A generic complete dynamic logic for reasoning about purity and effects

Till Mossakowski, Lutz Schröder, Sergey Goncharov

To appear in Formal Aspects of Computing

A Generic Complete Dynamic Logic for Reasoning about Purity and Effects

Till Mossakowski[1,2] and Lutz Schröder[1,2] and Sergey Goncharov[2]

[1] DFKI GmbH, Safe and secure cognitive systems, Bremen
[2] Department of Computer Science, University of Bremen

Abstract. For a number of programming languages, among them Eiffel, C, Java, and Ruby, Hoare-style logics and dynamic logics have been developed. In these logics, pre- and postconditions are typically formulated using potentially effectful programs. In order to ensure that these pre- and postconditions behave like logical formulae (that is, enjoy some kind of referential transparency), a notion of purity is needed. Here, we introduce a generic framework for reasoning about purity and effects. Effects are modelled abstractly and axiomatically, using Moggi's idea of encapsulation of effects as monads. We introduce a dynamic logic (from which, as usual, a Hoare logic can be derived) whose logical formulae are pure programs in a strong sense. We formulate a set of proof rules for this logic, and prove it to be complete with respect to a categorical semantics. Using dynamic logic, we then develop a relaxed notion of purity which allows for observationally neutral effects such writing on newly allocated memory.

Keywords: monads for effects; dynamic logic; observational equivalence; completeness

1. Introduction

Design and programming by contract reduces software errors by providing specifications of Hoare-style pre- and postconditions and invariants along with the program. Originating in the Eiffel language [Mey92], this paradigm has become a guiding design principle for a number of languages, including Sather [Omo91], Lisaac [SC02], Nice [BK03], and D [Bri02]. Moreover, for many existing languages, among them C, C++, Java, JavaScript, Scheme, Perl, Python, and Ruby, libraries, preprocessors and other tools have been developed that support programming by contract; for example, the Java modelling language JML comes with extended static analysis [CK05] and verification [vdBJ01] tools. Contracts in a higher-order functional setting are treated in [FF02].

Unlike Hoare's original logic [Hoa69] and unlike some known formalisations of Hoare-logics in theorem provers like Isabelle and PVS [Hui01, vO01, Nip02], most of these languages do not have a separate language for expressing pre- and postconditions. Instead, these are expressed in a *pure subset* of the programming language itself (this idea originates from JML [LBR06]). The restriction to a pure subset is necessary since specifications should not have side-effects; this serves both conceptual clarity and the possibility of run-time testing (which could easily yield wrong results if specifications could change the behaviour of a program).

Purity in this sense is closely related to referential transparency. Informally, pure programs have the following properties.

1. Discardability: Pure computations can be left out from a sequence of computation steps without changing its behaviour;
2. Determinism: Pure programs always return the same value;
3. Interchangeability: Pure programs can be interchanged with each other, that is, the order does not matter.

Of course, these properties depend on a suitable notion of observational equivalence of programs that explains the term "without changing its behaviour". The observational equivalence in turn depends on the specific set of possible operations; for example, allocating a new memory cell and writing a value into it will not be observable in Java, while in C, it can be observed if the integer representation of the memory cell's address is guessed or known by chance and converted from an integer to a pointer, an operation which is not available in Java.

The Hoare logics for the above-mentioned languages, while all following essentially the same style, may vary in subtle but important details. This greatly complicates the study of properties of such logics. Consequently, the goal of this paper is to formalise these notions and provide logical rules for Hoare-style reasoning in a way that abstracts from the details of the particular languages. The abstraction is achieved by encapsulating notions of effect and computation as monads, as originally suggested by Moggi [Mog91] and used in the design of the functional programming language Haskell [PJ03] as well as in programming semantics, e.g. for Java [JP03]. In earlier work [SM03, SM04a], we have developed a dynamic logic for generic side effects whose semantics is defined in terms of monads; in particular, we have given a sound proof calculus for this logic.[1] Here, we extend the proof calculus to a calculus which is strongly complete for the generic part of the calculus and linear sequential programs[2]. This setting is sufficient for our treatment of observational purity. Our logic is related to Pitts' evaluation logic [Pit91], which in turn essentially puts the language-specific logic of [Boe85] on a generic basis. Both these logics omit loops, like the restricted logic considered here, but unlike the full logic of [SM04a].

In more detail, the definition of our dynamic logic is based on a strict notion of purity where the above requirements for referential transparency (discardability, determinism, interchangeability) are postulated to hold as actual equalities. Once we have the logic available, we can define a weaker notion of *observational purity* where referential transparency holds only up to observational equivalence in the logic. We thus arrive at a practically usable notion of purity, which in particular allows harmless side-effects such as creating and writing on new references. This generic concept of observational purity is related to the programming language specific notion put forward in [Nau07]. An interesting point here is that as a byproduct of our completeness proof, we obtain the existence of fully abstract categorical models, where observational equalities hold on the nose, and in particular all observationally pure functions are pure in the original strict sense.

Besides the specification and verification of monadic programs, our logic serves also the *specification* of monads, i.e. notions of side effect and purity. That is, effects may be described in an axiomatic manner, which abstracts away from particular details of the implementation, and moreover allows for loose specifications that deliberately leave open particular design decisions.

This paper is an extended version of [MSG08]. It is organised as follows. Sect. 2 recalls Moggi's approach of encapsulating computational effects as monads. Sect. 3 introduces syntax and semantics of monad-based dynamic logic. A suitable proof calculus is provided in Sect. 4, including its soundness proof. The basic logic cannot be directly applied to e.g. Java, because it makes too many distinctions. Hence, Sect. 5 introduces observational equivalence and purity. Sect. 6 provides a completeness proof for the proof calculus, using a fully abstract term model. Sect. 7 concludes the paper.

2. Monads for Computations

The presence of computational effects, e.g. state, store, exceptions, input, output, non-determinism, or backtracking, substantially complicates reasoning about programs. Dealing with such features has in the past typically required dedicated logics, designed specifically for a particular combination of effects. In seminal work by Moggi [Mog91], monads have been established as a unifying framework for modelling computational effects (including in particular all effects mentioned above) in an elegant way.

Intuitively, a monad associates to each type A a type TA of computations of type A; a function with side effects

[1] There are two dynamic logics for Java, one for the KIV prover [Ste04], and on in the KeY project [Bec01]. However, they neither address purity, nor are they generic.

[2] The original calculus does feature generic loop constructs; however, absolute completeness results such as the one proved here are impossible for dynamic logics over Turing complete languages: this would entail recursive enumerability of the set of non-terminating programs.

that takes inputs of type A and returns values of type B is, then, just a function of type $A \to TB$. This approach abstracts from particular notions of computation such as store, non-determinism, non-termination etc.; a surprisingly large amount of reasoning can be carried out without commitment to a particular type of side effect.

A monad on a given category $\mathbf{C}$ can be defined as a *Kleisli triple* $\mathbb{T} = (T, \eta, \_^*)$, where $T : \mathrm{Ob}\,\mathbf{C} \to \mathrm{Ob}\,\mathbf{C}$ is a function, the *unit* η is a family of morphisms $\eta_A : A \to TA$, and $\_^*$ assigns to each morphism $f : A \to TB$ a morphism $f^* : TA \to TB$ such that

$$\eta_A^* = id_{TA}, \quad f^*\eta_A = f, \quad \text{and} \quad g^*f^* = (g^*f)^*.$$

This description is equivalent to the more familiar one [Mac97].

In order to support a language with finitary operations and multi-variable contexts (see below), one needs a further ingredient: a monad is called *strong* if it is equipped with a natural transformation

$$t_{A,B} : A \times TB \to T(A \times B)$$

called *strength*, subject to certain coherence conditions (see e.g. [Mog91]).

Example 2.1 ([Mog91]). Computationally relevant monads on $\mathbf{Set}$ (all monads on $\mathbf{Set}$ are strong [SM04a, Mog91]) include the following.

1. *Stateful computations with non-termination:* $TA = S \to ?\ (A \times S)$, where S is a fixed set of states and $\_ \to ?\ \_$ denotes the partial function type.
2. *Non-determinism:* $TA = \mathcal{P}(A)$, where $\mathcal{P}$ is the covariant power set functor.
3. *Exceptions*: $TA = A + E$, where E is a fixed set of exceptions.
4. a) *Interactive input:* TA is the smallest fixed point of $\gamma \mapsto A + (U \to \gamma)$, where U is a set of input values. b) *Interactive output:* TA is the smallest fixed point of $\gamma \mapsto A + (U \times \gamma)$, where U is a set of output values.
5. *Non-deterministic stateful computations:* $TA = (S \to \mathcal{P}(A \times S))$, where, again, S is a fixed set of states (here, we can use the total function arrow since the binding operator can treat undefinedness as the empty set of results).
6. These monads can also be combined. E.g. non-deterministic stateful computations are obtained as $TA = S \to \mathcal{P}(A \times S)$. A monad for Java has been defined as follows [JP03]

$$JA = S \to (A \times S + E \times S + 1),$$

with S the set of states and E the set of exceptions. A state typically will comprise a stack, a heap, and a heap pointer, that is,

$$S = V^* \times (\mathit{Loc} \mapsto V) \times \mathit{Loc} \times \cdots$$

where V is a set of values and Loc a set of locations. While Java will provide an operation $new : J\ \mathit{Loc}$ that allocates a new location, C will additionally provide a coercion operation $\mathit{int2loc} : \mathit{Int} \to \mathit{Loc}$ (written e.g. as "`(*char)`" in C, if characters are stored).

3. Monad-Based Dynamic Logic

In program specification, dynamic logic as introduced in [Pra76] and extended to monadic computations in [SM04a] has a number of advantages over less expressive formalisms such as Hoare logic, among them the ability to express both partial and total correctness in a natural way and the possibility of reusing a state, say for statements of the nature 'what would happen if'. Here, we examine the infrastructure that is needed in order to develop generic monad-based dynamic logic, and illustrate that this does indeed make sense when instantiated to typical concrete computational monads.

3.1. Syntax

We begin by fixing the syntax of *monad-based dynamic logic (MDL)*. Given a set Sort of *basic types*, the set of *types* over Sort is generated by

$$A ::= 1 \mid \Omega \mid PA \mid TA \mid A \times A \mid \mathit{Sort}.$$

$$\textbf{(var)}\ \frac{x : A \in \Gamma}{\Gamma \rhd x : A} \qquad \textbf{(app)}\ \frac{f : A \to B \in \Sigma\ \ \Gamma \rhd t : A}{\Gamma \rhd f(t) : B} \qquad \textbf{(1)}\ \frac{}{\Gamma \rhd * : 1}$$

$$\textbf{(fst)}\ \frac{\Gamma \rhd t : A \times B}{\Gamma \rhd \mathrm{fst}(t) : A} \qquad \textbf{(snd)}\ \frac{\Gamma \rhd t : A \times B}{\Gamma \rhd \mathrm{snd}(t) : A} \qquad \textbf{(pair)}\ \frac{\Gamma \rhd t : A\ \ \Gamma \rhd u : B}{\Gamma \rhd \langle t, u\rangle : A \times B}$$

$$(\top)\ \frac{}{\Gamma \rhd \top : \Omega} \qquad (\bot)\ \frac{}{\Gamma \rhd \bot : \Omega} \qquad (\neg)\ \frac{\Gamma \rhd a : \Omega}{\Gamma \rhd \neg a : \Omega} \quad \text{similarly for } \wedge, \vee, \Rightarrow, \iff$$

$$\textbf{(do)}\ \frac{\Gamma \rhd p : TA\ \ \Gamma, x : A \rhd q : TB}{\Gamma \rhd \mathrm{do}\ x \leftarrow p; q : TB} \qquad \textbf{(doP)}\ \frac{\Gamma \rhd p : PA\ \ \Gamma, x : A \rhd q : PB}{\Gamma \rhd \mathrm{do}\ x \leftarrow p; q : PB}$$

$$\textbf{(ret)}\ \frac{\Gamma \rhd t : A}{\Gamma \rhd \mathrm{ret}\ t : PA} \qquad (\Box)\ \frac{\Gamma \rhd p : T\Omega}{\Gamma \rhd \Box p : P\Omega} \qquad \textbf{(P)}\ \frac{\Gamma \rhd p : PA}{\Gamma \rhd p : TA}$$

Figure 1. Term language for monad-based dynamic logic

The type TA contains the monadic programs over A; PA is a subtype of TA containing pure programs. The type of Booleans is denoted Ω; consequently, we take $P\Omega$ as the type of formulae of monad-based dynamic logic.

A *signature* $\Sigma = (Sort, F)$ consists of a set $Sort$ of basic types and a set F of operation symbols $f : A \to B$, where A and B are types over $Sort$. Examples of such operation symbols are $new : T\ Loc$ and $int2loc : Int \to Loc$ as explained above. We make the general assumption that the operations in the signature have T*-free* argument types, more precisely, types containing neither T nor P. The term language over a signature Σ and a context Γ of typed variables is given in Fig. 1. A judgement $\Gamma \rhd t : A$ means that in context Γ, term t has type A. The symbol $\top$ stands for $true$, $\bot$ for $false$. We let metavariables t etc. range over terms, a and b over *(plain) formulae*, i.e. terms of type Ω, φ, ψ, χ over *monadic formulae*, i.e. terms of type $P\Omega$, and p, q etc. over *programs*, i.e. terms whose type is of the form TA. We use $\equiv$ to denote syntactic equality of terms, formulae and contexts. The term $\mathrm{ret}\ t$ is an effectless computation that just returns the value t. In the term $\mathrm{do}\ x \leftarrow p; q$, the variable x is locally bound in q (but not in p), the interpretation being 'perform computation p, bind the result to x, and then perform computation $q(x)$'. Repeated bindings such as $\mathrm{do}\ x_1 \leftarrow p_1, \ldots, x_n \leftarrow p_n; q$ are somewhat inaccurately denoted in the form $\mathrm{do}\ \bar{x} \leftarrow \bar{p}; q$. Term fragments of the form $\bar{x} \leftarrow \bar{p}$ are called *program sequences*. The operations fst and snd are the projection functions for binary products. We treat n-ary products as iterated binary products; then projections π_i^n can easily be defined in terms of fst and snd.

The term forming operation $\Box : T\Omega \to P\Omega$ is a closure operator. The monadic formula $\Box p$ intuitively expresses that all terminating runs of p return $\top$. Note that $\Box$ does not behave like a modal operator; in particular, for $\varphi : P\Omega$, we will have $\varphi \iff \Box\varphi$. However, $\Box$ serves to define modalities: For $\varphi : P\Omega$, we let $[\bar{x} \leftarrow \bar{p}]\,\varphi$ abbreviate the monadic formula $\Box\,\mathrm{do}\ \bar{x} \leftarrow \bar{p}; \varphi$, and omit $\bar{x}$ if it does not occur in φ. The monadic formula $\langle \bar{x} \leftarrow \bar{p}\rangle\varphi$ abbreviates $\neg[\bar{x} \leftarrow \bar{p}]\,\neg\varphi$. *Both in* do*-terms and within modal operators, we implicitly identify terms up to* α*-equivalence.*

Besides the specification and verification of monadic programs, MDL serves also the (potentially loose) *specification* of monads, i.e. notions of side effect.

The running example of [SM03, SM04a] involves references and non-determinism; numerous further examples can be found in [Wal05], including the Java monad and a parsing monad, as well as a queue monad over a fixed set U of entries. Here, we present the specification of a monad for dynamic references. It is axiomatised using Hoare triples for total correctness

$$[\varphi]\ p\ [\psi]$$

by interpreting them as partial correctness plus termination:

$$\varphi \Rightarrow (\langle p\rangle\top \wedge [p]\,\psi).$$

The implication $\varphi \Rightarrow \langle p\rangle\top$ is called the termination part, and the implication $\varphi \Rightarrow [p]\,\psi$ is called the partial correctness part of the Hoare triple.

Example 3.1. We assume, for any basic type A, a (basic) type $Ref\,A$ of references to objects of type A and an equality operation $\_ = \_ : A \times A \to \Omega$, the latter axiomatised by reflexivity $x = x$ and the substitution schema

$$\textbf{(var)}\ \frac{}{[\![x_1:A_1,\dots,x_n:A_n \rhd x_i:A_i]\!]=\pi_i^n} \qquad \textbf{(app)}\ \frac{f:A\to B\in\Sigma\ \ [\![\Gamma\rhd t:A]\!]=h}{[\![\Gamma\rhd f(t):B]\!]=[\![f]\!]\circ h}$$

$$\textbf{(1)}\ \frac{}{[\![\Gamma\rhd *:1]\!]=!} \qquad \textbf{(pair)}\ \frac{[\![\Gamma\rhd t:A]\!]=h_1\ \ [\![\Gamma\rhd u:B]\!]=h_2}{[\![\Gamma\rhd\langle t,u\rangle:A\times B]\!]=\langle h_1,h_2\rangle} \qquad \textbf{(fst)}\ \frac{[\![\Gamma\rhd t:A\times B]\!]=h}{[\![\Gamma\rhd \mathrm{fst}(t):A]\!]=\pi_1\circ h}$$

$$\textbf{(snd)}\ \frac{[\![\Gamma\rhd t:A\times B]\!]=h}{[\![\Gamma\rhd \mathrm{snd}(t):A]\!]=\pi_2\circ h} \qquad \textbf{(do)}\ \frac{[\![\Gamma\rhd p:TA]\!]=h_1\ \ [\![\Gamma,x:A\rhd q:TB]\!]=h_2}{[\![\Gamma\rhd \mathrm{do}\ x\leftarrow p;q:TB]\!]=h_2^*\circ t_{[\![\Gamma]\!],[\![A]\!]}\circ\langle id,h_1\rangle}$$

$$\textbf{(ret)}\ \frac{[\![\Gamma\rhd t:A]\!]=h}{[\![\Gamma\rhd \mathrm{ret}\ t:PA]\!]=\eta_{[\![A]\!]}\circ h} \qquad (\Box)\ \frac{[\![\Gamma\rhd p:T\Omega]\!]=h}{[\![\Gamma\rhd \Box p:P\Omega]\!]=\Box\circ h} \qquad \textbf{(P)}\ \frac{[\![\Gamma\rhd p:PA]\!]=h}{[\![\Gamma\rhd p:TA]\!]=\iota_A\circ h}$$

$$(\top)\ \frac{}{[\![\Gamma\rhd\top:\Omega]\!]=\top} \qquad (\bot)\ \frac{}{[\![\Gamma\rhd\bot:\Omega]\!]=\bot} \qquad (\neg)\ \frac{[\![\Gamma\rhd a:\Omega]\!]=h}{[\![\Gamma\rhd\neg a:\Omega]\!]=\neg(h)} \qquad \text{similarly for } \wedge,\vee,\Rightarrow,\iff$$

Figure 2. Semantics of monad-based dynamic logic

$\mathrm{ret}\,(x=y)\Rightarrow(\varphi\Leftrightarrow\varphi[y/x])$ for every formula φ (schematic axiomatisations are unproblematic in our framework, as we prove *strong* completeness below). The specification of dynamic references is then as follows.

$$\begin{aligned}
&{*__}:\mathit{Ref}A\to PA\\
&__:=__:(\mathit{Ref}A)\times A\to T1\\
&\mathit{new}:A\to T(\mathit{Ref}A)\\
&[\,]r:=x\ [x=*r] && \text{(read-write)}\\
&[x=*r]\ s:=y\ [x=*r\vee r=s] && \text{(read-write-other)}\\
&[\,]\ r\leftarrow \mathit{new}\ x\ [x=*r] && \text{(read-new)}\\
&[x=*r]\ s\leftarrow \mathit{new}\ y\ [x=*r\vee\neg r=s] && \text{(read-new-other)}\\
&[\,]\ r\leftarrow \mathit{new}\ x;p;s\leftarrow \mathit{new}\ y\ [\neg r=s] && \text{(new-distinct)}
\end{aligned}$$

The operation ${*\_\_}$ reads the value from the location specified by the reference, while $\_\_:=\_\_$ assigns a new value to the location, and new creates a new location.

Note that the profile of ${*\_\_}$ ensures that reading is pure. The first axiom expresses that after assignment, the assigned value can be read; the second one that assignment for a particular location does not affect the other locations. The next two axioms state similar properties for the new operation, and the last axiom ensures that two newly created locations are distinct.

3.2. Semantics

MDL is interpreted over a strong monad $\mathbb{T}$ on a cartesian category $\mathbf{C}$[3] with additional structure as follows. There is a distinguished internal Boolean algebra object Ω; note that this is equivalent to hom-sets into Ω coherently carrying a Boolean algebra structure, i.e. the hom-functor $\hom(\_,\Omega)$ factoring through Boolean algebras. In order to enforce a Boolean logic, we additionally require that $(id_A\times\top:A\times 1\to A\times\Omega, id_A\times\bot:A\times 1\to A\times\Omega)$ is an episink. Moreover, $\mathbb{T}$ needs to be equipped with a strong submonad $\mathbb{P}$, with functor part P (the inclusion is denoted $\iota:P\to T$); we require that PA consists of pure computations as defined further below.[4] Finally, we need a left inverse $\Box:T\Omega\to P\Omega$ of the inclusion $\iota:P\Omega\hookrightarrow T\Omega$. It will be axiomatised uniquely later on (cf. Prop. 3.11).

[3] $\times$ is product with projections π_1,π_2, 1 the terminal object with $!_A:A\to 1$ the unique arrow.

[4] One obvious possibility is to let PA consist of *all* pure computations; this always gives a strong submonad. We will use this possibility in the examples below.

Remark 3.2. In a distributive category, one obtains the Boolean algebra structure by defining Ω as $1+1$. E.g., every topos is distributive, and in a classical topos, the subobject classifier is just $1+1$. In a category with equalisers, there is also a canonical choice for the subfunctor P, namely to take PA as the subobject of TA determined by purity as defined below. Then the $\Box$ arrow is uniquely determined if it exists (see Prop. 3.11 below).

We then interpret the basic sorts as objects in $\mathbf{C}$. This is easily extended to all types, giving an interpretation $[\![A]\!]$ for each type A. Basic operations $f : A \to B$ are interpreted as morphisms $[\![A]\!] \to [\![B]\!]$, and terms $x_1 : A_1, \dots, x_n : A_n \rhd t : A$ as morphisms $[\![t]\!] : [\![A_1 \times \cdots \times A_n]\!] \to [\![A]\!]$, using the cartesian structure for pairing and projections, the monad for do and ret, and the Boolean algebra structure on Ω for the Boolean connectives as shown in Fig. 2. (Note that there is no particular rule for sequential composition within P: the fact that $\mathbb{P}$ is a submonad guarantees closedness under sequential composition). Observe, that logical connectives are also applicable to $P\Omega$. For instance $p \wedge q$ is an alias for $(\mathrm{do}\ x \leftarrow p; y \leftarrow q; \mathrm{ret}\ x \wedge y)$. The fact that $P\Omega$ inherits Boolean algebra structure from Ω strongly relies on the definition of P [SM04a]. Equations between terms are interpreted as equations between the corresponding morphisms; this will be used in a series of definitions below (we often leave the context implicit).

The following lemma has a straightforward proof:

Lemma 3.3 (Substitution). Let Γ and $\Delta \equiv \bar{y} : \bar{B}$ be contexts, let $\Gamma \rhd \bar{t} : \bar{B}$, and let $\Delta \rhd s : C$. Then

$$[\![\Gamma \rhd s[\bar{t}/\bar{y}] : C]\!] = [\![\Delta \rhd s : C]\!] \circ [\![\Gamma \rhd \bar{t} : \bar{B}]\!].$$

3.3. Purity

MDL formulae will be interpreted as computations of type $P\Omega$ (i.e. morphisms from the object interpreting the context into the object $P\Omega$). They are expected to have no side-effect, although they may e.g. read the state (if a notion of state is present in the monad). We now will abstractly capture the three conditions for purity listed in Sect. 1. The first condition, namely that computations can be left out from a sequence of computation steps without changing its behaviour, is captured as follows:

Definition 3.4. In a monad given as above, a program $p : TA$ is called *discardable* [Thi97] if

$$(\mathrm{do}\ y \leftarrow p; \mathrm{ret}\ *) = \mathrm{ret}\ *.$$

The following properties from [SM04a] substantiate the claim that discardability actually captures the stated intention: If p is discardable, then

$$(\mathrm{do}\ p; q) = q$$

for each program q. Moreover, if $p, q : T1$, then

$$(\mathrm{do}\ q; p) = q.$$

For the monads described in Example 2.1, discardability of p means:

1. State monad: p possibly reads the state, but does not change it;
2. Non-determinism: p returns at least one value;
3. Exceptions: p terminates normally;
4. In-/Output: p does not read/write;
5. Nondeterministic state monad: p possibly reads the state, but does not change it;
6. Java monad: p only reads the state and terminates normally.

The second condition concerns determinism: pure programs always return the same value. This is captured as follows: A program $p : TA$ is called *copyable* [Füh02, Thi97] if

$$(\mathrm{do}\ x \leftarrow p; y \leftarrow p; \mathrm{ret}\ \langle x, y\rangle) = \mathrm{do}\ x \leftarrow p; \mathrm{ret}\ \langle x, x\rangle.$$

Although copyability alone is not of much interest, we detail what copyability of p means in our example monads:

1. State monad: p is idempotent in the sense that applying its state transformation twice is the same as applying it once, and the returned values are the same;
2. Non-determinism: p returns at most one value;
3. Exceptions: p is always copyable;

4. In-/Output: p does not read/write;
5. Nondeterministic state monad: p is idempotent and deterministic;
6. Java monad: p is idempotent.

The third condition is interchangeability: Pure programs can be interchanged with each other, that is, the order does not matter. Hence, p is *pure* if it is both discardable and copyable, and *commutes* with all such programs q, that is,

$$(\mathrm{do}\ x \leftarrow p; y \leftarrow q; \mathrm{ret}\ \langle x, y\rangle) = \mathrm{do}\ y \leftarrow q; x \leftarrow p; \mathrm{ret}\ \langle x, y\rangle.$$

The last condition follows from the first two for simple monads, on which we focus here; cf. Prop. 4.8.

For the monads described in Example 2.1, purity of p means the expected things:

1. State monad: p possibly reads the state, but does not change it;
2. Non-determinism: p is deterministic, i.e. returns precisely one value;
3. Exceptions: p terminates normally;
4. In-/Output: p does not read/write;
5. Nondeterministic state monad: p is deterministic and only reads the state, but does not change it;
6. Java monad: p only reads the state and terminates normally.

The equation for discardability can be interpreted as a pair of arrows $dis_0, dis_1 : T[\![A]\!] \to T[\![1]\!]$; we require that $\iota_{[\![A]\!]} : P[\![A]\!] \to T[\![A]\!]$ equalises this pair. (We do *not* require that $\iota_{[\![A]\!]}$ is the equaliser, for the reconstruction of this equaliser in the term model would require a coercion of provably pure terms of type TA to type PA, which means that term formation rules would need to interact with proof rules.) A similar requirement holds for copyability; i.e. we require that PA contains only pure programs.

3.4. The Logic

We will want to regard programs that return truth values as monadic formulae with side effects in a modal logic setting. A basic notion we need for such formulae is that of global validity, which we denote explicitly by a 'global box' $\boxed{\mathrm{G}}$:

Definition 3.5. Given a term $p : T\Omega$, $\boxed{\mathrm{G}}p$ abbreviates the equation

$$p = \mathrm{do}\ p; \mathrm{ret}\ \top.$$

If p is discardable, then $\boxed{\mathrm{G}}p$ simplifies to $p = \mathrm{ret}\ \top$; otherwise, the equation above ensures that the right hand side has the same side-effect as p. We say that an MDL formula φ is *valid* in a model $\mathbb{T}$, and write $\mathbb{T} \models \varphi$, if $\boxed{\mathrm{G}}\varphi$; this is usually expressed by just writing φ. As usual, by $\mathbb{T} \models \Phi$ for a set Φ of MDL formulae we mean that $\mathbb{T} \models \varphi$ for all $\varphi \in \Phi$, and by $\Phi \models \psi$ that $\mathbb{T} \models \Phi$ implies $\mathbb{T} \models \psi$ for all $\mathbb{T}$.

In our example monads, satisfaction of $\boxed{\mathrm{G}}\varphi$ amounts to the following:

- State monad: successful execution of φ from any initial state yields $\top$;
- Non-determinism: φ yields at most the value $\top$ (or none at all);
- Exceptions: φ yields $\top$ whenever it terminates normally;
- Input/output: the value eventually produced by φ after some combination of inputs is always $\top$;
- Non-deterministic state monad: execution of φ from any initial state yields at most the value $\top$;
- Java monad: successful execution of φ from any initial state yields $\top$ whenever it terminates normally.

A related notion is that of *global dynamic judgements* of the form $[\bar{x} \leftarrow \bar{p}]\, a$, which intuitively state that a holds after $\bar{x} \leftarrow \bar{p}$, where $a : \Omega$ is a truth-valued term in variables $\bar{x}$. The idea is to work with formulae that have all side effects shoved to the outside, so that the usual logical rules apply to the remaining part.

Definition 3.6. Given a program sequence $\bar{x} \leftarrow \bar{p}$ and a formula a of type Ω, the notation $[\bar{x} \leftarrow \bar{p}]\, a$ abbreviates the equation

$$(\mathrm{do}\ \bar{x} \leftarrow \bar{p}; \mathrm{ret}\ \langle \bar{x}, a\rangle) = \mathrm{do}\ \bar{x} \leftarrow \bar{p}; \mathrm{ret}\ \langle \bar{x}, \top\rangle.$$

Definition 3.7. A monad is called *simple* if $\boxed{\mathrm{G}}\,\mathrm{do}\ \bar{x} \leftarrow \bar{p}; \mathrm{ret}\ a$ implies $[\bar{x} \leftarrow \bar{p}]\, a$. (The converse implication holds universally.) Roughly, an algebraic monad [Mac97] is simple if, in each of its equations, the two sides contain the same variables. All monads of Example 2.1 are simple. The continuation monad and the abelian group monad are not simple. *In the sequel, all monads are assumed to be simple*.

We adapt the axiomatic definition of the dynamic modal operators from [SM04a] to the simplified setting of simple monads:

Definition 3.8. $\mathbb{T}$ is said to *admit dynamic logic*, if for each $q : T\Omega$ and each $\bar{x} \leftarrow \bar{p}$ containing $x_i : \Omega$,

$$[\bar{x} \leftarrow \bar{p}; a \leftarrow \Box q]\,(x_i \Rightarrow a) \text{ iff } [\bar{x} \leftarrow \bar{p}; a \leftarrow q]\,(x_i \Rightarrow a).$$

This definition essentially axiomatises $\Box$ via its interaction with global dynamic judgements. In order to gain some intuitive understanding, let us assume that we are working in a non-deterministic state monad. Note that purity of $\Box q$ is enforced by its type; this means that in a given state, $\Box q$ returns either $\top$ or $\bot$ (but not both — it is deterministic). Hence, the above definition expresses that in any given state ($\bar{p}$ can be used to move to that state), $\Box q$ holds iff all executions of q (starting from the given state) return true.

For the next proposition, it is important to recall the difference between the *local box* $[\bar{x} \leftarrow \bar{p}]\,\varphi$ (a monadic value of type $P\Omega$ that may be used in formation of monadic terms) and the *global box* $[\bar{x} \leftarrow \bar{p}]\, a$ (a global equation between certain monadic terms).

Proposition 3.9. If a simple monad $\mathbb{T}$ admits dynamic logic, then

$$[\bar{x} \leftarrow \bar{p}; a \leftarrow [\bar{y} \leftarrow \bar{q}]\,\varphi]\,(x_i \Rightarrow a) \text{ iff } [\bar{x} \leftarrow \bar{p}; \bar{y} \leftarrow \bar{q}; a \leftarrow \varphi]\,(x_i \Rightarrow a),$$

i.e. $\mathbb{T}$ admits dynamic logic in the sense of [SM04a].

Proof. Follows from Def. 3.8 using rules (ctr) and (ctr-) of the calculus of global dynamic judgements in [SM04a]. □

Thus, the operator $[\bar{y} \leftarrow \bar{q}]$ in a sense serves to predict all *positive* statements to be made about the result $\bar{y}$ after executing the computations $\bar{q}$ *in a given state*; the formal content of the latter phrase is reflected in the quantification over program sequences $\bar{x} \leftarrow \bar{p}$ to be executed before $\bar{q}$.

Example 3.10. All monads described in Example 2.1 admit dynamic logic, with the meaning of the dynamic modal operators made explicit below. A typical example that fails to admit dynamic logic is the continuation monad $\lambda X.\,(X \to R) \to R$ [SM04a].

1. *Stateful computations with non-termination* ($TA = (S \to ?\ (A \times S))$): Elements of $P\Omega$ are state-dependent truth values, i.e. functions $S \to \Omega$. A state s satisfies $[x \leftarrow p]\,\varphi$ if the value x, if any, returned by p after terminating execution starting in state s satisfies φ.

2. *Non-determinism* ($TA = \mathcal{P}(A)$): $P\Omega$ is isomorphic to the set Ω of truth values; $[x \leftarrow p]\,\varphi$ holds if all $x \in p$ satisfy φ.

3. *Exceptions* ($TA = A + E$): $P\Omega$ is essentially Ω; $[x \leftarrow p]\,\varphi$ holds if p is either an exception or a value x satisfying φ.

4. *Interactive input* ($TA = \mu\gamma.\, A + (U \to \gamma)$): $P\Omega$ is essentially Ω; $[x \leftarrow p]\,\varphi$ holds if the value returned by p after reading a number of inputs satisfies φ. *Interactive output* ($TA = \mu\gamma.\, A + (U \times \gamma)$): again, $P\Omega$ is essentially Ω, and $[x \leftarrow p]\,\varphi$ holds if the value returned by p satisfies φ (the output is ignored).

5. *Non-deterministic stateful computations* ($TA = \mathcal{P}(S \to (A \times S))$): Elements of $P\Omega$ are state-dependent truth values $S \to \Omega$. A state s satisfies $[x \leftarrow p]\,\varphi$ if all values x possibly returned by p after terminating execution starting in state s satisfy φ.

6. *Java monad* ($TA = S \to (A \times S + E \times S + 1)$): again, $P\Omega$ is $S \to \Omega$, and $[x \leftarrow p]\,\varphi$ means that all values x returned by p after a terminating execution satisfy φ.

While at first sight, items 2–4 look uninteresting from the perspective of dynamic logic, it should be kept in mind that all these monads may be combined with 'dynamic' monads such as the state monad, as exemplified for the cases of the non-determinism monad in item 5 and the Java monad in item 6.

As stated in Sect. 3.2, $P\Omega$ is a Boolean algebra; hence, it is equipped with a partial order. For simple monads, this partial order is given by

$$\varphi \leq \psi \text{ iff } [a \leftarrow \varphi; b \leftarrow \psi]\,(a \Rightarrow b)$$

Rules:

$$\textbf{(nec)}\ \frac{\varphi}{[\bar{x} \leftarrow \bar{p}]\,\varphi}\quad \bar{x} \text{ not free in assumptions} \qquad \textbf{(mp)}\ \frac{\varphi \Rightarrow \psi; \quad \varphi}{\psi}$$

Axioms:

(K)	$[\bar{x} \leftarrow \bar{p}]\,(\varphi \Rightarrow \psi) \Rightarrow [\bar{x} \leftarrow \bar{p}]\,\varphi \Rightarrow [\bar{x} \leftarrow \bar{p}]\,\psi$	
(seq□)	$[\bar{x} \leftarrow \bar{p}; y \leftarrow q]\,\varphi \iff [\bar{x} \leftarrow \bar{p}]\,[y \leftarrow q]\,\varphi$	
(ctr□)	$[x \leftarrow p; y \leftarrow q]\,\varphi \iff [y \leftarrow (\mathrm{do}\ x \leftarrow p; q)]\,\varphi$	$(x \notin FV(\varphi))$
(ret□)	$[x \leftarrow \mathrm{ret}\ t]\,\varphi \iff \varphi[t/x]$	
(dis)	$[x \leftarrow p]\,\psi \iff \psi$	$p : PA$, x not free in ψ
(copy)	$[x \leftarrow p; y \leftarrow p]\,\psi \iff [x \leftarrow p]\,\psi[x/y]$	$p : PA$
(□)	$[a \leftarrow \Box p]\,\mathrm{ret}\,(t \Rightarrow a) \iff [a \leftarrow p]\,\mathrm{ret}\,(t \Rightarrow a)$	$p : T\Omega$
(unit)	$[x \leftarrow \varphi]\ \mathrm{ret}\ x \iff \varphi$	
(CC)	$\varphi[t/x] \iff \varphi[u/x]$	for $(t = u) \in CC$
(cong⇔)	$\mathrm{ret}\,(t \Leftrightarrow u) \Rightarrow (\varphi[t/x] \iff \varphi[u/x])$	
(taut)	$\mathrm{ret}\ a$	$a : \Omega$ a tautology

$$CC = \{\mathrm{fst}\langle x, y\rangle = x;\ \mathrm{snd}\langle x, y\rangle = y;\ \langle \mathrm{fst}(x), \mathrm{snd}(x)\rangle = x;\ x : 1 = *\}$$

Figure 3. The generic proof calculus for monad-based dynamic logic

for $\varphi, \psi : P\Omega$. The following two claims are proved similarly as in [SM04a].

Proposition 3.11. The monadic formula $\Box p$ is the greatest monadic formula $\varphi : P\Omega$ such that

$$[a \leftarrow \varphi; b \leftarrow p]\,(a \Rightarrow b).$$

Proposition 3.12. $[\bar{x} \leftarrow \bar{p}]\,a$ iff $\boxed{\mathrm{G}}[\bar{x} \leftarrow \bar{p}]\,\mathrm{ret}\ a$

Proposition 3.11 implies that $\Box p$ is uniquely determined if it exists. Proposition 3.12 relates global dynamic judgements and local modal formulae. Note the difference between *global* dynamic judgements $[\bar{x} \leftarrow \bar{p}]\,a$ and the similar-looking MDL formulae $[\bar{x} \leftarrow \bar{p}]\,\varphi$ involving a *local* modality. From a technical point of view, a global dynamic judgement is an equation between terms (and the component formula a has type Ω), while a local modal formula is a term (and the component formula φ has type $P\Omega$). But the difference is more fundamental: local modalities can be nested, and e.g. in the state monad one can think of them as being evaluated relative to a local state. This is not possible with global dynamic judgements: they always quantify over *all* states.

4. A Calculus for Dynamic Logic

Figure 3 shows a proof calculus for MDL. The calculus differs from the one in [SM04a] in that the rules for the diamond are omitted, because in classical logic, $\Diamond$ can be defined as $\neg\Box\neg$. Moreover, Axioms $(\Box)$ (resembling the implicit definition of $\Box$ in the formula for admission of dynamic logic), (dis) and (copy) (expressing purity of terms of type PA), (unit), and the congruence axioms (cong⇔), (cong=) and (CC) have been added. The axiom schema (CC) throws in the standard equations for tupling, projections, and $*$, i.e. the internal equations of cartesian categories. The usual logical connectives are lifted to $P\Omega$ by defining e.g.

$$\varphi \Rightarrow \psi := [a \leftarrow \varphi; b \leftarrow \psi]\,\mathrm{ret}\,(a \Rightarrow b) : P\Omega,$$

which by (□MDL) below is equivalent to

$$\mathrm{do}\ a \leftarrow \varphi; b \leftarrow \psi; \mathrm{ret}\,(a \Rightarrow b).$$

We write $\Phi \vdash \psi$ if a monadic formula is derivable in the calculus from a set Φ of axioms.

Proposition 4.1 (Soundness of MDL). If $\Phi \vdash \psi$, then $\Phi \models \psi$.

Proof. Rules (nec) and (mp) and axioms (K), (seq□), (ctr□), and (ret□) are proved to be sound in [SM04a], which carries over to our definition of admission of dynamic logic by Prop. 3.9 and the observation that the proof in [SM04a] does not use the partial cartesian closed structure of the category. We now cover the remaining axioms:

Axiom ($\Box$): We prove soundness for the special case that t is a variable x; the general case then follows with the substitution lemma 3.3. We prove that $\psi \equiv [a \leftarrow \Box\varphi]\,(x \Rightarrow a)$ has the defining property of the right hand side, i.e.

$$\llbracket \bar{y} \leftarrow \bar{q}; b \leftarrow \psi \rrbracket\,(y_i \Rightarrow b) \text{ iff } \llbracket \bar{y} \leftarrow \bar{q}; a \leftarrow \varphi; b \leftarrow \mathrm{ret}\,(x \Rightarrow a) \rrbracket\,(y_i \Rightarrow b).$$

We calculate

$$\begin{aligned}
&\llbracket \bar{y} \leftarrow \bar{q}; b \leftarrow [a \leftarrow \Box\varphi]\,(x \Rightarrow a) \rrbracket\,(y_i \Rightarrow b)\\
&\qquad \iff \llbracket \bar{y} \leftarrow \bar{q}; a \leftarrow \Box\varphi; b \leftarrow \mathrm{ret}\,(x \Rightarrow a) \rrbracket\,(y_i \Rightarrow b)\\
&\qquad \iff \llbracket \bar{y} \leftarrow \bar{q}; a \leftarrow \Box\varphi \rrbracket\,(y_i \Rightarrow x \Rightarrow a)\\
&\qquad \iff \llbracket \bar{y} \leftarrow \bar{q}; a \leftarrow \varphi \rrbracket\,(y_i \Rightarrow x \Rightarrow a)\\
&\qquad \iff \llbracket \bar{y} \leftarrow \bar{q}; a \leftarrow \varphi; b \leftarrow \mathrm{ret}\,(x \Rightarrow a) \rrbracket\,(y_i \Rightarrow b),
\end{aligned}$$

exploiting that we can use the defining property of $\Box$ with monadic formulae instead of just variables before the implication sign, and omitting some propositional reasoning.

Axiom (unit): By the η-rule for global dynamic judgements [SM04a],

$$\llbracket \bar{y} \leftarrow \bar{q}; a \leftarrow \varphi \rrbracket\,(y_i \Rightarrow a) \text{ iff } \llbracket \bar{y} \leftarrow \bar{q}; x \leftarrow \varphi; a \leftarrow \mathrm{ret}\ \ x \rrbracket\,(y_i \Rightarrow a),$$

i.e. φ has the defining property of the left hand side $[x \leftarrow \varphi]\ \mathrm{ret}\ x$.

Axiom (CC): immediate from the soundness of CC for cartesian categories.

Axiom (taut): if a is a tautology, then $[\![\Gamma \rhd a : \Omega]\!] = \top$, since $\hom([\![\Gamma]\!], \Omega)$ is a Boolean algebra. Thus, $[\![\Gamma \rhd \mathrm{ret}\ a : P\Omega]\!] = \eta_\Omega \circ \top$, i.e. $\mathrm{ret}\ a$ is valid.

Axiom (dis): by the (dis) rule of the calculus for global dynamic judgements in simple monads [SM04a],

$$\llbracket \bar{y} \leftarrow \bar{p}; a \leftarrow \psi \rrbracket\,(y_i \Rightarrow a) \text{ iff } \llbracket \bar{y} \leftarrow \bar{p}; x \leftarrow \varphi; a \leftarrow \psi \rrbracket\,(y_i \Rightarrow a),$$

i.e. the right hand side has the defining property of the left hand side.

Axiom (copy): by the structural rules for pure programs [SM04a],

$$\llbracket \bar{y} \leftarrow \bar{p}; x \leftarrow \varphi; y \leftarrow \varphi; a \leftarrow \psi \rrbracket\,(y_i \Rightarrow a) \text{ iff } \llbracket \bar{y} \leftarrow \bar{p}; x \leftarrow \varphi; a \leftarrow \psi[x/y] \rrbracket\,(y_i \Rightarrow a).$$

The left hand side is, by the defining property of $[x \leftarrow \varphi; y \leftarrow \varphi]\,\psi$, equivalent to

$$\llbracket \bar{y} \leftarrow \bar{p}; a \leftarrow [x \leftarrow \varphi; y \leftarrow \varphi]\,\psi \rrbracket\,(y_i \Rightarrow a),$$

i.e. the left hand side of axiom (copy) has the defining property of the right hand side.

Axiom (cong$\Leftrightarrow$): We prove the soundness of the special case $\mathrm{ret}\,(x \iff y) \Rightarrow (\varphi \iff \varphi[y/x])$; soundness of the full (cong$\Leftrightarrow$) axiom schema then follows with the substitution lemma 3.3. By applying the episink property twice, we obtain that $(\ 1 \times 1 \xrightarrow{id\times h} 1 \times \Omega \cong \Omega \times 1 \xrightarrow{id\times k} \Omega \times \Omega\ )_{h,k\in\{\bot,\top\}}$ and hence $\langle h, k\rangle_{h,k\in\{\bot,\top\}}$ is an episink as well. By the substitution lemma, it suffices to show that $[a \leftarrow \varphi[t_1/x]; b \leftarrow \varphi[t_2/x]]\,\mathrm{ret}\,((x \iff y) \Rightarrow a \Rightarrow b)$ holds for $t_1, t_2 \in \{\bot, \top\}$. If $t_1 = t_2$, by soundness of (copy), it suffices to show $[a \leftarrow \varphi[t_1/x]]\,(\mathrm{ret}\,((x \iff y) \Rightarrow a \Rightarrow a))$, which by soundness of (taut) reduces to $[a \leftarrow \varphi[t_1/x]]\,\mathrm{ret}\ \top$ and by soundness of (dis) to $\mathrm{ret}\ \top$. If $t_1 \neq t_2$, by soundness of (taut) we need to show $[a \leftarrow \varphi[t_1/x]; b \leftarrow \varphi[t_2/x]]\,(\mathrm{ret}\ \bot \Rightarrow a \Rightarrow b)$, which by soundness of (taut) and (dis) again reduces to $\mathrm{ret}\ \top$. $\square$

Lemma 4.2. The monadic formulae

(cong$\Box$)	$(\varphi \Leftrightarrow \psi) \Rightarrow ([x \leftarrow \varphi]\,\chi \Leftrightarrow [x \leftarrow \psi]\,\chi)$
($\Box$def)	$\Box p \iff [x \leftarrow p]\,\mathrm{ret}\ x$
($\Box$MDL)	$\varphi \iff \Box\varphi$
(do$\Box$)	$[\bar{x} \leftarrow \bar{p}]\,\Box q \iff \Box\,\mathrm{do}\ \bar{x} \leftarrow \bar{p}; q$
(K3)	$\mathrm{ret}\ a \implies [p]\,\mathrm{ret}\ a$

are derivable.

Proof. (cong$\Box$): The premise of the implication decodes to

$$[x \leftarrow \varphi; y \leftarrow \psi]\ \mathrm{ret}\,(x \Leftrightarrow y),$$

and the conclusion can be rewritten to

$$[x \leftarrow \varphi; y \leftarrow \psi]\,\chi \Leftrightarrow [x \leftarrow \varphi; y \leftarrow \psi]\,\chi[y/x]$$

using α-equivalence, (seq$\Box$), and (dis). Formula (cong$\Box$) then follows by propositional reasoning with (K) and (cong$\Leftrightarrow$). Formula ($\Box$def) follows from axioms (unit) and ($\Box$). From ($\Box$def), we obtain ($\Box$MDL), again using (unit).

(do$\Box$): $[\bar{x} \leftarrow \bar{p}]\,\Box q$ is by (seq$\Box$) and (unit) equivalent to $[\bar{x} \leftarrow \bar{p}; y \leftarrow \Box q]\,\mathrm{ret}\ y$, which by ($\Box$) is equivalent to $[\bar{x} \leftarrow \bar{p}; y \leftarrow q]\,\mathrm{ret}\ y$. By (ctr$\Box$), this is equivalent to $[y \leftarrow (\mathrm{do}\ \bar{x} \leftarrow \bar{p}; q)]\,\mathrm{ret}\ y$. Again by ($\Box$), this is equivalent to $[y \leftarrow (\Box\,\mathrm{do}\ \bar{x} \leftarrow \bar{p}; q)]\,\mathrm{ret}\ y$, which by (unit) is equivalent to $\Box\,\mathrm{do}\ \bar{x} \leftarrow \bar{p}; q$.

(K3): By (taut) and (nec), $[p; y \leftarrow \mathrm{ret}\ a]\ \mathrm{ret}\,(y \Rightarrow y)$. By (ret $\Box$), we obtain $[x \leftarrow \mathrm{ret}\ a; p; y \leftarrow \mathrm{ret}\ a]\ \mathrm{ret}\,(x \Rightarrow y)$. By (ctr$\Box$) this is equivalent to $[x \leftarrow \mathrm{ret}\ a; y \leftarrow (\mathrm{do}\ p; \mathrm{ret}\ a)]\ \mathrm{ret}\,(x \Rightarrow y)$. Then by ($\Box$) we obtain $[x \leftarrow \mathrm{ret}\ a; y \leftarrow [p]\,\mathrm{ret}\ a]\ \mathrm{ret}\,(x \Rightarrow y)$. By (K) this can be resolved into the implication:

$$[x \leftarrow \mathrm{ret}\ a; y \leftarrow [p]\,\mathrm{ret}\ a]\ \mathrm{ret}\ x \Rightarrow [x \leftarrow \mathrm{ret}\ a; y \leftarrow [p]\,\mathrm{ret}\ a]\ \mathrm{ret}\ y.$$

We will be done once we show that the left-hand side is equivalent to $\mathrm{ret}\ a$ and the right-hand side is equivalent to $[p]\,\mathrm{ret}\ a$. Indeed, by (dis) and (ret $\Box$), $[x \leftarrow \mathrm{ret}\ a; y \leftarrow [p]\,\mathrm{ret}\ a]\ \mathrm{ret}\ x \iff [x \leftarrow \mathrm{ret}\ a]\ \mathrm{ret}\ x \iff \mathrm{ret}\ a$, and by (dis) and (ret $\Box$), $[x \leftarrow \mathrm{ret}\ a; y \leftarrow [p]\,\mathrm{ret}\ a]\ \mathrm{ret}\ y \iff [y \leftarrow [p]\,\mathrm{ret}\ a]\ \mathrm{ret}\ y \iff [p]\,\mathrm{ret}\ a$. $\Box$

Definition 4.3. A term is in *product β-normal form* if it does not contain subterms of the form $\mathrm{fst}\langle t, u\rangle$ or $\mathrm{snd}\langle t, u\rangle$.

Lemma 4.4. Let $\Gamma \rhd t : B$ be in product β-normal form. If B is T-free, then t does not contain a subterm whose type is of the form TA.

Proof. Assume that B is T-free. By induction over the term structure of t, we prove that t does not have a subterm of type TA. For the term formation rules *(var)* and *(1)*, this is clear. For *(pair)* and the Ω-rules, we can apply the induction hypothesis. Due to the general assumption that the operations in the signature have T-free argument types, for rule *(app)*, we can directly apply the induction hypothesis as well. Concerning *(fst)* and *(snd)*, if $t \equiv g_1(\ldots g_n(v))$, where $g_i \in \{\mathrm{fst}, \mathrm{snd}\}$ and n is maximal, v must have been obtained either by rule *(pair)* (but this contradicts the fact that t is in product β-normal form) or by rule *(app)*, and then we can use the reasoning for *(app)*. The remaining rules do not yield a term of T-free type. $\Box$

We now discuss some structural properties of the calculus.

Theorem 4.5. The system of reduction rules

$$\begin{array}{lcl}
\mathrm{fst}\langle t, u\rangle & \rightarrowtail & t \\
\mathrm{snd}\langle t, u\rangle & \rightarrowtail & u \\
\mathrm{do}\ x \leftarrow \mathrm{ret}\ t; p & \rightarrowtail & p[t/x] \\
\mathrm{do}\ x \leftarrow (\mathrm{do}\ y \leftarrow p; q); r & \rightarrowtail & \mathrm{do}\ y \leftarrow p; x \leftarrow q; r \\
\Box\Box p & \rightarrowtail & \Box p
\end{array}$$

is confluent and strongly normalising.

Proof. Termination. Consider two reduction relations: $\overset{\lambda}{\rightarrowtail}$ and $\overset{\Box}{\rightarrowtail}$, the first of which represents one-step reduction by one of the first four reduction rules, and the second one is a one-step reduction by the $\Box$-rule. Both these relations are terminating. Termination of the first reduction relation is proved in [BBdP98]. Termination of the second reduction relation is trivial, because it decreases the number of boxes.

It is easy to check that the relations $\overset{\Box}{\rightarrowtail}$ and $\overset{\lambda}{\rightarrowtail}$ have the following interchange property:

$$\text{if } t \overset{\Box}{\rightarrowtail}\overset{\lambda}{\rightarrowtail} s \text{ than } t \overset{\lambda}{\rightarrowtail}\overset{\Box}{\rightarrowtail} \ldots \overset{\Box}{\rightarrowtail} s, \tag{1}$$

where the number of $\overset{\Box}{\rightarrowtail}$ reductions on the right-hand side depends on the chosen rule, the term t, and the redex being rewritten. Now, we prove termination by contradiction. Suppose we have an infinite sequence of reductions

$$t \rightarrowtail\rightarrowtail \ldots$$

If there are only finitely many $\overset{\lambda}{\rightarrowtail}$ reductions in the sequence, then the sequence eventually becomes a non-terminating sequence of $\overset{\Box}{\rightarrowtail}$ reductions, contradiction. Otherwise, we can gather reductions of type $\overset{\lambda}{\rightarrowtail}$ at the beginning, using property (1). We can thus construct a non-terminating sequence of $\overset{\lambda}{\rightarrowtail}$ reductions, a contradiction.

Confluence. One easily checks that $\overset{\Box}{\rightarrowtail}$ and $\overset{\lambda}{\rightarrowtail}$ *commute*, i.e.

$$\text{if } t \overset{\Box}{\rightarrowtail} s \text{ and } t \overset{\lambda}{\rightarrowtail} s' \text{ then there exists } r \text{ such that } s \overset{\lambda}{\rightarrowtail} r \text{ and } s' \overset{\Box}{\rightarrowtail} \ldots \overset{\Box}{\rightarrowtail} r.$$

Thus, confluence of $\rightarrowtail$ follows from confluence of $\overset{\Box}{\rightarrowtail}$ and $\overset{\lambda}{\rightarrowtail}$ by the Hindley-Rosen Lemma [Hin64]. □

In the sequel, the terms *rewriting* and *normal form* will always refer to the above rule system.

Proposition 4.6. Let $\varphi : P\Omega$ be a monadic formula in product β-normal form having a subterm $r : TA$. Then there is a provably equivalent monadic formula ψ with $\varphi \rightarrowtail^* \psi$ (ψ is even a subformula of φ) such that ψ has one of the following forms:

1. x, where $x : P\Omega$ is a variable,
2. $\Box x$, where $x : T\Omega$ is a variable,
3. $\mathrm{do}\ \bar{x} \leftarrow \bar{p}; q$,
4. $\Box(\mathrm{do}\ \bar{x} \leftarrow \bar{p}; q)$,
5. $\mathrm{ret}\ t$, where $t : \Omega$,
6. $\Box\,\mathrm{ret}\ t$, where $t : \Omega$.

Proof. We proceed by induction on φ. We have the following cases:

1. φ is a variable: then we are done.

2. $\varphi \equiv f(t)$, $f \in \Sigma$. By our general assumption, the type of t is T-free. Hence, by Lemma 4.4, r does not occur in φ, a contradiction.

3. $\varphi \equiv g_1(\dots g_n(t))$, where $g_i \in \{\mathrm{fst}, \mathrm{snd}\}$ and $n \geq 1$ is maximal. We have the following subcases:

- $t \equiv f(t_1)$. Again, Lemma 4.4 leads to a contradiction.
- $t \equiv \langle u, v\rangle$. This is a contradiction to the fact that φ is in product β-normal form.
- t is a variable. For typing reasons, t cannot be r, hence r does not occur in φ, a contradiction.

4. $\varphi \equiv \mathrm{ret}\ t$. If $\varphi \equiv r$, then we are done. Otherwise r is a subterm of t. But since $t : \Omega$, Lemma 4.4 leads to a contradiction.

5. $\varphi \equiv \mathrm{do}\ \bar{x} \leftarrow \bar{p}; q$: then we are done.

6. $\varphi \equiv \Box q$. We basically have the same case distinction for q again, with similar arguments. In the case that $q \equiv \Box q_0$, we have $\varphi \equiv \Box\Box q_0$, which by (□MDL) is equivalent to (and can also be reduced to) $\Box q_0$, hence we can use the induction hypothesis for $\Box q_0$.

□

Using this result, we can apply a leftmost-outermost reduction strategy to obtain

Proposition 4.7. An MDL formula φ is provably equivalent to its normal form.

Proof. Since the rewriting system is confluent and strongly normalising, we can use any strategy to compute the normal form. We always use the CC-rules (justified by axioms (CC)) to compute the CC-normal form between applications of the other rules, which are applied in a leftmost-outermost fashion.

We proceed by induction over φ. Assume that φ contains any further redex (i.e. not reducible with the CC-rules) $p : TA$. By Prop. 4.6, we can assume that φ is of form $\mathrm{do}\ \bar{x} \leftarrow \bar{p}; q$ or $\Box\,\mathrm{do}\ \bar{x} \leftarrow \bar{p}; q \equiv [\bar{x} \leftarrow \bar{p}]\, q$ — the other possible forms listed in Prop. 4.6 are already in normal form. We can use (seq□), (K) and (nec) to shift the focus among the p_i, and reflect the reductions in the calculus with (ret□) and (ctr□) — if φ is of form $\mathrm{do}\ \bar{x} \leftarrow \bar{p}; q$, additionally (□MDL) is needed. Using this outermost rewriting, we eventually arrive at some (possibly boxed) sequence of *do*-terms $\mathrm{do}\ \bar{y} \leftarrow \bar{q}; r$ such that r is not a *do*-term. If r contains any redex $p : TA$, an analysis similar to cases in Prop. 4.6 shows that cases 1. to 4. are not possible. Case 5. already has been excluded, and in case 6. we just apply the induction hypothesis. Similarly, the q_i cannot be of the form $\mathrm{ret}\ u$ or $\mathrm{do}\ z \leftarrow r_1; r_2$, since then outermost rewriting of φ would not have been completed yet. If q_i is a variable or of form $f(t)$, $\mathrm{fst}(t)$ or $\mathrm{snd}(t)$, using arguments similar to those in Prop. 4.6, we see that no further rewriting is possible. Finally, if q_i is of form $\Box r$, with (cong□), we can normalise $\Box r$ using the induction hypothesis. □

This result in turn leads to further admissible rules:

Proposition 4.8. The following rules are admissible:

$$\textbf{(cong)}\ \frac{\varphi \Leftrightarrow \psi}{\chi[\varphi/x] \Leftrightarrow \chi[\psi/x]} \qquad \textbf{(subst)}\ \frac{\varphi}{\varphi[t/x]}.$$

The following equivalences are derivable:

(comm) $[x \leftarrow \varphi; y \leftarrow \psi]\,\chi \iff [y \leftarrow \psi; x \leftarrow \varphi]\,\chi$.

Finally, each lifted tautology $\varphi : P\Omega$ can be derived.

Proof. (cong): Occurrences of φ outside (but possibly under) boxes $[x \leftarrow p]$ can be exchanged for ψ by means of propositional reasoning with (K) and necessitation. Inside boxes, normalise the program according to Prop. 4.7, split the box using (seq$\Box$), and then exchange φ for ψ by means of (cong$\Box$).

(subst) can be proved by induction over proofs, noting that these are closed under substitution.

(comm) is proved in a way similar to the proof of Proposition 4.28 in [SM04a].

Concerning the lifted tautologies, from (taut), the definition of lifted Boolean operations, and (ret$\Box$), we can derive each lifted propositional tautology a, as follows: the decoding of a can be brought into the form

$$[\bar{x} \leftarrow \bar{\psi}]\,\mathrm{ret}\ t,$$

where $t : \Omega$ is a propositional tautology, using

- ($\Box$MDL) and (cong) to get rid of nested boxes,
- (ctr$\Box$) to get rid of nested do's,
- (ret$\Box$) to collect all propositional connectives in t, and
- (copy) to remove duplicate occurrences of 'atoms'.

The normalised monadic formula can then be derived using (taut) and (nec).

□

Example 4.9 (Swap of two variables). We illustrate the usage of the calculus for MDL by proving a simple example about swapping the values of two variables. To that end we make use of the background axiomatisation of dynamic references from Example 3.1. Our goal is to prove that the program $(\mathrm{do}\ a \leftarrow {*r}; b \leftarrow {*s}; r := b; s := a)$ being run under the assumption $r \neq s$ swaps the values under the references r and s. This is justified by the following derivation:

$$\begin{aligned}
&(x = {*r} \wedge y = {*s} \wedge r \neq s) \\
&\quad\Longrightarrow [a \leftarrow {*r}](x = {*r} \wedge y = {*s} \wedge r \neq s \wedge a = x) && \text{(step 1)}\\
&\quad\Longrightarrow [a \leftarrow {*r}; b \leftarrow {*s}](x = {*r} \wedge y = {*s} \wedge r \neq s \wedge a = x \wedge b = y) && \text{(step 2)}\\
&\quad\Longrightarrow [a \leftarrow {*r}; b \leftarrow {*s}; r := b](y = {*s} \wedge r \neq s \wedge a = x \wedge b = y \wedge b = {*r}) && \text{(step 3)}\\
&\quad\Longrightarrow [a \leftarrow {*r}; b \leftarrow {*s}; r := b; s := a](a = x \wedge b = y \wedge b = {*r} \wedge a = {*s}) && \text{(step 4)}\\
&\quad\Longrightarrow [a \leftarrow {*r}; b \leftarrow {*s}; r := b; s := a](y = {*r} \wedge x = {*s}) && \text{(step 5)}
\end{aligned}$$

Here every step calls a series of axioms. Specifically, steps 1 and 2 essentially follow by (copy) and (comm), steps 3 and 4 call (K3), (read-write), (read-write-other), (taut), (nec) and step 5 essentially follows from the background axiomatisation of the equality predicate. Let us prove step 3 in detail. We have

$$\begin{aligned}
&x = {*r} \wedge y = {*s} \wedge r \neq s \wedge a = x \wedge b = y \\
&\quad\Longrightarrow y = {*s} \wedge r \neq s \wedge a = x \wedge b = y \wedge [r := b](r = {*b}) && \text{(read-write)}\\
&\quad\Longrightarrow [r := b](y = {*s} \vee s = r) \wedge r \neq s \wedge a = x \wedge b = y \wedge [r := b](r = {*b}) && \text{(read-write-other)}\\
&\quad\Longrightarrow [r := b](y = {*s} \vee s = r) \wedge [r := b](r \neq s \wedge a = x \wedge b = y) \wedge [r := b](r = {*b}) && \text{(K3)}\\
&\quad\Longrightarrow [r := b]((y = {*s} \vee s = r) \wedge r \neq s \wedge a = x \wedge b = y \wedge r = {*b}) && \text{(K), (taut), (nec)}\\
&\quad\Longrightarrow [r := b](y = {*s} \wedge r \neq s \wedge a = x \wedge b = y \wedge r = {*b}). && \text{(K), (taut), (nec)}
\end{aligned}$$

After applying **(nec)** to the obtained implication, we have

$$\begin{aligned}
[a \leftarrow {*r}; b \leftarrow {*s}](&x = {*r} \wedge y = {*s} \wedge r \neq s \wedge a = x \wedge b = y \Rightarrow \\
&[r := b](y = {*s} \wedge r \neq s \wedge a = x \wedge b = y \wedge r = {*b}))
\end{aligned}$$

and then we are done by (K).

5. Observational Purity

Our notion of purity has some practical limitations: The program $\mathrm{do}\ r \leftarrow new\ x; \mathrm{ret}\ *$ is *not* pure, because it modifies the state. In particular, it is not discardable, since it generally makes a difference whether a new memory cell is allocated or not. However, we generally cannot *observe* whether a memory cell is allocated (unless we program in C, where a function *int2loc*, written e.g. as "`(*char)`" is available). This leads to the following notions:

Given a background MDL theory Φ (that axiomatises a given combination of effects), two programs $p : TA$ and $q : TA$ in context $\Gamma = (x_1 : B_1, \dots, x_n : B_n)$ are *observationally equivalent*, $p \approx q$, iff for all closed program sequences $\bar{x} \leftarrow \bar{r}$ (where closedness means that every program r_i mentions only the previously bound variables $x_1, \dots, x_{i-1}$) and all monadic formulae $\varphi : P\Omega$ in context $\Gamma, y : A$,

$$\Phi \vdash [\bar{x} \leftarrow \bar{r}; y \leftarrow p]\,\varphi \iff [\bar{x} \leftarrow \bar{r}; y \leftarrow q]\,\varphi. \tag{2}$$

A program is *observationally pure* if the equations for discardability, copyability and commutation hold up to observational equivalence. For example, in the case of discardability, this means that

$$\Phi \vdash [\bar{x} \leftarrow \bar{r}; y \leftarrow (\mathrm{do}\ z \leftarrow p; \mathrm{ret}\ *)]\,\varphi \iff [\bar{x} \leftarrow \bar{r}; y \leftarrow \mathrm{ret}\ *]\,\varphi,$$

which amounts to $\Phi \vdash [\bar{x} \leftarrow \bar{r}]\,\varphi \iff [\bar{x} \leftarrow \bar{r}; p]\,\varphi$.

Deviating from [MSG08], in the definition of observational equivalence, we impose the restriction that the equivalence (2) above does not contain free variables. This avoids the possibility of injecting arbitrary semantic values into the formulae, which would make too many things observable. This restriction makes it necessary to introduce an additional context $\bar{x} \leftarrow \bar{r}$ of closed programs that makes all values obtainable by computations available for observations. Since the notion of observational equivalence depends on the observations that can be made, this also implies that it is dependent on the available operations and hence will vary from programming language to programming language.

This means that we cannot say in detail what observational purity is in our example monads, except from the general statement that a program is observationally pure if it is pure or differs from a pure program only in a way that cannot be observed. We will, however, elaborate some specific cases in more detail.

In particular, the notion of observational purity allows us to distinguish the behaviour of references in Java and in C. We therefore introduce two simple languages, Toy Java and Toy C. Toy Java has been specified in Example 3.1 above (although real-life Java does not have explicit reference types, it does pass every structured object using a reference). Recall that it provides, for any basic type A, a (basic) type $Ref\ A$ of references to objects of type A, and the following operations:

$$*\_\_ : Ref\ A \to P\ A$$
$$\_\_ := \_\_ : (Ref\ A) \times A \to T\ 1$$
$$new : A \to T\ (Ref\ A)$$

While Toy Java treats references as an abstract data type that is fully encapsulated, references in Toy C are just integers. Thus, we have exactly the same specification as for Toy Java, except that the profile of the operations is now

$$*\_\_ : Int \to P\ A$$
$$\_\_ := \_\_ : Int \times a \to T\ 1$$
$$new : A \to T\ Int$$

Example 5.1. Due to the lack of encapsulation of references, writing on a newly allocated reference in Toy C fails to be observationally pure: If we put

$$nwr\ v = \mathrm{do}\ u \leftarrow new\ v; u := 1; *u$$

then $nwr\ v$ is not even discardable. To see this, let φ be the monadic formula

$$[z; x \leftarrow *n]\,(x = 1)$$

where n is some integer literal. Then $\varphi[nwr\ v/z] \iff \varphi[\mathrm{ret}\,()/z]$ is not provable from the axiomatisation of Toy C, since it fails to be semantically valid: it may or may not happen that the integer n coincides with the newly allocated reference. Note that this effect is excluded in Toy Java, because we cannot write down the corresponding formula: the

only way to get hold of a reference in Toy Java is to create it via new, while in the above formula we have just used an integer literal. The following example formalises this point.

Example 5.2. In Toy Java, writing on a newly created reference and subsequently reading from it, i.e. programs such as

$$nwr\ v = \mathrm{do}\ u \leftarrow new\ v; u := 1; {*u}$$

are observationally pure. In fact we shall prove that

any closed term $t : T1$ *is observationally equal to* $\mathrm{ret} * $ *(and therefore observationally pure).* (3)

This implies e.g. observational purity of $nwr\ v$, as for closed $r : TA$ and a monadic formula φ in context x, y, the axiomatisation of Toy Java allows deriving the following chain of equivalences:

$$\begin{aligned}
&[x \leftarrow r; y \leftarrow nwr\ v]\,\varphi \\
&\iff [x \leftarrow r; u \leftarrow new\ v; u := 1; y \leftarrow {*u}]\,\varphi \\
&\iff [x \leftarrow r; u \leftarrow new\ v; u := 1; y \leftarrow {*u}]\,(\mathrm{ret}\,(y = v) \wedge \varphi) && \text{(read-write)} \\
&\iff [x \leftarrow r; u \leftarrow new\ v; u := 1; y \leftarrow {*u}]\,\varphi[v/y] && \text{(read-write)} \\
&\iff [x \leftarrow r; u \leftarrow new\ v; u := 1]\,\varphi[v/y] && \text{(dis)} \\
&\iff [x \leftarrow r]\,\varphi[v/y] && (3) \\
&\iff [x \leftarrow r; y \leftarrow \mathrm{ret}\ v]\,\varphi
\end{aligned}$$

(with implicit steps for propositional and equational reasoning, the latter supported by the axiomatisation of $=$) which proves that $nwr\ v$ is observationally equivalent to $\mathrm{ret}\ v$, hence observationally pure.

To prove (3), let $t : T1$, and let $z : T1 \rhd \varphi : P\Omega$; we have to show $\varphi[t/z] \iff \varphi[\mathrm{ret} * /z]$, as follows. First, using ($\Box$MDL), add a box in front of φ, if there is none. According to Proposition 4.7 we can assume φ to be normal. Moreover, successively replace every subterm in the form $\Box\,\mathrm{do}\ \bar{x} \leftarrow \bar{p}; \Box q$ by $\Box\,\mathrm{do}\ \bar{x} \leftarrow \bar{p}; q$. It is easily seen that this transformation is sound (using (do$\Box$)), terminating, and does not spoil normality. In the end φ reaches the form $\Box(\mathrm{do}\ \bar{x} \leftarrow \bar{p}; \mathrm{ret}\ b)$ or equivalently $[\bar{x} \leftarrow \bar{p}]\,(\mathrm{ret}\ b)$. By normality, all the variables in $[\bar{x} \leftarrow \bar{p}]\,(\mathrm{ret}\ b)$ are either of basic sorts or of reference sorts, and no projections or pairings occur. Note that b is a boolean combination of equalities (boolean variables are treated as equalities, i.e. $b = \top$ for $b : \Omega$) and boolean constants. For $b = \top$, $[\bar{x} \leftarrow \bar{p}]\ \mathrm{ret}\ \top \iff \mathrm{ret}\ \top$. For $b = \bot$, the equivalence $[\bar{x} \leftarrow \bar{p}]\ \mathrm{ret}\ \bot \iff \mathrm{ret}\ \bot$, which expresses termination of the program sequence, may easily be proved by induction, where the base case ('all atomic programs terminate') is directly expressed by the Toy Java axioms. Otherwise b is a boolean combination of equalities. Thus, we are done as soon as we prove the following.

For any variables x_i and x_j of coincident reference sorts, either $[\bar{x} \leftarrow \bar{p}]\ \mathrm{ret}\,(x_i = x_j)$ or $[\bar{x} \leftarrow \bar{p}]\ \mathrm{ret}\,(x_i \neq x_j)$ is provable, and for every variable x_i of base sort there is a closed term q_{x_i} such that $[\bar{x} \leftarrow \bar{p}]\ \mathrm{ret}\,(x_i = q_{x_i})$ is provable (note that this also gives complete information about equalities among non-closed terms at base type, by substituting q_{x_i} for x_i and evaluating the resulting closed terms). Moreover, the result of such an evaluation is invariant under exchanging closed subterms of type $T1$ in $\bar{p}$. We refer to this property briefly as *invariance* and to the formula $x_i = x_j$ in question as the *target test*.

We prove this claim by well-founded induction; a discussion of the actual induction ordering and the relevant cases to be considered will follow further below. If the program sequence $\bar{p}$ is non-empty (otherwise the statement becomes trivial), consider the last binding in it, say $x_n \leftarrow p_n$. We distinguish the following cases.

1. $p_n \equiv \Box q$. By the induction hypothesis and the fact that Ω is a boolean algebra, either $[x_1 \leftarrow p_1; \ldots; x_{i-1} \leftarrow p_{i-1}; x_n \leftarrow q]\ \mathrm{ret}\,(x_n = \top)$ or $[x_1 \leftarrow p_1; \ldots; x_{n-1} \leftarrow p_{n-1}; x_n \leftarrow q]\ \mathrm{ret}\,(x_n = \bot)$ is provable, and in each case invariance holds. In the former case, $[\bar{x} \leftarrow \bar{p}]\ \mathrm{ret}\,(x_n = \top)$ is provable by ($\Box$) and invariance holds. In the latter case, we have the same situation, except that the implication $[x_n \leftarrow q]\ \mathrm{ret}\,(\neg x_n) \implies [x_n \leftarrow \Box q]\ \mathrm{ret}\,(\neg x_n)$, rather than being universally valid, relies on the fact that q terminates as observed above, i.e. that $\neg[q]\,\mathrm{ret}\ \bot$ is derivable. In fact, using termination of q and the K-axiom, we derive from $[x_n \leftarrow q]\ \mathrm{ret}\,(\neg x_n)$ that $\neg[x_n \leftarrow q]\,\mathrm{ret}\ x_n$. The latter is, by (unit), equivalent to $\neg\Box q$, which in turn is by its definition equivalent to $[x_n \leftarrow \Box q]\ \mathrm{ret}\,(\neg x_n)$.
2. $p_n = \mathrm{do}\ y \leftarrow s; r$. For every b, the monadic formula $[\bar{x} \leftarrow \bar{p}]\ \mathrm{ret}\ b$, i.e. $[x_1 \leftarrow p_1; \ldots; x_{i-1} \leftarrow p_{i-1}; x_n \leftarrow (\mathrm{do}\ y \leftarrow s; r)]\ \mathrm{ret}\ b$, is provably equivalent to $[x_1 \leftarrow p_1; \ldots; x_{i-1} \leftarrow p_{i-1}; y \leftarrow s; x_n \leftarrow r]\ \mathrm{ret}\ b$ and we are done by the induction hypothesis.

3. $p_n = \mathrm{ret}\ s$. Here s must be of boolean type, as explained below. By induction, we can prove either $[x_1 \leftarrow p_1; \ldots; x_{i-1} \leftarrow p_{i-1}]\ \mathrm{ret}\ s$ or $[x_1 \leftarrow p_1; \ldots; x_{i-1} \leftarrow p_{i-1}]\ \mathrm{ret}\,(\neg s)$, and in each case invariance holds. This immediately implies that we can prove either $[\bar{x} \leftarrow \bar{p}]\ \mathrm{ret}\,(x_n = \top)$ or $[\bar{x} \leftarrow \bar{p}]\ \mathrm{ret}\,(x_n = \bot)$, respectively, and invariance holds in each case.
4. p_n is an atomic program, i.e. either assignment or allocation or dereferencing. If the truth value of the final monadic formula does not depend on x_n, then p_n may be safely discarded, thus discharging this case by induction; this is applicable if p_n is of type $T1$ (i.e. p_n is an assignment) and if the final monadic formula does not mention x_n. Note that in the first case, invariance stipulates that p_n itself can be invariantly exchanged for another term of the same type; this is evidently the case, as the previous argument applies to *any* term of type $T1$. Next, suppose that $p_n = new\ v$ for some v, and that the target test is of the form $x_n = x_i$ for some i. If p_i is atomic, then it can only be an another new operator, and we have $[\ldots; x_i \leftarrow new\ w; \ldots; x_n \leftarrow new\ v]\,\mathrm{ret}\,(x_n \neq x_i)$ by axiom (new-distinct) in the definition of Toy Java. If p_i is not atomic, then it is either a binding or a box; both cases are reduced to the induction hypothesis similarly as in the previous steps.
   Finally, let $p_n = {*}v$. We need to show that for some closed term t $[\bar{x} \leftarrow \bar{p}]\ \mathrm{ret}\,(x_n = t)$ is provable. We proceed by case distinction over p_{n-1}. If p_{n-1} is atomic, we are done by the following equivalences and the induction hypothesis.

$$
\begin{aligned}
[x_{n-1} \leftarrow new\ w; x_n \leftarrow {*}v]\,\mathrm{ret}\,(x_n = t) &\iff [x_n \leftarrow {*}v]\,\mathrm{ret}\,(x_n = t) && (x_{n-1} \neq v,\ \text{(read-new-other)})\\
[x_{n-1} \leftarrow new\ w; x_n \leftarrow {*}x_{n-1}]\,\mathrm{ret}\,(x_n = w) &\iff \mathrm{ret}\ \top && \text{(read-new)}\\
[u := w; x_n \leftarrow {*}v]\,\mathrm{ret}\,(x_n = t) &\iff [x_n \leftarrow {*}v]\,\mathrm{ret}\,(x_n = t) && (v \neq u,\ \text{(read-write-other)})\\
[u := w; x_n \leftarrow {*}u]\,\mathrm{ret}\,(x_n = w) &\iff \mathrm{ret}\ \top && \text{(read-write)}\\
[x_{n-1} \leftarrow {*}w; x_n \leftarrow {*}v]\,\mathrm{ret}\,(x_n = t) &\iff [x_n \leftarrow {*}v]\,\mathrm{ret}\,(x_n = t) && (t\ \text{closed; (dis)})
\end{aligned}
$$

   Otherwise, reduce to the induction hypothesis as in the previous cases.

A few comments about the above induction are needed. First, the induction is indeed well-founded, because every step decreases the number of bindings, boxes, returns, or atomic programs, respectively. Second, the normality of the formula as assumed in the beginning can be partially spoiled, because in case 1, the removal of a box may unblock application of a unit law or associativity. The arising additional cases show up above as cases 2 and 3, which would be unnecessary for a fully normal formula; we may restrict these additional cases to $p_n : T\Omega$, as removing boxes is the only source for such terms.

The example may by modified in various ways. For example, if switch to a non-deterministic Java monad and let new non-deterministically return an arbitrary free memory cell (instead of the next free one), nwr remains observationally pure in Toy Java, while in Toy C, it is not even copyable (since the non-determinism can be observed). If we drop the assignment operation from Toy Java, even new becomes pure, because we can no longer observe the allocation of new memory cells.

The example may also be used to illustrate that the notion of observational purity is, unsurprisingly, quite sensitive to modifications of the underlying logic. More to the point, already slight increases in expressiveness may destroy the above proof of observational purity. E.g. if the logic is extended by universal quantification, or even if we just admit free variables in the definition of observational equality, then the above program $nwr\ v$ fails to be observationally pure in the extended logic. To see this, consider the formula

$$\varphi \equiv \langle z; r \leftarrow new\ x\rangle(s = r)$$

with an additional free variable s of reference type besides the placeholder $z : T1$. It says that it may happen that after executing z, s is a free reference which might be allocated by the next new statement. Then $\varphi[nwr\ v/z] \iff \varphi[\mathrm{ret}\,()/z]$ is not provable from the axiomatisation of Toy Java (as it can easily be falsified semantically, for the obvious reason that $nwr\ v$ allocates a reference). This shows that monadic dynamic logic constitutes a well-balanced environment for observational purity.

6. Completeness

The completeness proof for MDL is based on a term model construction. Given a signature Σ and a set Φ of MDL formulae over Σ, we construct a category $\mathbf{C}_{\Sigma,\Phi}$ as follows: the objects of $\mathbf{C}_{\Sigma,\Phi}$ are the types of Σ, and morphisms $t : A \to B$ are terms in context

$$x : A \rhd t : B,$$

taken modulo *contextual equivalence*: $t \sim u$ iff for all MDL formulae φ with one free variable $x : A$,

$$\Phi \vdash \varphi[t/x] \iff \varphi[u/x] \tag{4}$$

(this is obviously a congruence). Identities are given by variables $[x : A \rhd x : A]_\sim$, and composition by substitution

$$[y : B \rhd u : C]_\sim \circ [x : A \rhd t : B]_\sim := [x : A \rhd u[t/y] : C]_\sim.$$

Using axiom (ret$\Box$), one easily proves that this is well-defined and obeys the identity and associativity laws of a category. The basic types of Σ are interpreted as themselves, and so are the basic operations:

$$[\![f : A \to B]\!] := [x : A \rhd f(x) : B]_\sim.$$

The category $\mathbf{C}_{\Sigma,\Phi}$ comes with a canonical cartesian structure, which on objects is just given by taking product types as categorical products, and the unit type as the terminal object. Projections are $\pi_1 := [x : A \times B \rhd \mathrm{fst}(x) : A]_\sim$ and $\pi_2 := [x : A \times B \rhd \mathrm{snd}(x) : B]_\sim$, pairing of morphisms is $\langle [t]_\sim, [u]_\sim \rangle := [\langle t, u \rangle]_\sim$, and the unique morphism into the terminal object is $!_A := [x : A \rhd * : 1]_\sim$. Axiom (CC) ensures that this does define a cartesian structure.

The problem with contextual equivalence is that, although its definition is simple and intuitive, it can be quite hard to prove that given monadic programs are contextually equivalent. The key result is

Theorem 6.1. For terms of type TA, contextual equivalence $\sim$ coincides with observational equivalence $\approx$.

Proof. We first show a lemma on contextual equivalence:

Lemma 6.2. For $p, q : TA$, $p \sim q$ iff (4) holds for the following cases.

1. $\varphi \equiv \Box x$
2. $\varphi \equiv \Box \,\mathrm{do}\ \bar{y} \leftarrow \bar{p}; p_0$, and if x occurs freely in p_i, then $p_i \equiv x$ $(i = 0, \dots, n)$.

Proof. By Prop. 4.7, we can assume that φ is in normal form. We proceed by induction on φ. If x is not a subterm of φ, we are done. If x is a subterm of φ, by Prop. 4.6, it suffices to consider four subcases:

- $\varphi \equiv x$. By ($\Box$MDL), we can reach the next case.
- $\varphi \equiv \Box x$. Use assumption 1.
- $\varphi \equiv \mathrm{do}\ \bar{x} \leftarrow \bar{p}; p_0$. By ($\Box$MDL), we can reach the last case.
- $\Box \,\mathrm{do}\ \bar{x} \leftarrow \bar{p}; p_0$. For each p_i $(i = 0, \dots, n)$, we have the following subcases, in all of which the premises of assumption 2 are fulfilled:

1. p_i is a variable. Then it is immediate that either $p_i \equiv x$, or x does not occur freely in p_i.
2. $p_i \equiv f(t)$, $f \in \Sigma$. By our general assumption, the type of t is T-free, and hence by Lemma 4.4, x does not occur freely in p_i.
3. $p_i \equiv g_1(\dots g_n(t))$, where $g_i \in \{\mathrm{fst}, \mathrm{snd}\}$ and $n \geq 1$ is maximal. As in Prop. 4.6, we can show that x does not occur freely in p_i.
4. $p_i \equiv \mathrm{ret}\ t$. For $i > 0$, this contradicts φ being in normal from. For $i = 0$, it follows that $t : \Omega$, and hence by Lemma 4.4, x does not occur freely in t.
5. $p_i \equiv \Box r$. We use the induction hypothesis and (cong) to rewrite $p_i[t/x]$ to $p_i[u/x]$ within φ, and hence get rid of the free occurrences of x in p_i.
6. $p_i \equiv \mathrm{do}\ \bar{y} \leftarrow \bar{q}; r$. This contradicts φ being in normal from.

□

This completes the proof of the lemma. We now return to the Proof of Theorem 6.1. Obviously, contextual equivalence is included in observational equivalence. For the converse, let $t \approx u : TA$. By ($\Box$def), the first assumption of Lemma 6.2 is a special case of the second assumption. The second assumption follows from iterated application of the equivalence (where all terms except from p and q only mention the previously bound variables)

$$\text{for all } \bar{p}, \bar{q}, \varphi, \quad \Phi \vdash [\bar{x} \leftarrow \bar{p}; x \leftarrow p; \bar{y} \leftarrow \bar{q}]\,\varphi \iff [\bar{x} \leftarrow \bar{p}; x \leftarrow q; \bar{y} \leftarrow \bar{q}]\,\varphi \tag{5}$$

by successively replacing each occurrence of p by an occurrence of q. Now (5) follows from observational equivalence by noting that $y \leftarrow \bar{q}$ can become part of the φ. This covers the occurrences of p in p_i for $i > 0$.

For $i = 0$, we need to show $\Phi \vdash \Box \,\mathrm{do}\ \bar{y} \leftarrow \bar{p}; p \iff \Box \,\mathrm{do}\ \bar{y} \leftarrow \bar{p}; q$. But this follows from the instance of (5) $\Phi \vdash [\bar{y} \leftarrow \bar{p}; x \leftarrow p]\,\mathrm{ret}\ x \iff [\bar{y} \leftarrow \bar{p}; x \leftarrow q]\,\mathrm{ret}\ x$ by rewriting both sides with (unit), ($\Box$) and (ctr). □

We are now ready to complete the term model construction by constructing a monad $\mathbb{T}_{\Sigma,\Phi}$ on $\mathbf{C}_{\Sigma,\Phi}$. It is given by the following data:

$$\mathbb{T}_{\Sigma,\Phi}\, A := T\, A, \qquad \eta_A := [x : A \rhd \mathrm{ret}\ x : TA]_\sim,$$

and given $x : A \rhd q : TB$,

$$[x : A \rhd q : TB]^*_\sim := [p : TA \rhd \mathrm{do}\ x \leftarrow p; q : TB]_\sim$$

Well-definedness follows easily by Theorem 6.1. Finally, the strength is given by

$$t_{A,B} := [p : A \times TB \rhd \mathrm{do}\ x \leftarrow \mathrm{snd}(p); \mathrm{ret}\ \langle \mathrm{fst}(p), x\rangle : T(A \times B)]_\sim$$

In $\mathbf{C}_{\Sigma,\Phi}$, $Hom(A, \Omega)$ is coherently turned into a Boolean algebra by defining e.g.

$$[x : A \rhd t : \Omega]_\sim \wedge [x : A \rhd u : \Omega]_\sim = [x : A \rhd t \wedge u : \Omega]_\sim$$

Lemma 6.3. $\hom(A, \Omega)$ is coherently a Boolean algebra, that is, Ω is an internal Boolean algebra object.

Proof. We show a sample Boolean algebra law, namely commutativity of $\wedge$. We note that by (ret$\Box$) $\mathrm{ret}\ a \wedge \mathrm{ret}\ b \iff \mathrm{ret}\ (a \wedge b)$. Now commutativity of $\wedge$ follows from $a \wedge b \sim b \wedge a$. By contextual equivalence, we need to show

$$\Phi \vdash \varphi[a \wedge b/x] \iff \varphi[b \wedge a/x]$$

for all φ. Now

$$\begin{aligned}
& \varphi[a \wedge b/x] \\
(ret\Box) \iff & [x \leftarrow \mathrm{ret}\ (a \wedge b)]\, \varphi \\
(cong) \iff & [x \leftarrow \mathrm{ret}\ a \wedge \mathrm{ret}\ b)]\, \varphi \\
(taut), (cong) \iff & [x \leftarrow \mathrm{ret}\ a \wedge \mathrm{ret}\ b)]\, \varphi \\
\cdots & \\
\iff & \varphi[b \wedge a/x]
\end{aligned}$$

Coherence of the Boolean algebra structure follows since composition is substitution (Lem. 3.3). □

We also need to show that the generalised elements of (i.e. morphisms into) PA are pure, i.e. discardable and copyable. Concerning discardability, for $\varphi : PA$, we need to show $(\mathrm{do}\ x \leftarrow \varphi; \mathrm{ret}\ *) \sim \mathrm{ret}\ *$, i.e.

$$\Phi \vdash [a \leftarrow (\mathrm{do}\ x \leftarrow \varphi; \mathrm{ret}\ *)]\, \psi \iff [a \leftarrow \mathrm{ret}\ *]\, \psi.$$

By (ret$\Box$), $[a \leftarrow \mathrm{ret}\ *]\, \psi \iff \psi$. By (ret$\Box$) and (dis), $\psi \iff [x \leftarrow \varphi; \mathrm{ret}\ *]\, \psi$. By (ctr$\Box$), this is in turn equivalent to $[a \leftarrow (\mathrm{do}\ x \leftarrow \varphi; \mathrm{ret}\ *)]\, \psi$. The proof of copyability is similar and uses axiom (copy).

Lemma 6.4. In $\mathbf{C}_{\Sigma,\Phi}$, $(id_A \times \top : A \times 1 \to A \times \Omega, id_A \times \bot : A \times 1 \to A \times \Omega)$ is an episink.

Proof. Since composition in the term model is substitution, we need to show that for all $x : A \times \Omega \rhd t : B$, $x : A \times \Omega \rhd u : B$, we have

$$t[\langle y : a, \top\rangle/x] \sim u[\langle y, \top\rangle/x] \text{ and } t[\langle y : a, \bot\rangle/x] \sim u[\langle y, \bot\rangle/x] \text{ imply } t \sim u.$$

By definition of contextual equivalence, this means that for any φ (w.l.o.g. assuming that x is not free in φ), we have, when defining $\psi :\equiv \varphi[t/z] \iff \varphi[u/z]$, that

$$\psi[\langle y, \top\rangle/x] \text{ and } \psi[\langle y, \bot\rangle/x] \text{ imply } \psi.$$

By (taut), we have $\mathrm{ret}\,((a \iff \top) \vee (a \iff \bot))$. Moreover, by (cong$\Leftrightarrow$), we have $\mathrm{ret}\,(a \iff \top) \Rightarrow \psi[\langle y, a\rangle/x] \iff \psi[\langle y, \top\rangle/x]$ and $\mathrm{ret}\,(a \iff \bot) \Rightarrow \psi[\langle y, a\rangle/x] \iff \psi[\langle y, \bot\rangle/x]$. Assuming $\psi[\langle y, \top\rangle/x]$ and $\psi[\langle y, \bot\rangle/x]$, with tautological reasoning, we obtain $\psi[\langle y, a\rangle/x]$. By the Substitution Lemma, $\psi[\langle \mathrm{fst}(x), \mathrm{snd}(x)\rangle/x]$. By (CC), ψ. □

To complete the construction of the term model, we put $\iota_A : PA \to TA := [p : PA \rhd p : TA]_\sim$, and $\Box : T\Omega \to P\Omega := [p : T\Omega \rhd \Box p : P\Omega]_\sim$.

As a first important property of the term model $\mathbb{T}_{\Sigma,\Phi}$, we prove:

Lemma 6.5. In $\mathbb{T}_{\Sigma,\Phi}$, terms are essentially (modulo tupling) interpreted by themselves, that is

$$[\![x_1 : A_1, \ldots, x_n : A_n \rhd t : A]\!] = [x : A_1 \times \ldots \times A_n \rhd t[\pi_i^n(x)/x_i] : A]_\sim$$

Proof. Induction over the term structure. The only interesting case is that of do-terms. Let $\Gamma \equiv x_1 : A_1 \times \ldots \times x_n : A_n$. Then we have

$$\begin{aligned}
&[\![\Gamma \rhd \text{do } x \leftarrow p; q : TB]\!] \\
&= [\![\Gamma, x : A \rhd q : TB]\!]^* \circ t_{[\![\Gamma]\!],A} \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd p : TA]\!]\rangle \\
&= \text{(induction hypothesis)} \\
&\quad [z : A_1 \times \ldots \times A_n \times A \rhd q[\pi_i^{n+1}(z)/x_i, \pi_{n+1}^{n+1}(z)/x]]^*_\sim \\
&\quad \circ [r : (A_1 \times \ldots \times A_n) \times TA \rhd \text{do } y \leftarrow \text{snd}(r); \text{ret}\,\langle \text{fst}(r), y\rangle]_\sim \\
&\quad \circ [v : A_1 \times \ldots \times A_n \rhd \langle v, p[\pi_i^n(v)/x_i]\rangle : (A_1 \times \ldots \times A_n) \times TA]_\sim \\
&= [s : T(A_1 \times \ldots \times A_n \times A) \rhd \text{do } x \leftarrow s; q[\pi_i^{n+1}(z)/x_i, \pi_{n+1}^{n+1}(z)/x]]_\sim \\
&\quad \circ [\text{do } y \leftarrow p[\pi_i^n(v)/x_i]; \text{ret}\,\langle x, y\rangle]_\sim \\
&= [v : A_1 \times \ldots \times A_n \rhd \text{do } z \leftarrow (\text{do } y \leftarrow p[\pi_i^n(v)/x_i]; \text{ret}\,\langle x, y\rangle); \\
&\qquad\qquad q[\pi_i^{n+1}(z)/x_i, \pi_{n+1}^{n+1}(z)/x]\ ]_\sim \\
&= [v : A_1 \times \ldots \times A_n \rhd (\text{do } x \leftarrow p; q)[\pi_i^n(v)/x_i]]_\sim
\end{aligned}$$

where the last steps involve several applications of Theorem 6.1 in connection with various proof rules. □

Corollary 6.6 (Full abstractness). In $\mathbb{T}_{\Sigma,\Phi}$, $[\![t]\!] = [\![u]\!]$ iff $t \sim u$.

Proof. $t[\pi_i^n(x)/x_i][\langle x_1, \ldots, x_n\rangle/x] \sim t$ by axiom (CC). □

By Theorem 6.1, we also have

Corollary 6.7. In $\mathbb{T}_{\Sigma,\Phi}$, for $p, q : TA$, $[\![p]\!] = [\![q]\!]$ iff $p \approx q$.

The crucial properties of the term model construction are summarised in Proposition 6.8 and Lemma 6.10 below.

Proposition 6.8. $\mathbb{T}_{\Sigma,\Phi}$ is a fully abstract simple strong monad admitting dynamic logic.

The proof needs the following lemma.

Lemma 6.9. If $\Phi \vdash [x \leftarrow p]\,\varphi$, then $\text{do } x \leftarrow p; \text{ret}\,(\bar{x}, \varphi) \sim \text{do } x \leftarrow p; \text{ret}\,(\bar{x}, \top)$

Proof. It suffices to prove $\Phi \vdash [x \leftarrow p; y \leftarrow \varphi]\,\psi \iff [x \leftarrow p; y \leftarrow \text{ret } \top]\,\psi$ for every monadic formula ψ in arbitrary context. Now the assumption implies $[x \leftarrow p]\,(\varphi \iff \text{ret } \top)$ by propositional reasoning. Thus we have

$$\begin{aligned}
\Phi \vdash &[x \leftarrow p; y \leftarrow \varphi]\,\psi \\
&\iff [x \leftarrow p]\,((\varphi \iff \top) \wedge [y \leftarrow \varphi]\,\psi) && \text{(seq}\Box\text{, prop. reasoning)} \\
&\iff [x \leftarrow p]\,((\varphi \iff \top) \wedge [y \leftarrow \text{ret } \top]\,\psi) && \text{(cong}\Box\text{, prop. reasoning)} \\
&\iff [x \leftarrow p; y \leftarrow \text{ret } \top]\,\psi && \text{(prop. reasoning, seq}\Box\text{)}
\end{aligned}$$

□

Proposition 6.8 By the results of [Mog91], it suffices to show the term variants of the monad laws in order to also ensure the coherence axioms for the strength. By Cor. 6.7, we need to show the equations using $\approx$.

- $\text{do } x \leftarrow p; \text{ret } x = p$: By (ctr□), $[x \leftarrow (\text{do } x \leftarrow p; \text{ret } x)]\,\psi$ is equivalent to $[x \leftarrow p; x \leftarrow \text{ret } x]\,\psi$. By (ret□), this in turn is equivalent to $[x \leftarrow p]\,\psi$.
- $\text{do } x \leftarrow \text{ret } y; p = p[y/x]$: By (ctr□), $[u \leftarrow (\text{do } x \leftarrow \text{ret } y; p)]\,\psi$ is equivalent to $[x \leftarrow \text{ret } y; u \leftarrow p]\,\psi$. By (ret□), this is equivalent to $[u \leftarrow p[y/x]]\,\psi$, noting that x is local to p.
- $\text{do } y \leftarrow (\text{do } x \leftarrow p; q); r = \text{do } x \leftarrow p; y \leftarrow q; r$: This follows easily with (ctr□).
- *Simplicity*: Let $(\text{do } \bar{x} \leftarrow \bar{p}; \text{ret}\,\varphi) = \text{do } \bar{x} \leftarrow \bar{p}; \text{ret } \top$. We have to show $(\text{do } \bar{x} \leftarrow \bar{p}; \text{ret}\,\langle \bar{x}, \varphi\rangle) = \text{do } \bar{x} \leftarrow \bar{p}; \text{ret}\,\langle \bar{x}, \top\rangle$, i.e. we have to prove the corresponding observational equivalence. For $\psi : P\Omega$, we have, as a special case of the assumption

$$\Phi \vdash [a \leftarrow (\text{do } \bar{x} \leftarrow \bar{p}; \text{ret } \varphi)]\,a \iff [a \leftarrow (\text{do } \bar{x} \leftarrow \bar{p}; \text{ret } \top)]\,a$$

and hence, by (ctr$\Box$) and (ret$\Box$),

$$\Phi \vdash [\bar{x} \leftarrow \bar{p}]\, \varphi \iff [\bar{x} \leftarrow \bar{p}]\ \mathrm{ret}\ \top.$$

The right hand side of the above equivalence is provable by (nec), so that $\Phi \vdash [\bar{x} \leftarrow \bar{p}]\, \varphi$. By Lemma 6.9, this finishes the proof of simplicity.

For showing that $\mathbb{T}_{\Sigma,\Phi}$ admits dynamic logic, we need to show that

$$\left[\bar{x} \leftarrow \bar{p}; a \leftarrow \Box p\right] x_i \Rightarrow a \text{ iff } \left[\bar{x} \leftarrow \bar{p}; a \leftarrow p\right] x_i \Rightarrow a$$

This follows from the following two equivalences:

$$\left[\bar{y} \leftarrow \bar{q}; a \leftarrow \Box p\right] (y_i \Rightarrow a) \text{ iff } \Phi \vdash [\bar{y} \leftarrow \bar{q}]\, y_i \Rightarrow \Box p \tag{6}$$

$$\left[\bar{y} \leftarrow \bar{q}; a \leftarrow p\right] (y_i \Rightarrow a) \text{ iff } \Phi \vdash [\bar{y} \leftarrow \bar{q}]\, y_i \Rightarrow \Box p \tag{7}$$

Proof of (6), "$\Rightarrow$": By Cor. 6.7, $\left[\bar{y} \leftarrow \bar{q}; a \leftarrow \Box p\right] (y_i \Rightarrow a)$ means that for all φ,

$$\Phi \vdash [\bar{y} \leftarrow \bar{q}; a \leftarrow \Box p; x \leftarrow \mathrm{ret}\, (y_i \Rightarrow a)]\, \varphi \iff [\bar{y} \leftarrow \bar{q}; a \leftarrow \Box p; x \leftarrow \mathrm{ret}\ \top]\, \varphi,$$

which by ret$\Box$ is just

$$\Phi \vdash [\bar{y} \leftarrow \bar{q}; a \leftarrow \Box p]\, \varphi[y_i \Rightarrow a/x] \iff [\bar{y} \leftarrow \bar{q}; a \leftarrow \Box p]\, \varphi[\top/x] \tag{8}$$

By (nec), $\vdash [\bar{y} \leftarrow \bar{q}; a \leftarrow \Box p]\, \top$, this is the right hand side of (8) for $\varphi \equiv \mathrm{ret}\ x$. We hence obtain $\Phi \vdash [\bar{y} \leftarrow \bar{q}; a \leftarrow \Box p]\ \mathrm{ret}\, (y_i \Rightarrow a)$. By ($\Box$), the latter is equivalent to $\Phi \vdash [\bar{y} \leftarrow \bar{q}; a \leftarrow p]\ \mathrm{ret}\, (y_i \Rightarrow a)$. Using (seq$\Box$), (cong$\Box$) and (taut), we obtain $\Phi \vdash [\bar{y} \leftarrow \bar{q}]\, (y_i \Rightarrow [a \leftarrow p]\, \mathrm{ret}\ a)$. With ($\Box$def), we obtain $\Phi \vdash [\bar{y} \leftarrow \bar{q}]\, (y_i \Rightarrow \Box p)$.

Proof of (6), "$\Leftarrow$": Conversely, assuming $\Phi \vdash [\bar{y} \leftarrow \bar{q}]\, (y_i \Rightarrow \Box p)$, by (cong$\Box$), (taut) and (nec) we get

$$\Phi \vdash [\bar{y} \leftarrow \bar{q}]\, (y_i \Rightarrow ([a \leftarrow \Box p]\, \varphi[y_i \Rightarrow a/x] \iff [a \leftarrow \Box p]\, \varphi[\top/x]).$$

Also, by (cong$\Box$), (taut) and (nec),

$$\Phi \vdash [\bar{y} \leftarrow \bar{q}]\, (\neg y_i \Rightarrow ([a \leftarrow \Box p]\, \varphi[y_i \Rightarrow a/x] \iff [a \leftarrow \Box p]\, \varphi[\top/x]).$$

Altogether, with (taut) and (K), we arrive at (8).

Proof of (7), "$\Rightarrow$": $\left[a \leftarrow p\right] a$ means (using (ret$\Box$) again) that for all φ,

$$[\bar{y} \leftarrow \bar{q}; a \leftarrow p]\, \varphi[y_i \Rightarrow a/x] \iff [\bar{y} \leftarrow \bar{q}; a \leftarrow p]\, \varphi[\top/x]$$

We obtain $\Phi \vdash [\bar{y} \leftarrow \bar{q}]\, (y_i \Rightarrow \Box p)$ in a similar way as for (6), "$\Rightarrow$", omitting the application of ($\Box$).

Proof of (7), "$\Leftarrow$": Conversely, assume $\Phi \vdash [\bar{y} \leftarrow \bar{q}]\, (y_i \Rightarrow \Box p)$, i.e. by ($\Box$def), $\Phi \vdash [\bar{y} \leftarrow \bar{q}]\, (y_i \Rightarrow [a \leftarrow p]\, \mathrm{ret}\ a)$. The rest of the argument is similar as for (6), "$\Leftarrow$".

□

Lemma 6.10 (Truth Lemma). $\mathbb{T}_{\Sigma,\Phi} \models \varphi$ iff $\Phi \vdash \varphi$.

Proof.

$\mathbb{T}_{\Sigma,\Phi} \models \boxed{\mathrm{G}}\varphi$	iff
$[\![\varphi]\!] = [\![\top]\!]$	iff (by Cor. 6.6)
$\varphi \sim \top$	iff
for all MDL formulae ψ, $\Phi \vdash \psi[\varphi/x] \iff \psi[\top/x]$	iff (by (cong))
$\Phi \vdash \varphi \iff \top$	iff (by (taut), (mp))
$\Phi \vdash \varphi$.	

□

The main result now follows straightforwardly:

Theorem 6.11 (Completeness of MDL). If $\Phi \models \psi$, then $\Phi \vdash \psi$.

Proof. Let $\Phi \models \psi$. By the Truth Lemma, $\mathbb{T}_{\Sigma,\Phi} \models \Phi$, hence $\mathbb{T}_{\Sigma,\Phi} \models \psi$ (noting that by Prop. 6.8 $\mathbb{T}$ is a simple monad admitting dynamic logic). Again by the Truth Lemma, $\Phi \vdash \psi$. □

7. Conclusion and Future Work

Inspired by the logics of design by contract languages, we have introduced monad-based dynamic logic (MDL) as a means of handling with effects, purity and observational equivalence in an abstract way that avoids the development of a specific theory for each language. Our generic notion of observational purity captures the same intuition as Naumann's notion of observational purity [Nau07] when instantiated to his specific imperative language. However, besides state, store and non-termination, our logic can also handle effects like non-determinism, input/output, and many others. Moreover, our notions of observational and contextual equivalence (which coincide in the term model) come out of the logic in a more natural way, compared with the variety of notions in [Nau07]. The sound and complete calculus allows for reasoning about Hoare style and dynamic logic assertions as well as about observational purity in a uniform way.

MDL is similar to Pitts' evaluation logic [Pit91], but equipped with a different semantics which is induced directly by the underlying monad — rather than relying on an extra hyperdoctrine structure, which must in all likelihood be considered additional data (e.g. in the case of the state monad, the interpretation of monadic formulae as state predicates is explicitly imposed by the chosen hyperdoctrine). Existing completeness results for monad-based logics rely on a *global* semantics [Mog95, GSM06], which e.g. for the state monad means that a sequence of nested modalities leads to universal quantification over all states at each new nesting level - the (implicit) state is not passed across nesting levels. By contrast, our logic allows for reasoning in a truly local way about changes of state. So far, no completeness result has been proved for such logics.

A Hoare calculus has been built on top of MDL [SM03] and extended to a treatment of Java-style abrupt termination [SM04b, WSM05]. Practical applications of MDL include reasoning about Haskell and the imperative fragment of Java. Numerous examples and a coding in the theorem prover Isabelle can be found in [Wal05].

The completeness result now guarantees closedness of the deduction system (after the extension w.r.t. [SM04a]) and paves the way for the future development of decision procedures. Of course, one cannot expect a decision procedure for arbitrary equational theories. In this respect, our calculus (unlike similar calculi [Pit91, Mog95, GSM06]) has the advantage that it makes use only of a rather limited and efficiently decidable equational theory. In particular, equations between programs are avoided; instead, properties of programs are expressed in terms of their observable behaviour.

More practical experience is needed for evaluating how practical actual proofs of purity and observational purity are. Typically, such proofs will not be semantical like those of the claims made in Sect. 5, but rather rely on the types of basic operations that deliver pure results, and on suitable axioms. Indeed, our calculus can be instantiated with a variety of different effects by *axiomatising* these effects; we have provided one sample axiomatisation for the dynamic reference monad.

The generalisation to non-simple monads is an important open question. Another important extension will be the treatment of control structures such as while loops (which currently happens at the meta-level) as well as of datatypes in the calculus proper.

Acknowledgements This work has been supported by the German Federal Ministry of Education and Research (Project 01 IW 07002 FormalSafe) and by the DFG (project HASCASL KR 1191/7-2). The authors wish to thank the anonymous referees for useful comments, and Erwin R. Catesbeiana for pointing out various pitfalls.

References

[BBdP98] P. N. Benton, Gavin M. Bierman, and Valeria de Paiva. Computational types from a logical perspective. *J. Funct. Program*, 8(2):177–193, 1998.

[Bec01] Bernhard Beckert. A dynamic logic for the formal verification of Java Card programs. In I. Attali and T. Jensen, editors, *Java on Smart Cards: Programming and Security. Revised Papers, Java Card 2000, International Workshop, Cannes, France*, volume 2041 of *LNCS*, pages 6–24. Springer-Verlag, 2001.

[BK03] Daniel Bonniot and Bryn Keller. The Nice user's manual. http://nice.sourceforge.net, 2003.

[Boe85] Hans-Juergen Boehm. Side effects and aliasing can have simple axiomatic descriptions. *ACM Trans. Program. Lang. Syst*, 7:637–655, 1985.

[Bri02] Walter Bright. The D programming language. *Dr. Dobb's Journal of Software Tools*, 27(2):36–40, February 2002.

[CK05] David R. Cok and Joseph R. Kiniry. ESC/Java2: Uniting ESC/Java and JML: Progress and issues in building and using ESC/Java2, including a case study involving the use of the tool to verify portions of an Internet voting tally system. In Gilles Barthe, Lilian Burdy, Marieke Huisman, Jean-Louis Lanet, and Traian Muntean, editors, *Construction and Analysis of Safe, Secure, and Interoperable Smart devices (CASSIS 2004)*, volume 3362 of *Lecture Notes in Computer Science*, pages 108–128. Springer-Verlag, 2005.

[FF02] Robert Bruce Findler and Matthias Felleisen. Contracts for higher-order functions. In *ICFP*, pages 48–59, 2002.

[Füh02] C. Führmann. Varieties of effects. In *Foundations of Software Science and Computation Structures*, volume 2303 of *LNCS*, pages 144–158. Springer, 2002.

[GSM06] Sergey Goncharov, Lutz Schröder, and Till Mossakowski. Completeness of global evaluation logic. In *Mathematical Foundations of Computer Science, MFCS 06*, volume 4162 of *LNCS*, pages 447–458. Springer, 2006.

[Hin64] J. R. Hindley. *The Church-Rosser Property and a Result in Combinatory Logic*. PhD thesis, University of Newcastle-upon-Tyne, 1964.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *CACM*, 12, 1969.

[Hui01] M. Huisman. *Java program verification in higher order logic with PVS and Isabelle*. PhD thesis, University of Nijmegen, 2001.

[JP03] B. Jacobs and E. Poll. Coalgebras and Monads in the Semantics of Java. *Theoret. Comput. Sci.*, 291:329–349, 2003.

[LBR06] Gary T. Leavens, Albert L. Baker, and Clyde Ruby. Preliminary design of JML: A behavioral interface specification language for Java. *ACM SIGSOFT Software Engineering Notes*, 31(3):1–38, 2006.

[Mac97] S. Mac Lane. *Categories for the Working Mathematician*. Springer, 1997.

[Mey92] Bertrand Meyer. *Eiffel: The Language*. Prentice-Hall, 1992.

[Mog91] E. Moggi. Notions of computation and monads. *Inform. and Comput.*, 93:55–92, 1991.

[Mog95] E. Moggi. A semantics for evaluation logic. *Fund. Inform.*, 22:117–152, 1995.

[MSG08] Till Mossakowski, Lutz Schröder, and Sergey Goncharov. A generic complete dynamic logic for reasoning about purity and effects. In J. Fiadeiro and P. Inverardi, editors, *Fundamental Approaches to Software Engineering (FASE 2008)*, volume 4961 of *Lecture Notes in Computer Science*, pages 199–214. Springer, 2008.

[Nau07] David A. Naumann. Observational purity and encapsulation. *Theoret. Comput. Sci*, 376:205–224, 2007.

[Nip02] Tobias Nipkow. Hoare logics in Isabelle/HOL. In H. Schwichtenberg and R. Steinbrüggen, editors, *Proof and System-Reliability*, pages 341–367. Kluwer, 2002.

[Omo91] Stephen M. Omohundro. The Sather language. Technical report, International Computer Science Institute, Berkeley, 1991.

[Pit91] A. Pitts. Evaluation logic. In *Higher Order Workshop*, Workshops in Computing, pages 162–189. Springer, 1991.

[PJ03] S. Peyton-Jones, editor. *Haskell 98 Language and Libraries — The Revised Report*. Cambridge, 2003. also: J. Funct. Programming **13** (2003).

[Pra76] V. Pratt. Semantical considerations on Floyd-Hoare logic. In *Foundations of Conputer Science, FOCS 76*, pages 109–121. IEEE, 1976.

[SC02] Benoit Sonntag and Dominique Colnet. Lisaac: the power of simplicity at work for operating system. In *Technology of Object-Oriented Languages and Systems, TOOLS Pacific 02*, volume 10 of *CRPIT*, pages 45–52. ACS, 2002.

[SM03] L. Schröder and T. Mossakowski. Monad-independent Hoare logic in HASCASL. In *Fundamental Aspects of Software Engineering, FASE 03*, volume 2621 of *LNCS*, pages 261–277, 2003.

[SM04a] L. Schröder and T. Mossakowski. Monad-independent dynamic logic in HASCASL. *J. Logic Comput.*, 14:571–619, 2004.

[SM04b] Lutz Schröder and Till Mossakowski. Generic exception handling and the Java monad. In *Algebraic Methodology and Software Technology, AMAST 04*, volume 3116 of *LNCS*, pages 443–459. Springer, 2004.

[Ste04] Kurt Stenzel. A formally verified calculus for full java card. In Charles Rattray, Savi Maharaj, and Carron Shankland, editors, *AMAST*, volume 3116 of *Lecture Notes in Computer Science*, pages 491–505. Springer, 2004.

[Thi97] H. Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, 1997.

[vdBJ01] Joachim van den Berg and Bart Jacobs. The LOOP compiler for java and JML. In Tiziana Margaria and Wang Yi, editors, *TACAS*, volume 2031 of *Lecture Notes in Computer Science*, pages 299–312. Springer, 2001.

[vO01] David von Oheimb. Hoare logic for java in isabelle/HOL. *Concurrency and Computation: Practice and Experience*, 13:1173–1214, 2001.

[Wal05] Dennis Walter. Monadic dynamic logic: Application and implementation. Master's thesis, University of Bremen, 2005. Available at http://www.cs.chalmers.se/∼denniswa.

[WSM05] Dennis Walter, Lutz Schröder, and Till Mossakowski. Parametrized exceptions. In *Algebra and Coalgebra in Computer Science, CALCO 05*, volume 3629 of *LNCS*, pages 424–438. Springer, 2005.