Two Applications of Standardization and Evaluation in Combinatory Reduction Systems

Robert Muller*
Boston College
http://www.cs.bc.edu/~muller/

J. B. Wells†
Heriot-Watt University
http://www.cee.hw.ac.uk/~jbw/

June 30, 2000

Abstract

We present two worked applications of a general framework that can be used to support reasoning about the operational equality relation defined by a programming language semantics. The framework, based on Combinatory Reduction Systems, facilitates the proof of standardization theorems for programming calculi. The importance of standardization theorems to programming language semantics was shown by Plotkin [Plo75]: standardization together with confluence guarantee that two terms equated in the calculus are semantically equal. We apply the framework to the λ_v-calculus and to an untyped version of the λ^{CIL}-calculus. The latter is a basis for an intermediate language being used in a compiler.

1 Introduction

Optimizing compilers make many transformations in which some program fragment M is replaced by another, hopefully more efficient, program fragment N. From the operational point of view, such transformations are justified when fragment N is *observationally equivalent* to M, i.e., when there are no program contexts in which distinctions between the two can be observed. The justification of such transformations has been one of the principal motivations in the development of programming language semantics.

One of the seminal results in this area is the approach developed by Plotkin [Plo75]. Among other important developments, Plotkin showed how an operational semantics of a programming language can be associated with an equational theory (or *calculus*) of program fragments. The equational theory is shown to be sound with respect to the operational semantics in the sense that fragments equated in the theory are guaranteed to be observationally equivalent under the operational semantics. A compiler writer is therefore justified in replacing M with N when equation $M = N$ is in the theory.

In order to use this general approach, one must define both an operational semantics of a programming language and a calculus of fragments and then establish the appropriate connection between them. In [Plo75] and many subsequent studies (e.g., [FF89, FH92, Mul92]) the operational semantics and calculus are defined independently and then connected via a *standardization* theorem for the calculus. More recently (e.g., [AF97]) it is common to start with the calculus, prove a standardization theorem for it and then simply define the operational semantics as a restriction of the standard rewriting sequence for the calculus. In either approach, a standardization theorem for the calculus is required.

Unfortunately, the usual approach to proving standardization for calculi intended for program reasoning is somewhat unsatisfying. In practice, such proofs have been hand crafted for each calculus (e.g., [FF89, FH92, Mul92, AF97]). The usually *ad hoc* nature of such proofs means that proving standardization for a new language is a laborious and error-prone task of adapting an existing proof to the new language. The

*Corresponding author. Voice: +1 617 552 3964. Fax: +1 617 552 2097. E-mail: muller@cs.bc.edu. This author's work was partially supported by NSF grant EIA–9806746 and by a Faculty Fellowship from the Wallace E. Carroll School of Management.

†This author's work was partially supported by NSF grant CCR–9417382 and EPSRC grant GR/L 36963. Part of this author's work was done while visiting Boston University and part while employed at the University of Glasgow.

process does not usually give much insight into the language and may need to be repeated from scratch if the rules are changed.

General-purpose methods for standardization have been developed, but have had difficulties. The earlier methods are too weak, e.g., the method of Klop [Klo80] only supports rule sets where evaluation can proceed from left to right. Some more recent methods are sufficiently general but are at such an abstract level that they provide little help with the proof burden. These methods include various syntax-free frameworks [GLM92, KG96]. They do not significantly reduce the burden in comparison with the hand-crafted approach because the programming language theorist still has to prove that their system satisfies the various axioms. In practice, this is very difficult problem in its own right.

1.1 Contributions of this Paper

In this paper we present two worked examples of a general framework that supports rigorous proofs of standardization theorems. The framework, developed in [WM00], is abstract enough to handle a large class of languages, yet it is nevertheless concrete enough that the programming language theorist can embed their language in it without too much difficulty. The framework uses higher-order rewriting systems (HORS's), specifically, combinatory reduction systems (CRSs). Although the framework is developed elsewhere, some brief remarks about its applicability will be helpful in understanding the handling of the examples in this paper.

1. Our framework is based on CRSs and using it for a calculus involves formulating a CRS version of the calculus. There can be immediate benefits to this outside what is provided by our framework, e.g., one can use standard results for CRSs to prove such properties as confluence.

2. Once the calculus has been formulated as a CRS, if the CRS is orthogonal and a *good* (satisfying certain criteria) redex ordering function can be found, then the framework provides a notion of standard rewriting and a theorem that any rewrite sequence can be converted into a standard one.

3. If a certain property can be shown, then the framework provides a *good* subterm ordering function which yields a good redex ordering function. The required property is that the CRS is orthogonal and the left-hand sides of the rewrite rules are such that when partial matches for more than one left-hand side can be found, there is always a position to check for the rest of the match which is in common among all of the partial matches. Programming language calculi generally seem to have this property.

4. If the CRS in addition qualifies as a *constructor system*, then the framework automatically derives definitions of *values*, *evaluation contexts*, and *evaluation* directly from the rules of the CRS and provides the appropriate theorems about these definitions. CRSs that arise in programming language semantics are often constructor systems.

The contributions of this paper involve applying the framework to two example call-by-value programming language calculi, the well known λ_{v}-calculus of Plotkin [Plo75] and the more complicated (but not atypical) calculus λ^{CIL}, a particular λ-calculus that is being used as the basis for a compiler intermediate language [WDMT97, DMTW97]. In detail, the contributions are:

1. Section 3 shows all of the technical details of how to encode λ_{v} and λ^{CIL} as two CRSs, Σ_{v} and Σ_{CIL}. The two CRSs are proved to be effectively isomorphic to the original calculi, thereby proving that results about the CRSs are in fact relevant to the original calculi. In addition, section 3 proves that Σ_{v} and Σ_{CIL} are orthogonal. As a side benefit, this proves that all of the systems are confluent.

2. Section 5 proves that Σ_{v} and Σ_{CIL} satisfy all of the criteria mentioned above and thereby obtains Plotkin/Wadsworth/Felleisen-style standardization theorems for these CRSs, thereby showing for each the consistency of its operational semantics with its rewriting system. Although standardization has been proven for λ_{v} before, this is the first proof of standardization for λ^{CIL}.

1.2 Related Work

Standardization of a rewriting system is the property that, for any rewriting sequence, there is another *permutation-equivalent* rewriting sequence that contracts the redexes in a nice order which is called "standard". One of the important properties of standard rewriting is that it provides a normalizing rewriting strategy. Standardization was first shown by Curry and Feys [CF58].

There are a variety of methods of proving standardization. Two important methods were devised by Klop, (1) identifying the "leftmost" (most needed) redex contracted in a rewriting sequence and performing it first and (2) replacing anti-standard pairs with standard complete developments [Klo80]. We use Klop's second method. Huet and Lévy define standard reductions for orthogonal term rewriting systems (TRS's) as outside-in reductions, using a leftmost choice function to determine a unique standard reduction for a permutation equivalence class [HL91a]. Their proof of termination of the standardization algorithm depends on the disjointness of residuals through arbitrary rewriting sequences, a property of orthogonal TRS's but not of orthogonal HORS's. Gonthier, Lévy, and Melliès proved a standardization theorem for abstract rewriting [GLM92]. Melliès has done further work on abstract standardization [Mel98].

Other research into standardization includes the following. Jim and Meyer use a combination of Klop's first and second method to prove standardization for a variant of PCF and then use that result to prove the context lemma [JM96]. Suzuki used Klop's second method to prove standardization for conditional term rewriting systems [Suz96]. There is some more discussion of Klop's second method in [vO96]. Khasidashvili and Glauert proved something they call abstract standardization [KG96] and what they call relative standardization [GK]. Standardization has been used extensively for validating the consistency of an operational semantics with a calculus by Plotkin, Felleisen, Ariola, Friedman, Hieb, Muller, and others not listed [Plo75, FF89, FH92, Mul92, AF97]. The method of Ariola and Felleisen depends on disjoint redexes having disjoint residuals.

Higher-order term rewriting has been presented in a number of different formalisms, including several variations on the format of CRSs [Klo80, Nip91, KvO95, Ken89, vR96, vO94, KvOvR93, Kha90, Tak93, Wol93].

Because one of the aims of standardization is finding normalizing rewriting strategies, much work on normalization is related. Huet and Lévy devised the idea of needed redexes, those which must be contracted in any rewriting sequence to normal form [HL91a, HL91b]. To aid in finding needed redexes, they devised the notions of sequentiality and strong sequentiality. Klop and Middeldorp provide a quite readable discussion of strong sequentiality [KM91].

Barendregt, Kennaway, Klop, and Sleep raised the idea of needed redexes to the λ-calculus [BKKS87]. Glauert, Khasidashvili, Nöcker, and Middeldorp have all written about "normalization" to sets of terms that are not exactly normal forms [GK, Nöc94, Mid97]. Van Raamsdonk showed that the outermost-fair (multi-step) strategy is normalizing for some HORS's [vR96]. Kennaway, Antoy, and Middeldorp have devised 1-step normalizing rewriting strategies for "non-sequential" systems [Ken89, AM96]. Sekar and Ramakrishnan have another approach to normalization [SR93].

1.3 Overview

Section 2 summarizes mathematical nomenclature and defines combinatory reduction systems (CRSs). Section 3 presents the two λ-calculi, defines their corresponding CRSs and proves confluence results. Section 4 summarizes the definitions and salient theorems from [WM00]. Subsection 4.2 summarizes the details of obtaining good redex ordering functions via a subterm ordering function generator which is parameterized over the CRS. Subsection 4.3 summarizes the details of deriving a notion of evaluation for certain constructor CRSs. Section 5 applies the results of the previous sections to the problem of verifying the desirable properties of the programming language calculi λ_{v} and λ^{CIL}.

2 Preliminaries

2.1 Mathematical Preliminaries

Most of the notation presented in this subsection is quite standard and is given here only to avoid ambiguities over minor differences in usage. However, some of the notation here is new.

Abbreviations for Sequences The notation $\vec{a}$ abbreviates the notation $a_1, a_2, \cdots$ where the number of items is unspecified or clear from the context. The notation $\vec{a}^n$ abbreviates the notation $a_1, \ldots, a_n$.

Binary Relations A *binary relation* $\Re$ is any set of pairs. Let $\Re$ range over binary relations. The statements $\Re(a, b)$ and $a \ \Re \ b$ mean the same as $(x, y) \in \Re$ and the expression $\Re^{-1}$ denotes the relation $\{ (b, a) \mid (a, b) \in \Re \}$. A relation $\Re$ is *transitive* iff $\Re(a, b)$ and $\Re(b, c)$ implies $\Re(a, c)$. A relation $\Re$ is *anti-symmetric* iff $\Re(a, b)$ and $\Re(b, a)$ implies $a = b$. A relation $\Re$ is *reflexive* on some set S iff $\Re(a, a)$ for every $a \in S$. A relation $\Re$ is *irreflexive* iff $\Re(a, b)$ implies $a \neq b$. A relation $\Re$ is *well founded* iff there is no infinite sequence a_1, a_2, a_3, ..., such that $\Re(a_i, a_{i+1})$ for all $i \geq 1$.[1] A relation $\Re$ is *finitely branching* iff $\{ b \mid \Re(a, b) \}$ is finite for all a.

Functions A *function* f is a binary relation such that if $(a, b) \in f$ and $(a, c) \in f$ then $b = c$. In this case, we write the pairs in f in the form $(a \mapsto b)$. Given a function f and a value a, if a value b exists such that $(a \mapsto b) \in f$, then $f(a)$ denotes the value b, else $f(a)$ is undefined. The *domain of definition* of a function f is $\mathrm{DomDef}(f) = \{ a \mid (a \mapsto b) \in f \}$ and the *range* of f is $\mathrm{Ran}(f) = \{ b \mid (a \mapsto b) \in f \}$. The assertion $f : S_1 \to S_2$ holds whenever $\mathrm{DomDef}(f) \subseteq S_1$ and $\mathrm{Ran}(f) \subseteq S_2$, in which case we can call S_1 and S_2 respectively a *domain* and a *codomain* of f. A function f is *total* w.r.t. a domain S iff $\mathrm{DomDef}(f) = S$. Given set S, if $S \subseteq \mathrm{DomDef}(f)$, then $f(S) = \{ f(a) \mid a \in S \}$, otherwise $f(S)$ is undefined. The *restriction* of a function f to a set S is the new function $f \downarrow S = \{ (a \mapsto b) \mid (a \mapsto b) \in f, a \in S \}$. The *composition* of functions f and g, notation $f \circ g$, is the function such that $(f \circ g)(x) = f(g(x))$. The *n-fold composition* of a function f is the function f^n such that $f^n(a) = f(f^{n-1}(a))$ if $n > 0$ and $f^0(a) = a$. The expression $f[a \mapsto b]$ denotes the function $(f \setminus \{ (a \mapsto c) \mid f(a) = c \}) \cup \{(a \mapsto b)\}$.

Orders An *order* O is a transitive and anti-symmetric binary relation. An order O is a *partial order* on set S iff O is reflexive on S. An order O is a *total order* on set S iff O is a partial order on S and $O(a, b)$ or $O(b, a)$ for every $a, b \in S$. When we use a symbol for a partial order like $\leq$ or $\preceq$ or $\trianglelefteq$, possibly superscripted or subscripted, the removal of the bottom line, e.g., $<$, flipping the symbol, e.g., $\geq$, and slashing the symbol, e.g., $\not\leq$, have the usual meaning. The set of minimal elements of a set S w.r.t. an order O is $\min_O S = \{ a \mid a \in S, \nexists b \in S.\, O(b, a) \text{ and } b \neq a \}$. If the context guarantees that $\min_O S = \{a\}$ will be a singleton set, we may freely use $\min_O S$ as the value a.

Strict Orders An order O is *strict* iff O is irreflexive. A *strict partial order* is any strict order. An order O is a *strict total order* on set S iff O is strict and $O(a, b)$, $O(b, a)$, or $a = b$ for every $a, b \in S$. (Following an unfortunate tradition, a strict partial order is not a partial order and a strict total order is not a total order.)

Sequences The expression $\langle a_1, a_2, \ldots \rangle$ is the sequence of a_1, a_2, ... treated as a single object. The expression $\langle\rangle$ denotes the 0-length sequence. Given a set S, the expression S^n denotes the set of sequences $\{ \langle \vec{a}^n \rangle \mid \{\vec{a}^n\} \subseteq S \}$. Given sequences $\chi_1 = \langle \vec{a}^n \rangle$ and $\chi_2 = \langle b_1, b_2, \ldots \rangle$ where the first sequence is finite, the expression $\chi_1 \cdot \chi_2$ (the concatenation of χ_1 and χ_2) denotes the sequence $\langle \vec{a}^n, b_1, b_2, \ldots \rangle$. Given a sequence χ, the expression $|\chi|$ evaluates to the number of elements in χ if χ is finite and evaluates to ω if χ is infinite. We consider only countable sequences. If a sequence $\langle a_1, a_2, \ldots \rangle$ is used in a context requiring a set, we treat

[1] Ideally, this notion would say that $\Re$ has no infinite *descending* chains. But which direction is descending, to the left or to the right? This differs between relations, e.g., $<$ descends to the left and $>$ descends to the right. Because associating a direction for "descent" with each relation seems too painful, we follow Baader and Nipkow [BN98, p. 14] in stating that descent is to the right. Some others have chosen that descent is to the left, e.g., Taylor [Tay99, p. 97].

it as the set $\{a_1, a_2, \ldots\}$ (where duplicates have the same effect as a single occurrence). We write $[a_1, a_2, \ldots]$ for a sequence that has no duplicates, i.e., where $a_i \neq a_j$ if $i \neq j$.

Relations and Orders from Sequences Given a sequence $\chi = \langle a_1, a_2, \ldots \rangle$, the statement $a_i \prec_\chi a_j$ holds iff $i < j$. If χ is a sequence with no duplicates, i.e., it can be written as $\chi = [a_1, a_2, \ldots]$, then $\prec_\chi$ is a strict total order on the set of elements of χ. In this case, we may refer to the sequence χ as a strict total order.

Lexicographic Order If O is a strict order, then its *lexicographic extension* O_{lex} is the strict order on sequences such that $O_{\text{lex}}(\chi_1, \chi_2)$ iff there exists some common prefix χ such that $\chi_1 = \chi \cdot \chi_1'$, $\chi_2 = \chi \cdot \chi_2'$, and either $\chi_1' = \langle\rangle \neq \chi_2'$ or $\chi_1' = \langle a, \ldots \rangle$, $\chi_2' = \langle b, \ldots \rangle$, and $O(a, b)$. Observe that if O is a strict total order, then O_{lex} is a strict total order. Let $\min_{\text{lex}}$ stand for $\min_{<_{\text{lex}}}$.

Naming Conventions When using symbols to range over various kinds of entities, we follow the following conventions:

a, b, c, X	arbitrary entity	f	function
i, j, k, l, m, n	natural number	p, q	path
r	reduction rule	s, t	metaterm
u, v	term	w	preterm
x, y, z	term variable	C	context
E	evaluation context	F, G, H, I	function symbol
F	piece of evaluation context	M, N	term of λ_{v} or λ^{CIL}
O	order	P	set of paths
R	set of reduction rules	R	renaming of metavariables
S	set	V	value (restricted term of λ_{v} or λ^{CIL})
Z	CRS metavariable	$\Re$	binary relation
γ	subterm ordering	δ	subterm ordering or "wrong"
χ	sequence	ν	valuation
Γ	subterm ordering function	Δ	redex occurrence
Σ	combinatory reduction system		

2.2 Combinatory Reduction Systems

This section defines *combinatory reduction systems* (CRSs). We use the *functional* presentation of combinatory reduction systems [KvOvR93] rather than the original *applicative* presentation [Klo80]. Both ways of presenting CRSs have the same expressiveness. They differ in minor ways such as the number of "garbage terms" that must be ignored and how to determine the head symbol of a redex. We find it much easier to rigorously prove theorems using the functional CRS presentation.

For those familiar with CRSs, the non-standard bits are (1) the definition of the tree of a term, (2) the definition of subterm occurrence, (3) the definition of valuation, (4) the restriction requiring reduction rules to be fully extended, and (5) some notation for reduction sequences.

2.2.1 Basic CRS Definitions

Alphabet The alphabet used in constructing the preterms of a CRS consists of the following:

1. The countably infinite set Fun of *function symbols*. Let F, G range over Fun.
2. The countably infinite set Var of (ordinary) *variables*. Let x, y, z range over Var.
3. The countably infinite set MVar of *metavariables*. Let Z range over MVar.
4. The symbols "(", ")", "[", "]", "$\Box$" and ",".

Each function symbol F or metavariable Z has a fixed arity, written $\mathrm{Arity}(F)$ or $\mathrm{Arity}(Z)$. There are an infinite number of functions symbols and metavariables of each arity. The arity will often be indicated by writing $F^{(n)}$ or $Z^{(n)}$ in a statement, which is equivalent to writing merely F or Z and adding the side condition that $\mathrm{Arity}(F) = n$ or $\mathrm{Arity}(Z) = n$. The arity will usually be obvious. Ordinary variables are considered to have arity 0. Let $<^{\mathrm{mv}}$ be some fixed strict total order on MVar such that $>^{\mathrm{mv}}$ is well founded.

Preterms The set PTer of *preterms* is the smallest set satisfying the following conditions. Let w range over preterms.

1. If $x \in \mathrm{Var}$, then $x \in \mathrm{PTer}$.
2. If $x \in \mathrm{Var}$ and $w \in \mathrm{PTer}$, then $[x]w \in \mathrm{PTer}$.
3. If $F^{(n)} \in \mathrm{Fun}$ and $\{\vec{w}^n\} \subset \mathrm{PTer}$, then $F(\vec{w}^n) \in \mathrm{PTer}$.
4. If $Z^{(n)} \in \mathrm{MVar}$ and $\{\vec{w}^n\} \subset \mathrm{PTer}$, then $Z(\vec{w}^n) \in \mathrm{PTer}$.
5. $\Box \in \mathrm{PTer}$.

All free occurrences of x in w are bound in $[x]w$. If $X^{(0)} \in \mathrm{Fun} \cup \mathrm{MVar}$, then the preterm $X()$ may (and usually will) be written as just X. Below, the sets of contexts, metaterms, and terms will be defined as subsets of PTer.

Paths The set $\mathcal{P}$ of *paths* is the set of sequences over $\mathbb{N}$ (the natural numbers). We generally write a path as $i_0 \cdot \ldots \cdot i_n$ rather than $\langle i_0, \ldots, i_n \rangle$. In a context requiring a path, the number i is implicitly coerced to the 1-length path $\langle i \rangle$. Let p, q range over paths, let P range over sets of paths, and let ϵ denote the 0-length path. The *prefix* partial order on paths is the order $\leq$ such that $p \leq q$ ("p is a prefix of q") iff there exists a path p' such that $q = p \cdot p'$. The statement $p \mid q$ ("p is incomparable with q") means $p \not\leq q$ and $q \not\leq p$.

Tree of a Preterm The *tree* of a preterm w is an alternate representation of the essential information in w. The function Tree is the least-defined partial function from PTer to $\mathcal{P}$ to $\mathrm{Fun} \cup \mathrm{Var} \cup \mathrm{MVar} \cup \{\, [X] \mid X \in \mathrm{Var} \cup \{\bullet\} \,\} \cup \mathcal{P} \cup \{\Box\}$ such that:

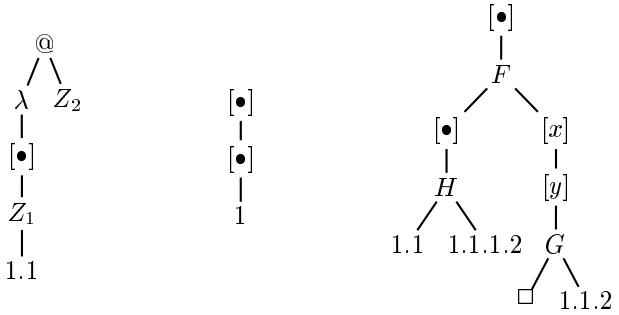
1. $\mathrm{Tree}(x) = \{\epsilon \mapsto x\}$.
2. $$\mathrm{Tree}([x]w)(p) = \begin{cases} [x] & \text{if } p = \epsilon \text{ and } \exists q.\ (\mathrm{Tree}(w)(q) = \Box \text{ and } \nexists q' \leq q.\ \mathrm{Tree}(w)(q') = [x]), \\ [\bullet] & \text{if } p = \epsilon \text{ and } \nexists q.\ (\mathrm{Tree}(w)(q) = \Box \text{ and } \nexists q' \leq q.\ \mathrm{Tree}(w)(q') = [x]), \\ \mathrm{Tree}(w)(q) & \text{if } p = 1 \cdot q \text{ and } \mathrm{Tree}(w)(q) \neq x, \\ p & \text{if } p = 1 \cdot q \text{ and } \mathrm{Tree}(w)(q) = x. \end{cases}$$
3. For $X^{(n)} \in \mathrm{Fun} \cup \mathrm{MVar}$,
$$\mathrm{Tree}\left(X(\vec{w}^n)\right)(p) = \begin{cases} X & \text{if } p = \epsilon, \\ \mathrm{Tree}(w_i)(q) & \text{if } p = i \cdot q \text{ and } 1 \leq i \leq n. \end{cases}$$
4. $\mathrm{Tree}(\Box) = \{\epsilon \mapsto \Box\}$.

The *skeleton* of a preterm w is $\mathrm{Skel}(w) = \mathrm{DomDef}(\mathrm{Tree}(w))$. A position p is *bound at q in w*, written $\mathrm{BindPos}(w, p) = q$, iff $p = q \cdot p'$ and $\mathrm{Tree}(w)(q) = [X]$ for some $X \in \mathrm{Var} \cup \{\bullet\}$ and $\mathrm{Tree}(w)(p) = p'$.

Here are three examples of the trees of preterms:

Preterm w: $@(\lambda([x]Z_1(x)), Z_2)$ $[x][y]y$ $[x]F([z]H(z,x),[x][y]G(\Box,x))$

Tree(w):



A binding is recorded in the tree as "[•]" to ignore the name of the bound variable when its name is irrelevant. A binding is recorded as "[x]" only when there is at least one hole in the scope of the binding such that filling the hole by a preterm w with free variable x should result in the capture of the free variable by the binding. Bound variables are represented by the path from the internal node of the binder to the leaf node of the bound variable. De Bruijn indices or some similar scheme could have been used instead, but it turned out to be quite convenient to record the actual path from the binder to the variable.

Quotienting by α-Conversion The statement $w \equiv w'$ ("w and w' are *tree-equivalent*") means $\mathrm{Tree}(w) = \mathrm{Tree}(w')$. For preterms without holes, tree-equivalence corresponds exactly to the standard notion of α-conversion. For preterms with holes, for which α-conversion is not usually defined, tree-equivalence gives the best possible definition of α-conversion.

Convention 2.1. Throughout the rest of this article, tree-equivalent preterms are considered equal. □

In interpreting this convention, the reader can think of the set PTer as really being a set of trees of the form given above rather than as a set of syntactic entities. The "≡" symbol is used instead of "=" to avoid confusion with equality on the syntactic entities used to write preterms and equality w.r.t. an equational theory.

Metaterms, Terms, and Contexts The set MTer of *metaterms* is the subset of PTer containing all of the preterms which do not mention □. Let s and t range over metaterms.

The set Ter of all *terms* is the subset of MTer containing all of the metaterms which do not mention metavariables. Let u and v range over terms.

The set Ctxt of *contexts* is the subset of PTer containing all of the preterms which mention "□" (the hole). Let C range over contexts. Unless otherwise specified, a context is assumed to have exactly one hole. The arity of a context is the number of holes in the context. Given a context C, its arity n (respectively the position p of its unique hole if it has only one) may be indicated by writing $C^{(n)}$ (respectively C^p) in a statement, which is equivalent to writing merely C and adding the side condition that $\mathrm{Arity}(C) = n$ (respectively $\mathrm{Arity}(C) = 1$ and $\mathrm{Tree}(C)(p) = \Box$). If $\mathrm{Arity}(C) \neq n$, then $C[\vec{w}^n]$ is undefined, otherwise $C[\vec{w}^n]$ denotes the result of replacing all of the occurrences of □ in C by $w_1, \ldots, w_n$ in order from left-to-right, possibly capturing free variables of w.

Variables The set of metavariables occurring in a preterm w is $\mathrm{MV}(w) = \mathrm{Ran}(\mathrm{Tree}(w)) \cap \mathrm{MVar}$. The statement $\mathrm{DMV}(w, w')$ ("w and w' have disjoint metavariables") holds iff $\mathrm{MV}(w) \cap \mathrm{MV}(w') = \varnothing$. For a set of preterms S, the statement $\mathrm{DMV}(S)$ holds iff $\mathrm{DMV}(w, w')$ holds for every $w, w' \in S$ such that $w \not\equiv w'$. The set of (ordinary) variables occurring free in a preterm w is $\mathrm{FV}(w) = \mathrm{Ran}(\mathrm{Tree}(w)) \cap \mathrm{Var}$.

Subterms and Subterm Occurrences A preterm w' is a *subterm* of a preterm w, written $w' \trianglelefteq w$, iff there is a context C such that $w \equiv C[w']$. Quite different from a subterm, a *subterm occurrence* is specified by its position. If $p \in \mathrm{Skel}(w)$, then $w.p$ denotes the subterm occurrence in w at position p. We write

$w_1.p_1 \equiv w_2.p_2$ to mean that $w_1 \equiv w_2$ and $p_1 = p_2$. When the context of discussion makes the preterm w obvious, p will sometimes stand for $w.p$.

REMARK 2.2. If $w \equiv C^p[w']$, some researchers will use w' to stand for subterm occurrence $w.p$. We avoid this because it can be ambiguous in two ways: (1) there may be another position $q \neq p$ and context C_1^q such that $w \equiv C_1^q[w']$ and (2) there may be another context C_2^p and metaterm w'' such that $w \equiv C_2^p[w'']$, where w' and w'' differ only in the names of free variables which are bound by C^p and C_2^p. □

Valuation A *valuation* is a function ν from $\mathrm{MVar} \cup \mathrm{Var}$ to MTer such that for each $X \in \mathrm{DomDef}(\nu)$ of arity n, the metaterm $\nu(X)$ mentions only metavariables in the distinguished set $\{\hat{Z}_1^{(0)}, \ldots, \hat{Z}_n^{(0)}\}$. (These distinguished metavariables play the part of the parameters of *substitutes* as used in other formulations of CRSs [Kah94].) Given valuation ν, let $\mathrm{FV}(\nu) = \mathrm{DomDef}(\nu) \cup \bigcup_{X \in \mathrm{DomDef}(\nu)} \mathrm{FV}(\nu(X))$. A *valuation for metaterm* s is a valuation ν such that $\mathrm{DomDef}(\nu) \supseteq \mathrm{MV}(s)$. The application $\nu(s)$ of a valuation to a metaterm is the term $\nu'(s)$ where ν' is the function from MTer to Ter defined as follows.

1. $\nu'(x) \equiv \begin{cases} x & \text{if } \nu(x) \text{ is undefined,} \\ \nu(x) & \text{otherwise.} \end{cases}$
2. $\nu'([x]t) \equiv [x']\nu'(t')$ where $[x]t \equiv [x']t'$ and $x' \notin \mathrm{FV}(\nu)$. We can always find such an x' and t' by α-conversion and it does not matter which ones we use.
3. $\nu'(F(\vec{s}^n)) \equiv F(\nu'(s_1), \ldots, \nu'(s_n))$.
4. $\nu'(Z(\vec{s}^n)) \equiv \nu''(\nu(Z))$ where $\nu'' = \{\, \hat{Z}_i^{(0)} \mapsto \nu'(s_i) \mid 1 \leq i \leq n \,\}$.

For compatibility with traditional substitution notation, let $t[X_1{:=}s_1, \ldots, X_n{:=}s_n]$ stand for $\nu(t)$ where $\nu = \{X_1 \mapsto s_1, \ldots, X_n \mapsto s_n\}$.

REMARK 2.3. Our definition of valuation differs from earlier definitions (e.g., [Klo80] and [KvOvR93]) in that (1) it uses a different (but equivalent) mechanism to handle the replacement of metavariables of arity greater than 0 and (2) it also allows replacing ordinary variables because it is needed in more situations. □

Reduction Rule A *pattern* is a metaterm s such that any metavariable $Z^{(n)}$ occurs in s only in the form $Z(\vec{x}^n)$ where $x_1, \ldots, x_n$ are n distinct (ordinary) variables. A *reduction rule* r is a pair $s \to t$ of metaterms satisfying the following conditions.

1. The metaterm s is a pattern.
2. The metaterm s is of the form $F(\vec{s}^n)$ for some function symbol F.
3. Both s and t are closed, i.e., in s and t each (ordinary) variable x occurs in the scope of a matching binder $[x]$.
4. Any metavariable which occurs in t also occurs in s.

For $r = s \to t$, let $\mathrm{LHS}(r) \equiv s$ (the *left-hand side*) and $\mathrm{RHS}(r) \equiv t$ (the *right-hand side*).

A position p is *internal* in pattern s iff p is the position in s of a function symbol, a binder, or an ordinary variable which is not a child of a metavariable. Formally, this is written $p \in \mathrm{Int}(s)$ and defined as follows.

$$\begin{aligned} \mathrm{Int}\,(s) = \quad & \{\, p \mid \mathrm{Tree}(s)(p) \in \mathrm{Fun} \cup \{[\bullet]\} \,\} \\ & \cup \{\, p \mid \mathrm{Tree}(s)(p) \in \mathcal{P} \cup \mathrm{Var}, (p = \epsilon \text{ or } (p = q \cdot i \text{ and } \mathrm{Tree}(s)(q) \notin \mathrm{MVar})) \,\} \end{aligned}$$

This notion is extended to reduction rules so that for rule $r = s \to t$ it holds that $\mathrm{Int}(r) = \mathrm{Int}(s)$ and position p is *r-internal* iff p is internal in s.

Rules Differing Only by Metavariable Names Two metaterms s_1 and s_2 are the *same up to metavariable renaming*, written $s_1 \simeq s_2$, iff s_2 is the result of renaming the metavariables in s_1 in a one-to-one manner. For metaterms that are the same up to metavariable renaming, we will define a strict total order to support the purpose of arbitrarily choosing one of them. Let $s_1 \blacktriangleleft s_2$ hold iff $s_1 \simeq s_2$ and $\langle \vec{Z} \rangle <^{\text{mv}}_{\text{lex}} \langle \vec{Z}' \rangle$ where $\langle \vec{Z} \rangle$ and $\langle \vec{Z}' \rangle$ are the sequences of metavariables occurring respectively in s_1 and s_2, in the order of occurrence from left to right.

We extend these notions to reduction rules as follows. Let $F^{(2)}$ be some fixed function symbol. Let $r = s \to t$ and $r' = s' \to t'$ be reduction rules. Then $r \simeq r'$ iff $F(s,t) \simeq F(s',t')$ and $r \blacktriangleleft r'$ iff $F(s,t) \blacktriangleleft F(s',t')$. Let $\text{LeastUpToRenaming}(r) = \min_{\blacktriangleleft}\{\, r' \mid r \simeq r' \,\}$.

Reduction Relation Let r be a reduction rule $s \to t$, ν a valuation for s, C^p a term context, and u and v terms. If $u \equiv C^p[\nu(s)]$ and $v \equiv C^p[\nu(t)]$, then define the following.

1. The term $\nu(s)$ is an *r-redex term* and $\nu(t)$ is its *r-contractum term*.
2. The subterm/rule pair $\Delta = (u.p, r)$ is a *redex (occurrence)*.
3. The term u *reduces* to its *reduct* v by *contracting* Δ, written $u \xrightarrow{\Delta} v$.
4. The triple $u \xrightarrow{\Delta} v$, also written $\langle u, \Delta, v \rangle$, is a *reduction step*.
5. Equivalent variations indicating the rule r are $u \xrightarrow{\Delta}_r v$ and $u \xrightarrow{p}_r v$.

If R is a set of reduction rules, $u \xrightarrow{\Delta}_R v$ means $u \xrightarrow{\Delta}_r v$ for some $r \in R$. For X which is either a rule r or a rule set R, we write $u \longrightarrow_X v$ to mean $u \xrightarrow{\Delta}_X v$ for some unspecified Δ. The transitive, reflexive closure of $\longrightarrow_X$ is $\twoheadrightarrow_X$. A term u is in *normal form* w.r.t. rule set R, written $R\text{-nf}(u)$, iff there is no term v such that $u \longrightarrow_R v$. A term u has R-normal form v, written $u \xrightarrow{\text{nf}}_R v$, iff $u \twoheadrightarrow_R v$ and $R\text{-nf}(v)$. If $\Delta = (u.p, r)$, $\Delta' = (u.q, r')$, and $\Re$ is a relation on paths, then $\Delta \mathrel{\Re} \Delta'$ iff $p \mathrel{\Re} q$. The equational theory of R, written $=_R$, is the least equivalence relation on Ter containing $\longrightarrow_R$.

Reduction Sequence A *reduction sequence* σ is an alternating sequence of terms and redex occurrences beginning with a term such that (1) every 3-element subsequence of the form $\langle u, \Delta, v \rangle$ is a valid reduction step and (2) σ either ends with a term or is infinite. We write $\sigma = u \twoheadrightarrow v$ to indicate both the initial and final term and $\sigma = u \twoheadrightarrow \cdots$ to indicate only the initial term. The length of a reduction sequence, written $|\sigma|$, is the number of reduction steps it contains (ω for infinite sequences). The expression $\sigma[i..j]$ for $0 \le i \le j \le |\sigma|$ denotes the subsequence of σ starting with the ith term (where terms and steps are both numbered starting with 0) and ending with the jth term and $\sigma[i..]$ denotes the suffix starting with the ith term. Let $\sigma[i]$ be the ith term in σ. We write $\sigma \sim \sigma'$ iff σ and σ' are *coinitial* and *cofinal*, i.e., $\sigma[0] \equiv \sigma'[0]$ and $\sigma[|\sigma|] \equiv \sigma'[|\sigma'|]$. Given $\sigma = \langle u_0, \Delta_0, u_1, \Delta_1, u_2, \ldots \rangle$ where $\Delta_i = (u_i.p_i, r_i)$, we will sometimes write this as $\sigma = u_0 \xrightarrow{p_0}_{r_0} u_1 \xrightarrow{p_1}_{r_1} u_2 \cdots$. Given $\sigma = \langle \ldots, u \rangle$ and $\sigma' = \langle u, \ldots \rangle$, their concatenation $\sigma \star \sigma'$ is $\langle \ldots, u, \ldots \rangle$. Given rule set R, a reduction sequence σ is an R-reduction sequence iff the rule of every redex occurrence in σ belongs to R.

Combinatory Reduction System A *combinatory reduction system* (CRS) Σ is specified by a set of *terms* $\text{Ter}(\Sigma)$ and a set of *reduction rules* $\text{Red}(\Sigma)$ (sometimes called *rewrite rules*). The set of terms $\text{Ter}(\Sigma)$ is some subset of Ter which must be closed under the reduction relation $\longrightarrow_{\text{Red}(\Sigma)}$, i.e., if $u \longrightarrow_{\text{Red}(\Sigma)} v$ and $u \in \text{Ter}(\Sigma)$ then $v \in \text{Ter}(\Sigma)$.[2] In contexts in which a set of rules is required, we may use Σ to stand for $\text{Red}(\Sigma)$, but when forming a reduction relation we restrict it to terms in $\text{Ter}(\Sigma)$, e.g., $\longrightarrow_\Sigma = \{\, (u,v) \mid u \longrightarrow_{\text{Red}(\Sigma)} v \text{ and } u, v \in \text{Ter}(\Sigma) \,\}$. The equational theory $=_\Sigma$ is similarly restricted to $\text{Ter}(\Sigma)$. This is the *only* way the restriction to terms in $\text{Ter}(\Sigma)$ affects the reduction relation.

[2] Previous definitions of a CRS required giving for a CRS Σ some fixed set S of function symbols and using all the terms that can be generated with those function symbols, i.e., setting $\text{Ter}(\Sigma) = \{\, u \mid u \in \text{Ter}, \text{Fun}(u) \subseteq S \,\}$ where $\text{Fun}(u) = \text{Ran}(\text{Tree}(u)) \cap \text{Fun}$. Further restricting the set of terms was called a *substructure* CRS by Klop [Klo80] and [KvOvR93]. Our definition makes this the default case, because in practice it is nearly always necessary to do so.

2.2.2 Restrictions on CRSs

Fully Extended A pattern s is *fully extended* [HP99, vR96, Def. 4.2.51] iff for any metavariable occurrence $Z(\vec{x})$ in s, the sequence $\vec{x}$ includes as many variables as possible, i.e., for each binding of some variable x whose scope includes the metavariable occurrence, x is one of $\vec{x}$. A reduction rule $s \to t$ is fully extended iff s is fully extended. A CRS Σ is *fully extended* iff each reduction rule $r \in \mathrm{Red}(\Sigma)$ is fully extended.

Convention 2.4. Throughout the rest of this article, reduction rules (and therefore also CRSs) are restricted to be fully extended. □

Constructor Systems A CRS Σ is a *constructor CRS* iff there exists a set $\mathrm{FCon}(\Sigma)$ of *constructors* and a set $\mathrm{FDes}(\Sigma)$ of *destructors*[3] such that both of the following conditions hold.

1. The constructors and destructors partition the function symbols, i.e., $\mathrm{FCon}(\Sigma) \cup \mathrm{FDes}(\Sigma) = \mathrm{Fun}$ and $\mathrm{FCon}(\Sigma) \cap \mathrm{FDes}(\Sigma) = \varnothing$.
2. For any rule $r \in \mathrm{Red}(\Sigma)$, the root function symbol of $\mathrm{LHS}(r)$ is a destructor and any other function symbol in $\mathrm{LHS}(r)$ is a constructor. Formally, if $r \in \mathrm{Red}(\Sigma)$ and $\mathrm{Tree}(\mathrm{LHS}(r))(p) = F$, then $p = \epsilon \Rightarrow F \in \mathrm{FDes}(\Sigma)$ and $p \neq \epsilon \Rightarrow F \in \mathrm{FCon}(\Sigma)$.

Left-Linear Rule A metaterm s is *linear* iff every metavariable in s occurs in s exactly once. A reduction rule $s \to t$ is *left-linear* iff s is linear. A set of reduction rules R is left-linear iff every reduction rule $r \in R$ is left-linear.

Ambiguous Rules Given metaterms s and t and position $p \in \mathrm{Int}(t)$, it is said that s *interferes with* t *at* p iff there exist valuations ν and ν' for respectively s and t and a term context C^p such that $C[\nu(s)] \equiv \nu'(t)$.[4] This interference is *at the root* iff $p = \epsilon$. Distinct reduction rules $s \to t$ and $s' \to t'$ are *ambiguous* (sometimes called *overlapping*) whenever s and s' interfere. A reduction rule $s \to t$ is ambiguous with itself whenever s interferes with itself provided the interference is not at the root. A set of reduction rules R is ambiguous iff there exists an ambiguous pair of rules $r, r' \in R$ (where r and r' may be the same rule).

Orthogonal CRS A CRS Σ is *orthogonal* (also called *regular*) iff the set of reduction rules $\mathrm{Red}(\Sigma)$ is left-linear and non-ambiguous.[5]

Theorem 2.5 (Confluence of Every Orthogonal CRS). *If Σ is orthogonal, $u \longrightarrow\!\!\!\!\!\rightarrow_\Sigma v_1$, and $u \longrightarrow\!\!\!\!\!\rightarrow_\Sigma v_2$, then there exists u' such that $v_1 \longrightarrow\!\!\!\!\!\rightarrow_\Sigma u'$ and $v_2 \longrightarrow\!\!\!\!\!\rightarrow_\Sigma u'$.* □

Proof. See [Klo80] or [KvOvR93]. □

3 Two Calculi and Their Representations as CRSs

This section presents two example programming language calculi, the λ_{v}-calculus [Plo75] and the λ^{CIL}-calculus [WDMT97]. The technology described in section 4 will be used to obtain definitions of evaluation for λ_{v} and λ^{CIL} and standardization theorems for λ_{v} and λ^{CIL}. Because the technology in section 4 requires orthogonal constructor CRSs, the CRSs Σ_{v} and Σ_{CIL} are defined in sections 3.1.1 and 3.2.1 and proven equivalent to λ_{v} and λ^{CIL} in sections 3.1.2 and 3.2.2. Although not the primary goal of this paper, this section makes an important contribution by giving examples of how to represent calculi as CRSs and to verify that the representations are faithful.

[3] These are sometimes called *functions*, but we avoid that terminology due to the ambiguity with "function symbol."

[4] This definition is equivalent to the definitions of [Klo80] and [KvOvR93] for metaterms which can be LHS's of reduction rules, but differs in how it is stated. Note that the definition of interference would be the same even without our restriction to fully extended reduction rules.

[5] The definition given here considers valuations producing terms that are outside of $\mathrm{Ter}(\Sigma)$ when determining the ambiguity of $\mathrm{Red}(\Sigma)$. The definition could be changed to avoid considering such garbage terms, but doing this without breaking something would make the definition very messy.

3.1 λ_{v}, the Call-By-Value Lambda Calculus

Syntax

$$
\begin{array}{rrcl}
x, y, z \in & \mathrm{Var} & & \\
C \in & \mathrm{Context}_{\lambda_{\mathrm{v}}} & ::= & \Box \mid x \mid C\ C \mid \lambda x.C \\
M, N, P \in & \mathrm{Term}_{\lambda_{\mathrm{v}}} & ::= & \{\, C \mid \Box \text{ does not occur in } C \,\} \\
V \in & \mathrm{Value}_{\lambda_{\mathrm{v}}} & ::= & x \mid \lambda x.M
\end{array}
$$

Reduction Relations

$$
\begin{array}{lll}
(\lambda x.M)\ V \rightsquigarrow_{\lambda_{\mathrm{v}}} M[x := V] & & \text{Redex/Contractum} \\
C[M] \longrightarrow_{\lambda_{\mathrm{v}}} C[M'] \quad \text{iff} \quad M \rightsquigarrow_{\lambda_{\mathrm{v}}} M' & & \text{One-step Reduction}
\end{array}
$$

Figure 1: The λ_{v}-calculus.

The first application of this paper will be the λ_{v}-calculus of Plotkin [Plo75]. The λ_{v}-calculus was developed to capture the essence of call-by-value evaluation. This subsection presents the λ_{v}-calculus and develops an equivalent orthogonal CRS Σ_{v} which is a constructor system. The equivalence together with the orthogonality of Σ_{v} yields confluence theorems for both Σ_{v} and the λ_{v}-calculus. In section 5.1 we use the results of the preceding sections to show that both Σ_{v} and the λ_{v}-calculus have a standardization property.

Figure 1 defines the λ_{v}-calculus. We adopt the usual conventions that the scope of a λ-binding extends as far to the right as possible and that application is left-associative. We use the standard notation, $M[x{:=}M']$, to denote the substitution of M' for free occurrences of x in M, renaming bound variables in M as necessary to avoid capturing free variables of M'. We identify terms that are α-convertible. We write $M \equiv M'$ to indicate equality on terms instead of $M = M'$ to remind the reader that α-equivalent terms are considered equal and to avoid confusion with the use of $M = M'$ to indicate convertibility in the calculus. Given that C is an n-hole context, the expression $C[\vec{M}^n]$ denotes the term that results from replacing the holes in C by $\vec{M}^n$ from left to right, possibly capturing free variables in the terms $\vec{M}^n$. We write $C_1 \equiv C_2$ iff C_1 and C_2 have exactly n holes and for all $\vec{M}^n$ it holds that $C_1[\vec{M}^n] \equiv C_2[\vec{M}^n]$. The symbol $\longrightarrow\!\!\!\!\!\rightarrow_{\lambda_{\mathrm{v}}}$ denotes the reflexive and transitive closure of the $\longrightarrow_{\lambda_{\mathrm{v}}}$ relation. Let δ range over λ_{v}-redexes, i.e., terms of the form $(\lambda x.M)\ V$. A λ_{v}-*redex occurrence* is a pair (C, δ) where C is a one-hole context and δ is a λ_{v}-redex. Let Δ range over λ_{v}-redex occurrences. Following the convention established for CRS reduction steps in section 2.2, the notation $M \xrightarrow{\Delta}_{\lambda_{\mathrm{v}}} N$ means that N is obtained from M in one step by the contraction of Δ.

3.1.1 The CRS Σ_{v}

Reduction Rule of Σ_{v}

$$
\begin{array}{ll}
r_{\beta_{\mathrm{v}}} & = @(\mathbf{val}(\lambda([x]Z_1(x))), \mathbf{val}(Z_2)) \to Z_1(Z_2) \\
\mathrm{Red}(\Sigma_{\mathrm{v}}) & = \{r_{\beta_{\mathrm{v}}}\}
\end{array}
$$

Translation Function from $\mathrm{Context}_{\lambda_{\mathrm{v}}}$

$$
\begin{array}{ll}
\mathcal{T}_{\mathrm{v}}(\Box) \equiv \Box & \mathcal{T}_{\mathrm{v}}(\lambda x.C) \equiv \mathbf{val}(\lambda([x]\mathcal{T}_{\mathrm{v}}(C))) \\
\mathcal{T}_{\mathrm{v}}(x) \equiv \mathbf{val}(x) & \mathcal{T}_{\mathrm{v}}(C_1\ C_2) \equiv @(\mathcal{T}_{\mathrm{v}}(C_1), \mathcal{T}_{\mathrm{v}}(C_2))
\end{array}
$$

Translation of Redex Occurrences

$$
\mathcal{T}_{\mathrm{v}}((C, \delta)) = (\mathcal{T}_{\mathrm{v}}(C)[\mathcal{T}_{\mathrm{v}}(\delta)].p, r_{\beta_{\mathrm{v}}}) \text{ where } \mathrm{Tree}(\mathcal{T}_{\mathrm{v}}(C))(p) = \Box
$$

Terms of Σ_{v}

$$
\mathrm{Ter}(\Sigma_{\mathrm{v}}) = \mathcal{T}_{\mathrm{v}}(\mathrm{Term}_{\lambda_{\mathrm{v}}})
$$

Figure 2: The CRS Σ_{v}.

Figure 2 defines the combinatory reduction system Σ_{v} which is intended to correspond to λ_{v}.

We begin by showing that Σ_{v} is a well defined and orthogonal CRS and that it is in good correspondence with λ_{v}.

Lemma 3.1.

1. *$\mathcal{T}_{\mathrm{v}}$ is a bijection between* $\mathrm{Context}_{\lambda_{\mathrm{v}}}$ *and* $\mathcal{T}_{\mathrm{v}}(\mathrm{Context}_{\lambda_{\mathrm{v}}})$.
2. *$\mathcal{T}_{\mathrm{v}} \downarrow \mathrm{Term}_{\lambda_{\mathrm{v}}}$ is a bijection between* $\mathrm{Term}_{\lambda_{\mathrm{v}}}$ *and* $\mathrm{Ter}(\Sigma_{\mathrm{v}})$. □

Proof.

1. That $\mathcal{T}_{\mathrm{v}}$ is injective when used with domain $\mathrm{Context}_{\lambda_{\mathrm{v}}}$ can be determined by inspecting its definition and observing that α-conversion works the same way in λ_{v} and Σ_{CIL}. The function $\mathcal{T}_{\mathrm{v}}$ is surjective on $\mathcal{T}_{\mathrm{v}}(\mathrm{Context}_{\lambda_{\mathrm{v}}})$ by definition.
2. Same. □

Lemma 3.2. *The translation of λ_{v}-redex occurrences is well defined.* □

Proof. Immediate for the definition. □

Lemma 3.3. *If $C \in \mathrm{Context}_{\lambda_{\mathrm{v}}}$ has one hole and $M \in \mathrm{Term}_{\lambda_{\mathrm{v}}}$, then $\mathcal{T}_{\mathrm{v}}(C)[\mathcal{T}_{\mathrm{v}}(M)] \equiv \mathcal{T}_{\mathrm{v}}(C[M])$.* □

Proof. By induction on C. □

Lemma 3.4 relates redex terms in λ_{v} to redex terms in Σ_{v}.

Lemma 3.4. *Let $s = \mathrm{LHS}(r_{\beta_{\mathrm{v}}})$ and let δ be a λ_{v}-redex term. Then there exists a valuation ν for s such that $\mathcal{T}_{\mathrm{v}}(\delta) \equiv \nu(s)$.* □

Proof. Suppose $\delta \equiv (\lambda x.M)\, V$. Then $\mathcal{T}_{\mathrm{v}}(\delta) \equiv @(\mathbf{val}(\lambda([x]\mathcal{T}_{\mathrm{v}}(M))), \mathbf{val}(v))$. Define the valuation $\nu = \{Z_1 \mapsto \mathcal{T}_{\mathrm{v}}(M)[x{:=}\hat{Z}_1], Z_2 \mapsto v\}$. It is simple to check that $\nu(s) \equiv \mathcal{T}_{\mathrm{v}}(\delta)$. □

Entities in Σ_{v} will now be related back to entities in λ_{v}. Lemma 3.7 addresses the well-definedness of $\mathcal{T}_{\mathrm{v}}^{-1}$. Lemma 3.8 relates redexes in Σ_{v} to the corresponding entities in λ_{v}. In this sense, it is the converse of lemma 3.4.

Definition 3.5 (Nice). In reasoning about λ_{v} and Σ_{v}, a preterm $w \in \mathrm{PTer}$ is *nice* iff $w \in \mathcal{T}_{\mathrm{v}}(\mathrm{Context}_{\lambda_{\mathrm{v}}})$. As derived notions, a term $u \in \mathrm{Ter}$ is nice iff $u \in \mathrm{Ter}(\Sigma_{\mathrm{v}})$ and a context $C \in \mathrm{Ctxt}$ is nice iff $C \in \mathcal{T}_{\mathrm{v}}(\mathrm{Context}_{\lambda_{\mathrm{v}}}) \cap \mathrm{Ctxt}$. □

Lemma 3.6.

1. *If $\mathbf{val}(\lambda([x]u))$ is nice, then u is nice.*
2. *If $@(u_1, u_2)$ is nice, then u_1 and u_2 are nice.* □

Proof. Immediate from the definition of $\mathcal{T}_{\mathrm{v}}$. □

Lemma 3.7.

1. *If term u and one-hole context C are nice, then $\mathcal{T}_{\mathrm{v}}^{-1}(C[u]) \equiv \mathcal{T}_{\mathrm{v}}^{-1}(C)[\mathcal{T}_{\mathrm{v}}^{-1}(u)]$.*
2. *For one-hole context C, if $C[@(u_1, u_2)]$ is nice, then C and $@(u_1, u_2)$ are nice.* □

Proof.

1. By induction on C using lemma 3.6.
2. By induction on C, using the definition of $\mathcal{T}_{\mathrm{v}}$. □

Lemma 3.8. *Let $s = \mathrm{LHS}(r_{\beta_{\mathrm{v}}})$ and let ν be a valuation for s such that $\nu(s)$ is nice. Then:*

1. *$\mathcal{T}_{\mathrm{v}}^{-1}(\nu(s))$ is a λ_{v}-redex, and*

2. $\nu(Z_1(x))$ *and* $\nu(\mathbf{val}(Z_2))$ *are nice.* □

Proof.

1. Similar to lemma 3.35 (1).
2. By several uses of lemma 3.6. □

Lemma 3.9. *If* $\nu(Z_1(x))$ *and* $\mathbf{val}(u)$ *are nice, then* $\nu(Z_1)[\hat{Z}_1{:=}u]$ *is nice.* □

Proof. From the first precondition it follows that there exists an s such that $\nu(Z_1(x)) \equiv s[\hat{Z}_1 := x]$ and that for any p if $\mathrm{Tree}(s)(p \cdot 1) = \hat{Z}_1$, then $\mathrm{Tree}(s)(p) = \mathbf{val}$. From the second precondition it follows that u is either a variable or a term of the form $\lambda([x]u')$ where u' is nice. The claim follows from these facts. □

Lemma 3.10. *Let* $s \to t = r_{\beta_\mathrm{v}}$*, let* ν *be a valuation for* s*, and let* $\nu(s)$ *be nice. Then* $\nu(t)$ *is nice.* □

Proof. First, observe that $\nu(t) \equiv \nu(Z_1(Z_2)) \equiv \nu(Z_1)[\hat{Z}_1 := \nu(Z_2)]$. Since $\nu(s)$ is nice, by lemma 3.8 (2), $\nu(Z_1(x))$ is nice and $\nu(\mathbf{val}(Z_2))$ is nice. The result then follows by lemma 3.9. □

Lemma 3.11 (Closure under Reduction).

$$\mathrm{Ter}\,(\Sigma_\mathrm{v}) = \mathrm{Ter}\,(\Sigma_\mathrm{v}) \cup \left\{\, v \mid u \in \mathrm{Ter}(\Sigma_\mathrm{v}), u \longrightarrow_{\mathrm{Red}(\Sigma_\mathrm{v})} v \,\right\}.$$ □

Proof. It suffices to show that if u is nice and $u \longrightarrow_{\mathrm{Red}(\Sigma_\mathrm{v})} v$, then v is nice. Suppose u is nice and $u \longrightarrow_{\mathrm{Red}(\Sigma_\mathrm{v})} v$. It must hold that $u \equiv C[\nu(s)]$ and $v \equiv C[\nu(t)]$ where $s \to t = r_{\beta_\mathrm{v}}$, $C \in \mathrm{Ctxt}$, and ν is a valuation for s. By lemma 3.7 (2), C and $\nu(s)$ are nice. Then $\nu(t)$ is nice by lemma 3.10. Finally, v is nice by lemma 3.7 (1). □

Corollary 3.12. Σ_v *is a CRS.* □

Lemma 3.13. Σ_v *is a constructor CRS with* $\mathrm{FCon}(\Sigma_\mathrm{v}) = \{\mathbf{val}, \lambda\}$ *and* $\mathrm{FDes}(\Sigma_\mathrm{v}) = \{@\}$. □

Proof. Obvious. □

Lemma 3.14 (Orthogonality of Σ_v). Σ_v *is an orthogonal CRS.* □

Proof. $\mathrm{Red}(\Sigma_\mathrm{v})$ is left-linear and unambiguous. □

Theorem 3.15 (Confluence of Σ_v). Σ_v *is confluent.* □

Proof. Theorem 2.5 and lemma 3.14. □

3.1.2 Correspondence between λ_v and Σ_v

This subsection will show a precise correspondence between λ_v and Σ_v. Specifically, it is proven here that the terms and reduction relation of Σ_v are isomorphic to the terms and reduction relation of λ_v, i.e., $(\mathrm{Term}_{\lambda_\mathrm{v}}, \longrightarrow_{\lambda_\mathrm{v}})$ and $(\mathrm{Ter}(\Sigma_\mathrm{v}), \longrightarrow_{\Sigma_\mathrm{v}})$ are isomorphic. The precise correspondence given here allows transferring results (e.g., confluence) obtained for Σ_v back to λ_v.

REMARK 3.16. When the edges of the rewrite relations are annotated with redex occurrences, there is no longer an isomorphism. Our method of identifying redex occurrences in λ_v allows many context/redex pairs to identify the same redex position. For example, the context/redex pairs $(\lambda x.\Box, (\lambda y.x)(\lambda y.x))$ and $(\lambda z.\Box, (\lambda y.z)(\lambda y.z))$ are distinct but identify the same redex position in the same term, $(\lambda x.(\lambda y.x)(\lambda y.x))$. □

Lemma 3.17. $\mathcal{T}_\mathrm{v}$ *is a surjection from* λ_v*-redex occurrences to* Σ_v*-redex occurrences.* □

Proof. Let $\Delta = (u.p, r_{\beta_\mathrm{v}})$ be a Σ_v-redex occurrence. It must be the case that $u \equiv \hat{C}[\nu(s)]$ where $s \equiv \mathrm{LHS}(r_{\beta_\mathrm{v}})$ for some context $\hat{C}$ and some valuation ν for s. By the definition of Σ_v-reduction, u is nice. Because $\nu(s)$ must have function symbol @ in outermost position, $\hat{C}$ and $\nu(s)$ are both nice by lemma 3.7 (2). Thus, there exists a $C \in \mathrm{Context}_{\lambda_\mathrm{v}}$ such that $C \equiv \mathcal{T}_\mathrm{v}^{-1}(\hat{C})$ and, by lemma 3.8, there exists a λ_v-redex $\delta \equiv \mathcal{T}_\mathrm{v}^{-1}(\nu(s))$. Then $\mathcal{T}_\mathrm{v}((C, \delta)) = \Delta$. □

Lemma 3.18 (Substitution). *Let* $\mathcal{T}_{\mathrm{v}}(V) \equiv \mathbf{val}(u)$. *Then* $\mathcal{T}_{\mathrm{v}}(M[x := V]) \equiv \mathcal{T}_{\mathrm{v}}(M)[x := u]$. □

Proof. By induction on M. See lemma 3.43 for a sample of the necessary reasoning. □

Lemma 3.19 (Redex Simulation). *Let* $s \to t = r_{\beta_{\mathrm{v}}}$. *Then* $\delta \rightsquigarrow_{\lambda_{\mathrm{v}}} P$ *iff there exists a valuation* ν *for* s *such that* $\mathcal{T}_{\mathrm{v}}(\delta) \equiv \nu(s)$ *and* $\mathcal{T}_{\mathrm{v}}(P) \equiv \nu(t)$. □

Proof. We prove the two directions separately.

⇒ Let $\delta \equiv (\lambda x.M)\, V$, let $P \equiv M[x := V]$, and let $\mathbf{val}(v) \equiv \mathcal{T}_{\mathrm{v}}(V)$. It is easy to calculate that $\mathcal{T}_{\mathrm{v}}(\delta) \equiv @(\mathbf{val}(\lambda([x]\mathcal{T}_{\mathrm{v}}(M))), \mathbf{val}(v))$. Using valuation $\nu = \{Z_1 \mapsto \mathcal{T}_{\mathrm{v}}(M)[x{:=}\hat{Z}_1], Z_2 \mapsto v\}$, it is easy to check that $\mathcal{T}_{\mathrm{v}}(\delta) \equiv \nu(s)$. For the RHS, $\nu(t) \equiv \mathcal{T}_{\mathrm{v}}(M)[x{:=}\hat{Z}_1][\hat{Z}_1{:=}v] \equiv \mathcal{T}_{\mathrm{v}}(M)[x{:=}v]$. Using lemma 3.18, it holds that $\nu(t) \equiv \mathcal{T}_{\mathrm{v}}(M)[x{:=}v] \equiv \mathcal{T}_{\mathrm{v}}(M[x{:=}V]) \equiv \mathcal{T}_{\mathrm{v}}(P)$, as required.

⇐ Suppose there exists a valuation $\nu = \{Z_1 \mapsto s', Z_2 \mapsto v\}$ such that $\mathcal{T}_{\mathrm{v}}(\delta) \equiv \nu(s)$ and $\mathcal{T}_{\mathrm{v}}(P) \equiv \nu(t)$. It must hold that $\nu(s) \equiv @(\mathbf{val}(\lambda([x]v')), \mathbf{val}(v))$ where $v' \equiv s'[\hat{Z}_1{:=}x]$ and $\nu(t) \equiv s'[\hat{Z}_1{:=}v] \equiv v'[x{:=}v]$. It must be the case that $\delta \equiv (\lambda x.M)\, V$ for some M and V such that $\mathcal{T}_{\mathrm{v}}(M) \equiv v'$ and $\mathcal{T}_{\mathrm{v}}(V) \equiv \mathbf{val}(v)$. By lemma 3.18, it holds that $\mathcal{T}_{\mathrm{v}}(M[x := V]) \equiv v'[x := v] \equiv \nu(t) \equiv \mathcal{T}_{\mathrm{v}}(P)$. By lemma 3.1 (2), it holds that $M[x := V] \equiv P$, so therefore $\delta \rightsquigarrow_{\lambda_{\mathrm{v}}} P$, as required. □

Lemma 3.20 (Reduction Simulation). $M \xrightarrow{\Delta}_{\lambda_{\mathrm{v}}} N$ *iff* $\mathcal{T}_{\mathrm{v}}(M) \xrightarrow{\mathcal{T}_{\mathrm{v}}(\Delta)}_{\Sigma_{\mathrm{v}}} \mathcal{T}_{\mathrm{v}}(N)$. □

Proof. Let $s \to t = r_{\beta_{\mathrm{v}}}$, let $\Delta = (C, \delta)$, let $u \equiv \mathcal{T}_{\mathrm{v}}(M)$, let $v \equiv \mathcal{T}_{\mathrm{v}}(N)$, let $\hat{C}^p \equiv \mathcal{T}_{\mathrm{v}}(C)$, and let $\Delta' = (u.p, r_{\beta_{\mathrm{v}}})$. We prove the two directions separately.

⇒ Suppose $M \xrightarrow{\Delta}_{\lambda_{\mathrm{v}}} N$. Thus, for some P it holds that $M \equiv C[\delta]$, $N \equiv C[P]$, and $\delta \rightsquigarrow_{\lambda_{\mathrm{v}}} P$. By lemma 3.19, there exists a valuation ν for s such that $\mathcal{T}_{\mathrm{v}}(\delta) \equiv \nu(s)$ and $\mathcal{T}_{\mathrm{v}}(P) \equiv \nu(t)$. By lemma 3.3 it holds that $u \equiv \hat{C}[\nu(s)]$ and $v \equiv \hat{C}[\nu(t)]$. Because $\nu(s) \equiv \mathcal{T}_{\mathrm{v}}(\delta)$, it holds that $\mathcal{T}_{\mathrm{v}}(\Delta) = (\hat{C}[\nu(s)].p, r_{\beta_{\mathrm{v}}}) = (u.p, r_{\beta_{\mathrm{v}}}) = \Delta'$. By the definition of Σ_{v}-reduction, $u \equiv \hat{C}[\nu(s)] \xrightarrow{\Delta'}_{\Sigma_{\mathrm{v}}} \hat{C}[\nu(t)] \equiv v$, which is the desired result.

⇐ Suppose $\mathcal{T}_{\mathrm{v}}(M) \xrightarrow{\mathcal{T}_{\mathrm{v}}(\Delta)}_{\Sigma_{\mathrm{v}}} \mathcal{T}_{\mathrm{v}}(N)$, i.e., $u \xrightarrow{\mathcal{T}_{\mathrm{v}}(\Delta)}_{\Sigma_{\mathrm{v}}} v$. Because it is supposed that $\mathcal{T}_{\mathrm{v}}(\Delta)$ is a well defined Σ_{v}-redex occurrence, it must be the case that there is some valuation ν for s such that $\mathcal{T}_{\mathrm{v}}(\Delta) = \mathcal{T}_{\mathrm{v}}((C, \delta)) = (\hat{C}[\nu(s)].p, r_{\beta_{\mathrm{v}}}) = (u.p, r_{\beta_{\mathrm{v}}}) = \Delta'$ where it must hold that $\mathcal{T}_{\mathrm{v}}(\delta) \equiv \nu(s)$ and that $u \equiv \hat{C}[\nu(s)]$. It must also hold that $v \equiv \hat{C}[\nu(t)]$. By lemma 3.10, $\nu(t)$ is nice. Let $P \equiv \mathcal{T}_{\mathrm{v}}^{-1}(\nu(t))$. By lemma 3.19 it holds that $\delta \rightsquigarrow_{\lambda_{\mathrm{v}}} P$. Because $\mathcal{T}_{\mathrm{v}}(M) \equiv u \equiv \hat{C}[\nu(s)] \equiv \mathcal{T}_{\mathrm{v}}(C)[\mathcal{T}_{\mathrm{v}}(\delta)]$, by lemma 3.7 (1) it holds that $M \equiv C[\delta]$. Similarly, because $\mathcal{T}_{\mathrm{v}}(N) \equiv v \equiv \hat{C}[\nu(t)] \equiv \mathcal{T}_{\mathrm{v}}(C)[\mathcal{T}_{\mathrm{v}}(P)]$, by lemma 3.7 (1) it holds that $N \equiv C[P]$. By definition, $M \xrightarrow{\Delta}_{\lambda_{\mathrm{v}}} N$, which is the desired result. □

Theorem 3.21 (Confluence of λ_{v}). λ_{v} *is confluent.* □

Proof. Theorem 3.15 and lemmas 3.20 and 3.17. □

3.2 $\lambda^{\mathbf{CIL}}$

Our second application is the $\lambda^{\mathrm{CIL}}_{\mathrm{ut}}$-calculus [WDMT97], an untyped version of the typed λ-calculus λ^{CIL}. The latter serves as a foundation for a compiler intermediate language [DMTW97]. In the remainder of this paper we will refer to the untyped version of the calculus simply as λ^{CIL}.

Figure 3 defines the λ^{CIL}-calculus. Like λ_{v}, the calculus has variables, abstractions, and applications. In addition, it includes constants, recursive terms, tuples, tuple projections, injections, and a case-dispatch form. Notation in this section is generally a straightforward adaptation of the corresponding notation presented in subsection 3.1. For example, in this section the symbol $\longrightarrow\!\!\!\!\!\rightarrow_{\lambda^{\mathrm{CIL}}}$ denotes the reflexive and transitive closure of the $\longrightarrow_{\lambda^{\mathrm{CIL}}}$ relation, the notation $M[x := N]$ denotes the capture-free substitution for λ^{CIL}-terms. The reader will also note that we are recycling metavariables M, N, V, etc., all of which stand for λ^{CIL} entities when appropriate.

Syntax

$c \in \mathrm{Const}_{\lambda^{\mathrm{CIL}}}$

$C \in \mathrm{Context}_{\lambda^{\mathrm{CIL}}} ::= \Box \mid c \mid x \mid \mathbf{rec}\ x.C \mid \lambda x.C \mid C_1 \mathbin{@} C_2 \mid \times(C_1, \ldots, C_n) \mid \pi_i\ C \mid \mathbf{in}_i\ C \mid \mathbf{case}\ C\ \mathbf{bind}\ x\ \mathbf{in}\ C_1, \ldots, C_m$
where $n \geq 0$ and $i, m \geq 1$

$M, N \in \mathrm{Term}_{\lambda^{\mathrm{CIL}}} = \{\, C \mid \Box \text{ does not occur in } C\}$

$V \in \mathrm{Value}_{\lambda^{\mathrm{CIL}}} ::= c \mid \lambda x.M \mid \times(V_1, \ldots, V_n) \mid \mathbf{in}_i\ V$
where $n \geq 0$ and $i \geq 1$

Redex/Contractum Relation

$(\lambda x.M) \mathbin{@} V$	$\rightsquigarrow_{\lambda^{\mathrm{CIL}}}$	$M[x{:=}V]$	
$\pi_i\ \times(V_1, \ldots, V_n)$	$\rightsquigarrow_{\lambda^{\mathrm{CIL}}}$	V_i	if $1 \leq i \leq n$
$\mathbf{case}\ (\mathbf{in}_i\ V)\ \mathbf{bind}\ x\ \mathbf{in}\ M_1, \ldots, M_n$	$\rightsquigarrow_{\lambda^{\mathrm{CIL}}}$	$M_i[x{:=}V]$	if $1 \leq i \leq n$
$\mathbf{rec}\ x.M$	$\rightsquigarrow_{\lambda^{\mathrm{CIL}}}$	$M[x{:=}(\mathbf{rec}\ x.M)]$	

One-Step Reduction Relation

$C[M] \longrightarrow_{\lambda^{\mathrm{CIL}}} C[M']$ iff $M \rightsquigarrow_{\lambda^{\mathrm{CIL}}} M'$ One-Step Reduction

Figure 3: The λ^{CIL}-calculus.

3.2.1 The CRS Σ_{CIL}

Figure 4 defines the CRS Σ_{CIL}. The members of $\mathrm{ValPatt}_{\mathrm{CIL}}$ are the *value patterns*. These will be shown to be in good correspondence with the set $\mathrm{Value}_{\lambda^{\mathrm{CIL}}}$. The value patterns are used as templates in the definition of the infinite set of reduction rules $\mathrm{Red}(\Sigma_{\mathrm{CIL}})$.

REMARK 3.22. The set $\mathrm{Red}(\Sigma_{\mathrm{CIL}})$ is defined as $\mathrm{LeastUpToRenaming}(S)$ for a larger set of rules S to eliminate extra rules that differ only in the names of metavariables. This is necessary in order for Σ_{CIL} to qualify as orthogonal and it also simplifies various proofs. For example, without this filtering, the rules $@(\lambda([x]Z_1(x)), c) \to Z_1(c)$ and $@(\lambda([x]Z_2(x)), c) \to Z_2(c)$ where $Z_1 \neq Z_2$ would both belong to $\mathrm{Red}(\Sigma_{\mathrm{CIL}})$. □

REMARK 3.23. Like Σ_{v}, the CRS Σ_{CIL} represents a call-by-value system. However, the two systems effect the call-by-value disciplines of λ_{v} and λ^{CIL} differently. In Σ_{v}, the call-by-value discipline is enforced by wrapping variables and abstractions with $\mathbf{val}(\,\cdot\,)$. This technique is not used here to derive a CRS corresponding to λ^{CIL} because the status of a CRS term would be able to change from non-value to value by a deeply nested reduction step with the result that the **val** markings would become inaccurate. For example, although $u \equiv \times_1(@(\lambda([x]x), c))$ is not a value, $u \longrightarrow_{\Sigma_{\mathrm{CIL}}} v \equiv \times_1(c)$ and v is a value.

It is possible to construct a CRS that effects call-by-value reduction by using **val** and **notval** symbols to track the value status of subterms and by using special rules to propagate changes in these symbols (see, e.g., [WDMT0X]). However, the constructed CRS will not be isomorphic to λ^{CIL}, because it will have extra steps to propagate these symbols. To avoid this issue, the CRS Σ_{CIL} has been designed with an infinite set of reduction rules generated according to the definition of the value patterns. This technique does give the desired isomorphism between λ^{CIL} and Σ_{CIL}. □

As in section 3.1, we will begin by showing that Σ_{CIL} is a well-defined and orthogonal CRS and that it is in good correspondence with λ^{CIL}. Note that Σ_{CIL} is a constructor system.

Lemma 3.24. *Every value pattern $s \in \mathrm{ValPatt}_{\mathrm{CIL}}$ is linear.* □

Proof. By induction on the size of s, relying on the side condition $\mathrm{DMV}(\{s_1, \ldots, s_n\})$ in the tuple case of the definition of $\mathrm{ValPatt}_{\mathrm{CIL}}$. □

Lemma 3.25.

1. *$\mathcal{T}_{\mathrm{CIL}}$ is a bijection between $\mathrm{Context}_{\lambda^{\mathrm{CIL}}}$ and $\mathcal{T}_{\mathrm{CIL}}(\mathrm{Context}_{\lambda^{\mathrm{CIL}}})$.*

Reduction Rules of Σ_{CIL}

$$\mathrm{Const}_{\Sigma_{\mathrm{CIL}}} = \{\, c^{(0)} \mid c \in \mathrm{Const}_{\lambda^{\mathrm{CIL}}} \,\}$$

$$\mathrm{ValPatt}_{\mathrm{CIL}} = \begin{pmatrix} \mathrm{Const}_{\Sigma_{\mathrm{CIL}}} \\ \cup\, \{\, \lambda([x]Z(x)) \mid Z \in \mathrm{MVar} \,\} \\ \cup\, \{\, \times_n(s_1, \ldots, s_n) \mid \{s_1, \ldots, s_n\} \subseteq \mathrm{ValPatt}_{\mathrm{CIL}},\ \mathrm{DMV}(\{s_1, \ldots, s_n\}) \,\} \\ \cup\, \{\, \mathbf{in}_i(s) \mid s \in \mathrm{ValPatt}_{\mathrm{CIL}},\ 1 \le i \,\} \end{pmatrix}$$

$$\mathrm{PreRed}_{\Sigma_{\mathrm{CIL}}} = \begin{pmatrix} \{\, @(\lambda([x]Z(x)), s) \to Z(s) \mid s \in \mathrm{ValPatt}_{\mathrm{CIL}}, Z \notin \mathrm{MV}(s) \,\} \\ \cup\, \{\, \pi_i(\times_n(s_1, \ldots, s_n)) \to s_i \\ \quad \mid \{s_1, \ldots, s_n\} \subseteq \mathrm{ValPatt}_{\mathrm{CIL}},\ \mathrm{DMV}(\{s_1, \ldots, s_n\}),\ 1 \le i \le n \,\} \\ \cup\, \{\, \mathbf{case}_{n+1}(\mathbf{in}_i(s), [x]Z_1(x), \ldots, [x]Z_n(x)) \to Z_i(s) \\ \quad \mid s \in \mathrm{ValPatt}_{\mathrm{CIL}}, \mathrm{DMV}(\{s, Z_1(x), \ldots, Z_n(x)\}), 1 \le i \le n \,\} \\ \cup\, \{\mu([x]Z(x)) \to Z(\mu([x]Z(x)))\} \end{pmatrix}$$

$$\mathrm{Red}(\Sigma_{\mathrm{CIL}}) = \mathrm{LeastUpToRenaming}(\mathrm{PreRed}_{\Sigma_{\mathrm{CIL}}})$$

Translation Function from $\mathrm{Context}_{\lambda^{\mathrm{CIL}}}$

$$\begin{array}{llll} \mathcal{T}_{\mathrm{CIL}}(\Box) & \equiv \Box & \mathcal{T}_{\mathrm{CIL}}(c) & \equiv c \\ \mathcal{T}_{\mathrm{CIL}}(x) & \equiv x & \mathcal{T}_{\mathrm{CIL}}(\mathbf{rec}\ x.C) & \equiv \mu([x]\mathcal{T}_{\mathrm{CIL}}(C)) \\ \mathcal{T}_{\mathrm{CIL}}(\lambda x.C) & \equiv \lambda([x]\mathcal{T}_{\mathrm{CIL}}(C)) & \mathcal{T}_{\mathrm{CIL}}(C_1\ @\ C_2) & \equiv @(\mathcal{T}_{\mathrm{CIL}}(C_1), \mathcal{T}_{\mathrm{CIL}}(C_2)) \\ \mathcal{T}_{\mathrm{CIL}}(\pi_i\ C) & \equiv \pi_i(\mathcal{T}_{\mathrm{CIL}}(C)) & \mathcal{T}_{\mathrm{CIL}}(\mathbf{in}_i\ C) & \equiv \mathbf{in}_i(\mathcal{T}_{\mathrm{CIL}}(C)) \end{array}$$

$$\mathcal{T}_{\mathrm{CIL}}(\times(C_1, \ldots, C_n)) \equiv \times_n(\mathcal{T}_{\mathrm{CIL}}(C_1), \ldots, \mathcal{T}_{\mathrm{CIL}}(C_n))$$

$$\mathcal{T}_{\mathrm{CIL}}(\mathbf{case}\ C\ \mathbf{bind}\ x\ \mathbf{in}\ C_1, \ldots, C_n) \equiv \mathbf{case}_{n+1}(\mathcal{T}_{\mathrm{CIL}}(C), [x]\mathcal{T}_{\mathrm{CIL}}(C_1), \ldots, [x]\mathcal{T}_{\mathrm{CIL}}(C_n))$$

Translation of Redex Occurrences

$$\mathcal{T}_{\mathrm{CIL}}\,((C, \delta)) = (\hat{C}[\nu(s)].p, s \to t)$$

where $\mathcal{T}_{\mathrm{CIL}}(C) \equiv \hat{C}^p$, $s \to t \in \mathrm{Red}(\Sigma_{\mathrm{CIL}})$, ν is a valuation for s, and $\mathcal{T}_{\mathrm{CIL}}(\delta) \equiv \nu(s)$.

Terms of Σ_{CIL}

$$\mathrm{Ter}\,(\Sigma_{\mathrm{CIL}}) = \mathcal{T}_{\mathrm{CIL}}\,(\mathrm{Term}_{\lambda^{\mathrm{CIL}}})$$

Figure 4: The CRS Σ_{CIL}.

2. $\mathcal{T}_{\text{CIL}} \downarrow \text{Term}_{\lambda^{\text{CIL}}}$ *is a bijection between* $\text{Term}_{\lambda^{\text{CIL}}}$ *and* $\text{Ter}(\Sigma_{\text{CIL}})$. □

Proof.

1. See proof of lemma 3.1.
2. Same. □

Lemma 3.26. *If C has one hole and $M \in \text{Term}_{\lambda^{\text{CIL}}}$ then $\mathcal{T}_{\text{CIL}}(C)[\mathcal{T}_{\text{CIL}}(M)] \equiv \mathcal{T}_{\text{CIL}}(C[M])$.* □

Proof. By induction on C. □

Lemmas 3.27 and 3.28, (respectively) relate values and redexes in λ^{CIL} to the corresponding entities in Σ_{CIL}.

Lemma 3.27. *If $V \in \text{Value}_{\lambda^{\text{CIL}}}$, then there exists an $s \in \text{ValPatt}_{\text{CIL}}$ and valuation ν for s such that $\mathcal{T}_{\text{CIL}}(V) \equiv \nu(s)$ and $\text{DomDef}(\nu) = \text{MV}(s)$.* □

Proof. The claim is proven by induction on the size of V and then by cases on the shape of V.

1. If $V \equiv c$, then take $s \equiv c^{(0)}$ and $\nu = \varnothing$.
2. If $V \equiv (\lambda x.M)$, then take $s \equiv \lambda([x]Z(x))$ and $\nu = \{Z \mapsto \mathcal{T}_{\text{CIL}}(M)[x := \hat{Z}_1]\}$.
3. Suppose $V \equiv \times(V_1, \ldots, V_n)$. Then, by the induction hypothesis, for $1 \leq i \leq n$ there exist $s_i \in \text{ValPatt}_{\text{CIL}}$ and ν_i such that ν_i is a valuation for s_i and $\mathcal{T}_{\text{CIL}}(V_i) \equiv \nu_i(s_i)$. Let $\vec{R}^n$ and $\vec{s'}^n$ be metavariable renamings and metaterms such that $\text{DMV}(\{s'_1, \ldots, s'_n\})$ and for $1 \leq i \leq n$ it holds that $s'_i \equiv R_i(s_i)$. Observe that $s'_i \in \text{ValPatt}_{\text{CIL}}$ for $1 \leq i \leq n$. Let $\nu'_i = \nu_i \circ R_i^{-1}$ for $1 \leq i \leq n$. Then take $s \equiv \times_n(s'_1, \ldots, s'_n)$ and $\nu = \bigcup_{1 \leq i \leq n} \nu'_i$.
4. Suppose $V \equiv \mathbf{in}_i\ V'$. Then by the induction hypothesis there exist $s' \in \text{ValPatt}_{\text{CIL}}$ and ν such that $\mathcal{T}_{\text{CIL}}(V') \equiv \nu(s')$. Take $s \equiv \mathbf{in}_i(s')$. □

Lemma 3.28. *If δ is a λ^{CIL}-redex, then there exists a unique rule $s' \to t' \in \text{Red}(\Sigma_{\text{CIL}})$ and a valuation ν' for s' such that $\mathcal{T}_{\text{CIL}}(\delta) \equiv \nu'(s')$ and $\text{DomDef}(\nu') = \text{MV}(s')$.* □

Proof. We consider each λ^{CIL}-redex/contractum rule in figure 3. For each rule, we will compute an $r = s \to t \in \text{PreRed}_{\Sigma_{\text{CIL}}}$ and a valuation ν for s such that $\mathcal{T}_{\text{CIL}}(\delta) \equiv \nu(s)$ and $\text{DomDef}(\nu) = \text{MV}(s)$. From each of these, the desired result is the rule $r' = \text{LeastUpToRenaming}(r)$ and the valuation $\nu' = \nu \circ R$ where R is a metavariable renaming such that $R(r') = r$.

1. Suppose $\delta \equiv (\lambda x.M)\ @\ V$. Then by lemma 3.27, there exist $s' \in \text{ValPatt}_{\text{CIL}}$ and valuation ν' for s' such that $\mathcal{T}_{\text{CIL}}(V) \equiv \nu'(s')$ and $\text{DomDef}(\nu') = \text{MV}(s')$. Let $Z \in \text{MVar}$ such that $Z \notin \text{MV}(s')$. Then take $s \to t = @(\lambda([x]Z(x)), s') \to Z(s')$ and $\nu = \nu'[Z \mapsto \mathcal{T}_{\text{CIL}}(M)[x := \hat{Z}_1]]$.
2. Suppose $\delta \equiv \pi_i\ \times(V_1, \ldots, V_n)$. Then because $\times(V_1, \ldots, V_n) \in \text{Value}_{\lambda^{\text{CIL}}}$, by lemma 3.27, there exists a valuation ν' and $s' \in \text{ValPatt}_{\text{CIL}}$ such that $\nu'(s') \equiv \mathcal{T}_{\text{CIL}}(\times_n(V_1, \ldots, V_n))$ and $\text{DomDef}(\nu') = \text{MV}(s')$. It must be the case that $s' \equiv \times_n(s'_1, \ldots, s'_n)$ for some $s'_1, \ldots, s'_n$. Then take $\nu = \nu'$ and $s \to t = \pi_i(\times_n(s'_1, \ldots, s'_n)) \to s'_i$.
3. Suppose $\delta \equiv \mathbf{case}\ (\mathbf{in}_i\ V)\ \mathbf{bind}\ x\ \mathbf{in}\ M_1, \ldots, M_n$. Then by lemma 3.27, there exist $s' \in \text{ValPatt}_{\text{CIL}}$ and valuation ν' for s' such that $\mathcal{T}_{\text{CIL}}(V) \equiv \nu'(s')$ and $\text{DomDef}(\nu') = \text{MV}(s')$. Choose $Z_1, \ldots, Z_n$ such that $\text{DMV}(\{Z_1(x), \ldots, Z_n(x), s'\})$. Then let $\nu = \nu' \cup \{\, Z_i \mapsto \mathcal{T}_{\text{CIL}}(M_i)[x := \hat{Z}_i] \mid 1 \leq i \leq n \,\}$ and let $s \to t = \mathbf{case}_{n+1}(\mathbf{in}_i(s'), [x]Z_1(x), \ldots, [x]Z_n(x)) \to Z_i(s')$.
4. If $\delta \equiv \mathbf{rec}\ x.M$, then let $s \to t = \mu([x]Z(x)) \to Z(\mu([x]Z(x)))$ and $\nu = \{Z \mapsto \mathcal{T}_{\text{CIL}}(M)[x := \hat{Z}_1]\}$. □

Entities in Σ_{CIL} will now be related back to entities in λ^{CIL}. Lemmas 3.31 through 3.33 address the well-definedness of $\mathcal{T}^{-1}_{\text{CIL}}$. Lemmas 3.34 and 3.35 (respectively) relate values and redexes in Σ_{CIL} to the corresponding entities in λ^{CIL}. In this sense, they are converses of lemmas 3.27 and 3.28.

Definition 3.29 (Nice). When reasoning about λ^{CIL} and Σ_{CIL}, a preterm $w \in \text{PTer}$ is *nice* iff $w \in \mathcal{T}_{\text{CIL}}(\text{Context}_{\lambda^{\text{CIL}}})$. As derived notions, a term $u \in \text{Ter}$ is nice iff $u \in \text{Ter}(\Sigma_{\text{CIL}})$ and a context $C \in \text{Ctxt}$ is nice iff $C \in \mathcal{T}_{\text{CIL}}(\text{Context}_{\lambda^{\text{CIL}}}) \cap \text{Ctxt}$. A valuation ν is *nice* iff for every $Z^{(n)} \in \text{DomDef}(\nu)$, there exists some $\{x_1, \ldots, x_n\} \subseteq \text{Var}$ such that $\nu(Z(x_1, \ldots, x_n))$ is nice. □

Lemma 3.30. *Let C and C' be nice one-hole contexts. Then $C[C']$ is nice.* □

Proof. Immediate from the definition of $\mathcal{T}_{\text{CIL}}$. □

Lemma 3.31.

1. *If any one of $\lambda([x]w)$, $\pi_i(w)$, $\mathbf{in}_i(w)$, or $\mu([x]w)$ is nice, then w is nice.*
2. *If $@(w_1, w_2)$ is nice, then w_1 and w_2 are nice.*
3. *If either $\times_n(w_1, \ldots, w_n)$ or $\mathbf{case}_n(w_1, [x]w_2, \ldots, [x]w_n)$ is nice, then w_i is nice for $1 \leq i \leq n$.* □

Proof. Immediate from inspection of the definition of $\mathcal{T}_{\text{CIL}}$. □

Lemma 3.32. *If term u and one-hole context C are nice, then $\mathcal{T}^{-1}_{\text{CIL}}(C)[\mathcal{T}^{-1}_{\text{CIL}}(u)] \equiv \mathcal{T}^{-1}_{\text{CIL}}(C[u])$.* □

Proof. By induction on C using lemma 3.31. □

Lemma 3.33. *For one-hole context $C \in \text{Ctxt}$ and $n \geq 0$, if $C[F(v_1, \ldots, v_n)]$ is nice, then both C and $F(v_1, \ldots, v_n)$ are nice.* □

Proof. Let $u \equiv F(v_1, \ldots, v_n)$. Assume that C is a one-hole context and $C[u]$ is nice. Proceed by induction on the size of C and then by cases on the shape of C.

Suppose $C \equiv \Box$. Then $C \equiv \mathcal{T}_{\text{CIL}}(\Box)$, so C is nice. Also, $C[u] \equiv u$, so u is nice because $C[u]$ is nice.

Suppose $C \not\equiv \Box$. It can be checked that $C \equiv \tilde{C}[\hat{C}]$ for some one-hole contexts $\tilde{C}, \hat{C} \in \text{Ctxt}$ where $\tilde{C}$ is of one of the following shapes, for some numbers $m, m', i \geq 0$ and metaterms s, s_1, …, s_m, s'_1, …, $s'_{m'}$:

$$\lambda([x]\Box), \quad \mu([x]\Box), \quad @(\Box, s_1), \quad @(s_1, \Box), \quad \pi_i(\Box), \quad \mathbf{in}_i(\Box),$$
$$\times_{m+m'+1}(s_1, \ldots, s_m, \Box, s'_1, \ldots, s'_{m'}), \quad \mathbf{case}_{m+m'+2}(s, [x]s_1, \ldots, [x]s_m, [x]\Box, [x]s'_1, \ldots, [x]s'_{m'}),$$
$$\mathbf{case}_{m+1}(\Box, [x]s_1, \ldots, [x]s_m)$$

In all cases, it can be verified that $\tilde{C}$ is nice, using lemma 3.31 to show that the subterms s, $\vec{s}$ and $\vec{s}'$ are nice. Also using lemma 3.31, it holds that $\hat{C}[u]$ is nice. By induction hypothesis, both $\hat{C}$ and u are nice. Because both $\tilde{C}$ and $\hat{C}$ are nice, $C \equiv \tilde{C}[\hat{C}]$ is nice by lemma 3.30. □

Lemma 3.34. *Let $s \in \text{ValPatt}_{\text{CIL}}$ and let ν be a valuation for s such that $\nu(s)$ is nice. Then:*

1. *$\mathcal{T}^{-1}_{\text{CIL}}(\nu(s)) \in \text{Value}_{\lambda^{\text{CIL}}}$, and*
2. *$\nu \downarrow \text{MV}(s)$ is nice.* □

Proof.

1. That $\mathcal{T}^{-1}_{\text{CIL}}(\nu(s)) \in \text{Value}_{\lambda^{\text{CIL}}}$ can be shown by induction on the size of s using the fact that, for every value pattern s, the leftmost symbol of $\nu(s)$ is the same as the leftmost symbol of s and by inspecting the definition of $\mathcal{T}_{\text{CIL}}$.
2. By induction on the size of s. If $s \equiv c$ or $s \equiv \lambda([x]Z(x))$ for some $Z \in \text{MVar}$, then the result is immediate. If $s = \mathbf{in}_i(s')$, then the result follows by induction hypothesis. Suppose $s \equiv \times_n(s_1, \ldots, s_n)$ with $\text{DMV}(\{s_1, \ldots, s_n\})$. Let $\nu_i = \nu \downarrow \text{MV}(s_i)$ for $1 \leq i \leq n$. By induction hypothesis, ν_i is nice for $1 \leq i \leq n$. This is sufficient, because $\nu \downarrow \text{MV}(s) = \bigcup_{1 \leq i \leq n} \nu_i$. □

Lemma 3.35. *Let $s \to t \in \text{Red}(\Sigma_{\text{CIL}})$ and let ν be a valuation for s such that $\nu(s)$ is nice. Then:*

1. *$\mathcal{T}^{-1}_{\text{CIL}}(\nu(s))$ is a λ^{CIL}-redex, and*
2. *$\nu \downarrow \text{MV}(s)$ is nice.* □

Proof. Suppose $s \to t \in \mathrm{Red}(\Sigma_{\mathrm{CIL}})$, ν is a valuation for s, and $\nu(s)$ is nice. By a case analysis on s. One case is shown and the other cases are handled similarly.

Suppose $s \equiv @(\lambda([x]Z(x)), s')$ where $s' \in \mathrm{ValPatt}_{\mathrm{CIL}}$ and $Z \notin \mathrm{MV}(s')$. It is easy to calculate that $\nu(s) \equiv @(\lambda([x]\nu(Z(x))), \nu(s'))$ where $x \notin \mathrm{FV}(\nu)$ is obtained by α-conversion. By lemma 3.31, it holds that $\nu(s')$, $\lambda([x]\nu(Z(x)))$, and $\nu(Z(x))$ are nice. Let $M \equiv \mathcal{T}_{\mathrm{CIL}}^{-1}(\nu(Z(x)))$. By lemma 3.34 (1), there exists a $V \in \mathrm{Value}_{\lambda^{\mathrm{CIL}}}$ such that $V \equiv \mathcal{T}_{\mathrm{CIL}}^{-1}(\nu(s'))$. Let $N \equiv (\lambda x.M)\ @\ V$. It is easy to calculate that $\mathcal{T}_{\mathrm{CIL}}(N) \equiv \nu(s)$. By the definition of $\rightsquigarrow_{\lambda^{\mathrm{CIL}}}$, it is clear that N is a λ^{CIL}-redex, proving part 1 of the claim. By lemma 3.34 (2), $\nu \downarrow \mathrm{MV}(s')$ is nice. Because $\nu(Z(x))$ is also nice, $\nu \downarrow \mathrm{MV}(s)$ is nice, proving part 2 of the claim. □

Lemma 3.36. *Let $s \to t \in \mathrm{Red}(\Sigma_{\mathrm{CIL}})$ and let $\nu \downarrow \mathrm{MV}(s)$ be nice. Then $\nu(t)$ is nice.* □

Proof. By a case analysis of the rules in $\mathrm{Red}(\Sigma_{\mathrm{CIL}})$. □

Lemma 3.37 (Closure under Reduction).

$$\mathrm{Ter}\,(\Sigma_{\mathrm{CIL}}) = \mathrm{Ter}\,(\Sigma_{\mathrm{CIL}}) \cup \left\{\, v \mid u \in \mathrm{Ter}(\Sigma_{\mathrm{CIL}}), u \longrightarrow_{\mathrm{Red}(\Sigma_{\mathrm{CIL}})} v \,\right\}. \qquad \square$$

Proof. It suffices to show that if u is nice and $u \xrightarrow{\Delta}_{\mathrm{Red}(\Sigma_{\mathrm{CIL}})} v$, then v is nice. Suppose u is nice and $u \xrightarrow{\Delta}_{\mathrm{Red}(\Sigma_{\mathrm{CIL}})} v$. It must hold that $u \equiv C[\nu(s)]$ and $v \equiv C[\nu(t)]$ where $s \to t \in \mathrm{Red}(\Sigma_{\mathrm{CIL}})$, $C \in \mathrm{Ctxt}$, and ν is a valuation for s. By lemma 3.33, C and $\nu(s)$ are nice. Then $\nu \downarrow \mathrm{MV}(s)$ is nice by lemma 3.35 (2). Then $\nu(t)$ is nice by lemma 3.36. Finally, v is nice by lemma 3.32, □

Corollary 3.38. *Σ_{CIL} is a CRS.* □

Lemma 3.39. *Σ_{CIL} is a constructor CRS with* $\mathrm{FCon}(\Sigma_{\mathrm{CIL}}) = \{\lambda, \times_n, \mathbf{in_i}\} \cup \mathrm{Const}_{\Sigma_{\mathrm{CIL}}}$ *and* $\mathrm{FDes}(\Sigma_{\mathrm{CIL}}) = \{@, \pi_i, \mathbf{case}_n, \mu\}$. □

Proof. Obvious. □

Lemma 3.40 (Orthogonality of Σ_{CIL}). *Σ_{CIL} is an orthogonal CRS.* □

Proof. Left-linearity of Σ_{CIL} follows from lemma 3.24 and by inspection of the definition of $\mathrm{Red}(\Sigma_{\mathrm{CIL}})$. That Σ_{CIL} is unambiguous follows from the fact that $\mathrm{Red}(\Sigma_{\mathrm{CIL}}) = \mathrm{LeastUpToRenaming}(\mathrm{PreRed}_{\Sigma_{\mathrm{CIL}}})$ and by inspection of the definition of $\mathrm{Red}(\Sigma_{\mathrm{CIL}})$. □

Theorem 3.41 (Confluence of Σ_{CIL}). *Σ_{CIL} is confluent.* □

Proof. Theorem 2.5 and lemma 3.40. □

3.2.2 Correspondence between λ^{CIL} and Σ_{CIL}

Lemma 3.42. *$\mathcal{T}_{\mathrm{CIL}}$ is a surjection from λ^{CIL}-redex occurrences to Σ_{CIL}-redex occurrences.* □

Proof. Let $\Delta = (u.p, s \to t)$ be a Σ_{CIL}-redex occurrence, where $s \to t \in \mathrm{Red}(\Sigma_{\mathrm{CIL}})$. By definition, this requires that u is nice and that $u \equiv \hat{C}[\nu(s)]$ for some $\hat{C}^p$ and some valuation ν for s. Because all redex terms are of the form $F(u_1, \ldots, u_n)$ where $n \geq 0$, by lemma 3.33 it holds that $\hat{C}$ and $\nu(s)$ are nice. Then by lemma 3.35 (1), there exists a λ^{CIL}-redex $\delta \equiv \mathcal{T}_{\mathrm{CIL}}^{-1}(\nu(s))$. By lemma 3.25 (1), there exists a $C \in \mathrm{Context}_{\lambda^{\mathrm{CIL}}}$ such that $C \equiv \mathcal{T}_{\mathrm{CIL}}^{-1}(\hat{C}^p)$. Then $\mathcal{T}_{\mathrm{CIL}}(C)[\mathcal{T}_{\mathrm{CIL}}(\delta)] \equiv \mathcal{T}_{\mathrm{CIL}}(C[\delta])$ by lemma 3.26. Then $\mathcal{T}_{\mathrm{CIL}}((C, \delta)) \equiv \Delta$. □

Lemma 3.43 (Substitution). $\mathcal{T}_{\mathrm{CIL}}(M[x := N]) \equiv \mathcal{T}_{\mathrm{CIL}}(M)[x := \mathcal{T}_{\mathrm{CIL}}(N)]$. □

Proof. Let $u \equiv \mathcal{T}_{\mathrm{CIL}}(M[x := N])$ and let $v \equiv \mathcal{T}_{\mathrm{CIL}}(M)[x := \mathcal{T}_{\mathrm{CIL}}(N)]$. We must prove that $u \equiv v$. The proof proceeds by induction on M and then by cases on the shape of M. If $M \equiv c$, then $u \equiv c \equiv v$. If

$M \equiv y \neq x$, then $u \equiv y \equiv v$. If $M \equiv x$, then $u \equiv \mathcal{T}_{\text{CIL}}(N) \equiv v$. Suppose $M \equiv (\lambda y.M')$. By α-conversion, let $y \notin \{x\} \cup \text{FV}(N)$. Thus, $y \notin \{x\} \cup \text{FV}(\mathcal{T}_{\text{CIL}}(N))$. Then

$$\begin{array}{rl} & u \\ & \equiv \mathcal{T}_{\text{CIL}}((\lambda y.M')[x{:=}N]) \\ & \equiv \mathcal{T}_{\text{CIL}}(\lambda y.(M'[x{:=}N])) \\ & \equiv \lambda([y]\mathcal{T}_{\text{CIL}}(M'[x{:=}N])) \\ \text{(by the induction hypothesis)} & \equiv \lambda([y]\mathcal{T}_{\text{CIL}}(M')[x{:=}\mathcal{T}_{\text{CIL}}(N)]) \\ & \equiv \lambda([y]\mathcal{T}_{\text{CIL}}(M'))[x{:=}\mathcal{T}_{\text{CIL}}(N)] \\ & \equiv \mathcal{T}_{\text{CIL}}(\lambda y.M')[x{:=}\mathcal{T}_{\text{CIL}}(N)] \\ & \equiv v \end{array}$$

The other cases are handled similarly. □

Lemma 3.44 (Redex Simulation). *The statement $\delta \rightsquigarrow_{\lambda^{\text{CIL}}} P$ holds iff there exists a unique $s \to t \in \text{Red}(\Sigma_{\text{CIL}})$ and valuation ν for s such that $\mathcal{T}_{\text{CIL}}(\delta) \equiv \nu(s)$ and $\mathcal{T}_{\text{CIL}}(P) \equiv \nu(t)$.* □

Proof.

⇒ Suppose $\delta \rightsquigarrow_{\lambda^{\text{CIL}}} P$. By lemma 3.28, there exists a unique $s \to t \in \text{Red}(\Sigma_{\text{CIL}})$ and valuation ν for s such that $\mathcal{T}_{\text{CIL}}(\delta) \equiv \nu(s)$. We must establish that $\mathcal{T}_{\text{CIL}}(P) \equiv \nu(t)$. In each of the cases below, the valuation ν is as constructed in the proof of lemma 3.28.

1. If $\delta \equiv (\lambda x.M')\ @\ V$, then $P = M'[x{:=}V]$, $s \to t = @(\lambda([x]Z(x)), s') \to Z(s')$ and

$$\nu = (\nu' \downarrow \text{MV}(s')) \cup \left\{ Z \mapsto \mathcal{T}_{\text{CIL}}(M')[x := \hat{Z}_1] \right\}$$

   where $s' \in \text{ValPatt}_{\text{CIL}}$ and ν' is a valuation for s' such that $\nu'(s') \equiv \mathcal{T}_{\text{CIL}}(V)$. Then

$$\begin{array}{rcl} \nu(t) & \equiv & \nu(Z(s')) \\ & \equiv & \nu''(\nu(Z)) \quad \text{where } \nu'' = \{\hat{Z}_i \mapsto \nu(s')\} \\ & \equiv & \mathcal{T}_{\text{CIL}}(M')[x := \mathcal{T}_{\text{CIL}}(V)] \\ & \equiv & \mathcal{T}_{\text{CIL}}(M'[x := V]) \quad \text{by lemma 3.43.} \end{array}$$

2. If $\delta = \pi_i \times (V_1, \ldots, V_n)$, then $P = V_i$, $s \to t = \pi_i(\times_n(s_1, \ldots, s_n)) \to s_i$, and $\nu = \bigcup_{1 \leq j \leq n} \nu'_j$ where for $1 \leq j \leq n$, $s_j \in \text{ValPatt}_{\text{CIL}}$, and ν_j is a valuation for s_j such that $\nu_j(s_j) \equiv \mathcal{T}_{\text{CIL}}(V_j)$. Then $\nu(t) \equiv \nu(s_i) \equiv \mathcal{T}_{\text{CIL}}(V_i)$.
3. If $\delta = \textbf{case}\ (\textbf{in}_i\ V)\ \textbf{bind}\ x\ \textbf{in}\ M_1, \ldots, M_n$, then $P = M_i[x{:=}V]$,

$$s \to t = \textbf{case}_{n+1}\ (\textbf{in}_i(s'), [x]Z_1(x), \ldots, [x]Z_n(x)) \to Z_i\,(s')$$

   and $\nu = \nu' \cup \{\, Z_j \mapsto \mathcal{T}_{\text{CIL}}(M_j)[x := \hat{Z}_j] \mid 1 \leq j \leq n \,\}$ where $s' \in \text{ValPatt}_{\text{CIL}}$ and ν' is a valuation for s' such that $\nu'(s') \equiv \mathcal{T}_{\text{CIL}}(V)$. Then

$$\begin{array}{rcl} \nu(t) & \equiv & \nu(Z_i(s')) \\ & \equiv & \nu''(\nu(Z_i)) \quad \text{where } \nu'' = \{\hat{Z}_i \mapsto \nu(s')\} \\ & \equiv & \mathcal{T}_{\text{CIL}}(M_i)[x := \mathcal{T}_{\text{CIL}}(V)] \\ & \equiv & \mathcal{T}_{\text{CIL}}(M_i[x := V]) \quad \text{by lemma 3.43.} \end{array}$$

4. If $\delta = \textbf{rec}\ x.M'$, then $P = M'[x{:=}(\textbf{rec}\ x.M')]$, $s \to t = \mu([x]Z(x)) \to Z(\mu([x]Z(x)))$, and

$\nu = \{Z \mapsto \mathcal{T}_{\text{CIL}}(M')[x := \hat{Z}_1]\}$. Then

$$\begin{array}{rcll}
\nu(t) & \equiv & \nu(Z(\mu([x]Z(x)))) & \\
& \equiv & \nu''(\nu(Z)) & \text{where } \nu'' = \{\hat{Z}_1 \mapsto \nu(\mu([x]Z(x)))\} \\
& \equiv & \mathcal{T}_{\text{CIL}}(M')[x := \nu(\mu([x]Z(x)))] & \\
& \equiv & \mathcal{T}_{\text{CIL}}(M')[x := \mu([x']\nu''(\nu(Z)))] & \text{where } \nu'' = \{\hat{Z}_1 \mapsto x'\} \\
& \equiv & \mathcal{T}_{\text{CIL}}(M')[x := \mu([x']\mathcal{T}_{\text{CIL}}(M')[x := x'])] & \\
& \equiv & \mathcal{T}_{\text{CIL}}(M')[x := \mathcal{T}_{\text{CIL}}(\textbf{rec } x.M')] & \\
& \equiv & \mathcal{T}_{\text{CIL}}(M'[x := (\textbf{rec } x.M')]) & \text{by lemma 3.43.}
\end{array}$$

$\Leftarrow$ Suppose $\mathcal{T}_{\text{CIL}}(\delta) \equiv \nu(s)$, $\mathcal{T}_{\text{CIL}}(P) \equiv \nu(t)$ where $s \to t \in \text{Red}(\Sigma_{\text{CIL}})$ and ν is a valuation for s. By lemma 3.42, there exists λ^{CIL}-redex $\delta = \mathcal{T}^{-1}_{\text{CIL}}(\nu(s))$. The form of δ is determined by $s \to t$ as follows.

1. If $s \to t = @(\lambda([x]Z(x)), s') \to Z(s')$, then there exist $u, v \in \text{Ter}(\Sigma_{\text{CIL}})$ such that $\nu(Z) \equiv u[x := \hat{Z}_1]$ and $\nu(s') \equiv v$. Then

$$\begin{array}{rcl}
\delta & \equiv & \mathcal{T}^{-1}_{\text{CIL}}(\nu(@(\lambda([x]Z(x)), s'))) \\
& \equiv & \mathcal{T}^{-1}_{\text{CIL}}(@(\lambda([x']u[x := x']), v)) \\
& \equiv & (\lambda x.P') \mathbin{@} V
\end{array}$$

   where $V \equiv \mathcal{T}^{-1}_{\text{CIL}}(v)$ and $P' \equiv \mathcal{T}^{-1}_{\text{CIL}}(u)$. Then

$$\begin{array}{rcll}
\mathcal{T}^{-1}_{\text{CIL}}(\nu(t)) & \equiv & \mathcal{T}^{-1}_{\text{CIL}}(\nu(Z(s'))) & \\
& \equiv & \mathcal{T}^{-1}_{\text{CIL}}(\nu''(\nu(Z))) & \text{where } \nu'' = \{\hat{Z}_1 \mapsto \nu(s')\} \\
& \equiv & \mathcal{T}^{-1}_{\text{CIL}}(u[x := v]) & \\
& \equiv & \mathcal{T}^{-1}_{\text{CIL}}(\mathcal{T}_{\text{CIL}}(P')[x := \mathcal{T}_{\text{CIL}}(V)]) & \text{by lemma 3.25} \\
& \equiv & \mathcal{T}^{-1}_{\text{CIL}}(\mathcal{T}_{\text{CIL}}(P'[x := V])) & \text{by lemma 3.43} \\
& \equiv & P'[x := V] & \text{by lemma 3.25.}
\end{array}$$

2. If $s \to t = \pi_i(\times_n(s_1, \ldots, s_n)) \to s_i$ then, for $1 \leq j \leq n$, there exist $u_j \in \text{Ter}(\Sigma_{\text{CIL}})$ such that $\nu(s_j) \equiv u_j$ and $V_j \in \text{Value}_{\lambda^{\text{CIL}}}$ such that $\mathcal{T}^{-1}_{\text{CIL}}(\nu(\pi_i(\times_n(s_1, \ldots, s_n)))) \equiv \pi_i \times (V_1, \ldots, V_n)$. Then $\mathcal{T}^{-1}_{\text{CIL}}(\nu(t)) \equiv \mathcal{T}^{-1}_{\text{CIL}}(\nu(s_i)) \equiv V_i$.
3. The case $s \to t = \textbf{case}_{n+1}(\textbf{in}_i(s), [x]Z_1(x), \ldots, [x]Z_n(x)) \to Z_i(s)$ is similar to 1.
4. The case $s \to t = \mu([x]Z(x)) \to Z(\mu([x]Z(x)))$ is similar to 1. □

Lemma 3.45 (Reduction Simulation). *$M \xrightarrow{\Delta}_{\lambda^{\text{CIL}}} N$ iff $\mathcal{T}_{\text{CIL}}(M) \xrightarrow{\mathcal{T}_{\text{CIL}}(\Delta)}_{\Sigma_{\text{CIL}}} \mathcal{T}_{\text{CIL}}(N)$.* □

Proof. The same as the proof of lemma 3.20 using lemmas 3.44, 3.26, 3.36 and 3.32 (resp.) in place of lemmas 3.19, 3.3, 3.10 and 3.7 (1).

□

Theorem 3.46 (Confluence for λ^{CIL}). *λ^{CIL} is confluent.* □

Proof. Theorem 3.41 and lemmas 3.45 and 3.42. □

4 Standardization and Evaluation in CRSs

This section summarizes essential definitions and theorems developed in [WM00]. Readers interested in proofs and further discussion of the definitions in this section are referred to [WM00]. Readers already familiar with [WM00] can safely skip ahead to section 5.

4.1 Standardization in CRSs

Let Σ be a CRS and let $u \in \mathrm{Ter}(\Sigma)$. A *$\Sigma$-redex (occurrence) ordering* of u is a sequence $\rho = [\vec{\Delta}^n]$ where $\vec{\Delta}_i$ is a Σ-redex occurrence in u for $1 \leq i \leq n$. Given such a sequence $\rho = [\vec{\Delta}^n]$, the redex occurrence Δ_i is *ρ-before* Δ_j iff $1 \leq i < j \leq n$. A Σ-redex ordering ρ is *complete* iff it mentions all of the Σ-redex occurrences in u. A *Σ-redex (occurrence) ordering function* is a function $\mathcal{R}$ such that $\mathcal{R}(u)$ is a complete Σ-redex ordering for all $u \in \mathrm{Ter}(\Sigma)$. Given a Σ-redex ordering function $\mathcal{R}$ and Σ-redex occurrences Δ and Δ' in $u \in \mathrm{Ter}(\Sigma)$, the redex occurrence Δ is *$\mathcal{R}$-before* Δ' iff Δ is ρ-before Δ' where $\rho = \mathcal{R}(u)$.

A notion of *standard* reduction can be defined relative to such redex ordering functions. Let $\sigma = \langle u_0, \Delta_0, u_1, \Delta_1, \ldots \rangle$ be a Σ-reduction sequence and let $\mathcal{R}$ be a Σ-redex ordering function. A Σ-redex occurrence Δ in u_i for $i \geq 0$ is *$\mathcal{R}$-freezing in σ* iff Δ is $\mathcal{R}$-before Δ_i (the to-be-contracted redex occurrence). A Σ-redex occurrence Δ in u_i for $i \geq 0$ is *$\mathcal{R}$-frozen in σ* iff Δ is a residual[6] of a $\mathcal{R}$-freezing redex occurrence in an earlier term in the sequence. The reduction sequence σ is *$\mathcal{R}$-standard* iff no $\mathcal{R}$-frozen redex occurrence is contracted in any reduction step in σ.

For the purpose of proving standardization, it is useful to identify properties of redex ordering functions that are sufficient to guarantee the existence of standard reduction sequences. The two conditions described here are sufficient when taken together. Given any one-step Σ-reduction sequence $\sigma = u \stackrel{\Delta}{\Rightarrow} v$, the required conditions are (1) that every $\mathcal{R}$-freezing Σ-redex occurrence in u has exactly one residual in v (which is necessarily $\mathcal{R}$-frozen) and (2) that all $\mathcal{R}$-frozen Σ-redex occurrences in v are $\mathcal{R}$-before all non-$\mathcal{R}$-frozen Σ-redex occurrences in v. A Σ-redex ordering function satisfying these conditions is *good*.

One of the main results in [WM00] is the following.

Theorem 4.1 (Standardization). *Let Σ be an orthogonal CRS and $\mathcal{R}$ a good Σ-redex ordering function. For any finite Σ-reduction sequence σ, there is an $\mathcal{R}$-standard Σ-reduction sequence σ' such that $\sigma \sim \sigma'$.* □

Proof. See [WM00]. □

4.2 Automatically Obtaining Good Redex Ordering Functions

In order to make use of theorem 4.1, it is necessary to find good redex ordering functions. Let Σ be a CRS with $u \in \mathrm{Ter}(\Sigma)$. A *subterm (occurrence) ordering* of u is a sequence $\gamma = [p_1, \ldots, p_n]$ of members of $\mathrm{Skel}(u)$. A subterm ordering γ is *complete* iff $\gamma = \mathrm{Skel}(u)$. A subterm ordering $\gamma = [\vec{p}^n]$ of u is *top-down* iff for $1 \leq i \leq n$, if $p_i = q \cdot j$ for some path q and number j, then $q \in \{p_1, \ldots, p_{i-1}\}$. A *subterm (occurrence) ordering function* for Σ is a function Γ such that for all $u \in \mathrm{Ter}(\Sigma)$, $\Gamma(u)$ is a complete subterm ordering for u. A subterm ordering function Γ is *top-down* iff for all $u \in \mathrm{DomDef}(\Gamma)$ it holds that $\Gamma(u)$ is a top-down subterm ordering of u. Informally, Γ is top-down iff it visits a node only after visiting the node's parent.

Given a CRS Σ with $u \in \mathrm{Ter}(\Sigma)$, a subterm ordering γ for u determines a redex ordering for u, notation $\lfloor\gamma\rfloor_\Sigma$ (written $\lfloor\gamma\rfloor$ when Σ is obvious), such that if there are Σ-redex occurrences $\Delta = (u.p, r)$ and $\Delta' = (u.p', r')$, then $\Delta \prec_{\lfloor\gamma\rfloor} \Delta'$ iff $p \prec_\gamma p'$. It is clear that if γ is complete then $\lfloor\gamma\rfloor$ is too. If Γ is a subterm ordering function, then $\lfloor\Gamma\rfloor_\Sigma$ (written $\lfloor\Gamma\rfloor$ when Σ is obvious) is the redex ordering function $\{\, u \mapsto \lfloor\Gamma(u)\rfloor \mid u \in \mathrm{Ter}(\Sigma) \,\}$. A subterm ordering function Γ is *good* for Σ iff $\lfloor\Gamma\rfloor$ is a good Σ-redex ordering function.

Figure 5 defines a subterm ordering function generator $\mathcal{G}$. The generator $\mathcal{G}$ can be applied to any CRS Σ and choice function Θ. The choice function Θ may be any fixed total function such that $\Theta(\{\vec{p}\}, \varphi) \in \{\vec{p}\} \cup \{\mathrm{wrong}\}$. The second argument to Θ is extra information that it may use in deciding which member of its first argument to return. If no choice function is specified as in $\mathcal{G}(\Sigma)$, then this stands for $\mathcal{G}(\Sigma, \Theta_{\mathrm{lex}})$ where Θ_{lex} is the choice function such that $\Theta_{\mathrm{lex}}(\{\vec{p}\}, \varphi) = \min_{\mathrm{lex}}\{\vec{p}\}$. In using the various functions defined in figure 5, the subscripts of Σ and Θ will sometimes be omitted when the CRS and choice function being used are clear from the surrounding text.

Intuitively, the meanings of the functions NextOrder, NextPos, Unex, Disc, Inv, Poss, PossUnex, Mand, and Opt are as follows. The function NextOrder either extends a subterm ordering by exploring one additional position in the term or propagates the failure symbol "wrong". Given a term u and a subterm ordering on some subset of $\mathrm{Skel}(u)$, The function NextPos determines the next position to explore in the term, if

[6] See [WM00] for a definition of residual.

$\mathcal{G}(\Sigma,\Theta)(s) = \mathrm{NextOrder}^{k}_{\Sigma,\Theta,s}([])$ where $k = |\mathrm{Skel}(s)|$ and

$$
\begin{array}{rcl}
\mathrm{NextOrder}_{\Sigma,\Theta,s}(\delta) & = & \begin{cases} [\vec{p}^{\,i}, p_{i+1}] & \text{if } \delta = [\vec{p}^{\,i}] \neq \mathrm{wrong} \text{ and } \mathrm{NextPos}_{\Sigma,\Theta}([\vec{p}^{\,i}], s) = p_{i+1} \neq \mathrm{wrong}, \\ \mathrm{wrong} & \text{otherwise}, \end{cases} \\
\mathrm{NextPos}_{\Sigma,\Theta}(\gamma, s) & = & \begin{cases} \Theta(\mathrm{Opt}_{\Sigma}(\gamma, s), \mathrm{Tree}(s) \downarrow \gamma) & \text{if } \mathrm{Opt}_{\Sigma}(\gamma, s) \neq \mathrm{wrong}, \\ \mathrm{wrong} & \text{otherwise}, \end{cases} \\
\mathrm{Unex}(\gamma, s) & = & \min_{\leq}(\mathrm{Skel}(s) \setminus \gamma) \\
\mathrm{Disc}_{\Sigma}(\gamma, s) & = & \{\, (s.p, r) \mid r \in \mathrm{Red}(\Sigma), \forall q \in \mathrm{Int}(r). \\
 & & \qquad p \cdot q \in \gamma \text{ and } \mathrm{Tree}(s)(p \cdot q) = \mathrm{Tree}(\mathrm{LHS}(r))(q) \,\} \\
\mathrm{Inv}_{\Sigma}(\gamma, s) & = & \{\, p \mid (s.q, r) \in \mathrm{Disc}_{\Sigma}(\gamma, s), q' \in \mathrm{Int}(r), p \leq q \cdot q' \,\} \\
\mathrm{Poss}_{\Sigma}(\gamma, s) & = & \{\, (s.p, s' \to t) \mid p \in \gamma \setminus \mathrm{Inv}_{\Sigma}(\gamma, s), s' \to t \in \mathrm{Red}(\Sigma), \\
 & & \qquad \forall q \in \mathrm{Int}(s'). \, p \cdot q \in \gamma \Rightarrow \mathrm{Tree}(s)(p \cdot q) = \mathrm{Tree}(s')(q) \,\} \\
\mathrm{PossUnex}(s.p, r) & = & \{\, p \cdot q \mid q \in \mathrm{Int}(r), p \cdot q \in \mathrm{Unex}(\gamma, s) \,\} \\
\mathrm{Mand}_{\Sigma}(\gamma, s) & = & \bigcap_{(s.p,r) \in \mathrm{Poss}_{\Sigma}(\gamma, s)} \mathrm{PossUnex}(s.p, r) \\
\mathrm{Opt}_{\Sigma}(\gamma, s) & = & \begin{cases} \mathrm{Unex}(\gamma, s) & \text{if } \mathrm{Poss}_{\Sigma}(\gamma, s) = \varnothing, \\ \mathrm{Mand}_{\Sigma}(\gamma, s) & \text{if } \mathrm{Poss}_{\Sigma}(\gamma, s) \neq \varnothing \text{ and } \mathrm{Mand}_{\Sigma}(\gamma, s) \neq \varnothing, \\ \mathrm{wrong} & \text{if } \mathrm{Poss}_{\Sigma}(\gamma, s) \neq \varnothing \text{ and } \mathrm{Mand}_{\Sigma}(\gamma, s) = \varnothing \end{cases}
\end{array}
$$

Figure 5: A subterm ordering function generator.

possible. The function Unex gives the "unexplored frontier positions", Disc the "discovered redex positions",[7] Inv the "invalid positions for undiscovered redexes", Poss the "possible redex occurrences", PossUnex the "unexplored positions on the frontier of a possible redex occurrence", Mand the "must-explore-next positions", and Opt the "options for next position to explore".

Theorem 4.2. *If Σ is orthogonal and $\Gamma = \mathcal{G}(\Sigma)$ is a subterm ordering function for Σ (i.e.,* wrong $\notin \Gamma(\mathrm{Ter}(\Sigma))$*), then Γ is good for Σ.* □

Proof. See [WM00]. □

In light of theorems 4.1 and 4.2, the standardization property for an orthogonal CRS Σ can be proved directly by showing that $\mathcal{G}(\Sigma)$ is a subterm ordering function. However, the applications presented in this paper give rise to a restricted form of CRS that admits further factorization of the proof burden.

4.3 Evaluation in Constructor CRSs

This section describes how to use the generic subterm ordering function generator $\mathcal{G}$ presented in the preceding section for the class of constructor CRSs. Constructor CRSs often arise in programming language semantics and both of the examples in this paper, Σ_{v} and Σ_{CIL}, are constructor CRSs. The subterm ordering function generator will be provided a choice function that is biased to fully explore the internal positions of values before considering other nodes. Such a choice function is called *value respecting*. This section also describes how to derive sets of *values* and *evaluation contexts* directly from the reduction rules of a CRS. These are then used to further define an *evaluation relation* for the CRS.

Context/Term Decomposition The *context/term decomposition* of a preterm w at a set of positions P such that $P \cap \mathrm{Skel}(w) \neq \varnothing$, written $\mathrm{Decomp}(w, P)$, is defined as follows:

$$
(C^{(n)}, \langle \vec{w}^{\,n} \rangle) \in \mathrm{Decomp}\,(w, P) \iff \begin{cases} C[\vec{w}] \equiv w \\ \text{and } \mathrm{Skel}(C) = \{\, q \in \mathrm{Skel}(w) \mid \nexists p \in P. \, p < q \,\} \\ \text{and } \mathrm{Tree}(C)(P \cap \mathrm{Skel}(C)) = \{\Box\} \end{cases}
$$

The set $\mathrm{Decomp}(w, P)$ will contain an infinite number of context/subterm decompositions iff at least one position $p \in \mathrm{Skel}(w) \cap P$ is below a binder in w.

[7] If we allowed non-fully-extended reduction rules, we would have to call the result of Disc the "discovered redex-like patterns".

Value Patterns and Values A pattern s *subsumes* a pattern t, written $s \sqsubseteq t$, iff $\mathrm{Tree}(s)(p) = \mathrm{Tree}(t)(p)$ for all $p \in \mathrm{Int}(s)$. Given constructor CRS Σ, define its sets of *value patterns* and *values* as follows:

$$\begin{aligned}\mathrm{ValPatt}(\Sigma) &= \min_{\sqsubseteq}\{\, s_i \mid s \to t \in \mathrm{Red}(\Sigma), s \equiv F(\vec{s}^n), 1 \le i \le n \,\}\\ \mathrm{Val}(\Sigma) &= \{\, \nu(s) \mid s \in \mathrm{ValPatt}(\Sigma),\ \nu \text{ is a valuation for } s \,\}\end{aligned}$$

Value-Respecting Choice Function Given a constructor CRS Σ, define the following:

$$\begin{aligned}\mathrm{PossVal}_\Sigma(f) &= \{s \in \mathrm{ValPatt}(\Sigma) \mid \exists p' \in \mathrm{Int}(s).f(p') \text{ is undefined},\\ &\qquad \forall p \in \mathrm{Int}(s).f(p) \text{ defined } \Rightarrow f(p) = \mathrm{Tree}(s)(p)\},\\ \mathrm{MandVal}_\Sigma(P,f) &= P \cap (\textstyle\bigcap_{s \in \mathrm{PossVal}_\Sigma(f)} \mathrm{Int}(s)),\\ \mathrm{ValChoose}(\Sigma)(P,f) &= \begin{cases}\min_{\mathrm{lex}}(P) & \text{if } \mathrm{PossVal}_\Sigma(f) = \varnothing,\\ \min_{\mathrm{lex}}(\mathrm{MandVal}_\Sigma(P,f)) & \text{if } \mathrm{PossVal}_\Sigma(f) \neq \varnothing \text{ and } \mathrm{MandVal}_\Sigma(P,f) \neq \varnothing,\\ \mathrm{wrong} & \text{if } \mathrm{PossVal}_\Sigma(f) \neq \varnothing \text{ and } \mathrm{MandVal}_\Sigma(P,f) = \varnothing.\end{cases}\\ \mathrm{ValOrder}(\Sigma) &= \mathcal{G}(\Sigma, \mathrm{ValChoose}(\Sigma))\end{aligned}$$

Given that f represents a partial top-down exploration of some term u, the meaning of $\mathrm{PossVal}_\Sigma(f)$ is that a value pattern $s \in \mathrm{PossVal}_\Sigma(f)$ iff u might be a value by virtue of s matching u, but u has not been explored enough to determine this for certain. The intuitive meaning of $\mathrm{MandVal}_\Sigma(P,f)$ is the set of positions that must be explored. The function $\mathrm{ValChoose}(\Sigma)$ is a "value respecting" choice function, i.e., a choice function that is biased to fully explore the internal positions of values before considering other nodes. $\mathrm{ValOrder}(\Sigma)$ is the subterm ordering function generator specialized with the value respecting choice function.

Let Σ be a constructor CRS, let $u \in \mathrm{Ter}(\Sigma)$, let γ be a top-down subterm ordering of u, and let $f = \mathrm{Tree}(u) \downarrow \gamma$. Then the statement $\mathrm{MaybeVal}(f,\Sigma)$ holds iff $\mathrm{PossVal}_\Sigma(f) \neq \varnothing$. The statement $\mathrm{IsVal}(f,\Sigma)$ holds if and only if there exists an $s \in \mathrm{ValPatt}(\Sigma)$, such that s is linear and fully extended, $\mathrm{Int}(s) \subseteq \mathrm{DomDef}(f)$, and $\forall p \in \mathrm{Int}(s)$ it holds that $\mathrm{Tree}(s)(p) = f(p)$. Let $\mathrm{NotVal}(f,\Sigma)$ hold iff neither $\mathrm{MaybeVal}(f,\Sigma)$ nor $\mathrm{IsVal}(f,\Sigma)$ hold.

Manageability A CRS Σ is *manageable* iff Σ is orthogonal and $\mathrm{ValOrder}(\Sigma)$ is a subterm ordering function (i.e., $\mathrm{wrong} \notin \mathrm{Ran}(\mathrm{ValOrder}(\Sigma))$).

Evaluation Contexts Given a manageable CRS Σ, define:

$$\begin{aligned}\mathrm{CtxtsPos}(s,p,P) &= \{\, C' \mid (C^{(n)},\chi) \in \mathrm{Decomp}(s,\{p\} \cup P),\ \vec{u}^n \in \mathrm{Ter},\ (C',\chi') \in \mathrm{Decomp}(C[\vec{u}],\{p\}) \,\}\\ \mathrm{ContextsMT}_\Sigma(s) &= \{C \mid \ \mathrm{ValOrder}(\Sigma)(s) = [\vec{p}^n], 2 \le i \le |\mathrm{Int}(s)|,\\ &\qquad C \in \mathrm{CtxtsPos}(s, p_i, \min_{\le}(\{p_{i+1},\ldots,p_n\} \cup \{\, p \mid \mathrm{Tree}(s)(p) \in \mathrm{MVar} \,\}))\}\\ \mathcal{E}_{\mathrm{red}}(\Sigma) &= \textstyle\bigcup_{s \to t \in \mathrm{Red}(\Sigma)} \mathrm{ContextsMT}_\Sigma(s)\\ \mathcal{E}_{\mathrm{val}}(\Sigma) &= \textstyle\bigcup_{s \in \mathrm{ValPatt}(\Sigma)} \mathrm{ContextsMT}_\Sigma(s) \cup \{\Box\}\\ \mathrm{EvalCont}(\Sigma) &= \{\, C[C_1[\cdots[C_n]\cdots]] \mid C \in \mathcal{E}_{\mathrm{val}}(\Sigma), \vec{C}^n \in \mathcal{E}_{\mathrm{red}}(\Sigma) \,\}\end{aligned}$$

A context $C \in \mathrm{CtxtsPos}(s,p,P)$ iff C is a one-hole context formed from s by chopping s at the positions $\{p\} \cup P$ and filling in the holes at positions P by arbitrary terms (not metaterms), leaving a single hole at p. A context $C \in \mathrm{ContextsMT}_\Sigma(s)$ iff C is a one-hole context formed from s by partially exploring s according to $\mathrm{ValOrder}(\Sigma)$, putting $\Box$ at the next position to explore, and replacing all other unexplored positions by arbitrary terms.

Evaluation Given redex occurrence $\Delta = (u.p, r)$, the statement $u \overset{\Delta}{\longmapsto}_\Sigma v$ holds iff $u \overset{\Delta}{\longrightarrow}_\Sigma v$ where $u \equiv C^p[u']$ and $C \in \mathrm{EvalCont}(\Sigma)$. In this case, we also write $u \overset{p}{\longmapsto}_\Sigma v$ and $u \longmapsto_\Sigma v$. Let $\longmapsto\!\!\!\!\to_\Sigma$ be the transitive, reflexive closure of $\longmapsto_\Sigma$. Let Σ-eval-nf(u) hold iff there exists no v such that $u \longmapsto_\Sigma v$. The operational semantics for Σ is a function Eval_Σ such that for $u \in \mathrm{Ter}(\Sigma)$,

$$\mathrm{Eval}_\Sigma(u) = \begin{cases}\mathrm{value} & \text{if } \exists v.\ u \longmapsto\!\!\!\!\to_\Sigma v, v \in \mathrm{Val}(\Sigma),\\ \mathrm{diverges} & \text{if } \nexists v.\ u \longmapsto\!\!\!\!\to_\Sigma v, \Sigma\text{-eval-nf}(v),\\ \mathrm{stuck} & \text{otherwise.}\end{cases}$$

Two terms u and v are *observationally equivalent*, written $u \simeq_\Sigma v$, iff $\mathrm{Eval}_\Sigma(C[u]) = \mathrm{Eval}_\Sigma(C[v])$ for every context C such that $\{C[u], C[v]\} \subseteq \mathrm{Ter}(\Sigma)$.

Theorem 4.3 (Plotkin/Wadsworth/Felleisen Standardization). *If Σ is manageable, $u \longrightarrow\!\!\!\!\!\rightarrow_\Sigma v$, and $v \in \mathrm{Val}(\Sigma)$, then there exists $v' \in \mathrm{Val}(\Sigma)$ such that $u \longmapsto\!\!\!\!\!\rightarrow_\Sigma v' \longrightarrow\!\!\!\!\!\rightarrow_\Sigma v$.* □

Proof. See [WM00].

The following properties of constructor systems and $\mathcal{G}$ will be used in section 5 when proving that the CRSs Σ_{v} and Σ_{CIL} are manageable.

Lemma 4.4. *Let Σ be an orthogonal CRS, let $k = |\mathrm{Skel}(s)|$, and let $\gamma = \mathrm{NextOrder}^i_{\Sigma,\Theta,s}([]) = [\vec{p}, p']$, where $1 \leq i < k$. Then for all $(s.p, r) \in \mathrm{Poss}_\Sigma(\gamma, s)$ it holds that $p \leq p'$.* □

Lemma 4.5. *Let Σ be an orthogonal constructor CRS, let $k = |\mathrm{Skel}(s)|$, and let $\gamma = \mathrm{NextOrder}^i_{\Sigma,\Theta,s}([]) = [\vec{p}, p']$, where $1 \leq i < k$. Then there exists a p such that $p \leq p'$ and for all $(s.p'', r) \in \mathrm{Poss}_\Sigma(\gamma, s)$ it holds that $p'' = p$.* □

Corollary 4.6. *Let Σ be an orthogonal constructor CRS such that $\mathrm{Red}(\Sigma) = \{r\}$ is a singleton, let $k = |\mathrm{Skel}(s)|$ and let $\gamma = \mathrm{NextOrder}^i_{\Sigma,\Theta,s}([]) = [\vec{p}, p']$, where $1 \leq i < k$. Then $\mathrm{Poss}_\Sigma(\gamma, s)$ is either $\varnothing$ or a singleton $\{(s.p, r)\}$ where $p \leq p'$.* □

5 Standardization for λ_{v} and λ^{CIL}

This section applies the results from [WM00] summarized in section 4 to the two CRS's Σ_{v} and Σ_{CIL} and then uses the correspondences shown in section 3 to transfer the results to λ_{v} and λ^{CIL}.

For each $(\lambda, \Sigma) \in \{(\lambda_{\mathrm{v}}, \Sigma_{\mathrm{v}}), (\lambda^{\mathrm{CIL}}, \Sigma_{\mathrm{CIL}})\}$, a set of evaluation contexts and a notion of evaluation is mathematically derived for Σ. The primary proof burden is to show that $\mathrm{ValOrder}(\Sigma)$ is a subterm ordering function, i.e., it does not yield "wrong" for any term $u \in \mathrm{Ter}(\Sigma)$. As a corollary, a Plotkin/Wadsworth/Felleisen-style standardization theorem is obtained for Σ. This notion of standardization has the implication that if one defines the operational semantics for Σ in the standard style, then the equational theory of Σ is sound with respect to observational equivalence for Σ, i.e., $=_\Sigma \subseteq \simeq_\Sigma$. To transfer these results from Σ to λ, the remaining burden here is to formulate a definition of evaluation contexts for λ and show that it is equivalent to $\mathrm{EvalCont}(\Sigma)$.

5.1 $(\lambda_{\mathrm{v}}, \Sigma_{\mathrm{v}})$

This section proves standardization for Σ_{v} and then transfers the result to the λ_{v}-calculus.

5.1.1 Standardization for Σ_{v}

Lemma 5.1. $\mathrm{ValOrder}(\Sigma_{\mathrm{v}})$ *is a subterm ordering function.* □

Proof. It is equivalent to show that wrong $\notin \mathrm{Ran}(\mathrm{ValOrder}(\Sigma_{\mathrm{v}}))$. Let $u \in \mathrm{Ter}$, $k = |\mathrm{Skel}(u)|$, and $\Theta_{\mathrm{v}} = \mathrm{ValChoose}(\Sigma_{\mathrm{v}})$. Let $\delta_i = \mathrm{NextOrder}^i_{\Sigma_{\mathrm{v}},\Theta_{\mathrm{v}},u}([])$ for $0 \leq i \leq k$. Because $\mathrm{ValOrder}(\Sigma_{\mathrm{v}})(u) = \delta_k$, it suffices to show that $\delta_k \neq$ wrong. The stronger statement that $\delta_i \neq$ wrong for $0 \leq i \leq k$ will now be proven by induction on i. For $i = 0$, it is trivial to check that $\delta_i = [] \neq$ wrong.

In this proof, the following general facts are used. For arbitrary Σ, Θ, v, and subterm ordering γ for v, all of the following statements hold:

1. $\mathrm{NextOrder}_{\Sigma,\Theta,v}(\gamma) =$ wrong iff $\mathrm{Opt}_\Sigma(\gamma, v) =$ wrong or $\Theta(\mathrm{Opt}_\Sigma(\gamma, v), \mathrm{Tree}(v) \downarrow \gamma) =$ wrong.
2. $\mathrm{Opt}_\Sigma(\gamma, v) =$ wrong iff $\mathrm{Poss}_\Sigma(\gamma, v) \neq \varnothing$ and $\mathrm{Mand}_\Sigma(\gamma, v) = \varnothing$.
3. $\mathrm{Poss}_\Sigma(\gamma, v) \neq \varnothing$ and $\mathrm{Mand}_\Sigma(\gamma, v) = \varnothing$ only if $|\mathrm{Poss}_\Sigma(\gamma, v)| \geq 2$.
4. For $P = \mathrm{Opt}_\Sigma(\gamma, v)$ and $f = \mathrm{Tree}(v) \downarrow \gamma$, it holds that $\mathrm{ValChoose}(\Sigma)(P, f) =$ wrong iff $\mathrm{PossVal}_\Sigma(f) \neq \varnothing$ and $\mathrm{MandVal}_\Sigma(P, f) = \varnothing$.

5. If $\mathrm{NextOrder}^{i}_{\Sigma,\Theta,v}([]) = \gamma \neq \mathrm{wrong}$ for $0 \leq i$, then γ is top-down.

Suppose now that $i \geq 1$. By induction hypothesis, $\delta_{i-1} = \gamma \neq \mathrm{wrong}$ for some subterm ordering γ for u. By fact 1 above, to prove that $\delta_i \neq \mathrm{wrong}$, it suffices to show that $\mathrm{Opt}_{\Sigma_\mathrm{v}}(\gamma, u) \neq \mathrm{wrong}$ and $\Theta_\mathrm{v}(\mathrm{Opt}_{\Sigma_\mathrm{v}}(\gamma, u), \mathrm{Tree}(u) \downarrow \gamma) \neq \mathrm{wrong}$.

That $\mathrm{Opt}_{\Sigma_\mathrm{v}}(\gamma, u) \neq \mathrm{wrong}$ is proven first. Because Σ_v is a one-rule orthogonal constructor CRS, by corollary 4.6 and fact 3 above, if $\mathrm{Poss}_{\Sigma_\mathrm{v}}(\gamma, u) \neq \varnothing$, then $\mathrm{Mand}_{\Sigma_\mathrm{v}}(\gamma, u) \neq \varnothing$. Then fact 2 above gives the desired result.

Next it is proven that $\Theta_\mathrm{v}(\mathrm{Opt}_{\Sigma_\mathrm{v}}(\gamma, u), \mathrm{Tree}(u) \downarrow \gamma) \neq \mathrm{wrong}$. By fact 4 above, it suffices to show that $\mathrm{PossVal}_{\Sigma_\mathrm{v}}(\mathrm{Tree}(u) \downarrow \gamma) \neq \varnothing$ implies $\mathrm{MandVal}_{\Sigma_\mathrm{v}}(\mathrm{Opt}_{\Sigma_\mathrm{v}}(\gamma, u), \mathrm{Tree}(u) \downarrow \gamma) \neq \varnothing$. This will now be established. Suppose that $\mathrm{PossVal}_{\Sigma_\mathrm{v}}(\mathrm{Tree}(u) \downarrow \gamma) \neq \varnothing$. First, observe that $\mathrm{ValPatt}(\Sigma_\mathrm{v}) = \{\mathbf{val}(Z_2)\}$. Because $\varnothing \neq \mathrm{PossVal}_{\Sigma_\mathrm{v}}(\mathrm{Tree}(u) \downarrow \gamma) \subseteq \mathrm{ValPatt}(\Sigma_\mathrm{v})$, it is clear that $\mathrm{PossVal}_{\Sigma_\mathrm{v}}(\mathrm{Tree}(u) \downarrow \gamma) = \{\mathbf{val}(Z_2)\}$. Next, observe that $\mathrm{Int}(\mathbf{val}(Z_2)) = \{\epsilon\}$ and, by the conditions on the definition of PossVal, $(\mathrm{Tree}(u) \downarrow \gamma)(\epsilon)$ must be undefined. By fact 5 above, $\gamma = [] = \delta_0$. Simple calculation reveals that $\mathrm{Opt}_{\Sigma_\mathrm{v}}([], u) = \{\epsilon\} = \mathrm{MandVal}_{\Sigma_\mathrm{v}}(\mathrm{Opt}_{\Sigma_\mathrm{v}}([], u), \mathrm{Tree}(u) \downarrow []) \neq \varnothing$, which is the desired result. □

Lemma 5.2. *Σ_v is manageable.* □

Proof. Lemmas 3.14 and 5.1. □

Corollary 5.3. *$\mathrm{EvalCont}(\Sigma_\mathrm{v})$ is well-defined.* □

Lemma 5.4 (Standardization for Σ_v). *If $u \longrightarrow\!\!\!\!\!\rightarrow_{\Sigma_\mathrm{v}} v$, and $v \in \mathrm{Val}(\Sigma_\mathrm{v})$, then there exists $v' \in \mathrm{Val}(\Sigma_\mathrm{v})$ such that $u \longmapsto\!\!\!\!\!\rightarrow_{\Sigma_\mathrm{v}} v' \longrightarrow\!\!\!\!\!\rightarrow_{\Sigma_\mathrm{v}} v$.* □

Proof. By theorem 4.3 and lemma 5.2. □

5.1.2 Standardization for λ_v

Using lemmas 5.4, 3.1, and 3.20, a standardization theorem for λ_v will now be obtained by observing that $\mathrm{Value}_{\lambda_\mathrm{v}}$ corresponds with the nice members of $\mathrm{Val}(\Sigma_\mathrm{v})$ and by defining a relation $\longmapsto_{\lambda_\mathrm{v}}$ on $\mathrm{Term}_{\lambda_\mathrm{v}}$ that corresponds with the relation $\longmapsto_{\Sigma_\mathrm{v}}$ on $\mathrm{Ter}(\Sigma_\mathrm{v})$.

REMARK 5.5. What is actually required is a one-sided correspondence, that is, that $u \longmapsto_{\Sigma_\mathrm{v}} v$ implies $\mathcal{T}_\mathrm{v}^{-1}(u) \longmapsto_{\lambda_\mathrm{v}} \mathcal{T}_\mathrm{v}^{-1}(v)$, but the two-sided correspondence will be proven. □

The following grammar will be shown to characterize exactly the nice members of $\mathrm{EvalCont}(\Sigma_\mathrm{v})$ (i.e., the set $\mathrm{EvalCont}(\Sigma_\mathrm{v}) \cap \mathrm{DomDef}(\mathcal{T}_\mathrm{v}^{-1})$). In this grammar, restrict u to range over the nice members of $\mathrm{Ter}(\Sigma_\mathrm{v})$ (i.e., the set $\mathrm{DomDef}(\mathcal{T}_\mathrm{v}^{-1})$):

$$e \in \mathrm{NiceEvalCtxts}_{\Sigma_\mathrm{v}} ::= \Box \mid @(e, u) \mid @(\mathbf{val}(\lambda([x]\, u)), e)$$

Lemma 5.6. *If $C \in \mathrm{NiceEvalCtxts}_{\Sigma_\mathrm{v}}$, then C is nice.* □

Proof. By induction on C. □

Lemma 5.7. *Let $C \equiv C_0[C_1[\cdots[C_n]\cdots]]$ where $n \geq 0$, $C_0 \in \mathcal{E}_{\mathrm{val}}(\Sigma_\mathrm{v})$, and $C_1, \ldots, C_n \in \mathcal{E}_{\mathrm{red}}(\Sigma_\mathrm{v})$. If $C \in \mathrm{DomDef}(\mathcal{T}_\mathrm{v}^{-1})$, then $C_0, C_1, \ldots, C_n \in \mathrm{DomDef}(\mathcal{T}_\mathrm{v}^{-1})$. (Informally, if C is nice, then C_0, C_1, ..., C_n are nice.) Furthermore, $C_0 \equiv \Box$.* □

Proof. Suppose C is nice. First, observe that $\mathcal{E}_{\mathrm{val}}(\Sigma_\mathrm{v}) = \{\Box\}$. Thus, $C_0 \equiv \Box$ is nice and $C \equiv C_1[\cdots[C_n]\cdots]$. Then, observe that:

$$\mathcal{E}_{\mathrm{red}}(\Sigma_\mathrm{v}) = \{\, @(\Box, u),\ @(\mathbf{val}(\Box), u),\ @(\mathbf{val}(\lambda(\Box)), u),\ @(\mathbf{val}(\lambda([x]u)), \Box) \mid u \in \mathrm{Ter} \,\}$$

Every element of $\mathcal{E}_{\mathrm{red}}(\Sigma_\mathrm{v})$ is a one-hole context with function symbol @ in the outermost position. Thus, for $C' \in \mathcal{E}_{\mathrm{red}}(\Sigma_\mathrm{v})$ and $u \in \mathrm{Ter}$, it holds that $C'[u]$ is of the form $@(u_1, u_2)$.

Let $u = @(\mathbf{val}(x), \mathbf{val}(x))$ and let $C_i' \equiv C_{i+1}[\ldots[C_n[\Box]]\ldots]$ and $C_i'' \equiv C_1[\ldots[C_{i-1}[\Box]]\ldots]$ for $1 \leq i \leq n$. It is clear that $C \equiv C_i''[C_i[C_i']]$ for $1 \leq i \leq n$. Because C is a nice one-hole context, $C[u]$ is nice by lemma 3.3. For $1 \leq i \leq n$, because $C_i[C_i'[u]]$ is of the form $@(u_1, u_2)$ and $C_i''[C_i[C_i'[u]]]$ is nice, it holds that C_i'' and $C_i[C_i'[u]]$ are nice by lemma 3.7 (2). Then, using lemma 3.7 (2) again, because $C_i'[u]$ is of the form $@(u_1, u_2)$ and $C_i[C_i'[u]]$ is nice, both C_i and $C_i'[u]$ are nice. Thus, it is proven that $C_0, C_1, \ldots, C_n \in \mathrm{DomDef}(\mathcal{T}_\mathrm{v}^{-1})$. □

Lemma 5.8. $\mathrm{EvalCont}(\Sigma_{\mathrm{v}}) \cap \mathrm{DomDef}(\mathcal{T}_{\mathrm{v}}^{-1}) = \mathrm{NiceEvalCtxts}_{\Sigma_{\mathrm{v}}}$. □

Proof. By lemma 5.7, if $C \in \mathrm{EvalCont}(\Sigma_{\mathrm{v}}) \cap \mathrm{DomDef}(\mathcal{T}_{\mathrm{v}}^{-1})$, then $C \equiv C_v[C_{r_1}[\ldots[C_{r_n}]\ldots]]$ with $C_v \equiv \Box$ and $\{C_{r_1}, \ldots, C_{r_n}\} \subset \mathrm{NiceRed}_{\Sigma_{\mathrm{v}}} \subset \mathcal{E}_{\mathrm{red}}(\Sigma_{\mathrm{v}})$ where

$$\mathrm{NiceRed}_{\Sigma_{\mathrm{v}}} = \{\, @(\Box, u), @(\mathbf{val}(\lambda([x]u)), \Box) \mid u \in \mathrm{DomDef}(\mathcal{T}_{\mathrm{v}}^{-1}) \,\}.$$

It is then easy to verify that $C \in \mathrm{EvalCont}(\Sigma_{\mathrm{v}}) \cap \mathrm{DomDef}(\mathcal{T}_{\mathrm{v}}^{-1})$ iff $C \in \mathrm{NiceEvalCtxts}_{\Sigma_{\mathrm{v}}}$. □

Evaluation in λ_{v} Evaluation contexts and an evaluation relation for λ_{v} are defined as follows.

$$E \in \mathrm{EvaluationContext}_{\lambda_{\mathrm{v}}} \ ::= \ \Box \ \mid \ E\,M \ \mid \ (\lambda x.M)\,E \tag{1}$$

$$M \longmapsto_{\lambda_{\mathrm{v}}} N \quad \text{iff} \quad M \equiv E[M'],\ M' \rightsquigarrow_{\lambda_{\mathrm{v}}} N',\ \text{and}\ E[N'] \equiv N$$

Let $\longmapsto\!\!\!\!\rightarrow_{\lambda_{\mathrm{v}}}$ be the reflexive and transitive closure of the $\longmapsto_{\lambda_{\mathrm{v}}}$ relation.

REMARK 5.9. The syntax of evaluation contexts defined in (1) is a restriction of the syntax usually specified in the literature (e.g., [Fel88, SF93]):

$$E \ ::= \ \Box \ \mid \ E\,M \ \mid \ V\,E \tag{2}$$

When using a reasonable type system and evaluation is restricted to closed well typed terms, the definitions in (1) and (2) yield equivalent evaluation relations.

As an example of the difference between (1) and (2), the context $C \equiv (\lambda x.(\lambda y.y)x)(y\Box)$ is an evaluation context by (2) but not by (1). The context C is not an evaluation context by (1) because there is no *chain of demand* from the root of the context to the position of the hole. Consider instead the slightly different evaluation context $E \equiv (\lambda x.(\lambda y.y)x)((\lambda z.z\,z)\Box)$. In this case, there is a chain of demand, because E can be decomposed as $E \equiv E_1[E_2]$ where $E_1 \equiv ((\lambda x.(\lambda y.y)x)\Box)$ and $E_2 \equiv ((\lambda z.z\,z)\Box)$. The evaluation context E_1 *demands* the evaluation of whatever is in its hole, because $E_1[M]$ can be a redex if M matches a specific pattern. Similarly, E_2 demands the evaluation of whatever is in its hole. Looking at the decomposition $C \equiv E_1[C']$ where $C' \equiv (y\Box)$, there is no chain of demand from the root to the hole in C, because C' does not demand the evalution of whatever is in its hole. This is because no amount of reduction done within M can make $C'[M]$ into a redex. □

Lemma 5.10. $\mathcal{T}_{\mathrm{v}}^{-1}(\mathrm{NiceEvalCtxts}_{\Sigma_{\mathrm{v}}}) = \mathrm{EvaluationContext}_{\lambda_{\mathrm{v}}}$. □

Proof. By lemma 5.6 and by inspection of the respective grammars and the definition of $\mathcal{T}_{\mathrm{v}}$. □

Lemma 5.11 (Evaluation Simulation). *$M \longmapsto_{\lambda_{\mathrm{v}}} N$ iff $\mathcal{T}_{\mathrm{v}}(M) \longmapsto_{\Sigma_{\mathrm{v}}} \mathcal{T}_{\mathrm{v}}(N)$.* □

Proof. By lemmas 3.20, 5.8, and 5.10. □

Theorem 5.12 (Standardization for λ_{v}). *If $M \longrightarrow\!\!\!\!\rightarrow_{\lambda_{\mathrm{v}}} V$, then $\exists\, V'$ such that $M \longmapsto\!\!\!\!\rightarrow_{\lambda_{\mathrm{v}}} V' \longrightarrow\!\!\!\!\rightarrow_{\lambda_{\mathrm{v}}} V$.* □

Proof. If M is a value, then take $V' \equiv M$. Otherwise, by lemmas 3.1 (2) and 3.20 there exist $u \equiv \mathcal{T}_{\mathrm{v}}(M) \in \mathrm{Ter}(\Sigma_{\mathrm{v}})$ and $v \equiv \mathcal{T}_{\mathrm{v}}(V) \in \mathrm{Val}(\Sigma_{\mathrm{v}})$ such that $u \longrightarrow\!\!\!\!\rightarrow_{\Sigma_{\mathrm{v}}} v$. By lemma 5.4, there exists a $v' \in \mathrm{Val}(\Sigma_{\mathrm{v}})$ such that $u \longmapsto\!\!\!\!\rightarrow_{\Sigma_{\mathrm{v}}} v' \longrightarrow\!\!\!\!\rightarrow_{\Sigma_{\mathrm{v}}} v$. Then by lemmas 3.1 (2), 5.11, and 3.20, for $V' \equiv \mathcal{T}_{\mathrm{v}}^{-1}(v') \in \mathrm{Value}_{\lambda_{\mathrm{v}}}$, $M \longmapsto\!\!\!\!\rightarrow_{\lambda_{\mathrm{v}}} V' \longrightarrow\!\!\!\!\rightarrow_{\lambda_{\mathrm{v}}} V$. □

5.2 $(\lambda^{\mathrm{CIL}}, \Sigma_{\mathrm{CIL}})$

This section proves standardization for Σ_{CIL} and then transfers the result to λ^{CIL}. The top-level structure follows that of section 5.1.

5.2.1 Standardization for Σ_{CIL}

Lemma 5.13. *Let $u \in \text{Ter}(\Sigma_{\text{CIL}})$, let $X \subseteq \text{Skel}(u)$ be prefix-closed, let $f = \text{Tree}(u) \downarrow X$, let $P = \text{Unex}(X, u)$ and let $\mathcal{S} = \text{PossVal}_{\Sigma_{\text{CIL}}}(f)$. Then $\exists p \in P$ such that $\forall s \in \mathcal{S}$, $p \in \text{Int}(s)$.* □

Proof. By induction on $\text{MaxPathLen}(f) = \max(\{\, |p| \mid p \in \text{DomDef}(f) \,\} \cup \{-1\})$. By cases on f.

1. $f = \varnothing$. Then $\mathcal{S} = \text{ValPatt}(\Sigma_{\text{CIL}})$ and $P = \{\epsilon\}$ and $\epsilon \in \text{Int}(s)$ for all $s \in \mathcal{S}$.
2. $f(\epsilon) = c$. Then $\mathcal{S} = \varnothing$ and the result is trivially true.
3. $f(\epsilon) = \lambda$. Then there are 2 cases.
   (a) $f(1)$ is defined. Then $\mathcal{S} = \varnothing$.
   (b) $f(1)$ is undefined. Then $P = \{1\}$ and $1 \in \text{Int}(s)$ for all $s \in \mathcal{S}$.
4. (a) $f(\epsilon) = \times_n$ with $u \equiv \times_n(u_1, \ldots, u_n)$. There are 3 cases to consider.
      i. $\exists\, i \in \{1, \ldots, n\}$ such that $\text{NotVal}(f_i, \Sigma_{\text{CIL}})$. Then $\text{NotVal}(f, \Sigma_{\text{CIL}})$ and $\mathcal{S} = \varnothing$.
      ii. $\forall\, i \in \{1, \ldots, n\}$, $\text{IsVal}(f_i, \Sigma_{\text{CIL}})$. Then $\text{IsVal}(f, \Sigma_{\text{CIL}})$ and therefore $\mathcal{S} = \varnothing$.
      iii. $\forall\, i \in \{1, \ldots, n\}$, $\neg\text{NotVal}(f_i, \Sigma_{\text{CIL}})$ and $\exists i \in \{1, \ldots, n\}$ such that $\text{MaybeVal}(f_i, \Sigma_{\text{CIL}})$. Then it follows that $\text{MaybeVal}(f, \Sigma_{\text{CIL}})$. Let j be any integer such that $\text{MaybeVal}(f_j, \Sigma_{\text{CIL}})$. Let $X_j = \{\, p \mid j \cdot p \in X \,\}$ and $f_j = \text{Tree}(u_j) \downarrow X_j$. By lemma 3.31 (3), $u_j \in \text{Ter}(\Sigma_{\text{CIL}})$. Note that $X_j \subseteq \text{Skel}(u_j)$ is prefix-closed. Let $\mathcal{S}_j = \{\, s_j \mid \times_n(s_1, \ldots, s_n) \in \mathcal{S} \,\}$. Note that $\mathcal{S}_j = \text{PossVal}_{\Sigma_{\text{CIL}}}(f_j)$. Let $P_j = \{\, p \mid j \cdot p \in P \,\}$. Note that $P_j = \text{Unex}(X_j, u_j)$. Then by the induction hypothesis, there exists a path $p \in P_j$ such that $p \in \text{Int}(s)$ for all $s \in \mathcal{S}_j$. Therefore $j \cdot p \in P$ and $j \cdot p \in \text{Int}(s)$ for all $s \in \mathcal{S}$.
   (b) $f(\epsilon) = \mathbf{in}_i$ with $u \equiv \mathbf{in}_i(u')$. This case is handled similarly (and is simpler).
5. $f(\epsilon) \in \{\, \mu, \mathbf{case}_i, @, \pi_i \mid i \geq 1 \,\}$. These cases are similar to case 2 except that $\mathcal{S} = \varnothing$ because the term being explored is definitely not a value. □

Lemma 5.14. *Let $u \in \text{Ter}(\Sigma_{\text{CIL}})$, let $k = |\text{Skel}(u)|$, let $\Theta_{\text{CIL}} = \text{ValChoose}(\Sigma_{\text{CIL}})$, let $0 \leq i \leq k$, and let $\gamma = \text{NextOrder}^i_{\Sigma_{\text{CIL}}, \Theta_{\text{CIL}}, u}([]) \neq \text{wrong}$. Let $p \in \gamma$ and let $\mathcal{S} = \{\, s \mid (u.p, s \to t) \in \text{Poss}_{\Sigma_{\text{CIL}}}(\gamma, u) \,\} \neq \varnothing$. Let $P = \bigcap_{s \in \mathcal{S}} \text{Int}(s)$. Then there exists a path $q \in P$ such that $p \cdot q \in \text{Unex}_{\Sigma_{\text{CIL}}}(\gamma, u)$.* □

Proof. Observe that $\text{Tree}(u)(p) = \text{Tree}(s)(\epsilon)$ for any $s \in \mathcal{S}$. By a case analysis on $\text{Tree}(u)(p)$.

1. Suppose $\text{Tree}(u)(p) = @$. Then, for every $s \in \mathcal{S}$, it holds that $s \equiv @(\lambda([x]Z(x)), s')$ for some metavariable Z and $s' \in \text{ValPatt}_{\text{CIL}}$. By cases on γ.
   (a) Suppose $p \cdot 1 \notin \gamma$. Then $q = 1$ satisfies the desired conclusion.
   (b) Suppose $p \cdot 1 \in \gamma$ and $p \cdot 1 \cdot 1 \notin \gamma$. Then $q = 1 \cdot 1$ satisfies the desired conclusion.
   (c) Suppose otherwise. Let $(C, \langle u' \rangle) \in \text{Decomp}(u, \{p \cdot 2\})$, and let $X = \{\, p' \mid p \cdot 2 \cdot p' \in \gamma \,\}$. It is not hard to see that $X \subseteq \text{Skel}(u')$ is prefix-closed and $u' \in \text{Ter}(\Sigma_{\text{CIL}})$ by lemma 3.31 (2). Let $f = \text{Tree}(u') \downarrow X$. Let $\mathcal{S}' = \{\, s \mid @(\lambda([x]Z(x)), s) \in \mathcal{S} \,\}$ and let $P' = \bigcap_{s \in \mathcal{S}'} \text{Int}(s)$. Observe that $\mathcal{S}' \subset \text{ValPatt}_{\text{CIL}}$ and that for every $s \in \mathcal{S}'$ there exists a $s' \in \text{PossVal}_{\Sigma_{\text{CIL}}}(f)$ such that $s \simeq s'$. Then, lemma 5.13 can be used to show there exists a path $q' \in \text{Unex}(X, u')$ such that $q' \in P'$. Then $q = 2 \cdot q' \in P$ satisfies the desired conclusion.
2. Suppose $\text{Tree}(u)(p) = \pi_i$ where $i \geq 1$. Then, for every $s \in \mathcal{S}$, it holds that $s \equiv \pi_i(s')$ for some $s' \in \text{ValPatt}_{\text{CIL}}$. Then the same reasoning used above can be repeated.
3. Suppose $\text{Tree}(u)(p) = \mathbf{case}_{n+1}$ where $n \geq 1$. Then, for every $s \in \mathcal{S}$, there exist some metavariables Z_1, $\ldots$, Z_n and metaterm $s' \in \text{ValPatt}_{\text{CIL}}$ such that $s \equiv \mathbf{case}_{n+1}(s', [x]Z_1(x), \ldots, [x]Z_n(x))$. By cases on γ.
   (a) Suppose $p \cdot i \notin \gamma$ where $2 \leq i \leq n + 1$. Then $q = i$ satisfies the desired conclusion.

(b) Suppose otherwise. Then the same reasoning used above can be repeated.

4. Suppose $\mathrm{Tree}(u)(p) = \mu$. Then, $\mathcal{S} = \{\mu([x]Z(x))\}$. Then $q = 1$ satisfies the desired conclusion. □

Let $\Gamma_{\mathrm{CIL}} = \mathrm{ValOrder}(\Sigma_{\mathrm{CIL}})$.

Lemma 5.15. *Γ_{CIL} is a subterm ordering function.* □

Proof. It is equivalent to show that wrong $\notin \mathrm{Ran}(\Gamma_{\mathrm{CIL}})$. Let $u \in \mathrm{Ter}(\Sigma_{\mathrm{CIL}})$, let $\Theta_{\mathrm{CIL}} = \mathrm{ValChoose}(\Sigma_{\mathrm{CIL}})$, and let $k = |\mathrm{Skel}(u)|$. We show by induction on i that $\mathrm{NextOrder}^i_{\Sigma_{\mathrm{CIL}},\Theta_{\mathrm{CIL}},u}([]) \neq \text{wrong}$ for $1 \leq i \leq k$. Observe that in general, if $\delta \neq \text{wrong}$, then $\mathrm{NextOrder}_{\Sigma,\Theta,u}(\delta) = \text{wrong}$ iff either $\mathrm{Opt}_\Sigma(\delta, u) = \text{wrong}$ or

$$\Theta\left(\mathrm{Opt}_\Sigma(\delta, u), \mathrm{Tree}(u) \downarrow \delta\right) = \text{wrong}.$$

For $i = 1$, $\mathrm{Poss}_{\Sigma_{\mathrm{CIL}}}([], u) = \varnothing$ and $\mathrm{Opt}_{\Sigma_{\mathrm{CIL}}}([], u) = \mathrm{Unex}_{\Sigma_{\mathrm{CIL}}}([], u) = \{\epsilon\}$. It is then easy to see that $\Theta_{\mathrm{CIL}}(\{\epsilon\}, \mathrm{Tree}(u) \downarrow \varnothing) = \epsilon$ and therefore $\mathrm{NextOrder}^1_{\Sigma_{\mathrm{CIL}},\Theta_{\mathrm{CIL}},u}([]) = [\epsilon]$.

For the induction step, we show that if $\mathrm{Poss}_{\Sigma_{\mathrm{CIL}}}(\delta_{i-1}, u) \neq \varnothing$, then $\mathrm{Mand}_{\Sigma_{\mathrm{CIL}}}(\delta_{i-1}, u) \neq \varnothing$ and that if $\mathrm{PossVal}_{\Sigma_{\mathrm{CIL}}}(\mathrm{Tree}(u) \downarrow \delta_{i-1}) \neq \varnothing$, then $\mathrm{MandVal}_{\Sigma_{\mathrm{CIL}}}(\mathrm{Opt}_{\Sigma_{\mathrm{CIL}}}(\delta_{i-1}, u), \mathrm{Tree}(u) \downarrow \delta_{i-1}) \neq \varnothing$.

To establish the former, let $\delta_{i-1} = [\vec{p}, p']$ and suppose $\mathrm{Poss}_{\Sigma_{\mathrm{CIL}}}(\delta_{i-1}, u) \neq \varnothing$. By lemma 4.5, there exists a path $p \in \delta_{i-1}$ such that for all $(u.p', s \to t) \in \mathrm{Poss}_{\Sigma_{\mathrm{CIL}}}(\delta_{i-1}, u)$, it holds that $p = p'$. Then, by lemma 5.14, there exists a q such that $p \cdot q \in \mathrm{Unex}_{\Sigma_{\mathrm{CIL}}}(\delta_{i-1}, u)$ and for all $(u.p, s \to t) \in \mathrm{Poss}_{\Sigma_{\mathrm{CIL}}}(\delta_{i-1}, u)$, it holds that $q \in \mathrm{Int}(s)$. Thus, $\mathrm{Mand}_{\Sigma_{\mathrm{CIL}}}(\delta_{i-1}, u) \neq \varnothing$.

To establish the latter, let $f = \mathrm{Tree}(u) \downarrow \delta_{i-1}$ and observe that if $\mathrm{PossVal}_{\Sigma_{\mathrm{CIL}}}(f) \neq \varnothing$, then $\mathrm{FDes}(\Sigma_{\mathrm{CIL}}) \cap \mathrm{Ran}(f) = \varnothing$. Therefore $\mathrm{Opt}_{\Sigma_{\mathrm{CIL}}}(\delta_{i-1}, u) = \mathrm{Unex}(\delta_{i-1}, u)$. Then by lemma 5.13,

$$\mathrm{MandVal}_{\Sigma_{\mathrm{CIL}}}\left(\mathrm{Opt}_{\Sigma_{\mathrm{CIL}}}(\delta_{i-1}, u), \mathrm{Tree}(u) \downarrow \delta_{i-1}\right) \neq \varnothing.$$ □

Lemma 5.16. *Σ_{CIL} is manageable.* □

Proof. Lemmas 3.40 and 5.15. □

Corollary 5.17. $\mathrm{EvalCont}(\Sigma_{\mathrm{CIL}})$ *is well-defined.* □

Lemma 5.18 (Standardization for Σ_{CIL}). *If $u \longrightarrow\!\!\!\!\!\rightarrow_{\Sigma_{\mathrm{CIL}}} v$, and $v \in \mathrm{Val}(\Sigma_{\mathrm{CIL}})$, then there exists $v' \in \mathrm{Val}(\Sigma_{\mathrm{CIL}})$ such that $u \longmapsto\!\!\!\!\!\rightarrow_{\Sigma_{\mathrm{CIL}}} v' \longrightarrow\!\!\!\!\!\rightarrow_{\Sigma_{\mathrm{CIL}}} v$.* □

Proof. By theorem 4.3 and lemma 5.16. □

5.2.2 Standardization for λ^{CIL}

Following the same top-level structure as the proof presented in section 5.1.2, standardization for λ^{CIL} will be obtained by defining $\longmapsto_{\lambda^{\mathrm{CIL}}}$ on $\mathrm{Term}_{\lambda^{\mathrm{CIL}}}$ and showing this relation corresponds with the evaluation relation $\longmapsto_{\Sigma_{\mathrm{CIL}}}$ of Σ_{CIL}.

We begin with a grammar that will be shown to characterize the set $\mathrm{EvalCont}(\Sigma_{\mathrm{CIL}}) \cap \mathrm{DomDef}(\mathcal{T}^{-1}_{\mathrm{CIL}})$, i.e., the set of nice evaluation contexts. In this grammar, restrict v to range over the set $\mathrm{Val}(\Sigma_{\mathrm{CIL}}) \cap \mathrm{DomDef}(\mathcal{T}^{-1}_{\mathrm{CIL}})$ and restrict u to range over $\mathrm{DomDef}(\mathcal{T}^{-1}_{\mathrm{CIL}})$.

$$e \in \mathrm{NiceEvalCtxts}_{\Sigma_{\mathrm{CIL}}} ::= f \mid \times_{n+m+1}(v_1, \ldots, v_n, e, u_1, \ldots, u_m) \mid \mathbf{in}_j(e)$$

$$\begin{aligned} f ::= {} & \Box \mid @(f, u) \mid @(\lambda([x]u), e) \\ & \mid \pi_i(f) \mid \pi_i(\times_{n+m+1}(v_1, \ldots, v_n, e, u_1, \ldots, u_m)) \\ & \mid \mathbf{case}_{l+1}(f, [x]u_1, \ldots, [x]u_l) \mid \mathbf{case}_{l+1}(\mathbf{in}_j(e), [x]u_1, \ldots, [x]u_l) \\ & \text{where } n, m \geq 0 \text{ and } 1 \leq i \leq n+m+1 \text{ and } 1 \leq j \leq l \end{aligned}$$

Lemma 5.19. *Any context e or f derived by the above grammar is nice.* □

Proof. By induction on the structure of e or f. □

Lemma 5.20. *Let $C \equiv C_v[C_{r_1}[\ldots[C_{r_n}]\ldots]]$ where $C_v \in \mathcal{E}_{\mathrm{val}}(\Sigma_{\mathrm{CIL}})$ and $\{C_{r_1}, \ldots, C_{r_n}\} \subset \mathcal{E}_{\mathrm{red}}(\Sigma_{\mathrm{CIL}})$. Then if $C \in \mathrm{EvalCont}(\Sigma_{\mathrm{CIL}}) \cap \mathrm{DomDef}(\mathcal{T}^{-1}_{\mathrm{CIL}})$ it holds that $\{C_v, C_{r_1}, \ldots, C_{r_n}\} \subset \mathrm{DomDef}(\mathcal{T}^{-1}_{\mathrm{CIL}})$.* □

Proof. First observe that $\mathcal{E}_{\mathrm{val}}(\Sigma_{\mathrm{CIL}})$ is the least set such that

$$\begin{aligned}\mathcal{E}_{\mathrm{val}}(\Sigma_{\mathrm{CIL}}) = \{\, C \mid\ & C \equiv \Box \text{ or } C \equiv \lambda(\Box) \text{ or } C \equiv \mathbf{in}_i(C_v) \text{ where } C_v \in \mathcal{E}_{\mathrm{val}}(\Sigma_{\mathrm{CIL}}) \text{ or}\\ & C \equiv\ \times_n(v_1, \ldots, v_{k-1}, C_v, u_{k+1}, \ldots, u_n) \text{ where } 1 \leq k \leq n,\\ & \quad \{v_1, \ldots, v_{k-1}\} \subset \mathrm{Val}(\Sigma_{\mathrm{CIL}}), C_v \in \mathcal{E}_{\mathrm{val}}(\Sigma_{\mathrm{CIL}}) \text{ and } \{u_{k+1}, \ldots, u_n\} \subset \mathrm{Ter}\,\}\end{aligned}$$

and that

$$\begin{aligned}\mathcal{E}_{\mathrm{red}}(\Sigma_{\mathrm{CIL}}) = \{\, C \mid\ & C \equiv @(\Box, u) \text{ where } u \in \mathrm{Ter} \text{ or } C \equiv @(\lambda(\Box), u) \text{ where } u \in \mathrm{Ter} \text{ or}\\ & C \equiv @(\lambda([x]u), C_v) \text{ where } u \in \mathrm{Ter} \text{ and } C_v \in \mathcal{E}_{\mathrm{val}}(\Sigma_{\mathrm{CIL}}) \text{ or } C \equiv \pi_i(\Box) \text{ or}\\ & C \equiv\ \pi_i(\times_n(v_1, \ldots, v_{k-1}, C_v, u_{k+1}, \ldots, u_n)) \text{ where } 1 \leq i \leq n, 1 \leq k \leq n,\\ & \quad \{v_1, \ldots, v_{k-1}\} \subset \mathrm{Val}(\Sigma_{\mathrm{CIL}}), C_v \in \mathcal{E}_{\mathrm{val}}(\Sigma_{\mathrm{CIL}}) \text{ and } \{u_{k+1}, \ldots, u_n\} \subset \mathrm{Ter} \text{ or}\\ & C \equiv \mathbf{case}_{n+1}(\Box, u_1, \ldots, u_n) \text{ where } \{u_1, \ldots, u_n\} \subset \mathrm{Ter} \text{ or}\\ & C \equiv\ \mathbf{case}_{n+1}(\mathbf{in}_j(C_v), u_1, \ldots, u_n) \text{ where } 1 \leq j \leq n, C_v \in \mathcal{E}_{\mathrm{val}}(\Sigma_{\mathrm{CIL}}) \text{ and}\\ & \quad \{u_1, \ldots, u_n\} \subset \mathrm{Ter} \text{ or}\\ & C \equiv\ \mathbf{case}_{n+1}(\mathbf{in}_j(v), [x]u_1, \ldots, [x]u_{k-1}, \Box, u_{k+1}, \ldots, u_n) \text{ where } 1 \leq j \leq n,\\ & \quad 1 \leq k \leq n, v \in \mathrm{Val}(\Sigma_{\mathrm{CIL}}) \text{ and } \{u_1, \ldots, u_n\} \subset \mathrm{Ter} \text{ or}\\ & C \equiv \mu(\Box)\,\}.\end{aligned}$$

Observe that every element of the set $\mathcal{E}_{\mathrm{red}}(\Sigma_{\mathrm{CIL}})$ is a one-hole context with a function symbol in the outermost position. Thus, for $C' \in \mathcal{E}_{\mathrm{red}}(\Sigma_{\mathrm{CIL}})$ and $u \in \mathrm{Ter}$, $C'[u]$ is of the form $F(u_1, \ldots, u_m)$.

Let $c \in \mathrm{Const}_{\Sigma_{\mathrm{CIL}}}$ and, for $1 \leq i \leq n$, let $C_i \equiv C_{r_i}[\ldots[C_{r_n}]\ldots]$. Observe that because C is a nice one-hole context, $C[c]$ is nice. We can now conclude the proof by n applications of lemma 3.33. Because $C[c] \equiv C_v[C_1[c]]$ is nice and $C_1[c]$ is of the form $F_1(u_1, \ldots, u_{m_1})$, by lemma 3.33, C_v and $C_1[c]$ are nice. Then, because $C_1[c] \equiv C_{r_1}[C_2[c]]$ is nice and $C_2[c]$ is of the form $F_2(u_1, \ldots, u_{m_2})$, C_{r_1} and $C_2[c]$ are nice. This reasoning holds for all C_i, $1 \leq i \leq n$. Therefore, $\{C_v, C_{r_1}, \ldots, C_{r_n}\} \subset \mathrm{DomDef}(\mathcal{T}_{\mathrm{CIL}}^{-1})$. □

Lemma 5.21. $\mathrm{EvalCont}(\Sigma_{\mathrm{CIL}}) \cap \mathrm{DomDef}(\mathcal{T}_{\mathrm{CIL}}^{-1}) = \mathrm{NiceEvalCtxts}_{\Sigma_{\mathrm{CIL}}}$. □

Proof. By lemma 5.20, if $C \in \mathrm{EvalCont}(\Sigma_{\mathrm{CIL}}) \cap \mathrm{DomDef}(\mathcal{T}_{\mathrm{CIL}}^{-1})$, then $C \equiv C_v[C_{r_1}[\ldots[C_{r_n}]\ldots]]$ with $C_v \in \mathrm{NiceVal}_{\Sigma_{\mathrm{CIL}}}$ and $\{C_{r_1}, \ldots, C_{r_n}\} \subset \mathrm{NiceRed}_{\Sigma_{\mathrm{CIL}}}$ where

$$\begin{aligned}\mathrm{NiceVal}_{\Sigma_{\mathrm{CIL}}} = \{\, C \mid\ & C \equiv \Box \text{ or } \text{ or } C \equiv \mathbf{in}_i(C_v) \text{ where } C_v \in \mathrm{NiceVal}_{\Sigma_{\mathrm{CIL}}} \text{ or}\\ & C \equiv\ \times_n(v_1, \ldots, v_{k-1}, C_v, u_{k+1}, \ldots, u_n) \text{ where } 1 \leq k \leq n,\\ & \quad \{v_1, \ldots, v_{k-1}\} \subset \mathrm{Val}(\Sigma_{\mathrm{CIL}}) \cap \mathrm{DomDef}(\mathcal{T}_{\mathrm{CIL}}^{-1}), C_v \in \mathrm{NiceVal}_{\Sigma_{\mathrm{CIL}}} \text{ and}\\ & \quad \{u_{k+1}, \ldots, u_n\} \subset \mathrm{DomDef}(\mathcal{T}_{\mathrm{CIL}}^{-1}),\ \}\end{aligned}$$

$$\begin{aligned}\mathrm{NiceRed}_{\Sigma_{\mathrm{CIL}}} = \{\, C \mid\ & C \equiv @(\Box, u) \text{ where } u \in \mathrm{DomDef}(\mathcal{T}_{\mathrm{CIL}}^{-1}) \text{ or}\\ & C \equiv @(\lambda([x]u), C_v) \text{ where } u \in \mathrm{DomDef}(\mathcal{T}_{\mathrm{CIL}}^{-1}) \text{ and } C_v \in \mathrm{NiceVal}_{\Sigma_{\mathrm{CIL}}} \text{ or}\\ & C \equiv \pi_i(\Box) \text{ or}\\ & C \equiv\ \pi_i(\times_n(v_1, \ldots, v_{k-1}, C_v, u_{k+1}, \ldots, u_n)) \text{ where } 1 \leq i \leq n, 1 \leq k \leq n,\\ & \quad \{v_1, \ldots, v_{k-1}\} \subset \mathrm{Val}(\Sigma_{\mathrm{CIL}}) \cap \mathrm{DomDef}(\mathcal{T}_{\mathrm{CIL}}^{-1}), C_v \in \mathrm{NiceVal}_{\Sigma_{\mathrm{CIL}}} \text{ and}\\ & \quad \{u_{k+1}, \ldots, u_n\} \subset \mathrm{DomDef}(\mathcal{T}_{\mathrm{CIL}}^{-1}) \text{ or}\\ & C \equiv \mathbf{case}_{n+1}(\Box, [x]u_1, \ldots, [x]u_n) \text{ where } \{u_1, \ldots, u_n\} \subset \mathrm{DomDef}(\mathcal{T}_{\mathrm{CIL}}^{-1}) \text{ or}\\ & C \equiv\ \mathbf{case}_{n+1}(\mathbf{in}_j(C_v), [x]u_1, \ldots, [x]u_n) \text{ where } 1 \leq j \leq n, C_v \in \mathrm{NiceVal}_{\Sigma_{\mathrm{CIL}}} \text{ and}\\ & \quad \{u_1, \ldots, u_n\} \subset \mathrm{DomDef}(\mathcal{T}_{\mathrm{CIL}}^{-1})\,\}\end{aligned}$$

It is then easy to verify that $C \in \mathrm{EvalCont}(\Sigma_{\mathrm{CIL}}) \cap \mathrm{DomDef}(\mathcal{T}_{\mathrm{CIL}}^{-1})$ iff $C \in \mathrm{NiceEvalCtxts}_{\Sigma_{\mathrm{CIL}}}$. □

Evaluation in λ^{CIL} Evaluation contexts and an evaluation relation $\longmapsto_{\lambda^{\mathrm{CIL}}}$ for λ^{CIL} are defined in figure 6. Let $\longmapsto\!\!\!\!\rightarrow_{\lambda^{\mathrm{CIL}}}$ be the reflexive and transitive closure of $\longmapsto_{\lambda^{\mathrm{CIL}}}$.

Lemma 5.22. $\mathcal{T}_{\mathrm{CIL}}^{-1}(\mathrm{NiceEvalCtxts}_{\Sigma_{\mathrm{CIL}}}) = \mathrm{EvaluationContext}_{\lambda^{\mathrm{CIL}}}$. □

Proof. By lemma 5.19 and by inspection of the grammars and the definition of $\mathcal{T}_{\mathrm{CIL}}$. □

The correspondence between the evaluation relations in λ^{CIL} and Σ_{CIL} will now be established as it was for λ_{v} and Σ_{v} in section 5.1.2.

Evaluation Contexts

$$E \in \text{EvaluationContext}_{\lambda^{\text{CIL}}} ::= F \mid \times(V_1, \ldots, V_n, E, M_1, \ldots, M_m) \mid \mathbf{in}_j\ E$$

$$\begin{aligned} F ::= {} & \Box \mid F \mathbin{@} M \mid (\lambda x.M) \mathbin{@} E \\ & \mid \pi_i\ F \mid \pi_i \times(V_1, \ldots, V_n, E, M_1, \ldots, M_m) \\ & \mid \mathbf{case}\ F\ \mathbf{bind}\ x\ \mathbf{in}\ M_1, \ldots, M_l \\ & \mid \mathbf{case}\ (\mathbf{in}_j\ E)\ \mathbf{bind}\ x\ \mathbf{in}\ M_1, \ldots, M_l \\ & \text{where } n, m \geq 0 \text{ and } 1 \leq i \leq n+m+1 \text{ and } 1 \leq j \leq l \end{aligned}$$

One-Step Evaluation Relation

$$M \longmapsto_{\lambda^{\text{CIL}}} N \quad \text{iff} \quad M \equiv E[M'],\ M' \rightsquigarrow_{\lambda^{\text{CIL}}} N',\ \text{and } E[N'] \equiv N$$

Figure 6: Evaluation in λ^{CIL}.

Lemma 5.23 (Evaluation Simulation). *$M \longmapsto_{\lambda^{\text{CIL}}} N$ iff $\mathcal{T}_{\text{CIL}}(M) \longmapsto_{\Sigma_{\text{CIL}}} \mathcal{T}_{\text{CIL}}(N)$.* □

Proof. By lemmas 3.45, 5.21, and 5.22. □

Theorem 5.24 (Standardization for λ^{CIL}). *If $M \longrightarrow\!\!\!\!\!\rightarrow_{\lambda^{\text{CIL}}} V$, then $M \longmapsto\!\!\!\!\!\rightarrow_{\lambda^{\text{CIL}}} V' \longrightarrow\!\!\!\!\!\rightarrow_{\lambda^{\text{CIL}}} V$ for some $V' \in \text{Value}_{\lambda^{\text{CIL}}}$.* □

Proof. If M is a value, then take $V' \equiv M$. Otherwise, by lemmas 3.25 and 3.45 there exist $u \equiv \mathcal{T}_{\text{CIL}}(M) \in \text{Ter}(\Sigma_{\text{CIL}})$ and $v \equiv \mathcal{T}_{\text{CIL}}(V) \in \text{Val}(\Sigma_{\text{CIL}})$ such that $u \longrightarrow\!\!\!\!\!\rightarrow_{\Sigma_{\text{CIL}}} v$. By lemma 5.18, there exists a $v' \in \text{Val}(\Sigma_{\text{CIL}})$ such that $u \longmapsto\!\!\!\!\!\rightarrow_{\Sigma_{\text{CIL}}} v' \longrightarrow\!\!\!\!\!\rightarrow_{\Sigma_{\text{CIL}}} v$. Then by lemmas 3.25, 5.23, and 3.45, for $V' \equiv \mathcal{T}^{-1}_{\text{CIL}}(v') \in \text{Value}_{\lambda^{\text{CIL}}}$, $M \longmapsto\!\!\!\!\!\rightarrow_{\lambda^{\text{CIL}}} V' \longrightarrow\!\!\!\!\!\rightarrow_{\lambda^{\text{CIL}}} V$.

□

6 Conclusion

This paper has presented two worked examples showing how to use the general standardization and evaluation framework presented in [WM00]. The first major contribution of this paper is to show in detail for the two examples how to correctly define a CRS exactly corresponding to a programming language calculus. As a side benefit of this, the two calculi are proven confluent. The second major contribution is to show that the conditions of the general framework are met thereby (1) automatically deriving notions of evaluation for the calculi and (2) proving that the two calculi have the desired standardization properties. The result guarantees that the calculi can be safely used for meaning-preserving program transformations.

References

[AF97] Zena M. Ariola and Matthias Felleisen. The call-by-need lambda calculus. *J. Funct. Prog.*, 3(7), May 1997.

[ALP96] *Proc. 5th Int'l Conf. Algebraic & Logic Programming*, 1996.

[AM96] Sergio Antoy and Aart Middeldorp. A sequential reduction strategy. *Theor. Comp. Sc.*, 165(1):75–95, 1996.

[BKKS87] H. P. Barendregt, J. R. Kennaway, Jan Willem Klop, and M. R. Sleep. Needed reduction and spine strategies for the lambda calculus. *Inf. & Comput.*, 75(3):191–231, 1987.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[CF58] H. B. Curry and R. Feys. *Combinatory Logic I.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1958.

[DMTW97] Allyn Dimock, Robert Muller, Franklyn Turbak, and J. B. Wells. Strongly typed flow-directed representation transformations. In *Proc. 1997 Int'l Conf. Functional Programming*, pages 11–24. ACM Press, 1997.

[Fel88] Matthias Felleisen. The theory and practice of first-class prompts. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 180–190, San Diego, California, January 1988.

[FF89] M. Felleisen and D. P. Friedman. A syntactic theory of sequential state. *Theor. Comp. Sc.*, 69(3):243–287, 1989.

[FH92] Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theor. Comp. Sc.*, 102:235–271, 1992.

[GK] John Glauert and Zurab Khasidashvili. Minimal and optimal relative normalization in orthogonal expression reduction systems. Presented at the Sept. 1996 Glasgow Int'l School on Type Theory & Term Rewriting. A refereed version is [GKK00].

[GKK00] John Glauert, Richard Kenneway, and Zurab Khasidashvili. Minimal and optimal relative normalization in orthogonal expression reduction systems. *J. Logic & Comput.*, 10(3), 2000. Special issue on Type Theory and Term Rewriting.

[GLM92] Georges Gonthier, Jean-Jacques Lévy, and Paul-André Melliès. An abstract standardisation theorem. In *Proc. 7th Ann. IEEE Symp. Logic in Computer Sci.*, 1992.

[HL91a] Gérard Huet and Jean-Jacques Lévy. Computations in orthogonal rewriting systems, I. In Lassez and Plotkin [LP91], pages 395–414.

[HL91b] Gérard Huet and Jean-Jacques Lévy. Computations in orthogonal rewriting systems, II. In Lassez and Plotkin [LP91], pages 415–443.

[HP99] M. Hanus and C. Prehofer. Higher-order narrowing with definitional trees. *Journal of Functional Programming*, 9(1):33–75, 1999.

[JM96] Trevor Jim and Albert R. Meyer. Full abstraction and the context lemma. *SIAM Journal of Computing*, 25(3):663–696, June 1996.

[Kah94] Stefan Kahrs. Compilation of combinatory reduction systems. In *Proc. 1st Int'l Workshop Higher-Order Algebra, Logic, & Term Rewriting*, 1994.

[Ken89] J. R. Kennaway. Sequential evaluation strategies for parallel-or and related reduction systems. *Ann. Pure & Appl. Logic*, 43:31–56, 1989.

[KG96] Zurab Khasidashvili and John Glauert. Discrete normalization and standardization in deterministic residual structures. In ALP '96 [ALP96], pages 135–149.

[Kha90] Zurab Khasidashvili. Expression reduction systems. In *Proceedings of I. Vekua Institute of Applied Mathematics of Tbilisi State University*, volume 36, pages 200–220. 1990. Technical report.

[Klo80] Jan Willem Klop. *Combinatory Reduction Systems*. Mathematisch Centrum, Amsterdam, 1980. Ph.D. Thesis.

[KM91] Jan Willem Klop and Aart Middeldorp. Sequentiality in orthogonal term rewriting systems. *J. Symbolic Comp.*, 12:161–195, 1991.

[KvO95] Zurab Khasidashvili and Vincent van Oostrom. Context-sensitive conditional expression reduction systems. In *Proc. Int'l Workshop Graph Rewriting and Computation, SEGRAGRA '95*, Elec. Notes Comp. Sci., pages 141–150. Elsevier Science B.V., August 1995.

[KvOvR93] Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theor. Comp. Sc.*, 121(1–2):279–308, 1993.

[LP91] Jean-Louis Lassez and Gordon Plotkin, editors. *Computational Logic: Essays in Honor of Alan Robinson*. MIT Press, 1991.

[Mar91] Luc Maranget. Optimal derivations in orthogonal term rerwiting systems and in weak lambda calculi. In *Conf. Rec. 18th Ann. ACM Symp. Princ. of Prog. Langs.*, January 1991.

[Mel98] Paul-André Melliès. Axiomatic Rewriting Theory IV: A stability theorem in Rewriting Theory. In *Proc. 13th Ann. IEEE Symp. Logic in Computer Sci.*, pages 287–298, 1998.

[Mid97] Aart Middeldorp. Call by need computations to root-stable form. In *Conf. Rec. POPL '97: 24th ACM Symp. Princ. of Prog. Langs.*, pages 94–105, 1997.

[Mul92] R. Muller. M-LISP: A representation-independent dialect of LISP with reduction semantics. *ACM Trans. on Prog. Langs. and Systs.*, 14(4):589–615, 1992.

[Nip91] Tobias Nipkow. Higher-order critical pairs. In *Proc. 6th Ann. IEEE Symp. Logic in Computer Sci.*, pages 342–349, 1991.

[Nöc94] E. Nöcker. *Efficient Functional Programming: Compilation and Programming Techniques*. PhD thesis, Katholic University of Nijmegen, 1994.

[Plo75] Gordon D. Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theor. Comp. Sc.*, 1:125–159, 1975.

[SF93] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3/4):289–360, 1993.

[SR93] R. C. Sekar and I. V. Ramakrishnan. Programming in equational logic: Beyond strong sequentiality. *Inf. & Comput.*, 104(1):78–109, 1993.

[Suz96] Taro Suzuki. Standardization theorem revisited. In ALP '96 [ALP96], pages 122–134.

[Tak93] M. Takahashi. λ-calculi with conditional rules. In *Proc. Int'l Conf. Typed Lambda Calculi and Applications*, 1993.

[Tay99] Paul Taylor. *Practical Foundations of Mathematics*. Cambridge University Press, 1999.

[vO94] Vincent van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit Amsterdam, 1994.

[vO96] Vincent van Oostrom. Higher-order families. In *Proc. 7th Int'l Conf. Rewriting Techniques and Applications*, 1996.

[vR96] Femke van Raamsdonk. *Confluence and Normalisation for Higher-Order Rewriting*. PhD thesis, Vrije Universiteit Amsterdam, 1996.

[WDMT97] J. B. Wells, Allyn Dimock, Robert Muller, and Franklyn Turbak. A typed intermediate language for flow-directed compilation. In *Proc. 7th Int'l Joint Conf. Theory & Practice of Software Development*, pages 757–771, 1997. Superseded by [WDMT0X].

[WDMT0X] J. B. Wells, Allyn Dimock, Robert Muller, and Franklyn Turbak. A calculus with polymorphic and polyvariant flow types. *J. Funct. Prog.*, 200X. Accepted subject to revisions. Supersedes [WDMT97].

[WM00] J. B. Wells and Robert Muller. Standardization and evaluation in combinatory reduction systems. Unpublished draft to be submitted, 2000.

[Wol93] D. A. Wolfram. *The Clausal Theory of Types*, volume 21 of *Cambridge Tracts in Theoretical Comp. Sci.* Cambridge University Press, 1993.

Appendix A

Figure 7 contains copies of definitions from sections 4.1 and 4.3. They are repeated here together in one place for ease of reference.

$\mathcal{G}(\Sigma,\Theta)(s) = \mathrm{NextOrder}^k_{\Sigma,\Theta,s}([])$ where $k = |\mathrm{Skel}(s)|$ and

$$
\begin{array}{rcl}
\mathrm{NextOrder}_{\Sigma,\Theta,s}(\delta) & = & \begin{cases} [\vec{p}^{\,i}, p_{i+1}] & \text{if } \delta = [\vec{p}^{\,i}] \neq \mathrm{wrong} \text{ and } \mathrm{NextPos}_{\Sigma,\Theta}([\vec{p}^{\,i}], s) = p_{i+1} \neq \mathrm{wrong}, \\ \mathrm{wrong} & \text{otherwise}, \end{cases} \\
\mathrm{NextPos}_{\Sigma,\Theta}(\gamma, s) & = & \begin{cases} \Theta(\mathrm{Opt}_\Sigma(\gamma,s), \mathrm{Tree}(s)\downarrow\gamma) & \text{if } \mathrm{Opt}_\Sigma(\gamma,s) \neq \mathrm{wrong}, \\ \mathrm{wrong} & \text{otherwise}, \end{cases} \\
\mathrm{Unex}(\gamma, s) & = & \min_{\leq}(\mathrm{Skel}(s) \setminus \gamma) \\
\mathrm{Disc}_\Sigma(\gamma, s) & = & \{\, (s.p, r) \mid r \in \mathrm{Red}(\Sigma), \forall q \in \mathrm{Int}(r). \\
 & & \qquad p \cdot q \in \gamma \text{ and } \mathrm{Tree}(s)(p \cdot q) = \mathrm{Tree}(\mathrm{LHS}(r))(q) \,\} \\
\mathrm{Inv}_\Sigma(\gamma, s) & = & \{\, p \mid (s.q, r) \in \mathrm{Disc}_\Sigma(\gamma, s), q' \in \mathrm{Int}(r), p \leq q \cdot q' \,\} \\
\mathrm{Poss}_\Sigma(\gamma, s) & = & \{\, (s.p, s' \to t) \mid p \in \gamma \setminus \mathrm{Inv}_\Sigma(\gamma, s), s' \to t \in \mathrm{Red}(\Sigma), \\
 & & \qquad \forall q \in \mathrm{Int}(s').\ p \cdot q \in \gamma \Rightarrow \mathrm{Tree}(s)(p \cdot q) = \mathrm{Tree}(s')(q) \,\} \\
\mathrm{PossUnex}(s.p, r) & = & \{\, p \cdot q \mid q \in \mathrm{Int}(r), p \cdot q \in \mathrm{Unex}(\gamma, s) \,\} \\
\mathrm{Mand}_\Sigma(\gamma, s) & = & \bigcap_{(s.p,r) \in \mathrm{Poss}_\Sigma(\gamma,s)} \mathrm{PossUnex}(s.p, r) \\
\mathrm{Opt}_\Sigma(\gamma, s) & = & \begin{cases} \mathrm{Unex}(\gamma, s) & \text{if } \mathrm{Poss}_\Sigma(\gamma,s) = \varnothing, \\ \mathrm{Mand}_\Sigma(\gamma, s) & \text{if } \mathrm{Poss}_\Sigma(\gamma,s) \neq \varnothing \text{ and } \mathrm{Mand}_\Sigma(\gamma,s) \neq \varnothing, \\ \mathrm{wrong} & \text{if } \mathrm{Poss}_\Sigma(\gamma,s) \neq \varnothing \text{ and } \mathrm{Mand}_\Sigma(\gamma,s) = \varnothing \end{cases}
\end{array}
$$

$$
(C^{(n)}, \langle \vec{w}^n \rangle) \in \mathrm{Decomp}(w, P) \iff \begin{cases} C[\vec{w}] \equiv w \\ \text{and } \mathrm{Skel}(C) = \{\, q \in \mathrm{Skel}(w) \mid \nexists p \in P.\, p < q \,\} \\ \text{and } \mathrm{Tree}(C)(P \cap \mathrm{Skel}(C)) = \{\Box\} \end{cases}
$$

$$
\begin{array}{l}
\mathrm{ValPatt}(\Sigma) = \min_{\sqsubseteq}\{\, s_i \mid s \to t \in \mathrm{Red}(\Sigma), s \equiv F(\vec{s}^n), 1 \leq i \leq n \,\} \\
\mathrm{Val}(\Sigma) \quad = \{\, \nu(s) \mid s \in \mathrm{ValPatt}(\Sigma),\ \nu \text{ is a valuation for } s \,\}
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{CtxtsPos}(s, p, P) & = & \{\, C' \mid (C^{(n)}, \chi) \in \mathrm{Decomp}(s, \{p\} \cup P),\ \vec{u}^n \in \mathrm{Ter},\ (C', \chi') \in \mathrm{Decomp}(C[\vec{u}], \{p\}) \,\} \\
\mathrm{ContextsMT}_\Sigma(s) & = & \{C \mid \ \mathrm{ValOrder}(\Sigma)(s) = [\vec{p}^n], 2 \leq i \leq |\mathrm{Int}(s)|, \\
 & & \qquad C \in \mathrm{CtxtsPos}(s, p_i, \min_{\leq}(\{p_{i+1}, \ldots, p_n\} \cup \{\, p \mid \mathrm{Tree}(s)(p) \in \mathrm{MVar} \,\}))\} \\
\mathcal{E}_{\mathrm{red}}(\Sigma) & = & \bigcup_{s \to t \in \mathrm{Red}(\Sigma)} \mathrm{ContextsMT}_\Sigma(s) \\
\mathcal{E}_{\mathrm{val}}(\Sigma) & = & \bigcup_{s \in \mathrm{ValPatt}(\Sigma)} \mathrm{ContextsMT}_\Sigma(s) \cup \{\Box\} \\
\mathrm{EvalCont}(\Sigma) & = & \{\, C[C_1[\cdots[C_n]\cdots]] \mid C \in \mathcal{E}_{\mathrm{val}}(\Sigma), \vec{C}^n \in \mathcal{E}_{\mathrm{red}}(\Sigma) \,\}
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{PossVal}_\Sigma(f) & = & \{s \in \mathrm{ValPatt}(\Sigma) \mid \exists p' \in \mathrm{Int}(s).f(p') \text{ is undefined}, \\
 & & \qquad \forall p \in \mathrm{Int}(s).f(p) \text{ defined } \Rightarrow f(p) = \mathrm{Tree}(s)(p)\}, \\
\mathrm{MandVal}_\Sigma(P, f) & = & P \cap (\textstyle\bigcap_{s \in \mathrm{PossVal}_\Sigma(f)} \mathrm{Int}(s)), \\
\mathrm{ValChoose}(\Sigma)(P, f) & = & \begin{cases} \min_{\mathrm{lex}}(P) & \text{if } \mathrm{PossVal}_\Sigma(f) = \varnothing, \\ \min_{\mathrm{lex}}(\mathrm{MandVal}_\Sigma(P, f)) & \text{if } \mathrm{PossVal}_\Sigma(f) \neq \varnothing \text{ and } \mathrm{MandVal}_\Sigma(P, f) \neq \varnothing, \\ \mathrm{wrong} & \text{if } \mathrm{PossVal}_\Sigma(f) \neq \varnothing \text{ and } \mathrm{MandVal}_\Sigma(P, f) = \varnothing. \end{cases} \\
\mathrm{ValOrder}(\Sigma) & = & \mathcal{G}(\Sigma, \mathrm{ValChoose}(\Sigma))
\end{array}
$$

Figure 7: The generic subterm ordering function generator and related functions.