Annals of Pure and Applied Logic 163 (2012) 935–950

Contents lists available at SciVerse ScienceDirect

Annals of Pure and Applied Logic

journal homepage: www.elsevier.com/locate/apal

Justification Logic as a foundation for certifying mobile computation☆

Eduardo Bonelli[a,b,*], Federico Feller[c]

[a] Depto. de Ciencia y Tecnología, Universidad Nacional de Quilmes, Roque Sáenz Peña, 352 Bernal - B1876BXD - Bs. As., Argentina

[b] CONICET, Argentina

[c] LIFIA, Facultad de Informática, Universidad Nacional de La Plata, Argentina

ARTICLE INFO

Article history:
Available online 19 October 2011

MSC:
68N18
03B40

Keywords:
Curry–de Bruijn–Howard isomorphism
Typed lambda calculus
Mobile computation
Justification Logic

ABSTRACT

We explore an intuitionistic fragment of Artëmov's *Justification Logic* as a type system for a programming language for *mobile units*. Such units consist of both a code and a certificate component. Our language, the *Certifying Mobile Calculus*, caters for code and certificate development in a unified theory. In the same way that mobile code is constructed out of code components and extant type systems track local resource usage to ensure the mobile nature of these components, our system *additionally* ensures correct *certificate construction* out of certificate components. We present proofs of type safety and strong normalization for a run-time system based on an abstract machine.

1. Introduction

We explore an intuitionistic fragment (IJL) of Artëmov's *Justification Logic* (JL) as a type system for a programming language for *mobile units*. This language caters for both code and certificate development in a unified theory. JL may be regarded as refinement of modal logic S4 in which $\Box A$ is replaced by $[s]A$, for s a *proof term* expression, and is read: *"s is a proof of A"*. It is sound and complete w.r.t. provability in PA (see [2] for a precise statement) and realizes all theorems of S4. It therefore provides an answer to the (long-standing) problem of associating an exact provability semantics to S4 [2]. JL is purported to have important applications not only in logic but also in Computer Science [5]. This work may be regarded as a small step in exploring the applications of JL in programming languages and type theory.

Modal necessity $\Box A$ may be read as the type of programs that compute values of type A and that do not depend on local resources [15,19,17] or resources not available at the current stage of computation [22,23,12]. The former reading refers to *mobile computation* ($\Box A$ as the type of mobile code that computes values of type A) while the latter to *staged computation* ($\Box A$ as the type of code that generates, at run-time, a program for computing a value of type A). See Section 8 for further references. We introduce the *Certifying Mobile Calculus* or $\lambda_{\Box}^{\mathsf{Cert}}$ by taking a mobile computation interpretation of IJL. IJL's mechanism for internalizing its own derivations provides a natural setting for code certification. A contribution of our approach is that, in the same way that mobile code is constructed out of code components and extant type systems track local resource usage to ensure the mobile nature of these components, our system *additionally* ensures correct *certificate construction* out of certificate components. A mobile unit is an expression of the form $box_s\, M$ consisting of both a code part M and a certificate part s. The syntax of code and certificates is described in detail in Section 3. The certificate is an encoding of a type derivation. This expression shall be well-typed only in the case where s encodes a typing derivation for M. Since

☆ Work partially supported by Instituto Tecnológico de Buenos Aires.

* Corresponding author. Tel.: +54 11 4365 7100x4310.
E-mail addresses: ebonelli@unq.edu.ar (E. Bonelli), federico.feller@gmail.com (F. Feller).

0168-0072/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.apal.2011.09.007

composite mobile units may be constructed out of simpler mobile units, our type system also guarantees that certificates for the former are correctly constructed out of certificates for the latter. See Section 7 for examples.

$\lambda_{\Box}^{\mathsf{Cert}}$ arises from a Curry–de Bruijn–Howard interpretation of a Natural Deduction presentation of IJL based on a judgemental analysis of Justification Logic given in [6]. Propositions and proofs of IJL correspond to types and terms of $\lambda_{\Box}^{\mathsf{Cert}}$. Regarding semantics, we provide an operational reading of expressions encoding proofs in this system in terms of global computation. An abstract machine is introduced that computes over multiple worlds. Apart from the standard lambda calculus expressions new expressions for constructing mobile units and for computing in remote worlds are introduced. We state and prove *type safety* of a type system for $\lambda_{\Box}^{\mathsf{Cert}}$ w.r.t. its operational semantics. Also, we prove strong normalization.

This paper is organized as follows. Section 2 briefly recapitulates ILPnd [6], a Natural Deduction presentation of IJL. We then introduce a term assignment for ILPnd and discuss differences with the term assignment in [6] including the splitting of validity variables [6] into code and certificate variables. Section 4 introduces the run-time system of $\lambda_{\Box}^{\mathsf{Cert}}$, the abstract machine for execution of $\lambda_{\Box}^{\mathsf{Cert}}$ programs. Section 5 analyses type safety and Section 6 strong normalization. Section 7 presents some examples. References to related work follows. Finally, we conclude and suggest further directions for research.

This work extends [8] by including a description of ILPnd [6] the Natural Deduction formulation of IJL, due to the first author and S. Artëmov, on which this work is based. It includes proofs of relevant results and corrects a bug in scheme (7) of the machine reduction semantics of $\lambda_{\Box}^{\mathsf{Cert}}$ together with updated proofs of subject reduction and strong normalization.

2. Natural deduction for IJL

As mentioned, JL [3,4] is a refinement of modal logic S4 in which $\Box A$ is replaced by $[s]A$. Here s is an expression representing a Hilbert style proof and is called a *proof polynomial*. In the minimal propositional logic fragment of JL without plus, IJL, proof polynomials are constructed from proof variables and constants using two operations: application "$\cdot$" and proof-checker "!". The usual propositional connectives are augmented by a new one: given a proof polynomial s and a proposition A build $[s]A$. The intended reading is: "*s is a proof of A*". The axioms and inference schemes of JL are:

A0. Axiom schemes of minimal logic in the language of JL	
A1. $[s]A \supset A$	*"verification"*
A2. $[s](A \supset B) \supset ([t]A \supset [s \cdot t]B)$	*"application"*
A3. $[s]A \supset [!s][s]A$	*"proof checker"*
R1. $\Gamma \rhd A \supset B$ and $\Gamma \rhd A$ implies $\Gamma \rhd B$	*"modus ponens"*
R2. If **A** is an axiom **A0**–**A3**, and c is a proof constant, then $\rhd [c]\mathbf{A}$	*"necessitation"*

For verification one reads:"*if s is a proof of A, then A holds*". As regards the proof polynomials the standard interpretation is as follows. For application one reads: "*if s is a proof of $A \supset B$ and t is a proof of A, then $s \cdot t$ is a proof of B*". Thus "$\cdot$" represents composition of proofs. For proof checking one reads: "*if s is a proof of A, then $!s$ is a proof of the sentence 's is a proof of A'*". Thus $!s$ is seen as a computation that verifies $[s]A$.

In previous work [6] a Natural Deduction presentation of IJL (ILPnd) is introduced by considering two sets of hypothesis, truth and validity hypothesis, and analysing the meaning of the following Hypothetical Judgement with Explicit Evidence:

$$\Delta;\ \Gamma \rhd A \mid s$$

The syntactic categories involved in this judgement are defined as follows:

Proof Term	s, t	$::=$	$a \mid v \mid s \cdot t \mid \lambda a : A.s \mid !s \mid \textsc{letc}\ s\ \textsc{be}\ v : A\ \textsc{in}\ t$
Proposition	A, B	$::=$	$P \mid A \supset B \mid [s]A$
Truth Context	Γ	$::=$	$\cdot \mid \Gamma, a : A$
Validity Context	Δ	$::=$	$\cdot \mid \Delta, v : A$

A brief description of the judgement follows. Δ is a sequence of *validity assumptions*, Γ a sequence of *truth assumptions*, A is a proposition and s is a proof term. A validity assumption is written $v : A$ where v ranges over a given infinite set of *validity variables* and states that A is true and moreover that its truth does not depend on other truth assumptions (although it may depend on validity assumptions). Likewise, a truth assumption is written $a : A$ where a ranges over a given infinite set of *truth variables* and states that A is true. If $\Delta = v_1 : A_1, \ldots, v_n : A_n$ and $\Gamma = a_1 : B_1, \ldots, a_m : B_m$, then in $\Delta;\ \Gamma$ we assume: (1) all v_i, $i \in 1..n$, to be distinct, (2) all a_j, $j \in 1..m$, to be distinct, (3) no v_i, $i \in 1..n$, should occur in $v_1 : A_1, \ldots, v_{i-1} : A_{i-1}, v_{i+1} : A_{i+1}, \ldots, v_n : A_n$; Γ and (4) no a_j, $j \in 1..m$, should occur in Δ; $a_1 : B_1, \ldots, a_{j-1} : B_{j-1}, a_{j+1} : B_{j+1}, \ldots, a_m : B_m$. We write x to denote either of these variables. The judgement is read as: *"A is true with evidence s under validity assumptions Δ and truth assumptions Γ"*. Note that s is a constituent of this judgement without whose intended reading is not possible. The meaning of this judgement is given by axiom and inference schemes (Fig. 1). We say a judgement is *derivable* if it has a derivation using these schemes.

All free occurrences of a (resp. v) in s are bound in $\lambda a : A.s$ (resp. $\textsc{letc}\ t\ \textsc{be}\ v : A\ \textsc{in}\ s$). A proposition is either a variable P, an implication $A \supset B$ or a validity proposition $[s]A$. We write "$\cdot$" for empty contexts and $s\{x/t\}$ for the result of substituting all free occurrences of x in s by t (bound variables are renamed whenever necessary); likewise for $A\{x/t\}$.

Minimal Propositional Logic Fragment

$$\frac{}{\Delta;\ \Gamma, a : A, \Gamma' \rhd A \mid a}\ \mathsf{oVar}$$

$$\frac{\Delta;\ \Gamma, a : A \rhd B \mid s}{\Delta;\ \Gamma \rhd A \supset B \mid \lambda a : A.s}\ \supset \mathsf{I} \qquad \frac{\Delta;\ \Gamma \rhd A \supset B \mid s \quad \Delta;\ \Gamma \rhd A \mid t}{\Delta;\ \Gamma \rhd B \mid s \cdot t}\ \supset \mathsf{E}$$

Provability Fragment

$$\frac{}{\Delta, v : A, \Delta';\ \Gamma \rhd A \mid v}\ \mathsf{mVar}$$

$$\frac{\Delta;\ \cdot \rhd A \mid s}{\Delta;\ \Gamma \rhd [s]A \mid !s}\ \Box\mathsf{I} \qquad \frac{\Delta;\ \Gamma \rhd [r]A \mid s \quad \Delta, v : A;\ \Gamma \rhd C \mid t}{\Delta;\ \Gamma \rhd C\{v/r\} \mid \textsc{letc}\ s\ \textsc{be}\ v : A\ \textsc{in}\ t}\ \Box\mathsf{E}$$

$$\frac{\Delta;\ \Gamma \rhd A \mid s \quad \Delta;\ \Gamma \vdash s \equiv t : A}{\Delta;\ \Gamma \rhd A \mid t}\ \mathsf{EqEvid}$$

Fig. 1. Explanation for hypothetical judgements with explicit evidence.

A brief informal explanation of some of these schemes follows. The axiom scheme oVar states that the judgement $\Delta;\ \Gamma, a : A, \Gamma' \rhd A \mid a$ is evident in itself. Indeed, if we assume that a is evidence that proposition A is true, then we immediately conclude that A is true with evidence a. The introduction scheme for the $[s]$ modality internalizes metalevel evidence into the object logic. It states that if s is unconditional evidence that A is true, then A is in fact valid with witness s (i.e. $[s]A$ is true). Evidence for the truth of $[s]A$ is constructed from the (verified) evidence that A is unconditionally true by prefixing it with a bang constructor. Finally, $\Box\mathsf{E}$ allows the discharging of validity hypothesis. In order to discharge the validity hypothesis $v : A$, a proof of the validity of A is required. In this system, this requires proving that $[r]A$ is true with evidence s, for some evidence of proof r and s. Note that r is evidence that A is unconditionally true (i.e. valid) whereas s is evidence that $[r]A$ is true. The former is then substituted in the place of all free occurrences of v in the proposition C. This construction is recorded with evidence $\textsc{letc}\ s\ \textsc{be}\ v : A\ \textsc{in}\ t$ in the conclusion.

Since ILPnd internalizes its own derivations and normalization introduces identities on derivations at the meta-level, such identities must be reflected in the object-logic too. This is the aim of EqEvid. The schemes defining the judgement of evidence equality $\Delta;\ \Gamma \vdash s \equiv t : A$ are the axioms for β equality and β equality on $\Box$ together with appropriate congruence schemes (consult [6] for details). It should be noted that soundness of ILPnd with respect to IJL does not require the presence of EqEvid. It is, however, required in order for normalization to be closed over the set of derivations.

A sample derivation in ILPnd of $[s](A \supset B) \supset [t]A \supset [s\cdot t]B$ follows, where $\Gamma = a : [s](A \supset B)$, $b : [t]A$, $\Delta = u : A \supset B$, $v : A$ and $r = \textsc{letc}\ a\ \textsc{be}\ u : A \supset B\ \textsc{in}\ \textsc{letc}\ b\ \textsc{be}\ v : A\ \textsc{in}\ !(u \cdot v)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta;\ \cdot \rhd A \supset B \mid u}\quad \dfrac{}{\Delta;\ \cdot \rhd A \mid v}}{\Delta;\ \cdot \rhd B \mid u \cdot v}\ \Box\mathsf{I}}{\Delta;\ \Gamma \rhd [u \cdot v]B \mid !(u \cdot v)} \qquad \dfrac{}{u : (A \supset B);\ \Gamma \rhd [t]A \mid b}}{u : (A \supset B);\ \Gamma \rhd [u \cdot t]B \mid \textsc{letc}\ b\ \textsc{be}\ v : A\ \textsc{in}\ !(u \cdot v)}\ \Box\mathsf{E} \qquad \dfrac{}{\cdot;\ \Gamma \rhd [s](A \supset B) \mid a}}{\cdot;\ \Gamma \rhd [s \cdot t]B \mid \textsc{letc}\ a\ \textsc{be}\ u : A \supset B\ \textsc{in}\ \textsc{letc}\ b\ \textsc{be}\ v : A\ \textsc{in}\ !(u \cdot v)}\ \Box\mathsf{E}}{\cdot;\ a : [s](A \supset B) \rhd [t]A \supset [s \cdot t]B \mid \lambda b : [s]A.r}\ \supset \mathsf{I}}{\cdot;\ \cdot \rhd [s](A \supset B) \supset [t]A \supset [s \cdot t]B \mid \lambda a : [s](A \supset B).\lambda b : [t]A.r}\ \supset \mathsf{I}$$

3. Term assignment

We assume a set $\{w_1, w_2, \ldots\}$ of worlds, a set $\{\overset{\bullet}{v}_1, \overset{\bullet}{v}_2, \ldots\}$ of code variables and a set $\{\overset{\circ}{v}_1, \overset{\circ}{v}_2, \ldots\}$ of certificate variables. We use Σ for a (finite) set of worlds. Δ and Γ are as before. The syntactic categories of *certificates*, *values* and *terms* are defined as follows:

$$\begin{array}{rcl} s, t & ::= & a \mid \overset{\circ}{v} \mid s \cdot t \mid \lambda a : A.s \mid\ !s \mid \mathit{letc}\ s\ \mathit{be}\ \overset{\circ}{v} : A\ \mathit{in}\ t \mid \mathit{fetch}(s) \\ V & ::= & \mathit{box}_s\, M \mid \lambda a.M \\ M, N & ::= & a \mid \overset{\bullet}{v} \mid V \mid M\, N \mid \mathit{unpack}\, M\ \mathit{to}\ \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\ \mathit{in}\ N \mid \mathit{fetch}[w]\, M \end{array}$$

Certificates have two kinds of variables. Local variables a are used for abstracting over local assumptions when constructing certificates. Certificate variables $\mathring{v}$ represent unknown certificates. $s \cdot t$ is certificate composition. $!s$ is certificate endorsement. *letc* s *be* $\mathring{v}$: A *in* t is certificate validation, the inverse operation to endorsement. Finally, *fetch*(s) certifies the *fetch* code movement operation to be described shortly. An example of a certificate is the following, which encodes the sample derivation of Section 2:

$$\lambda a : [s](A \supset B).\lambda b : [t]A.letc\ a\ be\ \mathring{u}: A \supset B\ in\ (letc\ b\ be\ \mathring{v}: A\ in\ !(\mathring{u} \cdot \mathring{v}))$$

Values are a subset of terms that represent the result of computations of well-typed, closed terms. A value of the form $\lambda a.M$ is an abstraction and one of the form $box_s\, M$ is a *mobile unit*. A *term* is either a term variable for local code a, a term variable for mobile code $\overset{\bullet}{v}$, a value V, an application term $M\,N$, an unpacking term for extraction of code–certificate pairs from mobile units *unpack* M *to* $\langle \overset{\bullet}{v}, \mathring{v}\rangle$ *in* N (free occurrences of $\mathring{v}$ and $\overset{\bullet}{v}$ in N are bound by this construct) or a fetch term $fetch[w]\,M$. In an unpacking term, M is the argument and N is the body; in a fetch term we refer to w as the target of the *fetch* and M as its body. The operational semantics of these constructs is discussed in Section 4.

The term assignment results essentially (the differences are explained below) from the schemes of Fig. 1 with terms encoding derivations and localizing the hypothesis in Δ, Γ at specific worlds. Also, a reference to the current world is added. Typing judgements take the form

$$\Sigma;\ \Delta;\ \Gamma \rhd M : A@w \mid s \tag{1}$$

Σ is a list of worlds; all worlds referenced in the judgement belong to Σ. Validity and truth contexts are now sequences of expressions of the form $v : A@w$ and $a : A@w$, respectively, with $w \in \Sigma$. The former indicates that mobile unit v computing a value of type A may be assumed to exist and to be located at world w. The latter indicates that a local value a of type A may be assumed to exist at world w. The truth of a proposition at w shall rely, on the one hand, on truth hypothesis in Γ that are located at w, and on the other, on validity hypothesis in Δ that have been fetched, from their appropriate hosts, to the current location w. Logical connectives bind tighter than @, therefore an expression such as $A \supset B@w$ should be read as $(A \supset B)@w$.

It should be mentioned that IJL is not a hybrid logic [1]. In other words, $A@w$ is not a proposition of our object logic. For example, expressions of the form $A@w \supset B@w'$ are not valid propositions.

Typing schemes defining (1) are presented in Fig. 2 and discussed below. A first difference with ILPnd is that the scheme EqEvid has been dropped. Although the latter is required for normalization of derivations to be a closed operation (as already mentioned), our operational interpretation of terms does not rely on normalization of Natural Deduction proofs. For a computational interpretation of IJL based on normalization the reader may consult [6]. A further difference is that $\Box$I has been refined into two schemes, namely $\Box I$ and *Fetch*. The first introduces a modal formula and states it to be true at the current world w. The second states that all worlds accessible to w may also assume this formula to be true.

In this work mobile code is accompanied by a certificate. We speak of *mobile units* rather than mobile code to emphasize this. Since mobile units are expressions of modal types and validity variables v represent holes for values of modal types, validity variables v may actually be seen as pairs $\langle \overset{\bullet}{v}, \mathring{v}\rangle$. Here $\overset{\bullet}{v}$ is the mobile code component and $\mathring{v}$ is the certificate component of the mobile unit.[1] As a consequence, the modality axiom mVar of ILPnd now takes the following form, where judgement $\Sigma \vdash w$ ensures w is a world in Σ (it is defined by requiring $w \in \Sigma$):

$$\frac{\Sigma \vdash w}{\Sigma;\ \Delta, v : A@w, \Delta';\ \Gamma \rhd \overset{\bullet}{v}: A@w \mid\ \mathring{v}}\ VarV$$

Substitution of code variables for terms in terms ($M\{\overset{\bullet}{v}/N\}$) and substitution of certificate variables for certificates in certificates ($t\{\mathring{v}/s\}$) and in terms ($M\{\mathring{v}/s\}$) is defined as expected. We illustrate the definition of the first of these notions.

$$\begin{aligned}
a\{\overset{\bullet}{v}/N\} &\overset{\text{def}}{=} a\\
\overset{\bullet}{v}\{\overset{\bullet}{v}/N\} &\overset{\text{def}}{=} N\\
\overset{\bullet}{u}\{\overset{\bullet}{v}/N\} &\overset{\text{def}}{=} \overset{\bullet}{u}\\
(PQ)\{\overset{\bullet}{v}/N\} &\overset{\text{def}}{=} P\{\overset{\bullet}{v}/N\}Q\{\overset{\bullet}{v}/N\}\\
(\lambda a.P)\{\overset{\bullet}{v}/N\} &\overset{\text{def}}{=} \lambda a.P\{\overset{\bullet}{v}/N\}\\
(box_s\, P)\{\overset{\bullet}{v}/N\} &\overset{\text{def}}{=} box_s\, P\{\overset{\bullet}{v}/N\}\\
(fetch[w]\,P)\{\overset{\bullet}{v}/N\} &\overset{\text{def}}{=} fetch[w]\,P\{\overset{\bullet}{v}/N\}\\
(unpack\,P\ to\ \langle \overset{\bullet}{u}, \mathring{u}\rangle\ in\ Q)\{\overset{\bullet}{v}/N\} &\overset{\text{def}}{=} unpack\,P\{\overset{\bullet}{v}/N\}\ to\ \langle \overset{\bullet}{u}, \mathring{u}\rangle\ in\ Q\{\overset{\bullet}{v}/N\}
\end{aligned}$$

[1] The "$\circ$" is reminiscent of a wrapping with which the interior "$\bullet$" is protected. Hence our use of the former symbol for certificates and the latter for code.

$$\frac{\Sigma \vdash w}{\Sigma;\ \Delta;\ \Gamma, a : A@w, \Gamma' \rhd a : A@w \mid a}\ VarT$$

$$\frac{\Sigma;\ \Delta;\ \Gamma, a : A@w \rhd M : B@w \mid s}{\Sigma;\ \Delta;\ \Gamma \rhd \lambda a.M : A \supset B@w \mid \lambda a : A.s}\ \supset I$$

$$\frac{\Sigma;\ \Delta;\ \Gamma \rhd M : A \supset B@w \mid s \quad \Sigma;\ \Delta;\ \Gamma \rhd N : A@w \mid t}{\Sigma;\ \Delta;\ \Gamma \rhd M\,N : B@w \mid s \cdot t}\ \supset E$$

$$\frac{\Sigma \vdash w}{\Sigma;\ \Delta, v : A@w, \Delta';\ \Gamma \rhd \overset{\bullet}{v} : A@w \mid \overset{\circ}{v}}\ VarV$$

$$\frac{\Sigma;\ \Delta;\ \cdot \rhd M : A@w \mid s}{\Sigma;\ \Delta;\ \Gamma \rhd box_s\, M : [s]A@w \mid !s}\ \Box I$$

$$\frac{\Sigma;\ \Delta;\ \Gamma \rhd M : [s]A@w' \mid t \quad \Sigma \vdash w}{\Sigma;\ \Delta;\ \Gamma \rhd fetch[w']\, M : [s]A@w \mid fetch(t)}\ Fetch$$

$$\frac{\Sigma;\ \Delta;\ \Gamma \rhd M : [r]A@w \mid s \quad \Sigma;\ \Delta, v : A@w;\ \Gamma \rhd N : C@w \mid t}{\Sigma;\ \Delta;\ \Gamma \rhd unpack\, M\ to\ \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\ in\ N : C\{\overset{\circ}{v}/r\}@w \mid letc\ s\ be\ v : A\ in\ t}\ \Box E$$

Fig. 2. Term assignment for ILPnd.

The schemes $\supset I$ and $\supset E$ form abstractions and applications at the current world w. Applications of these schemes are reflected in their corresponding certificates. Scheme $\Box I$ states that if we have a typing derivation of M that does not depend on local assumptions (although it may depend on assumptions universally true) and s is a witness to this fact, then M is in fact executable at an arbitrary location. Thus a mobile unit $box_s\, M$ is introduced. The *Fetch* scheme types the *fetch* instruction. A term of the form $fetch[w']\, M$ at world w is typed by considering M at world w'. We are in fact assuming that w sees w' (or that w' is accessible from w) at run-time. Moreover, since the result of this instruction is to compute M at w' and then *return* the result to w (cf. Section 4), worlds w' and w are assumed interaccesible.[2] The *unpack* instruction is typed using the scheme $\Box E$. Suppose we are given a term N that computes some value of type C at world w and depends on a validity hypothesis $v : A@w$. Suppose we also have a term M that computes a mobile unit of type $[r]A@w$ at the same world w. Then $unpack\, M\ to\ \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\ in\ N$ is well-typed at w and computes a value of type $C\{\overset{\circ}{v}/r\}$. The certificate $letc\ s\ be\ v : A\ in\ t$ encodes the application of this scheme.

The following substitution principles reveal the true hypothetical nature of hypothesis, both for truth and for validity. Both are proved by induction on the derivation of the second judgement.

Lemma 3.1 (*Substitution Principle for Truth Hypothesis*)**.** *If* $\Sigma;\ \Delta;\ \Gamma_1, \Gamma_2 \rhd M : A@w \mid s$ *and* $\Sigma;\ \Delta;\ \Gamma_1, a : A@w, \Gamma_2 \rhd N : B@w' \mid t$ *are derivable, then so is* $\Sigma;\ \Delta;\ \Gamma_1, \Gamma_2 \rhd N\{a/M\} : B@w' \mid t\{a/s\}$.

Lemma 3.2 (*Substitution Principle for Validity Hypothesis*)**.** *If* $\Sigma;\ \Delta_1, \Delta_2;\ \cdot \rhd M : A@w \mid s$ *and* $\Sigma;\ \Delta_1, v : A@w, \Delta_2;\ \Gamma \rhd N : B@w' \mid t$ *are derivable, then so is* $\Sigma;\ \Delta_1, \Delta_2;\ \Gamma \rhd N\{\overset{\circ}{v}/s\}\{\overset{\bullet}{v}/M\} : B\{\overset{\circ}{v}/s\}@w \mid t\{\overset{\circ}{v}/s\}$.

Regarding the relation of this type system for $\lambda^{\mathsf{Cert}}_{\Box}$ with ILPnd we have the following result, which may be verified by structural induction on the derivation of the first judgement. Applications of the *Fetch* scheme become instances of the scheme $\frac{\mathcal{J}}{\mathcal{J}}$ with copies of identical judgements in ILPnd.

Lemma 3.3. *If* $\Sigma;\ \Delta;\ \Gamma \rhd A@w \mid s$ *is derivable, then so is* $\Delta';\ \Gamma' \rhd A' \mid s'$ *in ILPnd, where* Δ' *and* Γ' *result from* Δ *and* Γ*, respectively, by dropping all location qualifiers and* A' *and* s' *result from A and s, respectively, by replacing all occurrences of* $\overset{\bullet}{v}$ *and* $\overset{\circ}{v}$ *by* v *and all certificates of the form* $fetch(s)$ *with s.*

4. Operational semantics

The operational semantics of $\lambda^{\mathsf{Cert}}_{\Box}$ follows ideas from [19]. We introduce an abstract machine over a network of nodes. Nodes are named using worlds. Computation takes place sequentially – we are, in effect, modelling sequential programs that

[2] We consider a term assignment for a Nat. Ded. presentation of a refinement of S4 (and not S5; see Lemma 3.3). This reading, which suggests symmetry of the accessibility relation in a Kripke style model (and hence S5), is part of the run-time interpretation of terms (cf. 8).

$$
\begin{array}{lrcl}
\textit{Machine state} & \mathbb{N} & ::= & \mathbb{W};\, w : [k, M] \\
\textit{Network environment} & \mathbb{W} & ::= & \{w_1 : C_1, \ldots w_n : C_n\} \\
\textit{Current continuation} & k & ::= & \mathsf{return}\ w \mid \mathsf{finish} \mid k \lhd l \\
\textit{Continuation layer} & l & ::= & \circ\, N \mid V\ \circ\ \mid \mathit{unpack} \circ \mathit{to}\ \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\ \mathit{in}\ N \\
\textit{Continuation stack} & C & ::= & \epsilon \mid C :: k
\end{array}
$$

Fig. 3. $\lambda^{\mathsf{Cert}}_{\Box}$ run-time system syntax.

$$
\begin{array}{rcl}
\mathbb{W};\, w : [k, MN] & \xrightarrow{(1)} & \mathbb{W};\, w : [k \lhd \circ\, N, M] \\
\mathbb{W};\, w : [k \lhd \circ\, N, V] & \xrightarrow{(2)} & \mathbb{W};\, w : [k \lhd V\, \circ, N] \\
\mathbb{W};\, w : [k \lhd (\lambda a.M)\, \circ, V] & \xrightarrow{(3)} & \mathbb{W};\, w : [k, M\{a/V\}] \\
\mathbb{W};\, w : [k, \mathit{unpack}(M, v, N)] & \xrightarrow{(4)} & \mathbb{W};\, w : [k \lhd \mathit{unpack}(\circ, v, N), M] \\
\mathbb{W};\, w : [k \lhd \mathit{unpack}(\circ, v, N), \mathit{box}_s\, M] & \xrightarrow{(5)} & \mathbb{W};\, w : [k, N\{\overset{\circ}{v}/s\}\{\overset{\bullet}{v}/M\}] \\
\{w : C;\, w_s\};\, w : [k, \mathit{fetch}[w']\, M] & \xrightarrow{(6)} & \{w : C :: k;\, w_s\};\, w' : [\mathsf{return}\ w, M] \\
\{w : C :: k;\, w_s\};\, w' : [\mathsf{return}\ w, V] & \xrightarrow{(7)} & \{w : C;\, w_s\};\, w : [k, V\{w'/w\}]
\end{array}
$$

Fig. 4. Reduction schemes of $\lambda^{\mathsf{Cert}}_{\Box}$.

are aware of other worlds (other than their local host), rather than concurrent computation – at some designated world. An *abstract machine state* is an expression of the form $\mathbb{W};\, w : [k, M]$ (Fig. 3). The world w indicates the node where computation is currently taking place. M is the code that is being executed under local context k (M is the current *focus* of computation). The context k is a stack of terms with holes (written "$\circ$") that represent the layers of terms that are peeled out in order to access the redex. This representation ensures a reduction relation that always operates at the root of an expression and thus allows us to properly speak of an abstract machine. An alternative presentation based on a small or big-step semantics on terms, rather than machine states, is also possible. Continuing our explanation of the context k, it is a sequence of terms with holes ending in either return w or finish. return w indicates that once the term currently in focus is computed to a value, this value is to be returned to world w. The type system ensures that this value is, in effect, a mobile unit. If k takes the form finish, then the value of the term currently in focus is the end result of the computation. Finally, $k \lhd l$ states that the outermost peeled term layer is l. This latter expression may be of one of the following forms: $\circ\, N$ indicates a pending argument, $V\, \circ$ a pending abstraction (that V is an abstraction rather than a mobile unit is enforced by the type system) and $\mathit{unpack} \circ \mathit{to}\ \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\ \mathit{in}\ N$ a pending unpack body.

Finally, $\mathbb{W}$ is called a *network environment* and encodes the current state of execution at the remaining nodes of the network. The *domain of* $\mathbb{W}$ is the set of worlds to which it refers. Also, we sometimes refer to $\mathbb{W};\, k$ as the network environment.

The *initial* machine state (over $\Sigma = \{w_1, \ldots, w_n\}$) is $\mathbb{W};\, w : [\mathsf{finish}, M]$, where $\mathbb{W} = \{w_1 : \epsilon, \ldots, w_n : \epsilon\}$, $w \in \Sigma$ and M is any term. Similarly, the *terminal* machine state is one of the form $\mathbb{W};\, w : [\mathsf{finish}, V]$. Note that in a terminal state the focus of computation is a fully evaluated term (i.e. a value).

The operational semantics is presented by means of a small-step call-by-value reduction relation whose definition is given by the *reduction schemes* depicted in Fig. 4. Each reduction scheme is assigned an identifying number (indicated on top of each arrow) for easier reference. In (4) and (5) we write $\mathit{unpack}(M, v, N)$ and $\mathit{unpack}(\circ, v, N)$ as abbreviations for $\mathit{unpack}\, M\ \mathit{to}\ \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\ \mathit{in}\ N$ and $\mathit{unpack} \circ \mathit{to}\ \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\ \mathit{in}\ N$, resp. The scheme (1) selects the leftmost term in an application for reduction and pushes the pending part of the term (in this case the argument of the application) into the context. Once a value V is attained (which the type system, described below, will ensure to be an abstraction) the pending argument is popped off the context for reduction and V is pushed onto the context. Finally, when the argument has been reduced to a value, the pending abstraction is popped off the context and the beta reduct placed into focus for the next computation step. In the case that reduction encounters an *unpack* term, the argument M is placed into focus whilst the rest of the term is pushed onto the context. When reduction of the argument of an *unpack* computes a value, more precisely a mobile unit, the code and certificate components are extracted from it and substituted in the body of the *unpack* term. Note that the schemes presented up to this point all compute locally, we now address those that operate non-locally. If computation's focus is on a *fetch* instruction, then the execution context k is pushed onto the network environment for the current world w' and control transfers to world w. Moreover, focus of computation is now placed on the term M. Finally, the context of computation at w is set to return w thus ensuring that, once a value is computed, control transfers back to the caller. The latter is the rôle of the final reduction scheme.

5. Type soundness

This section addresses *progress* (well-typed, non-terminal machine states are not stuck) and *subject reduction* (well-typed machine states are closed under reduction). Recall from above that a machine state $\mathbb{N}$ is *terminal* if it is of the form

$$\frac{}{\Sigma \vdash \mathbb{W}\,;\, \text{finish} : A@w}\; C.Finish$$

$$\frac{\Sigma \vdash \mathbb{W}\,;\, k : B@w \quad \Sigma;\, \cdot\,;\, \cdot \rhd N : A@w \mid s}{\Sigma \vdash \mathbb{W}\,;\, k \lhd \circ\, N : A \supset B@w}\; C.Abs$$

$$\frac{\Sigma \vdash \mathbb{W}\,;\, k : B@w \quad \Sigma;\, \cdot\,;\, \cdot \rhd V : A \supset B@w \mid s}{\Sigma \vdash \mathbb{W}\,;\, k \lhd V\, \circ : A@w}\; C.App$$

$$\frac{\Sigma \vdash \mathbb{W}\,;\, k : B\{\overset{\circ}{v}/t\}@w \quad \Sigma;\, v : A;\, \cdot \rhd N : B@w \mid s}{\Sigma \vdash \mathbb{W}\,;\, k \lhd \mathit{unpack} \circ \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\, \mathit{in}\, N : [t]A@w}\; C.Box$$

$$\frac{\Sigma \vdash \{w' : C;\, w_s\}\,;\, k : A@w'}{\Sigma \vdash \{w' : C :: k;\, w_s\}\,;\, \text{return}\ w' : A@w}\; C.Return$$

$$\frac{!\Sigma = \{w_1, \ldots, w_n\} \quad \mathbb{W} = \{w_1 : C_1, \ldots w_n : C_n\} \qquad \Sigma;\, \cdot\,;\, \cdot \rhd M : A@w_j \mid s \qquad \Sigma \vdash \mathbb{W}\,;\, k : A@w_j}{\Sigma \vdash \mathbb{W}\,;\, w_j : [k, M]}\; MState$$

Fig. 5. Typing schemes for machine states.

$\mathbb{W}$; w : [finish, V]. It is *stuck* if it is not terminal and there is no $\mathbb{N}'$ such that $\mathbb{N} \longrightarrow \mathbb{N}'$. Two new judgements are introduced, machine state judgements and network environment judgements:

- $\Sigma \vdash \mathbb{W}\,;\, w_j : [k, M]$
- $\Sigma \vdash \mathbb{W}\,;\, k : A@w_j$

The first states that $\mathbb{W}$; w_j : $[k, M]$ is a well-typed machine state under the set of worlds Σ. The second states that the network environment together with the local context is well-typed under the set of worlds Σ.

A machine state is well-typed (Fig. 5) if the following three requirements hold. First $\mathbb{W}$ is a network environment with domain Σ. Second, M is closed, well-typed code at world w_j with certificate s that produces a value of type A, if at all. Finally, the network environment should be well-typed. The type of $\mathbb{W}$; finish has to be the type of the term currently in focus and located at the same world as indicated in the machine state. A network environment $\mathbb{W}$; $k \lhd \circ\, N$ is well-typed with type $A \supset B$ at world w under Σ, if the argument is well-typed with type A at w, and the network environment $\mathbb{W}$; k is well-typed with type B at the same world and under the same set of worlds. Note that $A \supset B$ is the type of the hole in the next term layer in k, and shall be completed by applying the term in focus to N. This is reminiscent of the left introduction scheme for implication in the Sequent Calculus presentation of Intuitionistic Propositional Logic. This connection is explored in detail in [13,10]. The *C.App* and *C.Box* schemes may be described in similar terms. Regarding the judgement $\Sigma \vdash \{w' : C :: k;\, w_s\}$; return $w' : A@w$, in order to verify that the type A at w of the value to be returned to world w' is correct, the context at w' must be checked, at w', to see if its outermost hole is indeed expecting a value of this type.

We now state the promised results. Both are proved by structural induction on the derivation of the judgement $\Sigma \vdash \mathbb{N}$. Together these results imply soundness of the reduction relation w.r.t. the type system: if a machine state is typable under Σ and is not terminal, then a well-typed value shall be attained.

Proposition 5.1 (*Progress*). *If $\Sigma \vdash \mathbb{N}$ is derivable and $\mathbb{N}$ is not terminal, then there exists $\mathbb{N}'$ such that $\mathbb{N} \longrightarrow \mathbb{N}'$.*

Proof. If $\Sigma \vdash \mathbb{N}$, then there exist A, s such that: (a) $\Sigma;\, \cdot\,;\, \cdot \rhd M : A@w \mid s$ and (b) $\Sigma \vdash \mathbb{W}\,;\, k : A@w$. From (a), $M \neq a, \overset{\bullet}{v}$ since M is closed. Thus we consider the remaining possibilities for M and k.

- M is a value $V = box_t\, P$ or $V = \lambda x.P$.
  - (1.1). $k =$ finish. $\mathbb{N}$ is terminal and hence the result holds.
  - (1.2). $k = k' \lhd \circ\, N$. By the machine reduction scheme (2), $\mathbb{N} \to \mathbb{W}\,;\, w : [k' \lhd V\, \circ, N]$.
  - (1.3). $k = k' \lhd V'\, \circ$. By the typing scheme *C.App* there exist B, t' such that $\Sigma;\, \cdot\,;\, \cdot \rhd V' : A \supset B@w \mid t'$. Therefore, from $\supset I$, $V' = \lambda b.N$. Finally, by reduction scheme (3), $\mathbb{N} \to \mathbb{W}\,;\, w : [k', N\{a/V\}]$.
  - (1.4). $k = k' \lhd \mathit{unpack} \circ \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\, \mathit{in}\, N$. By the typing scheme *C.Box* there exist t', A' such that $A = [t']A'$. Therefore $V = box_t\, P$. Finally, by reduction scheme (5), $\mathbb{N} \to \mathbb{W}\,;\, w : [k', N\{\overset{\circ}{v}/s\}\{\overset{\bullet}{v}/M\}]$.
  - (1.5). $k =$ return w'. By the typing scheme *C.Return*, $\mathbb{W} = \{w' : C :: k';\, w_s\}$. Therefore, from the reduction scheme (7), $\mathbb{N} \to \{w' : C;\, w_s\}\,;\, w' : [k', V]$.

- $M = P\,Q$. By reduction scheme (1), $\mathbb{N} \to \mathbb{W}\,;\, w : [k \lhd \circ\, Q, P]$.
- $M = \mathit{unpack}\, P\, \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\, \mathit{in}\, Q$. By reduction scheme (5), $\mathbb{N} \to \mathbb{W}\,;\, w : [k \lhd \mathit{unpack} \circ \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\, \mathit{in}\, Q, P]$.
- $M = \mathit{fetch}[w']\, P$. By reduction scheme (6), $\mathbb{N} \to \{w : C :: k;\, w_s\}\,;\, w' : [\mathrm{return}\ w, P]$. □

The proof of Subject Reduction relies on the following result whose proof is by induction on the derivation of $\Sigma;\, \Delta;\, \Gamma \rhd M : A@w'' \mid s$.

Lemma 5.1 (*Substitution Principle for Worlds*). *If $\Sigma;\, \Delta;\, \Gamma \rhd M : A@w'' \mid s$ and $\Sigma \vdash w'$,then $\Sigma;\, \Delta\{w/w'\};\, \Gamma\{w/w'\} \rhd M\{w/w'\} : A@w'\{w/w'\} \mid s$.*

Proposition 5.2 (*Subject Reduction*). *If $\Sigma \vdash \mathbb{N}$ is derivable and $\mathbb{N} \longrightarrow \mathbb{N}'$, then $\Sigma \vdash \mathbb{N}'$ is derivable.*

Proof. By case analysis on the reduction scheme applied. We present two sample cases, the remaining ones are developed along similar lines.

- Case (5). Suppose $\mathbb{N} = \mathbb{W}\,;\, w : [k \lhd \mathit{unpack} \circ \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\, \mathit{in}\, N, \mathit{box}_s\, M]$ and $\mathbb{N}' = \mathbb{W}\,;\, w : [k, N\{\overset{\circ}{v}/s\}\{\overset{\bullet}{v}/M\}]$. Assume, moreover, $\Sigma \vdash \mathbb{N}$. Then by *MState* there exist A', s' such that:

  (5.1) $\Sigma;\, \cdot\,;\, \cdot \rhd \mathit{box}_s\, M : A'@w \mid s'$

  (5.2) $\Sigma \vdash \mathbb{W}\,;\, k \lhd \mathit{unpack} \circ \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\, \mathit{in}\, N : A'@w$

  From (5.1) and by $\Box I$ there exist s, A such that $A' = [s]A$ and $s' = !s$:

  (5.3) $\Sigma;\, \cdot\,;\, \cdot \rhd \mathit{box}_s\, M : [s]A@w \mid !s$

  From (5.2) and by $C.\mathit{Box}$ there exist B, t such that:

  (5.4) $\Sigma \vdash \mathbb{W}\,;\, k : B@w$

  (5.5) $\Sigma;\, v : A;\, \cdot \rhd N : B\{\overset{\circ}{v}/s\}@w \mid t$

  From (5.3) and by $\Box I$, $\Sigma;\, \cdot\,;\, \cdot \rhd M : A@\omega \mid s$. From this, (5.6) and the Substitution Principle por Validity Hypothesis:

  $$\Sigma;\, \cdot\,;\, \cdot \rhd N\{\overset{\circ}{v}/s\}\{\overset{\bullet}{v}/M\} : \{\overset{\circ}{v}/s\}\{\overset{\circ}{v}/s\}@w \mid t\{\overset{\circ}{v}/s\}.$$

  Since $B\{\overset{\circ}{v}/s\}\{\overset{\circ}{v}/s\} = B$, from (5.4) and (5.6) by *MState*, $\Sigma \vdash \mathbb{W}\,;\, w : [k, N\{\overset{\circ}{v}/s\}\{\overset{\bullet}{v}/M\}]$.
- Case (7). Suppose $\mathbb{N} = \{w : C :: k;\, \mathrm{w}_s\}\,;\, w' : [\mathrm{return}\ w, V]$ and $\mathbb{N}' = \{w : C;\, \mathrm{w}_s\}\,;\, w : [k, V\{w'/w\}]$. Assume, moreover, $\Sigma \vdash \mathbb{N}$. Then by *MState*, there exist A', s' such that:

  (7.1) $\Sigma \vdash \{w : C :: k;\, \mathrm{w}_s\}\,;\, \mathrm{return}\ w : A'@w'$

  (7.2) $\Sigma;\, \cdot\,;\, \cdot \rhd V : A'@w' \mid s'$

  From (7.1) and by $C.\mathit{Return}$, $\Sigma \vdash \{w : C;\, \mathrm{w}_s\}\,;\, k : A'@w$. Also, from (7.2) and by the Substitution Principle for Worlds, $\Sigma;\, \cdot\,;\, \cdot \rhd V\{w'/w\} : A'@w \mid s'$. Finally, from these last two judgements and *MState*, $\Sigma \vdash \{w : C;\, \mathrm{w}_s\}\,;\, w : [k, V\{w'/w\}]$. □

Notice that although certificates are substituted and combined at run-time they are *not* checked at run-time. Indeed, it is the type system that guarantees *statically* that all generated certificates will be correctly constructed in the sense that if a value $\mathit{box}_s\, M$ is ever produced at run-time, then s is an encoding of a typing derivation for M. Information on certificates in the run-time system is required, however, in order to formulate Subject Reduction.

Corollary 5.1 (*Safety*). *Suppose $\Sigma \vdash \mathbb{N}$ is derivable and $\mathbb{N} \longrightarrow^* \mathbb{W}; w : [k, \mathit{box}_s\, M]$, then $\Sigma;\, \cdot\,;\, \cdot \rhd M : A@w \mid s$.*

This result is an immediate consequence of Subject Reduction.

6. Strong normalization

We prove strong normalization (SN) of machine reduction by translating machine states to terms of the simply typed lambda calculus with unit type ($\lambda^{\mathbf{1},\to}$). For technical reasons (which we comment on shortly) we shall consider the following modification of the machine reduction semantics of $\lambda^{\mathsf{Cert}}_{\Box}$ obtained by replacing the reduction scheme:

$$(2) \quad \mathbb{W};\, w : [k \lhd \circ\, N, V] \quad \longrightarrow \quad \mathbb{W};\, w : [k \lhd V \circ, N]$$

by the following two new reduction schemes:

$$(2.1) \quad \mathbb{W};\, w : [k \lhd \circ\, N, V] \quad \longrightarrow \quad \mathbb{W};\, w : [k \lhd V \circ, N],\ N \text{ is not a value}$$

$$(2.2) \quad \mathbb{W};\, w : [k \lhd \circ\, V, \lambda a.M] \quad \longrightarrow \quad \mathbb{W};\, w : [k, M\{a/V\}]$$

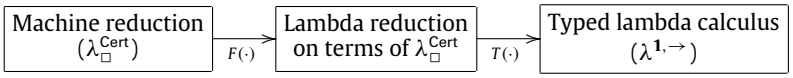
These schemes result from refining (2) by inspecting its behaviour in any non-terminating reduction sequence. If N happens to be a value, then each (2) step is followed by a (3) step. The juxtaposition of these two steps gives precisely (2.2). The reduction scheme (2.1) is just (2) when N is not a value. It is clear that every non-terminating reduction sequence in the original formulation can be mimicked by a non-terminating reduction sequence in the modified semantics in such a way that each (2) step

- either it is not followed by a (3) step and thus becomes a (2.1) step or
- it is followed by a (3) step and hence (2) followed by (3) become one (2.2) step.

$$
\begin{aligned}
F(\mathbb{W};\, w : [\mathrm{finish}, M]) &\stackrel{\mathrm{def}}{=} \overline{M} \\
F(\mathbb{W};\, w : [k \lhd \circ\, N, M]) &\stackrel{\mathrm{def}}{=} F(\mathbb{W};\, w : [k, M\, N]) \\
F(\mathbb{W};\, w : [k \lhd V\, \circ, N]) &\stackrel{\mathrm{def}}{=} F(\mathbb{W};\, w : [k, V\, N]) \\
F(\mathbb{W};\, w : [k \lhd \mathit{unpack} \circ \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\, \mathit{in}\, N, M]) &\stackrel{\mathrm{def}}{=} \\
F(\mathbb{W};\, w : [k, \mathit{unpack}\, M\, \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\, \mathit{in}\, N]) & \\
F(\{w : C :: k;\, w_s\};\, w' : [\mathrm{return}\, w, M]) &\stackrel{\mathrm{def}}{=} F(\{w : C;\, w_s\};\, w : [k, M])
\end{aligned}
$$

Fig. 6. Mapping from machine states in $\lambda_{\Box}^{\mathsf{Cert}}$ to terms in $\lambda_{\Box}^{\mathsf{Cert}}$.

Therefore, it suffices to prove SN of the modified system in order to deduce the same property for our original formulation. This is done in two phases, depicted below:



First we relate machine reduction with a notion of reduction that operates directly on lambda terms via a mapping $F(\cdot)$ (Fig. 6). Then we relate the latter with reduction in $\lambda^{\mathbf{1},\to}$ via a mapping $T(\cdot)$ (Fig. 7).

6.1. From machine reduction in $\lambda_{\Box}^{\mathsf{Cert}}$ to lambda reduction in $\lambda_{\Box}^{\mathsf{Cert}}$

The mapping $F(\cdot)$ flattens out the local context of a machine state in order to produce a term of $\lambda_{\Box}^{\mathsf{Cert}}$ and replaces all worlds by a distinguished world "$\bullet$" whose name is irrelevant. We write $\overline{M}$ for $M\{w_1/\bullet\} \ldots \{w_n/\bullet\}$, the result of replacing all worlds in M with $\bullet$. The mapping is injective on typable machine states whose focus of computation differ only in the names of the worlds that occur in them.

Lemma 6.1. *Let $\mathbb{N}$ be $\mathbb{W}\,;\, w : [k, M]$. If $\Sigma \vdash \mathbb{N}$ and $\overline{M} = \overline{M'}$ then $F(\mathbb{W}\,;\, w : [k, M]) = F(\mathbb{W}\,;\, w : [k, M'])$.*

This property is required for proving type preservation of $F(\cdot)$ (it is also used in Lemma 6.4), the result we present next. In its statement we assume $\bullet$ belongs to Σ. It is proved by induction on the pair $\langle |\mathbb{W}|, k \rangle$, where $|\mathbb{W}|$, the size of $\mathbb{W}$, is the sum of the length of the context stacks of all worlds in its domain: $|\{w_1 : C_1, \ldots, w_n : C_n\}| \stackrel{\mathrm{def}}{=} \sum_{i=1}^{n} |C_i|$ and $|\epsilon| \stackrel{\mathrm{def}}{=} 0$ and $|C :: k| \stackrel{\mathrm{def}}{=} 1 + |C|$.

Lemma 6.2. *Let $\mathbb{N}$ be $\mathbb{W};\, w : [k, M]$. If $\Sigma \vdash \mathbb{N}$ is derivable and $\Sigma \vdash \bullet$, then there exist A and s such that $\Sigma; \cdot; \cdot \rhd F(\mathbb{N}) : A@\bullet \mid s$ is derivable.*

In order to relate machine reduction in $\lambda_{\Box}^{\mathsf{Cert}}$ with reduction in $\lambda^{\mathbf{1},\to}$ we introduce *lambda reduction*. These schemes are standard except for the last one which states that *fetch* terms have no computational effect at the level of lambda terms. It should be mentioned that strong lambda reduction is considered (i.e. reduction under all term constructors).

Definition 6.1 (*Lambda Reduction for $\lambda_{\Box}^{\mathsf{Cert}}$*)**.** Lambda reduction is the contextual closure of the following reduction axioms:

$$
\begin{aligned}
(\lambda a.M)\, N &\longrightarrow_{\beta} M\{a/N\} \\
\mathit{unpack}\, \mathit{box}_s\, M\, \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v} \rangle\, \mathit{in}\, N &\longrightarrow_{\beta_{\Box}} N\{\overset{\bullet}{v}/M\}\{\overset{\circ}{v}/s\} \\
\mathit{fetch}[w]\, M &\longrightarrow_{\mathit{ftch}} M
\end{aligned}
$$

Before addressing the property that $F(\cdot)$ preserves abstract machine reduction (cf. Lemma 6.4) we mention an additional property which will be required for the proof of this fact. It states that $F(\cdot)$ preserves lambda reduction over terms that are placed in any network environment and context that yields typable machine states.

Lemma 6.3. *If $M \longrightarrow_{\beta,\beta_{\Box},\mathit{ftch}} M'$ and $\Sigma \vdash \mathbb{W}\,;\, w : [k, M]$ is derivable, then $F(\Sigma \vdash \mathbb{W}\,;\, w : [k, M]) \longrightarrow_{\beta,\beta_{\Box},\mathit{ftch}} F(\Sigma \vdash \mathbb{W}\,;\, w : [k, M'])$.*

We can now establish that the flattening map is also reduction preserving.

Lemma 6.4. 1. *If $\mathbb{N} \longrightarrow_{1,2.1,4,7} \mathbb{N}'$, then $F(\mathbb{N}) = F(\mathbb{N}')$.*
2. *If $\mathbb{N} \longrightarrow_{2.2,3,5,6} \mathbb{N}'$, then $F(\mathbb{N}) \longrightarrow_{\beta,\beta_{\Box},\mathit{ftch}} F(\mathbb{N}')$.*

Proof. Both items are proved by case analysis on the reduction scheme applied.

$$
\begin{array}{rcl}
T(a) & \stackrel{\text{def}}{=} & a \\
T(\overset{\bullet}{v}) & \stackrel{\text{def}}{=} & v\ unit \\
T(\lambda a.M) & \stackrel{\text{def}}{=} & \lambda a.T(M) \\
T(M\,N) & \stackrel{\text{def}}{=} & T(M)\,T(N) \\
T(box_s\,M) & \stackrel{\text{def}}{=} & \lambda a.T(M),\ a \text{ fresh of type } \mathbf{1} \\
T(unpack\,M\ to\ \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\ in\ N) & \stackrel{\text{def}}{=} & (\lambda v.T(N))\,T(M) \\
T(fetch[w]\,M) & \stackrel{\text{def}}{=} & (\lambda a.a)\,T(M)
\end{array}
$$

Fig. 7. Mapping from types and terms in $\lambda^{\text{Cert}}_{\Box}$ to $\lambda^{\mathbf{1},\to}$.

1. In the case of (1), (2.1) and (4), the result is immediate from the definition of $F(\cdot)$. Regarding (7), $\mathbb{N} = \{w : C :: k;\ w_s\};\ w' : [\text{return}\ w, V] \longrightarrow \{w : C;\ w_s\};\ w : [k, V\{w'/w\}] = \mathbb{N}'$. We reason as follows:

$$
\begin{array}{lcl}
F(\{w : C :: k;\ w_s\};\ w' : [\text{return}\ w, V]) & = & \\
F(\{w : C;\ w_s\};\ w : [k, V]) & = & \text{by Lemma 6.1} \\
F(\{w : C;\ w_s\};\ w : [k, V\{w'/w\}]) & = &
\end{array}
$$

2. Since case (3) is similar to (2.2) we develop all but the former:
   - Case (2.2). Suppose that $\mathbb{N} = \mathbb{W};\ w : [k \lhd \circ\, V, \lambda a.M] \longrightarrow \mathbb{W};\ w : [k, M\{a/V\}] = \mathbb{N}'$. Then $F(\mathbb{N}) = F(\mathbb{W};\ w : [k, (\lambda a.M)\,V])$. Since $(\lambda a.M)\,V \longrightarrow_{\beta} M\{a/V\}$, by Lemma 6.3:

$$F(\mathbb{W};\ w : [k, (\lambda a.M)\,V]) \longrightarrow_{\beta,\beta_{\Box},ftch} F(\mathbb{W};\ w : [k, M\{a/V\}])$$

   - Case (5). Suppose $\mathbb{N} = \mathbb{W};\ w : [k \lhd unpack \circ to\ \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\ in\ N, box_s\,M] \longrightarrow \mathbb{W};\ w : [k, N\{\overset{\circ}{v}/s\}\{\overset{\bullet}{v}/M\}] = \mathbb{N}'$. Then we have $F(\mathbb{N}) = F(\mathbb{W};\ w : [k, unpack\,box_s\,M\ to\ \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\ in\ N])$. We resort to the reduction step $unpack\,box_s\,M\ to\ \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\ in\ N \longrightarrow_{\beta_{\Box}} N\{\overset{\circ}{v}/s\}\{\overset{\bullet}{v}/M\}$ and then Lemma 6.3 to deduce:

$$F(\mathbb{W};\ w : [k, unpack\,box_s\,M\ to\ \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\ in\ N]) \longrightarrow_{\beta,\beta_{\Box},ftch} F(\mathbb{W};\ w : [k, N\{\overset{\circ}{v}/s\}\{\overset{\bullet}{v}/M\}])$$

   - Case (6). Suppose $\mathbb{N} = \{w : C;\ w_s\};\ w : [k, fetch[w']\,M] \longrightarrow \{w : C :: k;\ w_s\};\ w' : [\text{return}\ w, M] = \mathbb{N}'$. Since $fetch[w']\,M \longrightarrow_{ftch} M$, then by Lemma 6.3:

$$F(\{w : C;\ w_s\};\ w : [k, fetch[w']\,M]) \longrightarrow_{\beta,\beta_{\Box},ftch} F(\{w : C;\ w_s\};\ w : [k, M])$$

     We are left to verify that:

$$F(\{w : C :: k;\ w_s\};\ w' : [\text{return}\ w, M]) = F(\{w : C;\ w_s\};\ w : [k, M])$$

     This follows from the definition of $F(\cdot)$. □

6.2. From lambda reduction in $\lambda^{\text{Cert}}_{\Box}$ to reduction in $\lambda^{\mathbf{1},\to}$

$\lambda^{\mathbf{1},\to}$ (cf. Section 11.2 of [20]) is the standard simply typed lambda calculus with an additional type constructor **1** and an additional term *unit* of type **1**. The second part of the proof consists in relating lambda reduction in $\lambda^{\text{Cert}}_{\Box}$ with reduction in $\lambda^{\mathbf{1},\to}$. As mentioned, for that we introduce a mapping $T(\cdot)$ that associates types and terms in $\lambda^{\text{Cert}}_{\Box}$ with types and terms in $\lambda^{\mathbf{1},\to}$. Function types are mapped to function types and the modal type $[s]A$ is mapped to functional types whose domain is the unit type **1** and whose co-domain is the mapping of A.

$$
\begin{array}{rcl}
T(P) & \stackrel{\text{def}}{=} & P \\
T(A \supset B) & \stackrel{\text{def}}{=} & T(A) \supset T(B) \\
T([s]A) & \stackrel{\text{def}}{=} & \mathbf{1} \supset T(A)
\end{array}
$$

Mapping of terms (Fig. 7) is straightforward given that on types; the case for *fetch* guarantees that each $\longrightarrow_{ftch}$ step is mapped to a non-empty step in $\lambda^{\mathbf{1},\to}$. $T(\cdot)$ over terms is both type preserving and reduction preserving.

Lemma 6.5. *If $\Sigma;\ \Delta;\ \Gamma \rhd M : A@w \mid s$ is derivable in $\lambda^{\text{Cert}}_{\Box}$, then $\Delta',\ \Gamma' \rhd T(M) : T(A)$ is derivable in $\lambda^{\mathbf{1},\to}$, where*

1. *Γ' results from replacing each hypothesis $a : A@w$ by $a : T(A)$ and*
2. *Δ' results from replacing each hypothesis $v : A@w$ by $v : \mathbf{1} \supset T(A)$.*

Proof. By induction on the derivation of $\Sigma;\ \Delta;\ \Gamma \rhd M : A@w \mid s$. The base cases are straightforward. We include some sample inductive cases.

- Case $\Box I$. The derivation ends in:

$$\frac{\Sigma;\ \Delta;\ \cdot \rhd M : A@w \mid s}{\Sigma;\ \Delta;\ \Gamma \rhd box_s\, M : [s]A@w \mid !s}\ \Box I$$

From the IH $\Delta' \rhd T(M) : T(A)$ is derivable. From weakening in $\lambda^{\mathbf{1},\to}$ we obtain $\Delta', a : \mathbf{1} \rhd T(M) : T(A)$. Thus $\Delta' \rhd \lambda a.T(M) : \mathbf{1} \supset T(A)$ is derivable.

- Case *Fetch*. The derivation ends in:

$$\frac{\Sigma;\ \Delta;\ \Gamma \rhd M : [s]A@w' \mid t \quad \Sigma \vdash w}{\Sigma;\ \Delta;\ \Gamma \rhd fetch[w']\, M : [s]A@w \mid fetch(t)}\ Fetch$$

Then, if we let B stand for the type $\mathbf{1} \supset T(A)$, we can derive:

$$\frac{\Delta',\ \Gamma' \rhd \lambda a.a : B \supset B \quad \Delta',\ \Gamma' \rhd T(M) : B}{\Delta',\ \Gamma' \rhd (\lambda a.a)\, T(M) : B}\ \supset E$$

- Case $\Box E$. The derivation ends in:

$$\frac{\Sigma;\ \Delta;\ \Gamma \rhd M : [r]A@w \mid s \quad \Sigma;\ \Delta,\ v : A@w;\ \Gamma \rhd N : C@w \mid t}{\Sigma;\ \Delta;\ \Gamma \rhd unpack\, M\ to\ \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\ in\ N : C\{\overset{\circ}{v}/r\}@w \mid letc\ s\ be\ v : A\ in\ t}\ \Box E$$

From the IH, $\Delta',\ \Gamma' \rhd T(M) : \mathbf{1} \supset T(A)$ and also $\Delta',\ v : \mathbf{1} \supset T(A),\ \Gamma' \rhd T(N) : T(C)$. We conclude as follows, noting that $T(C) = T(C\{\overset{\circ}{v}/r\})$ and resorting, once again, to B as shorthand for the type $\mathbf{1} \supset T(A)$:

$$\frac{\dfrac{\Delta',\ v : B,\ \Gamma' \rhd T(N) : T(C)}{\Delta',\ \Gamma' \rhd \lambda v.T(N) : B \supset T(C)} \qquad \Delta',\ \Gamma' \rhd T(M) : B}{\Delta',\ \Gamma' \rhd (\lambda v.T(N))\, T(M) : T(C)}\ \supset E \quad \Box$$

It is standard when proving that a mapping is reduction preserving over some set of terms in a programming language based on some variant of the lambda calculus to require that this mapping also commute with substitution. Since we have substitution over both local variables and validity variables we need two such results. The first one holds: $T(M)\{a/T(N)\} = T(M\{a/N\})$. However, the second one fails. Indeed, T does not commute with substitution of (the mapping of) validity variables (i.e. $T(M)\{v/T(N)\} \neq T(M\{v/N\})$; take $M = \overset{\bullet}{v}$). However, the following does hold and suffices for our purposes: $T(M)\{v/\lambda a.T(N)\} \longrightarrow_\beta^* T(M\{\overset{\bullet}{v}/N\}\{\overset{\circ}{v}/s\})$. The arrow $\longrightarrow_\beta^*$ denotes the reflexive, transitive closure of $\longrightarrow_\beta$ while $\longrightarrow_\beta^+$ (below) denotes its transitive closure.

Lemma 6.6. *If* $M \longrightarrow_{\beta,\beta_\Box,ftch} N$, *then* $T(M) \longrightarrow_\beta^+ T(N)$.

Proof. By induction on M. The base cases are trivial since no reduction steps may originate from a or $\overset{\bullet}{v}$. The inductive cases follow from the fact that reduction is closed under all constructors in $\lambda^{\mathbf{1},\to}$. We illustrate the cases where reduction takes place at the root of M.

- Case $\longrightarrow_\beta$. Suppose $(\lambda a.M)\, N \longrightarrow_\beta M\{a/N\}$. We reason as follows:

$$\begin{array}{lll} = & T((\lambda a.M)\, N) & \\ = & T(\lambda a.M)\, T(N) & (\text{Def. of } T(\cdot)) \\ = & (\lambda a.T(M))\, T(N) & (\text{Def. of } T(\cdot)) \\ \longrightarrow_\beta & T(M)\{a/T(N)\} & \\ = & T(M\{a/N\}) & \end{array}$$

- Case $\longrightarrow_{\beta_\Box}$. Suppose $unpack\, box_s\, M\ to\ \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\ in\ N \longrightarrow_{\beta_\Box} N\{\overset{\bullet}{v}/M\}\{\overset{\circ}{v}/s\}$. We reason as follows:

$$\begin{array}{lll} & T(unpack\, box_s\, M\ to\ \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\ in\ N) & \\ = & (\lambda v.T(N))\, T(box_s\, M) & (\text{Def. of } T(\cdot)) \\ = & (\lambda v.T(N))\, \lambda a.T(M) & (\text{Def. of } T(\cdot)) \\ \longrightarrow_\beta & T(N)\{v/\lambda a.T(M)\} & \\ \longrightarrow_\beta^* & T(N\{\overset{\bullet}{v}/M\}\{\overset{\circ}{v}/s\}) & \end{array}$$

- Case $\longrightarrow_{ftch}$. Suppose $fetch[w]\, M \longrightarrow_{ftcs} M$. Then $T(fetch[w]\, M) = (\lambda a.a)\, T(M) \longrightarrow_\beta T(M)$. $\Box$

6.3. Combining the results

Our desired result (Proposition 6.1) may be proved by contradiction as follows. Assume that $\longrightarrow_{1,2.1,4,7}$ reduction is SN. Suppose, also, that there is an infinite reduction sequence starting from a machine state $\mathbb{N}_1$. From our assumption this sequence must have an infinite number of interspersed $\longrightarrow_{2.2,3,5,6}$ reduction steps:

$$\mathbb{N}_1 \longrightarrow^*_{1,2.1,4,7} \mathbb{N}_2 \longrightarrow_{2.2,3,5,6} \mathbb{N}_3 \longrightarrow^*_{1,2.1,4,7} \mathbb{N}_4 \longrightarrow_{2.2,3,5,6} \mathbb{N}_5 \longrightarrow^*_{1,2.1,4,7} \mathbb{N}_6 \longrightarrow_{2.2,3,5,6} \cdots$$

Then (Lemma 6.4) we have the following lambda reduction sequence over typable terms (Lemma 6.2):

$$F(\mathbb{N}_1) = F(\mathbb{N}_2) \longrightarrow_{\beta,\beta_\square,ftch} F(\mathbb{N}_3) = F(\mathbb{N}_4) \longrightarrow_{\beta,\beta_\square,ftch} F(\mathbb{N}_5) = F(\mathbb{N}_6) \longrightarrow_{\beta,\beta_\square,ftch} \cdots$$

Finally, we arrive at the following infinite reduction sequence (Lemma 6.6) of typable terms (Lemma 6.5) in $\lambda^{\mathbf{1},\rightarrow}$, thus contradicting SN of $\lambda^{\mathbf{1},\rightarrow}$:

$$T(F(\mathbb{N}_1)) \longrightarrow^+_\beta T(F(\mathbb{N}_3)) \longrightarrow^+_\beta T(F(\mathbb{N}_5)) \longrightarrow^+_\beta \cdots$$

In order to complete our proof we now address our claim, namely that $\longrightarrow_{1,2.1,4,7}$ reduction is SN. It is the proof of this result that has motivated the modified reduction semantics presented at the beginning of this section. Every (2) step followed by a (3) step will be mapped to a β step in $\lambda^{\mathbf{1},\rightarrow}$ (and hence contributes to the $\lambda^{\mathbf{1},\rightarrow}$ reduction sequence). However, those (2) steps that are not do not contribute to this reduction sequence. This splitting of (2) allows us to show that, in fact, the latter may be safely ignored.

First a simple yet useful result for proving SN of combinations of binary relations that we have implicitly made use of above.

Lemma 6.7. *Let $\longrightarrow_1$ and $\longrightarrow_2$ be binary relations over some set X. Suppose*

1. *$\longrightarrow_1$ is SN and*
2. *$\mathcal{M}$ is a mapping from X to some well-founded set such that*
   (a) *$x \longrightarrow_1 y$ implies $\mathcal{M}(x) = \mathcal{M}(y)$*
   (b) *$x \longrightarrow_2 y$ implies $\mathcal{M}(x) > \mathcal{M}(y)$*

Then $\longrightarrow_1 \cup \longrightarrow_2$ is SN.

Before we use this lemma for our proof of SN of $\longrightarrow_{1,2.1,4,7}$, some definitions are required. The size of a term M, written $|M|$, is defined as the number of variables and constructors in M:

$$\begin{aligned} |a| = |\overset{\bullet}{v}| &\stackrel{\text{def}}{=} 1 \\ |\lambda a.M| = |box_s\, M| &\stackrel{\text{def}}{=} |M| + 1 \\ |M\,N| = |unpack\, M\ to\ \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\ in\ N| &\stackrel{\text{def}}{=} |M| + |N| + 1 \\ |fetch[w]\, M| &\stackrel{\text{def}}{=} |M| + 1 \end{aligned}$$

The size of a context k, written $|k|$, is defined by taking the sum of the sizes of the terms with holes, where each hole counts as 1:

$$\begin{aligned} |\text{return } w| &\stackrel{\text{def}}{=} 1 & |\circ\ N| &\stackrel{\text{def}}{=} |N| + 1 \\ |\text{finish}| &\stackrel{\text{def}}{=} 1 & |V\ \circ| &\stackrel{\text{def}}{=} |V| + 1 \\ |k \lhd l| &\stackrel{\text{def}}{=} |k| + |l| & |unpack \circ to \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\ in\ N| &\stackrel{\text{def}}{=} |N| + 1 \end{aligned}$$

We will write $|k, M|$ to abbreviate $|k| + |M|$.

Lemma 6.8. *$\longrightarrow_{1,2.1,4,7}$ reduction is SN.*

Proof. First we prove SN of schemes (1) and (4). Then we conclude by resorting to Lemma 6.7, introducing a measure $\mathcal{M}_2$ such that:

1. $\mathbb{N} \longrightarrow_{1,4} \mathbb{N}'$ implies $\mathcal{M}_2(\mathbb{N}) = \mathcal{M}_2(\mathbb{N}')$ and
2. $\mathbb{N} \longrightarrow_{2.1,7} \mathbb{N}'$ implies $\mathcal{M}_2(\mathbb{N}) > \mathcal{M}_2(\mathbb{N}')$.

SN of schemes (1) and (4) follows from noting that the following measure $\mathcal{M}_1$ of machine states over pairs of natural numbers (ordered lexicographically) strictly decreases when schemes (1) and (4) are applied[3]:

$$\mathcal{M}_1(\mathbb{W}; w : [k, M]) \stackrel{\text{def}}{=} \langle |\mathbb{W}|, |M| \rangle$$

[3] It also decreases when (7) is applied. However, it does not necessarily decrease when (2.1) is applied.

We are left to verify that the following measure $\mathcal{M}_2$ enjoys the required properties stated above:

$$\mathcal{M}_2(\mathbb{W};\, w : [k, M]) \stackrel{\text{def}}{=} \langle |\mathbb{W}|, |k, M| - \mathit{len}(k) - m(M)\rangle$$

where $\mathit{len}(k)$ is the length of k and m is the following mapping from *closed* terms to positive integers:

$$\begin{aligned} m(V) &\stackrel{\text{def}}{=} 0 \\ m(M\,N) &\stackrel{\text{def}}{=} 1 + m(M) \\ m(\mathit{unpack}\, M\, \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\, \mathit{in}\, N) &\stackrel{\text{def}}{=} 1 + m(M) \\ m(\mathit{fetch}[w]\, M) &\stackrel{\text{def}}{=} 1 \end{aligned}$$

This measure decreases strictly for both (2.1) and (7), whereas it yields equal numbers for (1) and (4).

- Case (1).

$$\begin{aligned} & \mathcal{M}_2(\mathbb{W};\, w : [k, M\,N]) \\ = \;& \langle |\mathbb{W}|, |k, M\,N| - \mathit{len}(k) - m(M\,N)\rangle \\ = \;& \langle |\mathbb{W}|, |k, M\,N| - \mathit{len}(k) - 1 - m(M)\rangle \\ = \;& \langle |\mathbb{W}|, |k \lhd \circ\, N, M| - \mathit{len}(k) - 1 - m(M)\rangle \\ = \;& \mathcal{M}_2(\mathbb{W};\, w : [k \lhd \circ\, N, M]) \end{aligned}$$

- Case (2.1). Recall from above that N is not a value. Therefore, it is either an application, an unpack term or a fetch term. Note that for each of these $m(N) > 0$. Therefore, we reason as follows:

$$\begin{aligned} & \mathcal{M}_2(\mathbb{W};\, w : [k \lhd \circ\, N, V]) \\ = \;& \langle |\mathbb{W}|, |k \lhd \circ\, N, V| - \mathit{len}(k) - 1 - m(V)\rangle \\ = \;& \langle |\mathbb{W}|, |k, N, V| + 1 - \mathit{len}(k) - 1\rangle \\ > \;& \langle |\mathbb{W}|, |k, V, N| + 1 - \mathit{len}(k) - 1 - m(N)\rangle \\ = \;& \langle |\mathbb{W}|, |k \lhd V\, \circ, N| - \mathit{len}(k) - 1 - m(N)\rangle \\ = \;& \mathcal{M}_2(\mathbb{W};\, w : [k \lhd V\, \circ, N]) \end{aligned}$$

- Case (4).

$$\begin{aligned} & \mathcal{M}_2(\mathbb{W};\, w : [k, \mathit{unpack}\, M\, \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\, \mathit{in}\, N]) \\ = \;& \langle |\mathbb{W}|, |k, \mathit{unpack}\, M\, \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\, \mathit{in}\, N| - \mathit{len}(k) - m(\mathit{unpack}\, M\, \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\, \mathit{in}\, N)\rangle \\ = \;& \langle |\mathbb{W}|, |k, \mathit{unpack}\, M\, \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\, \mathit{in}\, N| - \mathit{len}(k) - 1 - m(M)\rangle \\ = \;& \langle |\mathbb{W}|, |k \lhd \mathit{unpack} \circ\, \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\, \mathit{in}\, N, M| - \mathit{len}(k) - 1 - m(M)\rangle \\ = \;& \mathcal{M}_2(\mathbb{W};\, w : [k \lhd \mathit{unpack} \circ\, \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\, \mathit{in}\, N, M]) \end{aligned}$$

- Case (7). Let $n = |\{w : C :: k;\, w_s\}|$.

$$\begin{aligned} & \mathcal{M}_2(\{w : C :: k;\, w_s\};\, w' : [\mathrm{return}\, w, V]) \\ = \;& \langle n, |\mathrm{return}\, w| + |V| - \mathit{len}(\mathrm{return}\, w) - m(V)\rangle \\ = \;& \langle n, 1 + |V| - 1 - m(V)\rangle \\ = \;& \langle n, |V|\rangle \\ = \;& \langle n, |V\{w'/w\}|\rangle \\ > \;& \langle n - 1, |k, V\{w'/w\}| - \mathit{len}(k)|\rangle \\ = \;& \langle n - 1, |k, V\{w'/w\}| - \mathit{len}(k) - m(V\{w'/w\})|\rangle \\ = \;& \mathcal{M}_2(\{w : C;\, w_s\};\, w : [k, V\{w'/w\}]) \quad \square \end{aligned}$$

We can finally state our desired result, whose proof we have presented above.

Proposition 6.1. $\longrightarrow$ *is SN.*

7. Examples

Consider the following $\lambda^{\mathsf{Cert}}_{\Box}$ expression encoding a proof of the IJL axiom scheme $[s](A \supset B) \supset [t]A \supset [s \cdot t]B$ (where s, t are any proof term expressions and A, B any propositions):

$$\lambda a.\lambda b.\mathit{unpack}\, a\, \mathit{to}\, \langle \overset{\bullet}{u}, \overset{\circ}{u}\rangle\, \mathit{in}\, (\mathit{unpack}\, b\, \mathit{to}\, \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\, \mathit{in}\, (\mathit{box}^{\circ}_{\overset{\circ}{u}\cdot\overset{\circ}{v}}\, \overset{\bullet}{u}\, \overset{\bullet}{v}))$$

This is read as follows: *"Given a mobile unit a and a mobile unit b, extract code* $\overset{\bullet}{v}$ *and certificate* $\overset{\circ}{v}$ *from b and extract code* $\overset{\bullet}{u}$ *and certificate* $\overset{\circ}{u}$ *from a. Then create new code* $\overset{\bullet}{u}\,\overset{\bullet}{v}$ *by applying* $\overset{\bullet}{u}$ *to* $\overset{\bullet}{v}$ *and a new certificate for this code* $\overset{\circ}{u} \cdot \overset{\circ}{v}$*. Finally, wrap both*

of these up into a new mobile unit". The new mobile unit is created at the some current (implicit) world w. Moreover, the example assumes that both a and b reside at w. The following variant M illustrates the case where mobile units a and b reside at worlds w_a and w_b which are assumed different from the current world w:

$$\mathit{unpack}\, \mathit{fetch}[w_a]\, a\ \mathit{to}\ \langle \overset{\bullet}{u}, \overset{\circ}{u}\rangle\ \mathit{in}\ (\mathit{unpack}\, \mathit{fetch}[w_b]\, b\ \mathit{to}\ \langle \overset{\bullet}{v}, \overset{\circ}{v}\rangle\ \mathit{in}\ (\mathit{box}_{\overset{\circ}{u}\cdot\overset{\circ}{v}}\ \overset{\bullet}{u}\ \overset{\bullet}{v}))$$

Here the expression $\mathit{fetch}[w_a]\, a$ is operationally interpreted as a remote call to compute the value of a (a mobile unit) at w_a and then return it to the current world. Note that a and b occur free in this expression. Since b is a non-local resource it cannot be bound straightforwardly by prefixing the above term with λb. Rather, the code first must be moved from the current world w to w_b; similarly for a:

$$\lambda a.\mathit{fetch}[w_b]\ (\lambda b.\mathit{fetch}[w]\, M)$$

The next example is a function that builds a certified mobile "plus k" function. It receives a mobile unit for computing a natural number, unpacks it in order to obtain the code $\overset{\bullet}{u}$ and the certificate $\overset{\circ}{u}$ and then builds a new mobile unit. This new mobile unit has a function as code. This function, when given a natural number b, adds the value of $\overset{\bullet}{u}$. The certificate for this function is built out of the certificate $\overset{\circ}{u}$.

$$\lambda a.\mathit{unpack}\, a\ \mathit{to}\ \langle \overset{\bullet}{u}, \overset{\circ}{u}\rangle\ \mathit{in}\ \mathit{box}_{\lambda b:\mathit{Nat}.\mathit{plus}(b,\overset{\circ}{u})}\ \lambda b : \mathit{Nat}.(b + \overset{\bullet}{u})$$

8. Related work

There are many foundational calculi for concurrent and distributed programming. Since the focus of this work is on logically motivated such calculi we comment on related work from this viewpoint. To the best of our knowledge, the extant literature does not address calculi for both mobility/concurrency and code certification in a unified theory. Regarding mobility, however, a number of ideas have been put forward. The closest to this article is the work of Moody [15], that of Murphy et al. [19,17,18] and that of Jia and Walker [14]. Moody suggests an operational reading of proofs in an intuitionistic fragment of S4 also based on a judgemental analysis of this logic [11]. It takes a step further in terms of obtaining a practical programming language for mobility in that it addresses effectful computation (references and reference update). Also, the diamond connective is considered. Worlds are deliberately left implicit. The author argues this "encourages the programmer to work locally". Murphy et al. also introduce a mobility inspired operational interpretation of a Natural Deduction presentation of propositional modal logic, although S5 is considered in there work (both intuitionistic [19] and classical [17]). They also introduce explicit reference to worlds in their programming model. Operational semantics in terms of abstract machines is considered [19,17] and also a big-step semantics on terms [16]. Both necessity and possibility modalities are considered. Finally, they explore a type preserving compiler for a prototype language for client/server applications based on their programming model [18]. Jia and Walker [14] also present a term assignment for a hybrid modal logic close to S5. They argue that the hybrid approach gives the programmer a tighter control over code distribution. Finally, Borghuis and Feijs [9] introduce a calculus of stationary services and mobile values whose type system is based on modal logic. Mobility however may not be internalized as a proposition. For example, $\Box^o(A \supset B)$ is the type of a service located at o that computes values of B given one of type A. None of the cited works incorporate the notion of certificate in their systems. A different direction in terms of the computational interpretation of JL is pursued in [7] where a *history-aware* lambda calculus is studied.

9. Conclusion

We present a Curry–de Bruijn–Howard analysis of an intuitionistic fragment (IJL) of Justification Logic JL. We start from a Natural Deduction presentation for IJL and associate propositions and proofs of this system to types and terms of a mobile calculus $\lambda^{\mathsf{Cert}}_{\Box}$. The modal type constructor $[s]A$ is interpreted as the type of *mobile units*, expressions composed of a code and certificate component. $\lambda^{\mathsf{Cert}}_{\Box}$ has thus language constructs for both code and certificates. Its type system is a unified theory in which both code and certificate construction are verified. Indeed, when mobile units are constructed from the code of other mobile units, the type system verifies not only that the former is mobile in nature (i.e. depends on no local resources) but also that the certificate for this new mobile unit is correctly assembled from the certificates of the latter.

We deal exclusively with the necessity modality given that JL does not treat $\Diamond$ (since it is based on classical logic and hence does not need to). However, in an intuitionistic setting the interpretation of $\Diamond$ in possible world semantics is not as uncontroversial as that of the necessity modality [21, Chapter 3]. Nevertheless, it would be quite straightforward to add inference schemes for a possibility modality in the line of related literature (cf. Section 8). A term of type $\Diamond A$ is generally interpreted to denote a value of a term at a remote location. Although this breaks the connection with JL it would make sense from the programming languages perspective.

Although $\lambda^{\mathsf{Cert}}_{\Box}$ is meant to be concept-of-proof language, it clearly does not provide the features needed to build extensive examples. Two basic additions that should be considered are references (and computation with effects) and recursion. Another one is polymorphism over proof variables. The first example in Section 7, rather than have type $[s](A \supset B) \supset$

$[t]A \supset [s \cdot t]B$ should have type $\forall\alpha.\forall\beta.[\alpha](A \supset B) \supset [\beta]A \supset [\alpha \cdot \beta]B$, where α and β are variables ranging over arbitrary certificates.

JL includes a proof term constructor "+" called *plus*. A proof term of the form $s + t$ is interpreted as the juxtaposition of (the interpretation of) s and (the interpretation of) t in provability semantics [2]. Plus is necessary for all S4 theorems to be realized in JL. As a consequence it would be necessary to include it in $\lambda_{\Box}^{\mathsf{Cert}}$. As a motivating example, consider the power function from [12] where modal necessity operators have been decorated with "$?_i$", $i \in 1..3$, meaning that these proof terms that have not yet been determined:

$$fix\ (\lambda p : Nat \to [?_1](Nat \to Nat).\lambda n : Nat.M) \tag{2}$$

where

$$\begin{aligned} M \stackrel{\text{def}}{=}\ & case\ n\ of \\ & \mathbf{z} \longrightarrow box_{?_2}\ (\lambda x : Nat.\mathbf{s}\,\mathbf{z}) \\ & \boxplus\ \mathbf{s}\,m \longrightarrow unpack\ p\ m\ to\ \langle \overset{\bullet}{u}, \overset{\circ}{u}\rangle\ in\ box_{?_3}\ (\lambda x : Nat.x * (\overset{\bullet}{u}\ x)) \end{aligned}$$

Here *Nat* denotes the base type of natural numbers, **z** and **s** are the constructors and *case* is the elimination scheme. The corresponding typing schemes are (we omit the typing scheme for the product operation "$*$' over natural numbers):

$$\frac{}{\Delta;\ \Gamma \rhd \mathbf{z} : Nat \mid zero} \qquad \frac{\Delta;\ \Gamma \rhd M : Nat \mid s}{\Delta;\ \Gamma \rhd \mathbf{s}\,M : Nat \mid suc(s)}$$

$$\frac{\Delta;\ \Gamma \rhd M : Nat \mid r \quad \Delta;\ \Gamma \rhd P : A \mid s \quad \Delta;\ \Gamma, x : Nat \rhd Q : A \mid t}{\Delta;\ \Gamma \rhd case\ M\ of\ \mathbf{z} \longrightarrow P \boxplus \mathbf{s}\,x \longrightarrow Q : A \mid case(r, s, [x : Nat]t)}$$

First let us concentrate on M. Clearly $?_2$ and $?_3$ shall be different proof terms, namely $\lambda x : Nat.suc(zero)$ and $\lambda x : Nat.times(x, \overset{\circ}{u} \cdot x)$, respectively. The type of the term on the right of the arrow (case $\mathbf{s}\,m$) will therefore have type $\lambda x : Nat.times(x, ?_1 \cdot x)$. Finally, M will need to type with type $[\lambda x : Nat.suc(zero) + \lambda x : Nat.times(x, ?_1 \cdot x)](Nat \to Nat)$.

Inclusion of an inference scheme for introducing plus (together with the above mentioned schemes for constructors and case elimination of natural numbers) would allow M to be typed. However, if we now want to type the whole term (2), then additional considerations arise. In particular, in order to type the whole term including the fixpoint operator we are required to solve recursive equation on proof terms. In our example, we must solve the equation:

$$?_1 = \lambda x : Nat.suc(zero) + \lambda x : Nat.times(x, ?_1 \cdot x)$$

The reason is the context sensitive nature of the typing scheme for *fix*: in the judgement of the hypothesis both the type of x and the type of M are required to be identical:

$$\frac{\Delta;\ \Gamma, x : A \rhd M : A \mid s}{\Delta;\ \Gamma \rhd fix\ (\lambda x : A.M) : A \mid fix(s)}$$

These are a selection of the issues that are the focus of our current efforts in developing a toy programming language based on $\lambda_{\Box}^{\mathsf{Cert}}$.

References

[1] Carlos Areces, Balder ten Cate, Hybrid logics, in: P. Blackburn, F. Wolter, J. van Benthem (Eds.), Handbook of Modal Logics, Elsevier, 2006.
[2] Sergei N. Artëmov, Logic of proofs, Ann. Pure Appl. Logic 67 (1–3) (1994) 29–59.
[3] Sergei Artëmov, Operational modal logic. Technical Report MSI 95-29, Cornell University, 1995.
[4] Sergei Artëmov, Unified semantics of modality and λ-terms via proof polynomials, in: Algebras, Diagrams and Decisions in Language, Logic and Computation, 2001, pp. 89–118.
[5] Sergei Artëmov, Leo Beklemishev, Provability logic, in: D. Gabbay, F. Guenthner (Eds.), Handbook of Philosophical Logic, vol. 13, 2nd ed., Kluwer, 2004, pp. 229–403.
[6] Sergei N. Artëmov, Eduardo Bonelli, The intensional lambda calculus, in: Sergei N. Artëmov, Anil Nerode (Eds.), LFCS, in: Lecture Notes in Computer Science, vol. 4514, Springer, 2007, pp. 12–25.
[7] Francisco Bavera, Eduardo Bonelli, Justification logic and history based computation, in: Ana Cavalcanti, David Déharbe, Marie-Claude Gaudel, Jim Woodcock (Eds.), ICTAC, in: Lecture Notes in Computer Science, vol. 6255, Springer, 2010, pp. 337–351.
[8] Eduardo Bonelli, Federico Feller, The logic of proofs as a foundation for certifying mobile computation, in: Sergei N. Artëmov, Anil Nerode (Eds.), LFCS, in: Lecture Notes in Computer Science, vol. 5407, Springer, 2009, pp. 76–91.
[9] Tijn Borghuis, Loe M.G. Feijs, A constructive logic for services and information flow in computer networks, Comput. J. 43 (4) (2000) 274–289.
[10] Pierre-Louis Curien, Hugo Herbelin, The duality of computation, in: ICFP, 2000, pp. 233–243.
[11] Rowan Davies, Frank Pfenning, A judgmental reconstruction of modal logic, Math. Structures Comput. Sci. 11 (2001) 511–540.
[12] Rowan Davies, Frank Pfenning, A modal analysis of staged computation, J. ACM 48 (3) (2001) 555–604.
[13] Hugo Herbelin, A lambda-calculus structure isomorphic to gentzen-style sequent calculus structure, in: Leszek Pacholski, Jerzy Tiuryn (Eds.), CSL, in: Lecture Notes in Computer Science, vol. 933, Springer, 1994, pp. 61–75.
[14] Limin Jia, David Walker, Modal proofs as distributed programs (extended abstract), in: David A. Schmidt (Ed.), ESOP, in: Lecture Notes in Computer Science, vol. 2986, Springer, 2004, pp. 219–233.

[15] Jonathan Moody, Logical mobility and locality types, in: Sandro Etalle (Ed.), LOPSTR, in: Lecture Notes in Computer Science, vol. 3573, Springer, 2004, pp. 69–84.
[16] Tom Murphy VII., Modal types for mobile code. Ph.D. Thesis, Carnegie Mellon, January 2008 (draft).
[17] Tom Murphy VII, Karl Crary, Robert Harper, Distributed control flow with classical modal logic, in: C.-H. Luke Ong (Ed.), CSL, in: Lecture Notes in Computer Science, vol. 3634, Springer, 2005, pp. 51–69.
[18] Tom Murphy VII, Karl Crary, Robert Harper, Type-safe distributed programming with ml5, in: Gilles Barthe, Cédric Fournet (Eds.), TGC, in: Lecture Notes in Computer Science, vol. 4912, Springer, 2007, pp. 108–123.
[19] Tom Murphy VII, Karl Crary, Robert Harper, Frank Pfenning, A symmetric modal lambda calculus for distributed computing, in: LICS, IEEE Computer Society, 2004, pp. 286–295.
[20] Benjamin C. Pierce, Types and Programming Languages, MIT Press, 2002.
[21] Alex Simpson, The proof theory and semantics of intuitionistic modal logic. Ph.D. Thesis, University of Edinburgh, 1994.
[22] Walid Taha, Tim Sheard, Multi-stage programming, in: ICFP, 1997, p. 321.
[23] Philip Wickline, Peter Lee, Frank Pfenning, Rowan Davies, Modal types as staging specifications for run-time code generation, ACM Comput. Surveys 30 (3es) (1998) 8.