{update}^{(2)} \circ a^{(1)}$.

Point 4 in Proposition A.2 is *not* dual to Point 4 in Proposition 4.3

Proposition A.3. *1. For all $a_1^{(1)}, a_2^{(1)} : X \to V$ and $u_1^{(0)}, u_2^{(0)} : V \to Y$, let $f_1^{(2)} = u_1 \circ \texttt{lookup} \circ \texttt{update} \circ a_1 : X \to Y$ and $f_2^{(2)} = u_2 \circ \texttt{lookup} \circ \texttt{update} \circ a_2 : X \to Y$. Then*

$$\begin{cases} f_1 \sim f_2 \iff u_1 \circ a_1 \equiv u_2 \circ a_2 \\ f_1 \equiv f_2 \iff a_1 \equiv a_2 \ \ \text{and} \ \ u_1 \circ a_1 \equiv u_2 \circ a_2 \end{cases}$$

2. For all $a_1^{(1)} : X \to V$, $u_1^{(0)} : V \to Y$ and $a_2^{(1)} : X \to Y$, let $f_1^{(2)} = u_1 \circ \texttt{lookup} \circ \texttt{update} \circ a_1 : X \to Y$. Then

$$\begin{cases} f_1 \sim a_2 \iff u_1 \circ a_1 \equiv a_2 \\ f_1 \equiv a_2 \iff u_1 \circ a_1 \equiv a_2 \ \ \text{and} \ \ a_1 \equiv \texttt{lookup} \circ \langle\ \rangle_X \end{cases}$$

3. Let us assume that $\langle\ \rangle_X^{(0)}$ is an epimorphism with respect to accessors. For all $v_1^{(0)}, v_2^{(0)} : V \to Y$ let $a_1^{(1)} = v_1 \circ \texttt{lookup} \circ \langle\ \rangle_X : X \to Y$ and $a_2^{(1)} = v_2 \circ \texttt{lookup} \circ \langle\ \rangle_X : X \to Y$. Then

$$a_1 \equiv a_2 \iff v_1 \equiv v_2$$

4. Let us assume that $\langle\ \rangle_V^{(0)}$ is an epimorphism with respect to accessors and that there exists a pure term $k_X^{(0)} : \mathbb{1} \to X$. For all $v_1^{(0)} : V \to Y$ and $v_2^{(0)} : X \to Y$, let $a_1^{(1)} = v_1 \circ \texttt{lookup} \circ \langle\ \rangle_X : X \to Y$. Then

$$a_1 \equiv v_2 \iff v_1 \equiv v_2 \circ k_X \circ \langle\ \rangle_V \ \ \text{and} \ \ v_2 \equiv v_2 \circ k_X \circ \langle\ \rangle_X$$

Proof. 1. Rule (eq$_2$) implies that $f_1 \equiv f_2$ if and only if $f_1 \sim f2$ and $\langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ f_2$. On the one hand, $f_1 \sim f_2$ if and only if $u_1 \circ a_1 \equiv u_1 \circ a_2$: indeed, for each $i \in \{1, 2\}$, by (ax) and (repl$_\sim$), since u_i is pure we have $f_i \sim u_i \circ a_i$. On the other hand, let us prove that $\langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ f_2$ if and only if $a_1 \equiv a_2$.

- For each $i \in \{1, 2\}$, the accessor $\langle\ \rangle_Y \circ u_i \circ \texttt{lookup} : \mathbb{1} \to \mathbb{1}$ satisfies $\langle\ \rangle_Y \circ u_i \circ \texttt{lookup} \equiv id_{\mathbb{1}}$, so that $\langle\ \rangle_Y \circ f_i \equiv \texttt{update} \circ a_i$. Thus, $\langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ f_2$ if and only if $\texttt{update} \circ a_1 \equiv \texttt{update} \circ a_2$.
- Clearly, if $a_1 \equiv a_2$ then $\texttt{update} \circ a_1 \equiv \texttt{update} \circ a_2$. Conversely, if $\texttt{update} \circ a_1 \equiv \texttt{update} \circ a_2$ then $\texttt{lookup} \circ \texttt{update} \circ a_1 \equiv \texttt{lookup} \circ \texttt{update} \circ a_2$, so that by (ax) and (subs$_\sim$) we get $a_1 \sim a_2$, which means that $a_1 \equiv a_2$ because a_1 and a_2 are accessors.

2. Rule (eq$_2$) implies that $f_1 \equiv a_2$ if and only if $f_1 \sim a_2$ and $\langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ a_2$. On the one hand, $f_1 \sim a_2$ if and only if $u_1 \circ a_1 \equiv a_2$: indeed, by (ax) and (repl$_\sim$), since u_1 is pure we have $f_1 \sim u_1 \circ a_1$. On the other hand, let us prove that $\langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ a_2$ if and only if $a_1 \equiv \texttt{lookup} \circ \langle\ \rangle_X$, in two steps.

- Since $\langle\ \rangle_Y \circ a_2 : X \to \mathbb{1}$ is an accessor, we have $\langle\ \rangle_Y \circ a_2 \equiv \langle\ \rangle_X$. Since $\langle\ \rangle_Y \circ f_1 = \langle\ \rangle_Y \circ u_1 \circ \mathtt{lookup} \circ \mathtt{update} \circ a_1$ with $\langle\ \rangle_Y \circ u_1 \circ \mathtt{lookup} : \mathbb{1} \to \mathbb{1}$ an accessor, we have $\langle\ \rangle_Y \circ u_1 \circ \mathtt{lookup} \equiv id_{\mathbb{1}}$ and thus we get $\langle\ \rangle_Y \circ f_1 \equiv \mathtt{update} \circ a_1$. Thus, $\langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ a_2$ if and only if $\mathtt{update} \circ a_1 \equiv \langle\ \rangle_X$.
- If $\mathtt{update} \circ a_1 \equiv \langle\ \rangle_X$ then $\mathtt{lookup} \circ \mathtt{update} \circ a_1 \equiv \mathtt{lookup} \circ \langle\ \rangle_X$, by (ax) and (subs$_\sim$) this implies $a_1 \sim \mathtt{lookup} \circ \langle\ \rangle_X$, which is a strong equality because both members are accessors. Conversely, if $a_1 \equiv \mathtt{lookup} \circ \langle\ \rangle_X$ then $\mathtt{update} \circ a_1 \equiv \mathtt{update} \circ \mathtt{lookup} \circ \langle\ \rangle_X$, by Point 2 in Lemma A.1 this implies $\mathtt{update} \circ a_1 \equiv \langle\ \rangle_X$. Thus, $\mathtt{update} \circ a_1 \equiv \langle\ \rangle_X$ if and only if $a_1 \equiv \mathtt{lookup} \circ \langle\ \rangle_X$.

3. Clearly, if $v_1 \equiv v_2$ then $a_1 \equiv a_2$. Conversely, if $a_1 \equiv a_2$, i.e., if $v_1 \circ \mathtt{lookup} \circ \langle\ \rangle_X \equiv v_2 \circ \mathtt{lookup} \circ \langle\ \rangle_X$, since $\langle\ \rangle_X$ is an epimorphism with respect to accessors we get $v_1 \circ \mathtt{lookup} \equiv v_2 \circ \mathtt{lookup}$. By Point 3 in Lemma A.1, this means that $v_1 \equiv v_2$.
4. Let $w_2^{(0)} = v_2 \circ k_X : \mathbb{1} \to Y$. Let us assume that $v_1 \equiv w_2 \circ \langle\ \rangle_V$ and $v_2 \equiv w_2 \circ \langle\ \rangle_X$. Equation $v_1 \equiv w_2 \circ \langle\ \rangle_V$ implies $a_1 \equiv w_2 \circ \langle\ \rangle_V \circ \mathtt{lookup} \circ \langle\ \rangle_X$. Since $\langle\ \rangle_V \circ \mathtt{lookup} \equiv id_{\mathbb{1}}$ we get $a_1 \equiv w_2 \circ \langle\ \rangle_X$. Then, equation $v_2 \equiv w_2 \circ \langle\ \rangle_X$ implies $a_1 \equiv v_2$. Conversely, let us assume that $a_1 \equiv v_2$, which means that $v_1 \circ \mathtt{lookup} \circ \langle\ \rangle_X \equiv v_2$. Then $v_1 \circ \mathtt{lookup} \circ \langle\ \rangle_X \circ k_X \circ \langle\ \rangle_V \equiv v_2 \circ k_X \circ \langle\ \rangle_V$, which reduces to $v_1 \circ \mathtt{lookup} \circ \langle\ \rangle_V \equiv w_2 \circ \langle\ \rangle_V$. Since $\langle\ \rangle_V$ is an epimorphism with respect to accessors we get $v_1 \circ \mathtt{lookup} \equiv w_2$, which means that $v_1 \equiv w_2 \circ \langle\ \rangle_V$ by Point 4 in Lemma A.1. Now let us come back to equation $v_1 \circ \mathtt{lookup} \circ \langle\ \rangle_X \equiv v_2$; since $v_1 \equiv w_2 \circ \langle\ \rangle_V$, it yields $w_2 \circ \langle\ \rangle_V \circ \mathtt{lookup} \circ \langle\ \rangle_X \equiv v_2$, so that $w_2 \circ \langle\ \rangle_X \equiv v_2$.

□

The assumption for Theorem A.6 comes form the fact that the existence of a pure term $k_X^{(0)} : \mathbb{1} \to X$, which is used in Point 4 of Proposition A.3, is incompatible with the intended model of states if X is interpreted as the empty set. The assumption for Theorem A.6 is *not* dual to the assumption for Theorem 4.5.

Definition A.4. A type X is *inhabited* if there exists a pure term $k_X^{(0)} : \mathbb{1} \to X$. A type $\mathbb{0}$ is *empty* if for each type Y there is a pure term $[\ ]_Y^{(0)} : \mathbb{0} \to Y$, and every term $f : \mathbb{0} \to Y$ is such that $f \equiv [\ ]_Y$.

Remark A.5. When X is inhabited then for any $k_X^{(0)} : \mathbb{1} \to X$ we have $\langle\ \rangle_X \circ k_X \equiv id_{\mathbb{1}}$, so that $\langle\ \rangle_X$ is a split epimorphism; it follows that $\langle\ \rangle_X$ is an epimorphism with respect to all terms, and especially with respect to accessors.

Theorem A.6. *If every non-empty type is inhabited and if V is non-empty, the theory of states T_{st} is relatively Hilbert-Post complete with respect to the pure sublogic $L_{st}^{(0)}$ of L_{st}.*

Proof. The proof relies upon Propositions A.2, A.3 and 2.7; it follows the same lines as the proofs of Theorems 3.5 and 4.5. The theory T_{st} is consistent: it cannot be proved that $\mathtt{update}^{(2)} \equiv \langle\ \rangle_V^{(0)}$ because the logic L_{st} is sound with respect to its intended model and the interpretation of this equation in the intended model is false as sson as V has at least two elements: indeed, for each state s and each $x \in V$, $\mathtt{lookup} \circ \mathtt{update}(x,s) = x$ because of (ax) while $\mathtt{lookup} \circ \langle\ \rangle_V(x,s) = \mathtt{lookup}(s)$ does not depend on x. Let us consider an equation (strong or weak) between terms with domain X in L_{st}; we distinguish two cases, whether X is empty or not. When X is empty, then all terms from X to Y are strongly equivalent to $[\ ]_Y$, so that the given equation is T_{st}-equivalent to the empty set of equations between pure terms. When X is non-empty then it is inhabited, thus by Remark A.5 $\langle\ \rangle_X$ is an epimorphism with respect to accessors. Thus, Propositions A.2 and A.3 prove that the given equation is T_{st}-equivalent to a finite set of equations between pure terms. Thus, in both cases, the result follows from Proposition 2.7. $\square$

Remark A.7. This can be generalized to an arbitrary number of locations. The logic L_{st} and the theory T_{st} have to be generalized as in [2], then Proposition A.2 has to be adapted using the basic properties of `lookup` and `update`, as stated in [12]; these properties can be deduced from the decorated theory for states, as proved in [6]. The rest of the proof generalizes accordingly, as in [11].