Hilbert-Post completeness for the state and the exception effects

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

October 8, 2015

Abstract

A theory is complete if it does not contain a contradiction, while all of its proper extensions do. In this paper, first we introduce a relative notion of syntactic completeness; then we prove that adding exceptions to a programming language can be done in such a way that the completeness of the language is not made worse. These proofs are formalized in a logical system which is close to the usual syntax for exceptions, and they have been checked with the proof assistant Coq.

1 Introduction

In computer science, an exception is an abnormal event occurring during the execution of a program. A mechanism for handling exceptions consists of two parts: an exception is *raised* when an abnormal event occurs, and it can be *handled* later, by switching the execution to a specific subprogram. Such a mechanism is very helpful, but it is difficult for programmers to reason about it. A difficulty for reasoning about programs involving exceptions is that they are *computational effects*, in the sense 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 interpreted as a function from $X+E$ to $Y+E$ where E is the set of exceptions. On the one hand, reasoning with $f : X \to Y$ is close to the syntax, but it is error-prone because it is not sound with respect to the semantics. On the other hand, reasoning with $f : X+E \to Y+E$ is sound but it loses most of the interest of the exception mechanism, where the propagation of exceptions is implicit: syntactically, $f : X \to Y$ may be followed by any $g : Y \to Z$,

*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.

since the mechanism of exceptions will take care of propagating the exceptions raised by f, if any. Another difficulty for reasoning about programs involving exceptions is that the handling mechanism is encapsulated in a `try-catch` block, while the behaviour of this mechanism is easier to explain in two parts (see for instance [10, Ch. 14] for Java or [3, §15] for C++): the `catch` part may recover from exceptions, so that its interpretation may be any $f : X + E \to Y + E$, but the `try-catch` block must propagate exceptions, so that its interpretation is determined by some $f : X \to Y + E$.

In [8] we defined a logical system for reasoning about states and exceptions and we used it for getting certified proofs of properties of programs in computer algebra, with an application to exact linear algebra. This logical system is called the *decorated logic* for states and exceptions. Here we focus on exceptions. The decorated logic for exceptions deals with $f : X \to Y$, without any mention of E, however it is sound thanks to a classification of the terms and the equations. Terms are classified, as in a programming language, according to the way they may interact with exceptions: a term either has no interaction with exceptions (it is "pure"), or it may raise exceptions and must propagate them, or it is allowed to catch exceptions (which may occur only inside the `catch` part of a `try-catch` block). The classification of equations follows a line that was introduced in [4]: besides the usual "strong" equations, interpreted as equalities of functions, in the decorated logic for exceptions there are also "weak" equations, interpreted as equalities of functions on non-exceptional arguments. This logic has been built so as to be sound, but little was known about its completeness. In this paper we prove a novel completeness result: the decorated logic for exceptions is *relatively Hilbert-Post complete*, which means that adding exceptions to a programming language can be done in such a way that the completeness of the language is not made worse. For this purpose, we first define and study the novel notion of *relative* Hilbert-Post completeness, which seems to be a relevant notion for the completeness of various computational effects: indeed, we prove that this notion is preserved when combining effects. Practically, this means that we have defined a decorated framework where reasoning about programs with and without exceptions are equivalent, in the following sense: if there exists an unprovable equation not contradicting the given decorated rules, then this equation is equivalent to a set of unprovable equations of the pure sublogic not contradicting its rules.

Informally, in classical logic, a consistent theory is one that does not contain a contradiction and a theory is complete if it is consistent, and none of its proper extensions is consistent. Now, the usual ("*absolute*") 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. Then, more formally, 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 [20, Def. 4].

All our completeness proofs have been verified with the Coq proof assistant.

First, this shows that it is possible to formally prove that programs involving exceptions comply to their specifications. Second, this is of help for improving the confidence in the results. Indeed, for a human prover, proofs in a decorated logic require some care: they look very much like familiar equational proofs, but the application of a rule may be subject to restrictions on the decoration of the premises of the rule. The use of a proof assistant in order to check that these unusual restrictions were never violated has thus proven to be quite useful. Then, many of the proofs we give in this paper require a structural induction. There, the correspondence between our proofs and their Coq counterpart was eased, as structural induction is also at the core of the design of Coq.

A major difficulty for reasoning about programs involving exceptions, and more generally computational effects, is that their syntax does not look like their interpretation: typically, a piece of program from X to Y is not interpreted as a function from X to Y, because of the effects. The best-known algebraic approach for dealing with this problem has been initiated by Moggi: an effect is associated to a monad T, in such a way that the interpretation of a program from X to Y is a function from X to $T(Y)$ [13]: typically, for exceptions, $T(Y) = Y + E$. Other algebraic approaches include effect systems [12], Lawvere theories [17], algebraic handlers [18], comonads [21, 15], dynamic logic [14], among others. Some completeness results have been obtained, for instance for (global) states [16] and for local states [19]. The aim of these approaches is to extend functional languages with tools for programming and proving side-effecting programs; implementations include Haskell [2], Idris [11], Eff [1], while Ynot [22] is a Coq library for writing and verifying imperative programs.

Differently, our aim is to build a logical system for proving properties of some families of programs written in widely used non-functional languages like Java or C++[1]. The salient features of our approach are that:
(1) The syntax of our logic is kept close to the syntax of programming languages. This is made possible by starting from a simple syntax without effect and by adding decorations, which often correspond to keywords of the languages, for taking the effects into account.
(2) We consider exceptions in two settings, the programming language and the core language. This enables for instance to separate the treatment, in proofs, of the matching between normal or exceptional behavior from the actual recovery after an exceptional behavior.

In Section 2 we introduce a *relative* notion of Hilbert-Post completeness in a logic L with respect to a sublogic L_0. Then in Section 3 we prove the relative Hilbert-Post completeness of a theory of exceptions based on the usual `throw` and `try-catch` statement constructors. We go further in Section 4 by establishing the relative Hilbert-Post completeness of a *core* theory for exceptions with individualized `TRY` and `CATCH` statement constructors, which is useful for expressing the behaviour of the `try-catch` blocks. All our completeness proofs have been verified with the Coq proof assistant and we therefore give the main

[1] For instance, a denotational semantics of our framework for exceptions, which relies on the common semantics of exceptions in these languages, was given in [8, § 4].

ingredients of the framework used for this verification and the correspondence between our Coq package and the theorems and propositions of this paper in Section 5.

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 containing 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. A 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 $F(G(T)) \subseteq T$. Until the end of Section 2, we consider: *a logic L_0, an extension L of L_0, and the associated Galois connection $F \dashv G$.*

Definition 2.4. 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. A theory T of L is *(relatively) Hilbert-Post complete with respect to L_0* if it is consistent and if any theory of L containing T is L_0-derivable from T.

Each theory T is L_0-derivable from itself, as $T = T + F(T_{min,0})$, 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, and Proposition 2.7 proves that relative completeness is well-suited to the combination of effects.

Lemma 2.5. *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 Hilbert-Post complete with respect to L_0 if and only if it is consistent and every theory T' of L containing 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. □

Theorem 2.6. *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 Hilbert-Post complete with respect to L_0, then T is Hilbert-Post complete (in L).*

Proof. Since T is 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 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 complete with respect to L_0, the theory T_{max} is L_0-derivable from T, which implies (by Lemma 2.5) that $T_{max} = T + F(T_{max,0}) = T'$. □

Proposition 2.7. *Let L_1 be an intermediate logic between L_0 and L, let $F_1 \dashv G_1$ and $F_2 \dashv G_2$ be the Galois connections associated to the extensions L_1 of L_0*

and L of L_1, respectively. Let $T_1 = F_1(T_0)$. If T_1 is Hilbert-Post complete with respect to L_0 and T is Hilbert-Post complete with respect to L_1 then T is Hilbert-Post complete with respect to L_0.

Proof. This is an easy consequence of the fact that $F = F_2 \circ F_1$. □

Corollary 2.10 provides a characterization of relative Hilbert-Post completeness which is used in the next Sections and in the Coq implementation.

Definition 2.8. For each set E of formulas let $Th(E)$ be the theory generated by E; and when $E = \{e\}$ let $Th(e) = Th(\{e\})$. Then two sets E_1, E_2 of formulas are *T-equivalent* if $T + Th(E_1) = T + Th(E_2)$; and a formula e of L is *L_0-derivable* from a theory T of L if $\{e\}$ is T-equivalent to E_0 for some set E_0 of formulas of L_0.

Proposition 2.9. *Let T be a theory of L. Each theory T' of L containing T is L_0-derivable from T if and only if each formula e in L is L_0-derivable from T.*

Proof. Let us assume that each theory T' of L containing T is L_0-derivable from T. Let e be a formula in L, let $T' = T + Th(e)$, and let T'_0 be a theory of L_0 such that $T' = T + F(T'_0)$. The definition of $Th(-)$ is such that $Th(T'_0) = F(T'_0)$, so that we get $T + Th(e) = T + Th(E_0)$ where $E_0 = T'_0$. Conversely, let us assume that each formula e in L is L_0-derivable from T. Let T' be a theory containing 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. □

Corollary 2.10. A theory T of L is Hilbert-Post complete with respect to L_0 if and only if it is consistent and for each formula e of L there is a set E_0 of formulas of L_0 such that $\{e\}$ is T-equivalent to E_0.

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 [8]. There, we proposed to deal with exceptions as a decorated effect: 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 the disjoint union operator and E is the set of exceptions. In this section, we prove the relative Hilbert-Post completeness of the decorated theory of exceptions in Theorem 3.5.

As in [8], 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 consider a translation of the `try-catch` construction in a more elementary language where some terms are *catchers*, which means that they may recover from 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 allows 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 $x \in X$. This 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 (more precisely, the composition of propagators is the Kleisli composition associated to the monad $X + E$ [13, § 1]). Then, exceptions may be classified according to their *name*, as in [8]. 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, first the definition of the *monadic equational logic* L_{eq} is recalled in Fig. 1, (as in [13], this terminology might be misleading: the logic is called *monadic* because all its operations are have exactly one argument, this is unrelated to the use of the *monad* of exceptions).

Terms are closed under composition:

$u_k \circ \cdots \circ u_1 : X_0 \to X_k$ for each $(u_i : X_{i-1} \to X_i)_{1 \le i \le k}$, and $id_X : X \to X$ when $k = 0$

Rules: (equiv) $\dfrac{u}{u \equiv u}$ $\dfrac{u \equiv v}{v \equiv u}$ $\dfrac{u \equiv v \quad v \equiv w}{u \equiv w}$

(subs) $\dfrac{u \colon X \to Y \quad v_1 \equiv v_2 \colon Y \to Z}{v_1 \circ u \equiv v_2 \circ u}$ (repl) $\dfrac{v_1 \equiv v_2 \colon X \to Y \quad w \colon Y \to Z}{w \circ v_1 \equiv w \circ v_2}$

Empty type $\mathbb{0}$ with terms $[\,]_Y : \mathbb{0} \to Y$ and rule: (initial) $\dfrac{u \colon \mathbb{0} \to Y}{u \equiv [\,]_Y}$

Figure 1: Monadic equational logic L_{eq} (with empty type)

A monadic equational logic is made of types, terms and operations, where all operations are unary, so that terms are simply paths. This constraint on arity will make it easier to focus on the completeness issue. For the same reason, we also assume that there is an *empty type* $\mathbb{0}$, which is defined as an *initial object*: for each Y there is a unique term $[\,]_Y : \mathbb{0} \to Y$ and each term $u^{(0)} : Y \to \mathbb{0}$ is the inverse of $[\,]_Y^{(0)}$. In the intended model, $\mathbb{0}$ is interpreted as the empty set.

Then, the monadic equational logic L_{eq} is extended to form the *decorated logic for exceptions* L_{exc} by applying the rules in Fig. 2, with the following intended meaning:

- (initial$_1$): the term $[\,]_Y$ is unique as a propagator, not only as a pure term.
- (propagate): exceptions are always propagated.
- (recover): the parameter used for throwing an exception may be recovered.
- (try): equations are preserved by the exceptions mechanism.
- (try$_0$): pure code inside `try` never triggers the code inside `catch`.
- (try$_1$): code inside `catch` is executed when an exception is thrown inside `try`.

The *theory of exceptions* T_{exc} is the theory of L_{exc} generated from some arbitrary consistent theory T_{eq} of L_{eq}; with the notations of Section 2, $T_{exc} = F(T_{eq})$. The soundness of the intended model follows: see [8, §5.1] and [6], which are based on the description of exceptions in Java [10, Ch. 14] or in C++ [3, §15].

Example 3.1. Using the naturals for P and the successor and predecessor functions (resp. denoted `s` and `p`) we can prove, e.g., that `try`(`s`(`throw` 3))`catch`(`p`) is equivalent to 2. Indeed, first the rule (propagate) shows that `s`(`throw` 3)) $\equiv$ `throw` 3, then the rules (try) and (try$_1$) rewrite the given term into `p`(3).

Now, in order to prove the completeness of the decorated theory for exceptions, we follow a classical method (see, e.g., [16, Prop 2.37 & 2.40]): we first determine canonical forms in Proposition 3.2, then we study the equations between terms in canonical form in Proposition 3.3.

Pure part: the logic L_{eq} with a distinguished type P

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

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

$(a_k \circ \cdots \circ a_1)^{(\max(d_1,\ldots,d_k))} : X_0 \to X_k$ for each $(a_i^{(d_i)} : X_{i-1} \to X_i)_{1 \le i \le k}$

with conversion from $u^{(0)} : X \to Y$ to $u^{(1)} : X \to Y$

Rules:

(equiv), (subs), (repl) for all decorations (initial$_1$) $\dfrac{a^{(1)} : \mathbb{0} \to Y}{a \equiv [\,]_Y}$

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

(propagate) $\dfrac{a^{(1)} : X \to Y}{a \circ \mathtt{throw}_X \equiv \mathtt{throw}_Y}$ (try) $\dfrac{a_1^{(1)} \equiv a_2^{(1)} : X \to Y \quad b^{(1)} : P \to Y}{\mathtt{try}(a_1)\mathtt{catch}(b) \equiv \mathtt{try}(a_2)\mathtt{catch}(b)}$

(try$_0$) $\dfrac{u^{(0)} : X \to Y \quad b^{(1)} : P \to Y}{\mathtt{try}(u)\mathtt{catch}(b) \equiv u}$ (try$_1$) $\dfrac{u^{(0)} : X \to P \quad b^{(1)} : P \to Y}{\mathtt{try}(\mathtt{throw}_Y \circ u)\mathtt{catch}(b) \equiv b \circ u}$

Figure 2: Decorated logic for exceptions L_{exc}

Proposition 3.2. *For each $a^{(1)} : X \to Y$, either there is a pure term $u^{(0)} : X \to Y$ such that $a \equiv u$ or there is a pure term $u^{(0)} : X \to P$ such that $a \equiv \mathtt{throw}_Y \circ u$.*

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 $a = b \circ \mathtt{op} \circ v$ where v is pure, $\mathtt{op}$ is either $\mathtt{throw}_Z$ for some Z or $\mathtt{try}(c)\mathtt{catch}(d)$ for some c and d, and b is the remaining part of a. If $a = b^{(1)} \circ \mathtt{throw}_Z \circ v^{(0)}$, then by (propagate) $a \equiv \mathtt{throw}_Y \circ v^{(0)}$. Otherwise, $a = b^{(1)} \circ (\mathtt{try}(c^{(1)})\mathtt{catch}(d^{(1)})) \circ v^{(0)}$, then by induction we consider two cases.

- 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 3.2, the study of equations in the logic L_{exc} can be restricted to pure terms and to propagators of the form $\mathtt{throw}_Y \circ v$ where v is pure.

Proposition 3.3. *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 $a_1^{(1)} \equiv a_2^{(1)}$ is T_{exc}-equivalent to $v_1^{(0)} \equiv v_2^{(0)}$.*

Proof. 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 $v_1 \equiv v_2$. □

In the intended model, for all $v_1^{(0)} : X \to P$ and $v_2^{(0)} : X \to Y$, it is impossible to have $\mathtt{throw}_Y(v_1(x)) = v_2(x)$ for some $x \in X$, because $\mathtt{throw}_Y(v_1(x))$ is in the E summand and $v_2(x)$ in the Y summand of the disjoint union $Y + E$. This means that the functions $\mathtt{throw}_Y \circ v_1$ and v_2 are distinct, as soon as their domain X is a non-empty set. For this reason, it is sound to make the following Assumption 3.4.

Assumption 3.4. In the logic L_{exc}, the type of parameters P is non-empty, and for all $v_1^{(0)} : X \to P$ and $v_2^{(0)} : X \to Y$ with X non-empty, let $a_1^{(1)} = \mathtt{throw}_Y \circ v_1 : X \to Y$. Then $a_1^{(1)} \equiv v_2^{(0)}$ is T_{exc}-equivalent to $T_{max,0}$.

Theorem 3.5. *Under Assumption 3.4, the theory of exceptions T_{exc} is Hilbert-Post complete with respect to the pure sublogic L_{eq} of L_{exc}.*

Proof. Using Corollary 2.10, the proof relies upon Propositions 3.2 and 3.3. The theory T_{exc} is consistent, because (by soundness) it cannot be proved that $\mathtt{throw}_P^{(1)} \equiv id_P^{(0)}$. 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. When X is non-empty, Propositions 3.2 and 3.3, together with Assumption 3.4, 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 $[\ ]_Y$ so that the given equation is T_{exc}-equivalent to the empty set of pure equations. □

4 Completeness of the core language for exceptions

In this section, following [8], we describe a translation of the language for exceptions from Section 3 in a *core* language with *catchers*. Thereafter, in Theorem 4.7, we state the relative Hilbert-Post completeness of this core language. 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 [10]) makes use of a *core language* for exceptions which is studied in [8]. 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 interpretation is extended as follows: each catcher $f^{(2)} : X \to Y$ is interpreted as a function $f : X + E \to Y + E$, and 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 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 an injective function $\mathtt{tag} : P \to E$ and the interpretation of $\mathtt{untag}^{(2)} : \mathbb{0} \to P$ is a 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 $\mathtt{untag} \circ \mathtt{tag} \sim id_P$.

Pure part: the logic L_{eq} with a distinguished type P

Decorated terms: $\mathtt{tag}^{(1)} \colon P \to \mathbb{0}$, $\mathtt{untag}^{(2)} \colon \mathbb{0} \to P$, 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$_\sim$), (repl$_\sim$) for all decorations, (subs$_\sim$) only when h is pure

(empty$_\sim$) $\dfrac{f \colon \mathbb{0} \to Y}{f \sim [\,]_Y}$ (≡-to-∼) $\dfrac{f \equiv g}{f \sim g}$ (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$_2$) $\dfrac{f_1, f_2 \colon X \to Y \quad f_1 \sim f_2 \quad f_1 \circ [\,]_X \equiv f_2 \circ [\,]_X}{f_1 \equiv f_2}$

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

Figure 3: Decorated logic for the core language for exceptions $L_{exc\text{-}core}$

More precisely, the *decorated logic for the core language for exceptions* $L_{exc\text{-}core}$ is defined in Fig. 3 as an extension of the monadic equational logic L_{eq}. There is an obvious conversion from strong to weak equations (≡-to-∼), 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 $f_1 \circ [\,]_X \circ \mathtt{tag} \sim f_2 \circ [\,]_X \circ \mathtt{tag}$ by rule (eq$_3$). The *core theory of exceptions* $T_{exc\text{-}core}$ is the theory of $L_{exc\text{-}core}$ generated from the theory T_{eq} of L_{eq}. Some easily derived properties are stated in Lemma 4.1; which will be used repeatedly.

Lemma 4.1. *1. For all pure terms $u_1^{(0)}, u_2^{(0)} : X \to P$, the equation $u_1 \equiv u_2$ is $T_{exc\text{-}core}$-equivalent to $\mathtt{tag} \circ u_1 \equiv \mathtt{tag} \circ u_2$ and also to $\mathtt{untag} \circ \mathtt{tag} \circ u_1 \equiv$*

$\mathtt{untag} \circ \mathtt{tag} \circ u_2$.

2. *For all pure terms* $u^{(0)} : X \to P$, $v^{(0)} : X \to \mathbb{0}$, *the equation* $u \equiv [\,]_P \circ v$ *is* $T_{exc\text{-}core}$*-equivalent to* $\mathtt{tag} \circ u \equiv v$.

Proof. 1. 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 $u_1 \equiv u_2$.

2. 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 1 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 a translation of the programmer's language in the core language, as in [8], which is reminded below; then Proposition 4.2 proves the correctness of this translation. In view of this translation 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 ran.
- 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, a translation of the programmer's language of exceptions in the core language is easily obtained: for each type Y, $\mathtt{throw}_Y^{(1)} = [\,]_Y \circ \mathtt{tag} : P \to Y$. and for each $a^{(1)} : X \to Y$, $b^{(1)} : P \to Y$, $(\mathtt{try}(a)\mathtt{catch}(b))^{(1)} = \mathtt{TRY}(a, \mathtt{CATCH}(b)) : X \to Y$. This translation is correct: see Proposition 4.2.

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 translation of the programmers' language for exceptions in the core language is correct.*

Proof. We have to prove that the image of each rule of L_{exc} is satisfied. It should be reminded that strong and weak equations coincide on L_{exc}.

- (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 1 in Lemma 4.1, we get $u_1 \equiv u_2$.

- (try) Since $\mathtt{try}(a_i)\mathtt{catch}(b) \sim \mathtt{catch}(b) \circ a_i$ for $i \in \{1,2\}$, we get $\mathtt{try}(a_1)\mathtt{catch}(b) \sim \mathtt{try}(a_2)\mathtt{catch}(b)$ as soon as $a_1 \equiv a_2$.
- (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 $\mathtt{TRY}(u, \mathtt{CATCH}(b)) \sim u$.
- (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 $\mathtt{TRY}([\,]_Y \circ \mathtt{tag} \circ u, \mathtt{CATCH}(b)) \sim b \circ u$.

□

Example 4.3 (Continuation of Example 3.1). We here show that it is possible to separate the matching between normal or exceptional behavior from the recovery after an exceptional behavior: to prove that $\mathtt{try}(\mathtt{s}(\mathtt{throw}\ 3))\mathtt{catch}(\mathtt{p})$ is equivalent to 2 in the core language, we first use the translation to get: $\mathtt{TRY}(\mathtt{s} \circ [\,] \circ \mathtt{tag} \circ 3, \mathtt{CATCH}(\mathtt{p}))$. Then ($\mathrm{empty}_\sim$) shows that $\mathtt{s} \circ [\,]\mathtt{tag} \circ 3 \sim [\,] \circ \mathtt{tag} \circ 3$. Now, the TRY and CATCH translations show that $\mathtt{TRY}([\,] \circ \mathtt{tag} \circ 3, \mathtt{CATCH}(\mathtt{p})) \sim \mathtt{CATCH}(\mathtt{p}) \circ [\,] \circ \mathtt{tag} \circ 3 \sim \mathtt{p} \circ \mathtt{untag} \circ \mathtt{tag} \circ 3$. Finally the axiom (ax) and (eq_1) give $\mathtt{p} \circ 3 \equiv 2$.

In order to prove the completeness of the core decorated theory for exceptions, as for the proof of Theorem 3.5, we first determine canonical forms in Proposition 4.4, then we study the equations between terms in canonical form in Proposition 4.5. Let us begin by proving the *fundamental strong equation for exceptions* (1): by replacement in the axiom (ax) we get $\mathtt{tag} \circ \mathtt{untag} \circ \mathtt{tag} \sim \mathtt{tag}$, then by rule ($\mathrm{eq}_3$):

$$\mathtt{tag} \circ \mathtt{untag} \equiv id_{\mathbb{0}} \tag{1}$$

Proposition 4.4. *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 4.4, 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 4.5. *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 $f_1 \sim f_2$ is $T_{exc\text{-}core}$-equivalent to $a_1 \circ u_1 \equiv a_2 \circ u_2$ and $f_1 \equiv f_2$ is $T_{exc\text{-}core}$-equivalent to $\{a_1 \equiv a_2\ ,\ a_1 \circ u_1 \equiv a_2 \circ u_2\}$.*

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 $f_1 \sim a_2$ is $T_{exc\text{-}core}$-equivalent to $a_1 \circ u_1 \equiv a_2$ and $f_1 \equiv a_2$ is $T_{exc\text{-}core}$-equivalent to $\{a_1 \circ u_1 \equiv a_2\ ,\ a_1 \equiv [\,]_Y \circ \mathtt{tag}\}$.

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 $a_1 \equiv a_2$ is $T_{exc\text{-}core}$-equivalent to $v_1 \equiv v_2$.

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 $\mathtt{tag} \circ u_i \circ [\,]_X : \mathbb{0} \to \mathbb{0}$ satisfies $\mathtt{tag} \circ u_i \circ [\,]_X \equiv id_{\mathbb{0}}$, so that $f_i \circ [\,]_X \equiv a_i \circ \mathtt{untag}$. Thus, $f_1 \circ [\,]_X \equiv f_2 \circ [\,]_X$ if and only if $a_1 \circ \mathtt{untag} \equiv a_2 \circ \mathtt{untag}$. Clearly, if $a_1 \equiv a_2$ then $a_1 \circ \mathtt{untag} \equiv a_2 \circ \mathtt{untag}$. Conversely, if $a_1 \circ \mathtt{untag} \equiv a_2 \circ \mathtt{untag}$

then $a_1 \circ \mathtt{untag} \circ \mathtt{tag} \equiv a_2 \circ \mathtt{untag} \circ \mathtt{tag}$, 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 $a_1 \equiv [\,]_Y \circ \mathtt{tag}$, in two steps. Since $a_2 \circ [\,]_X : \mathbb{0} \to Y$ is a propagator, we have $a_2 \circ [\,]_X \equiv [\,]_Y$. Since $f_1 \circ [\,]_X = a_1 \circ \mathtt{untag} \circ \mathtt{tag} \circ u_1 \circ [\,]_X$ with $\mathtt{tag} \circ u_1 \circ [\,]_X : \mathbb{0} \to \mathbb{0}$ a propagator, we have $\mathtt{tag} \circ u_1 \circ [\,]_X \equiv id_{\mathbb{0}}$ and thus we get $f_1 \circ [\,]_X \equiv a_1 \circ \mathtt{untag}$. Thus, $f_1 \circ [\,]_X \equiv a_2 \circ [\,]_X$ if and only if $a_1 \circ \mathtt{untag} \equiv [\,]_Y$. If $a_1 \circ \mathtt{untag} \equiv [\,]_Y$ then $a_1 \circ \mathtt{untag} \circ \mathtt{tag} \equiv [\,]_Y \circ \mathtt{tag}$, by (ax) and (repl$_\sim$) this implies $a_1 \sim [\,]_Y \circ \mathtt{tag}$, which is a strong equality because both members are propagators. Conversely, if $a_1 \equiv [\,]_Y \circ \mathtt{tag}$ then $a_1 \circ \mathtt{untag} \equiv [\,]_Y \circ \mathtt{tag} \circ \mathtt{untag}$, by the fundamental equation (1) this implies $a_1 \circ \mathtt{untag} \equiv [\,]_Y$. Thus, $a_1 \circ \mathtt{untag} \equiv [\,]_Y$ if and only if $a_1 \equiv [\,]_Y \circ \mathtt{tag}$.

3. Clearly, if $v_1 \equiv v_2$ then $[\,]_Y \circ \mathtt{tag} \circ v_1 \equiv [\,]_Y \circ \mathtt{tag} \circ v_2$. Conversely, if $[\,]_Y \circ \mathtt{tag} \circ v_1 \equiv [\,]_Y \circ \mathtt{tag} \circ v_2$ then since $[\,]_Y$ is a monomorphism with respect to propagators we get $\mathtt{tag} \circ v_1 \equiv \mathtt{tag} \circ v_2$, so that $\mathtt{untag} \circ \mathtt{tag} \circ v_1 \equiv \mathtt{untag} \circ \mathtt{tag} \circ v_2$. Now, from (ax) we get $v_1 \sim v_2$, which means that $v_1 \equiv v_2$ because v_1 and v_2 are pure.

□

Assumption 4.6 is the image of Assumption 3.4 by the above translation.

Assumption 4.6. In the logic $L_{exc\text{-}core}$, the type of parameters P is non-empty, and for all $v_1^{(0)} : X \to P$ and $v_2^{(0)} : X \to Y$ with X non-empty, let $a_1^{(1)} = [\,]_Y \circ \mathtt{tag} \circ v_1 : X \to Y$. Then $a_1^{(1)} \equiv v_2^{(0)}$ is T_{exc}-equivalent to $T_{max,0}$.

Theorem 4.7. *Under Assumption 4.6, the theory of exceptions $T_{exc\text{-}core}$ is Hilbert-Post complete with respect to the pure sublogic L_{eq} of $L_{exc\text{-}core}$.*

Proof. Using Corollary 2.10, the proof is based upon Propositions 4.4 and 4.5. 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, because (by soundness) it cannot be proved that $\mathtt{untag}^{(2)} \equiv [\,]_P^{(0)}$. Now, let us consider an equation between terms $f_1, f_2 : X \to Y$, and let us prove that it is $T_{exc\text{-}core}$-equivalent to a set of pure equations. When X is non-empty, Propositions 4.4 and 4.5, together with Assumption 4.6, prove that the given equation is $T_{exc\text{-}core}$-equivalent to a set of pure equations. When X is empty, then $f_1 \sim [\,]_Y$ and $f_2 \sim [\,]_Y$, so that if the equation is weak or if both f_1 and f_2 are propagators then the given equation is $T_{exc\text{-}core}$-equivalent to the empty set of equations between pure terms. When X is empty and the equation is $f_1 \equiv f_2$ with at least one of f_1 and f_2 a catcher, then by Point 1 or 2 of Proposition 4.5, the given equation is $T_{exc\text{-}core}$-equivalent to a set of equations between propagators; but 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 these sets of pure equations. □

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 `http://forge.imag.fr/frs/download.php/680/hp-0.7.tar.gz`, as well as an almost dual proof for the completeness of the state. They share the same framework, defined in [9]:

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 → 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.

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	execution time in Coq
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_trans	Translation.v	4 KB	2 KB	1.696 sec.
st-hp	HPCompleteCoq.v	48 KB	15 KB	7.183 sec.

The correspondence between the propositions and theorems in this paper and their proofs in Coq is given in Fig. 4, and the dependency chart for the main results in Fig. 5. For instance, Proposition 3.3 is expressed in Coq as:

```
forall {X Y} (a1 a2: term X Y) (v1 v2: term (Val e) Y),
    (is pure v1) /\ (is pure v2) /\
    (a1 = ((@throw X e) o v1)) /\ (a2 = ((@throw X e) o v2)) -> ((a1 == a2) <-> (v1 == v2)).
```

hp-0.7/exc_trans/Translation.v	
Proposition 4.2 (propagate)	`propagate`
Proposition 4.2 (recover)	`recover`
Proposition 4.2 (try)	`try`
Proposition 4.2 (try_0)	$\texttt{try}_0$
Proposition 4.2 (try_1)	$\texttt{try}_1$

hp-0.7/exc_pl-hp/HPCompleteCoq.v	
Proposition 3.2	`can_form_th`
Proposition 3.3	`eq_th_1_eq_pu`
Assumption 3.4	`eq_th_pu_abs`
Theorem 3.5	`HPC_exc_pl`

hp-0.7/exc_cl-hp/HPCompleteCoq.v	
Proposition 4.4 Point 1	`can_form_pr`
Proposition 4.4 Point 2	`can_form_ca`
Assumption 4.6	`eq_pr_pu_abs`
Proposition 4.5 Point 1	`eq_ca_2_eq_pr`
Proposition 4.5 Point 2	`eq_ca_pr_2_eq_pr`
Proposition 4.5 Point 3	`eq_pr_1_eq_pu`
Theorem 4.7	`HPC_exc_core`

Figure 4: Correspondence between theorems in this paper and their Coq counterparts

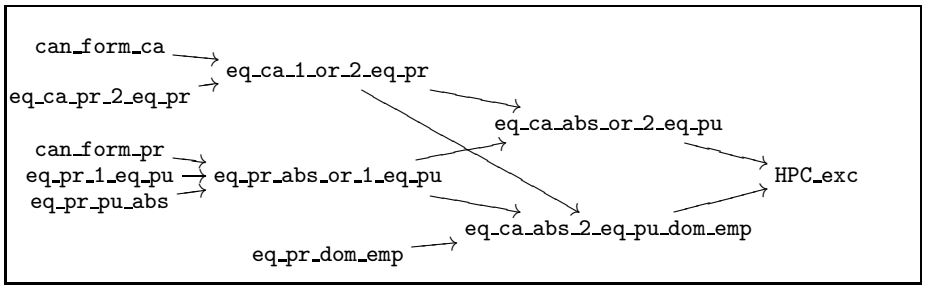


Figure 5: Dependency chart for the main results

6 Conclusion and future work

This paper is a first step towards the proof of completeness of decorated logics for computer languages. It has to be extended in several directions: adding basic features to the language (arity, conditionals, loops, ...), proving completeness of the decorated approach for other effects (not only states and exceptions); the combination of effects should easily follow, thanks to Proposition 2.7.

References

[1] Andrej Bauer, Matija Pretnar. Programming with algebraic effects and handlers. J. Log. Algebr. Meth. Program. 84, p. 108-123 (2015).

[2] Nick Benton, John Hughes, Eugenio Moggi. Monads and Effects. APPSEM 2000, LNCS, Vol. 2395, p. 42-122 (2000).

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

[4] César Domínguez, Dominique Duval. Diagrammatic logic applied to a parameterisation process. Math. Structures in Computer Science 20, p. 639-654 (2010).

[5] 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).

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

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

[8] Jean-Guillaume Dumas, Dominique Duval, Burak Ekici and Jean-Claude Reynaud. Certified proofs in programs involving exceptions. CICM 2014, CEUR Workshop Proceedings 1186 (2014).

[9] Jean-Guillaume Dumas, Dominique Duval, Burak Ekici, Damien Pous. Formal verification in Coq of program properties involving the global state. JFLA 2014, p. 1-15 (2014).

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

[11] Idris. The Effects Tutorial.

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

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

[14] Till Mossakowski, Lutz Schröder, Sergey Goncharov. A generic complete dynamic logic for reasoning about purity and effects. Formal Aspects of Computing 22, p. 363-384 (2010).

[15] Tomas Petricek, Dominic A. Orchard, Alan Mycroft. Coeffects: a calculus of context-dependent computation. Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, p. 123-135 (2014).

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

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

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

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

[20] 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).

[21] Tarmo Uustalu, Varmo Vene. Comonadic Notions of Computation. Electr. Notes Theor. Comput. Sci. 203, p. 263-284 (2008).

[22] Ynot. The Ynot Project.

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 [9], 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 noticed in [8], the logic $L_{exc\text{-}core}$ is exactly dual to the logic L_{st} for states (as reminded below). Thus, the dual of the completeness Theorem 4.7 and of all results in Section 4 are valid, with the dual proof. 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 4.7, 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 4.7. 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 4.7.

As in [5], 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. 6. 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. 6.

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 [5].

Here, for the sake of simplicity, we consider a single variable (as done, e.g., in [16] and [19]), 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 (eq_2). This can be expressed as a pair of weak equations $f_1 \sim f_2$ and $\texttt{lookup} \circ \langle\ \rangle_Y \circ f_1 \sim \texttt{lookup} \circ \langle\ \rangle_Y \circ f_2$, by rule ($\text{eq}_3$). Some easily derived properties are stated in Lemma A.1; Point 2 will be used repeatedly.

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 6: Decorated logic for states (dual to Fig. 3)

Lemma A.1. *1.* $\texttt{update} \circ \texttt{lookup} \equiv id_{\mathbb{1}}$. *(this is the fundamental strong equation for states).*

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

3. For all pure terms $u_1^{(0)}, u_2^{(0)} : V \to Y$, *one has:* $u_1 \equiv u_2$ *is* T_{st}*-equivalent to* $u_1 \circ \texttt{lookup} \equiv u_2 \circ \texttt{lookup}$ *and also to* $u_1 \circ \texttt{lookup} \circ \texttt{update} \equiv u_2 \circ \texttt{lookup} \circ \texttt{update}$.

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

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

2. Clear.

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 $\langle\ \rangle_V \circ \mathtt{lookup} : \mathbb{1} \to \mathbb{1}$ is an accessor we have $\langle\ \rangle_V \circ \mathtt{lookup} \equiv id_{\mathbb{1}}$. Now, if $u \equiv v \circ \langle\ \rangle_V$ then $u \circ \mathtt{lookup} \equiv v \circ \langle\ \rangle_V \circ \mathtt{lookup}$, so that $u \circ \mathtt{lookup} \equiv v$. Conversely, if $u \circ \mathtt{lookup} \equiv v$ then $u \circ \mathtt{lookup} \equiv v \circ \langle\ \rangle_V \circ \mathtt{lookup}$, 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 1 in Proposition 4.4

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 $f_1^{(2)} = u_1 \circ \mathtt{lookup} \circ \mathtt{update} \circ a_1 : X \to Y$ and $f_2^{(2)} = u_2 \circ \mathtt{lookup} \circ \mathtt{update} \circ a_2 : X \to Y$, then $f_1 \sim f_2$ is T_{st}-equivalent to $u_1 \circ a_1 \equiv u_2 \circ a_2$ and $f_1 \equiv f_2$ is T_{st}-equivalent to $\{a_1 \equiv a_2\ ,\ u_1 \circ a_1 \equiv u_2 \circ a_2\}$.*

2. For all $a_1^{(1)} : X \to V$, $u_1^{(0)} : V \to Y$ and $a_2^{(1)} : X \to Y$, let $f_1^{(2)} = u_1 \circ \mathtt{lookup} \circ \mathtt{update} \circ a_1 : X \to Y$, then $f_1 \sim a_2$ is T_{st}-equivalent to $u_1 \circ a_1 \equiv a_2$ $f_1 \equiv a_2$ is T_{st}-equivalent to $\{u_1 \circ a_1 \equiv a_2\ ,\ a_1 \equiv \mathtt{lookup} \circ \langle\ \rangle_X\}$.

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 $a_1^{(1)} = v_1 \circ \mathtt{lookup} \circ \langle\ \rangle_X : X \to Y$ and $a_2^{(1)} = v_2 \circ \mathtt{lookup} \circ \langle\ \rangle_X : X \to Y$. Then $a_1 \equiv a_2$ is T_{st}-equivalent to $v_1 \equiv v_2$.

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 $a_1^{(1)} = v_1 \circ \mathtt{lookup} \circ \langle\ \rangle_X : X \to Y$. Then $a_1 \equiv v_2$ is T_{st}-equivalent to $\{v_1 \equiv v_2 \circ k_X \circ \langle\ \rangle_V\ ,\ v_2 \equiv v_2 \circ k_X \circ \langle\ \rangle_X\}$.

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 $\langle\ \rangle_Y \circ u_i \circ \mathtt{lookup} : \mathbb{1} \to \mathbb{1}$ satisfies $\langle\ \rangle_Y \circ u_i \circ \mathtt{lookup} \equiv id_{\mathbb{1}}$, so that $\langle\ \rangle_Y \circ f_i \equiv \mathtt{update} \circ a_i$. Thus, $\langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ f_2$ if and only if $\mathtt{update} \circ a_1 \equiv \mathtt{update} \circ a_2$.
- Clearly, if $a_1 \equiv a_2$ then $\mathtt{update} \circ a_1 \equiv \mathtt{update} \circ a_2$. Conversely, if $\mathtt{update} \circ a_1 \equiv \mathtt{update} \circ a_2$ then $\mathtt{lookup} \circ \mathtt{update} \circ a_1 \equiv \mathtt{lookup} \circ \mathtt{update} \circ a_2$, 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 $a_1 \equiv \mathtt{lookup} \circ \langle\ \rangle_X$, 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 $\langle\ \rangle_Y \circ f_1 = \langle\ \rangle_Y \circ u_1 \circ \texttt{lookup} \circ \texttt{update} \circ a_1$ with $\langle\ \rangle_Y \circ u_1 \circ \texttt{lookup} : \mathbb{1} \to \mathbb{1}$ an accessor, we have $\langle\ \rangle_Y \circ u_1 \circ \texttt{lookup} \equiv id_{\mathbb{1}}$ and thus we get $\langle\ \rangle_Y \circ f_1 \equiv \texttt{update} \circ a_1$. Thus, $\langle\ \rangle_Y \circ f_1 \equiv \langle\ \rangle_Y \circ a_2$ if and only if $\texttt{update} \circ a_1 \equiv \langle\ \rangle_X$.
- If $\texttt{update} \circ a_1 \equiv \langle\ \rangle_X$ then $\texttt{lookup} \circ \texttt{update} \circ a_1 \equiv \texttt{lookup} \circ \langle\ \rangle_X$, by (ax) and (subs$_\sim$) this implies $a_1 \sim \texttt{lookup} \circ \langle\ \rangle_X$, which is a strong equality because both members are accessors. Conversely, if $a_1 \equiv \texttt{lookup} \circ \langle\ \rangle_X$ then $\texttt{update} \circ a_1 \equiv \texttt{update} \circ \texttt{lookup} \circ \langle\ \rangle_X$, by Point 1 in Lemma A.1 this implies $\texttt{update} \circ a_1 \equiv \langle\ \rangle_X$. Thus, $\texttt{update} \circ a_1 \equiv \langle\ \rangle_X$ if and only if $a_1 \equiv \texttt{lookup} \circ \langle\ \rangle_X$.

3. Clearly, if $v_1 \equiv v_2$ then $a_1 \equiv a_2$. Conversely, if $a_1 \equiv a_2$, i.e., if $v_1 \circ \texttt{lookup} \circ \langle\ \rangle_X \equiv v_2 \circ \texttt{lookup} \circ \langle\ \rangle_X$, since $\langle\ \rangle_X$ is an epimorphism with respect to accessors we get $v_1 \circ \texttt{lookup} \equiv v_2 \circ \texttt{lookup}$. 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 $a_1 \equiv w_2 \circ \langle\ \rangle_V \circ \texttt{lookup} \circ \langle\ \rangle_X$. Since $\langle\ \rangle_V \circ \texttt{lookup} \equiv id_{\mathbb{1}}$ 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 $v_1 \circ \texttt{lookup} \circ \langle\ \rangle_X \equiv v_2$. Then $v_1 \circ \texttt{lookup} \circ \langle\ \rangle_X \circ k_X \circ \langle\ \rangle_V \equiv v_2 \circ k_X \circ \langle\ \rangle_V$, which reduces to $v_1 \circ \texttt{lookup} \circ \langle\ \rangle_V \equiv w_2 \circ \langle\ \rangle_V$. Since $\langle\ \rangle_V$ is an epimorphism with respect to accessors we get $v_1 \circ \texttt{lookup} \equiv w_2$, 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 $v_1 \circ \texttt{lookup} \circ \langle\ \rangle_X \equiv v_2$; since $v_1 \equiv w_2 \circ \langle\ \rangle_V$, it yields $w_2 \circ \langle\ \rangle_V \circ \texttt{lookup} \circ \langle\ \rangle_X \equiv v_2$, 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 4.7.

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 Hilbert-Post complete with respect to the pure sublogic $L_{st}^{(0)}$ of L_{st}.*

Proof. Using Corollary 2.10, the proof relies upon Propositions A.2 and A.3. it follows the same lines as the proofs of Theorems 3.5 and 4.7. 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. □

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 [5], then Proposition A.2 has to be adapted using the basic properties of `lookup` and `update`, as stated in [17]; these properties can be deduced from the decorated theory for states, as proved in [9]. The rest of the proof generalizes accordingly, as in [16].