Non-Idempotent Intersection Types for Global State

Miguel Ramos[*]

DCC/FCUP - Dept. of Computer Science, Faculty of Sciences of the University of Porto
Rua do Campo Alegre s/n, 4169-007 Porto, Portugal
LIACC - Artificial Intelligence and Computer Science Laboratory
Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal

Abstract

We show how a type system based on non-idempotent intersection types can be used to characterize termination for an extension of the λ-calculus with a notion of global state.

1 Introduction

Type systems are syntactic methods for ensuring that programs behave well. Traditionally, they are used for ensuring safety during program execution [Mil78]. However, type systems can also be used to ensure other desirable properties of programs, such as termination.

Intersection Types. Type systems based on intersection types not only guarantee termination, but also characterize it [CD78]. Moreover, by considering intersections to be *non-idempotent* [KW99, NM04], none of the power of (idempotent) intersection types is lost but proving termination becomes much simpler. Indeed, while *idempotent* intersection types are purely *qualitative* in nature, *non-idempotent* intersection types have a *quantitative* nature. Very roughly, in type systems based on non-idempotent intersection types, the size of the derivation tree of a well-typed program is a good measure for termination because it strictly decreases with each evaluation step [BKV17]. Remark that associativity, commutativity and idempotency are granted if intersections are denoted by sets. Therefore, it is natural to represent non-idempotent intersections by multisets. This convention will be adopted throughout this work.

Algebraic Effects. Intersection types have been successfully used for reasoning about properties of the *pure* λ-calculus. However, current programming languages are *effectful*: they raise and handle exceptions, perform I/O, sample from distributions, interact with a global state, and so on. Therefore, it is important to understand how intersection types can be used to reason about properties of extensions of the λ-calculus with *effects*. In [Mog89, Mog91], *monads* are used to obtain a uniform denotational semantics for effects and a computational version of λ-calculus where effects are obtained by adding appropriate operations is introduced. However, there is no mention of how computational effects are actually produced. It was only in [PP02] that *algebraic operations* were

[*]Supported by: national funds by the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia -, through the individual research grant 2021.04731.BD; base funding - UIDB/00027/2020 of the Artificial Intelligence and Computer Science Laboratory - LIACC - funded by national funds through the FCT/MCTES (PIDDAC).

introduced as a way of giving monadic semantics to the operations that produce effects: effects are realized by families of operations and monads are generated by their equational theory. Effects that can be represented by an equational theory and whose operations produce the effects at hand are called *algebraic effects.*

Monadic Intersection Types. More recently, intersection types have been successfully used to reason about properties of extensions of the λ-calculus with specific effects, such as probabilistic choice [BL18, LFR21], pure nondeterminism [TK14], global state [dT21, dT23, AKR23], and database queries [CT23]. Notably, in [GTV24], intersection types are used to reason in a uniform way about properties of extensions of the λ-calculus with different combinations of effects, such as output, cost, and pure and probabilistic nondeterminism. This is achieved by extending the λ-calculus with algebraic operations and allowing intersection types to be effectful. This generalizes existing effectful intersection type systems, such the one for probabilistic nondeterminism in [LFR21], where instead of intersection types, *multidistributions* of intersection types are used. The reason why *multidistributions* are adopted instead of *distributions* is addressed in [LFR21], but made more precise in [GTV24]: results involving forms of reversibility, such as subject expansion, are simply not available when monads are not weakly cartesian, *i.e.* when there is loss of information about the performed effect during evaluation. Indeed, for the particular case of algebraic effects, this means that subject expansion only holds for those effects that are represented by an *linear* equational theory, *i.e.* one where equations cannot duplicate variables.

Contribution and Overview. Following the idea in [dT21, dT23] of using intersection types to reason about an extension of the λ-calculus with global state, and the monadic approach of [GTV24] to intersection types, we show how *non-idempotent* intersection types can be used in order to characterize termination for an extension of the λ-calculus with global state, as was first introduced in [AKR23]. At first glance, one would expect this to follow from simply extending a call-by-value (CbV) version of the λ-calculus with a global state and then taking the algebraic approach of [PP02] of adding to the calculus the operations whose equational theory generates the global state monad. Unfortunately, this is not possible due to the global state monad not being *weakly cartesian*, which means that subject expansion does not hold. Indeed, from an algebraic point-of-view, what this means it that the equational theory that generates the global state monad is not *linear*, *i.e.* some of its equations duplicate variables. Therefore, we first need to find an appropriate equational theory and notion of global state. To do this we simply linearize the equational theory that generates the global state monad, obtaining a notion of the global state monad with a log, which turns out to be a particular instance of the more general notion of *update monad* in [AU13]. We call this notion of global state the *persistent* global state monad, and it is for this notion of global state that we develop a type system based on *monadic* non-idempotent intersection types that characterizes termination.

2 Syntax and Operational Semantics

Let $\ell \in \mathcal{L}$ be a location from an enumerable set of locations $\mathcal{L}$. The sets of *values*, *computations*, *global state* and *configurations* of $\Lambda_{\texttt{gs}}$ are generated by the following grammars:

$$\begin{array}{rrcl}
\text{(Values)} & v, w & ::= & x \mid \lambda x.t \\
\text{(Computations)} & t, u & ::= & v \mid vt \mid \texttt{get}_\ell(\lambda x.t) \mid \texttt{set}_\ell(v, u) \\
\text{(Global State)} & s, q & ::= & \epsilon \mid \texttt{upd}_\ell(v, s) \\
\text{(Configurations)} & c & ::= & (t, s)
\end{array}$$

The sets of free and bound variables for terms, states, and configurations are defined as expected. Terms are considered modulo α-equivalence.

Let $\texttt{lkp}_\ell(s)$ be a meta-level algebraic operation. The global state monad is generated by the algebraic theory described by following set of equations between infinitary algebraic operations [PP08]:

$$\begin{array}{rrcl}
1. & \texttt{lkp}_\ell(\texttt{upd}_\ell(v,s)) & = & v \\
2. & \texttt{upd}_{\ell_1}(v,\texttt{upd}_{\ell_2}(w,s)) & = & \texttt{upd}_{\ell_2}(w,\texttt{upd}_{\ell_1}(v,s)) \text{ if } \ell_1 \neq \ell_2 \\
3. & \texttt{upd}_\ell(\texttt{lkp}_\ell(s),s) & = & s \text{ if } \texttt{lkp}_\ell(s) \neq \bot \\
4. & \texttt{upd}_\ell(v,\texttt{upd}_\ell(w,s)) & = & \texttt{upd}_\ell(v,s)
\end{array}$$

Note that equations (3.) and (4.) are not *linear*: in both equations, location ℓ appears twice on the left, but only once on the right. This means that the global state monad is not weakly cartesian, and thus subject expansion does not hold. Fortunately, if we simply drop equations (3.) and (4.) from our algebraic theory, we obtain a new algebraic theory that generates instead what we call the *persistent* global state monad, which is an instance of the update monad of [AU13].

The *small-step semantics* of $\Lambda_{\texttt{gs}}$ is defined over closed configurations, *i.e.* without occurrences of free variables, and consists of the following reduction steps:

$$\frac{}{((\lambda x.t)v,s) \to (t\{x\backslash v\},s)}\ (\texttt{beta}) \qquad \frac{(t,s) \to (t',q)}{(vt,s) \to (vt',q)}\ (\texttt{right})$$

$$\frac{\texttt{lkp}_\ell(s) = v}{(\texttt{get}_\ell(\lambda x.t),s) \to (t\{x\backslash v\},s)}\ (\texttt{get}) \qquad \frac{}{(\texttt{set}_\ell(v,t),s) \to (t,\texttt{upd}_\ell(v,s))}\ (\texttt{set})$$

Note that the operational semantics of $\Lambda_{\texttt{gs}}$ is based on a CBV strategy. In particular, the β-step only fires if the argument is a value. The reflexive and transitive closure of $\to$ is written $\twoheadrightarrow$.

The set of *closed blocked configurations* $\texttt{blocked}$ is the smallest set consisting of configurations of the following two forms:

$$(\texttt{get}_\ell(\lambda x.u),s) \text{ where } \texttt{lkp}_\ell(s) = \bot \qquad (vt,s) \text{ where } (t,s) \in \texttt{blocked}$$

The set of *closed configuration in normal form* is the smallest set consisting of all configurations in $\texttt{blocked}$ plus all *closed configurations* of the form $(\lambda x.t,s)$.

Proposition 2.1 (Syntactic Characterization of Closed Configurations in Normal Form). *Let $c = (t,s)$ be a closed configuration. Then, $c \not\to$ iff $c \in \texttt{blocked}$ or t is a λ-abstraction.*

The set of *unblocked* closed configurations in normal form is the smallest set consisting of all closed configurations of the form $(\lambda x.t,s)$. These configurations are called *final configurations.*

Proposition 2.2 (Syntactic Characterization of Final Configurations). *Let $c = (t,s)$ be a closed configuration. Then, $c \not\to$ and $c \notin \texttt{blocked}$ iff t is a λ-abstraction.*

A configuration c_1 is said to be *terminating* if there exists some final configuration c_2, such that $c_1 \twoheadrightarrow c_2$.

3 Monadic Type System

The key idea behind the monadic type system is to allow non-idempotent intersection types to be effecful. This is obtained by considering a combination of Girard's [Gir87] and Moggi's [Mog89] CBV translations of intuitionistic logic into linear logic and the monadic framework, respectively.

Let T be a monad. For the particular case of function types, we have $!A \multimap T(!A)$, and if we take T to be the persistent global state monad $TX = S^\star \Rightarrow (X \times S^\star)$, where $S^\star$ is the log of state S, and S is $(!A)^{\mathcal{L}}$, we have $!A \multimap S^\star \Rightarrow (!A \otimes \mathcal{S}^\star)$.

The sets of *multiset types*, *value types*, *computation types*, *state types*, and *configuration types*, are generated by the following grammars:

$$\begin{array}{rrcl}
\text{(Multiset Types)} & m, n & ::= & [\tau_i]_{i \in I} \text{ where } I \text{ is a finite set} \\
\text{(Value Types)} & \tau & ::= & m \multimap \mu \\
\text{(Computation Types)} & \mu & ::= & \sigma \Rightarrow \rho \\
\text{(Global State Types)} & \sigma & ::= & \epsilon \mid \mathtt{upd}_\ell(m, \sigma) \\
\text{(Configuration Types)} & \rho & ::= & (m \otimes \sigma)
\end{array}$$

Note that global state types are persistent global states *of* multiset types. Thus, we will consider two global state and global state types to be equal if they are equivalent up-to equation (2). This will also allow us to omit equation (1) and simply assume that any global state type will always be written with the relevant update operation in its outermost position.

The *typing system* of $\Lambda_{\mathtt{gs}}$ is defined by the following set of typing rules:

$$\frac{}{x : [\tau] \vdash x : \tau}\ (\mathtt{ax}) \qquad \frac{\Gamma \vdash v : m \multimap (\sigma' \Rightarrow \rho) \quad \Delta \vdash t : \sigma \Rightarrow (m \otimes \sigma')}{\Gamma \sqcup \Delta \vdash vt : \sigma \Rightarrow \rho}\ (\mathtt{app})$$

$$\frac{\Gamma; x : m \vdash t : \mu}{\Gamma \vdash \lambda x.t : m \multimap \mu}\ (\mathtt{abs}) \qquad \frac{(\Gamma_i \vdash v : \tau_i)_{i \in I}}{\sqcup_{i \in I} \Gamma_i \vdash v : [\tau_i]_{i \in I}}\ (\mathtt{many}) \qquad \frac{\Gamma \vdash v : m}{\Gamma \vdash v : \sigma \Rightarrow (m \otimes \sigma)}\ (\mathtt{unit})$$

$$\frac{\Gamma; x : m \vdash t : \mathtt{upd}_\ell(n, \sigma) \Rightarrow \rho}{\Gamma \vdash \mathtt{get}_\ell(\lambda x.t) : \mathtt{upd}_\ell(m \sqcup n, \sigma) \Rightarrow \rho}\ (\mathtt{get}) \qquad \frac{\Gamma \vdash v : m \quad \Delta \vdash t : \mathtt{upd}_\ell(m, \sigma) \Rightarrow \rho}{\Gamma \sqcup \Delta \vdash \mathtt{set}_\ell(v, t) : \sigma \Rightarrow \rho}\ (\mathtt{set})$$

$$\frac{}{\vdash \epsilon : \epsilon}\ (\mathtt{emp\text{-}state}) \qquad \frac{\Gamma \vdash v : m \quad \Delta \vdash s : \sigma}{\Gamma \sqcup \Delta \vdash \mathtt{upd}_\ell(v, s) : \mathtt{upd}_\ell(m, \sigma)}\ (\mathtt{upd\text{-}state})$$

$$\frac{\Gamma \vdash t : \sigma \Rightarrow \rho \quad \Delta \vdash s : \sigma}{\Gamma \sqcup \Delta \vdash (t, s) : \rho}\ (\mathtt{conf})$$

where $\sqcup$ is used to denote *multiset union.* And we assume the usual notions of contexts and judgements for type system based on multiset type and point the reader to [BKV17] for details. Formal *type derivations* are denoted by Φ. The *size of a type derivation* Φ is denoted by $\mathtt{sz}(\Phi)$ and defined as the number of typing rules of Φ except rule ($\mathtt{many}$) and rules ($\mathtt{emp\text{-}state}$), ($\mathtt{upd\text{-}state}$).

Lemma 3.1 (Typability of Configurations).

1. (Typed Configurations are Unblocked) *If* $\Phi \rhd \Gamma \vdash c : \rho$*, then* $c \notin \mathtt{blocked}$*.*

2. (Final Configurations are Typable) *If* c *is a final configuration, then* $\Phi \rhd \emptyset \vdash c : \rho$*.*

In order to show that our type system is sound and complete with respect to our notion of termination, we must first show the usual subject reduction and expansion properties. Fortunately, we can now take advantage of the *quantitative* nature of non-idempotent intersection types by stating a *weighted* version of the subject reduction property that simplifies the proof of termination.

Lemma 3.2 (Weighted Subject Reduction and Expansion).

1. (WSR) *Let* $\Phi_{c_1} \rhd \emptyset \vdash c_1 : \rho$*. If* $c_1 \to c_2$*, then* $\Phi_{c_2} \rhd \emptyset \vdash c_2 : \rho$*, such that* $\mathtt{sz}(\Phi_{c_1}) > \mathtt{sz}(\Phi_{c_2})$*.*

2. (WSE) *Let c_1 be a closed configuration and $\Phi_{c_2} \triangleright \emptyset \vdash c_2 : \rho$. If $c_1 \to c_2$, then $\Phi_{c_1} \triangleright \emptyset \vdash c_1 : \rho$.*

At this point, we can illustrate why it is necessary to drop equation 3. and 4. in order to have the subject expansion property. Consider the following evaluation step:

$$c_1 = (\texttt{set}_\ell(v_1, t), \texttt{upd}_\ell(v_2, \epsilon)) \to (t, \texttt{upd}_\ell(v_1, \epsilon)) = c_2.$$

and note that $\texttt{upd}_\ell(v_1, \epsilon) = \texttt{upd}_\ell(v_1, \texttt{upd}_\ell(v_2, \epsilon))$ by equation 4. The type derivation for c_2 must be of the following form:

$$\dfrac{\emptyset \vdash t : \texttt{upd}_\ell(m, \epsilon) \Rightarrow \rho \qquad \dfrac{\emptyset \vdash v_1 : m \qquad \dfrac{}{\vdash \epsilon : \epsilon}\,(\texttt{emp-state})}{\emptyset \vdash \texttt{upd}_\ell(v_1, \epsilon) : \texttt{upd}_\ell(m, \epsilon)}\,(\texttt{upd-state})}{\emptyset \vdash (t, \texttt{upd}_\ell(v_1, \epsilon)) : \rho}\,(\texttt{conf})$$

But then it is impossible to build a type derivation for c_1[1]:

$$\dfrac{\dfrac{\emptyset \vdash v_1 : m \qquad \emptyset \vdash t : \texttt{upd}_\ell(m, \epsilon) \Rightarrow \rho}{\emptyset \vdash \texttt{set}_\ell(v_1, t) : \epsilon \Rightarrow \rho}\,(\texttt{set}) \qquad \dfrac{\emptyset \vdash v_2 : [\,] \qquad \dfrac{}{\vdash \epsilon : \epsilon}\,(\texttt{emp-state})}{\emptyset \vdash \texttt{upd}_\ell(v_2, \epsilon) : \texttt{upd}_\ell([\,], \epsilon)}\,(\texttt{upd-state})}{\emptyset \vdash (\texttt{set}_\ell(v_1, t), \texttt{upd}_\ell(v_2, \epsilon)) : ?}\,(\texttt{conf})$$

since $\epsilon \neq \texttt{upd}_\ell([\,], \epsilon)$. It is worth noting that, even though we are using a more sophisticated notion of global state, since we are not losing any information, we can recover the original notion of the global state monad by simply ignoring the log that is present in the persistent global state monad.

Theorem 3.3 (Soundness and Completeness)**.** *Let c be a closed configuration. Then, c is typable iff c is terminating. Moreover, if $\Phi \triangleright \emptyset \vdash c : \rho$, then c terminates in at most $\texttt{sz}(\Phi)$ steps.*

Proof. Let c_1 be a closed configuration. If c_1 is typable, then $c_1 \notin \texttt{blocked}$ by Lemma 3.1.1, and c_1 is terminating by Lemma 3.2.1. If c_1 is terminating, there exists a final configuration c_2, such that $c_1 \twoheadrightarrow c_2$. Clearly, c_2 must be typable by Lemma 3.1.2. Thus, c_1 is typable by Lemma 3.2.2. In both cases, if $\Phi \triangleright \emptyset \vdash c : \rho$, then c terminates in at most $\texttt{sz}(\Phi)$ steps by the weighted nature of Lemma 3.2.1. □

4 Conclusion and Future Work

This work continues the recent trend of using idempotent and non-idempotent intersection types to reason about extensions of the λ-calculus with effects.

As future work, we would like to extend the results obtained in [GTV24] to the persistent global state monad and other effects, such as I/O, whose equational theories have not only an algebraic, but also coalgebraic structure. That is, whose behavior crucially depends on some notion of environment [PP08].

Acknowledgements. We would like to thank the reviewers for their valuable comments and suggestions, which helped us improve the quality of this work.

[1]It is interesting to note that this problem can be solved if we add a typing rule for typing global states that allows us to ignore locations that have not been used so far. However, this will simply lead to other problems, as we illustrate in Appendix A.

References

[AKR23] Sandra Alves, Delia Kesner, and Miguel Ramos. Quantitative global memory. In Helle Hvid Hansen, Andre Scedrov, and Ruy J. G. B. de Queiroz, editors, Logic, Language, Information, and Computation - 29th International Workshop, WoLLIC 2023, Halifax, NS, Canada, July 11-14, 2023, Proceedings, volume 13923 of Lecture Notes in Computer Science, pages 53–68. Springer, 2023.

[AU13] Danel Ahman and Tarmo Uustalu. Update monads: Cointerpreting directed containers. In Ralph Matthes and Aleksy Schubert, editors, 19th International Conference on Types for Proofs and Programs, TYPES 2013, April 22-26, 2013, Toulouse, France, volume 26 of LIPIcs, pages 1–23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[BKV17] Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. Log. J. IGPL, 25(4):431–464, 2017.

[BL18] Flavien Breuvart and Ugo Dal Lago. On intersection types and probabilistic lambda calculi. In David Sabel and Peter Thiemann, editors, Proceedings of the 20th International Symposium on Principles and Practice of Declarative Programming, PPDP 2018, Frankfurt am Main, Germany, September 03-05, 2018, pages 8:1–8:13. ACM, 2018.

[CD78] Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for λ-terms. Arch. Math. Log., 19(1):139–156, 1978.

[CT23] Claudio Sacerdoti Coen and Riccardo Treglia. Properties of a computational lambda calculus for higher-order relational queries. In Giuseppa Castiglione and Marinella Sciortino, editors, Proceedings of the 24th Italian Conference on Theoretical Computer Science, Palermo, Italy, September 13-15, 2023, volume 3587 of CEUR Workshop Proceedings, pages 254–267. CEUR-WS.org, 2023.

[dT21] Ugo de'Liguoro and Riccardo Treglia. Intersection types for a λ-calculus with global store. In Niccolò Veltri, Nick Benton, and Silvia Ghilezan, editors, PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming, Tallinn, Estonia, September 6-8, 2021, pages 5:1–5:11. ACM, 2021.

[dT23] Ugo de'Liguoro and Riccardo Treglia. From semantics to types: The case of the imperative λ-calculus. Theor. Comput. Sci., 973:114082, 2023.

[Gir87] Jean-Yves Girard. Linear logic. Theor. Comput. Sci., 50:1–102, 1987.

[GTV24] Francesco Gavazzo, Riccardo Treglia, and Gabriele Vanoni. Monadic intersection types, relationally. In Stephanie Weirich, editor, Programming Languages and Systems - 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I, volume 14576 of Lecture Notes in Computer Science, pages 22–51. Springer, 2024.

[KW99] A. J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In Andrew W. Appel and Alex Aiken, editors, POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999, pages 161–174. ACM, 1999.

[LFR21] Ugo Dal Lago, Claudia Faggian, and Simona Ronchi Della Rocca. Intersection types and (positive) almost-sure termination. Proc. ACM Program. Lang., 5(POPL):1–32, 2021.

[Mil78] Robin Milner. A theory of type polymorphism in programming. J. Comput. Syst. Sci., 17(3):348–375, 1978.

[Mog89] Eugenio Moggi. Computational lambda-calculus and monads. In Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989, pages 14–23. IEEE Computer Society, 1989.

[Mog91] Eugenio Moggi. Notions of computation and monads. Inf. Comput., 93(1):55–92, 1991.

[NM04] Peter Møller Neergaard and Harry G. Mairson. Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. In Chris Okasaki and Kathleen Fisher, editors, Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming, ICFP 2004, Snow Bird, UT, USA, September 19-21, 2004, pages 138–149. ACM, 2004.

[PP02] Gordon D. Plotkin and John Power. Notions of computation determine monads. In Mogens Nielsen and Uffe Engberg, editors, Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings, volume 2303 of Lecture Notes in Computer Science, pages 342–356. Springer, 2002.

[PP08] Gordon D. Plotkin and John Power. Tensors of comodels and models for operational semantics. In Andrej Bauer and Michael W. Mislove, editors, Proceedings of the 24th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2008, Philadelphia, PA, USA, May 22-25, 2008, volume 218 of Electronic Notes in Theoretical Computer Science, pages 295–311. Elsevier, 2008.

[TK14] Takeshi Tsukada and Naoki Kobayashi. Complexity of model-checking call-by-value programs. In Anca Muscholl, editor, Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings, volume 8412 of Lecture Notes in Computer Science, pages 180–194. Springer, 2014.

A On Ignoring Locations when Typing Global States

It might be tempting to add the following rule to the type system, which allows us to locations that have not been used so far:

$$\frac{\Gamma \vdash s : \sigma \qquad \mathtt{lkp}_\ell(\sigma) = \bot}{\Gamma \vdash \mathtt{upd}_\ell(v, s) : \sigma}\ (\mathtt{upd\text{-}skip\text{-}state})$$

This would allos us to build a derivation for configuration c_1:

$$\dfrac{\dfrac{\emptyset \vdash v_1 : m \qquad \emptyset \vdash t : \mathtt{upd}_\ell(m, \epsilon) \Rightarrow \rho}{\emptyset \vdash \mathtt{set}_\ell(v_1, t) : \epsilon \Rightarrow \rho}\ (\mathtt{set}) \qquad \dfrac{\dfrac{}{\vdash \epsilon : \epsilon}\ (\mathtt{emp\text{-}state})}{\emptyset \vdash \mathtt{upd}_\ell(v_2, \epsilon) : \epsilon}\ (\mathtt{upd\text{-}skip\text{-}state})}{\emptyset \vdash (\mathtt{set}_\ell(v_1, t), \mathtt{upd}_\ell(v_2, \epsilon)) : \rho}\ (\mathtt{conf})$$

However, our type system is not longer syntax directed. Indeed, not only can the type derivation for c_2 be of the form given in the main text, but it can also be of the following form:

$$\dfrac{\emptyset \vdash t : \epsilon \Rightarrow \rho \qquad \dfrac{\dfrac{}{\vdash \epsilon : \epsilon}\ (\mathtt{emp\text{-}state})}{\emptyset \vdash \mathtt{upd}_\ell(v_1, \epsilon) : \epsilon}\ (\mathtt{upd\text{-}skip\text{-}state})}{\emptyset \vdash (t, \mathtt{upd}_\ell(v_1, \epsilon)) : \rho}\ (\mathtt{conf})$$

And then it is again impossible to build a type derivation for c_1:

$$\dfrac{\dfrac{\emptyset \vdash v_1 : [\,] \qquad \emptyset \vdash t : \epsilon \Rightarrow \rho}{\emptyset \vdash \mathtt{set}_\ell(v_1, t) : ? \Rightarrow \rho}\ (\mathtt{set}) \qquad \dfrac{\dfrac{}{\vdash \epsilon : \epsilon}\ (\mathtt{emp\text{-}state})}{\emptyset \vdash \mathtt{upd}_\ell(v_2, \epsilon) : \epsilon}\ (\mathtt{upd\text{-}state})}{\emptyset \vdash (\mathtt{set}_\ell(v_1, t), \mathtt{upd}_\ell(v_2, \epsilon)) : ?}\ (\mathtt{conf})$$

since $\mathtt{upd}_\ell([\,], \epsilon) \neq \epsilon$.