Logical Methods in Computer Science
Vol. 11(1:5)2015, pp. 1–50
www.lmcs-online.org

Submitted Apr. 17, 2013
Published Mar. 6, 2015

PROGRAM LOGICS FOR HOMOGENEOUS GENERATIVE RUN-TIME META-PROGRAMMING

MARTIN BERGER [a] AND LAURENCE TRATT [b]

[a] Department of Informatics, University of Sussex, Falmer, Brighton BN1 9QJ, United Kingdom
e-mail address: M.F.Berger@sussex.ac.uk

[b] Software Development Team, Department of Informatics, King's College London, Strand, London WC2R 2LS, United Kingdom
e-mail address: laurie@tratt.net

Abstract. This paper provides the first program logic for homogeneous generative run-time meta-programming—using a variant of MiniML$_e^{\Box}$ by Davies and Pfenning as its underlying meta-programming language. We show the applicability of our approach by reasoning about example meta-programs from the literature. We also demonstrate that our logics are relatively complete in the sense of Cook, enable the inductive derivation of characteristic formulae, and exactly capture the observational properties induced by the operational semantics.

Dedicated to the memory of Kohei Honda.

1. Introduction

Meta-programming (MP) is the generation or manipulation of programs, or parts of programs, by other programs, i.e. in an algorithmic way. Many programming languages, going back at least as far as Lisp, have explicit MP features. These can be classified in various ways such as: generative (program creation), intensional (program analysis), compile-time (happening while programs are compiled), run-time (taking place as part of program execution), heterogeneous (where the system generating or analysing the program is different from the system being generated or analysed), homogeneous (where the systems involved are the same), and lexical (working on simple strings) or syntactical (working on abstract syntax trees). Compilers use MP to compile programs; web system languages such as PHP use MP to produce web pages containing Javascript; Javascript (in common with some other languages) performs MP by dynamically generating strings and then executing them using its `eval` function. In short, MP is a mainstream activity.

2012 ACM CCS: [**Software and its engineering**]: Software organization and properties—Software functional properties—Formal methods—Software verification; Software notations and tools—General programming languages—Language features; [**Theory of computation**]: Semantics and reasoning—Program semantics—Axiomatic semantics; Logic.

Key words and phrases: Program Logic, Specification, Verification, Meta-Programming, Types, Observational Completeness, Descriptive Completeness, Relative Completeness, Characteristic Formula.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-11(1:5)2015

One of the most important types of MP is homogeneous generative meta-programming. The first language to support this was Lisp with its S-expression based macros; Scheme's macros improve upon Lisp's by being fully hygienic, but are conceptually similar. Perhaps unfortunately, the power of Lisp-based macros was long seen to rest largely on Lisp's minimalistic syntax and subsequent work on HGMP struggled to transfer Lisp's power to languages with modern, large syntaxes. MetaML [31] was the first syntactically rich language capable of homogeneous generative meta-programming in a manner convenient enough to rival Lisp's, albeit it could only generate code at run-time rather than at compile-time. Since then, MetaOCaml has taken MetaML's [32] ideas further; while Template Haskell [30] and Converge [33] have developed compile-time generative meta-programming. These languages have clearly shown that a wide variety of modern programming languages can house homogeneous generative meta-programming, and that this allows powerful, safe programming of a type previously impractical or impossible.

This paper develops program logics for generative MP. An important question is which flavour? From this paper's perspective, the most obvious division is whether generative MP occurs solely at run-time (à la MetaML) or also at compile-time (à la Lisp). Since the latter case includes the former, a reasonable first step is to tackle run-time generative MP. In other words, this paper develops logics for languages in the MetaML vein, and we hope it provides a basis for extending that work to languages that can support compile-time generative MP. So that we are clear about what precisely form of MP we are tackling, we use the term homogeneous generative run-time meta-programming (HGRTMP). We appreciate that HGRTMP is not a snappy acronym, but, in the process of developing this work, we have found that MP's many flavours are too easily confused with one another.

Meta-programming & verification. There are currently no logics for MP capable languages, HGRTMP or otherwise. We believe that the following reasons might be partly responsible:

- Reasoning about MP languages is a strict superset of reasoning about non-MP languages. Developing logics for non-MP programming languages is a hard problem on its own, and satisfactory solutions for reasoning about programs with higher-order functions, state, pointers, continuations, or concurrency have only recently been discovered [3, 28, 38].
- MP correctness can sometimes be side-stepped by ignoring the MP itself and looking only at its output. Compilation is an example where the MP machinery is more complex than the program's output. However, verifying only the output of MP is limiting, because knowledge gathered from the program's input, and during the MP process, is lost.
- Static typing of generative MP still lacks a satisfactory solution. Consequently, most generative MP languages are at least partly dynamically typed (including MetaOCaml which checks for certain forms of code extrusion at run-time); Template Haskell on the other hand intertwines code generation with type-checking in complicated ways. Logics for such languages are not well understood in the absence of other MP features; moreover, many MP languages have additional features such as capturing substitution, pattern matching of code, and splicing of types, which are largely unexplored theoretically. Heterogeneous MP adds the complication of multi-language verification.

Contributions. The present paper is an extended version of [6] with proofs, simplifications, and other improvements. It is the first to investigate the use of program logics for the

specification and verification of HGRTMP[1]. The aim of the paper is to explore the axiomatic foundations of HGRTMP. The specific technical contributions of this paper are as follows:

- We provide the first program logic for an HGRTMP language—$\textsc{Pcf}_{\textsc{dp}}$, a variant of Davies and Pfenning's $\text{MiniML}_e^{\Box}$ [14], itself an extension of PCF [16]. The logic is for total correctness and smoothly generalises previous work on axiomatic semantics for the ML family of languages [3, 4, 18, 20, 21, 38]. A key feature of our logic is that $\textsc{Pcf}_{\textsc{dp}}$ programs in the PCF fragment (i.e. those that do not perform HGRTMP) can be reasoned about in the simpler PCF logic [18, 20]. Reasoning about HGRTMP therefore imposes no additional burden over reasoning about non-MP programs.
- We show that our logic is relatively complete in the sense of Cook [12].
- We demonstrate that the axiomatic semantics induced by our logic coincides precisely with the contextual semantics given by the reduction rules of $\textsc{Pcf}_{\textsc{dp}}$.
- We present an additional inference system for characteristic formulae which enables, for each program M, the inductive derivation of a pair A, B of formulae which describe completely M's behaviour (descriptive completeness [19]).

As the first work in this area, we do not pretend to tackle all the intricacies involved in a modern programming language. Instead, we work on a simplified language which allows us to focus on the fundamental issues.

2. $\textsc{Pcf}_{\textsc{dp}}$

This section introduces $\textsc{Pcf}_{\textsc{dp}}$[2], the MP language that is the basis of our study. $\textsc{Pcf}_{\textsc{dp}}$ is a variant of call-by-value PCF [16], extended with the HGRTMP features of Davies and Pfenning's Mini-$\text{ML}_e^{\Box}$ [14, Section 3]. Mini-$\text{ML}_e^{\Box}$ was the first typed MP language to provide facilities for executing generated code. Typing the execution of generated code is a difficult problem. Mini-$\text{ML}_e^{\Box}$ achieves type-safety with two substantial restrictions on meta-programming:

- Only code without free variables can be run (i.e. generated code which is not closed cannot be run).
- Variables free in code cannot be λ-abstracted or be recursion variables.

Mini-$\text{ML}_e^{\Box}$ was one of the first MP languages with a Curry-Howard correspondence, although the present paper does not investigate the connection between our program logic and the Curry-Howard correspondence. $\textsc{Pcf}_{\textsc{dp}}$ is essentially Mini-$\text{ML}_e^{\Box}$, but with a slightly different form of recursion that can be given a moderately simpler logical characterisation.

$\textsc{Pcf}_{\textsc{dp}}$ is an ideal vehicle for our investigation for two reasons. First, $\textsc{Pcf}_{\textsc{dp}}$ is designed to be a simple language, yet it has all the key features of HGRTMP; $\textsc{Pcf}_{\textsc{dp}}$'s operational semantics is substantially simpler than that of MetaML [31] and its descendants, for example. Second, $\textsc{Pcf}_{\textsc{dp}}$ is built atop PCF, a well-understood idealised programming language with existing program logics [18, 20]. This allows us to compare reasoning in the PCF-fragment with reasoning in full $\textsc{Pcf}_{\textsc{dp}}$.

[1] Since the original publication [6], Charlton has developed a logic for a simple, first-order HGRTMP language with a Javascript-like `eval` feature [9].

[2] The name is our tip of the hat to Davies and Pfenning's work.

2.1. **Language basics.** PCF is a traditional λ-calculus and we assume readers are familiar with such languages. PCF_{DP} extends PCF with two new constructs, $\langle M \rangle$ and $\texttt{let}\ \langle x \rangle = M\ \texttt{in}\ N$, as well as a new type $\langle \alpha \rangle$.

Quasi-quotes $\langle M \rangle$ were invented in the context of logic [35, 36], and introduced to programming languages in Lisp [2]. A quasi-quote $\langle M \rangle$ represents the code of M, and allows code fragments to be expressed using concrete syntax. If M has type α, then $\langle M \rangle$ is typed $\langle \alpha \rangle$. For example, $\langle 1 + 7 \rangle$ is the code of the program $1 + 7$, and $\langle 1 + 7 \rangle$ has type $\langle \mathsf{Int} \rangle$. $\langle M \rangle$ is a value for all M and hence $\langle 1 + 7 \rangle$ does not reduce to $\langle 8 \rangle$. Note that PCF_{DP}'s quasi-quotes are subtly different from the abstract syntax trees (ASTs) used in some languages (e.g. Template Haskell and Converge). In such languages, quasi-quotes are a 'front end' for ASTs, but ASTs can be manually instantiated to represent any program. In PCF_{DP}, in contrast, quasi-quotes are the only term constructors for meta-programs. This makes our formalism more tractable but prevents some seemingly reasonable meta-programs from being expressed (e.g. those that generate an `if` with an arbitrary number of `else if` clauses).

The *unquote* construct $\texttt{let}\ \langle x \rangle = M\ \texttt{in}\ N$ extracts code from a quasi-quote. It evaluates M to code $\langle M' \rangle$, extracts M' from the quasi-quote, names it x and makes M' available in N without reducing M'. The fact that M' is not evaluated after extraction from a quasi-quote is the essence of generative MP as it enables the construction of code *other than values* under λ-abstractions.

PCF_{DP}'s unquote unifies MetaML's separate notions of splicing (inserting quasi-quoted code into another quasi-quoted fragment) and executing quasi-quotes. The following example shows it being used for splicing:

$$\texttt{let}\ \langle x \rangle = (\lambda z.z)\langle 1 + 7 \rangle\ \texttt{in}\ \langle \lambda n.x^n \rangle$$

This first reduces the application to $\langle 1 + 7 \rangle$, then extracts the code from $\langle 1 + 7 \rangle$, names it x and makes it available unevaluated to the code $\langle \lambda n.x^n \rangle$:

$$\begin{array}{lcl} \texttt{let}\ \langle x \rangle = (\lambda z.z)\langle 1 + 7 \rangle\ \texttt{in}\ \langle \lambda n.x^n \rangle & \rightarrow & \texttt{let}\ \langle x \rangle = \langle 1 + 7 \rangle\ \texttt{in}\ \langle \lambda n.x^n \rangle \\ & \rightarrow & \langle \lambda n.x^n \rangle [1 + 7/x] \\ & = & \langle \lambda n.(1 + 7)^n \rangle \end{array}$$

The program $\texttt{let}\ \langle x \rangle = \langle N \rangle\ \texttt{in}\ x$ is an example of unquote executing a program, since N will be extracted from the quasi-quote and bound to x, which is then run.

2.2. **Syntax and types.** We now formalise PCF_{DP}'s syntax and semantics, assuming a set of variables, ranged over by $x, y, g, u, m, ...$ (for more details see [14, 16]).

$$\begin{array}{lcl} \alpha & ::= & \mathsf{Unit} \mid \mathsf{Bool} \mid \mathsf{Int} \mid \alpha \rightarrow \beta \mid \langle \alpha \rangle \\ V & ::= & \mathtt{c} \mid x \mid \lambda x^{\alpha}.M \mid \mu g^{\alpha \rightarrow \beta}.\lambda x^{\alpha}.M \mid \langle M \rangle \\ M & ::= & V \mid \mathtt{op}(\tilde{M}) \mid MN \mid \texttt{if}\ M\ \texttt{then}\ N\ \texttt{else}\ N' \mid \texttt{let}\ \langle x \rangle = M\ \texttt{in}\ N \end{array}$$

Here, α ranges over *types*, V over *values*, and M over *programs*. Constants $\mathtt{c}$ range over the integers $0, 1, 2, -1, ...$, booleans $\mathtt{t}, \mathtt{f}$, and $()$ of type Unit, $\mathtt{op}$ ranges over the usual first-order operators like addition, multiplication, equality, conjunction, negation, comparison, etc., with the restriction that equality is *not* defined on expressions of function type or of type $\langle \alpha \rangle$. The abbreviation $\tilde{M}$ means a (possibly empty) tuple $(M_1, ..., M_n)$. The recursion operator is $\mu g.\lambda x.M$. The *free variables* $\mathsf{fv}(M)$ of M are defined as usual with two new

$$\frac{(x,\alpha)\in\Gamma\cup\Delta}{\Gamma;\Delta\vdash x:\alpha}\qquad\frac{\Gamma,x:\alpha;\Delta\vdash M:\beta}{\Gamma;\Delta\vdash\lambda x^{\alpha}.M:\alpha\to\beta}\qquad\frac{\Gamma;\Delta\vdash M:\alpha\to\beta\quad\Gamma;\Delta\vdash N:\alpha}{\Gamma;\Delta\vdash MN:\beta}$$

$$\frac{\Gamma,f:(\alpha\to\beta);\Delta\vdash\lambda x^{\alpha}.M:\alpha\to\beta}{\Gamma;\Delta\vdash\mu f^{\alpha\to\beta}.\lambda x^{\alpha}.M:\alpha\to\beta}\qquad\frac{\Gamma;\Delta\vdash M:\mathsf{Bool}\quad\Gamma;\Delta\vdash N:\alpha\quad\Gamma;\Delta\vdash N':\alpha}{\Gamma;\Delta\vdash\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N':\alpha}$$

$$\frac{\Gamma;\Delta\vdash M:\mathsf{Int}\quad\Gamma;\Delta\vdash N:\mathsf{Int}}{\Gamma;\Delta\vdash M+N:\mathsf{Int}}\qquad\frac{\epsilon;\Delta\vdash M:\alpha}{\Gamma;\Delta\vdash\langle M\rangle:\langle\alpha\rangle}\qquad\frac{\Gamma;\Delta\vdash M:\langle\alpha\rangle\quad\Gamma;\Delta,x:\alpha\vdash N:\beta}{\Gamma;\Delta\vdash\mathtt{let}\ \langle x\rangle=M\ \mathtt{in}\ N:\beta}$$

Figure 1: Key typing rules for $\textsc{Pcf}_{\textsc{dp}}$.

clauses: $\mathsf{fv}(\langle M\rangle)\stackrel{def}{=}\mathsf{fv}(M)$ and $\mathsf{fv}(\mathtt{let}\ \langle x\rangle=M\ \mathtt{in}\ N)\stackrel{def}{=}\mathsf{fv}(M)\cup(\mathsf{fv}(N)\setminus\{x\})$. We write $\lambda().M$ for $\lambda x^{\mathsf{Unit}}.M$ and $\mathtt{let}\ x=M\ \mathtt{in}\ N$ for $(\lambda x.N)M$, assuming that $x\notin\mathsf{fv}(M)$ in both cases. We assume Barendregt's variable condition, and tacitly rename bound variables where necessary.

The *reduction relation* $\to$ is unchanged from PCF for the PCF-fragment of $\textsc{Pcf}_{\textsc{dp}}$, and adapted to $\textsc{Pcf}_{\textsc{dp}}$ as follows. First we define *reduction contexts*, by extending those for PCF with a construct for unquoting.

$$\begin{array}{rcl}\mathcal{E}[\cdot] & ::= & [.]\ \mid\ \mathcal{E}[\cdot]M\ \mid\ V\mathcal{E}[\cdot]\ \mid\ \mathtt{op}(\tilde{V}\mathcal{E}[\cdot]\tilde{M})\ \mid\ \mathtt{if}\ \mathcal{E}[\cdot]\ \mathtt{then}\ M\ \mathtt{else}\ N\\ & \mid & \mathtt{let}\ \langle x\rangle=\mathcal{E}[\cdot]\ \mathtt{in}\ M\end{array}$$

Now $\to$ is defined on *closed* programs by the clauses given next:

- $(\lambda x.M)V\to M[V/x]$.
- $(\mu g.\lambda x.M)V\to M[\mu g.\lambda x.M/g][V/x]$.
- $\mathtt{if}\ \mathtt{t}\ \mathtt{then}\ M\ \mathtt{else}\ N\to M$.
- $\mathtt{let}\ \langle x\rangle=\langle M\rangle\ \mathtt{in}\ N\to N[M/x]$.
- $M\to N$ implies $\mathcal{E}[M]\to\mathcal{E}[N]$.

We write $\twoheadrightarrow$ for $\to^*$. $M\Downarrow V$ means that $M\twoheadrightarrow V$ for some value V. We write $M\Downarrow$ if $M\Downarrow V$ for some appropriate V, and $M\Uparrow$ if not $M\Downarrow$.

A *typing environment* $(\Gamma,\Delta,...)$ is a finite map $x_1:\alpha_1,...,x_k:\alpha_k$ from variables to types. The *domain* $\mathsf{dom}(\Gamma)$ of Γ is the set $\{x_1,...,x_n\}$, assuming that Γ is $x_1:\alpha_1,...,x_n:\alpha_n$. We write ϵ for the empty environment. The *typing judgement* is written $\Gamma;\Delta\vdash M:\alpha$ where we assume that $\mathsf{dom}(\Gamma)\cap\mathsf{dom}(\Delta)=\emptyset$. We write $\vdash M:\alpha$ for $\epsilon;\epsilon\vdash M:\alpha$. We say a program M is *closed* if $\vdash M:\alpha$. We call Δ a *modal context* in $\Gamma;\Delta\vdash M:\alpha$. We say a variable x is *modal* or *modally typed* in $\Gamma;\Delta\vdash M:\alpha$ if $x\in\mathsf{dom}(\Delta)$. Modal variables represent code inside other code, and code to be run. The key type-checking rules are given in Figure 1. Typing for constants and first-order operations is standard.

Noteworthy features of the typing system are that modal variables cannot be λ- or μ-abstracted, that all free variables in quasi-quotes must be modal, and that modal variables can only be generated by unquotes. [14] gives detailed explanations of this typing system and its relationship to modal logics.

Contextual congruence. By $\lesssim_{\Gamma;\Delta;\alpha}$ (often abbreviated to just $\lesssim$) we denote the usual *typed contextual precongruence*: if $\Gamma;\Delta\vdash M_i:\alpha$ for $i=1,2$ then: $M_1\lesssim_{\Gamma;\Delta;\alpha}M_2$ iff for all

closing context $C[\cdot]$ such that $\vdash C[M_i] : \mathsf{Unit}$ $(i = 1, 2)$ we have

$$C[M_1] \Downarrow \text{ implies } C[M_2] \Downarrow .$$

We write $\simeq$ for $\lesssim \cap \lesssim^{-1}$ and call $\simeq$ *contextual congruence*. Other forms of congruence are possible, but we will use $\simeq$ in the rest of this paper. Our choice means that code can only be observed contextually, i.e. by running it in a context. Hence for example $\langle M \rangle$ and $\langle \lambda x.Mx \rangle$ are contextually indistinguishable if $x \notin \mathsf{fv}(M)$, as are $\langle 1 + 2 \rangle$ and $\langle 3 \rangle$. This facilitates a smooth integration of the logics for PCF_{DP} with the logics for PCF.[3]

2.3. **Basic lemmas.** We now present a collection of simple facts that we use later.

Proposition 2.1.

(1) *If M is closed and $M \simeq V$ then $M \Downarrow W$ for some value W with $V \simeq W$.*
(2) *If N is closed and $\langle M \rangle \simeq N$ then $N \Downarrow \langle M' \rangle$ and $M \simeq M'$ for some M'.*
(3) *$M[\mu g.M/g] \simeq \mu g.M$.*
(4) *Let M_1 and M_2 be closed. If $\Gamma; \Delta \vdash M_i : \alpha$ for $i = 1, 2$ then $M_1 \Uparrow$ and $M_2 \Uparrow$ implies $M_1 \simeq M_2$*
(5) *If $M \lesssim N$ then $L[M/x] \lesssim L[N/x]$.*
(6) *$\twoheadrightarrow \subseteq \simeq \subseteq \lesssim$.*
(7) *If $M \twoheadrightarrow N$ and $L \lesssim N$ then also $L \lesssim M$.*
(8) *$M \lesssim N$ if and only if $\langle M \rangle \lesssim \langle N \rangle$.*
(9) *If $MN \lesssim M'N$ for all N then $M \lesssim M'$.*
(10) *If $M[N/x] \Downarrow$ but $N \Uparrow$ then for all closed N': $M[N'/x] \Downarrow$.*
(11) *If $M[N/x] \Downarrow$ and $N \lesssim N'$ (with N' closed) then also $M[N'/x] \Downarrow$.*
(12) *If for all n we have $W_n \lesssim V$, then also $\mu g.\lambda x.M \lesssim V$ where $W_0 \stackrel{\text{def}}{=} \Omega$ and $W_{n+1} \stackrel{\text{def}}{=} \lambda x.M[W_n/g]$, cf.* [27].

Proof. All are straightforward yet laborious when carried out in detail, and can be tackled with standard techniques of operational semantics [16, 27]. As just one example, take ((12)): if for all n we have $W_n \lesssim V$, but at the same time $\mu g.\lambda x.M \not\lesssim V$, we could find a closing context $C[\cdot]$ such that $C[\mu g.\lambda x.M] \Downarrow$ but $C[V] \Uparrow$. But $C[\mu g.\lambda x.M] \Downarrow$ means that the computation towards a value is of finite length, hence only a finite number of recursive calls were made, so some n must exist, such that $C[W_n] \Downarrow$. This in turn means $C[V] \Downarrow$ by our assumptions, contradicting $C[V] \Uparrow$. □

2.4. **Some example programs.** Lifting is an important construct in generative MP, taking a run-time value and converting it into its quasi-quoted equivalent. For example $\langle 3 \rangle$ is the lifting of 3, and $\langle \lambda x^{\alpha}.x \rangle$ is the lifting of the identity function of type α.

We call a type α *basic* if it does not contain the function space constructor, i.e. if it has no sub-expressions of the form $\beta \to \beta'$. In PCF_{DP}, lifting takes an arbitrary value V of basic type α, and converts it to code $\langle V \rangle$ of type $\langle \alpha \rangle$. Note that we cannot simply write $\lambda x.\langle x \rangle$

[3]Some MP languages are more discriminating, allowing, e.g. printing of code, which can distinguish α-equivalent programs. It is unclear how to design logics for such languages. A detailed discussion of program equalities in meta-programming languages can be found in [23].

because modal variables (i.e. variables free in code) cannot be λ-abstracted. For $\alpha = \mathsf{Int}$ the function is defined as follows:

$$\mathsf{lift}_{\mathsf{Int}} \stackrel{def}{=} \mu g.\lambda n^{\mathsf{Int}}.\mathtt{if}\ n \leq 0\ \mathtt{then}\ \langle 0\rangle\ \mathtt{else}\ \mathtt{let}\ \langle x\rangle = g(n-1)\ \mathtt{in}\ \langle x+1\rangle.$$

Note that $\mathsf{lift}_{\mathsf{Int}}$ works properly only on non-negative integers. Note also that $\mathsf{lift}_{\mathsf{Int}}\ 3$ evaluates to $\langle 0+1+1+1\rangle$, not $\langle 3\rangle$. In more expressive meta-programming languages such as Converge the corresponding program would evaluate to $\langle 3\rangle$, which is more efficient, although $\langle 0+1+1+1\rangle$ and $\langle 3\rangle$ are observationally indistinguishable in $\textsc{Pcf}_{\textsc{dp}}$.

Lifting is easily extended to Unit and Bool, but not to function types, because of $\textsc{Pcf}_{\textsc{dp}}$'s inability to abstract modal variables. For basic types $\langle\alpha\rangle$ we can define lifting as follows.

$$\mathsf{lift}_{\langle\alpha\rangle} \stackrel{def}{=} \lambda x^{\langle\alpha\rangle}.\mathtt{let}\ \langle a\rangle = x\ \mathtt{in}\ \langle\langle a\rangle\rangle$$

We reason about $\mathsf{lift}_{\mathsf{Int}}$ in Section 4.

Another example is the function eval, a function of type $\langle\alpha\rangle \to \alpha$ for running code [14]. This function is essentially a wrapper around unquoting:

$$\mathsf{eval} \stackrel{def}{=} \lambda x^{\langle\alpha\rangle}.\mathtt{let}\ \langle y\rangle = x\ \mathtt{in}\ y.$$

Clearly, $\mathsf{eval}\ \langle 17+3\rangle$ converges to 20.

The last example in this section is the well-known power generative MP program which creates a function that raises a number to a given power [31]. Although somewhat contrived, this function shows how generative MP can be used for efficiency purposes: rather than using run-time recursion on every call, HGRTMP turns this into a fixed expression. In essence, if a program contains many applications $(\lambda n a.a^n)\ 3$, it makes sense to specialise such applications to $\lambda a.a \times a \times a$. A simple encoding of power in $\textsc{Pcf}_{\textsc{dp}}$ is the following:

$$\mathsf{power} \stackrel{def}{=} \mu p.\lambda n.\mathtt{if}\ n \leq 0\ \mathtt{then}\ \langle\lambda x.1\rangle\ \mathtt{else}\ \mathtt{let}\ \langle q\rangle = p(n-1)\ \mathtt{in}\ \langle\lambda x.x \times (q\ x)\rangle$$

This function has type $\vdash \mathsf{power} : \mathsf{Int} \to \langle\mathsf{Int} \to \mathsf{Int}\rangle$. This type says that power takes an integer and returns code. That code, when run, is a function from integers to integers. power can can be used as follows:

$$\mathsf{power}\ 2 \quad \twoheadrightarrow \quad \langle\lambda a.a \times ((\lambda b.b \times ((\lambda c.1)b))a)\rangle$$

3. A logic for total correctness

Our logic is a Hoare logic with pre- and post-conditions in the tradition of logics for ML-like languages [3, 4, 20, 21]. In this section we define its syntax and semantics.

3.1. Syntax and types.

Expressions, ranged over by $e, e', ...$ and *formulae* $A, B, ...$ of the logic are given by the grammar below, using the types and variables of $\textsc{Pcf}$:

$$\begin{array}{lll} e & ::= & \mathsf{c} \mid x \mid \mathsf{op}(\tilde{e}) \\ A & ::= & e = e' \mid \neg A \mid A \wedge B \mid \forall x^{\alpha}.A \mid u \bullet e = m\{A\} \mid u = \langle m\rangle\{A\} \end{array}$$

Our logical language is an extension of first-order logic with equality (and axioms for arithmetic e.g. Peano arithmetic or some set theory). Other quantifiers, logical constants like T, F and propositional connectives like $\supset$ (implication) are defined by de Morgan duality. Quantifiers range over values of appropriate type. Constants c and operations op are those of Section 2.2.

Our logic extends that of PCF [18, 19, 20] with a new *code evaluation* predicate $u = \langle m \rangle \{A\}$. It says that u, which must be of type $\langle \alpha \rangle$, denotes (up to contextual congruence) a quasi-quoted program $\langle M \rangle$, such that whenever M is unquoted and executed, it converges to a value; if that value is denoted by m then A makes a true statement about that value. We recall from [18, 19, 20] that $u \bullet e = m\{A\}$ says that (assuming u is of the function type) u denotes a function, which, when fed with the value denoted by e, terminates and yields another value. If we name this latter value m, A holds. The variable m is an *anchor* in both $u \bullet e = m\{A\}$ and $u = \langle m \rangle \{A\}$, bound within scope A. The *free variables* of e and A, written $\mathsf{fv}(e)$ and $\mathsf{fv}(A)$, respectively, are defined by the following clauses:

- $\mathsf{fv}(\mathsf{c}) \stackrel{def}{=} \emptyset$.
- $\mathsf{fv}(x) \stackrel{def}{=} \{x\}$.
- $\mathsf{fv}(\mathsf{op}(\tilde{e})) \stackrel{def}{=} \bigcup_i \mathsf{fv}(e_i)$.
- $\mathsf{fv}(e = e') \stackrel{def}{=} \mathsf{fv}(e) \cup \mathsf{fv}(e')$.
- $\mathsf{fv}(\neg A) \stackrel{def}{=} \mathsf{fv}(A)$.
- $\mathsf{fv}(A \wedge B) \stackrel{def}{=} \mathsf{fv}(A) \cup \mathsf{fv}(B)$.
- $\mathsf{fv}(\forall x^{\alpha}.A) \stackrel{def}{=} \mathsf{fv}(A) \setminus \{x\}$.
- $\mathsf{fv}(u \bullet e = m\{A\}) \stackrel{def}{=} (\mathsf{fv}(A) \setminus \{m\}) \cup \{u\} \cup \mathsf{fv}(e)$.
- $\mathsf{fv}(u = \langle m \rangle \{A\}) \stackrel{def}{=} (\mathsf{fv}(A) \setminus \{m\}) \cup \{u\}$.

In the presentation below we often use the following abbreviations and conventions:

- $A^{\text{-}x}$ means that $x \notin \mathsf{fv}(A)$.
- $x \Downarrow$ means $\exists y^{\alpha}.x = y$, assuming that x has type α, y is fresh and not modally typed. This abbreviation is interesting primarily when x is modally typed.
- $x \bullet e \Downarrow$ for $x \bullet e = m\{\mathsf{T}\}$.
- $m = \langle \cdot \rangle$ is a shorthand for $m = \langle x \rangle \{\mathsf{T}\}$ where x is fresh.
- $m = \langle e \rangle$ is short for $m = \langle x \rangle \{x = e\}$ where x is fresh, e.g. $m = \langle x \rangle$ is short for $m = \langle y \rangle \{x = y\}$. Note that e.g. x is free in $m = \langle x \rangle$, unlike in $m = \langle x \rangle \{A\}$.
- $m \bullet e = e'$ abbreviates $m \bullet e = x\{x = e'\}$ where x is fresh.
- We often omit typing annotations in expressions and formulae.

We have the usual capture avoiding substitutions of expressions for variables in expressions $e[e'/x]$ and formulae $A[e/x]$. They are defined by the following straightforward clauses.

- $y[e/x] \stackrel{def}{=} \begin{cases} y & x \neq y \\ e & x = y \end{cases}$.
- $\mathsf{c}[e/x] \stackrel{def}{=} \mathsf{c}$.
- $\mathsf{op}(\tilde{e})[e'/x] \stackrel{def}{=} \mathsf{op}(\tilde{e}[e'/x])$.
- $(e_1 = e_2)[e/x] \stackrel{def}{=} (e_1[e/x]) = (e_2[e/x])$.
- $(\neg A)[e/x] \stackrel{def}{=} \neg(A[e/x])$.
- $(A \wedge B)[e/x] \stackrel{def}{=} (A[e/x]) \wedge (B[e/x])$.
- $(\forall y.A)[e/x] \stackrel{def}{=} \forall y.(A[e/x])$ assuming $x \neq y$ and $y \notin \mathsf{fv}(e)$.
- $(u \bullet e = m\{A\})[e'/x] \stackrel{def}{=} u[e'/x] \bullet e[e'/x] = m\{A[e'/x]\}$ assuming $m \neq y$ and $m \notin \mathsf{fv}(e')$.
- $(u = \langle m \rangle \{A\})[e/x] \stackrel{def}{=} u[e/x] = \langle m \rangle \{A[e/x]\}$, assuming $m \neq x$ and $m \notin \mathsf{fv}(e)$.

$$\frac{(x,\alpha) \in \Gamma \cup \Delta}{\Gamma;\Delta \vdash x : \alpha} \quad \frac{\Gamma;\Delta \vdash u : \alpha \to \beta \quad \Gamma;\Delta \vdash e : \alpha \quad \Gamma, m : \beta;\Delta \vdash A}{\Gamma;\Delta \vdash u \bullet e = m\{A\}}$$

$$\frac{\Gamma;\Delta \vdash e : \alpha \quad \Gamma;\Delta \vdash e' : \alpha}{\Gamma;\Delta \vdash e = e'} \quad \frac{\Gamma;\Delta \vdash A \quad \Gamma;\Delta \vdash B}{\Gamma;\Delta \vdash A \wedge B} \quad \frac{\Gamma, x : \alpha;\Delta \vdash A}{\Gamma;\Delta \vdash \forall x^{\alpha}.A}$$

$$\frac{\Gamma;\Delta \vdash u : \langle\alpha\rangle \quad \Gamma;\Delta, m : \alpha \vdash A}{\Gamma;\Delta \vdash u = \langle m\rangle\{A\}} \quad \frac{\Gamma;\Delta \vdash A}{\Gamma;\Delta \vdash \neg A}$$

$$\frac{\Gamma;\Delta \vdash A \quad m \notin \mathsf{dom}(\Gamma) \cup \mathsf{dom}(\Delta) \quad \Gamma;\Delta \vdash M : \alpha \quad \Gamma, m : \alpha;\Delta \vdash B}{\Gamma;\Delta;\alpha \vdash \{A\}\ M :_m \{B\}}$$

Figure 2: Typing rules for expressions, formulae and judgements. Rules for constants and first-order operations omitted.

In the last two cases we assume that if $x = u$ then e' must be a variable.

The *judgements* for total correctness are of the form

$$\{A\}\ M :_m \{B\}.$$

The variable m is the *anchor* of the judgement, is a bound variable with scope B, and cannot be modal. The judgement is to be understood as follows: if A holds, then M terminates to a value (more precisely, the closure of M with arbitrary values meeting the precondition[4]), and if we denote that value by m, then B holds. In other words, our judgements are entirely conventional for total correctness program logics. If a variable x occurs freely in A or in B, but not in M, then x is an *auxiliary variable* of the judgement $\{A\}\ M :_m \{B\}$.

Typing expressions, formulae and judgements. Program logics are typed (although for simple programming languages, types can be implicit), and ours is no exception. We use the following typing judgements.

- For expressions, the typing judgement is $\Gamma;\Delta \vdash e : \alpha$.
- For formulae, the typing judgement is $\Gamma;\Delta \vdash A$.
- For judgements, the typing judgement is $\Gamma;\Delta;\alpha \vdash \{A\}\ M :_m \{B\}$.

The typing rules for all three judgements are given in Figure 2. Several points are worth noting.

- The anchor in $u = \langle m\rangle\{A\}$ is modal, while it is not modal in $u \bullet e = m\{A\}$ and in judgements.
- Normal quantification $\forall x.A$ quantifies only non-modal variables x.

From now on, we assume all occurring programs, expressions, formulae and judgements to be well-typed.

Examples of assertions & judgements. We continue with a few simple examples to help explain the use of our logic.

[4]In the remainder, we will sometimes be informal and say that a program M reduces or terminates, even when M may not be closed. What we mean is that the closure of M in the ambient model reduces or terminates.

– The assertion $m = \langle 3 \rangle$, which is short for $m = \langle x \rangle \{x = 3\}$ says that m denotes code which, when executed, will evaluate to 3. It can be used to make the following assertion on the program $\langle 1 + 2 \rangle$:

$$\{\mathsf{T}\}\ \langle 1 + 2 \rangle :_m \{m = \langle 3 \rangle\}.$$

– Let Ω_α be a non-terminating program of type α (we usually drop the type subscript). When we quasi-quote Ω, the judgement $\{\mathsf{T}\}\ \langle \Omega \rangle :_m \{\mathsf{T}\}$ says (*qua* precondition) that $\langle \Omega \rangle$ is a terminating program. Indeed, that is the strongest statement we can make about $\langle \Omega \rangle$ in a logic for total correctness, cf. Section 5.

– The assertion $\forall x^{\mathsf{Int}}.m \bullet x = y\{y = \langle x \rangle\}$ says that m denotes a terminating function which receives an integer and returns code which evaluates to that integer. Later, we use this assertion when reasoning about $\mathsf{lift}_{\mathsf{Int}}$ which has the following specification:

$$\{\mathsf{T}\}\ \mathsf{lift}_{\mathsf{Int}} :_u \{\forall n.n \geq 0 \supset u \bullet n = m\{m = \langle n \rangle\}\}$$

– The formula

$$A_u \stackrel{def}{=} \forall n^{\mathsf{Int}} \geq 0.\exists f^{\mathsf{Int} \to \mathsf{Int}}.(u \bullet n = \langle f \rangle \wedge \forall x^{\mathsf{Int}}.f \bullet x = x^n)$$

says that u denotes a function which receives an integer n as argument, to return code which when evaluated and fed another integer x, computes the power x^n, provided $n \geq 0$. We can then show that

$$\{\mathsf{T}\}\ \mathsf{power} :_u \{A_u\}$$

and

$$\{A_u\}\ u\ 7 :_r \{r = \langle f \rangle \{\forall x.f \bullet x = x^7\}\}.$$

– The formula $\forall x^{\langle \alpha \rangle} y^\alpha.(x = \langle y \rangle \supset u \bullet x = y)$ can be used to specify the evaluation function from Section 2:

$$\{\mathsf{T}\}\ \mathsf{eval} :_u \{\forall x^{\langle \alpha \rangle} y^\alpha.(x = \langle y \rangle \supset u \bullet x = y)\}.$$

3.2. **Models and the satisfaction relation.** This subsection formally presents the semantics of our logic. We begin with the notion of model. Our models are conventional, with the key difference from the models of PCF-logics [20] being that modal variables denote possibly non-terminating programs.

Let Γ, Δ be two contexts with disjoint domains (the idea is that Δ is modal while Γ is not). A *model* of type $\Gamma; \Delta$ is a pair (ξ, σ) such that:

– ξ is a map from $\mathsf{dom}(\Gamma)$ to closed *values* such that $\vdash \xi(x) : \Gamma(x)$;
– σ is a map from $\mathsf{dom}(\Delta)$ to closed *programs* $\vdash \sigma(x) : \Delta(x)$.

We use the following conventions in our subsequent presentation:

– We write $(\xi, \sigma)^{\Gamma;\Delta}$ to indicate that (ξ, σ) is a model of type $\Gamma; \Delta$.
– We write $\xi \cdot x : V$ for $\xi \cup \{(x, V)\}$ assuming that $x \notin \mathsf{dom}(\xi)$.
– Likewise for $\sigma \cdot x : M$.
– Let $\eta = (\xi, \sigma)$ be a model of type $\Gamma; \Delta$.
  – We write $\mathsf{dom}(\eta)$ for $\mathsf{dom}(\Gamma) \cup \mathsf{dom}(\Delta)$.
  – We write $\eta(x) = V$ to indicate that $x \in \mathsf{dom}(\eta)$, and $(x, V) \in (\xi \cup \sigma)$.

We can now present the semantics of expressions. Let $\Gamma; \Delta \vdash e : \alpha$ and assume that (ξ, σ) is a $\Gamma; \Delta$-model, we define $[\![e]\!]_{(\xi,\sigma)}$ by the following inductive clauses:

– $[\![\mathsf{c}]\!]_{(\xi,\sigma)} \stackrel{def}{=} \mathsf{c}$,
– $[\![\mathsf{op}(\tilde{e})]\!]_{(\xi,\sigma)} \stackrel{def}{=} \mathsf{op}([\![\tilde{e}]\!]_{(\xi,\sigma)})$,

– $[\![x]\!]_{(\xi,\sigma)} \stackrel{def}{=} (\xi \cup \sigma)(x)$.

The satisfaction relation for formulae has the following shape. Let $\Gamma;\Delta \vdash A$ and assume that (ξ,σ) is a $\Gamma;\Delta$-model.

– $(\xi,\sigma) \models e = e'$ iff $[\![e]\!]_{(\xi,\sigma)} \simeq [\![e']\!]_{(\xi,\sigma)}$.
– $(\xi,\sigma) \models \neg A$ iff $(\xi,\sigma) \not\models A$.
– $(\xi,\sigma) \models A \wedge B$ iff $(\xi,\sigma) \models A$ and $(\xi,\sigma) \models B$.
– $(\xi,\sigma) \models \forall x^{\alpha}.A$ iff for all closed values V of type α: $(\xi \cdot x : V,\sigma) \models A$.
– $(\xi,\sigma) \models u \bullet e = x\{A\}$ iff $([\![u]\!]_{(\xi,\sigma)}[\![e]\!]_{(\xi,\sigma)}) \Downarrow V$ and $(\xi \cdot x : V,\sigma) \models A$.
– $(\xi,\sigma) \models u = \langle m\rangle\{A\}$ iff $[\![u]\!]_{(\xi,\sigma)} \Downarrow \langle M\rangle$, $M \Downarrow V$ and $(\xi,\sigma \cdot m : V) \models A$.

The concept of upwards-closedness is important in the context of completeness and defined as follows. Let A be a formula typeable under $\Gamma, u : \alpha;\Delta \vdash A$. We say A is *upwards closed at* u if whenever $V \lesssim W$ then also

$$(\xi \cdot u : V,\sigma) \models A \qquad \text{implies} \qquad (\xi \cdot u : W,\sigma) \models A$$

for all suitable ξ and σ.

For defining the semantics of judgements, we need to explain what it means to apply a model $\eta \stackrel{def}{=} (\xi,\sigma)$ to a program M, written $M\eta$. We also refer to $M\eta$ as the *closure* of M with η. That is defined as usual, using the following inductive clauses, where we assume that free variables are not caught when a model is moved under a binder:

– $x\eta \stackrel{def}{=} \eta(x)$.
– $(MN)\eta \stackrel{def}{=} (M\eta)(N\eta)$.
– $\mathtt{c}\eta \stackrel{def}{=} \mathtt{c}$.
– $(\lambda x.M)\eta \stackrel{def}{=} \lambda x.(M\eta)$.
– $(\mu g.\lambda x.M)\eta \stackrel{def}{=} \mu g.\lambda x.(M\eta)$.
– $\langle M\rangle\eta \stackrel{def}{=} \langle M\eta\rangle$.
– $(\mathtt{op}(\tilde{M}))\eta \stackrel{def}{=} \mathtt{op}(\tilde{M\eta})$.
– $(\mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N')\eta \stackrel{def}{=} \mathtt{if}\ M\eta\ \mathtt{then}\ N\eta\ \mathtt{else}\ N'\eta$.
– $(\mathtt{let}\ \langle x\rangle = M\ \mathtt{in}\ N)\eta \stackrel{def}{=} \mathtt{let}\ \langle x\rangle = M\eta\ \mathtt{in}\ N\eta$.

We record the following simple fact for subsequent use.

Observation 3.1. Let $\mathcal{R}$ be one of $\lesssim, \cong$, then: $M\ \mathcal{R}\ N$ if and only iff for all appropriately typed models η: $M\eta\ \mathcal{R}\ N\eta$.

The *satisfaction relation* $\models \{A\}\ M :_m \{B\}$ is given next. Let $\Gamma;\Delta;\alpha \vdash \{A\}\ M :_m \{B\}$. Then $\models \{A\}\ M :_m \{B\}$ holds if and only if for all models $(\xi,\sigma)^{\Gamma;\Delta}$:

$$(\xi,\sigma) \models A \qquad \text{implies} \qquad \exists V.(M(\xi,\sigma) \Downarrow V \text{ and } (\xi \cdot m : V,\sigma) \models B)).$$

This is the standard notion for total correctness, adapted to the present logic.

A note on models. The reader might wonder why our notion of model uses *values* (which always terminate) as denotations for non-modal variables, but general programs (which may not terminate) for modal variables. The answer is a combination of two factors:

– Our logic is part of a tradition of constructing Hoare logics, where models provide denotations for the free variables of the program that a judgement is about. Moreover, the type of the denotation should be the same as the type of the corresponding free

variable. This simple model-building heuristic has proven to be robust for a wide variety of programming languages [5], and we decided to build our logic for $\textsc{Pcf}_{\textsc{dp}}$ in the same way.

– Although our logic is for total correctness, we can still make assertions about non-terminating programs, and programs that contain non-terminating sub-programs, for example:
  – $\{\mathsf{F}\}\ \Omega :_u \{A\}$.
  – $\{\mathsf{T}\}\ \lambda x.\Omega :_u \{\mathsf{T}\}$.
  – $\{\mathsf{T}\}\ \langle\Omega\rangle :_u \{\mathsf{T}\}$.

  Since judgements like $\{\mathsf{T}\}\ \lambda x.\Omega :_u \{\mathsf{T}\}$ are already derivable in the logic for total correctness for Pcf, the question arises as to how models used in logics for Pcf need only values as denotations? The answer is that there is a substantial difference between quasi-quotes and λ-abstractions in how the (non-terminating) sub-programs they harbour are accessed. The only way Ω can be executed in $\lambda x.\Omega$ is by application. This does not involve creating a new free variable bound to Ω (which would need a denotation in a corresponding model).

  In contrast, when unquoting a $\textsc{Pcf}_{\textsc{dp}}$ quasi-quote, e.g.

$$\texttt{let}\ \langle x\rangle = \langle\Omega\rangle\ \texttt{in}\ M,$$

  then x is free (as well as modal) in M, and will be bound to Ω, but without attempting to evaluate Ω. This is quite different from evaluating e.g. $\texttt{let}\ x = \lambda x.\Omega\ \texttt{in}\ M$, as we can see when comparing the evaluation of both terms side-by-side.

$$\begin{array}{rcl} \texttt{let}\ \langle x\rangle = \Omega\ \texttt{in}\ M & \to & M[\Omega/x] \\ \texttt{let}\ x = \lambda().\Omega\ \texttt{in}\ M & \to & M[\lambda().\Omega/x] \end{array}$$

  Models must accommodate this behaviour, and allowing modal variables to denote non-value programs does just this.

3.3. **Axioms and rules.** We have now ready to present the rules and axioms of our logic.

Axioms. The axioms come in two forms: those that are germane to $\textsc{Pcf}_{\textsc{dp}}$'s meta-programming extensions, and those that are not. All axioms for the Pcf logics of [18, 19, 20] remain valid, and are listed in Appendix A for completeness. Here we present only the axioms for the logical constructs not already available in the logics for Pcf, i.e. for the code evaluation predicate $x = \langle m\rangle\{A\}$.

Tacitly, we assume typability of all axioms. That means not only that all axioms must be typable, but conversely also that whenever an axiom is typable, it is a valid axiom. The axioms are given in Figure 3. The presentation uses the following abbreviations:

$$\mathsf{Ext_q}(xy) \text{ stands for } \forall a.(x = \langle z\rangle\{z = a\} \equiv y = \langle z\rangle\{z = a\}).$$

Axiom ($q1$) says that if the quasi-quote denoted by x makes A true (assuming the program in that quasi-quote is denoted by y), and in the same way makes B true, then it also makes $A \wedge B$ true, and vice versa. Axiom ($q2$) says that if the quasi-quote denoted by x contains a program, denoted by y, and makes $\neg A$ true, then it cannot be the case that under the same conditions A holds. The reverse implication is false, because $\neg x = \langle m\rangle\{A\}$ is also true when x denotes a quasi-quote whose contained program is diverging. But in this case, $x = \langle m\rangle\{\neg A\}$ is still false due to lacking termination. Next is ($q3$): $x = \langle m\rangle\{A\}$ says in particular that x denotes a quasi-quote containing a terminating program, so $\neg x = \langle m\rangle\{B\}$

$(q1)$	$x = \langle m\rangle\{A\} \wedge x = \langle m\rangle\{B\}$	$\equiv$	$x = \langle m\rangle\{A \wedge B\}$	
$(q2)$	$x = \langle m\rangle\{\neg A\}$	$\supset$	$\neg x = \langle m\rangle\{A\}$	
$(q3)$	$x = \langle m\rangle\{A\} \wedge \neg x = \langle m\rangle\{B\}$	$\equiv$	$x = \langle m\rangle\{A \wedge \neg B\}$	
$(q4)$	$x = \langle m\rangle\{A \wedge B\}$	$\equiv$	$A \wedge x = \langle m\rangle\{B\}$	$m \notin \mathsf{fv}(A)$
$(q5)$	$x = \langle m\rangle\{\forall a^{\alpha}.A\}$	$\equiv$	$\forall a^{\alpha}.x = \langle m\rangle\{A\}$	$a \neq x, m$
$(q6)$	$(A \supset B) \wedge x = \langle m\rangle\{A\}$	$\supset$	$x = \langle m\rangle\{B\}$	
$(term)$	$x \Downarrow$			x non modal
$(term_q)$	$x = \langle m\rangle\{A\}$	$\equiv$	$x = \langle m\rangle\{A \wedge m \Downarrow\}$	
(div)	$\neg\forall m^{\langle\alpha\rangle}.m = \langle\cdot\rangle$			
(ext_q)	$x = y$	$\equiv$	$\mathsf{Ext_q}(xy)$	x, y of type $\langle\alpha\rangle$ both non-modal
(q_α)	$x = \langle m\rangle\{A\}$	$\equiv$	$x = \langle n\rangle\{x = \langle m\rangle\{A \wedge m = n\}\}$	$n \neq x, n \in \mathsf{fv}(A)$ implies $n = m$

Figure 3: Key total correctness axioms for $\textsc{Pcf}_{\textsc{dp}}$. The remaining axioms are as for Pcf. Except where noted otherwise, free variables can be modal or non-modal.

can only be true because B is false. Axioms $(q4, q5)$ let us move formulae and quantifiers in and out of code-evaluation formulae, as long as free variables do not become bound in the process nor bound variables become free. Axiom (q6) allows us to weaken the assertion inside the code evaluation predicate. The reverse implication is trivially false. The axiom $(term)$ formalises that denotations of non-modal variables always terminate. The axiom $(term_q)$ enables us explicitly to express as a logical formula the fact that $x = \langle m\rangle\{A\}$ guarantees that the code denoted by x terminates. The axiom (q_α) may appear confusing on first sight, but it states something simple: namely that we can easily nest code evaluation predicates. The equality $m = n$ relates the two anchors. The axiom (div) simply states that not every quasi-quote holds code that terminates when executed. The code-extensionality axiom (ext_q) formalises what it means for two quasi-quotes to be equal: they must contain observationally indistinguishable code. The corresponding axiom (ext) for functions can be found in Appendix A together with other axioms for the Pcf-part of the language. Note that it is vital for x and y to be non-modal. The direction $\mathsf{Ext_q}(xy) \supset x = y$ is unsound otherwise, because $\mathsf{Ext_q}(xy)$ cannot distinguish between e.g. appropriately typed Ω and $\langle\Omega\rangle$.

Rules. The rules of inference can be found in Figures 4 and 5. We write $\vdash \{A\}\ M :_m \{B\}$ to indicate that $\{A\}\ M :_m \{B\}$ is derivable using these rules. Structural rules like Hoare's rule of consequence, are standard (see e.g. [18, 19, 20]) and used without further comment. All rules are typed. The typing of rules follows the corresponding typing of the programs occurring in the judgements, but with additions to account for auxiliary variables. Rather than detailing the typing for all rules, we exhibit an example. The typing rule for the unquote-construct is this:

$$\frac{\Gamma; \Delta \vdash M : \langle\alpha\rangle \quad \Gamma; \Delta, x : \alpha \vdash N : \beta}{\Gamma; \Delta \vdash \mathtt{let}\ \langle x\rangle = M\ \mathtt{in}\ N : \beta}$$

$$\frac{}{\{A[x/m] \wedge x \Downarrow\}\ x :_m \{A\}}\ \textsc{Var} \qquad \frac{}{\{A[\mathsf{c}/m]\}\ \mathsf{c} :_m \{A\}}\ \textsc{Const} \qquad \frac{\{A^{-g}\}\ M :_u \{B\}}{\{A\}\ \mu g.M :_u \{B[u/g]\}}\ \textsc{Rec}$$

$$\frac{\{A^{-x} \wedge B\}\ M :_m \{C\}}{\{A\}\ \lambda x^{\alpha}.M :_u \{\forall x.(B \supset u \bullet x = m\{C\})\}}\ \textsc{Abs} \qquad \frac{\{A\}\ M :_m \{B\} \quad \{B\}\ N :_n \{C[m+n/u]\}}{\{A\}\ M + N :_u \{C\}}\ \textsc{Add}$$

$$\frac{\{A\}\ M :_m \{B\} \quad \{B[b_i/m]\}\ N_i :_u \{C\} \quad b_1 = \mathtt{t} \quad b_2 = \mathtt{f} \quad i = 1,2}{\{A\}\ \mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2 :_u \{C\}}\ \textsc{If}$$

$$\frac{\{A\}\ M :_m \{B\} \quad \{B\}\ N :_n \{m \bullet n = u\{C\}\}}{\{A\}\ MN :_u \{C\}}\ \textsc{App} \qquad \frac{\{A\}\ M :_m \{B\}}{\{\mathsf{T}\}\ \langle M\rangle :_u \{A \supset u = \langle m\rangle\{B\}\}}\ \textsc{Quote}$$

$$\frac{\begin{array}{c}\{A\}\ M :_m \{E^{-x} \wedge (B^{-x} \supset m = \langle x\rangle\{C^{-m}\})\} \\ \{E \wedge (B \supset C) \wedge (m = \langle\cdot\rangle \supset m = \langle x\rangle)\}\ N :_u \{D^{-xm}\}\end{array}}{\{A\}\ \mathtt{let}\ \langle x\rangle = M\ \mathtt{in}\ N :_u \{D\}}\ \textsc{Unquote}^{+}$$

Figure 4: $\textsc{Pcf}_{\textsc{dp}}$ inference rules for total correctness.

The corresponding typing for [UNQUOTE$^+$] is rather similar:

$$\frac{\begin{array}{c}\Gamma; \Delta; \langle\alpha\rangle \vdash \{A\}\ M :_m \{E^{-x} \wedge (B^{-x} \supset m = \langle x\rangle\{C^{-m}\})\} \\ \Gamma, m : \langle\alpha\rangle; \Delta, x : \alpha; \beta \vdash \{E \wedge (B \supset C) \wedge (m = \langle\cdot\rangle \supset m = \langle x\rangle)\}\ N :_u \{D^{-xm}\}\end{array}}{\Gamma; \Delta; \beta \vdash \{A\}\ \mathtt{let}\ \langle x\rangle = M\ \mathtt{in}\ N :_u \{D\}}$$

All rules in Figure 5 and most rules in Figure 4 are standard and unchanged from [18, 19, 20] with three significant exceptions, explained next.

[VAR] adds $x \Downarrow$, i.e. $\exists a.x = a$ in the precondition. By construction of our models, $x \Downarrow$ is trivially true if x is non-modal. If x is modal, the situation is different because x may denote a non-terminating program. In this case $x \Downarrow$ constrains x so that it really denotes a value, as is required in a logic for total correctness.

[QUOTE] says that $\langle M\rangle$ always terminates (because the conclusion's precondition is simply T). Moreover, if u denotes the result of evaluating $\langle M\rangle$, i.e. $\langle M\rangle$ itself, then, assuming A holds (i.e., given the premise, if M terminates), u contains a terminating program, denoted m, making B true. Clearly, in a logic for total correctness, if M is not a terminating program, A will be equivalent to F, in which case, [QUOTE] does not make a non-trivial assertion about $\langle M\rangle$ beyond stating that it terminates.

[UNQUOTE$^+$] is similar to the usual rule for $\mathtt{let}\ x = M\ \mathtt{in}\ N$ which is easily derivable using [ABS, APP]:

$$\frac{\{A\}\ M :_x \{B\} \quad \{B\}\ N :_u \{C\}}{\{A\}\ \mathtt{let}\ x = M\ \mathtt{in}\ N :_u \{C\}}\ \textsc{Let}$$

The rule for $\mathtt{let}\ \langle x\rangle = M\ \mathtt{in}\ N$ is more difficult because a quasi-quote always terminates, but the code it contains may not. Moreover, even if M evaluates to a quasi-quote containing a divergent program, the overall expression may still terminate, because N uses the destructed quasi-quote in a way that cannot detect divergence. An example is as follows:

$$\mathtt{let}\ \langle x\rangle = \langle\Omega\rangle\ \mathtt{in}\ \lambda y.x.$$

$$\frac{\{A'\}\ M :_m \{B'\} \quad A \supset (A' \wedge (B' \supset B))}{\{A\}\ M :_m \{B\}}\ \textsc{Conseq-Kl} \qquad \frac{\{A \wedge B\}\ V :_m \{C\}}{\{A\}\ V :_m \{B \supset C\}}\ \wedge\text{-}\supset$$

$$\frac{\{\mathsf{F} \wedge A\}\ M :_m \{B\}}{\{\mathsf{F}\}\ M :_m \{A \wedge B\}}\ \wedge\text{-}\mathsf{F} \qquad \frac{\{A\}\ M :_m \{B \supset C\}}{\{A \wedge B\}\ M :_m \{C\}}\ \supset\text{-}\wedge \qquad \frac{\{A\}\ M :_m \{B\} \quad \{A'\}\ M :_m \{B\}}{\{A \vee A'\}\ M :_m \{B\}}\ \vee\text{-Pre}$$

$$\frac{\{A\}\ M :_m \{B\} \quad \{A\}\ M :_m \{B'\}}{\{A\}\ M :_m \{B \wedge B'\}}\ \wedge\text{-Post} \qquad \frac{\{A\}\ M :_m \{B^{-i}\} \quad i \text{ auxiliary}}{\{\exists i.A\}\ M :_m \{B\}}\ \textsc{Aux}_\exists$$

$$\frac{\{A^{-i}\}\ M :_m \{B\} \quad i \text{ auxiliary}}{\{A\}\ M :_m \{\forall i.B\}}\ \textsc{Aux}_\forall \qquad \frac{\{A\}\ M :_m \{B\}}{\{A \wedge C\}\ M :_m \{B \wedge C\}}\ \textsc{Invar}$$

Figure 5: Structural rules for total correctness.

Our rule [Unquote$^+$] deals with this complication in the following way. Assume

$$\{A\}\ M :_m \{B \supset m = \langle x \rangle \{C\}\}$$

holds. If M evaluates to a quasi-quote containing a divergent program, B would be equivalent to F. This is because in a logic for total correctness, $m = \langle x \rangle \{C\}$ means that the quasi-quote denoted by m must contain a converging program. Hence the only way that

$$B \supset m = \langle x \rangle \{C\}$$

can be true if it doesn't is if B is equivalent to F. In this case $B \supset m = \langle x \rangle \{C\}$ as a whole is equivalent to T, i.e. conveys no information Hence, since x does not occur freely in E, the denotation of x is not constrained by the left premise, hence the termination behaviour of N cannot depend on x. In other words N uses whatever x denotes in a way that makes the termination or otherwise of N independent of x. The additional formula E enables us easily to carry information from the conclusion of the assertion for M to the premise of the assertion about N.

Finally, the requirement

$$m = \langle \cdot \rangle \supset m = \langle x \rangle$$

in the precondition of the assertion for N makes the following fact available for reasoning about N: whenever M evaluates to a quasi-quote $\langle M' \rangle$, then M' is bound to x. This fact is not used in the reasoning about example programs in this paper. However, it appears to be vital for proving completeness, see Proposition 5.5 in Section 5.[5] In reasoning about programs we typically use the following simpler rule.

$$\frac{\{A\}\ M :_m \{E^{-x} \wedge (B^{-x} \supset m = \langle x \rangle \{C^{-m}\})\} \quad \{E \wedge (B \supset C)\}\ N :_u \{D^{-xm}\}}{\{A\}\ \mathtt{let}\ \langle x \rangle = M\ \mathtt{in}\ N :_u \{D\}}\ \textsc{Unquote}$$

In many derivations, E is simply T and omitted. Clearly, [Unquote] is easily derivable from [Unquote$^+$].

The rule [Conseq-Kl] is slightly more elaborate than Hoare's original rule of consequence, present already in [17], and repeated below for comparison:

[5]The previous, short version of this paper [5] used only [Unquote], not [Unquote$^+$]. It is unclear if Prop. 5.5 can be established with [Unquote] alone.

$$\frac{A \supset A' \quad \{A'\}\ M :_m \{B'\} \quad B' \supset B}{\{A\}\ M :_m \{B\}}\ \textsc{Conseq}$$

[Conseq] is usually sufficient in practise. But for proving relative completeness in Section 5, [Conseq-Kl], going back at least as far Kleymann [25], is more convenient. Using [Conseq-Kl], Hoare's [Conseq] is easily derivable.

We note that the rules for programs in the Pcf-fragment of $\textsc{Pcf}_{\textsc{dp}}$ are the same as those in the logic for Pcf [20], apart from a slightly different presentation. The only apparent difference is in the respective rules for variables (with $\textsc{Pcf}_{\textsc{dp}}$ on the left, Pcf on the right):

$$\{A[x/m] \wedge x \Downarrow\}\ x :_m \{A\} \qquad\qquad \{A[x/m]\}\ x :_m \{A\}$$

However, this is misleading for two reasons:

- For non-modal variables x, by axiom $(term)$, $x \Downarrow$ always holds, so $A[x/m]$ can be inferred trivially from $A[x/m] \wedge x \Downarrow$ and vice versa.
- We could have split the $\textsc{Pcf}_{\textsc{dp}}$ rule for variables into two as follows:

$$\frac{x \text{ non-modal}}{\{\mathsf{T}\}\ x :_m \{x = m\}}\ \textsc{Var} \qquad \frac{x \text{ modal}}{\{x \Downarrow\}\ x :_m \{x = m\}}\ \textsc{Var}_m$$

  Indeed that is what we will do later in Section 5. The reason for using a combined rule in this Section is economy of presentation.

The ability to reason about the Pcf-fragment in our logic for $\textsc{Pcf}_{\textsc{dp}}$ is significant for two reasons. First, on the theoretical side, it shows that adding MP features is a modular extension of our base language and base logic, raising the intriguing question if modularity can be retained in situations where the base language has rich effects like state or exceptions, or where the MP features are more extensive (e.g. compile-time meta-programming), or allow MP on open code (that is, code with free variables). Secondly, on the pragmatic side, it makes life easier, because the specification and verification of programs and program parts that do not use MP features do not have to pay a price in terms of additional complexity vis-a-vis the logic for Pcf.

3.4. **Soundness.** We now establish that the axioms and rules introduced in the previous subsection are sound.

Theorem 3.2.

(1) *All axioms are sound.*
(2) *All rules are sound.*

Proofs for axioms and rules not relating to $\textsc{Pcf}_{\textsc{dp}}$'s meta-programming extensions are straightforward extensions of the corresponding proofs for Pcf-logics like [4, 20, 22, 38] and mostly omitted. Before embarking on proofs, we collect facts that will be useful later.

Proposition 3.3.

(1) *Assume the formula A is typable under $\Gamma, x : \alpha; \Delta$ and $\Gamma; \Delta, x : \alpha$. Let $(\xi \cdot x : V, \sigma)$ be a model of type $\Gamma, x : \alpha; \Delta$ and $(\xi, \sigma \cdot x : V)$ be a model of type $\Gamma; \Delta, x : \alpha$. Then*

$$(\xi \cdot x : V, \sigma) \models A \qquad \textit{iff} \qquad (\xi, \sigma \cdot x : V) \models A.$$

(2) *Let $\Gamma_i; \Delta_i \vdash e : \alpha$, $\Gamma_i; \Delta_i \vdash A$, and assume that η_i is a $\Gamma_i; \Delta_i$-model for $i = 1, 2$. Then:*
   - *$\eta_1(x) \simeq \eta_2(x)$ for all $x \in \mathsf{fv}(e)$ implies $[\![e]\!]_{\eta_1} \simeq [\![e]\!]_{\eta_2}$.*
   - *$\eta_1(x) \simeq \eta_2(x)$ for all $x \in \mathsf{fv}(A)$ implies $\eta_1 \models A$ iff $\eta_2 \models A$.*

(3) *Let* $\Gamma;\Delta \vdash e : \alpha$, $\Gamma, x:\alpha;\Delta \vdash e' : \beta$, $\Gamma, x:\alpha;\Delta \vdash A$ *and* $\eta \stackrel{\text{def}}{=} (\xi,\sigma)$ *be an appropriately typed model such that* $[\![e]\!]_\eta \Downarrow$.
- $[\![e'[e/x]]\!]_\eta \simeq [\![e']\!]_{(\xi\cdot x:[\![e]\!]_\eta,\sigma)}$.
- $\eta \models A[e/x]$ *iff* $\forall V.(V \simeq [\![e]\!]_{\xi,\sigma} \supset (\xi \cdot x : V, \sigma) \models A)$.
- $\eta \models A[e/x]$ *iff* $\eta \models \exists x.(A \wedge x = e)$.

(4) *Let* $\Gamma;\Delta, x:\alpha \vdash M : \beta$, *assume* N *is a closed program of type* α. *Given a* $\Gamma;\Delta$*-model* $\eta \stackrel{\text{def}}{=} (\xi,\sigma)$ *it holds that*

$$M\eta[N/x] = M(\xi, \sigma \cdot x : N).$$

Proof. The content of this proposition is straightforward, hence proofs are omitted. □

Proof [of Theorem 3.2.1]. The proof of axioms $(q1)$ - $(q5)$ for quasi-quotes are essentially just trivial instances of first-order logical laws and omitted, except that we explicitly prove $(q3)$ as a representative example.

To establish $(q3)$, let $\eta \stackrel{def}{=} (\xi,\sigma)$ be an appropriately typed model. Then we reason as follows:

1	$\eta \models x = \langle y\rangle\{A\} \wedge \neg x = \langle y\rangle\{B\}$	
2	$\exists M, V.\eta(x) = M, M \Downarrow V, (\xi, \sigma \cdot y : V) \models A$	*1*
3	$\eta(x) = M, M \Downarrow V$	*2*
4	$(\xi, \sigma \cdot y : V) \not\models B$	*1, 3*
5	$(\xi, \sigma \cdot y : V) \models A \wedge \neg B$	*2, 4*

The reverse implication is similar.

The axiom $(term)$ is immediate from the definition of models.

Regarding $(term_q)$, let $\eta \stackrel{def}{=} (\xi,\sigma)$ be an appropriately typed model.

1	$\eta \models x = \langle m\rangle\{A\}$	
2	$\exists M, V.\eta(x) = M, M \Downarrow V, (\xi, \sigma \cdot m : V) \models A$	*1*
3	$\exists M, V.\eta(x) = M, M \Downarrow V, (\xi \cdot y : V, \sigma \cdot m : V) \models m = y$	
4	$\exists M, V.\eta(x) = M, M \Downarrow V, (\xi, \sigma \cdot m : V) \models \exists y.m = y$	*3*
5	$\exists M, V.\eta(x) = M, M \Downarrow V, (\xi, \sigma \cdot m : V) \models m \Downarrow$	*4*
6	$\exists M, V.\eta(x) = M, M \Downarrow V, (\xi, \sigma \cdot m : V) \models A \wedge m \Downarrow$	*2, 5*
7	$(\xi, \sigma) \models x = \langle m\rangle\{A \wedge m \Downarrow\}$	*6*

The reverse implication is immediate.

The soundness of (div) is immediate from the construction of the model and the satisfaction relation: (div) states that not every quasi-quote contains a terminating program, e.g. $\langle\Omega\rangle$.

Finally, for (ext_q), let $\eta \stackrel{def}{=} (\xi, \sigma)$ be an appropriately typed model, where $x, y \in \mathsf{dom}(\eta)$. We assume that x, y are of type $\langle\alpha\rangle$.

1	$\eta \models x = y$	
2	$\eta' \stackrel{def}{=} (\xi \cdot a : V, \sigma)$	*V arbitrary value*
3	$\eta' \models x = \langle m\rangle\{m = a\}$	*Assumption*
4	$\exists M, W.[\![x]\!]_{\eta'} = \langle M\rangle, M \Downarrow W, \eta'' \models m = a,$ where $\eta'' \stackrel{def}{=} (\xi \cdot a : V, \sigma \cdot m : W)$	*3*
5	$W = [\![m]\!]_{\eta''} \cong [\![a]\!]_{\eta''} = V$	*2, 4*
6	$[\![x]\!]_{\eta} \simeq [\![y]\!]_{\eta}$	*1*
7	$[\![x]\!]_{\eta'} \simeq [\![y]\!]_{\eta'}$	*6, Prop. 3.3.2*
8	$\exists M', W'.[\![y]\!]_{\eta'} = \langle M'\rangle, M' \Downarrow W', \eta''' \models m = a,$ where $\eta''' \stackrel{def}{=} (\xi \cdot a : V, \sigma \cdot m : W')$	*4, 7, Prop. 2.1*
9	$\eta' \models y = \langle m\rangle\{m = a\}$	*8*

For the reverse implication we assume that x and y are both of type $\langle\alpha\rangle$, and that $\eta = (\xi, \sigma)$.

$$\eta \models \mathsf{Ext_q}(xy)$$

which means there are two cases for any chosen value V of appropriate type, where $\eta' = (\xi \cdot a : V, \sigma)$.

– $[\![x]\!]_{\eta'} \Downarrow \langle M\rangle, M \Downarrow W, W \simeq V$, and also $[\![y]\!]_{\eta'} \Downarrow \langle M'\rangle, M' \Downarrow W', W' \simeq V$. By Proposition 2.1. (6) this implies

$$[\![x]\!]_{\eta'} \simeq \langle M\rangle, M \simeq W, W \simeq V$$

and

$$[\![y]\!]_{\eta'} \simeq \langle M'\rangle, M' \simeq W', W' \simeq V.$$

Hence $W \simeq W'$. As $\simeq$ is a congruence, the above in turn implies $[\![x]\!]_{\eta'} \simeq [\![y]\!]_{\eta'}$, whence

$$\eta \models x = y$$

using the fact that a is different from x, y and Proposition 3.3.2.

– $[\![x]\!]_{\eta'} \Uparrow$ and $[\![y]\!]_{\eta'} \Uparrow$. In this case $[\![x]\!]_{\eta'} \simeq [\![y]\!]_{\eta'}$ by Proposition 2.1. (4). Hence again

$$\eta \models x = y$$

as in the previous case. □

Proof [of Theorem 3.2.2]. We proceed by induction on the derivation of the inference. All rules but [VAR, QUOTE, UNQUOTE$^+$] are essentially unchanged from [20] so we concentrate on these three.

For [VAR] we have two subcases, depending on whether the variable under assertion is modal or not. The latter case is trivial, so we deal only with the former (which is also easy). In this case the rule is typed as follows:

$$\overline{\Gamma; \Delta, x : \alpha; \alpha \vdash \{A[x/m] \wedge x \Downarrow\}\ x :_m \{A\}}$$

Let $\eta \stackrel{def}{=} (\xi, \sigma \cdot x : M)$ be a $\Gamma; \Delta, x : \alpha$-model such that

$$\eta \models A[x/m] \wedge x \Downarrow . \tag{3.1}$$

Then in particular

$$\eta \models \exists a^{\alpha}.x = a.$$

By the semantics of quantification, we know that some *value* V must exist with $(\xi \cdot a : V, \sigma \cdot x : M) \models x = a$. Thus by definition of the interpretation of equality,

$$M \simeq V.$$

Hence, since $x\eta = M$ immediately

$$x\eta \Downarrow V$$

which means the program under assertion terminates. From (3.1) we also get that

$$\eta \models A[x/m]$$

and since $[\![x]\!]_\eta \Downarrow V$, we can apply Proposition 3.3.3 and obtain.

$$(\xi \cdot m : V, \sigma \cdot x : M) \models A.$$

This concludes our discussion of [Var].

The case of [Quote] is straightforward and based on Proposition 3.3.1. The rule is typed as follows.

$$\frac{\Gamma; \Delta; \alpha \vdash \{A\}\ M :_m \{B\}}{\Gamma; \Delta; \langle\alpha\rangle \vdash \{\mathsf{T}\}\ \langle M\rangle :_u \{A \supset u = \langle m\rangle\{B\}\}}$$

Let $\eta = (\xi, \sigma)$ be a $\Gamma; \Delta$-model and define $\eta' \stackrel{def}{=} (\xi \cdot u : \langle M\eta\rangle, \sigma)$. Since $\langle M\rangle\eta = \langle M\eta\rangle$ is already a value, we need only show that $\eta' \models A \supset u = \langle m\rangle\{B\}$. To this end, let $\eta' \models A$. Then we reason as follows (in this derivation and others, (IH) is used as an abbreviation of 'induction hypothesis'):

1	$\eta' \models A$	
2	$\eta \models A$	*1, $u \notin \mathsf{fv}(A)$, Prop. 3.3.2*
3	$M\eta \Downarrow V$ and $(\xi \cdot m : V, \sigma) \models B$	*(IH), 2*
4	$(\xi, \sigma \cdot m : V) \models B$	*3, Prop. 3.3.1*
5	$M\eta \simeq V$	*3, Prop. 2.1. (6)*
6	$(\xi, \sigma \cdot m : M\eta) \models B$	*4, 5, Prop. 3.3.2*
7	$(\xi \cdot u : \langle M\eta\rangle, \sigma \cdot m : M\eta) \models B$	*6, $u \notin \mathsf{fv}(B)$, Prop. 3.3.2*

This concludes the case of [Quote].

Finally, we establish the soundness of [Unquote$^+$]. Choose a $\Gamma; \Delta$-model $\eta \stackrel{def}{=} (\xi, \sigma)$ such that $\eta \models A$. By the (IH) we know that

$$M\eta \Downarrow \langle M'\rangle \qquad \underbrace{(\xi \cdot m : \langle M'\rangle, \sigma)}_{\eta'} \models E \wedge (B \supset m = \langle x\rangle\{C\}) \tag{3.2}$$

Now we have two cases:

– $\eta' \models E$, $\eta' \models B$ and $\eta' \models m = \langle x\rangle\{C\}$.
– $\eta' \models E$ but $\eta' \not\models B$.

We start with the former:

1	$\eta' \models E$ and $\eta' \models B$ and $\eta' \models m = \langle x \rangle \{C\}$	
2	$[\![m]\!]_{\eta'} = \langle M' \rangle, M' \Downarrow V$ and $(\xi \cdot m : \langle M' \rangle, \sigma \cdot x : V) \models C$	*1*
3	$\eta'' \stackrel{def}{=} (\xi \cdot m : \langle M' \rangle, \sigma \cdot x : V)$	
4	$\eta'' \models E \wedge B$	*1, Prop. 3.3.2*
5	$\eta'' \models E \wedge B \wedge C$	*2, 4*
6	$\eta'' \models E \wedge (B \supset C)$	*5*
7	$\eta''' \stackrel{def}{=} (\xi \cdot m : \langle M' \rangle, \sigma \cdot x : M')$	
8	$M' \simeq V$	*2, Prop. 2.1. (6)*
9	$\eta''' \models E \wedge (B \supset C)$	*6, 8, Prop. 3.3.2*
10	$\eta''' \models m = \langle \cdot \rangle \wedge m = \langle x \rangle$	*2, 8, Prop. 3.3.2*
11	$\eta''' \models m = \langle \cdot \rangle \supset m = \langle x \rangle$	*10*
12	$N\eta''' \Downarrow W$ and $(\xi \cdot m : \langle M' \rangle \cdot u : W, \sigma \cdot x : M') \models D$	*11, (IH)*

Now we can consider the reductions of $(\texttt{let}\ \langle x \rangle = M\ \texttt{in}\ N)\eta$:

$$
\begin{array}{lcll}
(\texttt{let}\ \langle x \rangle = M\ \texttt{in}\ N)\eta & = & \texttt{let}\ \langle x \rangle = M\eta\ \texttt{in}\ N\eta & \\
& \twoheadrightarrow & \texttt{let}\ \langle x \rangle = \langle M' \rangle\ \texttt{in}\ N\eta & \textit{by } (3.2) \\
& \to & N\eta[M'/x] & \\
& = & N\eta''' & \textit{Prop. } 3.3.4 \\
& \Downarrow & W & \textit{by } 10
\end{array}
$$

By Line 10 and Proposition 3.3.2 we know that $(\xi \cdot u : W, \sigma) \models D$ since $m, x \notin \mathsf{fv}(D)$.

Now we consider the second case $\eta' \not\models B$. In this simpler case we reason as follows:

1	$\eta' \models E$ and $\eta' \not\models B$	
2	$\eta'' \stackrel{def}{=} (\xi \cdot m : \langle M' \rangle, \sigma \cdot x : M')$	
3	$\eta'' \models E$ and $\eta'' \not\models B$	*1, Prop. 3.3.2*
4	$\eta'' \models E \wedge (B \supset C)$	*3*
5	$\eta'' \models m = \langle \cdot \rangle \supset m = \langle x \rangle$	*by construction*
6	$\eta'' \models E \wedge (B \supset C) \wedge (m = \langle \cdot \rangle \supset m = \langle x \rangle)$	*4, 5*
7	$N\eta'' \Downarrow W$ and $(\xi \cdot m : \langle M' \rangle \cdot u : W, \sigma \cdot x : M') \models D$	*6, (IH)*

As in the previous case we now consider the reductions of $(\texttt{let}\ \langle x\rangle = M\ \texttt{in}\ N)\eta$:

$$\begin{array}{lcll}(\texttt{let}\ \langle x\rangle = M\ \texttt{in}\ N)\eta & = & \texttt{let}\ \langle x\rangle = M\eta\ \texttt{in}\ N\eta & \\ & \twoheadrightarrow & \texttt{let}\ \langle x\rangle = \langle M'\rangle\ \texttt{in}\ N\eta & \textit{by } (3.2)\\ & \to & N\eta[M'/x] & \\ & = & N\eta'' & \textit{Prop. } 3.3.4\\ & \Downarrow & W & \textit{by } 7\end{array}$$

By Line 7 and Proposition 3.3.2 we know that $(\xi \cdot u : W, \sigma) \models D$ since $m, x \notin \mathsf{fv}(D)$. This concludes the reasoning for [Unquote$^+$].

We now verify the structural rule [$\wedge$-$\supset$] to demonstrate that the soundness of the structural rules (which have already been shown sound in the context of Pcf [20]) is not affected by our extension of Pcf with facilities for meta-programming. The proofs for the other structural rules are similarly straightforward.

Let $\eta = (\xi, \sigma)$ be an appropriately typed model. Then we reason as follows:

1	$\eta \models A$	
2	$V\eta \Downarrow V\eta$	*V value*
3	$\eta' \stackrel{def}{=} (\xi \cdot m : V\eta, \sigma)$	
4	$\eta' \models B$	*Assumption*
5	$\eta \models B$	*4, $m \notin \mathsf{fv}(B)$, Prop. 3.3.2*
6	$\eta \models A \wedge B$	*1, 5*
7	$V\eta \Downarrow V\eta, \eta' \models C$	*V value, 6, (IH)*
8	$V\eta \Downarrow V\eta, \eta' \models B \supset C$	*4, 7*
9	$\eta \models \{A\}\ V :_m \{B \supset C\}$	*1, 8*

Note that as with Pcf, the condition that the program be a value in [$\wedge$-$\supset$] cannot be dropped in a logic for total correctness, because

$$\frac{\{\mathsf{T} \wedge \mathsf{F}\}\ \Omega :_m \{C\}}{\{\mathsf{T}\}\ \Omega :_m \{\mathsf{F} \supset C\}}$$

is unsound.

Finally, we prove sound [Rec], the rule for recursion in a total correctness setting. It's compellingly simple form was first given in [19]. Here we need to show that the addition of MP facilities does not void soundness. This is straightforward.

Let $\eta = (\xi, \sigma)$ be an appropriately typed model with $g \notin \mathsf{dom}(\eta)$, and $\eta' \stackrel{def}{=} (\xi \cdot g : \mu g.M\eta, \sigma)$.

1	$\eta \models A$	*Assumption*
2	$\eta' \models A$	*1, $g \notin \mathsf{fv}(A)$, Prop. 3.3.2*
3	$M\eta' \Downarrow V,\quad \overbrace{(\xi \cdot g : \mu g.M\eta, u : V\sigma)}^{\eta''} \models B$	*2, (IH)*
4	$V \cong M\eta' \cong \mu g.M\eta$	*3, Prop. 2.1. (3)*

5	$\eta'' \models g = u$	*3, 4*
6	$\eta'' \models B \wedge g = u$	*5*
7	$(\xi \cdot u : V, \sigma) \models \exists g.(B \wedge g = u)$	*6*
8	$(\xi \cdot u : V, \sigma) \models B[u/g]$	*7, Prop. 3.3.3*
9	$(\xi \cdot u : \mu g.M\eta, \sigma) \models B[u/g]$	*8, Prop. 3.3.2*
10	$(\xi \cdot u : \mu g.M\eta, \sigma) \models B$	*3, $g \notin \mathsf{fv}(B)$, Prop. 3.3.2*
11	$\mu g.M\eta \Downarrow \mu g.M\eta, \quad (\xi \cdot u : \mu g.M\eta, \sigma) \models B$	*10*

This last proof is unchanged from the soundness proof for [Rec] in Pcf, although 'under the hood' some used propositions need additional work to do with the generalised language and notion of model. This is also true for all other rules and axioms that involve only Pcf syntax. □

4. Reasoning examples

We now put our logic to use by reasoning about some of the programs introduced in Section 2. The derivations use the abbreviations of Section 3 and and often omit steps that are trivial, or irrelevant from the perspective of meta-programming.

Example 4.1. We begin with the simple program $\{\mathsf{T}\}\ \langle 1{+}2\rangle :_m \{m = \langle 3\rangle\}$. The derivation is straightforward.

1	$\{\mathsf{T}\}\ 1 + 2 :_a \{a = 3\}$	
2	$\{\mathsf{T}\}\ \langle 1 + 2\rangle :_m \{\mathsf{T} \supset m = \langle a\rangle\{a = 3\}\}$	Quote, *1*
3	$\{\mathsf{T}\}\ \langle 1 + 2\rangle :_m \{m = \langle 3\rangle\}$	Conseq, *2*

Example 4.2. This example deals with the code of a non-terminating program. We derive $\{\mathsf{T}\}\ \langle\Omega\rangle :_m \{\mathsf{T}\}$. This is the strongest total correctness assertion about $\langle\Omega\rangle$. In the proof, we assume that $\{\mathsf{F}\}\ \Omega :_a \{\mathsf{T}\}$ is derivable, which is easy to show.

1	$\{\mathsf{F}\}\ \Omega :_a \{\mathsf{T}\}$	
2	$\{\mathsf{T}\}\ \langle\Omega\rangle :_m \{\mathsf{F} \supset m = \langle a\rangle\{\mathsf{T}\}\}$	Quote, *1*
3	$\{\mathsf{T}\}\ \langle\Omega\rangle :_m \{\mathsf{T}\}$	Conseq, *2*

Example 4.3. The third example destructs a quasi-quote and then injects the resulting program into another quasi-quote.

$$\{\mathsf{T}\}\ \mathtt{let}\ \langle x\rangle = \langle 1 + 2\rangle\ \mathtt{in}\ \langle x + 3\rangle :_m \{m = \langle 6\rangle\}$$

We derive the assertion in small steps to demonstrate how to apply our logical rules.

1	$\{\mathsf{T}\}\ \langle 1+2\rangle :_m \{m = \langle 3\rangle\}$	*Ex. 1*
2	$\{(a=3)[x/a] \wedge x \Downarrow\}\ x :_a \{a = 3\}$	Var
3	$\{x = 3 \wedge x \Downarrow\}\ x :_a \{a = 3\}$	Conseq, *2*
4	$\{\mathsf{T}\}\ 3 :_b \{b = 3\}$	Const, Conseq
5	$\{a = 3 \wedge x \Downarrow\}\ 3 :_b \{a = 3 \wedge b = 3\}$	Invar, *4*
6	$\{a = 3\}\ 3 :_b \{(c=6)[a+b/c]\}$	Conseq, *5*
7	$\{x = 3 \wedge x \Downarrow\}\ x + 3 :_c \{c = 6\}$	Add, *3, 6*
8	$\{\mathsf{T}\}\ \langle x+3\rangle :_u \{(x = 3 \wedge x \Downarrow) \supset u = \langle c\rangle\{c = 6\}\}$	Quote, *7*
9	$\{x = 3 \wedge x \Downarrow\}\ \langle x+3\rangle :_u \{u = \langle c\rangle\{c = 6\}\}$	$\supset$-$\wedge$, *8*
10	$\{\mathsf{T}\}\ \langle 1+2\rangle :_m \{\mathsf{T} \supset m = \langle x\rangle\{x = 3 \wedge x \Downarrow\}\}$	Conseq, *1*
11	$\{\mathsf{T} \supset (x = 3 \wedge x \Downarrow)\}\ \langle x+3\rangle :_u \{u = \langle 6\rangle\}$	Conseq, *9*
12	$\{\mathsf{T}\}\ \mathtt{let}\ \langle x\rangle = \langle 1+2\rangle\ \mathtt{in}\ \langle x+3\rangle :_u \{u = \langle 6\rangle\}$	Unquote, *10, 11*

In Line 10, we use the ($term_q$) axiom among others.

Example 4.4. We now show that when a quasi-quote containing a non-terminating sub-program is destructed, but the resulting sub-program is not used, the overall program still terminates. This reflects the operational semantics in Section 2.

$$\{\mathsf{T}\}\ \mathtt{let}\ \langle x\rangle = \langle \Omega\rangle\ \mathtt{in}\ \langle 1+2\rangle :_m \{m = \langle 3\rangle\}$$

The derivation follows:

1	$\{\mathsf{T}\}\ \langle \Omega\rangle :_m \{\mathsf{T}\}$	*Ex. 2*
2	$\{\mathsf{T}\}\ \langle \Omega\rangle :_m \{\mathsf{F} \supset m = \langle a\rangle\{\mathsf{T}\}\}$	Conseq, *1*
3	$\{\mathsf{T}\}\ \langle 1+2\rangle :_m \{m = \langle 3\rangle\}$	*Ex. 1*
4	$\{\mathsf{F} \supset \mathsf{T}\}\ \langle 1+2\rangle :_m \{m = \langle 3\rangle\}$	Conseq, *3*
5	$\{\mathsf{T}\}\ \mathtt{let}\ \langle x\rangle = \langle \Omega\rangle\ \mathtt{in}\ \langle 1+2\rangle :_m \{m = \langle 3\rangle\}$	Unquote, *2, 4*

Example 4.5. This example extracts a non-terminating program from a quasi-quote, and injects it into a new quasi-quote. Our total-correctness logic cannot say anything non-trivial about the resulting quasi-quote (cf. Example 2):

$$\{\mathsf{T}\}\ \mathtt{let}\ \langle x\rangle = \langle \Omega\rangle\ \mathtt{in}\ \langle x\rangle :_u \{\mathsf{T}\}$$

The derivation is straightforward.

1	$\{\mathsf{T}\}\ \langle \Omega\rangle :_m \{\mathsf{T}\}$	*Ex. 2*
2	$\{\mathsf{T}\}\ \langle \Omega\rangle :_m \{\mathsf{F} \supset m = \langle x\rangle\{\mathsf{T}\}\}$	*1,* Conseq

3	$\{\mathsf{F}[x/a] \wedge x \Downarrow\}\ x :_a \{\mathsf{F}\}$	VAR
4	$\{\mathsf{F}\}\ x :_a \{\mathsf{T}\}$	CONSEQ, *3*
5	$\{\mathsf{T}\}\ \langle x\rangle :_u \{\mathsf{F} \supset u = \langle a\rangle\{\mathsf{T}\}\}$	QUOTE, *4*
6	$\{\mathsf{F} \supset \mathsf{T}\}\ \langle x\rangle :_u \{\mathsf{T}\}$	CONSEQ, *5*
7	$\{\mathsf{T}\}\ \mathtt{let}\ \langle x\rangle = \langle\Omega\rangle\ \mathtt{in}\ \langle x\rangle :_u \{\mathsf{T}\}$	UNQUOTE, *2, 6*

The examples below make use of the following convenient forms of the recursion rule and [UNQUOTE].

$$\frac{\{A^{\text{-}gn} \wedge \forall 0 \le i < n.B[i/n][g/u]\}\ \lambda x.M :_u \{B^{\text{-}g}\}}{\{A\}\ \mu g.\lambda x.M :_u \{\forall n \ge 0.B\}}\ \textsc{Rec'}$$

It is easily derived from [REC] using [AUX$_\forall$].

Example 4.6. We now reason about $\mathsf{lift}_{\mathsf{Int}}$ from Section 2. In the proof we assume that i, n range over non-negative integers. Let $A^u_n \stackrel{def}{=} u \bullet n = m\{m = \langle n\rangle\}$. We now establish the following assertion from Section 3: $\{\mathsf{T}\}\ \mathsf{lift}_{\mathsf{Int}} :_u \{\forall n.A^u_n\}$. We set $C \stackrel{def}{=} i \le n \wedge \forall j < n.A^g_j$, $D \stackrel{def}{=} i > 0 \wedge \forall r.(0 \le r < n \supset g \bullet r = m\{m = \langle r\rangle\})$ and $P \stackrel{def}{=} \mathtt{let}\ \langle x\rangle = g(i-1)\ \mathtt{in}\ \langle x+1\rangle$.

1	$\{C\}\ i \le 0 :_b \{C \wedge (b = \mathsf{t} \equiv i \le 0)\}$	
2	$\{\mathsf{T}\}\ \langle 0\rangle :_m \{m = \langle 0\rangle\}$	*Like Ex. 1*
3	$\{i = 0\}\ \langle 0\rangle :_m \{m = \langle i\rangle\}$	INVAR, CONSEQ, *2*
4	$\{(C \wedge b = \mathsf{t} \equiv i \le 0)[\mathsf{t}/b]\}\ \langle 0\rangle :_m \{m = \langle i\rangle\}$	CONSEQ, *3*
5	$\{x = i - 1\}\ \langle x+1\rangle :_m \{m = \langle i\rangle\}$	*Like Ex. 3*
6	$\{\mathsf{T} \supset x = i - 1\}\ \langle x+1\rangle :_m \{m = \langle i\rangle\}$	CONSEQ, *5*
7	$\{(C \wedge b = \mathsf{t} \equiv i \le 0)[\mathsf{f}/b]\}\ g :_s \{D\}$	VAR
8	$\{D\}\ i - 1 :_r \{g \bullet r = t\{t = \langle i-1\rangle\}\}$	
9	$\{(C \wedge b = \mathsf{t} \equiv i \le 0)[\mathsf{f}/b]\}\ g(i-1) :_t \{t = \langle i-1\rangle\}$	APP, *7, 8*
10	$\{(C \wedge b = \mathsf{t} \equiv i \le 0)[\mathsf{f}/b]\}\ P :_m \{m = \langle i\rangle\}$	UNQUOTE, CONSEQ, *6, 9*
11	$\{C\}\ \mathtt{if}\ i \le 0\ \mathtt{then}\ \langle 0\rangle\ \mathtt{else}\ P :_m \{m = \langle i\rangle\}$	IF, *4, 10*
12	$\{\mathsf{T}\}\ \lambda i.\mathtt{if}\ i \le 0\ \mathtt{then}\ \langle 0\rangle\ \mathtt{else}\ P :_u \{\forall i.(C \supset A^u_i)\}$	ABS, *11*
13	$\{\forall j < n.A^g_j\}\ \lambda i.\mathtt{if}\ i \le 0\ \mathtt{then}\ \langle 0\rangle\ \mathtt{else}\ P :_u \{\forall i \le n.A^u_i\}$	CONSEQ $\supset$-$\wedge$, *12*
14	$\{\mathsf{T}\}\ \mathsf{lift}_{\mathsf{Int}} :_u \{\forall n.\forall i \le n.A^u_n\}$	REC', *13*
15	$\{\mathsf{T}\}\ \mathsf{lift}_{\mathsf{Int}} :_u \{\forall n.A^u_n\}$	CONSEQ, *14*

Example 4.7. We close this section by reasoning about the staged power function from Section 2. Assuming that i, j, k, n range over non-negative integers, we define $B^u_n \stackrel{def}{=} u \bullet n =$

$m\{m = \langle y\rangle\{\forall j.y \bullet j = j^n\}\}$. In the derivation, we provide less detail than in previous proofs for readability.

1	$C \stackrel{def}{=} n \leq k \wedge \forall i < k.B_i^p \qquad D \stackrel{def}{=} C \wedge (b = \mathtt{t} \wedge n \leq 0)$	
2	$P \stackrel{def}{=} \mathtt{let}\ \langle q\rangle = p(n-1)\ \mathtt{in}\ \langle \lambda x.x \times (q\ x)\rangle$	
3	$\{C\}\ n \leq 0 :_b \{D\}$	
4	$\{D[\mathsf{t}/b]\}\ \langle \lambda x.1\rangle :_m \{m = \langle y\rangle\{\forall j.y \bullet j = j^n\}\}$	*Like prev. examples*
5	$\{D[\mathsf{f}/b]\}\ p(n-1) :_r \{\mathsf{T} \supset r = \langle q\rangle\{\forall j.q \bullet j = j^{n-1}\}\}$	*Like Ex. 6*
6	$\{\mathsf{T} \supset \forall j.q \bullet j = j^{n-1}\}\ \langle \lambda x.x \times (q\ x)\rangle :_m \{m = \langle y\rangle\{\forall j.y \bullet j = j^n\}\}$	*Like Ex. 6*
7	$\{D[\mathsf{f}/b]\}\ P :_m \{m = \langle y\rangle\{\forall j.y \bullet j = j^n\}\}$	UNQUOTE, *5, 6*
8	$\{C\}\ \mathtt{if}\ n \leq 0\ \mathtt{then}\ \langle \lambda x.1\rangle\ \mathtt{else}\ P :_m \{m = \langle y\rangle\{\forall j.y \bullet j = j^n\}\}$	IF, *7*
9	$\{\mathsf{T}\}\ \lambda n.\mathtt{if}\ n \leq 0\ \mathtt{then}\ \langle \lambda x.1\rangle\ \mathtt{else}\ P :_u \{\forall n \leq k.((\forall i < k.B_i^p) \supset B_n^u)\}$	ABS, *8*
10	$\{\forall i < k.B_i^p\}\ \lambda n.\mathtt{if}\ n \leq 0\ \mathtt{then}\ \langle \lambda x.1\rangle\ \mathtt{else}\ P :_u \{\forall n \leq k.B_n^u\}$	CONSEQ, *9*
11	$\{\mathsf{T}\}\ \mathsf{power} :_u \{\forall k.\forall n \leq k.B_n^u\}$	REC', *10*
12	$\{\mathsf{T}\}\ \mathsf{power} :_u \{\forall n.B_n^u\}$	CONSEQ, *11*

5. Completeness

This section poses, and then answers in the affirmative, three important meta-logical questions about the logic introduced in previous sections:

– Is the logic *relatively complete* in the sense of Cook [12]?
– Is the logic *observationally complete* [19]?
– Does the logic have *characteristic formulae* [1]?

The first question can be seen as a reversal of soundness: does

$$\models \{A\}\ M :_m \{B\} \text{ imply } \vdash \{A\}\ M :_m \{B\}$$

for all appropriate A, B? Relative completeness means that in the presence of an oracle for the ambient theory of arithmetic, e.g. Peano arithmetic or ZFC set-theory (used with [CONSEQ]), the logic can syntactically derive all semantically true assertions, and reasoning about programs does not need to concern itself with models. Another way of saying this is that in relatively complete program logics, the expressive power of the ambient theory of arithmetic is the only source of incompleteness.[6]

The second question investigates if the program logic makes the same distinctions as the observational congruence. In other words, is the following characterisation true?

$$M \simeq N \text{ exactly when for all } A, B\text{: } \{A\}\ M :_m \{B\} \text{ iff } \{A\}\ N :_m \{B\}$$

Observational completeness means that the operational semantics (given by the contextual congruence) and the axiomatic semantics given by logic cohere with each other. We believe

[6] This does not violate Clarke's result [11] because our logic has higher-order features (evaluation formulae and code evaluation predicates). See [22] for a more extensive discussion.

that observational completeness is a key property of program logics because it guarantees that any operationally relevant program property can be expressed.

If a logic is observationally complete, we may ask the third question above about characteristic formulae: given a program M, can we find, by induction on the syntax of M, a pair of formulae A, B such that

– $\models \{A\}\ M :_m \{B\}$
– for all programs N: $M \lesssim N$ iff $\models \{A\}\ N :_m \{B\}$?

Such formulae are called *characteristic*. If characteristic formulae always exist, the semantics of each program can be expressed succinctly in the logic, using just a pair of formulae, and we call the logic *descriptively complete* [19]. The reason we use the contextual precongruence $\lesssim$ from Section 2 in the definition of characteristic formulae above, and not the congruence $\simeq$, is that our logic is for total correctness, and cannot express program divergence. More precisely, the following holds:

$$\left.\begin{array}{l} \models \{A\}\ M :_m \{B\} \\ M \lesssim N \end{array}\right\} \text{implies } \models \{A\}\ N :_m \{B\}.$$

In other words, if $\{A\}\ M :_m \{B\}$ holds and we make some parts of M more defined (e.g. by replacing a divergent with a convergent subterm), obtaining N, then $\{A\}\ N :_m \{B\}$ holds, too. Let's look at an example:

$$\models \{\mathsf{T}\}\ \lambda x.\Omega :_m \{\mathsf{T}\}.$$

If we replace Ω with 17, we obtain $\lambda x.17$, and clearly $\lambda x.\Omega \lesssim \lambda x.17$. But also:

$$\models \{\mathsf{T}\}\ \lambda x.17 :_m \{\mathsf{T}\}.$$

This indicates that in logics for total correctness, pairs A, B talk about upwards-closed sets of programs. Upwards-closed sets with a least member (up to $\simeq$) are especially nice, and for each such set, its least element can be seen as representing the set.

Proof strategy. We prove the three completeness theorems promised at the beginning of this section following ideas developed in [5, 19, 21, 38], but adapted to the present logic. The proofs are broken down into the following steps where we:

(1) make precise the relevant notion of characteristic formula.
(2) present an inference system for characteristic formulae.
(3) prove that the inference system computes characteristic formulae.
(4) show that the characteristic formulae are derivable using the rules and axioms of Section 3.
(5) use characteristic formulae to prove observational completeness.
(6) employ characteristic formulae to prove relative completeness.

5.1. **Formalising characteristic formulae.** We now precisely define we mean by characteristic formulae. Our definition is split into three parts, one guaranteeing the soundness of characteristic formulae, one to do with termination, and one that is about divergence-related aspects of program behaviour.

Definition 5.1. A pair (A, B) is a *total characteristic assertion pair*, or *TCAP*, *of* M *at* u, if the following conditions hold (in each clause we assume well-typedness).

– (soundness) $\models \{A\}\ M :_u \{B\}$.

$$\frac{x \text{ non-modal}}{\{\mathsf{T}\}\ x :_m \{x = m\}}\ \textsc{Var}^t \qquad \frac{x \text{ modal}}{\{x \Downarrow\}\ x :_m \{x = m\}}\ \textsc{Var}^t_m \qquad \frac{}{\{\mathsf{T}\}\ \mathsf{c} :_m \{\mathsf{c} = m\}}\ \textsc{Const}^t$$

$$\frac{\{A\}\ M :_m \{B\}}{\{\mathsf{T}\}\ \lambda x^\alpha.M :_u \{\forall x.(A \supset u \bullet x = m\{B\})\}}\ \textsc{Abs}^t$$

$$\frac{\{A_i\}\ M :_{m_i} \{B_i\} \quad i = 1, ..., n}{\{\bigwedge_i A_i\}\ \mathsf{op}(\tilde{M}) :_u \{\exists \tilde{m}.(u = \mathsf{op}(\tilde{m}) \wedge \bigwedge_i B_i)\}}\ \textsc{Op}^t$$

$$\frac{\{A_1\}\ M :_m \{B_1\} \quad \{A_2\}\ N :_n \{B_2\}}{\begin{array}{c}\{A_1 \wedge A_2 \wedge \forall mn.((B_1 \wedge B_2) \supset m \bullet n = z\{\mathsf{T}\})\} \\ MN :_u \\ \{\exists mn.m \bullet n = z\{B_1 \wedge B_2 \wedge z = u\}\}\end{array}}\ \textsc{App}^t$$

$$\frac{\{A\}\ M :_m \{B\} \quad \{A_i\}\ N :_u \{B_i\} \quad b_1 = \mathsf{t} \quad b_2 = \mathsf{f}}{\{A \wedge \bigwedge_i (B[b_i/m] \supset A_i)\}\ \mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2 :_u \{\bigvee_i (B[b_i/m] \wedge B_i)\}}\ \textsc{If}^t$$

$$\frac{\{\mathsf{T}\}\ \lambda x.M :_m \{A\}}{\{\mathsf{T}\}\ \mu g.\lambda x.M :_m \{A[m/g]\}}\ \textsc{Rec}^t \qquad \frac{\{A\}\ M :_m \{B\}}{\{\mathsf{T}\}\ \langle M \rangle :_u \{A \supset u = \langle m \rangle\{B\}\}}\ \textsc{Quote}^t$$

$$\frac{\{A_1\}\ M :_m \{B_1\} \quad \{A_2\}\ N :_u \{B_2\}}{\begin{array}{c}\{A_1 \wedge ((\forall x^\Box.A_2) \vee \forall m.(B_1 \supset m = \langle x \rangle\{A_2\}))\} \\ \mathtt{let}\ \langle x \rangle = M\ \mathtt{in}\ N :_u \\ \{\exists m x^\Box.((m = \langle \cdot \rangle \supset m = \langle x \rangle) \wedge B_1 \wedge B_2)\}\end{array}}\ \textsc{Unquote}^t$$

Figure 6: Inference system for TCAPs.

– (MTC, minimal terminating condition) For all appropriately typed models η, $M\eta \Downarrow$ if and only if $\eta \models A$.
– (closure) If $\eta \models \{E\}\ N :_u \{B\}$ and $E \supset A$, then $\eta \models E$ implies $M\eta \lesssim N\eta$.

A TCAP of M denotes a set of programs whose minimum element (up to $\simeq$) is M, and in that sense characterises that behaviour uniquely up to $\lesssim$. As mentioned above, characterisation up to $\simeq$ is not possible in a logic for total correctness. Logics of partial correctness suffer from a dual problem because they cannot express convergence. To achieve logical characterisation up to $\simeq$ in a single pair of formulae, we need both total and partial correctness, ideally combined into a logic of general correctness, see e.g. [5].

An inference system for TCAPs. The definition of TCAPs is semantic. We now present an algorithm that enables us to derive TCAPs for each $\textsc{Pcf}_{\textsc{dp}}$-program by induction on the typing derivation. The rules are given in Figure 6 and follow ideas from [5, 19, 21, 38]. Rulenames are derived from those in Figure 4 but with a superscript (e.g. [$\textsc{Var}^t$] instead of [Var]). The assertion language is that of Section 3 with one extension: quantification over modal variables. That means the assertions are now generated by the following extended grammar:

$$A \quad ::= \quad ... \mid \forall x^{\Box\alpha}.A$$

We call $\forall x^{\Box\alpha}.A$ *modal universal quantification*, where the bound variable x ranges over arbitrary programs, not just values. For $\forall x^{\Box\alpha}.A$ to be well-formed, x must be modal and of type α in A, and modal quantification is typed as follows:

$$\frac{\Gamma; \Delta, x : \alpha \vdash A}{\Gamma; \Delta \vdash \forall x^{\Box\alpha}.A}$$

The semantics of modal quantification is given by the following:

$$(\xi, \sigma) \models \forall x^{\Box\alpha}.A \text{ iff for all closed programs } M \text{ of type } \alpha : \ (\xi, \sigma \cdot x : M) \models A.$$

Since the addition of modal quantification does not change our notions of model and satisfaction relation, all proofs in Section 3 stay valid.

The existential modal quantifier $\exists x^{\Box\alpha}.A$ is given by de Morgan duality. We often drop type annotations in modal quantifiers, e.g. writing $\forall x^{\Box}.A$. Axiomatising modal quantification uses the standard axioms for first-order quantifiers with the following addition:

$$(div_m) \qquad \neg\forall x^{\Box}.x \Downarrow$$

This axiom states that not all modal variables denote terminating programs, which is immediately true from the model.

We write $\vdash^{\mathsf{tcap}} \{A\}\ M :_u \{B\}$ to indicate that the assertion $\{A\}\ M :_u \{B\}$ is derivable using the rules of Figure 6 only (i.e. without application of rules from Figures 4 and 5). As before, we assume that assertions, programs and rules are well-typed, and newly introduced variables are always fresh.

Before presenting proofs, we make a small observation: the pre- and postcondition pairs in Figure 6 constrain exactly the free variables of a program, together with the anchor:

Observation 5.2. Let $\vdash^{\mathsf{tcap}} \{A\}\ M :_m \{B\}$ then $\mathsf{fv}(A) = \mathsf{fv}(M)$ and $\mathsf{fv}(B) = \mathsf{fv}(M) \cup \{m\}$.

Informal explanation of the rules. Except for [UNQUOTE^t] and [VAR^t_m], all rules in Figure 6 are either unchanged from the corresponding rules in Figure 4 or have already been used in some of [5, 19, 21, 38]. We now give an informal explanation of the rules not already in Figure 4.

[VAR^t] says that the TCAP of a non-modal variable x at m is $(\mathsf{T}, x = m)$. The precondition is T because non-modal variables always denote values and thus always terminate. The postcondition $x = m$ says that whenever a program M satisfies $\{\mathsf{T}\}\ M :_m \{x = m\}$, then M must be contextually equal to x in the ambient model.

[VAR^t_m] for modal variables x has a more elaborate precondition than [VAR^t], because modal variables can denote non-terminating programs. The formula $x \Downarrow$ is true exactly when the denotation of x is terminating. The postcondition is the same as in the case of [VAR^t].

[CONST^t] says that the TCAP for constants c at m is $(\mathsf{T}, \mathsf{c} = m)$. As with non-modal variables, the precondition is T because constants are values. The postcondition $\mathsf{c} = m$ says that whenever a program M satisfies $\{\mathsf{T}\}\ M :_m \{\mathsf{c} = m\}$, then M must be contextually equal to $\mathtt{c}$. For example, under the typing $x : \mathsf{Int}; \epsilon$, the program $\mathtt{if}\ x\ \mathtt{then}\ 5\ \mathtt{else}\ 5$ has this property, and indeed $5 \simeq \mathtt{if}\ x\ \mathtt{then}\ 5\ \mathtt{else}\ 5$ when x is non-modal.

[Op^t] computes all TCAPs for operands in the premise. As an operation (e.g. addition) terminates exactly when all operands terminate, the precondition of the rule's conclusion is simply the conjunction of all preconditions for operands. The postcondition of the rule conclusion states that the result of the computation is the operation applied to some operands, and each operand is constrained by the postconditions of the rule premises. Depending on the operations used, additional constraints might be needed in the precondition: for example division M/N requires N to evaluate to a non-zero value.

[App^t] works as follows. In a call-by-value language an application MN terminates if: the evaluations of both M and N terminate to V and W, respectively; and, in addition, the application VW itself terminates. The first two requirements are stated by putting $A_1 \wedge A_2$ into the precondition of the conclusion on the rule. Here A_i is obtained recursively by computing the TCAPs of M and N, so e.g. A_1 holds exactly when M terminates. The additional assumption

$$\forall mn.((B_1 \wedge B_2) \supset m \bullet n = z\{\mathsf{T}\})$$

says that no matter what M and N evaluate to, the program terminates as long as m is as constrained by B_i, n is constrained by B_2, and the application $m \bullet n$ terminates. The postcondition of the conclusion says that the program MN evaluates to the result of applying M to N.

[If^t] makes the following assertion. A conditional terminates exactly when the condition terminates and the branch chosen by the conditional does, too. This is formalised by:

$$A \wedge \bigwedge\nolimits_i (B[b_i/m] \supset A_i)$$

As exactly one of $B[\mathsf{t}/m]$ and $B[\mathsf{f}/m]$ is true and exactly one is false, one implication is vacuously true, and the other requires the corresponding A_i to hold, giving the correct termination condition. For the same reason exactly one of

$$B[b_i/m] \wedge B_i$$

must be false, and one must hold exactly when the corresponding B_i holds. Since these two formulae are connected by an outer disjunction, the postcondition of the rule's conclusion give exactly the behaviour of the program.

[Unquote^t] This rule is the main intellectual novelty of the present section. Clearly, $\mathtt{let}\ \langle x\rangle = M\ \mathtt{in}\ N$ terminates exactly when:

– M evaluates to some $\langle M'\rangle$, and
– $N[M'/x]$ terminates.

The former is reflected in the precondition of the conclusion of the rule by adding A_1, which controls the termination of M. The second condition is more complicated, because it is possible that M evaluates to e.g. $\langle\Omega\rangle$, and yet $N[\Omega/x]$ terminates, for example in

$$\mathtt{let}\ \langle x\rangle = \langle\Omega\rangle\ \mathtt{in}\ \langle x\rangle$$

This case is covered by the clause $\forall x^{\Box}.A_2$. We see here the reason for using modal quantification. If the quantifier were to range over values only, programs such as

$$\mathtt{let}\ \langle x\rangle = \langle\Omega\rangle\ \mathtt{in}\ x, \tag{5.1}$$

which do not terminate, would cause trouble without modal quantification, because only when x is bound to a non-terminating term would N diverge. The TCAP for x at u is $(x \Downarrow, x = u)$, making $\forall x.x \Downarrow$, unlike $\forall x^{\Box}.x \Downarrow$, trivially true, leading to the erroneous precondition T for (5.1).

One may also ask, why not use a simpler precondition like

$$A_1 \wedge \forall m.(B_1 \supset m = \langle x \rangle \{A_2\}) \tag{5.2}$$

in the conclusion of [UNQUOTE^t]? The answer is that this would also be too weak for completeness. To see why, consider the program:

$$\texttt{let } \langle x \rangle = \langle \Omega \rangle \texttt{ in } 8.$$

The TCAPs of $\langle \Omega \rangle$ is $\{\mathsf{T}\}\ \langle \Omega \rangle :_m \{\mathsf{T}\}$, cf. Example 16, and using the rule [CONST^t], we see that $\{\mathsf{T}\}\ 8 :_u \{u = 8\}$ is the TCAP of 8. That means (5.2) gives us a precondition

$$\{\mathsf{T} \wedge \forall m.m = \langle x \rangle \{\mathsf{T}\}\}\ \texttt{let } \langle x \rangle = \langle \Omega \rangle \texttt{ in } 8 :_u \{...\}$$

which is equivalent to:

$$\{\mathsf{F}\}\ \texttt{let } \langle x \rangle = \langle \Omega \rangle \texttt{ in } 8 :_u \{...\} \tag{5.3}$$

since $\forall m.m = \langle x \rangle \{\mathsf{T}\}$ is equivalent to F. Now (5.3) is clearly sound, but the precondition too weak to capture the full meaning of the program $\texttt{let } \langle x \rangle = \langle \Omega \rangle \texttt{ in } 8$.

Next we look at the postcondition. It says that the result of evaluating $\texttt{let } \langle x \rangle = M \texttt{ in } N$ is, among other things, as described by B_2, which is the postcondition of N at u. However, by Observation 5.2, B_2 contains x as free variable. We hide it with a modal existential quantifier. But x cannot be arbitrary, as it is the result of unquoting what M evaluates to. Note that the postcondition B_1 speaks about M named m. So x is the unquoting of m. We cannot assert $m = \langle x \rangle \{...\}$, because that would stipulate that M evaluates to a term that, when unquoted, terminates, which cannot be guaranteed (e.g. if M is $\langle \Omega \rangle$). To deal with this issue, we explicitly require that x is the unquoting of m, provided m denotes a terminating meta-program:

$$m = \langle \cdot \rangle \supset m = \langle x \rangle \tag{5.4}$$

which means, if M converges to a quasi-quote $\langle M' \rangle$ and M' converges, say to V, then x describes this value V. Finally, we hide m by an existential quantifier, and constrain m by B_1. Note that the conditional constraining of x in (5.4) does not hold if M diverges, or converges to e.g. $\langle \Omega \rangle$. In the former case, the precondition must be (equivalent to) F, because the whole program diverges. In the latter case, a logic for total correctness cannot make an interesting assertion about the use of x in N.

Theorem 5.3.

(1) (descriptive completeness for total correctness) *Assume* $\Gamma; \Delta \vdash M : \alpha$. *Then* $\vdash^{\mathsf{tcap}} \{A\}\ M :_u \{B\}$ *implies* (A, B) *is a TCAP of* M *at* u.
(2) (observational completeness) $M \simeq N$ *if and only if, for each* A *and* B, *we have* $\models \{A\}\ M :_u \{B\}$ *iff* $\models \{A\}\ N :_u \{B\}$.
(3) (relative completeness) *Let* B *be upward-closed at* u. *Then* $\models \{A\}\ M :_u \{B\}$ *implies* $\vdash \{A\}\ M :_u \{B\}$.

Before giving a proof of Theorem 5.3 establish some helpful facts.

Proposition 5.4.

(1) *If* (A, B) *is a TCAP of* M *at* u *and if* $\models \{A\}\ N :_u \{B\}$, *then* $M \lesssim N$.
(2) (A, B) *is a TCAP of* M *at* u *iff (soundness), (MTC) and the following condition hold: (*closure-2*): if* $(\xi, \sigma) \models A$ *and for closed* V *we have* $(\xi \cdot u : V, \sigma) \models B$ *then* $M(\xi, \sigma) \lesssim V$.

Proof. We begin with (1). Assume that $\eta = (\xi, \sigma) \models \{A\}\ N :_u \{B\}$. There are two cases.

– $\eta \models A$. In this case $M\eta \Downarrow V$ by soundness, and $(\xi \cdot u : V, \sigma) \models B$. Now $M\eta \lesssim N\eta$ follows by (*closure*).
– $\eta \not\models A$. In this case, by (MTC) we have $M\eta \Uparrow$ and hence trivially $M\eta \lesssim N\eta$.

Now the result follows from Observation 3.1.

For (2) we begin with the (if) direction. Assume $\eta \models \{E\}\ N :_u \{B\}$ where $E \supset A$ and $\eta \models E$. Hence $N\eta \Downarrow V$ with $(\xi \cdot u : V, \sigma) \models B$ by soundness. From $E \supset A$ we get $\eta \models A$, but then by (*closure-2*) it must be the case that $M\eta \lesssim V$ which in turn implies $M\eta \lesssim N\eta$ since $V \simeq N\eta$ by Proposition 2.1. (6).

For the reverse direction, suppose (A, B) is a TCAP for M at u. We must show that (*closure-2*) holds. So let $\eta = (\xi, \sigma) \models A$ and $(\xi \cdot u : V, \sigma) \models B$, with V being closed and appropriately typed. Define:

$$E \stackrel{def}{=} A \wedge \exists u.B$$

Then clearly:

– $\eta \models E$,
– $E \supset A$,
– $\eta \models \{E\}\ V :_u \{B\}$.

Hence by (*closure*) $M\eta \lesssim V\eta = V$ □

Proposition 5.4.1 shows that TCAPs of a program M really represent a set of behaviours whose minimal element is M.

Proof [of Theorem 5.3.1]. The proof we are about to embark on is somewhat lengthy, and benefits from having the following convenient proposition available.

Proposition 5.5. *If* $\vdash^{\mathsf{tcap}} \{A\}\ M :_u \{B\}$ *then also* $\vdash \{A\}\ M :_u \{B\}$.

Proof. We proceed by induction on the derivation of $\vdash^{\mathsf{tcap}} \{A\}\ M :_u \{B\}$. The cases [Abs^t, Quote^t] follow immediate from the (IH), since these rules are identical in the rule systems of Figures 4 and 6.

Var^t: We proceed as follows.

1	$\{x = m[x/m] \wedge x \Downarrow\}\ x :_m \{x = m\}$	Var
2	$\{x \Downarrow\}\ x :_m \{x = m\}$	Conseq, *1*
3	$\{\mathsf{T}\}\ x :_m \{x = m\}$	*(term)*, Conseq, *2*

Var^t_m: This case is exactly like the previous, except that the last line is omitted.
Const^t: Similar to [Var^t].

OP^t: We treat the special case of addition.

1	$\{A_i\}\ M_i :_{m_i} \{B_i\}$	*(IH)*
2	$\{A_1 \wedge A_2\}\ M_1 :_{m_1} \{B_1 \wedge A_2\}$	INVAR, *1*
3	$\{B_1 \wedge A_2\}\ M_2 :_{m_2} \{B_1 \wedge B_2\}$	INVAR, *1*
4	$\{B_1 \wedge A_2\}\ M_2 :_{m_2} \{m_1 + m_2 = m_1 + m_2 \wedge B_1 \wedge B_2\}$	CONSEQ, *3*
5	$\{B_1 \wedge A_2\}\ M_2 :_{m_2} \{(u = m_1 + m_2)[m_1 + m_2/u] \wedge B_1 \wedge B_2\}$	*4*
6	$\{A_1 \wedge A_2\}\ M_1 + M_2 :_u \{u = m_1 + m_2 \wedge B_1 \wedge B_2\}$	ADD, *2, 5*
7	$\{A_1 \wedge A_2\}\ M_1 + M_2 :_u \{\exists m_1 m_2.(u = m_1 + m_2 \wedge B_1 \wedge B_2)\}$	CONSEQ, *6*

APP^t: The proof for this rule is the sole place in this paper where the (q_α) axiom is used. It is an open question as to whether this axiom is strictly needed, but we have not yet managed without it.

1	$\{A_1\}\ M :_m \{B_1\}$	*(IH)*
2	$C \stackrel{def}{=} \forall mn.((B_1 \wedge B_2) \supset m \bullet n = u\{\mathsf{T}\})$	
3	$\{A_1 \wedge A_2 \wedge C\}\ M :_m \{A_2 \wedge B_1 \wedge C\}$	INVAR, *1*
4	$\{A_2\}\ N :_n \{B_2\}$	*(IH)*
5	$\{A_2 \wedge B_1 \wedge C\}\ N :_n \{B_1 \wedge B_2 \wedge C\}$	INVAR, *4*
6	$\{A_2 \wedge B_1 \wedge C\}\ N :_n \{B_1 \wedge B_2 \wedge m \bullet n = u\{\mathsf{T}\}\}$	*5*
7	$\{A_2 \wedge B_1 \wedge C\}\ N :_n \{m \bullet n = u\{B_1 \wedge B_2\}\}$	*(q4), 6*
8	$\{A_2 \wedge B_1 \wedge C\}\ N :_n \{m \bullet n = u\{m \bullet n = z\{B_1 \wedge B_2 \wedge u = z\}\}\}$	*(q_α), 7*
9	$\{A_1 \wedge A_2 \wedge C\}\ MN :_u \{m \bullet n = z\{B_1 \wedge B_2 \wedge u = z\}\}$	APP, *3, 8*
10	$\{A_1 \wedge A_2 \wedge C\}\ MN :_u \{\exists mn.(m \bullet n = z\{B_1 \wedge B_2 \wedge u = z)\}\}$	*9*

IF^t: In the derivation of this rule we make unusually heavy tacit use of the [CONSEQ] rule.

1	$\{A\}\ M :_m \{B\}$	*(IH)*
2	$C \stackrel{def}{=} \bigwedge_i (B[b_i/m] \supset A_i)$	
3	$\{A \wedge C\}\ M :_m \{B \wedge C\}$	INVAR, *1*
4	$B[\mathsf{t}/m] \equiv \mathsf{T} \qquad B[\mathsf{f}/m] \equiv \mathsf{F}$	*Wlog.*
5	$\{A_i\}\ N_i :_u \{B_i\}$	*(IH)*
6	$\{B[b_i/m] \wedge C\}\ N_i :_u \{B_i\}$	*5*
7	$\{B[b_i/m] \wedge C[b_i/m]\}\ N_i :_u \{B_i\}$	$m \notin \mathsf{fv}(C)$, *6*
8	$\{B[b_i/m] \wedge B[b_i/m] \wedge C[b_i/m]\}\ N_i :_u \{B[b_i/m] \wedge B_i\}$	INVAR, *7*
9	$\{B[b_i/m] \wedge C[b_i/m]\}\ N_i :_u \{B[b_i/m] \wedge B_i\}$	*8*
10	$\{(B \wedge C)[b_i/m]\}\ N_i :_u \{B[b_i/m] \wedge B_i\}$	*9*
11	$D \stackrel{def}{=} \bigvee_i (B[b_i/m] \wedge B_i)$	
12	$\{(B \wedge C)[b_i/m]\}\ N_i :_u \{D\}$	*10*
13	$\{A \wedge C\}\ \mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2 :_u \{D\}$	*3, 12*

UNQUOTE^t: This is the last step in our proof. The derivation uses [UNQUOTE$^+$], the only use of that rule in the paper. It is unclear if the simpler version of [UNQUOTE$^+$] presented in Section 3 is strong enough to carry out this part of the proof.

1	$\{A_1\}\ M :_m \{B_1\}$	*(IH)*
2	$C \stackrel{def}{=} (\forall x^{\Box}.A_2) \vee \forall m.(B_1 \supset m = \langle x \rangle \{A_2\}$	
3	$\{A_1 \wedge C\}\ M :_m \{B_1 \wedge C\}$	INVAR, *1*
4	$\{A_1 \wedge C\}\ M :_m \{B_1 \wedge ((\forall x^{\Box}.A_2) \vee m = \langle x \rangle \{A_2\})\}$	*3*
5	$\{A_1 \wedge C\}\ M :_m \{B_1 \wedge ((\neg\forall x^{\Box}.A_2) \supset m = \langle x \rangle \{A_2\})\}$	*4*
6	$D \stackrel{def}{=} m = \langle \cdot \rangle \supset m = \langle x \rangle$	
7	$\{A_2\}\ N :_u \{B_2\}$	*(IH)*
8	$\{A_2 \wedge D \wedge B_1\}\ N :_u \{D \wedge B_1 \wedge B_2\}$	INVAR, *7*
9	$((\neg\forall x^{\Box}.A_2) \supset A_2) \ \supset \ A_2$	*see below*
10	$\{B_1 \wedge ((\neg\forall x^{\Box}.A_2) \supset A_2) \wedge D\}\ N :_u \{D \wedge B_1 \wedge B_2\}$	CONSEQ, *8, 9*
11	$\{A_1 \wedge C\}\ \mathtt{let}\ \langle x \rangle = M\ \mathtt{in}\ N :_u \{D \wedge B_1 \wedge B_2\}$	UNQUOTE$^+$, *5, 10*
12	$\{A_1 \wedge C\}\ \mathtt{let}\ \langle x \rangle = M\ \mathtt{in}\ N :_u \{\exists m x^{\Box}.(D \wedge B_1 \wedge B_2)\}$	CONSEQ, *11*

It remains to justify Line 9. Rewriting

$$((\neg\forall x^{\Box}.A_2) \supset A_2) \ \supset \ A_2$$

in the equivalent form

$$((\forall x^{\Box}.A_2) \vee A_2) \ \supset \ A_2$$

lets us see immediately that Line 9 is true. □

Proposition 5.5 together with the soundness of the rules in Figure 4 immediately implies the soundness of the TCAP rules. We record this fact:

Corollary 5.6. *The TCAP rules in Figure 6 are sound.*

Now we establish the first part of the theorem by induction on the derivation of $\vdash^{\mathsf{tcap}} \{A\}\ M :_u \{B\}$, using (*closure-2*) from Proposition 5.4.2 instead of (*closure*) for simplicity. We focus on the interesting cases [$\textsc{Var}^t_m$, $\textsc{Rec}^t$, $\textsc{Quote}^t$, $\textsc{Unquote}^t$], leaving the remaining ones to Appendix B.

We start the proof of Theorem 5.3.1 with [$\textsc{Var}^t_m$]. For soundness, assume that $\eta \stackrel{def}{=} (\xi, \sigma \cdot x : M) \models x \Downarrow$. By definition of $x \Downarrow$ we can find a value V of appropriate type such that $(\xi, y : V, \sigma \cdot x : M) \models x = y$ where y is some fresh variable. Thus $M \simeq V$, hence $x\eta \Downarrow W$ for some value W with $W \simeq V$ (Proposition 2.1. (1)) and clearly $(\xi \cdot m : W, \sigma) \models x = m$. (MTC) follows by the assumption in the precondition that $x \Downarrow$. For (*closure-2*), we choose a model $\eta \stackrel{def}{=} (\xi, \sigma \cdot x : M)$ with $\eta \models x \Downarrow$ and $(\xi \cdot m : V, \sigma \cdot x : M) \models m = x$. As above, $\eta \models x \Downarrow$ means that $M \Downarrow W$ for some appropriate closed value W. Hence $V \simeq M \Downarrow W$ which means in particular $M \lesssim V$ hence $x\eta \lesssim V$ as required.

Next is [$\textsc{Quote}^t$]. Let $\eta \stackrel{def}{=} (\xi \cdot u : \langle N\rangle, \sigma)$. Soundness has already been proven in Theorem 3.2.1, and (MTC) is trivial. So suppose $\eta \models A \supset u = \langle m\rangle\{B\}$. There are two cases.

- $\eta \not\models A$. By (IH) we know that A is an MTC for M, hence it must be the case that $M\eta \Uparrow$, thus trivially $M\eta \lesssim N$. Since $\lesssim$ is a congruence by definition, we know that $\langle M\eta\rangle \lesssim \langle N\rangle$. Now $\langle M\rangle\eta = \langle M\eta\rangle$ hence $\langle M\rangle\eta \lesssim \langle N\rangle$ as required.
- $\eta \models A$ and $\eta \models u = \langle m\rangle\{B\}$. Now we reason as given next.

1	$\eta \models A$	*Assumption*
2	$(\xi, \sigma) \models A$	*Prop. 3.3.2,* $u \notin \mathsf{fv}(A)$*, 1*
3	$\eta \models u = \langle m\rangle\{B\}$	*Assumption*
4	$N \Downarrow V$ and $(\xi \cdot u : \langle N\rangle, \sigma \cdot m : V) \models B$	*3*
5	$\xi, \sigma \cdot m : V \models B$	*Prop. 3.3.2,* $u \notin \mathsf{fv}(B)$*, 4*
6	$M(\xi, \sigma) \lesssim V$	*by (IH), (*closure-2*), 2, 5*
7	$\langle M\rangle(\xi, \sigma) = \langle M(\xi, \sigma)\rangle \lesssim \langle V\rangle$	$\lesssim$ *is a congruence, 6*
8	$\langle M\rangle(\xi, \sigma) \lesssim \langle N\rangle$	*Lem. 2.1. (7), 4, 7*

Next we deal with [$\textsc{Unquote}^t$], the most complicated case. We begin with soundness. Let A be the formula

$$A_1 \wedge (\forall x^{\Box}.A_2) \vee \forall m.(B_1 \supset m = \langle x\rangle\{A_2\})$$

Assume that $(\xi, \sigma) \models A$.

1	$(\xi, \sigma) \models A$	*Assumption*
2	$(\xi, \sigma) \models A_1$	
3	$M(\xi, \sigma) \Downarrow \langle M' \rangle$ and $(\xi \cdot m : \langle M' \rangle, \sigma) \models B_1$	*(IH), 2*

Now we have two cases, here is the first.

4	$(\xi, \sigma) \models \forall x^{\Box}.A_2$	*First case*
5	For all appropriate programs $U : (\xi, \sigma \cdot x : U) \models A_2$	*4*
6	$(\xi, \sigma \cdot x : M') \models A_2$	*Specialisation of 5*
7	$N(\xi, \sigma \cdot x : M') \Downarrow V$ and $(\xi \cdot u : V, \sigma \cdot x : M') \models B_2$	*(IH), 6*

We now consider reductions where we set $\eta \stackrel{def}{=} (\xi, \sigma)$.

$$
\begin{array}{lcll}
(\texttt{let}\ \langle x \rangle = M\ \texttt{in}\ N)\eta & = & \texttt{let}\ \langle x \rangle = M\eta\ \texttt{in}\ N\eta & \\
& \twoheadrightarrow & \texttt{let}\ \langle x \rangle = \langle M' \rangle\ \texttt{in}\ N\eta & \textit{by } 3 \\
& \twoheadrightarrow & N\eta[M'/x] & \\
& = & N(\xi, \sigma \cdot x : M') & \textit{Prop. } 3.3.4 \\
& \Downarrow & V & \textit{by } 7
\end{array}
$$

Using this fact, we continue to reason as follows. Define

$$\eta'' \stackrel{def}{=} (\xi \cdot u : V \cdot m : \langle M' \rangle, \sigma \cdot x : M')$$

We need to show that

$$\eta'' \models (m = \langle \cdot \rangle \supset m = \langle x \rangle) \wedge B_1 \wedge B_2. \tag{5.5}$$

Since by (3) we have $(\xi \cdot m : \langle M' \rangle, \sigma) \models B_1$ and $x, u \notin \mathsf{fv}(B_1)$ by Observation 5.2, we can apply Proposition 3.3.3 to get $\eta'' \models B_1$. By similar reasoning we get $\eta'' \models B_2$ from (8). Hence $\eta'' \models B_1 \wedge B_2$. That leaves the implication. Assume $\eta'' \models m = \langle \cdot \rangle$. By definition that means there is a value W such that $M' \Downarrow W$ and

$$(\xi \cdot u : V \cdot m : \langle M' \rangle, \sigma \cdot x : W) \models m = \langle x \rangle$$

By Proposition 2.1. (6) then $M' \simeq W$, hence we can apply Proposition 3.3.2, giving us the required (5.5), which in turn implies

$$(\xi \cdot u : V, \sigma) \models \exists m x^{\Box}.((m = \langle \cdot \rangle \supset m = \langle x \rangle) \wedge B_1 \wedge B_2)$$

which finishes the soundness proof for this case.

We now consider the second case.

4	$(\xi,\sigma) \models \forall m.(B_1 \supset m = \langle x\rangle\{A_2\})$	*Second case*
5	For all appropriate programs $L.(\xi \cdot m : L, \sigma) \models B_1 \supset m = \langle x\rangle\{A_2\}$	*4*
6	$(\xi \cdot m : \langle M'\rangle, \sigma) \models B_1 \supset m = \langle x\rangle\{A_2\}$	*Specialisation of 5*
7	$(\xi \cdot m : \langle M'\rangle, \sigma) \models m = \langle x\rangle\{A_2\}$	*3, 6*
8	$M' \Downarrow W$ and $(\xi \cdot m : \langle M'\rangle, \sigma \cdot x : W) \models A_2$	*(IH), 7*
9	$(\xi, \sigma \cdot x : W) \models A_2$	*Obs. 5.2, $m \notin \mathsf{fv}(A_2)$, Prop. 3.3.2, 8*
10	$N(\xi, \sigma \cdot x : W) \Downarrow V$ and $(\xi \cdot u : V, \sigma \cdot x : W) \models B_2$	*(IH), 9*

Now we consider reductions where we set $\eta \stackrel{def}{=} (\xi, \sigma)$.

$$
\begin{array}{lcll}
(\texttt{let}\ \langle x\rangle = M\ \texttt{in}\ N)\eta & = & \texttt{let}\ \langle x\rangle = M\eta\ \texttt{in}\ N\eta & \\
& \twoheadrightarrow & \texttt{let}\ \langle x\rangle = \langle M'\rangle\ \texttt{in}\ N\eta & \textit{by } 3 \\
& \twoheadrightarrow & N\eta[M'/x] & \\
& = & N(\xi, \sigma \cdot x : M') & \textit{Prop. } 3.3.4 \\
& \Downarrow & V & \textit{by } 10
\end{array}
$$

The rest of this case is essentially identical to the corresponding reasoning for the first case, and omitted.

Now we establish (MTC). Choose a model (ξ, σ) and assume that

$$(\texttt{let}\ \langle x\rangle = M\ \texttt{in}\ N)(\xi,\sigma) \Downarrow .$$

We will show that $(\xi, \sigma) \models A$. The reverse implication is part of soundness. Notice that this assumption implies the existence of a reduction sequence as follows.

$$
\begin{array}{lcll}
(\texttt{let}\ \langle x\rangle = M\ \texttt{in}\ N)(\xi,\sigma) & = & \texttt{let}\ \langle x\rangle = M(\xi,\sigma)\ \texttt{in}\ N(\xi,\sigma) & \\
& \twoheadrightarrow & \texttt{let}\ \langle x\rangle = \langle M'\rangle\ \texttt{in}\ N(\xi,\sigma) & (5.6) \\
& \to & N(\xi,\sigma)[M'/x] & \\
& = & N(\xi, \sigma \cdot x : M') & (5.7) \\
& \Downarrow & V & (5.8)
\end{array}
$$

by (5.6) we know that

$$M(\xi,\sigma) \Downarrow \langle M'\rangle$$

Since by (IH) A_1 is an MTC for M, we know that

$$(\xi,\sigma) \models A_1 \quad \text{and hence} \quad (\xi \cdot m : \langle M'\rangle, \sigma) \models B_1 \tag{5.9}$$

We have two cases. First assume that $M' \Uparrow$. By (5.8) and Lemma 2.1. (10) we know that the following holds.

For all appropriately typed and closed programs $L :\ N(\xi, \sigma \cdot x : L) \Downarrow .$

By (IH) A_2 is an MTC for N at u, so we can reason as follows.

1	for all $L.(\xi, \sigma \cdot x : L) \models A_2$	
2	$(\xi,\sigma) \models \forall x^{\Box}.A_2$	*1*
3	$(\xi,\sigma) \models A_1 \wedge ((\forall x^{\Box}.A_2) \wedge \forall m.(B_1 \supset m = \langle x\rangle\{A_1\}))$	*2, (5.9)*

The second case is that $M' \Downarrow$. We proceed as follows.

1	$(\xi \cdot m : L, \sigma) \models B_1$	*Fresh assumption,* $L = \langle L' \rangle$ *arbitrary value*
2	$(\xi, \sigma) \models A_1$	*5.9*
3	$M(\xi, \sigma) \lesssim \langle L' \rangle$	*By (IH) (closure-2) holds for* A_1, B_1*, 1, 2*
4	$M(\xi, \sigma) \Downarrow \langle M' \rangle$	*5.6*
5	$\langle M' \rangle \lesssim \langle L' \rangle$	*Lem. 2.1. (11), 3, 4*
6	$M' \lesssim L$	*Lem. 2.1. (8), 5*
7	$N(\xi, \sigma \cdot x : M') \Downarrow$	*5.8*
8	$N(\xi, \sigma \cdot x : L') \Downarrow$	*Lem. 2.1. (11), 6, 7*
9	$(\xi, \sigma \cdot x : L') \models A_2$	*By (IH)* A_2 *is (MTC) for* N*, 8*
10	$(\xi \cdot m : L, \sigma \cdot x : L') \models A_2$	$m \notin \mathsf{fv}(A_2)$*, Prop. 3.3.3, 9*
11	$(\xi \cdot m : L, \sigma) \models m = \langle x \rangle \{A_2\}$	*10*
12	$(\xi \cdot m : L, \sigma) \models B_1 \supset m = \langle x \rangle \{A_2\}$	*1, 11*
13	$(\xi, \sigma) \models \forall m.(B_1 \supset m = \langle x \rangle \{A_2\})$	*L was arbitrary, 12*
14	$(\xi, \sigma) \models A_1 \wedge ((\forall x^{\Box}.A_2) \wedge \forall m.(B_1 \supset m = \langle x \rangle \{A_1\}))$	*2, 13*

This establishes (MTC).

We conclude this case by proving (*closure-2*). Let $\eta \stackrel{def}{=} (\xi, \sigma)$ be an appropriately typed model such that:

– $\eta \models A_1 \wedge ((\forall x^{\Box}.A_2) \vee \forall m.(B_1 \supset m = \langle x \rangle \{A_2\}))$.
– $(\xi \cdot u : V, \sigma) \models \exists m x^{\Box}.((m = \langle \cdot \rangle \supset m = \langle x \rangle) \wedge B_1 \wedge B_2)$.

This means in particular that there are M_m and M_x such that

$$(\xi \cdot u : V \cdot m : \langle M_m \rangle, \sigma \cdot x : M_x) \models (m = \langle \cdot \rangle \supset m = \langle x \rangle) \wedge B_1 \wedge B_2. \tag{5.10}$$

We first note that since

$$(\xi \cdot u : V \cdot m : \langle M_m \rangle, \sigma \cdot x : M_x) \models (m = \langle \cdot \rangle \supset m = \langle x \rangle)$$

it must be the case that:

$$M_m \Downarrow \text{ implies } M_m \simeq M_x.$$

This is an immediate consequence of the definition of the satisfaction relation for $m = \langle \cdot \rangle$ and $m = \langle x \rangle$. At the same time, trivially:

$$M_m \Uparrow \text{ implies } M_m \lesssim M_x.$$

Taking those two facts together, we see that the following holds.

$$M_m \lesssim M_x. \tag{5.11}$$

Since $u, x \notin \mathsf{fv}(B_1)$, we can use (5.10) and Proposition 3.3.2 to conclude that

$$\eta \models A_1 \qquad (\xi \cdot m : \langle M_m \rangle, \sigma) \models B_1 \tag{5.12}$$

which, in turn, enables us to use the (IH), so by (*closure-2*) we know that

$$M\eta \lesssim \langle M_m \rangle. \tag{5.13}$$

In a similar way we establish that

$$(\xi \cdot u : V, \sigma \cdot x : M_x) \models B_2 \tag{5.14}$$

Now we have to distinguish the following two cases.
– $\eta \models \forall x^{\Box}.A_2$.
– $\eta \models \forall m.(B_1 \supset m = \langle x \rangle\{A_2\})$.
In the first case clearly

$$\eta' \stackrel{def}{=} (\xi, \sigma \cdot x : M_x) \models A_2$$

which together with (5.14) means we can use the (IH) on $\vdash^{\mathsf{tcap}} \{A_2\}\ N :_u \{B_2\}$, where (*closure-2*) means that

$$N\eta' \lesssim V. \tag{5.15}$$

This together with (5.13) means

$$\begin{array}{lcll} (\mathtt{let}\ \langle x \rangle = M\ \mathtt{in}\ N)\eta & = & \mathtt{let}\ \langle x \rangle = M\eta\ \mathtt{in}\ N\eta & \\ & \lesssim & \mathtt{let}\ \langle x \rangle = \langle M_m \rangle\ \mathtt{in}\ N\eta & \textit{by } (5.13) \\ & \to & N\eta[M_m/x] & \\ & \lesssim & N\eta[M_x/x] & \textit{by } (5.11) \\ & = & N\eta' & \textit{Prop. } 3.3.4 \\ & \lesssim & V & \textit{by } 5.15 \end{array}$$

Since $\to \subseteq \simeq \subseteq \lesssim$ (Theorem 2.1. (6)), and $\lesssim$ is transitive, we can thus conclude to:

$$(\mathtt{let}\ \langle x \rangle = M\ \mathtt{in}\ N)\eta \ \lesssim\ V$$

as required.

Now we consider the second case $\eta \models \forall m.(B_1 \supset m = \langle x \rangle\{A_2\})$.

1	$\eta \models \forall m.(B_1 \supset m = \langle x \rangle\{A_2\})$	
2	$(\xi \cdot m : \langle M_m \rangle, \sigma) \models B_1 \supset m = \langle x \rangle\{A_2\}$	*1*
3	$(\xi \cdot m : \langle M_m \rangle, \sigma) \models m = \langle x \rangle\{A_2\}$	*(5.12), 2*

4	$\langle M_m \rangle \Downarrow W, \quad (\xi \cdot m : \langle M_m \rangle, \sigma \cdot x : W) \models A_2$	*3*
5	$\eta' \stackrel{def}{=} (\xi, \sigma \cdot x : W) \models A_2$	*Prop. 3.3.2, $m \notin \mathsf{fv}(A_2)$, 4*

The rest of this case is handled exactly like the previous case, concluding the proof of (*closure-2*).

5.2. **Proofs of Theorems 5.3.2 and 5.3.3.** We conclude this section by proving observational and relative completeness.

Proof [of Theorem 5.3.2]. Assume that $M \simeq N$. Now let $\eta \stackrel{def}{=} (\xi, \sigma)$, $\eta \models A$, $M\eta \Downarrow V$ and $(\xi \cdot u : V, \sigma) \models B$. Since $M \simeq N$ we know that $M\eta \simeq N\eta$ by Observation 3.1, $N\eta \Downarrow W$ and $V \simeq W$. Hence we can apply Proposition 3.3.2 to obtain $(\xi \cdot u : W, \sigma) \models B$ as required. The remaining case, $\eta \not\models A$, is immediate.

For the reverse direction, let $\vdash^{\mathsf{tcap}} \{A\}\ M :_u \{B\}$. Then (A, B) is a TCAP of M at u, hence by soundness of TCAPs and Theorem 5.3.1 we know that $\models \{A\}\ M :_u \{B\}$. Then by assumption also $\models \{A\}\ N :_u \{B\}$. Since (A, B) is a TCAP for M at u we apply Proposition 5.4.1 to obtain $M \lesssim N$. Similarly we derive $N \lesssim M$, which together implies $M \simeq N$. This establishes observational completeness. □

Proof [of Theorem 5.3.3]. Relative completeness is equally easy to justify. We start from the following assumption.

$$\models \{A\}\ M :_u \{B\}$$

Using the rules in Figure 6, we obtain a TCAP (A', B') for M at u such that

$$\vdash^{\mathsf{tcap}} \{A'\}\ M :_u \{B'\}$$

holds. With these assumptions, the proof of Theorem 5.3.3 has the following form:

$$\dfrac{\dfrac{\models \{A\}\ M :_u \{B\} \quad \vdash^{\mathsf{tcap}} \{A'\}\ M :_u \{B'\}}{A \supset (A' \wedge (B' \supset B))}\ (*) \quad \dfrac{\vdash^{\mathsf{tcap}} \{A'\}\ M :_u \{B'\}}{\vdash \{A'\}\ M :_u \{B'\}}\ \textsc{Prop. 5.5}}{\vdash \{A\}\ M :_u \{B\}}\ \textsc{Conseq-Kl}$$

It remains to establish step (*). For this purpose, let $\eta \stackrel{def}{=} (\xi \cdot u : W, \sigma)$ be a model and assume $\eta \models A$.

We first establish that $\eta \models A'$. Since $u \notin \mathsf{fv}(A)$, we know from Proposition 3.3.2 that with $\eta' \stackrel{def}{=} (\xi, \sigma)$ also $\eta' \models A$. This fact together with the assumption $\models \{A\}\ M :_u \{B\}$ means that $M\eta' \Downarrow V$ for some closed value V. As (A', B') is a TCAP for M at u, (MTC) holds so it must also be the case that $\eta' \models A'$. Applying $u \notin \mathsf{fv}(A)$ with Proposition 3.3.2 again we now obtain $\eta \models A'$ as required. This shows that $A \supset A'$.

To prove that $(A \wedge B') \supset B$, assume $\eta \models A \wedge B'$. From $\eta \models B'$ and Proposition 5.4.2 we obtain $M\eta' \lesssim W$. We showed above that $M\eta' \Downarrow V$, so in fact $M\eta' \simeq V$ by Proposition 2.1. (6), hence clearly

$$V \lesssim W.$$

From $\eta \models A$, $M\eta' \Downarrow V$ (see above) and the assumption that $\models \{A\}\ M :_u \{B\}$ we obtain

$$(\xi \cdot u : V, \sigma) \models B.$$

Now we use the upwards-closure of B to conclude that:

$$\eta = (\xi \cdot u : W, \sigma) \models B.$$

6. Examples of characteristic formulae

In this section we look at some example inferences for TCAPs. To make the derivations more readable, we will make simplifications such as writing T for $\mathsf{T} \wedge \mathsf{T}$. Note that these simplifications are not admissible using the inference system of Figure 6 only. □

Example 6.1. We begin with a simple program $2 + 3$.

1	$\{\mathsf{T}\}\ 2 :_a \{a = 2\}$	$\textsc{Const}^t$
2	$\{\mathsf{T}\}\ 3 :_b \{b = 3\}$	$\textsc{Const}^t$
3	$\{\mathsf{T} \wedge \mathsf{T}\}\ 2 + 3 :_c \{\exists ab.(c = a + b \wedge a = 2 \wedge b = 3)\}$	$\textsc{Plus}^t$, *1, 2*

Clearly the conclusion in Line (3) is logically equivalent to

$$\{\mathsf{T}\}\ 2 + 3 :_c \{c = 5\}$$

as expected.

Example 6.2. We continue with a variant of Example 8, using a non-modal variable x. This example is preparation, of sorts, for more involved examples.

1	$\{\mathsf{T}\}\ x :_a \{a = x\}$	$\textsc{Var}^t$
2	$\{\mathsf{T}\}\ 1 :_b \{b = 1\}$	$\textsc{Const}^t$
3	$\{\mathsf{T} \wedge \mathsf{T}\}\ x + 1 :_c \{\exists ab.(c = a + b \wedge a = x \wedge b = 1)\}$	$\textsc{Plus}^t$, *1, 2*

As expected, the conclusion in Line (3) is logically equivalent to

$$\{\mathsf{T}\}\ x + 1 :_c \{c = x + 1\}$$

Example 6.3. We use the previous example to derive the TCAP for a abstraction $\lambda x.x+1$.

1	$\{\mathsf{T}\}\ x + 1 :_c \{c = x + 1\}$	*Ex. 9*
2	$\{\mathsf{T}\}\ \lambda x.x + 1 :_u \{\forall x.(\mathsf{T} \supset u \bullet x = x + 1)\}$	$\textsc{Abs}^t$, *1*

As before, the derived TCAP is easily seen to be logically equivalent to

$$\{\mathsf{T}\}\ \lambda x.x + 1 :_u \{\forall x.u \bullet x = x + 1\}$$

Example 6.4. This example shows how the rule for application works.

1	$\{\mathsf{T}\}\ \lambda x.x + 1 :_m \{\forall x.m \bullet x = x + 1\}$	*Ex. 10*
2	$\{\mathsf{T}\}\ 2 :_n \{n = 2\}$	$\textsc{Const}^t$
3	$\{\mathsf{T} \wedge \mathsf{T} \wedge \forall mn.((\forall x.m \bullet x = x + 1 \wedge n = 2) \supset m \bullet n \Downarrow)\}$ $(\lambda x.x + 1)2 :_u$ $\{\exists mn.(m \bullet n = z\{\forall x.m \bullet x = x + 1 \wedge n = 2 \wedge z = u\})\}$	$\textsc{App}^t$, *1, 2*

It is easy to see that

$$\mathsf{T} \wedge \mathsf{T} \wedge \forall mn.((\forall x.m \bullet x = x + 1 \wedge n = 2) \supset m \bullet n \Downarrow)$$

simplifies to T via $\forall mn.(m \bullet n \Downarrow \supset\ m \bullet n \Downarrow)$. The postcondition can be simplified to $u = 3$ as follows:

1	$\exists mn.(m \bullet n = z\{\forall x.m \bullet x = x + 1 \wedge n = 2 \wedge z = u\})$	
2	$\exists mn.(\forall x.m \bullet x = x + 1 \wedge n = 2 \wedge m \bullet n = z\{z = u\})$	*1*
3	$\exists mn.(m \bullet 2 = 3 \wedge m \bullet 2 = z\{z = u\})$	*2*
4	$\exists mn.(m \bullet 2 = z\{z = 3\} \wedge m \bullet 2 = z\{z = u\})$	*Unwinding of shorthand, 3*
5	$\exists mn.(m \bullet 2 = z\{z = 3 \wedge z = u\})$	*Axiom (e1) from Fig. 7, 4*
6	$\exists mn.(m \bullet 2 = z\{u = 3\})$	*5*
7	$\exists mn.(u = 3 \wedge m \bullet 2 = z\{\mathsf{T}\})$	*Axiom (e4) from Fig. 7, 6*
8	$u = 3$	*7*

Example 6.5. The TCAP in the previous example turned out to be logically equivalent to a very simple assertion, albeit only after simplification starting with rather large formulae. This simplification was possible because both parts of the application were concrete terms. In an application like gx this is not the case as we show now, even when neither g nor x are modal.

1	$\{\mathsf{T}\}\ g :_a \{a = g\}$	$\textsc{Var}^t$
2	$\{\mathsf{T}\}\ x :_b \{b = x\}$	$\textsc{Var}^t$
3	$\{\mathsf{T} \wedge \mathsf{T} \wedge \forall ab.((a = g \wedge b = x) \supset a \bullet b \Downarrow)\}$ $gx\ :_m$ $\{\exists ab.(a \bullet b = z\{a = g \wedge b = x \wedge z = m\})\}$	$\textsc{App}^t$, *1, 2*
4	$\{g \bullet x \Downarrow\}\ gx :_m \{g \bullet x = m\}$	*3*

Here the (simplified) TCAP explicitly assumes that the application converges, and states that the result of the program is simply the result of the application.

Example 6.6. We use the previous example to derive the TCAP for ω. Our preceding discussion indicated that $\{\mathsf{T}\}\ \omega :_u \{\mathsf{T}\}$ is the strongest assertion we can make about a program such as ω in a logic for total correctness. This is borne out by the derivation to follow.

1	$\{g \bullet x \Downarrow\}\ gx :_m \{g \bullet x = m\}$	*Ex. 11*
2	$\{\mathsf{T}\}\ \lambda x.gx :_u \{\forall x.(g \bullet x \Downarrow \supset\ u \bullet x = m\{g \bullet x = m\})\}$	$\textsc{Abs}^t$, *1*
3	$\{\mathsf{T}\}\ \omega :_u \{\forall x.(u \bullet x \Downarrow \supset\ u \bullet x = m\{u \bullet x = m\})\}$	$\textsc{Rec}^t$, *2*
4	$\{\mathsf{T}\}\ \omega :_u \{\forall x.(u \bullet x \Downarrow \supset\ u \bullet x = u \bullet x)\}$	*3*
5	$\{\mathsf{T}\}\ \omega :_u \{\mathsf{T}\}$	*4*

Example 6.7. We build on Example 13 to derive the TCAP for Ω. As Ω diverges, the precondition of the TCAP must be falsity.

1	$\{\mathsf{T}\}\ \omega :_m \{\mathsf{T}\}$	*Ex. 12*
2	$\{\mathsf{T}\}\ () :_n \{n = ()\}$	$\textsc{Const}^t$
3	$\{\forall m.m \bullet () \Downarrow\}\ \Omega :_u \{\exists m.m \bullet () = u\}$	$\textsc{App}^t$, *3*

Clearly $\forall m.m \bullet () \Downarrow$ is false, because not every function is terminating. This is intuitively obvious, and follows formally from the axiom (*div*) in Figure 7. Consequently, the TCAP for Ω is logically equivalent to

$$\{\mathsf{F}\}\ \Omega :_u \{\mathsf{T}\}$$

as expected.

Example 6.8. We will now look at examples involving MP.

1	$\{\mathsf{T}\}\ 3 :_m \{m = 3\}$	$\textsc{Const}^t$
2	$\{\mathsf{T}\}\ \langle 3\rangle :_u \{\mathsf{T} \supset u = \langle m\rangle\{m = 3\}\}$	$\textsc{Quote}^t$, *1*
3	$\{\mathsf{T}\}\ \langle 3\rangle :_u \{u = \langle 3\rangle\}$	*2*

The result is not surprising because [$\textsc{Quote}^t$] is unchanged from Figure 4.

Example 6.9. Next we tackle an example that uses [$\textsc{Unquote}^t$]. Clearly the program $\texttt{let}\ \langle x\rangle = \langle 3\rangle\ \texttt{in}\ x$ evaluates to 3. Note that x is modal.

1	$\{\mathsf{T}\}\ \langle 3\rangle :_m \{m = \langle 3\rangle\}$	*Ex. 14*
2	$\{x \Downarrow\}\ x :_n \{n = x\}$	$\textsc{Var}^t_m$
3	$\{\mathsf{T} \wedge ((\forall x^{\Box}.x \Downarrow) \vee \forall m.(m = \langle 3\rangle \supset m = \langle x\rangle\{x \Downarrow\}))\}$ $\texttt{let}\ \langle x\rangle = \langle 3\rangle\ \texttt{in}\ x\ :_n$ $\{\exists m x^{\Box}.((m = \langle\cdot\rangle \supset m = \langle x\rangle) \wedge m = \langle 3\rangle \wedge n = x)\}$	$\textsc{Unquote}^t$, *1, 2*
4	$\{\mathsf{T}\}$ $\texttt{let}\ \langle x\rangle = \langle 3\rangle\ \texttt{in}\ x\ :_n$ $\{\exists m x^{\Box}.((m = \langle\cdot\rangle \supset m = \langle x\rangle) \wedge m = \langle 3\rangle \wedge n = x)\}$	*3*
5	$\{\mathsf{T}\}\ \texttt{let}\ \langle x\rangle = \langle 3\rangle\ \texttt{in}\ x :_n \{n = 3\}$	*4*

We now explain the last two simplification steps. First $(\forall x^{\Box}.x \Downarrow)$ must be equivalent to F because not all denotations of the modal variable x terminate. This is formalised by Axiom (div_m) from Section 5. The formula $\forall m.(m = \langle 3\rangle \supset m = \langle x\rangle\{x \Downarrow\}$ is equivalent to T because $m = \langle 3\rangle$ is a shorthand for $m = \langle x\rangle\{x = 3\}$, and clearly $x = 3$ implies $x \Downarrow$. This justifies Line 4. Regarding the last line, clearly $m = \langle 3\rangle$ implies $m = \langle\cdot\rangle$, so $\exists m x^{\Box}.(m = \langle x\rangle \wedge m = \langle 3\rangle \wedge n = x)$ holds. Using Axioms ($q1$) and ($q4$) allows us to obtain $\exists m x^{\Box}.(x = 3 \wedge n = x)$, which in turn simplifies to $n = 3$ as required.

Example 6.10. We now derive a simple result from Example 14 that is used later.

1	$\{\mathsf{F}\}\ \Omega :_x \{\mathsf{T}\}$	*Ex. 13*
2	$\{\mathsf{T}\}\ \langle\Omega\rangle :_m \{\mathsf{F} \supset m = \langle x\rangle\{\mathsf{T}\}\}$	$\textsc{Quote}^t$, *1*
3	$\{\mathsf{T}\}\ \langle\Omega\rangle :_m \{\mathsf{T}\}$	*2*

Example 6.11. We continue with an example where quasi-quotes divergent code gets unquoted, and then re-quoted without further use.

1	$\{\mathsf{T}\}\ \langle\Omega\rangle :_m \{\mathsf{T}\}$	*Ex. 16*
2	$\{x \Downarrow\}\ x :_b \{x = b\}$	$\textsc{Var}^t_m$
3	$\{\mathsf{T}\}\ \langle x\rangle :_u \{x \Downarrow \supset\ \ u = \langle b\rangle\{x = b\}\}$	$\textsc{Quote}^t$, *2*
4	$\{\mathsf{T} \wedge ((\forall x^{\Box}.\mathsf{T}) \vee \forall m.(\mathsf{T} \supset m = \langle x\rangle\{\mathsf{T}\}))\}$ $\mathtt{let}\ \langle x\rangle = \langle\Omega\rangle\ \mathtt{in}\ \langle x\rangle\ :_u$ $\{\exists m x^{\Box}.((m = \langle\cdot\rangle \supset m = \langle x\rangle) \wedge \mathsf{T} \wedge (x \Downarrow\supset u = \langle b\rangle\{x = b\}))\}$	$\textsc{Unquote}^t$, *1, 3*
5	$\{\mathsf{T}\}$ $\mathtt{let}\ \langle x\rangle = \langle\Omega\rangle\ \mathtt{in}\ \langle x\rangle\ :_u$ $\{\exists m x^{\Box}.((m = \langle\cdot\rangle \supset m = \langle x\rangle) \wedge \mathsf{T} \wedge (x \Downarrow\supset u = \langle b\rangle\{x = b\}))\}$	*4*
6	$\{\mathsf{T}\}\ \mathtt{let}\ \langle x\rangle = \langle\Omega\rangle\ \mathtt{in}\ \langle x\rangle :_u \{\mathsf{T}\}$	*5*

Line 6 follows because clearly $\forall x^{\Box}.\mathsf{T}$ is equivalent to T. Finally, the simplification in Line 6 is immediate, because everything implies T.

Example 6.12. We continue by determining the TCAP for $\mathtt{let}\ \langle x\rangle = \langle\Omega\rangle\ \mathtt{in}\ x$, which is a divergent program, which we expect to be equivalent to

$$\{\mathsf{F}\}\ \mathtt{let}\ \langle x\rangle = \langle\Omega\rangle\ \mathtt{in}\ x :_u \{\mathsf{T}\}.$$

We infer the TCAP as follows:

1	$\{\mathsf{T}\}\ \langle\Omega\rangle :_x \{\mathsf{T}\}$	*Ex. 16*
2	$\{x \Downarrow\}\ x :_u \{u = x\}$	$\textsc{Var}^t_m$
3	$\{\mathsf{T} \wedge ((\forall x^{\Box}.x \Downarrow) \vee \forall m.(\mathsf{T} \supset m = \langle x\rangle\{x \Downarrow\}))\}$ $\mathtt{let}\ \langle x\rangle = \langle\Omega\rangle\ \mathtt{in}\ x\ :_u$ $\{\exists m x^{\Box}.((m = \langle\cdot\rangle \supset m = \langle x\rangle) \wedge \mathsf{T} \wedge u = x)\}$	$\textsc{Unquote}^t$, *1, 2*
4	$\{\mathsf{F}\}\ \mathtt{let}\ \langle x\rangle = \langle\Omega\rangle\ \mathtt{in}\ x :_u \{\mathsf{T}\}$	*3*

We now explain the simplifications leading to Line 4. By the axiom (div_m) which says that not all modal variables denote a terminating program, we know that $\forall x^{\Box}.x \Downarrow$ is false. Likewise $\forall m.(\mathsf{T} \supset m = \langle x\rangle\{x \Downarrow\})$ is equivalent to $\forall m.m = \langle x\rangle\{x \Downarrow\}$ which claims that any quasi-quote must contain a terminating program, which is false by the axiom (div) in Figure 3. Consequently, the precondition in Line 3 is equivalent to F. Simplification of the postcondition to T is immediate.

7. Conclusion

We have proposed a program logic for an HGRTMP language, and established key metalogical properties like completeness and the correspondence between axiomatic and operational semantics. We are not aware of previous work on program logics for meta-programming. So far, only typing systems for statically enforcing program properties have been investigated; the two most expressive are Ωmega [29] and Concoqtion [15]. Both use indexed typed to achieve expressivity. Ωmega is a call-by-value variant of Haskell with generalised algebraic

datatypes (GADTs) and an extensible kind system. In Ωmega, GADTs can express easily datatypes representing object-programs, whose meta-level types encode the object-level types of the programs represented. Tagless interpreters can directly be expressed and typed for these object programs. Ωmega is expressive enough to encode the MetaML typing system together with a MetaML interpreter in a type-safe manner. Concoqtion is an extension of MetaOCaml and uses the term language of the theorem prover Coq to define index types, specify index operations, represent the properties of indexes and construct proofs. Basing indices on Coq terms opens all mathematical insight available as Coq libraries to use in typing meta-programs. Types in both languages are not as expressive with respect to properties of *meta-programs themselves* as our logics, which capture exactly the observable properties. Nevertheless, program logic and type-theory are not mutually exclusive; on the contrary, reconciling both in the context of meta-programming is an important open problem.

$\textsc{Pcf}_{\textsc{dp}}$ lacks the ability, vital for realistic meta-programming, to manipulate *open* code, i.e. code with free variables. The λ°-calculus [13] is a small language for HGRTMP where code with free variables can be manipulated. As with $\textsc{Pcf}_{\textsc{dp}}$ (without recursion), there is a Curry-Howard correspondence: λ° is a proof calculus for a temporal logic. Due to its simplicity, λ° is an ideal object of study to see how the logic presented here can be generalised to open code. The simplicity of λ° has a price: the calculus cannot be directly extended with a construct expressing the evaluation of generated code. A more ambitious target that allows the manipulation of terms with free variables, but also the evaluation of generated code, is Taha's and Nielsen's system of environment classifiers [32], which also forms the basis of MetaOCaml, the most widely studied meta-programming language in the MetaML tradition. Moreover, [34] presents a Curry-Howard correspondence between a typing system closely related to that of [32] and a modal logic. We believe that a logical account of meta-programming with open code is a key challenge in bringing program logics to realistic meta-programming languages.

A different challenge is to add state to $\textsc{Pcf}_{\textsc{dp}}$ and extend the corresponding logics. We expect the logical treatment of state given in [4, 38] to extend smoothly to a meta-programming setting. The main question is what typing system to use to type stateful meta-programming. The system used in MetaOCaml, based on [32], is unsound in the presence of state due to a form of scope extrusion. Recent versions of MetaOCaml add a dynamic check to detect this behaviour. As an alternative to dynamic typing, the Java-like meta-programming language Mint [37] simply prohibits the *sharing* of state between different meta-programming stages, resulting in a statically sound typing system. Yet another approach is given in [24] where a two-level HGRTMP language is introduced with delimited control operators and a restriction of side effects during code generation to the scope of generated binders. That guarantees well-typedness. We believe that these approaches can all be made to coexist with modern logics for higher-order state [4, 38].

Relatedly, [14] presents a unstaging translation from $\textsc{Pcf}_{\textsc{dp}}$ to $\textsc{Pcf}$. The key idea is that a quasi-quote $\langle M \rangle$ is turned into a thunk $\lambda().M'$ where M' is the translation of M. Consequently the type $\langle \alpha \rangle$ is translated to $\mathsf{Unit} \to \alpha'$ where α' is the translation to α. What are the properties of this translation? Is it fully abstract? The translation can be extended to translating $\textsc{Pcf}_{\textsc{dp}}$ assertions and proofs into the logic for $\textsc{Pcf}$. Would this latter translation be logically fully abstract in the sense of [26]? If unstaging is fully abstract, then it should be possible to recover the logic presented here from the logic for $\textsc{Pcf}$ and the translation.

Reasoning about HGRTMP using unstaging translations looks promising. In [10] a complex HGRTMP language that allows the manipulation of open code and the capture of free variables is unstaged. However, logical reasoning about meta-programs in the target language of an unstaging translation incurs a cost: it leads to larger formulae and proofs in comparison with reasoning about the meta-programs directly using the source language. Moreover, this cost is paid in every reasoning process. In contrast, the cost of developing a logic for the meta-programming language is paid only once. An additional question is whether unstaging translations are fully abstract for more complicated HGRTMP languages.

A technical issue we left open is to do with the size of characteristic formulae. The inference system in Section 5 may lead to an exponential blow up of TCAPs vis-a-vis the programs they are derived from. We believe that it is possible to give an alternative inference system for TCAPs such that the size of the TCAP is linear, i.e. $O(n)$, in the size of the program. In [7, 19] this is achieved for logics of partial correctness for PCF-like languages, and in [8] for a simple imperative language.

Finally we have a question about modal quantification: 'normal' reasoning about PCF_{DP}-programs using the rules and axioms of Section 3 appears to be possible entirely without modal quantification. Can we abolish modal quantification altogether? If not, why is the lack of modal quantification no issue in practise?

Acknowledgements. We thank Dana Xu for careful comments on the short version of this article, Arthur Charguéraud for discussions about characteristic formulae and completeness, and Jacques Carette, Billiejoe Charlton, Rowan Davies, Oleg Kiselyov, Chung-chieh Shan, and Walid Taha for answering questions about meta-programming. We also thank the anonymous reviewers for their insightful comments.

References

[1] L. Aceto and A. Ingólfsdóttir. Characteristic formulae: From automata to logic. BRICS Report Series RS-07-2, BRICS, Department of Computer Science, University of Aarhus, 2007.

[2] A. Bawden. Quasiquotation in LISP. In *Proc. Workshop on Partial Evaluation and Semantics-Based Program Manipulation*, pages 88 – 99, 1999.

[3] M. Berger. Program Logics for Sequential Higher-Order Control. In *Proc. FSEN*, pages 194–211, 2009.

[4] M. Berger, K. Honda, and N. Yoshida. A Logical Analysis of Aliasing in Imperative Higher-Order Functions. *Journal of Functional Programming*, 17(4-5):473–546, 2007.

[5] M. Berger, K. Honda, and N. Yoshida. Completeness and Logical Full Abstraction in Modal Logics for Typed Mobile Processes. In *Proc. ICALP*, pages 99–111, 2008.

[6] M. Berger and L. Tratt. Program Logics for Homogeneous Meta-Programming. In *Proc. LPAR*, pages 64–81, 2010.

[7] A. Charguéraud. Program verification through characteristic formulae. In *Proc. ICFP*, pages 321–332, 2010.

[8] A. Charguéraud. Characteristic formulae for the verification of imperative programs. In *Proc. ICFP*, pages 418–430, 2011.

[9] N. Charlton. Reasoning about string-based runtime code generation. Unpublished, October 2011.

[10] W. Choi, B. Aktemur, K. Yi, and M. Tatsuta. Static Analysis of Multi-staged Programs via Unstaging Translation. In *Proc. POPL*, pages 81–92, 2011.

[11] E. M. Clarke, Jr. Programming Language Constructs for Which It Is Impossible To Obtain Good Hoare Axiom Systems. *J. ACM*, 26(1):129–147, Jan. 1979.

[12] S. A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.*, 7(1):70–90, 1978.

[13] R. Davies. A temporal-logic approach to binding-time analysis. In *Proc. LICS*, pages 184–195, 1996.

[14] R. Davies and F. Pfenning. A modal analysis of staged computation. *J. ACM*, 48(3):555–604, 2001.

[15] S. Fogarty, E. Pašalić, J. Siek, and W. Taha. Concoqtion: Indexed Types Now! In *Proc. PEPM*, pages 112–121, 2007.
[16] C. A. Gunter. *Semantics of Programming Languages*. MIT Press, 1995.
[17] T. Hoare. An Axiomatic Basis of Computer Crogramming. *CACM*, 12, 1969.
[18] K. Honda. From Process Logic to Program Logic. In *ICFP'04*, pages 163–174. ACM Press, 2004.
[19] K. Honda, M. Berger, and N. Yoshida. Descriptive and Relative Completeness of Logics for Higher-Order Functions. In *Proc. ICALP*, pages 360–371, 2006.
[20] K. Honda and N. Yoshida. A compositional logic for polymorphic higher-order functions. In *Proc. PPDP*, pages 191–202, 2004.
[21] K. Honda, N. Yoshida, and M. Berger. An Observationally Complete Program Logic for Imperative Higher-Order Functions. In *Proc. LICS*, pages 270–279, 2005.
[22] K. Honda, N. Yoshida, and M. Berger. An Observationally Complete Program Logic for Imperative Higher-Order Functions. Technical Report DTR13-2, Imperial College, Department of Computing, 2013.
[23] J. Inoue and W. Taha. Reasoning about multi-stage programs. In *Proc. ESOP*, pages 357–376, 2012.
[24] Y. Kameyama, O. Kiselyov, and C.-C. Shan. Shifting the Stage: Staging with Delimited Control. In *Proc. PEPM*, pages 111–120, 2009.
[25] T. Kleymann. Hoare Logic and Auxiliary Variables. Technical Report ECS-LFCS-98-399, LFCS, Univ. of Edinburgh, October 1998.
[26] J. Longley and G. Plotkin. Logical Full Abstraction and PCF. In *Tbilisi Symposium on Logic, Language and Information*, CSLI, 1998.
[27] A. M. Pitts. Operationally-based theories of program equivalence. In *Semantics and Logics of Computation*, pages 241–298. Cambridge University Press, 1997.
[28] J. C. Reynolds. Separation logic: a logic for shared mutable data structures. In *Proc. LICS'02*, pages 55–74, 2002.
[29] T. Sheard and N. Linger. Programming in Ωmega. In *Proc. Central European Functional Programming School*, pages 158–227, 2007.
[30] T. Sheard and S. Peyton Jones. Template meta-programming for Haskell. In *Proc. Haskell workshop*, pages 1–16, 2002.
[31] W. Taha. *Multi-Stage Programming: Its Theory and Applications*. PhD thesis, Oregon Graduate Institute of Science and Technology, 1993.
[32] W. Taha and M. F. Nielsen. Environment classifiers. In *Proc. POPL*, pages 26–37, 2003.
[33] L. Tratt. Compile-time meta-programming in a dynamically typed OO language. In *Proc. DLS*, pages 49–64, Oct. 2005.
[34] T. Tsukada and A. Igarashi. A Logical Foundation for Environment Classifiers. *Logical Methods in Computer Science*, 6(4:8):1–43, 2010.
[35] W. van Orman Quine. *From a Logical Point of View*. Harvard Univ. Press, 2003.
[36] W. van Orman Quine. *Mathematical Logic (Revised Edition)*. Harvard Univ. Press, 2003.
[37] E. Westbrook, M. Ricken, J. Inoue, Y. Yao, T. Abdelatif, and W. Taha. Mint: Java multi-stage programming using weak separability. In *Proc. PLDI*, pages 400–411, 2010.
[38] N. Yoshida, K. Honda, and M. Berger. Logical reasoning for higher-order functions with local state. *Logical Methods in Computer Science*, 4(2), 2008.

Appendix A. Axioms for Pcf that are also valid for Pcf_{DP}

Section 3 presented the axioms of our logic that involve meta-programming features. Other axioms are listed in Figure 7. The axioms are standard, and explanation, as well as soundness proofs can be found in [4, 22, 38]. Moreover, soundness proofs for the axioms given in Figure 7 are straightforward adaptations of the proofs for the axioms in Figure 3. The presentation uses the following abbreviations:

$$\mathsf{Ext}(xy) \text{ stands for } \forall az.(x \bullet z = w\{w = a\} \equiv y \bullet z = w\{w = a\}).$$

Note that it is vital for x and y to be non-modal. The direction $\mathsf{Ext}(xy) \supset x = y$ is unsound otherwise, because $\mathsf{Ext}(xy)$ cannot distinguish between e.g. appropriately typed Ω and $\lambda x.\Omega$.

$(e1)$	$x \bullet y = z\{A\} \wedge x \bullet y = z\{B\}$	$\equiv$	$x \bullet y = z\{A \wedge B\}$	
$(e2)$	$x \bullet y = z\{\neg A\}$	$\supset$	$\neg x \bullet y = z\{A\}$	
$(e3)$	$x \bullet y = z\{A\} \wedge \neg x \bullet y = z\{B\}$	$\equiv$	$x \bullet y = z\{A \wedge \neg B\}$	
$(e4)$	$x \bullet y = z\{A \wedge B\}$	$\equiv$	$A \wedge x \bullet y = z\{B\}$	$z \notin \mathsf{fv}(A)$
$(e5)$	$x \bullet y = z\{\forall a^\alpha.A\}$	$\equiv$	$\forall a^\alpha.x \bullet y = z\{A\}$	$a \neq x, y, z$
$(e6)$	$(A \supset B) \wedge x \bullet y = z\{A\}$	$\supset$	$x \bullet y = z\{B\}$	$z \notin \mathsf{fv}(A, B)$
(div)	$\neg\forall x^\alpha.m \bullet x \Downarrow$			
(ext)	$x = y$	$\equiv$	$\mathsf{Ext}(xy)$	x, y of function type both non-modal
(e_α)	$x \bullet y = z\{A\}$	$\equiv$	$x \bullet y = a\{x \bullet y = z\{A \wedge a = z\}\}$	$a \notin \{x, y\}, a \in \mathsf{fv}(A)$ implies $a = z$

Figure 7: PCF_{DP} axioms not involving meta-programming constructs. Except were noted otherwise, all free variables can be modal or non-modal.

Appendix B. Omitted proofs for Section 5 (Completeness)

It remains to establish Theorem 5.3.2 for the rules [Var^t, Const^t, Abs^t, App^t, Op^t, If^t, Rec^t]. All proofs here are variants of the proofs in the unpublished long version of [19].

Var^t: The MTC is trivially true. For (*closure-2*), assume that $(\xi \cdot x : V, \sigma) \models \mathsf{T}$ and $(\xi \cdot x : V, m : W, \sigma) \models x = m$. Then immediately $V \simeq W$, hence $V \lesssim W$ as required.

Const^t: Similar to [Var^t] and omitted.

Abs^t: Since abstractions are values, (MTC) is trivially true. For (*closure-2*), assume that (ξ, σ) is a model and $(\xi \cdot u : V, \sigma) \models \forall x.(A \supset u \bullet x = m\{B\})$.

1	$(\xi \cdot u : V, \sigma) \models \forall x.(A \supset u \bullet x = m\{B\})$	*Assumption*
2	$(\xi \cdot u : V \cdot x : W, \sigma) \models A \supset u \bullet x = m\{B\}$	*W arbitrary, 1*
3	$(\xi \cdot u : V \cdot x : W, \sigma) \models A$	*Assumption*
4	$(\xi \cdot u : V \cdot x : W, \sigma) \models u \bullet x = m\{B\}$	*2, 3*
5	$VW \Downarrow U \qquad (\xi \cdot u : V \cdot x : W \cdot m : U, \sigma) \models B$	*4*
6	$\eta \stackrel{def}{=} (\xi \cdot x : W \cdot m : U, \sigma) \models B$	$u \notin \mathsf{fv}(B)$, *Prop. 3.3.2, 5*
7	$(\xi \cdot x : W, \sigma) \models A$	$u \notin \mathsf{fv}(A)$, *Prop. 3.3.2, 3*
8	$M\eta \lesssim U$	*(IH), (closure-2), 6, 7*
9	$VW \simeq U$	*Prop. 2.1. (6), 5*
10	$M\eta \lesssim VW$	*8, 9*
11	$\eta' \stackrel{def}{=} (\xi \cdot m : U, \sigma)$	
12	$(\lambda x.M\eta')W \to M\eta'[W/x] = M\eta$	*Prop. 3.3.4*

13	$(\lambda x.M\eta')W \lesssim M\eta$	*Prop. 2.1. (6), 12*
14	$(\lambda x.M\eta')W = (\lambda x.M)\eta' W \lesssim M\eta \lesssim U \lesssim VW$	*9, 10, 13*
15	for all W $(\lambda x.M)\eta' W \lesssim VW$	*W arbitrary, 14*
16	$(\lambda x.M)\eta' \lesssim V$	*Lem. 2.1 (8), 15*

App^t: We begin with (MTC). Let η be an appropriately typed model such that

$$(MN)\eta \Downarrow V$$

Then in particular $M\eta \Downarrow W$, $N\eta \Downarrow U$ and $WU \Downarrow C$. By (IH) the first two mean that

$$\eta \models A_1 \qquad \eta \models A_2$$

Op^t: We treat the special case of addition. The MTC follows directly from the (IH), noting that $(M + N)\eta \Downarrow$ holds exactly when $M\eta \Downarrow$ and $N\eta \Downarrow$. For (*closure-2*) we reason as follows.

1	$\eta \stackrel{def}{=} (\xi, \sigma) \models A_1 \wedge A_2$	*Assumption*
2	$(\xi \cdot u : V, \sigma) \models \exists m_1 m_2.(u = m_1 + m_2 \wedge B_1 \wedge B_2)$	*Assumption*
3	$(\xi \cdot u : V \cdot m_1 : W_1 \cdot m_2 : W_2, \sigma) \models u = m_1 + m_2 \wedge B_1 \wedge B_2$	*2*
4	$(\xi \cdot m_i : W_i, \sigma) \models B_i \quad i = 1, 2, u, m_{3-i} \notin \mathsf{fv}(B_i)$	*Prop. 3.3.2, 3*
5	$M_i\eta \lesssim W_i$	*(IH), 1, 4*
6	$(M_i + M_2)\eta \lesssim W_1 + W_2$	*5*
7	$W_1 + W_2 \simeq V$	*3*
8	$(M_i + M_2)\eta \lesssim V$	*6, 7*

If^t: For (MTC), with $\eta \stackrel{def}{=} (\xi, \sigma)$, $b_1 = \mathsf{t}$, $b_2 = \mathsf{f}$, assume wlog that:

$$\begin{array}{lll} (\mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2)\eta & \twoheadrightarrow & \mathtt{if}\ \mathsf{t}\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2 \qquad \text{(B.1)} \\ & \to & N_1 \\ & \Downarrow & W_1 \qquad \text{(B.2)} \end{array}$$

Now we reason as follows.

1	$M\eta \Downarrow \mathsf{t}$	*(B.1)*
2	$\eta \models A \quad (\xi \cdot m : \mathsf{t}, \sigma) \models B$	*(IH), (MTC), 1*
3	$\eta \models B[\mathsf{t}/m]$	*Prop. 3.3.4, 2*
4	$N\eta \Downarrow W_1$	*(B.2)*
5	$\eta \models A_1 \quad (\xi \cdot m : \mathsf{t}, \sigma) \models B_1$	*(IH), (MTC), 4*
6	$\eta \models B[\mathsf{t}/m] \supset A_1$	*3, 5*
7	$\eta \models B[\mathsf{f}/m]$	*Assumption towards a contradiction*

8	$\eta \models A \quad (\xi \cdot m : \mathsf{f}, \sigma) \models B$	*2, 7*
9	$M\eta \lesssim \mathsf{f}$	*(IH), (closure-2), 8*
10	$\eta \not\models B[\mathsf{f}/m]$	*19 contradicts 1*
11	$\eta \models B[\mathsf{f}/m] \supset A_2$	*10*
12	$\eta \models A \wedge \bigwedge_i (B[b_i/m] \supset A_i)$	*6, 11*

The reverse direction follows from soundness.

For (*closure-2*) the following derivation gets us towards the result.

1	$\eta \stackrel{def}{=} (\xi, \sigma) \models A \wedge \bigwedge_i (B[b_i/m] \supset A_i)$	*Assumption*
2	$\eta' \stackrel{def}{=} (\xi \cdot V, \sigma) \models B[\mathsf{t}/m] \wedge B_1$	*Assumption wlog*
3	$\eta \models B[\mathsf{t}/m]$	$u \notin \mathsf{fv}(B)$, *Prop. 3.3.2, 2*
4	$\eta \models A_1$	*1, 3*
5	$N_1\eta \lesssim V$	*(IH), closure-2, 2, 4*
6	$\eta \models A$	*1*
7	$M\eta \Downarrow$	*(MTC), 6*
8	$(\xi \cdot m : \mathsf{t}, \sigma) \models B$	*3*
9	$M\eta \lesssim \mathsf{t}$	*(IH), closure-2, 6, 8*
10	$M\eta \Downarrow \mathsf{t}$	*7, 9*

We use these facts to derive:

$$\begin{array}{lcll} (\mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2)\eta & \twoheadrightarrow & \mathtt{if}\ \mathsf{t}\ \mathtt{then}\ N_1\eta\ \mathtt{else}\ N_2\eta & (10) \\ & \to & N_1\eta & \\ & \lesssim & V & (5) \end{array}$$

Using Proposition 2.1. (6), this implies the required

$$(\mathtt{if}\ M\ \mathtt{then}\ N_1\ \mathtt{else}\ N_2)\eta \lesssim V.$$

Rec^t: In this case too, (MTC) is trivial. For (*closure-2*) let $\eta \stackrel{def}{=} (\xi, \sigma)$ and assume that

$$(\xi \cdot m : V, \sigma) \models A[m/g]$$

which, by Proposition 3.3.4 is equivalent to

$$(\xi \cdot m : V \cdot g : V, \sigma) \models A.$$

We now show by nested induction on n that for all $n \geq 0$ it is the case that

$$W_n\eta \ \lesssim\ V$$

where the W_n are defined as follows (cf. Proposition 2.1. (12)).

$$W_0 \stackrel{def}{=} \Omega \qquad\qquad W_{n+1} \stackrel{def}{=} \lambda x.M[W_n/g].$$

The base case $n = 0$ is trivial. For the inductive step of the inner induction, let $W_n\eta \lesssim V$.

$$\begin{array}{rcll} W_{n_1} & = & (\lambda x.M[W_n/g])\eta & \\ & = & \lambda x.M[W_n\eta/g] & \\ & \lesssim & \lambda x.M[V/g] & \text{(B.3)} \\ & = & \lambda x.M(\xi \cdot g : V, \sigma) & \\ & \lesssim & V & \text{(B.4)} \end{array}$$

Here (B.3) follows from the inner (IH) together with $[\cdot/g]$'s being monotonic w.r.t. to $\lesssim$ (Lemma 2.1. (5)). On the other hand, (B.4) is directly by the outer (IH) and (*closure-2*).

Hence we have $W_n\eta \lesssim V$ for all n. Since $\mu g.\lambda x.M\eta \not\lesssim V$, then $W_n \not\lesssim$ for some n. Using Proposition 2.1. (12) we conclude that $\mu g.\lambda x.M\eta \lesssim V$.