JFP **28**, e1, 20 pages, 2018.
doi:10.1017/S0956796817000144

*The adequacy of Launchbury's natural semantics for lazy evaluation**

JOACHIM BREITNER

Computer and Information Science, University of Pennsylvania, Philadelphia, PA-19146, USA
(*e-mail:* joachim@cis.upenn.edu)

Abstract

In his seminal paper "A Natural Semantics for Lazy Evaluation", John Launchbury proves his semantics correct with respect to a denotational semantics, and outlines a proof of adequacy. Previous attempts to rigorize the adequacy proof, which involves an intermediate natural semantics and an intermediate resourced denotational semantics, have failed. We devised a new, direct proof that skips the intermediate natural semantics. It is the first rigorous adequacy proof of Launchbury's semantics. We have modeled our semantics in the interactive theorem prover Isabelle and machine-checked our proofs. This does not only provide a maximum level of rigor, but also serves as a tool for further work, such as a machine-checked correctness proof of a compiler transformation.

1 Introduction

The Natural Semantics for Lazy Evaluation created by Launchbury (1993) has turned out to be a popular and successful base for theoretical treatment of lazy evaluation, especially as the basis of analyzing language extensions (Baker-Finch *et al*., 2000; Eekelen & Mol, 2004; Nakata & Hasegawa, 2009; Nakata, 2010; Sánchez-Gil *et al*., 2010). Therefore, its correctness and adequacy is important in this field of research. The original paper defines a standard denotational semantics to prove the natural semantics (NS) correct and adequate against.

Launchbury presents its correctness proof in sufficient detail, and it endures formal verification with only small changes and clarifications.

His adequacy proof is only a rough outline, though. It suggests to establish the computational adequacy by two intermediate semantics:

- A modified NS with slightly different rules for variable lookup and function application, that is supposed to be closer to how the denotational semantic works, and
- a resourced denotational semantics, i.e. one that keeps track of the number of steps required to evaluate an expression.

* This work was carried out while the author was a member of the Programming Paradigms Group of the Karlsruhe Institute of Technology, Germany. The author was supported by the Deutsche Telekom Stiftung.

The equivalence between the NS and the alternative natural semantics (ANS) was not proven by Launchbury. Although quite intuitive, a rigorous proof is yet to be found. Sánchez-Gil *et al.* have attempted to perform this proof and obtained an equivalency proof for just the change to the application rule (2014); the other half is still pending.

Having seen the difficulty of performing these proof steps on the side of the NS, we departed from the outline provided by Launchbury and skipped the ANS altogether. With a small, but important modification to the resourced denotational semantics, which ensures that while evaluating an expression with finite resources, values on the heap are used with fewer resources, the proof was possible.

We have implemented and mechanically verified all definitions, propositions and proofs using the theorem prover Isabelle/HOL (Breitner, 2013). This way we can be confident that there are no holes left in the proof that would again have to be filled by later generations. Furthermore, it provides a tool that can be used in further work: In Breitner (2015c), we use the denotational semantics to show the functional correctness of a compiler transformation, while we use the operational semantics to prove that the transformation does not degrade the program's performance (measured by the number of heap allocations).

Our contributions are as follows:

- We reproduce and clarify Launchbury's correctness proof (Section 3).
- We analyze the ANS used in his adequacy proof outline and show how to handle the differences on the denotational sides (Section 4.2).
- This way, we can provide a new and more direct proof of adequacy (Section 4.3).
- We provide an Isabelle implementation of the various definitions and a machine-checked proof.
- We identify and discuss all adjustments to original definitions that we found to be required or helpful when rigorizing this work (Sections 2.3, 3.1 and 4.6), and discuss.

2 Launchbury's semantics

Launchbury defines a semantics for a simple untyped lambda calculus consisting of variables, lambda abstraction, applications and mutually recursive bindings:

$$x, y, z, w \in \mathsf{Var}$$
$$e \in Exp ::= \lambda x.\, e \mid e\ x \mid x \mid \mathsf{let}\ x_1 = e_1, \ldots, x_n = e_n\ \mathsf{in}\ e$$

The set of free variables of an expression e is denoted by $\mathrm{fv}(e)$; we overload this notation and use fv with arguments of other types that may contain variable names.

We equate alpha-equivalent lambda abstractions and let bindings, i.e. $\lambda x.\, x = \lambda y.\, y$ and $\mathrm{fv}(\lambda x.\, y\ x) = \{y\}$. The theoretical foundation used is Nominal logic (Urban & Kaliszyk, 2012). This does impose a few well-formedness side conditions, such as equivariance of definition over expressions. We skip them in this presentation, and do so with good conscience, as they have been covered in the machine-checked proof.

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

$$\frac{}{\Gamma : \lambda x.e \Downarrow_L \Gamma : \lambda x.e}\textsc{Lam} \qquad \frac{\Gamma : e \Downarrow_L \Delta : \lambda y.e' \qquad \Delta : e'[x/y] \Downarrow_L \Theta : v}{\Gamma : e\, x \Downarrow_L \Theta : v}\textsc{App}$$

$$\frac{\Gamma : e \Downarrow_{L \cup \{x\}} \Delta : v}{x \mapsto e, \Gamma : x \Downarrow_L x \mapsto v, \Delta : v}\textsc{Var} \qquad \frac{\mathrm{dom}\,\Delta \cap \mathrm{fv}(\Gamma, L) = \{\} \qquad \Gamma, \Delta : e \Downarrow_L \Theta : v}{\Gamma : \mathsf{let}\ \Delta\ \mathsf{in}\ e \Downarrow_L \Theta : v}\textsc{Let}$$

Fig. 1. Launchbury natural semantics, as revised by Sestoft.

Note that the term on the right hand side of an application has to be a variable. A general lambda term of the form $e_1\ e_2$ would have to be pre-processed to $\mathsf{let}\ x = e_2\ \mathsf{in}\ e_1\ x$ before it can be handled by this semantics.

2.1 Natural semantics

Launchbury gives this language meaning by a NS, specified with the rules in Figure 1, which obey the following naming convention for heaps and values:

$$\Gamma, \Delta, \Theta \in \mathsf{Heap} = \mathsf{Var} \rightarrow\!\!\!\!\!\!\cdot\ \ \mathsf{Exp}$$
$$v \in \mathsf{Val} \quad ::= \lambda x.e$$

A heap is a partial function from variables to expressions; the same type is used for the list of bindings in a let. The domain of a heap Γ, written dom Γ, is the set of variables bound in the heap. Heaps are not alpha-equated, so dom $\Gamma \subseteq \mathrm{fv}(\Gamma)$. We write $x \mapsto e$ for the singleton heap and use commas to combine heaps with distinct domain.

A value is an expression in weak head normal form. Here, the only values are lambda abstractions. A judgment of the form $\Gamma : e \Downarrow_L \Delta : v$ means that the expression e with the heap Γ reduces to v, while modifying the heap to Δ.

The set L was not present in Launchbury's rules, but added by Sestoft (1997) to keep track of variables that must be avoided when choosing new names in the Let rule, but would otherwise not be present in the judgment any more (cf. Section 2.3.1).

We consider a judgment $\Gamma : e \Downarrow_L \Delta : v$ to be *closed* if $\mathrm{fv}(\Gamma, e) \subseteq \mathrm{dom}\ \Gamma \cup L$. Note that this property is preserved by our semantics, as $\mathrm{fv}(\Delta, v) \subseteq \mathrm{dom}\ \Delta \cup L$ holds for closed judgments as well.

The evaluation does not forget bindings, i.e. $\Gamma : e \Downarrow_L \Delta : v$ implies dom $\Gamma \subseteq$ dom Δ.

2.2 Denotational semantics

In order to show that the NS behaves as expected, Launchbury defines a standard denotational semantics for expressions and heaps, following Abramsky (1990). The semantic domain Value is the initial solution to the domain equation:

$$\mathsf{Value} = (\mathsf{Value} \rightarrow \mathsf{Value})_\perp,$$

which distinguishes $\perp$ from $\lambda x.\perp$. Lifting between $\mathsf{Value} \rightarrow \mathsf{Value}$ and Value is performed using the injection $\mathsf{Fn}\,(\_)$ and projection $\_ \downarrow_{\mathsf{Fn}} \_$. Values are partially ordered by $\sqsubseteq$.

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

A semantic environment maps variables to values:

$$\rho \in \mathsf{Env} = \mathsf{Var} \to \mathsf{Value}$$

and the initial environment $\bot$ maps all variables to $\bot$. Environments are ordered by lifting the order on Value pointwise.

The environment $\rho|_S$, where S is a set of variables, is the restriction of ρ to S:

$$(\rho|_S)\, x = \begin{cases} \rho\, x, & \text{if } x \in S \\ \bot & \text{if } x \notin S. \end{cases}$$

The environment $\rho \setminus S$ is defined as the restriction of ρ to the complement of S, i.e. $\rho \setminus S \coloneqq \rho|_{\mathsf{Var} \setminus S}$.

The semantics of expressions and heaps are mutually recursive. The meaning of an expression $e \in \mathsf{Exp}$ in an environment $\rho \in \mathsf{Env}$ is written as $[\![e]\!]_\rho \in \mathsf{Value}$ and is defined by

$$\begin{aligned} [\![\lambda x.\, e]\!]_\rho &\coloneqq \mathsf{Fn}\,(\lambda v.[\![e]\!]_{\rho \sqcup \{x \to v\}}) \\ [\![e\ x]\!]_\rho &\coloneqq [\![e]\!]_\rho\ \downarrow_{\mathsf{Fn}}\ \rho\, x \\ [\![x]\!]_\rho &\coloneqq \rho\, x \\ [\![\mathsf{let}\ \Delta\ \mathsf{in}\ e]\!]_\rho &\coloneqq [\![e]\!]_{\{\Delta\}\rho}. \end{aligned}$$

We can map this over a heap to obtain an environment:

$$[\![x_1 \mapsto e_1, \ldots, x_n \mapsto e_n]\!]_\rho \coloneqq [x_1 \mapsto [\![e_1]\!]_\rho, \ldots, x_n \mapsto [\![e_n]\!]_\rho]$$

The semantics of a heap $\Gamma \in \mathsf{Heap}$ in an environment ρ, written $\{\Gamma\}\rho \in \mathsf{Env}$, is then obtained as a least fixed-point:

$$\{\Gamma\}\rho = (\mu\rho'.\, \rho +\!+_{\mathrm{dom}\,\Gamma} [\![\Gamma]\!]_{\rho'})$$

where

$$(\rho +\!+_S \rho')\ x \coloneqq \begin{cases} \rho\ x, & \text{if } x \notin S \\ \rho'\ x, & \text{if } x \in S. \end{cases}$$

is a restricted update operator.

The least fixed-point exists, as all involved operations are monotone and continuous, and by unrolling the fixed-point once, we can see that

Lemma 1 (*Application of the heap semantics*)

$$(\{\Gamma\}\rho)\, x = \begin{cases} [\![e]\!]_{\{\Gamma\}\rho}, & \text{if } (x \mapsto e) \in \Gamma \\ \rho\, x, & \text{if } x \notin \mathrm{dom}\,\Gamma. \end{cases}$$

The following substitution lemma plays an important role in finding a more direct proof of adequacy, but is also required for the correctness proof, as performed by Launchbury:

Lemma 2 (*Semantics of substitution*)

$$[\![e]\!]_{\rho(y \to \rho\ x)} = [\![e[x/y]]\!]_\rho.$$

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

Proof
We first show $\forall \rho.\, \rho\, x = \rho\, y \implies \{\!|e|\!\}\rho = \{\!|e[x/y]|\!\}\rho$ by induction on e, using parallel fixed-point induction in the case for let. This allows us to calculate

$$\begin{aligned} [\![e]\!]_{\rho(y \mapsto \rho\, x)} &= [\![e[x/y]]\!]_{\rho(y \mapsto \rho\, x)} \quad \{\text{as } \rho(y \mapsto \rho\, x)\, x = \rho(y \mapsto \rho\, x)\, y\} \\ &= [\![e[x/y]]\!]_{\rho} \quad \{\text{as } y \notin \mathrm{fv}(e[x/y])\}. \end{aligned}$$

□

We sometimes write $\{\!|\Gamma|\!\}$ instead of $\{\!|\Gamma|\!\}\bot$. In an expression $\{\!|\Gamma|\!\}(\{\!|\Delta|\!\}\rho)$, we omit the parentheses and write $\{\!|\Gamma|\!\}\{\!|\Delta|\!\}\rho$.

2.3 Discussions of modifications

It is rare that a formal system developed with pen and on paper can be formalized to the letter, partly because of vagueness (what, exactly, is a "completely" fresh variable?), partly because of formalization convenience, and partly because the stated facts – even if morally correct – are wrong when read scrupulously, and our work is no exception. We discuss any such divergence from Launchbury's work here.

2.3.1 Naming

Getting the naming issues right is one of the major issues when formalizing anything involving bound variables. In Launchbury's work, the names are manifestly part of the syntax, i.e. $\lambda x.\, x \neq \lambda y.\, y$, and his rules involve explicit renaming of bound variables to fresh ones in the rule Var. His definition of freshness is a global one, so the validity of a derivation using Var depends on everything around it. This is morally what we want, but very unpractical.

Sestoft (1997) noticed this problem and fixed it by adding a set L of variables to the judgment, so that every variable to be avoided occurs somewhere in Γ, e, or L. Instead of renaming all bound variables in the rule Var, he chooses fresh names for the new heap bindings in the Let.

We build on that, but go one step further and completely avoid bound names in the expressions, i.e. $\lambda x.\, x = \lambda y.\, y$. We still have them in the syntax, of course, but these are just representatives of the an α-equivalency class. Nominal logic (Urban & Kaliszyk, 2012), which is implemented in Isabelle, forms the formal foundation for this. So in our rule Let we do not have to rename the variables, but simply may assume that the variables used in the representation of the let-expression are sufficiently fresh.

The names of bindings on the heap are not abstracted away in that manner; this follows (Sánchez-Gil *et al.*, 2012).

2.3.2 Closed judgments

Launchbury deliberately allows non-closed configurations in his derivations, i.e. configurations with free variables in the terms that have no corresponding binding on the heap. This is a necessity, as rule Var models blackholing by removing a binding from the heap during its evaluation.

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

With the addition of the set of variables to avoid, which will always contain such variables, the question can be revisited. And indeed, Sestoft defines the notion of *L-good configurations*, where all free variables are either bound on the heap, or contained in L. He shows that this property is preserved by the operational semantics and subsequently considers only L-good configurations. We follow this example with our definition of *closed* judgments. Threading the closedness requirement through a proof by rule induction is a typical chore contributing to the overhead of a machine-checked formalization.

2.3.3 Join versus update

Launchbury specifies his denotational semantics using a binary operation $\sqcup$ on environments. He does not define it explicitly, but the statements in his Section 5.2.1 leave no doubt that he indeed intended this operation to denote the least upper bound of its arguments, as one would expect. Unfortunately, with this definition, his Theorem 2 (which corresponds to our Theorem 2) is false.

A counter example is $e = x$, $v = (\lambda a.\,\mathsf{let}\ b = b\ \mathsf{in}\ b)$, $\Gamma = \Delta = (x \mapsto v)$ and $\rho = (x \mapsto \mathsf{Fn}\,(\lambda\_.\mathsf{Fn}\,(\lambda x.x)))$. Note that the denotation of v is $\mathsf{Fn}\,(\lambda\_.\bot)$ in every environment. We have $\Gamma : e \Downarrow_{\{\}} \Delta : v$, so according to the theorem, $[\![e]\!]_{\{\Gamma\}\rho} = [\![v]\!]_{\{\Delta\}\rho}$ should hold, but

$$\begin{aligned}
[\![e]\!]_{\{\Gamma\}\rho} &= (\{\Gamma\}\rho)\,x \\
&= \rho\,x \sqcup [\![v]\!]_{\{\Gamma\}\rho} \\
&= \mathsf{Fn}\,(\lambda_.\mathsf{Fn}\,(\lambda x.x)) \sqcup \mathsf{Fn}\,(\lambda_.\bot) \\
&= \mathsf{Fn}\,(\lambda_.\mathsf{Fn}\,(\lambda x.x) \sqcup \bot) \\
&= \mathsf{Fn}\,(\lambda_.\mathsf{Fn}\,(\lambda x.x)) \\
&\neq \mathsf{Fn}\,(\lambda_.\bot) \\
&= [\![v]\!]_{\{\Delta\}\rho}.
\end{aligned}$$

The crucial property of the counter-example is that ρ contains compatible, but better information for a variable also bound in Γ. The mistake in his correctness proof is in the step $(\{x \mapsto v, \Delta\}\rho)\,x = [\![v]\!]_{\{x\to v,\Delta\}\rho}$ in the case for VAR, which should be $(\{x \mapsto v, \Delta\}\rho)\,x = [\![v]\!]_{\{x\to v,\Delta\}\rho} \sqcup \rho\,x$.

Intuitively, such rogue ρ are not relevant for a proof of the main Theorem 1. Nevertheless, this issue needs to be fixed before attempting a formal proof. One possible fix is to replace $\sqcup$ by a right-sided update operation that just throws away information from the left argument for those variables bound on the right. We use the syntax $\rho +\!+_S \rho'$ for this operation, and by using that the proof goes through in full rigor.

It is slightly annoying having to specify the set S in this operation explicitly, as it is usually clear "from the context": Morally, it is the set of variables that the object on the right talks about. But as environments, i.e. total functions from $\mathsf{Var} \to \mathsf{Value}$, do not distinguish between variables not mentioned at all and variables mentioned, but bound to $\bot$, this information is not easily exploitable in a formal setting.

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

For the same reason we replace Launchbury's ordering $\leqslant$ on environments by the more explicit equality between restricted environments in the formulation of Theorem 2.

3 Correctness

The main correctness theorem for the NS is

Theorem 1 (*Correctness*)
If $\Gamma : e \Downarrow_L \Delta : v$ holds and is closed, then $[\![e]\!]_{\{\Gamma\}} = [\![v]\!]_{\{\Delta\}}$.

In order to prove this by rule induction, we have to generalize it to

Theorem 2 (*Correctness, generalized*)
If $\Gamma : e \Downarrow_L \Delta : v$ holds and is closed, then for all environments $\rho \in \mathsf{Env}$ we have $[\![e]\!]_{\{\Gamma\}\rho} = [\![v]\!]_{\{\Delta\}\rho}$ and $(\{\Gamma\}\rho)|_{\mathrm{dom}\,\Gamma} = (\{\Delta\}\rho)|_{\mathrm{dom}\,\Gamma}$.

Our proof follows Launchbury's steps, but differs in some details. Two required technical lemmas are stated and proved subsequently.

For clarity, we write $\rho =_{|S} \rho'$ for $\rho|_S = \rho'|_S$.

Proof
by induction on the derivation of $\Gamma : e \Downarrow_L \Delta : v$. Note that in such a derivation, all occurring judgments are closed.

Case: Lam
This case is trivial.

Case: App
The induction hypotheses are $[\![e]\!]_{\{\Gamma\}\rho} = [\![\lambda y.\, e']\!]_{\{\Delta\}\rho}$ and $\{\Gamma\}\rho =_{|\mathrm{dom}\,\Gamma}\ \{\Delta\}\rho$ as well as $[\![e'[x/y]]\!]_{\{\Delta\}\rho} = [\![v]\!]_{\{\Theta\}\rho}$ and $\{\Delta\}\rho =_{|\mathrm{dom}\,\Delta}\ \{\Theta\}\rho$.

We have $\{\Gamma\}\rho\ x = \{\Delta\}\rho\ x$: If $x \in \mathrm{dom}\ \Gamma$, this follows from the induction hypothesis. Otherwise, we know $x \in L$, as the judgment is closed, and the new names bound in Δ avoid L, so we have $\rho\, x$ on both sides.

While the second part follows from the corresponding inductive hypotheses and $\mathrm{dom}\ \Gamma \subseteq \mathrm{dom}\ \Delta$, the first part is a simple calculation:

$$
\begin{aligned}
[\![e\ x]\!]_{\{\Gamma\}\rho} &= [\![e]\!]_{\{\Gamma\}\rho}\ \downarrow_{\mathsf{Fn}}\ \{\Gamma\}\rho\, x \\
&\quad \{\ \text{by the denotation of application}\ \} \\
&= [\![\lambda y.\, e']\!]_{\{\Delta\}\rho}\ \downarrow_{\mathsf{Fn}}\ \{\Gamma\}\rho\ x \\
&\quad \{\ \text{by the induction hypothesis}\ \} \\
&= [\![\lambda y.\, e']\!]_{\{\Delta\}\rho}\ \downarrow_{\mathsf{Fn}}\ \{\Delta\}\rho\ x \\
&\quad \{\ \text{see above}\ \} \\
&= [\![e']\!]_{(\{\Delta\}\rho)(y \mapsto \{\Delta\}\rho\ x)} \\
&\quad \{\ \text{by the denotation of lambda abstraction}\ \} \\
&= [\![e'[x/y]]\!]_{\{\Delta\}\rho} \\
&\quad \{\ \text{by Lemma 2}\ \} \\
&= [\![v]\!]_{\{\Theta\}\rho} \\
&\quad \{\ \text{by the induction hypothesis}\ \}
\end{aligned}
$$

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

Case: VAR

We know that $[\![e]\!]_{\{\Gamma\}\rho'} = [\![v]\!]_{\{\Delta\}\rho'}$ and $\{\Gamma\}\rho' =_{|\mathrm{dom}\,\Gamma} \{\Delta\}\rho'$ for all $\rho' \in \mathsf{Env}$.

We begin with the second part:

$$\begin{aligned}
\{x \mapsto e, \Gamma\}\rho &= \mu\rho'.(\rho +\!\!+_{\mathrm{dom}\,\Gamma} \{\Gamma\}\rho')[x \mapsto [\![e]\!]_{\{\Gamma\}\rho'}] \\
&\quad \{\text{ by Lemma 3 }\} \\
&= \mu\rho'.(\rho +\!\!+_{\mathrm{dom}\,\Gamma} \{\Gamma\}\rho')[x \mapsto [\![v]\!]_{\{\Delta\}\rho'}] \\
&\quad \left\{ \begin{array}{l} \text{by the induction hypothesis. Note that} \\ \text{we invoke it for } \rho' \text{ with } \rho' \neq \rho! \end{array} \right\} \\
&=_{|\mathrm{dom}\,(x \mapsto e,\Gamma)} \mu\rho'.(\rho +\!\!+_{\mathrm{dom}\,\Delta} \{\Delta\}\rho')[x \mapsto [\![v]\!]_{\{\Delta\}\rho'}] \\
&\quad \{\text{ by the induction hypothesis; see below }\} \\
&= \{x \mapsto v, \Delta\}\rho \\
&\quad \{\text{ by Lemma 3 }\}
\end{aligned}$$

The second but last step is quite technical, as we need to push the $|_{\mathrm{dom}\,(x \mapsto e,\Gamma)}$ inside the fixed-point operator. This goes through by parallel fixed-point induction if we first generalize it to $|_{\mathsf{Var} \setminus \mathrm{dom}\,\Delta \cup \mathrm{dom}\,(x \mapsto e,\Gamma)}$, the restriction to the complement of the new variables added to the heap during evaluation of x.

The first part now follows from the second part:

$$\begin{aligned}
[\![x]\!]_{\{x \mapsto e,\Gamma\}\rho} &= (\{x \mapsto e, \Gamma\}\rho)\, x \\
&= (\{x \mapsto v, \Delta\}\rho)\, x && \{\text{ by the first part and } x \in \mathrm{dom}\,(x \mapsto e, \Gamma)\ \} \\
&= [\![v]\!]_{\{x \mapsto v,\Delta\}\rho} && \{\text{ by Lemma 1. }\}
\end{aligned}$$

Case: LET

We know that $[\![e]\!]_{\{\Gamma,\Delta\}\rho} = [\![v]\!]_{\{\Theta\}\rho}$ and $\{\Gamma, \Delta\}\rho =_{|\mathrm{dom}\,(\Gamma,\Delta)} \{\Theta\}\rho$. For the first part, we have

$$\begin{aligned}
[\![\mathsf{let}\ \Delta\ \mathsf{in}\ e]\!]_{\{\Gamma\}\rho} &= [\![e]\!]_{\{\Delta\}\{\Gamma\}\rho} && \{\text{ by the denotation of let-expressions }\} \\
&= [\![e]\!]_{\{\Gamma,\Delta\}\rho} && \{\text{ by the following Lemma 4 }\} \\
&= [\![v]\!]_{\{\Theta\}\rho} && \{\text{ by the induction hypothesis }\}
\end{aligned}$$

and for the second part, we have

$$\begin{aligned}
\{\Gamma\}\rho &=_{|\mathrm{dom}\,\Gamma} \{\Delta\}\{\Gamma\}\rho && \{\text{ because dom } \Delta \text{ are fresh }\} \\
&= \{\Gamma, \Delta\}\rho && \{\text{ again by Lemma 4 }\} \\
&=_{|\mathrm{dom}\,(\Gamma,\Delta)} \{\Theta\}\rho. && \{\text{ by the induction hypothesis. }\}
\end{aligned}$$

□

In the case for VAR, we switched from the usual, simultaneous definition of the heap semantics to an iterative one, in order to be able to make use of the induction hypothesis:

Lemma 3 (*Iterative definition of the heap semantics*)

$$\{x \mapsto e, \Gamma\}\rho = \mu\rho'.\left((\rho +\!\!+_{\mathrm{dom}\,\Gamma} \{\Gamma\}\rho')[x \mapsto [\![e]\!]_{\{\Gamma\}\rho'}]\right).$$

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

A corresponding lemma can be found in Launchbury (1993), but without proof. As the proof involves some delicate fixed-point-juggling, we include it here in detail:

Proof
Let $L = (\lambda\rho'.\, \rho +\!\!+_{\mathrm{dom}\,(x \mapsto e,\Gamma)} [\![x \mapsto e, \Gamma]\!]_{\rho'})$ be the functorial of the fixed point on the left hand side, R be the functorial on the right hand side.

By Lemma 1, we have

1. $(\mu L)\, y = [\![e']\!]_{\mu L}$ for $y \mapsto e' \in \mathrm{dom}\ \Gamma$,
2. $(\mu L)\, x = [\![e]\!]_{\mu L}$,
3. $(\mu L)\, y = \rho\, y$ for $y \notin \mathrm{dom}\ (x \mapsto e, \Gamma)$

Similarly, by unrolling the fixed points, we have

4. $(\mu R)\, y = [\![e']\!]_{\{\Gamma\}(\mu R)}$ for $y \mapsto e' \in \mathrm{dom}\ \Gamma$,
5. $(\mu R)\, x = [\![e]\!]_{\{\Gamma\}(\mu R)}$,
6. $(\mu R)\, y = \rho\, y$ for $y \notin \mathrm{dom}\ (x \mapsto e, \Gamma)$,

and also for $\rho' \in \mathsf{Env}$ (in particular for $\rho' = (\mu L)$, (μR)), again using Lemma 1,

7. $(\{\Gamma\}\rho')\, y = [\![e]\!]_{\{\Gamma\}\rho'}$ for $y \mapsto e' \in \mathrm{dom}\ \Gamma$,
8. $(\{\Gamma\}\rho')\, y = \rho'\, y$ for $y \notin \mathrm{dom}\ \Gamma$.

We obtain

9. $\{\Gamma\}(\mu R) = (\mu R)$

from comparing (4)–(6) with (7) and (8). We can also show

10. $\{\Gamma\}(\mu L) = (\mu L)$,

by antisymmetry and using that least fixed points are least pre-fixed points:

$\sqsubseteq$: We need to show that $(\mu L) +\!\!+_{\mathrm{dom}\,\Gamma} [\![\Gamma]\!]_{(\mu L)} \sqsubseteq (\mu L)$, which follows from (1).

$\sqsupseteq$: We need to show that $\{\Gamma\}(\mu L) +\!\!+_{\mathrm{dom}\,(x \mapsto e,\Gamma)} [\![x \mapsto e, \Gamma]\!]_{\{\Gamma\}(\mu L)} \sqsubseteq \{\Gamma\}(\mu L)$. For dom Γ, this follows from (7), so we show $[\![e]\!]_{\{\Gamma\}(\mu L)} \sqsubseteq (\mu L)\, x = [\![e]\!]_{(\mu L)}$, which follows from the monotonicity of $[\![e]\!]_{\_}$ and case $\sqsubseteq$.

To show the lemma, $(\mu L) = (\mu R)$, we use the antisymmetry of $\sqsubseteq$ and the leastness of least fixed points:

$\sqsubseteq$: We need to show that $L\,(\mu R) = \mu R$, i.e.

- $\rho\, y = (\mu R)\, y$ for $y \notin \mathrm{dom}\ (x \mapsto e, \Gamma)$, which follows from (6),
- $[\![e']\!]_{\mu R} = (\mu R)\, y$ for $y \mapsto e' \in \Gamma$, which follows from (4) and (9) and
- $[\![e]\!]_{\mu R} = (\mu R)\, x$, which follows from (5) and (9).

$\sqsupseteq$: Now we have to show that $R\ (\mu L) = (\mu L)$, i.e.

- $\rho\, y = (\mu L)\, y$ for $y \notin \mathrm{dom}\ (x \mapsto e, \Gamma)$, which follows from (3),
- $[\![e']\!]_{\{\Gamma\}(\mu L)} = (\mu L)\, y$ for $y \mapsto e' \in \Gamma$, which follows from (1) and (10), and
- $[\![e]\!]_{\{\Gamma\}(\mu L)} = (\mu L)\, x$, which follows from (2) and (10). □

The final lemma required for the correctness proof shows that the denotation of a set of bindings with only fresh variables can be merged with the heap it was defined over:

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

Lemma 4 (*Merging the heap semantics*)
If dom Γ is fresh with regard to Δ and ρ, then

$$\{\Gamma\}\{\Delta\}\rho = \{\Gamma, \Delta\}\rho.$$

Proof
We use the antisymmetry of $\sqsubseteq$, and the leastness of least fixed points.

$\sqsubseteq$: We need to show that $\{\Delta\}\rho +\!+_{\mathrm{dom}\,\Gamma}[\![\Gamma]\!]_{\{\Delta,\Gamma\}\rho} = \{\Delta, \Gamma\}\rho$, which we verify pointwise.

— For $x \in \mathrm{dom}\,\Gamma$, this follows directly from Lemma 1.
— For $x \notin \mathrm{dom}\,\Gamma$, this holds as the variables bound in Γ are fresh, so the bindings in $\{\Delta\}\rho$ keep their semantics.

$\sqsupseteq$: We need to show that $\rho +\!+_{\mathrm{dom}\,(\Gamma,\Delta)}[\![\Gamma, \Delta]\!]_{\{\Gamma\}\{\Delta\}\rho} = \{\Gamma\}\{\Delta\}\rho$.

— For $x \in \mathrm{dom}\,\Gamma$, this follows from unrolling the fixed point on the right hand side once.
— For $x \mapsto e \in \mathrm{dom}\,\Delta$ (and hence $x \notin \mathrm{dom}\,\Gamma$), we have

$$\begin{aligned}
&(\rho +\!+_{\mathrm{dom}\,(\Gamma,\Delta)}[\![\Gamma, \Delta]\!]_{\{\Gamma\}\{\Delta\}\rho})\, x \\
&\qquad = [\![e]\!]_{\{\Gamma\}\{\Delta\}\rho} && \{\ \text{by Lemma 1}\ \} \\
&\qquad = [\![e]\!]_{\{\Delta\}\rho} && \{\ \text{because dom } \Gamma \text{ is fresh with regard to e}\ \} \\
&\qquad = (\{\Delta\}\rho)\, x && \{\ \text{by unrolling the fixed point}\ \} \\
&\qquad = ([\![\Gamma]\!]_{\{\Delta\}\rho})\, x && \{\ \text{because } x \notin \mathrm{dom}\,\Gamma \text{ and Lemma 1}\ \}.
\end{aligned}$$

— For $x \notin \mathrm{dom}\,\Gamma \cup \mathrm{dom}\,\Delta$, we have $\rho\, x$ on both sides.

□

3.1 Discussions of modifications

Our main Theorem 1 and the generalization in Theorem 2 differ from Launchbury's corresponding Theorem 2. The additional requirement that the judgments are closed is discussed in Section 2.3.

Furthermore, the second part of Theorem 2 is phrased differently. Launchbury states $\{\Gamma\}\rho \leqslant \{\Delta\}\rho$, where $\rho \leqslant \rho'$ is defined as $(\forall x.\, \rho\, x \neq \bot \implies \rho x = \rho' x)$, i.e. ρ' agrees with ρ on all variables that have a meaning in ρ.

The issue with this definition is that there are two reasons why $\{\Gamma\}\rho\, x = \bot$ can hold: Either $x \notin \mathrm{dom}\,\Gamma$, or $x \in \mathrm{dom}\,\Gamma$, but bound to a diverging value. Only the first case is intended here, and actually $\leqslant$ is used as if only that case can happen, e.g. in the treatment of Var in the correctness proof. We therefore avoid the problematic $\leqslant$ relation and explicitly show $\{\Gamma\}\rho =_{|\mathrm{dom}\,\Gamma} \{\Delta\}\rho$.

4 Adequacy

A correctness theorem for a NS is not worth much on its own. Imagine a mistake in side condition of the Let rule that accidentally prevents any judgment to be derived for programs with a `let` – the correctness theorem would still hold.

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

So we want to ensure that all programs that have a meaning, in our case according to the denotational semantics, also have a derivation:

Theorem 3 (*Adequacy*)
For all e, Γ and L, if $[\![e]\!]_{\{\Gamma\}} \neq \bot$, then there exists Δ and v so that $\Gamma : e \Downarrow_L \Delta : v$.

The proof uses a modified denotational semantics that keeps track of the number of steps required to determine the non-bottomness of e, which we will now introduce, prove adequate and show its relationship to the standard denotational semantics.

4.1 The resourced denotational semantics

The domain used to count the resources is a solution to the equation $\mathsf{C} = \mathsf{C}_\bot$. The lifting is done by the injection function $C : \mathsf{C} \to \mathsf{C}$, so the elements are $\bot \sqsubset C\ \bot \sqsubset C\ (C\ \bot) \sqsubset \cdots \sqsubset C^\infty$ – this is isomorphic to the extended naturals. We use r for variables ranging over C.

The resourced semantics $\mathcal{N}[\![e]\!]_\rho\ r$ now takes an additional argument $r \in \mathsf{C}$, which indicates the number of steps the semantics is still allowed to perform: Every recursive call in the definition of $\mathcal{N}[\![e]\!]_\rho\ r$ peels off one application of C until none are left.

The intuition is that if we pass in an infinite number of resources, the two semantics coincide:

$$\forall x.\, \rho\ x = \sigma\ x\ C^\infty \implies [\![e]\!]_\rho = \mathcal{N}[\![e]\!]_\sigma\ C^\infty,$$

as Launchbury puts it. While the intuition is true, it cannot be stated that naively: Because the semantics of an expression is now a function taking a C, this needs to be reflected in the domain equation, so we obtain a different domain, as observed by Sánchez-Gil *et al.* (2011):

$$\mathsf{CValue} = ((\mathsf{C} \to \mathsf{CValue}) \to (\mathsf{C} \to \mathsf{CValue}))_\bot,$$

the lifting and the projection functions are hence

$$\mathsf{CFn}\,(\_) : (\mathsf{C} \to \mathsf{CValue}) \to (\mathsf{C} \to \mathsf{CValue}) \to \mathsf{CValue}$$
$$\_ \downarrow_{\mathsf{CFn}} \_ : \mathsf{CValue} \to (\mathsf{C} \to \mathsf{CValue}) \to (\mathsf{C} \to \mathsf{CValue}).$$

We use σ for variables ranging over resourced environments, $\sigma \in \mathsf{Var} \to (\mathsf{C} \to \mathsf{CValue})$.

The definition of the resourced semantics resembles the definition of the standard semantics, with some resource bookkeeping added:

$$\begin{aligned}
\mathcal{N}[\![e]\!]_\sigma\ \bot &\coloneqq \bot \\
\mathcal{N}[\![\lambda x.\, e]\!]_\sigma\ (C\ r) &\coloneqq \mathsf{CFn}\,(\lambda v.\mathcal{N}[\![e]\!]_{\sigma \sqcup \{x \mapsto v\}}|_r) \\
\mathcal{N}[\![e\ x]\!]_\sigma\ (C\ r) &\coloneqq ((\mathcal{N}[\![e]\!]_\sigma\ r) \downarrow_{\mathsf{CFn}} (\sigma\ x)|_r)\ r \\
\mathcal{N}[\![x]\!]_\sigma\ (C\ r) &\coloneqq \sigma\ x\ r \\
\mathcal{N}[\![\mathsf{let}\ \Delta\ \mathsf{in}\ e]\!]_\sigma\ (C\ r) &\coloneqq \mathcal{N}[\![e]\!]_{\{\Delta\}\sigma}\ r
\end{aligned}$$

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

where $f|_r$ restricts a function f with domain C to take at most r resources: $f|_r := (\lambda r'.\, f\ (r \sqcap r'))$.

The semantics of the heap is defined as before:

$$\mathcal{N}\{\!|\Gamma|\!\}\sigma := (\mu\sigma'.\, \sigma +\!\!+_{\mathrm{dom}\,\Gamma} \mathcal{N}[\![\Gamma]\!]_{\sigma'}).$$

Given the similarity between this semantics and the standard semantics, it is not surprising that Lemmas 1–4 hold as well. In fact, in the formal development, they are stated and proven abstractly, using *locales* (Ballarin, 2014) as a modularization tool, and then simply instantiated for both variants of the semantics.

The correctness lemma needs some adjustments, as a more evaluated expression requires fewer resources. We therefore obtain an inequality:

Lemma 5 (*Correctness, resourced*)
If $\Gamma : e \Downarrow_L \Delta : v$ holds and is closed, then for all environments σ we have $\mathcal{N}[\![e]\!]_{\{\!|\Gamma|\!\}\sigma} \sqsubseteq \mathcal{N}[\![v]\!]_{\{\!|\Delta|\!\}\sigma}$ and $(\mathcal{N}\{\!|\Gamma|\!\}\sigma)|_{\mathrm{dom}\,\Gamma} \sqsubseteq (\mathcal{N}\{\!|\Delta|\!\}\sigma)|_{\mathrm{dom}\,\Gamma}$.

Proof
Analogously to the proof of Theorem 2. □

4.2 Denotational black holes

The major difficulty in proving computational adequacy is the blackholing behavior of the operational semantics: During the evaluation of a variable x the corresponding binding is removed from the heap. Operationally, this is desirable: If the variable is called again during its own evaluation, we would have an infinite loop anyways.

But obviously, the variable is still mentioned in the current configuration, and simply removing the binding will change the denotation of the configuration in unwanted ways: There is no hope of proving $\mathcal{N}[\![e]\!]_{\mathcal{N}\{\!|x \mapsto e, \Gamma|\!\}} = \mathcal{N}[\![e]\!]_{\mathcal{N}\{\!|\Gamma|\!\}}$.

But we can prove a weaker statement, which reflects the idea of "not using x during its own evaluation" more closely:

Lemma 6 (*Denotational blackholing*)

$$\mathcal{N}[\![e]\!]_{\mathcal{N}\{\!|x \mapsto e, \Gamma|\!\}} r \neq \bot \implies \mathcal{N}[\![e]\!]_{\mathcal{N}\{\!|\Gamma|\!\}} r \neq \bot$$

This is a consequence of the following lemma, which states that during the evaluation of an expression using finite resources, only fewer resources will be passed to the members of the environment (which are of type $\mathsf{C} \to \mathsf{CValue}$):

Lemma 7

$$\mathcal{N}[\![e]\!]_\sigma|_{C\ r} = \mathcal{N}[\![e]\!]_{(\sigma|_r)}|_{C\ r}$$

Proof
by induction on the expression e.

In order to show $\mathcal{N}[\![e]\!]_\sigma|_{C\ r} = \mathcal{N}[\![e]\!]_{(\sigma|_r)}|_{C\ r}$, it suffices to show $\mathcal{N}[\![e]\!]_\sigma\ (C\ r') = \mathcal{N}[\![e]\!]_{(\sigma|_r)}\ (C\ r)$, for an arbitrary $r' \sqsubseteq r$.

The critical case is the one for variables, where $e = x$. We have

$$\mathcal{N}[\![x]\!]_\sigma\ (C\ r') = \sigma\ x\ r' = (\sigma\ x|_r)\ r' = \mathcal{N}[\![x]\!]_{(\sigma|_r)}\ (C\ r')$$

as $r' \sqsubseteq r$.

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

In the other cases, the result follows from the fact that nested expressions are evaluated with r' resources or, in the case of lambda abstraction, wrapped inside a $|_{r'}$ restriction operator.

For the case of let, a related lemma for heaps needs to be proven by parallel fixed-point induction, namely $\forall r.\, (\mathcal{N}\{\!|\Gamma|\!\}\sigma)|_r = (\mathcal{N}\{\!|\Gamma|\!\}(\sigma|_r))|_r$. □

Equipped with this lemma, we can begin the

Proof of Lemma 6

Let r' be the least resource such that $\mathcal{N}[\![e]\!]_{\mathcal{N}\{\!|x \mapsto e, \Gamma|\!\}}(C\ r') \neq \bot$. Such an r' exists by the assumption, and $C\ r' \sqsubseteq r$, and by the continuity of the semantics $r' \neq C^\infty$. In particular, $\mathcal{N}[\![e]\!]_{\mathcal{N}\{\!|x \mapsto e, \Gamma|\!\}} r' = \bot$.

We first show

$$\mathcal{N}\{\!|x \mapsto e, \Gamma|\!\}|_{r'} \sqsubseteq \mathcal{N}\{\!|\Gamma|\!\} \tag{*}$$

by bounded fixed-point induction. So given an arbitrary $\sigma \sqsubseteq \mathcal{N}\{\!|x \mapsto e, \Gamma|\!\}$, we may assume $\sigma|_{r'} \sqsubseteq \mathcal{N}\{\!|\Gamma|\!\}$ and have to prove $\mathcal{N}[\![x \mapsto e, \Gamma]\!]_\sigma|_{r'} \sqsubseteq \mathcal{N}\{\!|\Gamma|\!\}$, which we do point-wise:

For $y \mapsto e' \in \Gamma$, this follows from

$$\begin{aligned}
\mathcal{N}[\![x \mapsto e, \Gamma]\!]_\sigma|_{r'}\ y &= \mathcal{N}[\![e']\!]_\sigma|_{r'} \\
&= \mathcal{N}[\![e']\!]_{\sigma|_{r'}}|_{r'} && \{\text{ by Lemma 1 }\} \\
&\sqsubseteq \mathcal{N}[\![e']\!]_{\sigma|_{r'}} \\
&\sqsubseteq \mathcal{N}[\![e']\!]_{\mathcal{N}\{\!|\Gamma|\!\}} && \{\text{ by the induction hypothesis }\} \\
&= \mathcal{N}\{\!|\Gamma|\!\}\ y && \{\text{ by Lemma 1 }\}
\end{aligned}$$

while for x, this follows from

$$\begin{aligned}
&\mathcal{N}[\![x \mapsto e, \Gamma]\!]_\sigma|_{r'}\ x \\
&= \mathcal{N}[\![e]\!]_\sigma|_{r'} \\
&\sqsubseteq \mathcal{N}[\![e]\!]_{\mathcal{N}\{\!|x \mapsto e, \Gamma|\!\}}|_{r'} && \{\text{ using} \sigma \sqsubseteq \mathcal{N}\{\!|x \mapsto e, \Gamma|\!\}\} \\
&= \bot && \{\text{by the choice of } r'\} \\
&= \mathcal{N}\{\!|\Gamma|\!\}\ x && \{\text{as } x \notin \text{dom } \Gamma\}.
\end{aligned}$$

So we can conclude the proof with

$$\begin{aligned}
\bot \sqsubset \mathcal{N}[\![e]\!]_{\mathcal{N}\{\!|x \mapsto e, \Gamma|\!\}}(C\ r') &&& \{\text{ by the choice of } r' \} \\
= \mathcal{N}[\![e]\!]_{\mathcal{N}\{\!|x \mapsto e, \Gamma|\!\}|_{r'}}(C\ r') &&& \{\text{ by Lemma 1 }\} \\
\sqsubseteq \mathcal{N}[\![e]\!]_{\mathcal{N}\{\!|\Gamma|\!\}}(C\ r') &&& \{\text{ by (*) }\} \\
\sqsubseteq \mathcal{N}[\![e]\!]_{\mathcal{N}\{\!|\Gamma|\!\}} r &&& \{\text{ as } C\ r' \sqsubseteq r \}
\end{aligned}$$

□

4.3 Resourced adequacy

With the necessary tools in place to handle blackholing, we can do the adequacy proof for the resourced semantics:

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

Lemma 8 (*Resourced semantics adequacy*)
For all e, Γ and L, if $\mathcal{N}[\![e]\!]_{\{\Gamma\}}\ r \neq \bot$, then there exists Δ and v so that $\Gamma : e \Downarrow_L \Delta : v$.

Proof
Because the semantics is continuous, it suffices to show this for $r = C^n\ \bot$, and perform induction on this n, with arbitrary e, Γ and L.

The case $r = C^0\ \bot = \bot$ is vacuously true, as $\mathcal{N}[\![e]\!]_{\{\Gamma\}}\ \bot = \bot$.

For the inductive case assume that the lemma holds for r, and that $\mathcal{N}[\![e]\!]_{\{\Gamma\}}\ (C\ r) \neq \bot$. We proceed by case analysis on the expression e.

Case: $e = x$.
From the assumption, we know that $\Gamma = x \mapsto e', \Gamma'$ for some e' and Γ', as otherwise the denotation would be bottom, and furthermore that $\mathcal{N}[\![e']\!]_{\mathcal{N}\{x \mapsto e', \Gamma'\}}\ r \neq \bot$

With Lemma 6 this implies $\mathcal{N}[\![e']\!]_{\mathcal{N}\{\Gamma'\}}\ r \neq \bot$, so we can apply the induction hypothesis and obtain Δ and v with $\Gamma' : e' \Downarrow_{L \cup \{x\}} \Delta : v$. This implies $x \mapsto e', \Gamma' : x \Downarrow_L \Delta : v$ by rule Var, as desired.

Case: $e = e'\ x$.
Assume that $\mathrm{fv}(\Gamma, e') \subseteq L$. We do not lose generality here: If we can show a derivation in the NS with a larger set of variables to avoid than required, then the same derivation is also valid with the required set L.

From the assumption, we know that $(\mathcal{N}[\![e']\!]_{\mathcal{N}\{\Gamma\}}\ r \downarrow_{\mathsf{CFn}} (\mathcal{N}\{\Gamma\}\ x)|_r)\ r \neq \bot$. In particular $(\mathcal{N}[\![e']\!]_{\mathcal{N}\{\Gamma\}}\ r) \neq \bot$, so by the induction hypothesis, we have Δ, y and e'' with $\Gamma : e' \Downarrow_L \Delta : \lambda y.\, e''$, the first hypothesis of App.

This judgment is closed by our extra assumption, so we use Lemma 5 to ensure that $\mathcal{N}[\![e']\!]_{\mathcal{N}\{\Gamma\}} \sqsubseteq \mathcal{N}[\![\lambda y.\, e'']\!]_{\mathcal{N}\{\Delta\}}$ and $\mathcal{N}\{\Gamma\} \sqsubseteq \mathcal{N}\{\Delta\}$. We can insert that into the inequality above to calculate

$$
\begin{aligned}
\bot &\sqsubset (\mathcal{N}[\![e']\!]_{\mathcal{N}\{\Gamma\}}\ r \downarrow_{\mathsf{CFn}} (\mathcal{N}\{\Gamma\}\ x)|_r)\ r \\
&\sqsubseteq (\mathcal{N}[\![\lambda y.\, e'']\!]_{\mathcal{N}\{\Delta\}}\ r \downarrow_{\mathsf{CFn}} (\mathcal{N}\{\Delta\}\ x)|_r)\ r \\
&\sqsubseteq (\mathcal{N}[\![\lambda y.\, e'']\!]_{\mathcal{N}\{\Delta\}}\ r \downarrow_{\mathsf{CFn}} \mathcal{N}\{\Delta\}\ x)\ r \\
&\sqsubseteq (\mathsf{CFn}\,(\lambda v.\, \mathcal{N}[\![e'']\!]_{\mathcal{N}\{\Delta\} \sqcup \{y \mapsto v\}}) \downarrow_{\mathsf{CFn}} \mathcal{N}\{\Delta\}\ x)\ r \\
&= \mathcal{N}[\![e'']\!]_{\mathcal{N}\{\Delta\} \sqcup \{y \mapsto (\mathcal{N}\{\Delta\}\ x)\}}\ r \\
&= \mathcal{N}[\![e''[x/y]]\!]_{\mathcal{N}\{\Delta\}}\ r \qquad \{\text{ by Lemma 2 }\}
\end{aligned}
$$

which, using the induction hypothesis again, provides us with Θ and v so that the second hypothesis of App, $\Delta : e''[x/y] \Downarrow_L \Theta : v$, holds, concluding this case.

Case: $e = \lambda y.\, e'$
This case follows immediately from rule Lam with $\Delta = \Gamma$ and $v = \lambda y.\, e'$.

Case: $e = \mathsf{let}\ \Delta\ \mathsf{in}\ e'$
We have

$$
\begin{aligned}
&\bot \\
\sqsubset\ &\mathcal{N}[\![\mathsf{let}\ \Delta\ \mathsf{in}\ e']\!]_{\mathcal{N}\{\Gamma\}}\ r \\
\sqsubseteq\ &\mathcal{N}[\![e']\!]_{\mathcal{N}\{\Delta\}\mathcal{N}\{\Gamma\}} \\
=\ &\mathcal{N}[\![e']\!]_{\mathcal{N}\{\Delta,\Gamma\}} \qquad \{\text{ by Lemma 4 }\}
\end{aligned}
$$

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

so we have Θ and v with $\Delta, \Gamma : e' \Downarrow_L \Theta : v$ and hence $\Gamma : \mathsf{let}\ \Delta\ \mathsf{in}\ e' \Downarrow_L \Theta : v$ by rule LET, as desired. □

4.4 Relating the denotational semantics

Lemma 8 is almost what we want, but it talks about the resourced denotational semantics. In order to obtain that result for the standard denotational semantics, we need to relate these two semantics. We cannot simply equate them, as they have different denotational domains Value and $\mathsf{C} \to \mathsf{CValue}$. So we are looking for a relation $\lhd\rhd$ between Value and CValue that expresses the intuition that they behave the same, if the latter is given infinite resources. In particular, it is specified by the two equations

$$\bot \lhd\rhd \bot$$

and

$$(\forall x\, y.\, x \lhd\rhd y\ C^\infty \implies f\ x \lhd\rhd g\ y\ C^\infty) \iff \mathsf{Fn}(f) \lhd\rhd \mathsf{CFn}(g).$$

Unfortunately, this is not admissible as an inductive definition, as it is self-referential in a non-monotone way, so the construction of this relation is non-trivial. This was observed and performed by Sánchez-Gil *et al.* (2011), and we have subsequently implemented this construction in Isabelle.

We lift this relation to environments $\rho \in \mathsf{Env}$ and resourced environments $\sigma \in \mathsf{Var} \to (\mathsf{C} \to \mathsf{Value})$ by

$$\rho \lhd\rhd^* \sigma \iff \forall x.\, \rho\ x \lhd\rhd \sigma\ x\ C^\infty.$$

This allows us to state precisely how the two denotational semantics are related:

Lemma 9 (*The denotational semantics are related*)
For all environments $\rho \in \mathsf{Env}$ and $\sigma \in \mathsf{Var} \to (\mathsf{C} \to \mathsf{Value})$ with $\rho \lhd\rhd^* \sigma$, we have

$$[\![e]\!]_\rho \lhd\rhd \mathcal{N}[\![e]\!]_\sigma\ C^\infty.$$

Proof
Intuitively, the proof is obvious: As we are only concerned with infinite resources, all the resource counting added to the denotational semantics becomes moot and the semantics are obviously related. A more rigorous proof can be found in Sánchez-Gil *et al.* (2011) and in our formal verification. □

Corollary 10
For all heaps Γ, we have $\{\Gamma\} \lhd\rhd^* \mathcal{N}\{\Gamma\}$.

Proof
by parallel fixed-point induction and Lemma 9. □

4.5 Concluding the adequacy

With this in place, we can give the

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

$$\frac{\Gamma : e \Downarrow_L \Delta : \lambda y.e' \qquad y \mapsto x, \Delta : e' \Downarrow_L \Theta : v}{\Gamma : e\, x \Downarrow_L \Theta : v}\textsc{App}', \qquad \frac{x \mapsto e, \Gamma : e \Downarrow_L \Delta : v}{x \mapsto e, \Gamma : x \Downarrow_L \Delta : v}\textsc{Var}',$$

Fig. 2. Launchbury alternative natural semantics.

proof of Theorem 3

By Corollary 10 we have $\{\Gamma\} \mathrel{\triangleleft\triangleright}^* \mathcal{N}\{\Gamma\}$, and with Lemma 9 this implies $[\![e]\!]_{\{\Gamma\}} \mathrel{\triangleleft\triangleright} \mathcal{N}[\![e]\!]_{\mathcal{N}\{\Gamma\}}\, C^\infty$.

With our assumption $[\![e]\!]_{\{\Gamma\}} \neq \bot$ and the definition of $\mathrel{\triangleleft\triangleright}$ this ensures that $\mathcal{N}[\![e]\!]_{\{\Gamma\}}\, C^\infty \neq \bot$, and we can apply Lemma 8, as desired. □

4.6 Discussions of modifications

Our adequacy proof diverges quite a bit from Launchbury's. As this new proof constitutes a major part of this paper's contribution, we discuss the differences in greater detail.

Launchbury performs the adequacy proof by introducing an ANS that is closer to the denotational semantics than the original NS. He replaces the rules App and Var with the two rules given in Figure 2. There are three differences to be spotted:

1. In the rule for applications, instead of substituting the argument x for the parameter y, the variable y is added to the heap, bound to x, adding an *indirection*.
2. In the rule for variables, no update is performed: Even after x has been evaluated to the value v, the binding x on the heap is not modified at all.
3. Also in the rule for variables, no blackholing is performed: The binding for x stays on the heap during its evaluation.

Without much ado, Launchbury states that the original NS and the ANS are equivalent, which is intuitively obvious. Unfortunately, it turned out that a rigorous proof of this fact is highly non-trivial, as the actual structure of the heaps during evaluation differs a lot: The modification to the application rule causes many indirections, which need to be taken care of. Furthermore, the lack of updates in the variable rules causes possibly complex, allocating expressions to be evaluated many times, each time adding further copies of already existing expressions to the heap. On the other side, the updates in the original semantics further obscure the relationship between the heaps in the original and the alternative semantics. On top of all that add the technical difficulty that is due to naming issues: Variables that are fresh in one derivation might not be fresh in the other, and explicit renamings need to be carried along.

Sánchez-Gil *et al.* have attempted to perform this proof. They broke it down into two smaller steps, going from the original semantics to one with only the variable rule changes (called No-update natural semantics, NNS), and from there to the ANS. So far, they have performed the second step, the equivalence between NNS and ANS, in a pen-and-paper proof (2014), while relation between NS and NNS has yet resisted a proper proof.

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

Considering these difficulties, we went to a different path, and bridged the differences not on the side of the NS, but on the denotational side, which turned out to work well:

1. The denotational semantics for lambda expressions involves a change to the environment ($[\![\lambda x.\, e]\!]_\rho := \mathsf{Fn}\,(\lambda v.[\![e]\!]_{\rho \sqcup \{x \mapsto v\}})$), while the NS uses substitution into the expression: $e[x/y]$.
   This difference is easily bridged on the denotational side by the substitution Lemma 2, which is needed anyways for the correctness proof. See the last line of the application case in the proof of Lemma 8 for this step.
2. The removal of updates had surprisingly no effect on the adequacy proof: The main chore of the adequacy proof is to produce evidence for the *assumptions* of the corresponding NS inference rule, which is then, in the last step, applied to produce the desired judgment. The removal of updates only changes the *conclusion* of the rule, so the adequacy proof is unchanged.
   Of course updates are not completely irrelevant, and they do affect the adequacy proof indirectly. The adequacy proof uses the correctness theorem for the resourced NS (Lemma 5), and there the removal of updates from the semantics would make a noticeable difference.
3. Finally, and most trickily, there is the issue of blackholing. We explain our solution in Section 4.2, which works due to a small modification to the resourced denotational semantics.
   Our proof relies on the property that when we calculate the semantics of $\mathcal{N}[\![e]\!]_\sigma\, r$, we never pass more than r resources to the values bound in σ (Lemma 1). This concurs with our intuition about resources.
   In the original definition of the resourced semantics, this lemma does not hold: The equation for lambda expression ignores the resources passed to it and returns a function involving the semantics of the body:

$$\mathcal{N}[\![\lambda x.\, e]\!]_\sigma\ (C\ r) := \mathsf{CFn}\,(\lambda v.\mathcal{N}[\![e]\!]_{\sigma \sqcup \{x \mapsto v\}})$$

   With that definition, $\mathcal{N}[\![\lambda x.\, y]\!]_\sigma\ (C\ \bot) = \mathsf{CFn}\,(\sigma\ y)$, which depends on $\sigma\ y\ r$ for all r, contradicting Lemma 1.
   Therefore we restrict the argument of $\mathsf{CFn}\,(\_)$ to cap any resources passed to it at r. Analogously, we adjust the equation for applications to cap any resources passed to the value of the argument in the environment, $\sigma\ x$.
   These modifications do not affect the proof relating the two denotational semantics (Lemma 9), as there we always pass infinite resources, and $|_{C^\infty}$ is the identity function.

5 Related work

A large number of developments on formal semantics of functional programming languages in the last two decades build on Launchbury's work; here is a short selection: Van Eekelen & de Mol (2004) add strictness annotations to the syntax and semantics of Launchbury's work. Nakata & Hasegawa (2009) define a small-step

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

semantics for call-by-need and relate it to a Launchbury-derived big-step semantics. Nakata (2010) modifies the denotational semantics to distinguish direct cycles from looping recursion. Sánchez-Gil *et al.* (2010) extend Launchbury's semantics with distributed evaluation. Baker-Finch *et al.* (2000) create a semantics for parallel call-by-need based on Launchbury's.

While many of them implicitly or explicitly rely on the correctness and adequacy proof as spelled out by Launchbury, some stick with the original definition of the heap semantics using $\sqcup$, for which the proofs do not got through (Baker-Finch *et al.*, 1999; Eekelen & Mol, 2004; Nakata & Hasegawa, 2009; Sánchez-Gil *et al.*, 2010), while others use right-sided updates, without further explanation (Baker-Finch *et al.*, 2000; Nakata, 2010). The work by Baker-Finch *et al.* is particularly interesting, as they switched from the original to the fixed definition between the earlier tech report and the later ICFP publication, unfortunately without motivating that change.

Such disagreement about the precise definition of the semantics is annoying, as it creates avoidable incompatibilities between these publications. We hope that our fully rigorous treatment will resolve this confusion and allows future work to standardize on the "right" definition.

Furthermore, none of these works discuss the holes in Launchbury's adequacy proof, even those that explicitly state the adequacy of their extended semantics. Our adequacy proof is better suited for such extensions, as it is rigorous and furthermore avoids the intermediate NS.

This list is just a small collection of many more Launchbury-like semantics. Often the relation to a denotational semantics is not stated, but nevertheless they are standing on the foundations laid by Launchbury. Therefore, it is not surprising that others have worked on formally fortifying these foundations as well:

In particular, Sánchez-Gil *et al.* worked toward rigorously proving Launchbury's semantics correct and adequate. They noted that the relation between the standard and the resourced denotational semantics is not as trivial as it seemed at first, and worked out a detailed pen-and-paper proof (2011). We have formalized this, fixing mistakes in the proof, and build on their result here (Lemma 9).

They also bridged half the gap between Launchbury's natural and ANS (Sánchez-Gil *et al.*, 2014), and plan to bridge the other half. We avoided these very tedious proofs by bridging the difference on the denotational side (Section 4.6).

As a step toward a mechanization of their work in Coq, they address the naming issues and suggest a mixed representation, using de Bruijn indices for locally bound variables and names for free variables (2012). This corresponds to our treatment of names in the formal development, using the Nominal logic machinery (Urban & Kaliszyk, 2012) locally but not for names bound in heaps.

Having an implementation of the present work in Isabelle does not only give us great assurance about the correctness of our work, but additionally is a tool to formalize further work. We have used these semantics to prove that the compiler analysis and transformation "Call Arity" (Breitner, 2015a), which is implemented in the Haskell compiler GHC, is semantics-preserving and does not degrade performance (Breitner, 2015b, 2015c). We measure performance on a sufficiently

abstract level by counting allocations on the heap, for which Launchbury's semantics provides just the right level of detail.

6 Conclusion and future work

Was beweisbar ist, soll in der Wissenschaft nicht ohne Beweis geglaubt werden.[1]
— Richard Dedekind, *Was sind und was sollen die Zahlen*, 1888

In computer science, we are in the happy situation that we can use the tools of logic and mathematics to describe our artifacts, and hence to give proofs of our claims, and indeed, we often do. But we are also in the unfortunate situation that our models become large and complex, so our proofs becomes large and complex, without necessarily becoming harder or more interesting. So classical pen-and-paper proofs are likely to be incomplete or erroneous, without anyone noticing.

We can help ourselves here by using our computers check the proofs. This is what we have done: We took an established formalism and with the help of the theorem prover Isabelle, weeded out all mistakes, filled all the holes and clarified a lot of details.

Such mechanization is a thankless task: With today's theorem provers, it is still very laborious, and at the end one usually finds that everything is all right, and even though there were no complete proofs before, the results still hold. Nevertheless, we should invest the effort to fortify at least our foundations this way. We deem Launchbury's semantics important enough to warrant this effort.

We used our formalization to prove a compiler transformation correct, but the formalization gap between Launchbury's or Sestoft's core calculus, and the full Core language used by GHC is rather large. It would be very useful to have GHC Core formalized, possibly building on Eisenberg's specification (2013), allowing for much more use of formal methods in the development of the Haskell compiler.

References

Abramsky, S. (1990) The lazy lambda calculus. In *Research Topics in Functional Programming*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., pp. 65–116.

Baker-Finch, C., King, D., Hall, J. & Trinder, P. (1999) *An Operational Semantics for Parallel Call-by-Need*. Technical Report 99/1. Faculty of Mathematics and Computing, The Open University.

Baker-Finch, C., King, D. J. & Trinder, P. W. (2000) An operational semantics for parallel lazy evaluation. In *Proceedings of ACM SIGPLAN International Conference on Functional Programming*.

Ballarin, C. (2014) Locales: A module system for mathematical theories. *J. Autom. Reason.* **52**(2), 123–153.

Breitner, J. (2013) The correctness of Launchbury's natural semantics for lazy evaluation. *Archive of formal proofs*, Jan. Available at: `http://afp.sf.net/entries/Launchbury.html`, Formal proof development.

[1] What is provable in science, is not to be believed without proof.

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press

Breitner, J. (2015a) Call arity. In *TFP'14*. LNCS, vol. 8843. Springer, pp. 34–50.

Breitner, J. (2015b) Formally proving a compiler transformation safe. In *Proceedings of Haskell Symposium*. ACM.

Breitner, J. (2015c) The safety of call arity. *Archive of formal proofs*, Feb. Available at: `http://afp.sf.net/entries/Call_Arity.shtml`, Formal proof development.

Eekelen, M. van, & Mol, M. de. (2004 January) *Mixed lazy/strict graph semantics*. Technical Report. NIII-R0402. Radboud University Nijmegen.

Eisenberg, R. (2013) *System FC, as Implemented in GHC*.

Launchbury, J. (1993) A natural semantics for lazy evaluation. In *Principles of Programming Languages*, pp. 144–154.

Nakata, K. (2010) Denotational semantics for lazy initialization of letrec: Black holes as exceptions rather than divergence. In *Proceedings of the 7th Workshop on Fixed Points in Computer Science*.

Nakata, K. & Hasegawa, M. (2009) Small-step and big-step semantics for call-by-need. *J. Funct. Program.* **19**(6), 699–722.

Sánchez-Gil, L., Hidalgo-Herrero, M. & Ortega-Mallén, Y. (2010) An operational semantics for distributed lazy evaluation. In *Trends in Functional Programming*, vol. 10. New York, NY, USA: Intellect Books, pp. 65–80.

Sánchez-Gil, L., Hidalgo-Herrero, M. & Ortega-Mallén, Y. (2011) Relating function spaces to resourced function spaces.In *Proceedings of ACM Symposium on Applied Computing*, pp. 1301–1308.

Sánchez-Gil, L., Hidalgo-Herrero, M. & Ortega-Mallén, Y. (2012) A locally nameless representation for a natural semantics for lazy evaluation. In *Proceedings of International Colloquium on Theoretical Aspects of Computing*, pp. 1301–1308.

Sánchez-Gil, L., Hidalgo-Herrero, M. & Ortega-Mallén, Y. (2014) The role of indirections in lazy natural semantics. In *Proceedings of PSI*.

Sestoft, P. (1997) Deriving a lazy abstract machine. *J. Funct. Program.* **7**(4), 231–264.

Urban, C. & Kaliszyk, C. (2012) General bindings and alpha-equivalence in nominal Isabelle. *Logical Methods Comput. Sci.* **8**(2). See `https://lmcs.episciences.org/page/lmcs-ev` for details.

https://doi.org/10.1017/S0956796817000144 Published online by Cambridge University Press