Diagrams for Meaning Preservation*

J. B. Wells[1], Detlef Plump[2], and Fairouz Kamareddine[1]

[1] Heriot-Watt University
http://www.cee.hw.ac.uk/ultra/
[2] University of York
http://www.cs.york.ac.uk/~det/

Abstract. This paper presents an abstract framework and multiple diagram-based methods for proving *meaning preservation*, i.e., that all rewrite steps of a rewriting system preserve the meaning given by an operational semantics based on a rewriting strategy. While previous rewriting-based methods have generally needed the treated rewriting system as a whole to have such properties as, e.g., *confluence*, *standardization*, and/or *termination* or *boundedness* of *developments*, our methods can work when *all* of these conditions fail, and thus can handle more rewriting systems. We isolate the new *lift/project with termination* diagram as the key proof idea and show that previous rewriting-based methods (Plotkin's method based on *confluence* and *standardization* and Machkasova and Turbak's method based on distinct *lift* and *project* properties) implicitly use this diagram. Furthermore, our framework and proof methods help reduce the proof burden substantially by, e.g., supporting separate treatment of partitions of the rewrite steps, needing only *elementary diagrams* for rewrite step interactions, excluding many rewrite step interactions from consideration, needing weaker termination properties, and providing generic support for using developments in combination with any method.

1 Discussion

1.1 Background and Motivation

A programming language is defined as a set of programs and a way to evaluate (or "execute") the programs. It is increasingly popular to define evaluation via program rewriting [25, 9, 10, 11, 3, 19, 12, 26]. In this approach, evaluation rewrite rules are repeatedly applied at particular program positions which are typically specified using *evaluation contexts* [9].

Other kinds of program rewriting than evaluation are also desirable. Potential uses of rewriting-based program transformations include optimizing compilers, partial evaluators, and program simplifiers. These transformations may use the already existing evaluation rules in arbitrary contexts or use additional rewrite

* This work was partly supported by NSF grants CCR 9417382, CCR 9988529, and EIA 9806745, EPSRC grants GR/L 36963 and GR/R 41545/01, and Sun Microsystems equipment grant EDUD-7826-990410-US.

rules. Some transformations may involve global reasoning about the entire program, but many are local and a good match for rewriting-based techniques.

It is important to know when program transformations preserve a program's meaning as given by evaluation. There are many non-rewriting based approaches, such as denotational semantics (models), logical relations, applicative bisimulation and coinduction, etc., but they will not be discussed here because this paper focuses on rewriting-based techniques. Plotkin [25] first devised a rewriting-based method to prove meaning preservation for the call-by-name and call-by-value λ-calculus using *confluence* and *standardization*. At the same time, Plotkin proved that evaluation via rewriting was equivalent to evaluation via abstract machine. Subsequently, this approach has been applied to many systems, including systems with imperative features such as assignments and continuations (examples include [10, 11, 21, 3, 19, 12, 26, 17]).

Warning 1.1 (Not Quite Same as Observational Equivalence) What we call *meaning preservation* is related to *observational equivalence* (sometimes called *observational soundness* [18], *operational equivalence*, *consistency* [25], etc.), but is only the same for *contextually closed* rewriting systems. In this paper, terms have the same *meaning* iff evaluating them yields the same result (divergence or the same halted state). Terms t_1 and t_2 are *observationally equivalent*, written $t_1 \simeq t_2$, iff $C[t_1]$ and $C[t_2]$ have the same meaning for every context C where $C[t]$ places t in the hole of the context C. Proving a rewriting relation R to be meaning preserving implies that $R \subseteq \simeq$ only when R is contextually closed; see corollary 7.3 for an example. This paper presents an abstract (syntax-free) framework which does not have any features to represent notions like contexts, so we do not discuss observational equivalence except for specific examples. □

1.2 Summary of Contributions

The existing rewriting-based tools for proving meaning preservation are difficult to use and sometimes completely inapplicable. To address this problem, this paper presents an abstract framework and multiple diagram-based methods for proving meaning preservation. The new knowledge presented here improves on what is already known as follows.

1. Our methods can be used for rewriting systems that as a whole fail to have confluence, standardization, and/or *termination* or *boundedness* of *developments*. While some of our methods ask for confluence or standardization-like properties, they do so only for subsets of all rewrite steps.
2. We isolate the new *lift/project with termination* diagram (LPT in definition 4.1) and show that it is the key proof idea for previous methods for proving meaning preservation (Plotkin's method based on *confluence* and *standardization* and Machkasova and Turbak's method based on *lift* and *project* [17]). We show that the confluence & standardization method is incomparable in proving power with the lift & project method. We present new LPT-based methods that can handle systems that previous methods can not such as systems without standardization.

3. All of the proof methods dealt with in this paper (including the earlier methods of Plotkin and Machkasova & Turbak) are presented abstractly (free of syntax). Because our methods are abstract, there are no restrictions on the kinds of rewrite rules used. Rewrite rules may be non-left-linear, non-orthogonal (overlapping), non-first-order, etc. Also, our approach does not need a notion of closed *programs* as a subset of terms.
4. All our methods support partitioning the rewrite steps into subsets treated separately with different methods. These subsets need only be closed under (an informal and only intuitive notion of) "residuals with respect to evaluation steps". This partitioning also makes proving termination properties easier.
5. Our framework provides generic support for using developments (i.e., contracting only preexisting marked redexes) together with any method, so each method only needs to work for marked rewrite steps. This makes proving termination properties easier. No notion of *residuals* is needed, which is helpful for systems with highly overlapping rules where defining residuals is hard.
6. In addition to a number of high-level diagram-based methods for proving meaning preservation, we also present low-level methods that are easier to use for people who are not researchers in rewriting. We give as many as possible of the details needed for the non-specialist to use and adapt the proof methods. These low-level methods use simple termination properties and diagrams.
   (a) Termination properties are only needed for ordinary rewriting, not for rewriting of rewrite step sequences (perhaps this should be called meta-rewriting?) as in some abstract standardization methods [20]. The different termination properties that each method requires are simple and easy for the non-specialist to understand, ranging over boundedness (Bnd) and *(weak) normalization* (Nrm) and a bound on the number of evaluation steps in any rewrite sequence (BE in definition 5.1).
   (b) For analyzing rewrite step interactions, each method needs only the completion of *elementary diagrams*, i.e., diagrams where the only given edges are two adjacent single rewrite steps. In contrast, some abstract standardization methods require completing *cubes* [13, 20]. The method choice can depend on which elementary diagrams are completable. All of our methods exclude many rewrite step interactions from consideration.
7. To help rewriting researchers, as much as possible we identify intermediate diagrams to make it easier for new diagrams to be added as needed.
8. Our methods use only the simplest notion of standardization, that a rewrite sequence $t_1 \twoheadrightarrow t_2$ can be rearranged into a sequence $t_1 \xrightarrow{\mathbb{E}}\!\!\!\!\twoheadrightarrow t_3 \xrightarrow{\mathbb{N}}\!\!\!\!\twoheadrightarrow t_2$ where $\mathbb{E}$ and $\mathbb{N}$ indicate respectively evaluation and non-evaluation steps. Standardization in the literature is a rich and interesting notion [20], but other standardization definitions always imply our definition and the extra details are not useful here, so they are omitted.

Due to tight space limits, most proofs are omitted and also some proofs omit many details, but a long version of this paper with full proof details is available from the first author's home page.

1.3 Acknowledgements

An early informal presentation by Elena Machkasova on the *lift* and *project* diagrams gave significant inspiration, although this work then proceeded independently. Stefan Blom, Elena Machkasova, Vincent van Oostrom, and Lyn Turbak carefully read drafts of this paper and pointed out confusing terminology and errors. Zena Ariola is partly responsible for this work by convincing us to use rewrite rules for letrec that are difficult to prove correct.

2 Mathematical Definitions

Let $S \uplus S'$ denote $S \cup S'$ if $S \cap S' = \emptyset$ and otherwise be undefined. In a proof, "IH" means "by the induction hypothesis" and "w/o.l.o.g." means "without loss of generality".

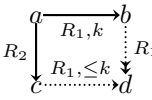
Let R range over binary relations. Let $\xrightarrow{R}$ and $\overrightarrow{R}$ be alternate notations for R which are usable infix, i.e., both $a \xrightarrow{R} b$ and $a \overrightarrow{R} b$ stand for $R(a,b)$ which in turn stands for $(a,b) \in R$.

Define the following operators on binary relations. Let $R;R'$ be the composition of R with R' (i.e., $\{\,(a,b) \,|\, \exists c.\ R(a,c) \text{ and } R'(c,b)\,\}$). Let $\xrightarrow{R,0} = R^0$ be equality at the type intended for R. Let $\xrightarrow{R,i+1} = R^{i+1} = (R^i;R)$ when $0 \leq i$. Let $\xrightarrow{R,\geq k} = R^{\geq k} = \bigcup_{i \geq k} R^i$. Let $\xrightarrow{R,\leq k} = R^{\leq k} = \bigcup_{i \leq k} R^i$. Let $\xrightarrow{R,j,\leq k} = R^j \cap R^{\leq k}$ (useful in diagrams when j is existentially quantified). Let $\xrightarrow[R]{\!\!\!\!\!\!\rightarrow} = R^* = R^{\geq 0}$ (the transitive, reflexive closure). Let $\xleftarrow{R} = R^{-1}$ be the inverse of R (i.e., $\{\,(a,b) \,|\, R(b,a)\,\}$). Let $\xleftarrow[R]{\!\!\!\!\!\!\leftarrow} = (R^{-1})^{\geq 0}$. Let $\xleftrightarrow{R} = (R \cup R^{-1})$ (the symmetric closure). When $R = R^{-1}$ (i.e., R is symmetric), let $\xleftrightarrow{R} = R$. Let $\xleftrightarrow{R,k} = (\xleftrightarrow{R})^k$. Let $\xleftrightarrow[R]{\!\!\!\!\!\!\leftrightarrow} = (\xleftrightarrow{R})^{\geq 0}$.

Let an entity a *be* a R-*normal form*, written $\mathsf{is\text{-}nf}(R,a)$, iff there does not exist some entity b such that $a \xrightarrow{R} b$. Let an entity a *have* a R-normal form, written $\mathsf{has\text{-}nf}(R,a)$, iff there exists some b such that $a \xrightarrow[R]{\!\!\!\!\!\!\rightarrow} b$ and $\mathsf{is\text{-}nf}(R,b)$. Let $\xrightarrow[R,\mathsf{nf}]{\!\!\!\!\!\!\rightarrow}$ be the relation such that $a \xrightarrow[R,\mathsf{nf}]{\!\!\!\!\!\!\rightarrow} b$ iff $a \xrightarrow[R]{\!\!\!\!\!\!\rightarrow} b$ and $\mathsf{is\text{-}nf}(R,b)$. A relation R is *bounded*, written $\mathsf{Bnd}(R)$, iff for every entity a there is some $k \geq 0$ such that there does not exist an entity b such that $R^k(a,b)$. A relation R is *terminating* (a.k.a. *strongly normalizing*), written $\mathsf{Trm}(R)$, iff there does not exist any total function f with as domain the natural numbers such that $R(f(i), f(i+1))$ for all $i \geq 0$. A relation R is *(weakly) normalizing*, written $\mathsf{Nrm}(R)$, iff for every entity a there is some entity b such that $a \xrightarrow[R,\mathsf{nf}]{\!\!\!\!\!\!\rightarrow} b$. Note that $\mathsf{Bnd}(R) \Rightarrow \mathsf{Trm}(R) \Rightarrow \mathsf{Nrm}(R)$.

Diagrams make statements about relations where solid and dotted edges indicate quantification. Metavariables already mentioned outside the diagram are unquantified. Other metavariables (e.g., for node names or used in edge labels) are universally quantified if attached to a solid edge and existentially quantified if attached only to dotted edges. As an example, in a context where

R_1 and R_2 have already been given, the following equivalence holds:



$$\iff \forall a, b, c, k.\ (a \xrightarrow{R_1,k} b \wedge a \xrightarrow{R_2} c) \Rightarrow \exists d.\ c \xrightarrow{R_1,\leq k} d \wedge b \xrightarrow{R_1} d$$

In proofs, the reason for each diagram polygon will usually be written inside it.

3 Abstract Evaluation Systems

An *abstract evaluation system* (AES) is a tuple

$$(\mathbb{T}, \mathbb{S}, \mathbb{R}, \mathsf{endpoints}, \mathbb{E}, \mathsf{result})$$

satisfying the conditions given below by axioms 3.3 and 3.4 and the immediately following conditions. The carriers of an AES are the sets $\mathbb{T}$, $\mathbb{S}$, and $\mathbb{R}$. The function $\mathsf{endpoints}$ maps $\mathbb{S}$ to $\mathbb{T} \times \mathbb{T}$. The set $\mathbb{E}$ is a subset of $\mathbb{S}$. The function result maps $\mathbb{T}$ to $\mathbb{R}$. Let t range over $\mathbb{T}$, let s range over $\mathbb{S}$, let r range over $\mathbb{R}$, and let $\mathcal{S}$ range over subsets of $\mathbb{S}$.

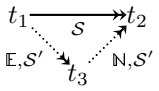
The intended meaning is as follows. $\mathbb{T}$ should be a set of terms. $\mathbb{S}$ should be a set of rewrite steps. $\mathbb{R}$ should be a set of evaluation results which by axiom 3.4(1) will most likely contain the symbol $\mathsf{diverges}$ and one or more other members, typically symbols such as halt, error, etc. The halt case might be subdivided into possible constant values of final results. If $\mathsf{endpoints}(s) = (t_1, t_2)$, this should mean that step s rewrites term t_1 to term t_2. The members of $\mathbb{E}$ are the rewrite steps used for evaluation. Let $\mathbb{N} = \mathbb{S} \setminus \mathbb{E}$ (where "$\mathbb{N}$" stands for "non-evaluation"). If $\mathsf{result}(t) = r$, this should mean that r is the observable result of evaluating term t, where $\mathsf{diverges}$ is reserved by axiom 3.4(1) for non-halting evaluations.

Convention 3.1 *In this paper, wherever no specific AES is being considered, statements are about every possible AES.* □

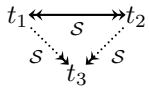
Let rewriting notation be defined as follows. Given a rewrite step set $\mathcal{S}$, let $\llcorner\mathcal{S}\lrcorner$ be the binary relation $\{\, (t, t') \mid \exists s \in \mathcal{S}.\ \mathsf{endpoints}(s) = (t, t') \,\}$. Thus, $t \xrightarrow{\llcorner\mathcal{S}\lrcorner} t'$ iff there exists $s \in \mathcal{S}$ such that $\mathsf{endpoints}(s) = (t, t')$. When a rewrite step set $\mathcal{S}$ is used in a context *requiring* a binary relation on $\mathbb{T}$, then let $\mathcal{S}$ implicitly stand for $\llcorner\mathcal{S}\lrcorner$. Thus, as examples, $t \xrightarrow{\mathcal{S}} t'$ stands for $t \xrightarrow{\llcorner\mathcal{S}\lrcorner} t'$ and an $\mathcal{S}$-normal form is simply a $\llcorner\mathcal{S}\lrcorner$-normal form. When used in a position *requiring* a subset of $\mathbb{S}$ or a binary relation on $\mathbb{T}$, let s stand for $\{s\}$ and let $\mathcal{S}, \mathcal{S}'$ stand for $\mathcal{S} \cap \mathcal{S}'$. Thus, as an example, $t \xrightarrow{\mathcal{S},s} t'$ stands for $t \xrightarrow{\llcorner\mathcal{S}\cap\{s\}\lrcorner} t'$. When a binary relation on $\mathbb{T}$ is *required* and none is supplied, then let the relation $\llcorner\mathbb{S}\lrcorner$ be implicitly supplied. Thus, as examples, $t \longrightarrow t'$ stands for $t \xrightarrow{\llcorner\mathbb{S}\lrcorner} t'$ and $t \xrightarrow{k} t'$ stands for $t \xrightarrow{\llcorner\mathbb{S}\lrcorner,k} t'$.

Definition 3.2 (Rewrite Step Set Properties). *Define the following rewrite step sets and properties of rewrite step sets:*

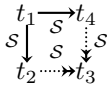
Standardization:

$\mathsf{Std}(\mathcal{S},\mathcal{S}') \iff$ 

Confluence:

$\mathsf{Conf}(\mathcal{S}) \iff$ 

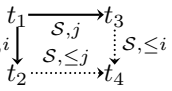
Local Confluence:

$\mathsf{LConf}(\mathcal{S}) \iff$ 

Meaning Preservation:

$s \in \mathsf{MP} \iff$ 

Subcommutativity:

$\mathsf{SubComm}(\mathcal{S},i,j) \iff$ 

Let $\mathsf{Std}(\mathcal{S})$ *abbreviate* $\mathsf{Std}(\mathcal{S},\mathbb{S})$. *Let* $\mathsf{SubComm}(\mathcal{S})$ *abbreviate* $\mathsf{SubComm}(\mathcal{S},1,1)$. *Traditionally, only* $\mathsf{Std}(\mathbb{S}) = \mathsf{Std}(\mathbb{S},\mathbb{S})$ *is considered. The simple definition of* MP *is reasonable because axiom 3.4(1) (given below) means* MP *implies preservation of the existence of* $\mathbb{E}$*-normal forms. See also warning 1.1 and convention 3.1 and do not confuse* MP *with observational equivalence.* □

Axiom 3.3 (Subcommutativity of Evaluation) $\mathsf{SubComm}(\mathbb{E})$. □

Non-deterministic evaluation is useful for rewriting systems with non-deterministic syntax, e.g., the system of [17] where the top syntax level is a set with unordered components. Often, it will be simpler to make evaluation deterministic so that $t_2 \xleftarrow{\mathbb{E},s_1} t_1 \xrightarrow{\mathbb{E},s_2} t_3$ implies that $t_2 = t_3$ or even that $s_1 = s_2$.

Axiom 3.3 does not ensure that any strategy for $\xrightarrow{\mathbb{E}}$ will find $\mathbb{E}$-normal forms when they exist. Strengthening axiom 3.3 so that the bottom and right diagram edges have the same length would ensure this, but is not needed otherwise.

Axiom 3.4 (Evaluation Sanity)

1. *"*$\mathsf{diverges}$*" **Means Evaluation Diverges**:*
   $\mathsf{result}(t) = \mathsf{diverges} \Leftrightarrow \neg\mathsf{has\text{-}nf}(\mathbb{E},t)$.
2. ***Evaluation Steps Preserve Meaning**:*
   $\mathbb{E} \subseteq \mathsf{MP}$.
3. ***Non-Evaluation Steps Preserve Evaluation Steps**:*

   Consequently, if $t_1 \xrightarrow{\mathbb{N}} t_2$, *then* $\mathsf{is\text{-}nf}(\mathbb{E},t_1) \Leftrightarrow \mathsf{is\text{-}nf}(\mathbb{E},t_2)$.
4. ***Non-Evaluation Steps on** $\mathbb{E}$**-Normal Forms Preserve Meaning**:*
   If $t \xrightarrow{\mathbb{N}} t'$ *and* $\mathsf{is\text{-}nf}(\mathbb{E},t)$, *then* $\mathsf{result}(t) = \mathsf{result}(t')$. □

When defining an AES for a rewriting system, it is trivial to satisfy axioms 3.4(1) and 3.4(2) by using an auxiliary function result' which maps $\{\, t \,|\, \mathsf{is\text{-}nf}(\mathbb{E},t) \,\}$ to $\mathbb{R} \setminus \{\mathsf{diverges}\}$ and defining result as follows:

$$\mathsf{result}(t) = \begin{cases} \mathsf{diverges} & \text{if } \neg\mathsf{has\text{-}nf}(\mathbb{E},t), \\ \mathsf{result}'(t') & \text{if } t \xrightarrow{\mathbb{E},\mathsf{nf}} t'. \end{cases}$$

Indeed, the model of how evaluation should be computed expects to work this way. When $\neg\mathsf{is\text{-}nf}(\mathbb{E}, t)$, it is expected that computing $\mathsf{result}(t)$ involves first finding t' such that $t \xrightarrow{\mathbb{E},\mathsf{nf}}\!\!\!\!\rightarrow\, t'$, then computing $\mathsf{result}(t')$, and otherwise diverging if no such t' exists. Thus, the value of $\mathsf{result}(t)$ is unimportant if $\neg\mathsf{has\text{-}nf}(\mathbb{E}, t)$. Reserving the value $\mathsf{diverges}$ for this case simplifies things.

Satisfying axioms 3.4(3) and 3.4(4) requires more care in the design of the rewriting system and the AES, but is not hard. Anyway, axiom 3.4(3) is a consequence of the properties WL1 and WP1 or WLP1 from definition 5.1 which typically must also be proven. At first glance, the reader might think that axiom 3.4(3) is simpler than what is needed because in its diagram no relationship is required between t_3 and t_4; however this issue is handled by the LPT diagram from definition 4.1 and in any case a relationship between t_3 and t_4 is only needed when $\mathsf{has\text{-}nf}(\mathbb{E}, t_1)$. The condition of axiom 3.4(3) appears in other abstract frameworks as early as [13] and appears in non-abstract form in [25]. The first explicit statement of the condition of axiom 3.4(4) that we are aware of appears in [16], although the condition is partially present in [3].

Lemma 3.5 (Non-Evaluation Steps on Eval-Normal Forms). *If* $t_1 \xleftarrow{\mathbb{N}}\!\!\!\!\rightarrow\, t_2$ *and* $\mathsf{is\text{-}nf}(\mathbb{E}, t_1)$*, then* $t_1 \xleftarrow{\mathsf{MP}}\!\!\!\!\rightarrow\, t_2$. □

4 Lift/Project Diagrams for Meaning Preservation

This section presents properties of rewrite step sets in definition 4.1 and shows how to use them to prove *meaning preservation*, the important connection between arbitrary-strategy rewriting and evaluation. When evaluation is defined by a subset of the rewrite steps (specified in an AES by the set $\mathbb{E}$), it is necessary to show that arbitrary rewriting preserves the evaluation result in order to have confidence that the non-evaluation rewrite steps are at all meaningful. Traditionally, this has been done by proving confluence (Conf) and standardization (Std), the preconditions of Plotkin's approach [25] (presented in lemma 4.5(1,2)).

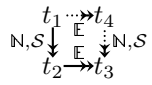
Needing confluence and standardization is a big weakness, as shown by the non-confluent system in [17] and the $\lambda^{:=,\mathsf{letrec}}$ calculus we mention in section 9 which has neither confluence nor standardization. In contrast, our new method in theorem 4.3 needs only the *lift/project with termination* (LPT) property. By lemma 4.2, LPT can be obtained from the lift (Lift) and project (Proj) properties. Because lift and project do not imply confluence (lemma 4.5(4)), theorem 4.3 does not need confluence. Furthermore, because LPT implies neither lift nor project (lemma 4.2(8,10)) and lift is equivalent to standardization (lemma 4.4(1)), theorem 4.3 does not need standardization when lift is not used.

Theorem 4.3 differs from earlier work of Machkasova and Turbak [17] in several important ways. First, it is abstract (syntax-free). Second, it provides explicit support for separately proving meaning preservation for different subsets of the non-evaluation rewrite steps. This vastly simplifies auxiliary termination proofs (e.g., for properties Bnd or BE as used in definition 5.1) and is vital when a single method fails to cover all $\mathbb{N}$ steps (e.g., section 9). Third, it needs

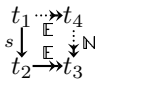
only the weaker LPT property rather than lift and project. This is vital because lift is equivalent to standardization so the Machkasova/Turbak method fails for systems without standardization (e.g., section 9).

Definition 4.1 (Lift, Project, and Related Properties). *Define the following rewrite step sets and properties of rewrite step sets:*

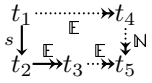
Strong Lift:

$\mathsf{SLift}(\mathcal{S}) \iff$ 

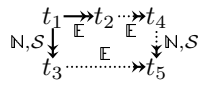
Lift:

$s \in \mathsf{Lift} \iff$ 

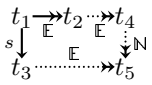
Lift′:

$s \in \mathsf{Lift}' \iff$ 

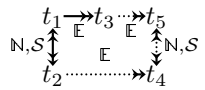
Strong Project:

$\mathsf{SProj}(\mathcal{S}) \iff$ 

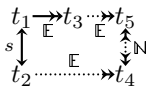
Project:

$s \in \mathsf{Proj} \iff$ 

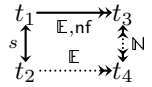
Strong Lift/Project:

$\mathsf{SLP}(\mathcal{S}) \iff$ 

Lift/Project:

$s \in \mathsf{LP} \iff$ 

Lift/Project when Terminating:

$s \in \mathsf{LPT} \iff$ 

□

The Lift and Proj properties given here match the properties by the names "Lift" and "Project" in [17], except that there both properties are defined on the entire rewriting system rather than on individual rewrite steps and both properties specify the step on the left diagram edge to be a $\mathbb{N}$ step (the latter difference being inessential). Only the weaker Lift' which is symmetrical with Proj is actually needed together with Proj to obtain LPT (lemma 4.2(7,9)). However, Lift' can not replace Lift in the statement of lemma 4.4(1).

Lemma 4.2 (Relationships between Lift and Project Properties).

1. $\mathbb{E} \subseteq \mathsf{Lift} \cap \mathsf{Proj}$.
2. *If* $\mathsf{SLift}(\mathcal{S})$, *then* $\mathcal{S} \subseteq \mathsf{Lift}$.
3. *If* $\mathsf{SProj}(\mathcal{S})$, *then* $\mathcal{S} \subseteq \mathsf{Proj}$.
4. *If* $\mathsf{SLP}(\mathcal{S})$, *then* $\mathcal{S} \subseteq \mathsf{LP}$.
5. $\mathsf{Lift} \subseteq \mathsf{Lift}'$.
6. $\mathsf{Lift}' \subseteq \mathsf{Lift}$ *need not be true.*
7. $\mathsf{Lift}' \cap \mathsf{Proj} \subseteq \mathsf{LP}$.
8. *None of* $\mathsf{LP} \subseteq \mathsf{Lift}' \cap \mathsf{Proj}$, $\mathsf{LP} \subseteq \mathsf{Lift}'$, *and* $\mathsf{LP} \subseteq \mathsf{Proj}$ *need to be true.*
9. $\mathsf{LP} \subseteq \mathsf{LPT}$.
10. $\mathsf{LPT} \subseteq \mathsf{LP}$ *need not be true.*

□

Theorem 4.3 (Relationships between Lift, Project, and Meaning Preservation).

1. $\mathsf{LPT} \subseteq \mathsf{MP}$.
2. $\mathsf{MP} \subseteq \mathsf{LPT}$ *need not be true.* □

Proof.

1. Suppose $s \in \mathsf{LPT}$. Let $t_1 \xrightarrow{s} t_2$. Suppose neither $\mathsf{has\text{-}nf}(\mathbb{E}, t_1)$ nor $\mathsf{has\text{-}nf}(\mathbb{E}, t_2)$. By axiom 3.4(1), it holds that $\mathsf{result}(t_1) = \mathsf{diverges} = \mathsf{result}(t_2)$, so $s \in \mathsf{MP}$. Suppose instead that either $\mathsf{has\text{-}nf}(\mathbb{E}, t_1)$ or $\mathsf{has\text{-}nf}(\mathbb{E}, t_2)$. Suppose $\mathsf{has\text{-}nf}(\mathbb{E}, t_1)$ (w/o.l.o.g. because only $t_1 \xleftrightarrow{s} t_2$ is used). Then $t_1 \xrightarrow{\mathbb{E},\mathsf{nf}}\!\!\!\!\rightarrow t_3$ for some t_3. By $s \in \mathsf{LPT}$, it holds that $t_3 \xleftarrow{\mathbb{N}}\!\!\!\!\rightarrow t_4 \xleftarrow{\mathbb{E}} t_2$ for some t_4. Because $\mathsf{is\text{-}nf}(\mathbb{E}, t_3)$, by lemma 3.5 it holds that $t_3 \xleftarrow{\mathsf{MP}}\!\!\!\!\rightarrow t_4$. By axiom 3.4(2) and induction on the lengths of rewrite sequences, it holds that $t_1 \xrightarrow{\mathsf{MP}}\!\!\!\!\rightarrow t_3$ and $t_2 \xrightarrow{\mathsf{MP}}\!\!\!\!\rightarrow t_4$. Thus, $t_1 \xleftarrow{\mathsf{MP}}\!\!\!\!\rightarrow t_2$. Thus, $s \in \mathsf{MP}$.
2. Consider this 4-term 3-step AES where all results are the same:

$$\begin{array}{ccc} t_1 & \xrightarrow{\mathbb{E},s_1} & t_2 \\ {\scriptstyle \mathbb{N},s_2}\downarrow & & \\ t_3 & \xrightarrow{\mathbb{E},s_3} & t_4 \end{array}$$

   Then $\mathsf{MP} = \mathbb{S}$, but $\mathsf{MP} \setminus \mathsf{LPT} = \{s_2\}$. □

4.1 Comparison with Traditional Approach

This subsection compares the lift & project method of Machkasova and Turbak and our LPT method with the traditional confluence & standardization method. Plotkin's traditional approach [25] was separated out and presented abstractly by Machkasova [16] in a form similar to the combination of the proofs of lemma 4.5(1), lemma 4.2(9), and theorem 4.3(1). We have reformulated the argument for the AES framework and modified it to work on subsets of $\mathbb{S}$. Furthermore, we have factored the argument to show it goes through LP (lemma 4.5(1)) and LPT before reaching MP. Thus, it appears that the main previously known rewriting-based methods of showing meaning preservation implicitly use the LPT diagram. Interestingly, in lemma 4.5(3,4) it is shown that the confluence & standardization method and the lift & project method are incomparable in their power; each can address problems that the other can not. Section 5 will develop another method ($\mathsf{WB}\backslash\mathsf{Std}$ in definition 5.1) of proving LPT which can address yet more problems, because it does not require standardization.

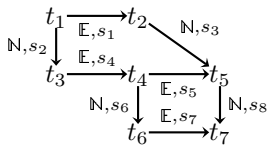
The following equivalence of Lift and standardization in lemma 4.4(1) appears in [16], although here it has been parameterized on rewrite step sets.

Lemma 4.4 (Lift Equivalent to Standardization).

1. $\mathcal{S} \subseteq \mathsf{Lift}$ *iff* $\mathsf{Std}(\mathcal{S} \cup \mathbb{E})$. *(Consequently,* $\mathsf{Lift} = \mathbb{S}$ *iff* $\mathsf{Std}(\mathbb{S})$*.)*
2. *The above statement need not be true with* Lift *replaced by* Lift'. □

Proof.

1. $\mathsf{Std}(\mathcal{S} \cup \mathbb{E}) \Rightarrow \mathcal{S} \subseteq \mathsf{Lift}$ is immediate. $\mathcal{S} \subseteq \mathsf{Lift} \Rightarrow \mathsf{Std}(\mathcal{S} \cup \mathbb{E})$ is proven by induction on the length of rewrite sequences.
2. Consider this 7-term 8-step AES where all results are the same:



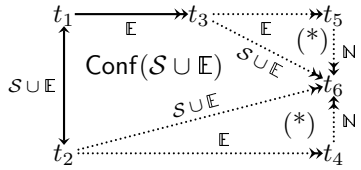
   Note that $\mathsf{Lift}' = \mathbb{S}$, but $\mathsf{Lift}' \setminus \mathsf{Lift} = \{s_2\}$. The desired $\mathsf{Std}(\mathbb{S})$ is false, because $t_1 \longrightarrow\!\!\!\!\!\rightarrow t_6$ but there is no t such that $t_1 \xrightarrow{\mathbb{E}}\!\!\!\!\!\rightarrow t \xrightarrow{\mathbb{N}}\!\!\!\!\!\rightarrow t_6$. □

Lemma 4.5 (Relationships between Confluence + Standardization and Lift + Project).

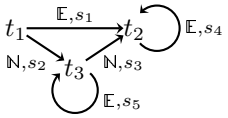
1. *If* $\mathsf{Conf}(\mathcal{S} \cup \mathbb{E})$ *and* $\mathsf{Std}(\mathcal{S} \cup \mathbb{E})$*, then* $\mathcal{S} \subseteq \mathsf{LP}$.
2. *Consequently,* $\mathsf{Conf}(\mathcal{S} \cup \mathbb{E})$ *and* $\mathcal{S} \subseteq \mathsf{Lift}$ *imply* $\mathcal{S} \subseteq \mathsf{MP}$.
3. *If* $\mathsf{Conf}(\mathcal{S} \cup \mathbb{E})$ *and* $\mathsf{Std}(\mathcal{S} \cup \mathbb{E})$*, then* $\mathcal{S} \subseteq \mathsf{Proj}$ *need not be true.*
4. $\mathsf{Conf}(\mathsf{Lift} \cap \mathsf{Proj})$ *need not be true.* □

Proof.

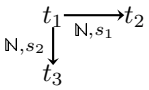
1. Suppose that $\mathsf{Conf}(\mathcal{S} \cup \mathbb{E})$ and (*) $\mathsf{Std}(\mathcal{S} \cup \mathbb{E})$ hold. Using the reason (*) as indicated, the following diagram proves $\mathcal{S} \cup \mathbb{E} \subseteq \mathsf{LP}$ and thus $\mathcal{S} \subseteq \mathsf{LP}$:



2. By lemmas 4.2(9), 4.4, and 4.5(1) and theorem 4.3(1).
3. Consider this 3-term 5-step AES:



   Then $\mathsf{Conf}(\mathbb{N} \cup \mathbb{E})$ and $\mathsf{Std}(\mathbb{N} \cup \mathbb{E})$, but $\mathbb{N} \setminus \mathsf{Proj} = \{s_2\}$.
4. Consider this 3-term 2-step AES where all results are the same:



   Then $\mathsf{Lift} = \mathsf{Proj} = \mathbb{S}$, but $\neg\mathsf{Conf}(\mathbb{S})$. □

5 Elementary Diagrams for Strong Lift/Project

According to section 4, one can prove rewrite step sets to have the LPT property in order to prove meaning preservation. Furthermore, LPT can be obtained via stronger properties such as the lift and project properties. However, proving these properties can be very difficult.

To help, this section provides abstract methods for proving strong lift, strong project, and/or strong lift/project for particular rewrite step sets. Definition 5.1 defines that a rewrite step set is *well behaved* when it satisfies either the WB+Std or WB\Std properties. In turn, each of these are conjunctions of a small number of specific properties, one termination property and some *elementary diagrams*, i.e., diagrams where the given edges are two adjacent single rewrite steps. The WB+Std and WB\Std properties are about rewrite step *sets* rather than individual steps because it is necessary to simultaneously treat all the steps in a set that is closed under (an informal and only intuitive notion of) "residuals with respect to evaluation steps". This section's main result (theorem 5.4) is that a well behaved rewrite step set $\mathcal{S}$ has either the strong lift and strong project properties or the strong lift/project property.

Each of WB+Std and WB\Std has particular advantages. The termination property of WB+Std requires only a bound on the number of $\mathbb{E}$ steps in a rewrite sequence (BE), not full termination. When used together with the methods of section 6, this is significantly weaker than the *finite developments* property needed by some other proof methods, because it allows infinite developments (and there is no requirement that coinitial developments can be completed to be cofinal). In contrast, WB\Std requires a stronger termination property, but replaces the WL1 and WP1 elementary diagrams with the weaker diagram WLP1. The big advantage of WLP1 is that it does not require standardization. Although WB\Std($\mathcal{S}$) requires local confluence for $\mathcal{S}$, in fact it is sufficient to have only confluence (lemmas 5.2(3) and 5.3(3)) and the local confluence requirement is only there so that the preconditions of WB\Std($\mathcal{S}$) are elementary diagrams.

Definition 5.1 (Well Behaved Rewrite Step Sets). *Let* $\mathsf{N}^*\mathsf{E}\mathsf{N}^*(\mathcal{S})$ *be the relation* $\xrightarrow{\mathbb{N},\mathcal{S}}\!\!\!\!\rightarrow;\xrightarrow{\mathbb{E},\mathcal{S}};\xrightarrow{\mathbb{N},\mathcal{S}}\!\!\!\!\rightarrow$. *Define the following rewrite step set properties:*

Bounded $\mathbb{E}$*-Steps:*

$$\mathsf{BE}(\mathcal{S}) \iff \mathsf{Bnd}(\mathsf{N}^*\mathsf{E}\mathsf{N}^*(\mathcal{S}))$$

$\mathbb{N}$*-Steps Do Not Create* $\mathbb{E}$*-Steps:*

$$\mathsf{NE}(\mathcal{S}) \iff \begin{array}{ccc} t_1 & \overset{\mathbb{E},\mathcal{S}}{\cdots\!\!\rightarrow} & t_4 \\ {\scriptstyle \mathbb{N},\mathcal{S}}\downarrow & & \downarrow{\scriptstyle \mathbb{E},\mathcal{S}} \\ t_2 & \xrightarrow{\quad} & t_3 \end{array}$$

Weak Lift 1-Step:

$$\mathsf{WL1}(\mathcal{S},\mathcal{S}') \iff \begin{array}{ccc} t_1 & \overset{\mathbb{E},\mathcal{S}'}{\cdots\!\!\rightarrow} & t_4 \\ {\scriptstyle \mathbb{N},\mathcal{S}}\downarrow & & \vdots{\scriptstyle \mathcal{S}} \\ t_2 & \xrightarrow[\mathbb{E},\mathcal{S}']{} & t_3 \end{array}$$

Weak Project 1-Step:

$$\mathsf{WP1}(\mathcal{S}) \iff \begin{array}{ccc} t_1 & \xrightarrow{\mathbb{E}} & t_2 \\ {\scriptstyle \mathbb{N},\mathcal{S}}\downarrow & & \vdots{\scriptstyle \mathcal{S}} \\ t_3 & \overset{\mathbb{E}}{\cdots\!\!\rightarrow} & t_4 \end{array}$$

Weak Lift/Project 1-Step:

$$\mathsf{WLP1}(\mathcal{S}) \iff \begin{array}{ccc} t_1 & \xrightarrow{\mathbb{E}} & t_4 \\ {\scriptstyle \mathbb{N},\mathcal{S}}\updownarrow & & \vdots{\scriptstyle \mathcal{S}} \\ t_2 & \overset{\mathbb{E}}{\cdots\!\!\rightarrow} & t_3 \end{array}$$

Standardization to Normal Form:

$$\mathsf{Std\text{-}nf}(\mathcal{S}) \iff \begin{array}{ccccc} t_1 & & \xrightarrow[\mathcal{S},\mathsf{nf}]{} & & t_2 \\ & {\scriptstyle \mathbb{E},\mathcal{S},\mathsf{nf}}\searrow & & \nearrow{\scriptstyle \mathbb{N},\mathcal{S},\mathsf{nf}} & \\ & & t_3 & & \end{array}$$

Well Behaved with Standardization:

$$\mathsf{WB{+}Std}(\mathcal{S}) \iff \mathsf{BE}(\mathcal{S}) \wedge \mathsf{WL1}(\mathcal{S},\mathcal{S}) \wedge \mathsf{WL1}(\mathcal{S},\mathbb{S}) \wedge \mathsf{WP1}(\mathcal{S})$$

Well Behaved without Standardization:

$$\mathsf{WB}\backslash\mathsf{Std}(\mathcal{S}) \iff \mathsf{Trm}(\mathcal{S}) \wedge \mathsf{LConf}(\mathcal{S}) \wedge \mathsf{NE}(\mathcal{S}) \wedge \mathsf{WLP1}(\mathcal{S})$$ □

Lemma 5.2 (Confluence and Standardization-Like Properties).

1. *If* $\mathsf{BE}(\mathcal{S})$ *and* $\mathsf{WL1}(\mathcal{S},\mathcal{S})$*, then* $\mathsf{Std}(\mathcal{S},\mathcal{S})$*.*
2. *If* $\mathsf{LConf}(\mathcal{S})$ *and* $\mathsf{Trm}(\mathcal{S})$*, then* $\mathsf{Conf}(\mathcal{S})$ *(Newman's Lemma).*
3. *If* $\mathsf{Conf}(\mathcal{S})$*,* $\mathsf{Trm}(\mathcal{S})$*, and* $\mathsf{NE}(\mathcal{S})$*, then* $\mathsf{Std\text{-}nf}(\mathcal{S})$*.* □

Lemma 5.3 (Strong Lift and Project Properties).

1. *If* $\mathsf{WL1}(\mathcal{S},\mathbb{S})$ *and* $\mathsf{Std}(\mathcal{S},\mathcal{S})$*, then* $\mathsf{SLift}(\mathcal{S})$*.*
2. *If* $\mathsf{WP1}(\mathcal{S})$ *and* $\mathsf{Std}(\mathcal{S},\mathcal{S})$*, then* $\mathsf{SProj}(\mathcal{S})$*.*
3. *If* $\mathsf{Conf}(\mathcal{S})$*,* $\mathsf{Trm}(\mathcal{S})$*,* $\mathsf{Std\text{-}nf}(\mathcal{S})$*, and* $\mathsf{WLP1}(\mathcal{S})$*, then* $\mathsf{SLP}(\mathcal{S})$*.* □

Theorem 5.4 (Well Behaved Rewrite Step Sets).

1. *If* $\mathsf{WB{+}Std}(\mathcal{S})$*, then* $\mathsf{SLift}(\mathcal{S})$ *and* $\mathsf{SProj}(\mathcal{S})$*.*
2. *If* $\mathsf{WB}\backslash\mathsf{Std}(\mathcal{S})$*, then* $\mathsf{SLP}(\mathcal{S})$*.* □

Proof. By definition 5.1 and lemmas 5.2 and 5.3. □

6 Marked Rewriting and Developments

Sometimes, a desired termination property (e.g., BE from definition 5.1, Bnd, Trm, or Nrm) fails for a step set $\mathcal{S}$ generated by some rewrite rule(s), but holds for $\mathcal{S} \cap \mathbb{M}$ where $\mathbb{M}$ is a set of *marked* steps. The marks typically force termination by forbidding contracting unmarked redexes and ensuring that "created" redexes are unmarked. To use this method, the desired rewriting system is embedded in a larger marked system with additional marked terms and rewrite steps, so proving the larger system correct also proves the desired system correct.

This section defines conditions on marking and theorem 6.4 proves that when these conditions hold, proving LPT for $\mathcal{S} \cap \mathbb{M}$ (i.e., the marked fragment of the larger marked system) is sufficient to prove LPT for $\mathcal{S}$ (i.e., both the marked and unmarked steps in the larger system). Thus, when any of this paper's methods for proving meaning preservation work for $\mathcal{S} \cap \mathbb{M}$, the methods also work for $\mathcal{S}$. It is worth observing that the style of proof of theorem 6.4 can be repeated for many properties other than LPT, e.g., for Lift (and therefore for standardization).

This section's methods are related to *developments*. A development is a rewrite step sequence starting from a term t where each step contracts a redex which represents work that was already in t and "created" redexes are not contracted. Usually, the notions of "work already present" and "created" are defined using *residuals* of redexes across rewrite steps, sometimes defining residuals using marks. This section's methods do not need any notion of residual.

This is important because there do not seem to be good ways to define residuals for many rewriting systems, e.g., those with highly overlapping rewrite rules.

A *mark structure* for an AES is a tuple

$$(\mathsf{Marks}, \mathsf{markOf}, \mathsf{noMark}, \mathsf{rename})$$

satisfying axiom 6.1 below and the following conditions. The set Marks is non-empty and does not contain $\star$. The function markOf maps $\mathbb{S}$ to $\mathsf{Marks} \cup \{\star\}$. The mark noMark is a member of Marks. The function rename is of type $(\mathsf{Marks} \times \mathsf{Marks}) \to \mathbb{T} \to \mathbb{T}$. Let m range over Marks. Let $\mathbb{M} = \{\, s \in \mathbb{S} \mid \mathsf{markOf}(s) \neq \mathsf{noMark} \,\}$. Let the statement $\mathsf{markOccurs}(m, t)$ hold iff there exist s and t' such that $t \xrightarrow{s} t'$ and $\mathsf{markOf}(s) = m$.

The intended meaning is as follows. The set Marks should contain marks used to track redexes. Each rewrite step s should be marked by the mark $\mathsf{markOf}(s)$. The special mark noMark means "no mark at all". The symbol $\star$ means "can be considered to be any mark because we do not track this kind of rewrite step with marks"; this is a convenience for systems where only some steps have marked versions. The operation $\mathsf{rename}(m_1, m_2)(t)$ should produce a new term t' resulting from renaming all occurrences of the mark m_1 in t to m_2.

Axiom 6.1 (Marking Sanity)

1. ***Marked Erasure:***
   For $\mathcal{S} \in \{\mathbb{E}, \mathbb{N}\}$,
   $$\begin{array}{ccc} t_1 & \xrightarrow{\mathcal{S}} & t_2 \\ {\scriptstyle \mathsf{rename}(m,m')}\downarrow & & \downarrow{\scriptstyle \mathsf{rename}(m,m')} \\ t_3 & \xrightarrow{\mathcal{S}} & t_4 \end{array}$$
2. $\mathbb{E}$ ***Marked Unerasure:***
   $$\begin{array}{ccc} t_1 & \xrightarrow{\mathbb{E}} & t_2 \\ {\scriptstyle \mathsf{rename}(m,m')}\uparrow & & \vdots{\scriptstyle \mathsf{rename}(m,m')} \\ t_3 & \xrightarrow{\mathbb{E}} & t_4 \end{array}$$
3. ***Erasing Nonexistent Mark:***
   If $\neg\mathsf{markOccurs}(m, t)$*, then* $\mathsf{rename}(m, m')(t) = t$.
4. ***Marks Not Introduced by Rewriting:***
   If $\neg\mathsf{markOccurs}(m, t)$*,* $m \neq \mathsf{noMark}$*, and* $t \longrightarrow t'$*, then* $\neg\mathsf{markOccurs}(m, t')$.
5. ***Fresh Marks:***
   For any term t*, there exists a mark* $m \neq \mathsf{noMark}$ *such that* $\neg\mathsf{markOccurs}(m, t)$.

□

Convention 6.2 *In this paper, wherever no specific mark structure is being considered, statements are about every possible mark structure.* □

Definition 6.3 (Rewrite Step Set Property for Marks).

$\mathbb{N}$ *Step Can Be Marked:*

$$\mathsf{NM}(\mathcal{S}) \iff \left(\begin{pmatrix} m \neq \mathsf{noMark} \\ \wedge\, \neg\mathsf{markOccurs}(m, t_1) \end{pmatrix} \Rightarrow \begin{array}{ccc} t_1 & \xrightarrow{\mathbb{N},\mathcal{S}} & t_2 \\ {\scriptstyle \mathsf{rename}(m,m')}\searrow & & \nearrow{\scriptstyle \mathbb{N},\mathbb{M},\mathcal{S}} \\ & t_3 & \end{array} \right)$$

□

Theorem 6.4 (Lift/Project when Terminating via Marks). *If* $\mathcal{S} \cap \mathbb{M} \subseteq \mathsf{LPT}$ *and* $\mathsf{NM}(\mathcal{S})$*, then* $\mathcal{S} \subseteq \mathsf{LPT}$. □

Proof. Using axiom 6.1, lemma 4.2(1), and definitions 6.3 and 4.1. □

7 Example: The Call-by-Name λ-Calculus

This section gives an example of the use of our AES framework and our diagram-based methods for proving meaning preservation. The AES and a mark structure will be defined and then the top-level proof strategy will be presented.

We choose the call-by-name λ-calculus with left-most outermost evaluation to weak head normal forms because it is a small system, needs the mark structure features of section 6, will already be familiar to most readers, and is one of the two systems treated by Plotkin's seminal paper [25]. This system has both confluence and standardization. To illustrate the extra power of our proof methods, we would have preferred to present an example system which does not have these properties, but unfortunately our smallest worked-out example takes many pages in LNCS format to even define and does not need the features of section 6.

Define the AES for the call-by-name λ-calculus as follows. First, define the AES carrier sets $\mathbb{T}$, $\mathbb{S}$, and $\mathbb{R}$ as well as the evaluation step subset $\mathbb{E}$.

$$
\begin{array}{lll}
x, y, z \in \mathsf{Variable} & & \\
\overline{t} \in \mathsf{Context} & ::= \Box \mid x \mid (\lambda x\, \overline{t}) \mid (\overline{t_1}\ \overline{t_2}) \mid (\mathsf{let}^n x = \overline{t_2}\ \mathsf{in}\ \overline{t_1}) & (n \geq 1) \\
t \in \mathbb{T} & = \{\, \overline{t} \mid \overline{t} \text{ has no hole } \Box \,\} & \\
\overline{E} \in \mathsf{EvalContext} & ::= \Box \mid (\overline{E}\, t) & \\
R \in \mathsf{Redex} & ::= (\mathsf{let}^n x = t_2\ \mathsf{in}\ t_1) \mid ((\lambda x\, t_1)\, t_2) & (n \geq 1) \\
s \in \mathbb{S} & = \{\, (\overline{t}, R) \mid \overline{t} \text{ has 1 hole } \Box \,\} & \\
\mathbb{E} & ::= (\overline{E}, R) & \\
r \in \mathbb{R} & = \{\mathsf{diverges}, \mathsf{stuck}, \mathsf{halt}\} &
\end{array}
$$

In the term syntax, $(\mathsf{let}^n x = t_2\ \mathsf{in}\ t_1)$ is used to indicate a *marked* β-redex. Terms and contexts are identified modulo α-conversion as usual. For contexts, α-conversion can not rename bound variables whose scope includes a hole. Substitution of t for x in t', written $t'[x := t]$, is defined as usual. Placing a term or context X in the hole of a one-hole context $\overline{t}$, written $\overline{t}[X]$, is defined as usual.

Now, finish defining the AES by supplying the functions.

$$
\begin{array}{lll}
\mathsf{endpoints}(\overline{t}, (\mathsf{let}^n x = t_2\ \mathsf{in}\ t_1)) & = & (\overline{t}[\mathsf{let}^n x = t_2\ \mathsf{in}\ t_1], \overline{t}[t_1[x := t_2]]) \\
\mathsf{endpoints}(\overline{t}, (\lambda x\, t_1)\, t_2) & = & (\overline{t}[(\lambda x\, t_1)\, t_2], \overline{t}[t_1[x := t_2]]) \\
\mathsf{result}(t) & = & \begin{cases} \mathsf{diverges} & \text{if } \neg \mathsf{has\text{-}nf}(\mathbb{E}, t) \\ \mathsf{halt} & \text{if } t \xrightarrow{\mathbb{E},\mathsf{nf}}\!\!\!\!\rightarrow\ \lambda x\, t' \\ \mathsf{stuck} & \text{if } t \xrightarrow{\mathbb{E},\mathsf{nf}}\!\!\!\!\rightarrow\ t' \neq \lambda x\, t'' \end{cases}
\end{array}
$$

Define an accompanying mark structure as follows.

$$
\begin{array}{ll}
\mathsf{Marks} & = \{0, 1, 2, \ldots\} \\
\mathsf{noMark} & = 0 \\
\mathsf{markOf}(\overline{t}, (\mathsf{let}^n x = t_2\ \mathsf{in}\ t_1)) & = n
\end{array}
$$

$$\begin{aligned}
&\mathsf{markOf}(\overline{t}, (\lambda x\, t_1)t_2) && = 0 \\
&\mathsf{rename}(m_1, m_2) && = \theta \\
&\text{where } \begin{cases}
\theta(x) & = x \\
\theta(\lambda x\, t) & = \lambda x\, \theta(t) \\
\theta(t_1\, t_2) & = \theta(t_1)\, \theta(t_2) \\
\theta(\mathsf{let}^{m_1} x = t_2 \,\mathsf{in}\, t_1) & = (\mathsf{let}^{m_2} x = \theta(t_2) \,\mathsf{in}\, \theta(t_1)) & \text{if } m_2 \neq 0 \\
\theta(\mathsf{let}^{m_1} x = t_2 \,\mathsf{in}\, t_1) & = (\lambda x\, \theta(t_1))\, \theta(t_2) & \text{if } m_2 = 0 \\
\theta(\mathsf{let}^{m} x = t_2 \,\mathsf{in}\, t_1) & = (\mathsf{let}^{m} x = \theta(t_2) \,\mathsf{in}\, \theta(t_1)) & \text{if } m \neq m_1
\end{cases}
\end{aligned}$$

Lemma 7.1 (The Framework User's Proof Burden).

1. *Axioms 3.3, 6.1, and 3.4 hold.*
2. $\mathsf{WB}{+}\mathsf{Std}(\mathbb{M})$.
3. $\mathsf{NM}(\mathbb{S})$.
4. *If* $t_1 \longrightarrow t_2$, *then* $\overline{t}[t_1] \longrightarrow \overline{t}[t_2]$ *for any context* $\overline{t}$. □

Proof. Many standard proofs by induction which are left to the reader. The only difficult bit is $\mathsf{BE}(\mathbb{M})$ (part of $\mathsf{WB}{+}\mathsf{Std}(\mathbb{M})$). First, $\mathsf{Trm}(\mathbb{M})$ is proven by a known argument (e.g., see [5]) of rearranging the mark values so that rewriting decreases the multiset of all marks in the term in the multiset extension of $<$. Because the rewriting system is finitely branching, this is equivalent to $\mathsf{Bnd}(\mathbb{M})$, which in turn implies $\mathsf{BE}(\mathbb{M})$. □

Theorem 7.2 (Meaning Preservation). $\mathbb{S} \subseteq \mathsf{MP}$. □

Proof. Everything implicitly relies on lemma 7.1(1). By lemma 7.1(2) and theorem 5.4(1), $\mathsf{SLift}(\mathbb{M})$ and $\mathsf{SProj}(\mathbb{M})$. By lemma 4.2(2,3,5,7,9), $\mathbb{S} \cap \mathbb{M} = \mathbb{M} \subseteq \mathsf{LPT}$. By lemma 7.1(3) and theorem 6.4, $\mathbb{S} \subseteq \mathsf{LPT}$. By theorem 4.3(1), $\mathbb{S} \subseteq \mathsf{MP}$. □

Corollary 7.3 (Observational Equivalence). *If* $t_1 \longrightarrow t_2$, *then* $\mathsf{result}(\overline{t}[t_1]) = \mathsf{result}(\overline{t}[t_2])$. □

Proof. Suppose $t_1 \longrightarrow t_2$. By lemma 7.1(4), $\overline{t}[t_1] \longrightarrow \overline{t}[t_2]$. By theorem 7.2 and the definition of MP, $\mathsf{result}(\overline{t}[t_1]) = \mathsf{result}(\overline{t}[t_2])$. □

8 Related Work

The most closely related work is by Machkasova and Turbak [16, 17, 18]. Their work is discussed throughout this paper, so only a few points will be made here. First, our BE property corresponds to their complicated notion of γ-development [18, sec. 4.5]. The γ-development idea may be implicitly the same as BE [18, p. 193], but the exact relationship is unclear due to the complexity. Second, Machkasova's requirement of γ-confluence on evaluation is incomparable with our requirement of evaluation subcommutativity (axiom 3.3). Because γ-confluence involves the complicated γ-development machinery, we prefer our simpler requirement. Third, our proof diagrams for parts 1 and 2 of lemma 5.3 are similar to some in [18], but are simpler because we do not use γ-developments and we treat marks for developments separately (section 6).

Ariola and Blom [2] define the notion ARSI (ARS (abstract rewriting system) with information content). Using the ordering of an ARSI $\mathcal{A}$, they obtain the *infinite normal form* of a term t from the *information content* of all terms t' such that $t \twoheadrightarrow t'$. They show how to prove $\mathcal{A}$ preserves infinite normal forms by finding a subset $\mapsto \subseteq \rightarrow$ satisfying a diagram roughly like this [2, cor. 4.14]:

$$\begin{array}{ccc} t_1 & \mapsto\!\!\!\!\twoheadrightarrow & t_2 \\ \updownarrow & & \vdots \circ \\ t_3 & \mapsto\!\!\!\!\twoheadrightarrow & t_4 \end{array}$$

The quickest explanation of their $\circ$ relation is to point out that the closest corresponding diagram in our AES framework would be this:

$$s \in \mathsf{GLP} \iff \begin{array}{ccc} t_1 & \overset{\mathbb{E}}{\twoheadrightarrow} & t_2 \\ s\updownarrow & \mathbb{E} & \vdots \mathsf{S}\cup(\mathbb{N}^{-1}) \\ t_3 & \overset{\mathbb{E}}{\twoheadrightarrow} & t_4 \end{array}$$

Key differences between the Ariola/Blom approach and ours are as follows. First, they provide no abstract methods for proving their diagram (corresponding to our elementary diagrams in section 5) but instead prove it individually for each use. Second, the GLP diagram is a stronger requirement than LPT (in fact, $\mathsf{LP} \subseteq \mathsf{GLP} \subseteq \mathsf{LPT}$), so our methods in section 4 are more general. Third, their framework does not provide help in showing the correspondence (needed to prove observational equivalence for the rewriting system) between infinite normal forms (their notion of meaning) and the actual operational semantics, so this burden is left to the user. Fourth, they encourage using a notion of information content which is more complicated than needed for proving meaning preservation (unlike our set $\mathbb{R}$); in fact, their information content seems enough to build a fully abstract model.

Odersky [23] gives conditions proving that a proposed contextually closed transformation $\sim$ is an observational equivalence. One condition is that $\sim$ is *locally stable* [23, p. 2, diagram (2)]:

$$\begin{array}{ccc} t_1 & \rightarrow & t_2 \\ {\scriptstyle\sim_1}\downarrow & \simeq & \vdots {\scriptstyle\sim_1} \\ t_3 & \cdots & t_4 \end{array}$$

The relation $\sim_1$ is *parallel similarity*, i.e., the use of $\sim$ simultaneously at many different (presumably non-overlapping) positions. Another condition is that $\sim$ *preserves answers*, i.e., $t_1 \sim t_2 \Rightarrow (\mathsf{is\text{-}nf}(\mathbb{E}, t_2) \Rightarrow t_1 \twoheadrightarrow t_2)$.

Odersky's approach is related as follows. Where Odersky uses $\rightarrow$ (normal rewriting) and $\simeq$ (observational equivalence), we would use $\xrightarrow{\mathbb{E}}$ and $\xrightarrow{\mathbb{E}}\!\!\!\!\twoheadrightarrow$. Odersky's approach has two versions. In the version shown above, meaning preservation is defined as convertibility in the entire rewriting system with a set of answers ($\mathbb{E}$-normal forms in our setting). The question is then whether more rewrite rules can be safely added. In this case, the diagram must be proven for all rewrite steps. The other version takes an evaluation strategy like we do. In this case, using $\simeq$ on the bottom edge seems more general, but it also seems that in practice this diagram edge would be completed with $\mathbb{E}$ steps. Where Odersky uses $\sim_1$,

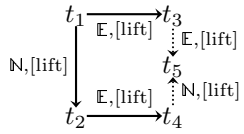
we would typically use $\xrightarrow{\mathcal{S},\mathbb{M}}$» and a combination of one of the well-behavedness conditions of section 5 and the marks of section 6. Odersky's use of parallel (simultaneous) rewriting corresponds to our use of a termination property.

Key differences between Odersky's approach and ours are as follows. Much of Odersky's approach is tied to syntactic extensions of the λ-calculus while our approach is abstract. Odersky does not provide elementary diagrams where each given edge is a single use of a rewrite rule; it seems that one must work with full parallel similarity. Odersky appears to assume standardization is already proven while our approach proves whatever standardization is needed and can work without it. Odersky's approach requires a notion of "preserving evaluation contexts" which we do not fully understand but which we are fairly sure one of our intended applications does not satisfy. Odersky does not distinguish terms that go wrong from those that either diverge or halt normally; thus his framework can not verify that rewriting does not switch between non-wrong and wrong.

9 Future Work

The generalizations of our AES framework and LPT diagrams were developed to handle $\lambda^{:=,\mathsf{letrec}}$, a calculus we are developing for reasoning about call-by-value higher-order programs with mutable reference cells and mutually recursive definitions (i.e., letrec). Evaluation of assignment statements can introduce cycles in the store, so evaluation results may need letrec even if the initial program was letrec-free. A specific evaluation strategy is given for the $\lambda^{:=,\mathsf{letrec}}$ calculus to define the meaning of programs. Calculi for assignments have been done before (e.g., [11]), but $\lambda^{:=,\mathsf{letrec}}$ also includes improvements like very simple evaluation contexts as well as rules for letrec in the style of the work of Ariola and Blom [2]. The only previously known methods for reasoning about the correctness of Ariola/Blom style letrec rules seem more difficult to us.

The development of $\lambda^{:=,\mathsf{letrec}}$ is nearing completion. Because $\lambda^{:=,\mathsf{letrec}}$ is non-confluent (due to using rules for letrec that Ariola and Klop [5] proved non-confluent), we were using the lift & project method to prove meaning preservation. It does not have finite developments, but has a number of rule subsets whose associated rewrite step sets satisfy the BE property. The last barrier to completing the proof of meaning preservation was several critical pairs of a rule named [lift] (name unrelated to the Lift diagram from definition 4.1). One particularly irritating critical pair is only completable as follows:



Unfortunately, this breaks standardization, so the lift & project proof method fails. We considered changing the definition of $\lambda^{:=,\mathsf{letrec}}$, but felt that the changes to "fix" this critical pair would probably break something else. Also, the rules of $\lambda^{:=,\mathsf{letrec}}$ are clearly meaning preserving, so we felt that rather than forcing

$\lambda^{:=,\mathsf{letrec}}$ through awkward contortions to fit a weak proof method, it was the proof method that should be fixed. Fortunately, the $\mathsf{WB}\backslash\mathsf{Std}$ property can be proven for the [lift] rule steps, so we expect to complete the $\lambda^{:=,\mathsf{letrec}}$ work soon.

After $\lambda^{:=,\mathsf{letrec}}$ is completed, we want to apply our proof methods to equational reasoning for assembly language and maybe also to explicit substitutions.

References

[1] Z. M. Ariola, S. Blom. Cyclic lambda calculi. In *Theoretical Aspects Comput. Softw. : Int'l Conf.*, Berlin, 1997. Springer.

[2] Z. M. Ariola, S. Blom. Skew confluence and the lambda calculus with letrec. *Ann. Pure Appl. Logic*, 117(1–3), 2002.

[3] Z. M. Ariola, M. Felleisen. The call-by-need lambda calculus. *J. Funct. Programming*, 3(7), 1997.

[4] Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, P. Wadler. The call-by-need lambda calculus. In *Conf. Rec. 22nd Ann. ACM Symp. Princ. of Prog. Langs.*, 1995.

[5] Z. M. Ariola, J. W. Klop. Lambda calculus with explicit recursion. *Inform. & Comput.*, 139, 1997.

[6] F. Baader, T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[7] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland, revised edition, 1984.

[8] *Programming Languages & Systems, 9th European Symp. Programming*, vol. 1782 of *LNCS.* Springer-Verlag, 2000.

[9] M. Felleisen, D. Friedman. Control operators, the SECD-machine, and the λ-calculus. In M. Wirsing, ed., *Formal Description of Programming Concepts — III.* North-Holland, 1986.

[10] M. Felleisen, D. P. Friedman. A syntactic theory of sequential state. *Theoret. Comput. Sci.*, 69(3), 1989.

[11] M. Felleisen, R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoret. Comput. Sci.*, 102, 1992.

[12] K. Fisher, J. Reppy, J. G. Riecke. A calculus for compiling and linking classes. In ESOP '00 [8].

[13] G. Gonthier, J.-J. Lévy, P.-A. Melliès. An abstract standardisation theorem. In *Proc. 7th Ann. IEEE Symp. Logic in Comput. Sci.*, 1992.

[14] D. J. Howe. Equality in lazy computation systems. In *Proc. 4th Ann. Symp. Logic in Comput. Sci.*, Pacific Grove, CA, U.S.A., 1989. IEEE Comput. Soc. Press.

[15] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Inform. & Comput.*, 124(2), 1996.

[16] E. Machkasova. Techniques for proving observational equivalence. ASCII notes that later turned into [17]. Not sure about year, 1998.

[17] E. Machkasova, F. A. Turbak. A calculus for link-time compilation. In ESOP '00 [8].

[18] E. L. Machkasova. *Computational Soundness of Non-Confluent Calculi with Applications to Modules and Linking.* PhD thesis, Boston Univ., 2002.

[19] J. Maraist, M. Odersky, P. Wadler. The call-by-need lambda calculus. *J. Funct. Programming*, 8(3), 1998.

[20] P.-A. Melliès. Axiomatic Rewriting Theory IV: A diagrammatic standardization theorem. Submitted, 2001.

[21] R. Muller. M-LISP: A representation-independent dialect of LISP with reduction semantics. *ACM Trans. on Prog. Langs. & Systs.*, 14(4), 1992.

[22] M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Math.*, 43(2), 1942.

[23] M. Odersky. A syntactic method for proving observational equivalences. Research Report YALEU/DCS/RR-964, Yale Univ., Dept. of Comp. Science, 1993.

[24] A. M. Pitts. Operationally-based theories of program equivalence. In *Semantics and Logics of Computation*, vol. 14 of *Publications of the Newton Institute.* Cambridge University Press, 1997.

[25] G. D. Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theoret. Comput. Sci.*, 1, 1975.

[26] W. Taha. A sound reduction semantics for untyped CBN multi-stage computation: Or, the theory of MetaML is non-trivial. In *Proceedings of the 2000 ACM SIGPLAN Workshop on Evaluation and Semantics-Based Program Manipulation (PEPM-00)*, N.Y., 2000. ACM Press.