Sequent Calculus as a Compiler Intermediate Language

Appendix

Paul Downen Luke Maurer
Zena M. Ariola
University of Oregon, USA
{pdownen,maurerl,ariola}@cs.uoregon.edu

Simon Peyton Jones
Microsoft Research Cambridge, UK
simonpj@microsoft.com

A. Benchmarks

Tables 1 and 2 show the results of the `spectral` and `real` NoFib tests for GHC 7.8.4 modified to use the new Sequent Core version of the simplifier, versus the baseline GHC 7.8.4. There are wins and losses; the losses are relatively few but serious (most notably `spectral/rewrite` and `real/cacheprof`).

It is difficult to glean much from the details, largely because rewriting the simplifier with a new intermediate representation is such a drastic change. We hope to use the Sequent Core experience to make more modest changes to the original simplifier, for which it should be easier to tease out the effects of particular changes.

B. Contification Algorithm

The algorithm A is shown in Figure 9. At each command or continuation, the traversal produces a triple $(\mathcal{F}, \mathcal{G}, \mathcal{C})$ of a *free set* $\mathcal{F}$, a *good set* $\mathcal{G}$, and a *contifiable set* $\mathcal{C}$, with $\mathcal{G} \subseteq \mathcal{F}$ and $\mathcal{C} \cap \mathcal{F} = \emptyset$. The free set contains the variables occurring free in the command; the good set contains just the "good" ones, that is, those variables that only occur free as tail-called functions; and the contifiable set contains the functions marked for contification. We assume here that all binders are distinct.[1] For terms, the procedure is the same, except that only $\mathcal{F}$ and $\mathcal{C}$ are returned—since terms are continuation-closed, *no* function occurring free in a term can be contified, so the good set for a term is always empty.

At each binding $\mathbf{let}\ f = v\ \mathbf{in}\ c$, if $A\,[\![c]\!] = (\mathcal{F}, \mathcal{G}, \mathcal{C})$, we contify f (that is, add it to $\mathcal{C}$) if and only if $f \in \mathcal{G}$. For recursive bindings, the procedure is the same, only of course the combined analysis for the body and the definitions must be used.

The definition of $\oplus$ says that, in an expression with two subexpressions, the good variables are those that are

- good on the left and absent on the right, or
- good on the right and absent on the left, or
- good on both sides.

Alternatively, we could track the free set and the *bad* set, and then $\oplus$ would simply take the unions. Using the good set makes the algoritm more flexible, however; many extensions require tracking something about the calls to each function, such as the arity, which is easy if the good set is represented as the domain of a finite map.

C. Proof of Correspondence (Proposition 1)

The proof is by bisimulation. After establishing some reduction relations and their algebraic properties, we will define a readback function, use it to define a bisimulation, then prove that the bisimulation preserves termination.

C.1 Reduction

We define $\to$ as the compatible closure of $\mapsto$ along with one additional rule. Note that while $\mapsto$ only relates commands, $\to$ extends to terms and continuations as well.

The new rule is a form of η-rule:

$$\mu\mathsf{ret}.\langle v \,\|\, \mathsf{ret}\rangle \to v \qquad (\mu_\eta)$$

Applying the μ_η-rule does not affect observable behavior, but it will be necessary for relating the two calculi. Note that it is never a standard reduction (unless it happens to coincide with a standard μ-reduction).

We now call $\mapsto$ *standard reduction*. Accordingly, $\to$ includes *non-standard reduction*; to denote non-standard reduction specifically, we write $\rightarrowtail$. Hence $\to$ is the disjoint union of $\mapsto$ and $\rightarrowtail$.

To prove standardization, we will also make use of a *parallel reduction* relation $\Rightarrow$. Parallel reduction consists of the simultaneous reduction of some number of redexes, possibly zero, appearing in the same term. Clearly, parallel reduction sits between reduction and its reflexive-transitive closure:

$$\to\ \subset\ \Rightarrow\ \subset\ \to^\star$$

Finally, we have *non-standard parallel reduction*, $\ggg$, which may contract several redexes but *not* the standard redex.

C.2 Algebraic Properties

We will need a few algebraic properties of the CBN calculus. Most important is standardization (Proposition 8).

An easy property is that irreducibility is preserved by general reduction, and unaffected by non-standard reduction:

Lemma 5. *If $c \to c'$ then $c \not\mapsto$ implies $c' \not\mapsto$. Furthermore, if $c \rightarrowtail c'$ then $c \not\mapsto$ iff $c' \not\mapsto$.*

Proof. The second property (if $c \rightarrowtail c'$ then $c \not\mapsto$ iff $c' \not\mapsto$) can be shown by cases on the possible non-standard reductions, since a non-standard reduction never introduces or destroys a standard redex. The first property is implied by the second because when a command without a standard step is reduced, that reduction must have been non-standard to begin with. ☐

Also relatively simple is a standard substitution lemma, though there are fine points to its statement.

Lemma 6 (Substitution). *1. If $c \to c'$ then $c\,\{\sigma/a\} \to c'\,\{\sigma/a\}$.*
2. If $c \to c'$ then $c\,\{v/x\} \to c'\,\{v/x\}$.
3. If $c \to c'$ then $c\,\{k/\mathsf{ret}\} \to c'\,\{k/\mathsf{ret}\}$.

[1] The actual code annotates the binders rather than gathering a set, so it avoids making this assumption.

Contification analysis of terms: $\boxed{(\mathcal{F},\mathcal{C}) = A\,[\![v]\!]}$

$$
\begin{aligned}
A\,[\![x]\!] &= (\{x\},\emptyset) \\
A\,[\![\lambda x{:}\tau.v]\!] &= (\mathcal{F}\setminus\{x\},\mathcal{C}) && \text{where } (\mathcal{F},\mathcal{C}) = A\,[\![v]\!] \\
A\,[\![\Lambda a{:}\kappa.v]\!] &= A\,[\![v]\!] \\
A\,[\![K(\vec{\sigma},\vec{v})]\!] &= (\bigcup\vec{\mathcal{F}},\bigcup\vec{\mathcal{C}}) && \text{where } \overrightarrow{(\mathcal{F},\mathcal{C})} = \overrightarrow{A\,[\![v]\!]} \\
A\,[\![\mu\mathsf{ret}.c]\!] &= (\mathcal{F},\mathcal{C}) && \text{where } (\mathcal{F},\emptyset,\mathcal{C}) = A\,[\![c]\!]
\end{aligned}
$$

Contification analysis of continuations: $\boxed{(\mathcal{F},\mathcal{G},\mathcal{C}) = A\,[\![k]\!]}$

$$
\begin{aligned}
A\,[\![v\cdot k]\!] &= (\mathcal{F},\emptyset,\mathcal{C})\oplus A\,[\![k]\!] && \text{where } (\mathcal{F},\mathcal{C}) = A\,[\![v]\!] \\
A\,[\![\sigma\cdot k]\!] &= A\,[\![k]\!] \\
A\,[\![\mathsf{ret}]\!] &= (\emptyset,\emptyset,\emptyset) \\
A\,\left[\!\!\left[\mathbf{case\,of}\ \overrightarrow{alt}\right]\!\!\right] &= \bigoplus\overrightarrow{A\,[\![alt]\!]}
\end{aligned}
$$

Contification analysis of commands: $\boxed{(\mathcal{F},\mathcal{G},\mathcal{C}) = A\,[\![c]\!]}$

$$
\begin{aligned}
A\,[\![\mathbf{let}\ bind\ \mathbf{in}\ c]\!] &= A\,[\![bind]\!]_{A[\![c]\!]} \\
A\,[\![\langle v \parallel k\rangle]\!] &= (\mathcal{F},\mathcal{G},\mathcal{C})\oplus A\,[\![k]\!] && \text{where } (\mathcal{F},\mathcal{C}) = A\,[\![v]\!] \\
& && \mathcal{G} = \begin{cases}\{f\} & \text{if } v = f, k = \vec{v'}\cdot\mathsf{ret}, |\vec{v'}| = arity(f) \\ \emptyset & \text{otherwise}\end{cases} \\
A\,[\![\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}]\!] &= (\bigcup\vec{\mathcal{F}},\bigcup\vec{\mathcal{C}}) && \text{where } \overrightarrow{(\mathcal{F},\mathcal{C})} = \overrightarrow{A\,[\![v]\!]}
\end{aligned}
$$

Contification analysis of bindings: $\boxed{(\mathcal{F},\mathcal{G},\mathcal{C}) = A\,[\![bind]\!]_{(\mathcal{F}_b,\mathcal{G}_b,\mathcal{C}_b)}}$

$$
\begin{aligned}
A\,[\![f{:}\tau = v]\!]_{(\mathcal{F}_b,\mathcal{G}_b,\mathcal{C}_b)} &= (\mathcal{F}'\setminus\{f\},\mathcal{G}'\setminus\{f\},\mathcal{C}'') \\
&\text{where}\quad (\mathcal{F},\mathcal{C}) = A\,[\![v]\!] \\
&\qquad (\mathcal{F}',\mathcal{G}',\mathcal{C}') = (\mathcal{F},\emptyset,\mathcal{C})\oplus(\mathcal{F}_b,\mathcal{G}_b,\mathcal{C}_b) \\
&\qquad \mathcal{C}'' = \mathcal{C}'\cup(\{f\}\cap\mathcal{G}_b) \\
A\,[\![j{:}\tau = \tilde{\mu}[\overrightarrow{a{:}\kappa},\overrightarrow{x{:}\sigma}].c]\!]_{(\mathcal{F}_b,\mathcal{G}_b,\mathcal{C}_b)} &= (\mathcal{F}\setminus\{\vec{x}\},\mathcal{G}\setminus\{\vec{x}\},\mathcal{C})\oplus(\mathcal{F}_b,\mathcal{G}_b,\mathcal{C}_b) \\
&\text{where } (\mathcal{F},\mathcal{G},\mathcal{C}) = A\,[\![c]\!] \\
A\,\left[\!\!\left[\mathbf{rec}\left\{\overrightarrow{f{:}\tau = v}\right\}\right]\!\!\right]_{(\mathcal{F}_b,\mathcal{G}_b,\mathcal{C}_b)} &= (\mathcal{F}''\setminus\{\vec{f}\},\mathcal{G}'\setminus\{\vec{f}\},\mathcal{C}''') \\
&\text{where}\quad \overrightarrow{(\mathcal{F},\mathcal{C})} = \overrightarrow{A\,[\![v]\!]} \\
&\qquad (\mathcal{F}'',\mathcal{G}',\mathcal{C}'') = (\bigcup\vec{\mathcal{F}},\emptyset,\bigcup\vec{\mathcal{C}})\oplus(\mathcal{F}_b,\mathcal{G}_b,\mathcal{C}_b) \\
&\qquad \mathcal{C}''' = \begin{cases}\mathcal{C}''\cup\mathcal{G}' & \text{if } \{\vec{f}\}\subseteq\mathcal{G}' \\ \mathcal{C}'' & \text{otherwise}\end{cases} \\
A\,\left[\!\!\left[\mathbf{rec}\left\{\overrightarrow{j{:}\tau = \tilde{\mu}[\overrightarrow{a{:}\kappa},\overrightarrow{x{:}\sigma}].c}\right\}\right]\!\!\right]_{(\mathcal{F}_b,\mathcal{G}_b,\mathcal{C}_b)} &= (\mathcal{F}',\mathcal{G}',\mathcal{C}') \\
&\text{where}\quad \overrightarrow{(\mathcal{F},\mathcal{G},\mathcal{C})} = \overrightarrow{A\,[\![c]\!]} \\
&\qquad (\mathcal{F}',\mathcal{G}',\mathcal{C}') = \left(\bigoplus\overrightarrow{(\mathcal{F}\setminus\{\vec{x}\},\mathcal{G}\setminus\{\vec{x}\},\mathcal{C})}\right)\oplus(\mathcal{F}_b,\mathcal{G}_b,\mathcal{C}_b)
\end{aligned}
$$

Contification analysis of alternatives: $\boxed{(\mathcal{F},\mathcal{G},\mathcal{C}) = A\,[\![alt]\!]}$

$$
\begin{aligned}
A\,[\![x{:}\tau\to c]\!] &= (\mathcal{F}\setminus\{x\},\mathcal{G}\setminus\{x\},\mathcal{C}) && \text{where } (\mathcal{F},\mathcal{G},\mathcal{C}) = A\,[\![c]\!] \\
A\,[\![K(\overrightarrow{a{:}\kappa},\overrightarrow{x{:}\tau})\to c]\!] &= (\mathcal{F}\setminus\{\vec{x}\},\mathcal{G}\setminus\{\vec{x}\},\mathcal{C}) && \text{where } (\mathcal{F},\mathcal{G},\mathcal{C}) = A\,[\![c]\!]
\end{aligned}
$$

Combination of contification analyses: $\boxed{(\mathcal{F}',\mathcal{G}',\mathcal{C}') = (\mathcal{F}_1,\mathcal{G}_1,\mathcal{C}_1)\oplus(\mathcal{F}_2,\mathcal{G}_2,\mathcal{C}_2)}$

$$
(\mathcal{F}_1,\mathcal{G}_1,\mathcal{C}_1)\oplus(\mathcal{F}_2,\mathcal{G}_2,\mathcal{C}_2) = (\mathcal{F}_1\cup\mathcal{F}_2,\ (\mathcal{G}_1\setminus\mathcal{F}_2)\cup(\mathcal{G}_2\setminus\mathcal{F}_1)\cup(\mathcal{G}_1\cap\mathcal{G}_2),\mathcal{C}_1\cup\mathcal{C}_2)
$$

Figure 9. The analysis phase A of the contification pass, including the operator $\oplus$ for combining analyses.

Test	Size	Allocs	Time	Elapsed	Memory
ansi	-0.0%	-11.8%	0.000	0.000	0.0%
atom	-0.0%	0.0%	+0.9%	+0.9%	0.0%
awards	0.0%	0.0%	0.000	0.000	0.0%
banner	0.0%	0.0%	0.000	0.000	0.0%
boyer	+0.0%	0.0%	0.020	0.020	0.0%
boyer2	-0.3%	+5.2%	0.000	0.000	0.0%
calendar	-0.0%	-0.6%	0.000	0.000	0.0%
cichelli	+0.2%	+2.3%	0.040	0.040	0.0%
circsim	+0.0%	-0.1%	-5.5%	-5.5%	+5.9%
clausify	+0.0%	0.0%	0.020	0.020	0.0%
comp_lab_zift	-0.0%	+0.1%	0.100	0.100	+14.3%
constraints	-0.0%	-2.9%	-6.3%	-6.2%	0.0%
cryptarithm1	0.0%	0.0%	-0.7%	-0.7%	0.0%
cryptarithm2	-0.2%	+0.0%	0.010	0.010	0.0%
cse	-0.1%	-1.0%	0.000	0.000	0.0%
eliza	-0.0%	-1.8%	0.000	0.000	0.0%
event	-0.0%	-2.2%	0.074	0.074	0.0%
expert	+0.0%	-0.6%	0.000	0.000	0.0%
fft	+0.1%	+1.3%	0.020	0.020	-10.0%
fft2	-0.0%	+0.1%	0.030	0.030	0.0%
fibheaps	0.0%	0.0%	0.020	0.020	0.0%
fish	0.0%	0.0%	0.010	0.010	0.0%
gcd	-0.0%	0.0%	0.016	0.016	0.0%
genfft	-0.0%	-0.0%	0.020	0.020	0.0%
ida	+0.0%	+1.0%	0.050	0.050	0.0%
integer	+0.0%	0.0%	-0.7%	-0.9%	0.0%
knights	0.0%	-0.0%	0.000	0.000	0.0%
lcss	-0.0%	-0.0%	-2.6%	-2.6%	0.0%
life	-0.0%	-0.0%	0.140	0.140	0.0%
listcompr	+0.0%	+0.0%	0.050	0.050	0.0%
listcopy	+0.0%	+0.0%	0.050	0.050	0.0%
mandel	-0.0%	-0.0%	0.030	0.030	0.0%
mandel2	-0.0%	-0.0%	0.000	0.000	0.0%
minimax	+0.0%	+0.0%	0.000	0.000	0.0%
multiplier	+0.0%	-3.1%	0.070	0.070	0.0%
nucleic2	0.0%	0.0%	0.030	0.030	0.0%
para	+0.3%	-2.7%	0.162	0.162	0.0%
parstof	+0.0%	-0.3%	0.000	0.000	0.0%
power	+0.1%	-0.0%	-3.1%	-3.9%	0.0%
pretty	-0.0%	+0.0%	0.000	0.000	0.0%
primetest	-0.0%	-0.0%	0.056	0.056	0.0%
puzzle	-0.0%	-17.1%	0.082	0.082	0.0%
rewrite	+0.0%	+19.2%	0.010	0.010	0.0%
scc	0.0%	0.0%	0.000	0.000	0.0%
sched	0.0%	0.0%	0.010	0.010	0.0%
simple	-0.6%	-4.9%	0.150	0.150	+3.4%
solid	-0.0%	0.0%	0.080	0.080	0.0%
sorting	0.0%	0.0%	0.000	0.000	0.0%
sphere	+0.0%	0.0%	0.022	0.022	0.0%
transform	-0.2%	-0.6%	0.198	0.200	0.0%
treejoin	+0.0%	0.0%	0.090	0.090	0.0%
typecheck	-0.0%	-0.0%	0.132	0.134	0.0%
wang	-0.0%	0.0%	0.064	0.064	+5.0%
wave4main	+0.0%	-0.0%	0.130	0.130	0.0%
Min	-0.6%	-17.1%	-6.3%	-6.2%	-10.0%
Max	+0.3%	+19.2%	+0.9%	+0.9%	+14.3%
Geometric Mean	-0.0%	-0.5%	-2.6%	-2.7%	+0.3%

Table 1. Results for the `spectral` NoFib tests.

Test	Size	Allocs	Time	Elapsed	Memory
anna	-0.1%	+1.2%	0.152	0.152	0.0%
bspt	+1.8%	0.0%	0.020	0.020	0.0%
cacheprof	+0.5%	+19.7%	+10.9%	+10.9%	+4.5%
compress	+0.1%	0.0%	-0.6%	-0.6%	0.0%
compress2	-0.2%	0.0%	+0.5%	+0.5%	0.0%
fem	+0.1%	-0.1%	0.046	0.046	0.0%
fluid	+0.4%	+0.1%	0.010	0.010	0.0%
fulsom	+0.7%	-8.8%	-4.9%	-4.9%	-7.1%
gamteb	-0.1%	+0.1%	0.056	0.056	0.0%
gg	-0.1%	+2.3%	0.028	0.028	0.0%
grep	+0.0%	0.0%	0.000	0.000	0.0%
hidden	-0.2%	-0.9%	+0.7%	+1.0%	0.0%
hpg	-0.1%	-0.2%	0.132	0.132	0.0%
infer	-0.5%	-0.1%	0.100	0.100	0.0%
lift	-0.2%	-1.2%	0.000	0.000	0.0%
maillist	+0.0%	-0.0%	0.060	0.060	+2.6%
mkhprog	-0.1%	-0.0%	0.000	0.000	0.0%
parser	-0.2%	+2.1%	0.050	0.050	0.0%
pic	-0.0%	-0.7%	0.010	0.010	0.0%
prolog	+0.2%	+0.3%	0.000	0.000	0.0%
reptile	+0.1%	+0.0%	0.022	0.022	0.0%
rsa	-0.0%	-0.0%	0.010	0.010	0.0%
scs	+0.2%	-0.6%	-0.0%	-0.0%	0.0%
symalg	-0.2%	-0.0%	0.000	0.000	0.0%
veritas	+0.4%	-0.1%	0.000	0.000	0.0%
Min	-0.5%	-8.8%	-4.9%	-4.9%	-7.1%
Max	+1.8%	+19.7%	+10.9%	+10.9%	+4.5%
Geometric Mean	+0.1%	+0.4%	+1.0%	+1.0%	-0.0%

Table 2. Results for the `real` NoFib tests.

4. *If* $c \to c'$ *then* $c\left\{c''\overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\right\} \to^\star c'\left\{c''\overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\right\}$.
5. *If* $v \to v'$ *then* $c\{v/x\} \to^\star c\{v'/x\}$.
6. *If* $k \to k'$ *then* $c\{k/\mathsf{ret}\} \to^\star c\{k'/\mathsf{ret}\}$.
7. *If* $c' \to c''$ *then* $c\left\{c'\overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\right\} \to^\star c\left\{c''\overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\right\}$.

Proof. By mutual induction. The one subtlety is that substituting a type, term, or continuation into a command cannot duplicate or destroy redexes in the original command, but substituting a join point can. Hence clauses 1–3 specify $\to$ but clause 4 specifies $\to^\star$.

Here is the crucial case of clause 4: Suppose $c \equiv \mathsf{jump}\ j\ \vec{\sigma'}\ \vec{v'}$. Since c cannot take a standard reduction, the reduction must occur in some subterm. Hence $\vec{v'} \to \vec{v''}$ and $c' \equiv \mathsf{jump}\ j\ \vec{\sigma'}\ \vec{v''}$. Now:

$$\begin{aligned} & c\left\{c''\overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\right\} \\ &\quad \equiv c''\overrightarrow{\{\sigma'/a\}}\overrightarrow{\{v'/x\}} \\ &\quad \to^\star c''\overrightarrow{\{\sigma'/a\}}\overrightarrow{\{v''/x\}} && \text{(by 5)} \\ &\quad \equiv c'\left\{c''\overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\right\} \end{aligned}$$ □

Parallel reduction enjoys a similar substitution lemma (in fact, it is why parallel reduction is useful!).

Lemma 7. *Let z denote any term, continuation, or command, and likewise z'.*

1. *If* $z \Rightarrow z'$, *then* $z\{\tau/a\} \Rightarrow z'\{\tau/a\}$.
2. *If* $z \Rightarrow z'$ *and* $v \Rightarrow v'$, *then* $z\{v/x\} \Rightarrow z'\{v'/x\}$.
3. *If* $z \Rightarrow z'$ *and* $k \Rightarrow k'$, *then* $z\{k/\mathsf{ret}\} \Rightarrow z'\{k/\mathsf{ret}\}$.

$$\begin{tikzcd} c \arrow[r, dashed, mapsto, "\star"] \arrow[d, tail, "\star"'] & c' \arrow[d, tail, dashed, "\star"] \\ d \arrow[r, mapsto, "\star"] & d' \end{tikzcd}$$

Figure 10. Core of standardization proof.

4. If $z \Rightarrow z'$ and $c \Rightarrow c'$, then $z\left\{\overrightarrow{c\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\right\} \Rightarrow z'\left\{\overrightarrow{c'\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\right\}$.

Proof. Each proceeds by induction, with 4 relying on 1 and 2. We show the crucial case in 4.

Suppose $z \equiv \mathsf{jump}\ j\ \vec{\sigma'}\ \vec{v'}$. Hence $\overrightarrow{v' \Rightarrow v''}$ and $z' \equiv \mathsf{jump}\ j\ \vec{\sigma'}\ \vec{v''}$. Then:

$$\begin{aligned} & z\left\{\overrightarrow{c\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\right\} \\ &\equiv c\overrightarrow{\{\sigma'/a\}}\overrightarrow{\{v'/x\}} \\ &\Rightarrow c'\overrightarrow{\{\sigma'/a\}}\overrightarrow{\{v''/x\}} && \text{(by 1 and 2)} \\ &\equiv z'\left\{\overrightarrow{c'\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\right\} && \square \end{aligned}$$

Now we have the tools to prove standardization:

Proposition 8 (Standardization)**.** *If $c \rightarrow^\star c' \not\mapsto$ then $c \mapsto^\star c_1 \not\mapsto$ and $c_1 \rightarrow^\star c'$.*

We will prove Proposition 8 in several parts. The crux is the diagram in Fig. 10, which allows us to take any series of standard reductions and move them upward, postponing any non-standard reductions until later.

A direct attempt at this diagram will fail, however, as we cannot get a footing. What holds for single reductions is:

$$\begin{tikzcd} c \arrow[r, dashed, mapsto, "\star"] \arrow[d, tail] & c' \arrow[d, tail, dashed, "\star"] \\ d \arrow[r, mapsto] & d' \end{tikzcd}$$

Moving a standard reduction forward may produce extra work, so there may be extra steps between c' and d'; also, the non-standard reduction may happen to be the next standard reduction, leading to *more* standard steps between c and c'. But this diagram cannot be "tiled" by induction to produce Fig. 10.

Happily, the extra work created by moving a standard reduction forward always has a particular form—the duplicated redexes can all be reduced in parallel. Thus we can obtain a more helpful diagram using parallel reduction.

Lemma 9. *If $c \Rightarrow d$ and $d \mapsto d'$, then $c \mapsto^\star c'$ with $c' \Rightarrow d'$.*

$$\begin{tikzcd} c \arrow[r, dashed, mapsto, "\star"] \arrow[d, Rightarrow] & c' \arrow[d, Rightarrow, dashed] \\ d \arrow[r, mapsto] & d' \end{tikzcd}$$

Proof. Proceed by induction on the number of reductions in $c \Rightarrow d$. If $c \equiv d$, we can take $c' \equiv d$ and we are done.

Otherwise, consider whether $c \Rightarrow d$ takes the standard reduction. If it does, we can refactor it as $c \mapsto c_1 \Rightarrow d$:

$$\begin{tikzcd} c \arrow[r, mapsto] \arrow[d, Rightarrow] & c_1 \arrow[r, dashed, mapsto, "\star"] \arrow[dl, Rightarrow] & c' \arrow[d, Rightarrow, dashed] \\ d \arrow[rr, mapsto] & & d' \end{tikzcd}$$

Since $c_1 \Rightarrow d$ by fewer reductions than $c \Rightarrow d$, we can finish using the induction hypothesis.

Otherwise, since we are not performing the standard reduction (which is always at the top level in our language), c and d have the same top-level form, simplifying the case analysis:

- The case where c is a jump is impossible, since then d would have to be a jump and a jump cannot take a standard reduction.
- Suppose $c \equiv \mathbf{let}\, x = v \,\mathbf{in}\, c_0$ and $d \equiv \mathbf{let}\, x = v' \,\mathbf{in}\, c_0'$ with $v \Rightarrow v'$ and $c_0 \Rightarrow c_0'$. Then $d \mapsto d' \equiv c_0'\,\{v'/x\}$. Let $c' \equiv c_0\,\{v/x\}$; then $c \mapsto c'$, and by Lemma 7, $c' \Rightarrow d'$.
- The case for $c \equiv \mathbf{let}\, j = \tilde{\mu}[\vec{a}\,,\vec{x}].d_0 \,\mathbf{in}\, c_0$ is similar.
- Finally, we have that $c \equiv \langle v \,\|\, k\rangle$ and $d \equiv \langle v' \,\|\, k'\rangle$ with $v \Rightarrow v'$ and $k \Rightarrow k'$. Continue by case analysis on v:
  - The case $v \equiv x$ is impossible since $\langle x \,\|\, k'\rangle$ cannot take a standard reduction.
  - If $v \equiv \mu\mathsf{ret}.c_0$, we must account for a possible μ_η-reduction. In this case, $c_0 \equiv \langle v_0 \,\|\, \mathsf{ret}\rangle$ and $v_0 \Rightarrow v'$.

    $$\begin{tikzcd} v_0 \arrow[d, Rightarrow] & k \arrow[d, Rightarrow] & c \equiv \langle \mu\mathsf{ret}.\,\langle v_0 \,\|\, \mathsf{ret}\rangle \,\|\, k\rangle \arrow[r, dashed, mapsto] \arrow[d, Rightarrow] & \langle v_0 \,\|\, k\rangle \arrow[dl, Rightarrow, dashed] \\ v' & k' & d \equiv \langle v' \,\|\, k'\rangle \arrow[r, mapsto] & d' \end{tikzcd}$$

    Since we have removed one reduction, we can use the induction hypothesis to finish:

    $$\begin{tikzcd} c \equiv \langle \mu\mathsf{ret}.\,\langle v_0 \,\|\, \mathsf{ret}\rangle \,\|\, k\rangle \arrow[r, mapsto] \arrow[d, Rightarrow] & \langle v_0 \,\|\, k\rangle \arrow[r, dashed, mapsto, "\star"] \arrow[dl, Rightarrow] & c' \arrow[d, Rightarrow, dashed] \\ d \equiv \langle v' \,\|\, k'\rangle \arrow[rr, mapsto] & & d' \end{tikzcd}$$

    If there is no μ_η-reduction in v, then $v' \equiv \mu\mathsf{ret}.c_0'$ with $c_0 \Rightarrow c_0'$:

    $$\begin{tikzcd} c_0 \arrow[d, Rightarrow] & k \arrow[d, Rightarrow] & c \equiv \langle \mu\mathsf{ret}.c_0 \,\|\, k\rangle \arrow[r, dashed, mapsto] \arrow[d, Rightarrow] & c_0\,\{k/\mathsf{ret}\} \arrow[d, Rightarrow, dashed] \\ c_0' & k' & d \equiv \langle \mu\mathsf{ret}.c_0' \,\|\, k'\rangle \arrow[r, mapsto] & c_0'\,\{k'/\mathsf{ret}\} \end{tikzcd}$$

    Here we take $c' \equiv c_0\,\{k/\mathsf{ret}\}$ and finish by Lemma 7.
  - In the other cases, v is a WHNF, so k is a matching continuation (that is, not ret and not a mismatched **case**). We show the case for $v \equiv \lambda x.v_0$; the Λ-case is simpler, and the constructor case is more complex but no more illuminating.

    $$\begin{tikzcd} v_0, v_1, k_0 \arrow[d, Rightarrow] & \langle \lambda x.v_0 \,\|\, v_1 \cdot k_0\rangle \arrow[r, dashed, mapsto] \arrow[d, Rightarrow] & \langle v_0\,\{v_1/x\} \,\|\, k_0\rangle \arrow[d, Rightarrow, dashed] \\ v_0', v_1', k_0' & \langle \lambda x.v_0' \,\|\, v_1' \cdot k_0\rangle \arrow[r, mapsto] & \langle v_0'\,\{v_1'/x\} \,\|\, k_0'\rangle \end{tikzcd}$$

    As before, along the right side we use Lemma 7. $\square$

Now we need to reconcile Lemma 9 with Fig. 10. First, we can break down $\Rightarrow$ into standard reduction ($\mapsto$) followed by non-standard parallel reduction ($\not\Rrightarrow$):

Lemma 10. *If $c \Rightarrow c'$ then $c \mapsto^\star\!\not\Rrightarrow c'$.*

Proof. By induction on the number of reductions taken by the derivation of $c \Rightarrow c'$. If $c \equiv c'$, we are done. Otherwise, if $c \Rightarrow c'$

does not take the standard reduction, we are again done. Finally, if it does take the standard reduction, then $c \mapsto c_1 \Rightarrow c'$ where $c_1 \Rightarrow c'$ takes fewer reductions, so we finish using the induction hypothesis. □

Now we can move Lemma 9 closer by referring to non-standard (parallel) reduction:

Lemma 11. *If $c \mathrel{>\!\!\!\Rightarrow} d$ and $d \mapsto d'$, then $c \mapsto^\star c'$ with $c' \mathrel{>\!\!\!\Rightarrow} d'$.*

$$\begin{tikzcd} c \arrow[r, mapsto, dashed, "\star"] \arrow[d, Rightarrow, tail] & c' \arrow[d, Rightarrow, tail, dashed] \\ d \arrow[r, mapsto] & d' \end{tikzcd}$$

Proof. Immediate from Lemmas 9 and 10. □

This is the diagram that we can "tile" to produce Fig. 10.

Lemma 12. *If $c \rightarrowtail^\star d$ and $d \mapsto^\star d'$, then $c \mapsto^\star c'$ with $c' \rightarrowtail^\star d'$ (see Fig. 10).*

Proof. Follows from Lemma 11. By induction, its diagram can be "tiled horizontally" to give:

$$\begin{tikzcd} c \arrow[r, mapsto, dashed, "\star"] \arrow[d, Rightarrow, tail] & c' \arrow[d, Rightarrow, tail, dashed] \\ d \arrow[r, mapsto, "\star"] & d' \end{tikzcd}$$

Now that the top and bottom match, we can "tile vertically":

$$\begin{tikzcd} c \arrow[r, mapsto, dashed, "\star"] \arrow[d, Rightarrow, tail, "\star"] & c' \arrow[d, Rightarrow, tail, dashed, "\star"] \\ d \arrow[r, mapsto, "\star"] & d' \end{tikzcd}$$

But then $\mathrel{>\!\!\!\Rightarrow}^\star = \rightarrowtail^\star$, and we're done. □

Now we are ready to prove standardization.

Proof of Proposition 8.

$$\begin{tikzcd} c \arrow[r, mapsto, dashed, "\star"] \arrow[dr, "\star"] & c_1 \not\mapsto \arrow[d, dashed, tail, "\star"] \\ & c' \not\mapsto \end{tikzcd}$$

Proceed by induction on $c \to^\star c'$. If there are no standard reductions in the sequence, then we can take $c_1 \equiv c$ and we're done by Lemma 5.

Thus assume there is at least one standard reduction; isolating the first one, we have $c \rightarrowtail^\star d \mapsto d' \to^\star c'$.

$$\begin{tikzcd} c \arrow[d, tail, "\star"] \arrow[dr, "\star"] & & \\ d \arrow[r, mapsto] & d' \arrow[dr, "\star"] & \\ & & c' \not\mapsto \end{tikzcd}$$

We can fill in the upper-left corner by Lemma 12 and the lower-right corner by the induction hypothesis:

$$\begin{tikzcd} c \arrow[r, mapsto, dashed, "\star"] \arrow[d, tail, "\star"] \arrow[dr, "\star"] & {} \arrow[d, dashed, tail, "\star"] & \\ d \arrow[r, mapsto] & d' \arrow[r, mapsto, dashed, "\star"] \arrow[dr, "\star"] & {} \not\mapsto \arrow[d, dashed, tail, "\star"] \\ & & c' \not\mapsto \end{tikzcd}$$

Finally, applying Lemma 12 again, along with Lemma 5, gives us our c_1:

$$\begin{tikzcd} c \arrow[r, mapsto, dashed, "\star"] \arrow[d, tail, "\star"] \arrow[dr, "\star"] & {} \arrow[r, mapsto, dashed, "\star"] \arrow[d, dashed, tail, "\star"] & c_1 \not\mapsto \arrow[d, dashed, tail, "\star"] \\ d \arrow[r, mapsto] & d' \arrow[r, mapsto, dashed, "\star"] \arrow[dr, "\star"] & {} \not\mapsto \arrow[d, dashed, tail, "\star"] \\ & & c' \not\mapsto \end{tikzcd}$$

□

We will also have occasion to move a standard reduction *after* a non-standard one:

Lemma 13. *If $c \mapsto c'$ and $c \to^\star d$, then $d \mapsto^{0/1} d'$ with $c' \to^\star d'$.*

$$\begin{tikzcd} c \arrow[r, mapsto] \arrow[d, "\star"] & c' \arrow[d, dashed, "\star"] \\ d \arrow[r, mapsto, dashed, "0/1"] & d' \end{tikzcd}$$

Proof. Proceed by induction on $c \to^\star d$. If $c \equiv d$, then trivially we take $d' \equiv c'$.

Now suppose $c \to c_1 \to^\star d$. By the induction hypothesis, if we can find c_1' with $c' \to^\star c_1'$ and $c_1 \mapsto c_1'$, then we get d' with $d \mapsto^{0/1} d'$ and $c_1' \to^\star d'$ and we're done.

$$\begin{tikzcd} c \arrow[r, mapsto] \arrow[d] & c' \arrow[d, dashed, "\star"] \\ c_1 \arrow[r, mapsto, dashed] \arrow[d, "\star"] & c_1' \arrow[d, dotted, "\star"] \\ d \arrow[r, mapsto, dotted, "0/1"] & d' \end{tikzcd}$$

Alternatively, if we find that $c_1' \equiv c_1$ (i.e., that $c' \to^\star c_1$), then we can pick $d' \equiv d$ as well.

Now, if in fact $c \mapsto c_1$, then we must have that $c' \equiv c_1$ so we take $c_1' \equiv c_1$. Otherwise, $c \to c_1$ by a non-standard reduction. Considering the cases for $c \mapsto c_1$, none can be interfered with by a reduction in a subterm, and such a reduction can always be performed afterward (though it may be replicated if the subterm is a substituend in the right-hand side of the rule). Thus the standard reduction can still take place in c_1, with the nonstandard reduction postponed to part of $c' \to^\star c_1'$. □

C.3 Readback Function

To show the correspondence between the call-by-name and call-by-need calculi, we will use a *bisimulation*. Key to defining the bisimulation will be our *readback function*, defined in Fig. 11.

The key property of the readback is this:

Lemma 14. *If*

$$\langle \mathcal{H};\ \mathcal{J}, \mathcal{R};\ c\rangle \rightsquigarrow \langle \mathcal{H}';\ \mathcal{J}', \mathcal{R}';\ c'\rangle,$$

$$\mathcal{U}(\langle\mathcal{H};\ \mathcal{J},\mathcal{R};\ c\rangle) = \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(c)))$$

$$\begin{aligned}
\mathcal{U}_H(\varepsilon) &= id\\
\mathcal{U}_H(\mathcal{H}, x = v) &= \mathcal{U}_H(\mathcal{H}) \circ \{v/x\}\\
\mathcal{U}_H(\mathcal{H}, x = \bullet) &= \mathcal{U}_H(\mathcal{H})
\end{aligned}$$

$$\begin{aligned}
\mathcal{U}_J(\varepsilon) &= id\\
\mathcal{U}_J(\mathcal{J}, j = \tilde{\mu}[\overrightarrow{a{:}\kappa}\ \overrightarrow{x{:}\tau}].c) &= \mathcal{U}_J(\mathcal{J}) \circ \left\{ c\overrightarrow{\{\tau/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\tau}\ \vec{v} \right\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{U}_R(\varepsilon)(c) &= id\\
\mathcal{U}_R((k,\mathcal{J}) : \mathcal{R})(c) &= \mathcal{U}_R(\mathcal{R})(c\,\{\mathcal{U}_J(\mathcal{J})(k)/\mathsf{ret}\})\\
\mathcal{U}_R(\mathbf{upd}\,x : \mathcal{R})(c) &= (\mathcal{U}_R(\mathcal{R})(c))\,\{\mu\mathsf{ret}.c/x\}
\end{aligned}$$

Figure 11. Readback function.

then

$$\mathcal{U}(\langle\mathcal{H};\ \mathcal{J},\mathcal{R};\ c\rangle) \to^{\star} \mathcal{U}(\langle\mathcal{H}';\ \mathcal{J}',\mathcal{R}';\ c'\rangle).$$

First, we will need a simple fact about reduction:

Lemma 15. *If $c \to c'$, then*

1. $\mathcal{U}_H(\mathcal{H})(c) \to \mathcal{U}_H(\mathcal{H})(c')$
2. $\mathcal{U}_J(\mathcal{J})(c) \to^{\star} \mathcal{U}_J(\mathcal{J})(c')$
3. $\mathcal{U}_R(\mathcal{R})(c) \to^{\star} \mathcal{U}_R(\mathcal{R})(c')$

Proof. Parts 1 and 2 are immediate from Lemma 6, since $\mathcal{U}_H$ and $\mathcal{U}_J$ produce substitutions.

For part 3, note that $\mathcal{U}_R(\mathcal{R})(c)$ always has the form $c\sigma$ for some substitution σ (which may include c in a substituend), so the substitution argument still applies—except that multiple reductions may be necessary since c may be copied. ☐

Proof of Lemma 14. By case analysis on the reduction:

- $(\beta^{\to})$

$$\begin{aligned}
&\mathcal{U}(\langle\mathcal{H};\ \mathcal{J},\mathcal{R};\ \langle\lambda x{:}\tau.v_1 \,\|\, v_2\cdot k\rangle\rangle)\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle\lambda x{:}\tau.v_1 \,\|\, v_2\cdot k\rangle)))\\
&\quad\to^{\star} \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle v_1\,\{v_2/x\} \,\|\, k\rangle)))\\
&\quad\equiv (\mathcal{U}_H(\mathcal{H})\circ\{v_2/x\})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle v_1 \,\|\, k\rangle)))\\
&\quad\equiv \mathcal{U}(\langle\mathcal{H}, x = v_2;\ \mathcal{J},\mathcal{R};\ \langle v_1 \,\|\, k\rangle\rangle)
\end{aligned}$$

- $(\beta^{\forall})$

$$\begin{aligned}
&\mathcal{U}(\langle\mathcal{H};\ \mathcal{J},\mathcal{R};\ \langle\Lambda a{:}\kappa.v \,\|\, \tau\cdot k\rangle\rangle)\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle\Lambda a{:}\kappa.v \,\|\, \tau\cdot k\rangle)))\\
&\quad\to^{\star} \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle v\,\{\tau/a\} \,\|\, k\rangle)))\\
&\quad\equiv \mathcal{U}(\langle\mathcal{H};\ \mathcal{J},\mathcal{R};\ \langle v\,\{\tau/a\} \,\|\, k\rangle\rangle)
\end{aligned}$$

- Similarly for $\mathbf{case}_{cons}$ and $\mathbf{case}_{def}$.

- (μ)

$$\begin{aligned}
&\mathcal{U}(\langle\mathcal{H};\ \mathcal{J},\mathcal{R};\ \langle\mu\mathsf{ret}.c \,\|\, k\rangle\rangle)\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle\mu\mathsf{ret}.c \,\|\, k\rangle)))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\langle\mu\mathsf{ret}.c \,\|\, \mathcal{U}_J(\mathcal{J})(k)\rangle))\\
&\quad\to^{\star} \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(c\,\{\mathcal{U}_J(\mathcal{J})(k)/\mathsf{ret}\}))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R((k,\mathcal{J}) : \mathcal{R})(c))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R((k,\mathcal{J}) : \mathcal{R})(\mathcal{U}_J(\varepsilon)(c)))\\
&\quad\equiv \mathcal{U}(\langle\mathcal{H};\ \varepsilon, (k,\mathcal{J}) : \mathcal{R};\ c\rangle)
\end{aligned}$$

Note that in pushing the substitution $\mathcal{U}_J(\mathcal{J})$ into the continuation of the command above, specifically:

$$\mathcal{U}_J(\mathcal{J})(\langle\mu\mathsf{ret}.c \,\|\, k\rangle) \equiv \langle\mu\mathsf{ret}.c \,\|\, \mathcal{U}_J(\mathcal{J})(k)\rangle$$

we exploit the fact that the term $\mu\mathsf{ret}.c$ must be continuation-closed, so that it is unaffected by $\mathcal{U}_J(\mathcal{J})$.

- (jump)
Assuming that

$$j = \tilde{\mu}[\overrightarrow{a{:}\kappa}, \overrightarrow{x{:}\tau}].c \in \mathcal{J},$$

$$\begin{aligned}
&\mathcal{U}(\langle\mathcal{H};\ \mathcal{J},\mathcal{R};\ \mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\rangle)\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v})))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(c\overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}})))\\
&\quad\equiv (\mathcal{U}_H(\mathcal{H})\circ\overrightarrow{\{v/x\}})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(c\overrightarrow{\{\sigma/a\}})))\\
&\quad\equiv \mathcal{U}(\langle\mathcal{H}, \overrightarrow{x = v};\ \mathcal{J},\mathcal{R};\ c\overrightarrow{\{\sigma/a\}}\rangle)
\end{aligned}$$

- (lookup)
Assuming that

$$x = V \in \mathcal{H},$$

$$\begin{aligned}
&\mathcal{U}(\langle\mathcal{H};\ \mathcal{J},\mathcal{R};\ \langle x \,\|\, k\rangle\rangle)\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle x \,\|\, k\rangle)))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle V \,\|\, k\rangle)))\\
&\quad\equiv \mathcal{U}(\langle\mathcal{H};\ \mathcal{J},\mathcal{R};\ \langle V \,\|\, k\rangle\rangle)
\end{aligned}$$

- (lazysubst)
Assuming that

$$x = K(\vec{\sigma}, \vec{v}) \in \mathcal{H},$$

$$
\begin{aligned}
&\mathcal{U}(\langle \mathcal{H};\ \mathcal{J}, \mathcal{R};\ \langle x \,\|\, k\rangle\rangle)\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle x \,\|\, k\rangle)))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H}, x = K(\vec{\sigma}, \vec{v}))\\
&\qquad(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle K(\vec{\sigma}, \vec{v}) \,\|\, k\rangle)))\\
&\quad\equiv (\mathcal{U}_H(\mathcal{H}) \circ \{K(\vec{\sigma}, \vec{v})/x\})\\
&\qquad(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle K(\vec{\sigma}, \vec{v}) \,\|\, k\rangle)))\\
&\quad\equiv (\mathcal{U}_H(\mathcal{H}) \circ \{K(\vec{\sigma}, \vec{v})/x\} \circ \overrightarrow{\{v/y\}})\\
&\qquad(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle K(\vec{\sigma}, \vec{y}) \,\|\, k\rangle)))\\
&\quad\equiv (\mathcal{U}_H(\mathcal{H}) \circ \overrightarrow{\{v/y\}} \circ \{K(\vec{\sigma}, \vec{y})/x\})\\
&\qquad(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle K(\vec{\sigma}, \vec{y}) \,\|\, k\rangle)))\\
&\quad\equiv \langle \mathcal{H}, \overline{y = v}, x = K(\vec{\sigma}, \vec{y});\ \mathcal{J}, \mathcal{R};\ \langle K(\vec{\sigma}, \vec{y}) \,\|\, k\rangle\rangle
\end{aligned}
$$

- (force)
  Assuming that
  $$x = \mu\mathsf{ret}.c \in \mathcal{H},$$

$$
\begin{aligned}
&\mathcal{U}(\langle \mathcal{H};\ \mathcal{J}, \mathcal{R};\ \langle x \,\|\, k\rangle\rangle)\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle x \,\|\, k\rangle)))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle \mu\mathsf{ret}.c \,\|\, k\rangle)))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\langle \mu\mathsf{ret}.c \,\|\, \mathcal{U}_J(\mathcal{J})(k)\rangle))\\
&\quad\to^\star \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(c\,\{\mathcal{U}_J(\mathcal{J})(k)/\mathsf{ret}\}))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R((k, \mathcal{J}) : \mathcal{R})(c))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H}, x = \bullet, x = \mu\mathsf{ret}.c)(\mathcal{U}_R((k, \mathcal{J}) : \mathcal{R})(c))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H}, x = \bullet)((\mathcal{U}_R((k, \mathcal{J}) : \mathcal{R})(c))\,\{\mu\mathsf{ret}.c/x\})\\
&\quad\equiv \mathcal{U}_H(\mathcal{H}, x = \bullet)(\mathcal{U}_R(\mathbf{upd}\,x : (k, \mathcal{J}) : \mathcal{R})(c))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H}, x = \bullet)(\mathcal{U}_R(\mathbf{upd}\,x : (k, \mathcal{J}) : \mathcal{R})(\mathcal{U}_J(\varepsilon)(c)))\\
&\quad\equiv \mathcal{U}(\langle \mathcal{H}, x = \bullet;\ \varepsilon, \mathbf{upd}\,x : (k, \mathcal{J}) : \mathcal{R};\ c\rangle)
\end{aligned}
$$

- (update)
$$
\begin{aligned}
&\mathcal{U}(\langle \mathcal{H};\ \mathcal{J}, \mathbf{upd}\,x : \mathcal{R};\ \langle W \,\|\, \mathsf{ret}\rangle\rangle)\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathbf{upd}\,x : \mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle W \,\|\, \mathsf{ret}\rangle)))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle W \,\|\, \mathsf{ret}\rangle))\,\{\mu\mathsf{ret}.\langle W \,\|\, \mathsf{ret}\rangle/x\})\\
&\quad\to^\star \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle W \,\|\, \mathsf{ret}\rangle))\,\{W/x\})\\
&\quad\equiv (\mathcal{U}_H(\mathcal{H}) \circ \{W/x\})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle W \,\|\, \mathsf{ret}\rangle)))\\
&\quad\equiv \mathcal{U}(\langle \mathcal{H}, x = W;\ \mathcal{J}, \mathcal{R};\ \langle W \,\|\, \mathsf{ret}\rangle\rangle)
\end{aligned}
$$

- (ret)
$$
\begin{aligned}
&\mathcal{U}(\langle \mathcal{H};\ \mathcal{J}, (k', \mathcal{J}') : \mathcal{R};\ \langle W \,\|\, \mathsf{ret}\rangle\rangle)\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\langle W \,\|\, \mathsf{ret}\rangle))\,\{\mathcal{U}_J(\mathcal{J}')(k')/\mathsf{ret}\})\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\langle W \,\|\, \mathcal{U}_J(\mathcal{J}')(k')\rangle))\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J}')(\langle W \,\|\, k'\rangle)))\\
&\quad\equiv \mathcal{U}(\langle \mathcal{H};\ \mathcal{J}', \mathcal{R};\ \langle W \,\|\, k'\rangle\rangle)
\end{aligned}
$$

  Note that in discarding $\mathcal{U}_J(\mathcal{J})$ above, we exploit the fact that W is continuation-closed, and hence the substitution $\mathcal{U}_J(\mathcal{J})$ was accomplishing nothing. Similarly, W is unaffected by $\mathcal{U}_J(\mathcal{J}')$, so we can move the latter out.
- ($\mathbf{let}_{val}$)
$$
\begin{aligned}
&\mathcal{U}(\langle \mathcal{H};\ \mathcal{J}, \mathcal{R};\ \mathbf{let}\,x = v\,\mathbf{in}\,c\rangle)\\
&\quad\equiv \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(\mathbf{let}\,x = v\,\mathbf{in}\,c)))\\
&\quad\to^\star \mathcal{U}_H(\mathcal{H})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(c\,\{v/x\})))\\
&\quad\equiv (\mathcal{U}_H(\mathcal{H}) \circ \{v/x\})(\mathcal{U}_R(\mathcal{R})(\mathcal{U}_J(\mathcal{J})(c)))\\
&\quad\equiv \mathcal{U}(\langle \mathcal{H}, x = v;\ \mathcal{J}, \mathcal{R};\ c\rangle)
\end{aligned}
$$
- Similarly for $\mathbf{let}_{cont}$. □

We will also require that the readback respects termination:

Lemma 16. *If $\langle \mathcal{H};\ \mathcal{J}, \mathcal{R};\ c\rangle \not\rightsquigarrow$, then $\mathcal{U}(\langle \mathcal{H};\ \mathcal{J}, \mathcal{R};\ c\rangle) \not\mapsto$.*

Proof. There are two forms of irreducible state in the call-by-need semantics: a missing **case** alternative and a WHNF passed to ret in an empty stack. The former reads back as a similarly stuck term, and the latter reads back as a WHNF. □

C.4 Bisimulation

Now we use the readback to define our bisimulation:

Definition 17. *Let $\sim$ relate call-by-name terms to call-by-need states, such that $c \sim \mathcal{S}$ when $c \to^\star \mathcal{U}(\mathcal{S})$.*

Lemma 18 (Bisimulation)**.** *Let $c \sim \mathcal{S}$.*

1. *If $c \mapsto c'$, then $\mathcal{S} \rightsquigarrow^\star \mathcal{S}'$ with $c' \sim \mathcal{S}'$.*
2. *If $\mathcal{S} \rightsquigarrow \mathcal{S}'$, then $c \mapsto^\star c'$ with $c' \sim \mathcal{S}'$.*

$$
\begin{tikzcd}
c \arrow[r, mapsto] \arrow[d, "\star"] & c' \arrow[d, dashed, "\star"] \\
d & d' \\
\mathcal{U} & \mathcal{U} \\
\mathcal{S} \arrow[r, dashed, rightsquigarrow, "\star"] & \mathcal{S}'
\end{tikzcd}
\qquad
\begin{tikzcd}
c \arrow[r, dashed, mapsto, "\star"] \arrow[d, "\star"] & c' \arrow[d, dashed, "\star"] \\
d & d' \\
\mathcal{U} & \mathcal{U} \\
\mathcal{S} \arrow[r, rightsquigarrow] & \mathcal{S}'
\end{tikzcd}
$$

Proof. 2 is a corollary of Lemma 14; we simply take $c' \equiv c$ so that we get $c \equiv c' \to^\star d \to^\star d'$.[2]

For 1, suppose $c \sim S$ and $c \mapsto c'$. By definition of $\sim$, we have $c \to^\star d \equiv \mathcal{U}(\mathcal{S})$:

$$
\begin{tikzcd}
c \arrow[r, mapsto] \arrow[d, "\star"] & c' \\
d & \\
\mathcal{U} & \\
\mathcal{S} &
\end{tikzcd}
$$

By Lemma 13, we have $d \mapsto^{0/1} d' \leftarrow^\star c'$:

$$
\begin{tikzcd}
c \arrow[r, mapsto] \arrow[d, "\star"] & c' \arrow[d, dashed, "\star"] \\
d \arrow[r, dashed, mapsto, "0/1"] & d' \\
\mathcal{U} & \\
\mathcal{S} &
\end{tikzcd}
$$

If $d \equiv d'$, then we can pick $\mathcal{S}' \equiv \mathcal{S}$ and we're done. Otherwise, assume $d \mapsto d'$. It will suffice to show that $\mathcal{S} \rightsquigarrow^\star \mathcal{S}'$ with $\mathcal{U}(\mathcal{S}') \equiv d'$:

$$
\begin{tikzcd}
c \arrow[r, mapsto] \arrow[d, "\star"] & c' \arrow[d, "\star"] \\
d \arrow[r, mapsto] & d' \\
\mathcal{U} & \mathcal{U} \\
\mathcal{S} \arrow[r, dashed, rightsquigarrow, "\star"] & \mathcal{S}'
\end{tikzcd}
$$

[2] Allowing $c \mapsto^\star c'$ is not necessary here, but it is *pro forma* for a bisimulation.

As noted before, $\mathcal{U}$ produces a substitution—$\mathcal{U}(\langle \mathcal{H};\ \mathcal{J}, \mathcal{R};\ c\rangle) \equiv c\sigma$ for some σ. In general, for a substitution σ, if $c\sigma \mapsto c'\sigma$, then at least one of the following is true:

1. $c \equiv \langle v \parallel k\rangle \mapsto c'$.
2. $c \equiv \mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}$ and $j \in \operatorname{dom}\sigma$.
3. $c \equiv \langle v \parallel \mathsf{ret}\rangle$ and $\mathsf{ret} \in \operatorname{dom}\sigma$.
4. $c \equiv \langle x \parallel k\rangle$ and $x \in \operatorname{dom}\sigma$.

The last three cases may apply multiple times, but not infinitely many as bindings are not recursive (each substitution reduces the size of the context). Eventually we must land on case 1. Thus we may proceed by induction on the number of substitutions required to expose a redex.

In case 1, we have $c \equiv \langle v \parallel k\rangle \mapsto c'$. This means that $\mathcal{S} \equiv \langle \mathcal{H};\ \mathcal{J}, \mathcal{R};\ \langle v \parallel k\rangle\rangle$, and the substitution produced was not crucial to forming the redex. Therefore one of the "external" reduction rules—namely $\beta^{\rightarrow}$, $\beta^{\forall}$, $\mathbf{case}_{cons}$, $\mathbf{case}_{def}$, μ, $\mathbf{let}_{val}$, and $\mathbf{let}_{cont}$—must apply; each of them makes precisely the same substitutions as a corresponding call-by-name rule, only delaying some work by adding to $\mathcal{H}$, $\mathcal{J}$, or $\mathcal{R}$.

In case 2, the jump rule applies, and we apply the induction hypothesis. Similarly, case 3 is covered by some number of updates (each of which consumes an update frame) followed by a ret, and case 4 is covered by one of lookup, $\mathsf{lazysubst}$, and force. $\square$

Lemma 19 (Bisimulation respects termination)**.** *Let $c \sim \mathcal{S}$.*

1. *If $c \not\mapsto$, then $\mathcal{S} \rightsquigarrow^{\star}\not\rightsquigarrow$.*
2. *If $\mathcal{S} \not\rightsquigarrow$, then $c \mapsto^{\star}\not\mapsto$.*

Proof. 1. By Lemma 5, $\mathcal{U}(\mathcal{S}) \not\mapsto$. From there, the case analysis is similar to that for Lemma 18, as internal reductions perform whatever substitutions are necessary for the stuck command to appear.
2. We know that $c \rightarrow^{\star} \mathcal{U}(\mathcal{S})$. By Lemma 16, we have that $\mathcal{U}(\mathcal{S}) \not\mapsto$; then the result holds by standardization (Proposition 8). $\square$

The proposition is now reduced to a corollary:

Proof of Proposition 1. Since $c \sim \langle \varepsilon;\ \varepsilon, \varepsilon;\ c\rangle$, both directions follow directly from Lemmas 18 and 19 by induction on the reduction sequence. $\square$

D. Proof of Type Safety (Proposition 2)

As is typical when proving type safety, we will require a lemma dealing with substitution and typing.

Lemma 20 (Substitution)**.** *1. If $\Gamma \vdash \tau : \kappa$, then:*

- *(a) If $\Gamma, a : \kappa \vdash \sigma : \kappa'$, then $\Gamma\{\tau/a\} \vdash \sigma\{\tau/a\} : \kappa'$.*
- *(b) If $\Gamma, a : \kappa \vdash v : \sigma$, then $\Gamma\{\tau/a\} \vdash v\{\tau/a\} : \sigma\{\tau/a\}$.*
- *(c) If $\Gamma, a : \kappa \mid k : \sigma \vdash \Delta$, then $\Gamma\{\tau/a\} \mid k\{\tau/a\} : \sigma\{\tau/a\} \vdash \Delta\{\tau/a\}$.*
- *(d) If $c : (\Gamma, a : \kappa \vdash \Delta)$, then $c\{\tau/a\} : (\Gamma\{\tau/a\} \vdash \Delta\{\tau/a\})$.*

2. If $\Gamma \vdash v : \tau$, then:

- *(a) If $\Gamma, x : \tau \vdash v' : \sigma$, then $\Gamma \vdash v'\{v/x\} : \sigma$.*
- *(b) If $\Gamma, x : \tau \mid k : \sigma \vdash \Delta$, then $\Gamma \mid k\{v/x\} : \sigma \vdash \Delta$.*
- *(c) If $c : (\Gamma, x : \tau \vdash \Delta)$, then $c\{v/x\} : (\Gamma \vdash \Delta)$.*
- *(d) For any k, $v \equiv v\{k/\mathsf{ret}\}$.*
- *(e) For any c and j, $v \equiv v\left\{c\overrightarrow{\{\sigma/a\}}\overrightarrow{\{v'/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v'}\right\}$.*

3. If $\Gamma \mid k : \tau \vdash \Delta, \mathsf{ret} : \sigma$, then:

- *(a) If $\Gamma \mid k' : \sigma' \vdash \Delta, \mathsf{ret} : \tau$, then $\Gamma \mid k'\{k/\mathsf{ret}\} : \sigma' \vdash \Delta, \mathsf{ret} : \sigma$.*
- *(b) If $c : (\Gamma \vdash \Delta, \mathsf{ret} : \tau)$, then $c\{k/\mathsf{ret}\} : (\Gamma \vdash \Delta, \mathsf{ret} : \sigma)$.*

4. If $c : (\Gamma, \overrightarrow{a : \kappa}, \overrightarrow{x : \tau} \vdash \Delta)$, then:

- *(a) If $\Gamma \mid k : \sigma' \vdash \Delta, j : \exists\overrightarrow{a{:}\kappa}.(\vec{\tau})$, then $\Gamma \mid k\left\{c\overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\right\} : \sigma' \vdash \Delta$.*
- *(b) If $c' : (\Gamma \vdash \Delta, j : \exists\overrightarrow{a{:}\kappa}.(\vec{\tau}))$, then $c'\left\{c\overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\right\} : (\Gamma \vdash \Delta)$.*

Proof. A series of straightforward (if large) mutual inductions. 2(d) and 2(e) are trivial since well-typed terms have no free continuation variables. 4 relies on the first three to handle the structural substitution.

For example, in 4(b), for the case where $c' \equiv \mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}$, we have

$$c'\left\{c\overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}\right\} \equiv c\overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}$$

The result then follows by (repeated applications of) 1(d) and 2(c). $\square$

Proof of Proposition 2. 1. A simple case analysis on c:

- If c is a $\mathbf{let}$, one of the $\mathbf{let}$ rules applies.
- c cannot be a jump because its context has no join variables.
- Suppose $c \equiv \langle v \parallel k\rangle$. Then v is not a variable since it is closed. If it is a μ-abstraction, we can reduce no matter what k is. Otherwise, v is a WHNF; if k is ret, we are done, and otherwise either we can reduce or k is a stuck $\mathbf{case}$.

2. An easy case analysis, applying Lemma 20 in each case. $\square$

E. Proof of Round-Trip Equivalence (Proposition 3)

E.1 Equational Reasoning

As is standard, we will avoid proving observational equivalence directly and instead rely on equational reasoning. To this end, we define equality in Sequent Core ($=$) as the reflexive-transitive-symmetric closure of $\rightarrow$ as defined in Section C.1. Note that the reduction theory of Sequent Core is confluent (here z and z' may be terms, continuations, or commands).

Proposition 21 (Confluence)**.** *If $z_1 \leftarrow^{\star} z \rightarrow^{\star} z_2$ then there is a z' such that $z_1 \rightarrow^{\star} z' \leftarrow^{\star} z_2$.*

Proof. All reduction rules are left-linear, and there are no critical pairs. In particular, the only overlapping redexes are between μ reduction and μ_η reduction, but they lead to exactly the same result:

$$\langle \mu\mathsf{ret}.\ \langle v \parallel \mathsf{ret}\rangle \parallel k\rangle \rightarrow \langle v \parallel k\rangle$$

It then follows that the parallel reduction relation also defined in Section C.1 has the diamond property:

$$\begin{tikzcd} & z \arrow[dl, Rightarrow] \arrow[dr, Rightarrow] & \\ z_1 \arrow[dr, Rightarrow, dashed] & & z_2 \arrow[dl, Rightarrow, dashed] \\ & z' & \end{tikzcd}$$

And thus the single-step reduction relation is confluent. $\square$

Standardization (Proposition 8) and confluence (Proposition 21) then give us license to use equational reasoning to prove observational equivalence:

Proposition 22. *If $z = z'$, then $z \cong z'$.*

$$
\begin{aligned}
E \in EvalCxt &::= \Box \mid E\ e \mid \mathbf{case}\, E\, \mathbf{of}\, \overrightarrow{alt} \\
W \in WHNF &::= \lambda x{:}\tau.e \mid \Lambda a{:}\kappa.e \mid x \mid K\, \vec{\sigma}\ \vec{e}
\end{aligned}
$$

$$
\begin{aligned}
(\lambda x{:}\tau.e)\ e' &\mapsto e\,\{e'/x\} \\
(\Lambda a{:}\kappa.e)\ \sigma &\mapsto e\,\{\sigma/a\} \\
\mathbf{case}\, K\,\vec{\sigma}\ \vec{e}\ \mathbf{of}\ \overrightarrow{alt} &\mapsto e'\overrightarrow{\{\sigma/b\}}\overrightarrow{\{e/x\}} && K\,\overrightarrow{b{:}\kappa}\ \overrightarrow{x{:}\tau} \to e' \in \overrightarrow{alt} \\
\mathbf{case}\, W\ \mathbf{of}\ \overrightarrow{alt} &\mapsto e'\,\{W/x\} && x \to e' \in \overrightarrow{alt} \\
\mathbf{let}\, x{:}\tau = e\, \mathbf{in}\, e' &\mapsto e'\,\{e/x\} \\
E[e] &\mapsto E[e'] && e \mapsto e'
\end{aligned}
$$

Figure 12. Call-by-name operational semantics for Core

Proof. Suppose $z = z'$ and, without loss of generality for arbitrary C, $C[z] \mapsto^\star c \not\mapsto$. To show $z \cong z'$, we need to show that there is a c' such that $C[z'] \mapsto^\star c' \not\mapsto$. By confluence (Proposition 21), $z \to^\star z_1 \leftarrow^\star z'$, so since $\to$ is a congruence, $C[z] \to^\star C[z_1] \leftarrow^\star C[z']$.

$$
\begin{tikzcd}
C[z] \arrow[r, mapsto, "\star"] \arrow[d, equal] \arrow[dr, dashed, "\star"] & c \not\mapsto \\
C[z'] \arrow[r, dashed, "\star"] & C[z_1]
\end{tikzcd}
$$

Invoking confluence again, we get $c \to^\star c_1 \leftarrow^\star C[z_1]$. By Lemma 5, $c_1 \not\mapsto$. Now standardization (Proposition 8) gives us c' with $C[z'] \mapsto^\star c' \not\mapsto$.

$$
\begin{tikzcd}
C[z] \arrow[rr, mapsto, "\star"] \arrow[dr, "\star"] \arrow[dd, equal] & & c \not\mapsto \arrow[dr, dashed, "\star"] & \\
& C[z_1] \arrow[rr, dashed, "\star"] & & c_1 \not\mapsto \\
C[z'] \arrow[ur, "\star"] \arrow[rr, mapsto, dashed, "\star"] & & c' \not\mapsto \arrow[ur, dashed, "\star"] &
\end{tikzcd}
$$ $\square$

In addition to the equational reasoning about Sequent Core terms (and commands and continuations), we will also need to reason equationally about Core terms to establish the round-trip equivalence. The theory for Core equations is built up in the same way as we did for Sequent Core. In particular, we equip Core with a standard call-by-name operational semantics ($\mapsto$), with the basic single-step rules and compatible closure under evaluation contexts illustrated in Figure 12. For the general reduction of Core expressions ($\to$), we take the compatible closure of the single-step operational relation ($\mapsto$) along with the additional rule for performing a generalized case-of-case:

$$E[\mathbf{case}\, e'\, \mathbf{of}\, \overrightarrow{pat \to e}] \to \mathbf{case}\, e'\, \mathbf{of}\, \overrightarrow{pat \to E[e]}$$

As before, we write the reflexive-transitive closures of $\mapsto$ and $\to$ for Core as $\mapsto^\star$ and $\to^\star$, respectively, and the reflexive-transitive-symmetric closure of $\to$ as $=$. Note that other forms of commutative conversions besides the generalized case-of-case hold up to equational reasoning due to other steps from the operational semantics, including:

$$
\begin{aligned}
E[\mathbf{let}\, x{:}\tau = e'\, \mathbf{in}\, e] &= \mathbf{let}\, x{:}\tau = e'\, \mathbf{in}\, E[e] \\
E[(\lambda x{:}\tau.e)\ e'] &= (\lambda x{:}\tau.E[e])\ e' \\
E[(\Lambda a{:}\kappa.e)\ \sigma] &= (\Lambda a{:}\kappa.E[e])\ \sigma \\
E[\mathbf{case}\, e'\, \mathbf{of}\, \overrightarrow{pat \to e}] &= \mathbf{case}\, e'\, \mathbf{of}\, \overrightarrow{pat \to E[e]}
\end{aligned}
$$

The first and the last equations in particular will be useful for reflecting the μ-reduction of Sequent Core back into Core.

Just like with Sequent Core, the standard semantics of Core enjoys both confluence and standardization. Therefore equational reasoning in Core is a valid method of establishing an observational equivalence in Core.

Proposition 23 (Confluence of Core)**.** *If $e_1 \leftarrow^\star e \to^\star e_2$ then there is an e' such that $e_1 \to^\star e' \leftarrow^\star e_2$.*

Proposition 24 (Standardization of Core)**.** *If $e \to^\star e' \not\mapsto^\star$ then $e \to^\star e_1 \not\mapsto^\star$ and $e_1 \to^\star e'$.*

Proposition 25. *If $e = e'$ then $e \cong e'$.*

Proof. The same reasoning as for Proposition 22, except for using confluence (Proposition 23) and standardization (Proposition 24) for Core instead of for Sequent Core. $\square$

E.2 Proof

For simplicity, we will prove round-trip equivalence for the compositional translation S rather than the administrative-free translation S_a. In other words, we will use the following fact:

Proposition 26. $S\,[\![e]\!] = S_a\,[\![e]\!]$.

Proof. Note that the S_a transformation has two forms on expressions:

$$S_a\,[\![e]\!] \qquad\qquad S_a\,[\![e]\!]\ k$$

It can be shown simultaneously that both $S\,[\![e]\!] = S_a\,[\![e]\!]$ and $\langle S\,[\![e]\!] \,\|\, k\rangle = S_a\,[\![e]\!]\ k$ hold by induction on the Core expression e. The most common difference between $S\,[\![e]\!]$ and $S_a\,[\![e]\!]$ is that $S\,[\![e]\!]$ μ-reduces to $S_a\,[\![e]\!]$. For example, in the case where $e \equiv e_1\ e_2$, we have:

$$
\begin{aligned}
\langle S\,[\![e_1\ e_2]\!] \,\|\, k\rangle &\equiv \langle \mu\mathsf{ret}.\, \langle S\,[\![e_1]\!] \,\|\, S\,[\![e_2]\!] \cdot \mathsf{ret}\rangle \,\|\, k\rangle \\
&= \langle S\,[\![e_1]\!] \,\|\, S\,[\![e_2]\!] \cdot k\rangle \\
&= S_a\,[\![e_1]\!]\ (S_a\,[\![e_2]\!] \cdot k) && \text{(by I.H.)} \\
&\equiv S_a\,[\![e_1\ e_2]\!]\ k
\end{aligned}
$$

The only other difference to account for is the shrink operation, which is undone by inlining the created **let** bindings. $\square$

To make full use of this equivalence, we need to know that D preserves this equality, at least for programs without join points. (Join points can be accommodated, but it would complicate the proof.)

Lemma 27. *In the join-point-free fragment:*

1. *If $v = v'$, then $D\,[\![v]\!] = D\,[\![v']\!]$.*
2. *If $k = k'$ and $e = e'$, then $D\,[\![k]\!]\ [e] = D\,[\![k']\!]\ [e']$.*
3. *If $c = c'$, then $D\,[\![c]\!] = D\,[\![c']\!]$.*

Proof. By mutual induction on the derivation of $=$. Because the D translation is compositional and hygienic (it does not cause escape or capture of static variables), it suffices to show that each reduction rule is preserved. Crucially, we must deal with how translation interacts with continuation substitution. We claim:

$$
\begin{aligned}
D\,[\![c\,\{k/\mathsf{ret}\}]\!] &= D\,[\![k]\!]\ [D\,[\![c]\!]] \\
D\,[\![k'\,\{k/\mathsf{ret}\}]\!]\ [e] &= D\,[\![k]\!]\ [D\,[\![k']\!]\ [e]]
\end{aligned}
$$

The claim is proved by mutual induction. Important cases:

- For $c \equiv \langle v \parallel k' \rangle$:
$$\begin{aligned} & D\left[\!\left[\langle v \parallel k' \rangle \{k/\mathsf{ret}\}\right]\!\right] \\ & \quad \equiv D\left[\!\left[k' \{k/\mathsf{ret}\}\right]\!\right] [D [\![v]\!]] \\ & \quad = D [\![k]\!] \left[D \left[\!\left[k'\right]\!\right] [D [\![v]\!]]\right] && \text{(by I.H.)} \\ & \quad \equiv D [\![k]\!] \left[D \left[\!\left[\langle v \parallel k' \rangle\right]\!\right]\right] \end{aligned}$$
- For $k' \equiv v \cdot k''$:
$$\begin{aligned} & D\left[\!\left[(v \cdot k'') \{k/\mathsf{ret}\}\right]\!\right] [e] \\ & \quad \equiv D\left[\!\left[v \cdot k'' \{k/\mathsf{ret}\}\right]\!\right] [e] \\ & \quad \equiv D\left[\!\left[k'' \{k/\mathsf{ret}\}\right]\!\right] [\Box\ D [\![v]\!]][e] \\ & \quad \equiv D\left[\!\left[k'' \{k/\mathsf{ret}\}\right]\!\right] [e\ D [\![v]\!]] \\ & \quad = D [\![k]\!] \left[D \left[\!\left[k''\right]\!\right] [e\ D [\![v]\!]]\right] && \text{(by I.H.)} \\ & \quad \equiv D [\![k]\!] \left[D \left[\!\left[k''\right]\!\right] [\Box\ D [\![v]\!]][e]\right] \\ & \quad \equiv D [\![k]\!] \left[D \left[\!\left[v \cdot k''\right]\!\right] [e]\right] \end{aligned}$$
- For $k' \equiv \mathbf{case\,of}\ \overrightarrow{pat \to c}$ (letting p stand for a pattern, which may be a default pattern):
$$\begin{aligned} & D\left[\!\left[\mathbf{case\,of}\ \overrightarrow{pat \to c}\ \{k/\mathsf{ret}\}\right]\!\right] [e] \\ & \quad \equiv (\mathbf{case}\,\Box\,\mathbf{of}\ \overrightarrow{pat \to D [\![c]\!]) \{k/\mathsf{ret}\}}[e] \\ & \quad \equiv \mathbf{case}\,\Box\,\mathbf{of}\ \overrightarrow{pat \to D [\![c \{k/\mathsf{ret}\}]\!]}[e] \\ & \quad \equiv \mathbf{case}\,e\,\mathbf{of}\ \overrightarrow{pat \to D [\![c \{k/\mathsf{ret}\}]\!]} \\ & \quad = \mathbf{case}\,e\,\mathbf{of}\ \overrightarrow{pat \to D [\![k]\!] [D [\![c]\!]]} && \text{(by I.H.)} \\ & \quad \leftarrow D [\![k]\!] [\mathbf{case}\,e\,\mathbf{of}\ \overrightarrow{pat \to D [\![c]\!]}] \\ & \quad \equiv D [\![k]\!] [(\mathbf{case}\,\Box\,\mathbf{of}\ \overrightarrow{pat \to D [\![c]\!]})[e]] \\ & \quad \equiv D [\![k]\!] \left[D \left[\!\left[\mathbf{case\,of}\ \overrightarrow{pat \to c}\right]\!\right] [e]\right] \end{aligned}$$
Note that we have made use of the extra reduction rule to perform the case-of-case transform.

With the claim proved, we can handle μ-reduction. If $c \mapsto c'$ by μ, then $c \equiv \langle \mu\mathsf{ret}.c'' \parallel k \rangle$ and $c' \equiv c'' \{k/\mathsf{ret}\}$. Then:
$$\begin{aligned} & D\left[\!\left[\langle \mu\mathsf{ret}.c'' \parallel k \rangle\right]\!\right] \\ & \quad \equiv D [\![k]\!] \left[D \left[\!\left[\mu\mathsf{ret}.c''\right]\!\right]\right] \\ & \quad \equiv D [\![k]\!] \left[D \left[\!\left[c''\right]\!\right]\right] \\ & \quad = D \left[\!\left[c'' \{k/\mathsf{ret}\}\right]\!\right] \end{aligned}$$
The other cases of reduction are straightforward. □

Now we are prepared to show that $D\left[\!\left[S_a [\![e]\!]\right]\!\right] = e$. To show the other direction, that $S_a\left[\!\left[D [\![v]\!]\right]\!\right] = v$, we must deal with the erasure of join points—since our direct-style language has no join points, we translate them back as ordinary functions. We can describe the effect this has in terms of the sequent calculus; this will simplify the proofs greatly.

Definition 28. *Define the* decontification *function* $V [\![-]\!]$ *as homomorphic on all syntax except*
$$V [\![j = \tilde{\mu}[\vec{a}, \vec{x}].c]\!] \equiv j = \overrightarrow{\Lambda a.}\overrightarrow{\lambda x.}\mu\mathsf{ret}.V [\![c]\!]$$
and
$$V [\![\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}]\!] \equiv \left\langle j \,\middle\|\, \vec{\sigma} \cdot \overrightarrow{V [\![v]\!]} \cdot \mathsf{ret} \right\rangle.$$

Decontification is purely syntactic—it does not affect the observable behavior of the program.

Lemma 29. *For all v, k, and c with no free continuation variables,*

1. $V [\![v]\!] = v$,
2. $V [\![k]\!] = k$, and
3. $V [\![c]\!] = c$.

Proof. To investigate the effect of structural substitution versus substitution of a decontified function, we will need a version of V that only decontifies *some* variables. Hence for each set of variables ρ, let V_ρ be homomorphic on all syntax except that
$$\begin{aligned} & V_\rho\left[\!\left[\mathbf{let}\, j = \tilde{\mu}[\vec{a}, \vec{x}].c\,\mathbf{in}\,c'\right]\!\right] \equiv \\ & \qquad \mathbf{let}\, j = \overrightarrow{\Lambda a.}\overrightarrow{\lambda x.}\mu\mathsf{ret}.V_\rho [\![c]\!]\ \mathbf{in}\ V_{\rho \cup \{j\}}\left[\!\left[c'\right]\!\right] \end{aligned}$$
and
$$V_\rho [\![\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}]\!] \equiv \begin{cases} \left\langle j \,\middle\|\, \vec{\sigma} \cdot \overrightarrow{V_\rho [\![v]\!]} \cdot \mathsf{ret} \right\rangle, & j \in \rho \\ \mathsf{jump}\ j\ \vec{\sigma}\ \overrightarrow{V_\rho [\![v]\!]}, & j \notin \rho. \end{cases}$$
It is obvious that
$$V [\![z]\!] \equiv V_{\mathrm{fjv}(z)} [\![z]\!]$$
for any term, continuation, or command z, where fjv gives the free join variables in its argument.

Now we can characterize the interaction of V with structural substitution. Supposing that $j \notin \rho$ and $\mathrm{fjv}(c') \subseteq \rho$, we claim:
$$\begin{aligned} & V_\rho [\![c]\!] \left\{ V_\rho \left[\!\left[c'\right]\!\right] \overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v} \right\} = \\ & \qquad V_{\rho \cup \{j\}} [\![c]\!] \left\{ \overrightarrow{\Lambda a.}\overrightarrow{\lambda x.}\mu\mathsf{ret}.V_\rho \left[\!\left[c'\right]\!\right]/j \right\} \end{aligned}$$
Note that we have tacitly made use of the fact that $V_\rho [\![c']\!]$ has no free join variables (since c' has only ρ as free join variables) so that $\mu\mathsf{ret}.V_\rho[\![c']\!]$ is well-typed.

The claim is proved by induction; the interesting case is this:
$$\begin{aligned} & V_\rho [\![\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}]\!] \left\{ V_\rho \left[\!\left[c'\right]\!\right] \overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v} \right\} \\ & \quad \equiv (\mathsf{jump}\ j\ \vec{\sigma}\ \overrightarrow{V_\rho [\![v]\!]}) \left\{ V_\rho \left[\!\left[c'\right]\!\right] \overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v} \right\} \\ & \quad \equiv V_\rho \left[\!\left[c'\right]\!\right] \overrightarrow{\{\sigma/a\}}\overrightarrow{\{V_\rho [\![v]\!]/x\}} \\ & \quad \leftarrow^{\star} \left\langle \overrightarrow{\Lambda a.}\overrightarrow{\lambda x.}\mu\mathsf{ret}.V_\rho \left[\!\left[c'\right]\!\right] \,\middle\|\, \vec{\sigma} \cdot \overrightarrow{V_\rho [\![v]\!]} \cdot \mathsf{ret} \right\rangle \\ & \quad \equiv \left\langle j \,\middle\|\, \vec{\sigma} \cdot \overrightarrow{V_\rho [\![v]\!]} \cdot \mathsf{ret} \right\rangle \left\{ \overrightarrow{\Lambda a.}\overrightarrow{\lambda x.}\mu\mathsf{ret}.V_\rho \left[\!\left[c'\right]\!\right]/j \right\} \\ & \quad \equiv \left\langle j \,\middle\|\, \vec{\sigma} \cdot \overrightarrow{V_{\rho \cup \{j\}} [\![v]\!]} \cdot \mathsf{ret} \right\rangle \left\{ \overrightarrow{\Lambda a.}\overrightarrow{\lambda x.}\mu\mathsf{ret}.V_\rho \left[\!\left[c'\right]\!\right]/j \right\} \\ & \quad \equiv V_{\rho \cup \{j\}} [\![\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v}]\!] \left\{ \overrightarrow{\Lambda a.}\overrightarrow{\lambda x.}\mu\mathsf{ret}.V_\rho \left[\!\left[c'\right]\!\right]/j \right\} \end{aligned}$$
Note that the penultimate step makes use of the fact that the subscript of V does not matter when operating on terms, since terms have no free join variables.

Now we can prove the lemma by mutual induction, generalizing each clause by replacing V with V_ρ. The crucial case is in clause 3:
$$\begin{aligned} & V_\rho\left[\!\left[\mathbf{let}\, j = \tilde{\mu}[\vec{a}, \vec{x}].c\,\mathbf{in}\,c'\right]\!\right] \\ & \quad \equiv \mathbf{let}\, j = \overrightarrow{\Lambda a.}\overrightarrow{\lambda x.}\mu\mathsf{ret}.V_\rho [\![c]\!]\ \mathbf{in}\ V_{\rho \cup \{j\}}\left[\!\left[c'\right]\!\right] \\ & \quad \to V_{\rho \cup \{j\}}\left[\!\left[c'\right]\!\right] \left\{ \overrightarrow{\Lambda a.}\overrightarrow{\lambda x.}\mu\mathsf{ret}.V_\rho [\![c]\!]/j \right\} \\ & \quad = V_\rho \left[\!\left[c'\right]\!\right] \left\{ V_\rho [\![c]\!] \overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v} \right\} \\ & \quad = c' \left\{ c\overrightarrow{\{\sigma/a\}}\overrightarrow{\{v/x\}}/\mathsf{jump}\ j\ \vec{\sigma}\ \vec{v} \right\} && \text{(by I.H.)} \\ & \quad \leftarrow \mathbf{let}\, j = \tilde{\mu}[\vec{a}, \vec{x}].c\,\mathbf{in}\,c' \end{aligned}$$
□

Decontification also does not affect how the program translates back to direct style.

Lemma 30. *1.* $D\left[\!\left[V[\![v]\!]\right]\!\right] \equiv D[\![v]\!]$.
2. $D\left[\!\left[V[\![k]\!]\right]\!\right] \equiv D[\![k]\!]$.
3. $D\left[\!\left[V[\![c]\!]\right]\!\right] \equiv D[\![c]\!]$.

Proof. Easy induction. For example:

$$\begin{aligned} & D\left[\!\left[V[\![\mathsf{jump}\; j\; \vec{\sigma}\; \vec{v}]\!]\right]\!\right] \\ & \quad \equiv D[\![\langle j \parallel \vec{\sigma} \cdot \vec{v} \cdot \mathsf{ret}\rangle]\!] \\ & \quad \equiv j\; \vec{\sigma}\; \vec{v} \\ & \quad \equiv D[\![\mathsf{jump}\; j\; \vec{\sigma}\; \vec{v}]\!] \end{aligned}$$

□

Now for the meat of the proof. Decontification will free us from having to consider join points when translating to direct style and back.

Lemma 31. *1.* $D\left[\!\left[S[\![e]\!]\right]\!\right] \equiv e$.
2. (a) $S\left[\!\left[D[\![v]\!]\right]\!\right] = v$.
(b) $S\left[\!\left[D[\![k]\!]\;[e]\right]\!\right] = \mu\mathsf{ret}.\left\langle S[\![e]\!] \parallel k\right\rangle$.
(c) $S\left[\!\left[D[\![c]\!]\right]\!\right] = \mu\mathsf{ret}.c$.

Proof. 1. By induction on e:

- All cases where e is a value are trivial.
- For $e \equiv \mathbf{let}\; bind\; \mathbf{in}\; e$:

$$\begin{aligned} & D\left[\!\left[S[\![\mathbf{let}\; bind\; \mathbf{in}\; e]\!]\right]\!\right] \\ & \quad \equiv D\left[\!\left[\mu\mathsf{ret}.\,\mathbf{let}\; S[\![bind]\!]\; \mathbf{in}\; \left\langle S[\![e]\!] \parallel \mathsf{ret}\right\rangle\right]\!\right] \\ & \quad \equiv \mathbf{let}\; D\left[\!\left[S[\![bind]\!]\right]\!\right]\; \mathbf{in} (D[\![\mathsf{ret}]\!])[D\left[\!\left[S[\![e]\!]\right]\!\right]] \\ & \quad \equiv \mathbf{let}\; D\left[\!\left[S[\![bind]\!]\right]\!\right]\; \mathbf{in}\; D\left[\!\left[S[\![e]\!]\right]\!\right] \\ & \quad \equiv \mathbf{let}\; bind\; \mathbf{in}\; e && \text{(by I.H.)} \end{aligned}$$

  We used in passing the fact that $D\left[\!\left[S[\![bind]\!]\right]\!\right] \equiv bind$, which (under the induction hypothesis) is obvious in both cases of $bind$.
- For $e \equiv e'\; e''$:

$$\begin{aligned} & D\left[\!\left[S[\![e'\; e'']\!]\right]\!\right] \\ & \quad \equiv D\left[\!\left[\mu\mathsf{ret}.\left\langle S[\![e']\!] \parallel S[\![e'']\!] \cdot \mathsf{ret}\right\rangle\right]\!\right] \\ & \quad \equiv D\left[\!\left[S[\![e'']\!] \cdot \mathsf{ret}\right]\!\right] [D\left[\!\left[S[\![e']\!]\right]\!\right]] \\ & \quad \equiv (D[\![\mathsf{ret}]\!]\; [\square\; D\left[\!\left[S[\![e'']\!]\right]\!\right]])[D\left[\!\left[S[\![e']\!]\right]\!\right]] \\ & \quad \equiv (\square\; D\left[\!\left[S[\![e'']\!]\right]\!\right])[D\left[\!\left[S[\![e']\!]\right]\!\right]] \\ & \quad \equiv D\left[\!\left[S[\![e']\!]\right]\!\right]\; D\left[\!\left[S[\![e'']\!]\right]\!\right] \\ & \quad \equiv e'\; e'' && \text{(by I.H.)} \end{aligned}$$

- The case for $e \equiv e'\; \tau$ is similar.
- For $e \equiv \mathbf{case}\; e'\; \mathbf{of}\; \overrightarrow{alt}$:

$$\begin{aligned} & D\left[\!\left[S\left[\!\left[\mathbf{case}\; e'\; \mathbf{of}\; \overrightarrow{alt}\right]\!\right]\right]\!\right] \\ & \quad \equiv D\left[\!\left[\mu\mathsf{ret}.\left\langle S[\![e']\!] \;\middle\|\; \mathbf{case\; of}\; \overrightarrow{S[\![alt]\!]}\right\rangle\right]\!\right] \\ & \quad \equiv D\left[\!\left[\mathbf{case\; of}\; \overrightarrow{S[\![alt]\!]}\right]\!\right] [D\left[\!\left[S[\![e']\!]\right]\!\right]] \\ & \quad \equiv (\mathbf{case}\; \square\; \mathbf{of}\; \overrightarrow{D\left[\!\left[S[\![alt]\!]\right]\!\right]})[D\left[\!\left[S[\![e']\!]\right]\!\right]] \\ & \quad \equiv \mathbf{case}\; D\left[\!\left[S[\![e']\!]\right]\!\right]\; \mathbf{of}\; \overrightarrow{D\left[\!\left[S[\![alt]\!]\right]\!\right]} \\ & \quad \equiv \mathbf{case}\; e'\; \mathbf{of}\; \overrightarrow{alt} && \text{(by I.H.)} \end{aligned}$$

  As with bindings, it is obvious that $D\left[\!\left[S[\![alt]\!]\right]\!\right] \equiv alt$.

2. We can assume without loss of generality that we're in the join-point-free fragment of the language, since then by Lemmas 29 and 30, we will have

$$S\left[\!\left[D[\![v]\!]\right]\!\right] \equiv S\left[\!\left[D\left[\!\left[V[\![v]\!]\right]\!\right]\right]\!\right] = V[\![v]\!] = v$$

(and similar statements for continuations and commands). Thus proceed by mutual induction on v, k, and c, assuming that none of them contain join points.

(a)
- All cases where v is a variable or WHNF are trivial.
- For $v \equiv \mu\mathsf{ret}.c$:

$$\begin{aligned} & S\left[\!\left[D[\![\mu\mathsf{ret}.c]\!]\right]\!\right] \\ & \quad \equiv S\left[\!\left[D[\![c]\!]\right]\!\right] \\ & \quad = \mu\mathsf{ret}.c && \text{(by (c))} \end{aligned}$$

(b)
- For $k \equiv \mathsf{ret}$:

$$\begin{aligned} & S\left[\!\left[D[\![\mathsf{ret}]\!]\;[e]\right]\!\right] \\ & \quad \equiv S[\![\square[e]]\!] \\ & \quad \equiv S[\![e]\!] \\ & \quad = \mu\mathsf{ret}.\left\langle S[\![e]\!] \parallel \mathsf{ret}\right\rangle \end{aligned}$$

- For $k \equiv v \cdot k'$:

$$\begin{aligned} & S\left[\!\left[D[\![v \cdot k']\!]\;[e]\right]\!\right] \\ & \quad \equiv S\left[\!\left[D[\![k']\!]\;[\square\; D[\![v]\!]][e]\right]\!\right] \\ & \quad \equiv S\left[\!\left[D[\![k']\!]\;[e\; D[\![v]\!]]\right]\!\right] \\ & \quad = \mu\mathsf{ret}.\left\langle S\left[\!\left[e\; D[\![v]\!]\right]\!\right] \parallel k'\right\rangle && \text{(by I.H.)} \\ & \quad \equiv \mu\mathsf{ret}.\left\langle \mu\mathsf{ret}.\left\langle S[\![e]\!] \parallel S\left[\!\left[D[\![v]\!]\right]\!\right] \cdot \mathsf{ret}\right\rangle \parallel k'\right\rangle \\ & \quad \to \mu\mathsf{ret}.\left\langle S[\![e]\!] \parallel S\left[\!\left[D[\![v]\!]\right]\!\right] \cdot k'\right\rangle \\ & \quad = \mu\mathsf{ret}.\left\langle S[\![e]\!] \parallel v \cdot k'\right\rangle && \text{(by (a))} \end{aligned}$$

- The case for $k \equiv \tau \cdot k'$ is similar.
- For $k = \mathbf{case\; of}\; \overrightarrow{alt}$:

$$\begin{aligned} & S\left[\!\left[D\left[\!\left[\mathbf{case\; of}\; \overrightarrow{alt}\right]\!\right]\;[e]\right]\!\right] \\ & \quad \equiv S\left[\!\left[(\mathbf{case}\; \square\; \mathbf{of}\; \overrightarrow{D[\![alt]\!]})[e]\right]\!\right] \\ & \quad \equiv S\left[\!\left[\mathbf{case}\; e\; \mathbf{of}\; \overrightarrow{D[\![alt]\!]}\right]\!\right] \\ & \quad \equiv \mu\mathsf{ret}.\left\langle S[\![e]\!] \;\middle\|\; \mathbf{case\; of}\; \overrightarrow{S\left[\!\left[D[\![alt]\!]\right]\!\right]}\right\rangle \\ & \quad = \mu\mathsf{ret}.\left\langle S[\![e]\!] \;\middle\|\; \mathbf{case\; of}\; \overrightarrow{alt}\right\rangle && \text{(by (c))} \end{aligned}$$

(c)
- For $c \equiv \mathbf{let}\; bind\; \mathbf{in}\; c$:

$$\begin{aligned} & S\left[\!\left[D[\![\mathbf{let}\; bind\; \mathbf{in}\; c]\!]\right]\!\right] \\ & \quad \equiv S\left[\!\left[\mathbf{let}\; D[\![bind]\!]\; \mathbf{in}\; D[\![c]\!]\right]\!\right] \\ & \quad \equiv \mathbf{let}\; S\left[\!\left[D[\![bind]\!]\right]\!\right]\; \mathbf{in}\; S\left[\!\left[D[\![c]\!]\right]\!\right] \\ & \quad = \mathbf{let}\; bind\; \mathbf{in}\; c && \text{(by (a), I.H.)} \end{aligned}$$

  In the last step, we use the assumption that there are no join points, and thus the binding is a value binding.
- For $c \equiv \langle v \parallel k\rangle$:

$$\begin{aligned} & S\left[\!\left[D[\![\langle v \parallel k\rangle]\!]\right]\!\right] \\ & \quad \equiv S\left[\!\left[D[\![k]\!]\;[D[\![v]\!]]\right]\!\right] \\ & \quad = \mu\mathsf{ret}.\left\langle S\left[\!\left[D[\![v]\!]\right]\!\right] \parallel k\right\rangle && \text{(by (b))} \\ & \quad = \mu\mathsf{ret}.\langle v \parallel k\rangle && \text{(by (a))} \end{aligned}$$

- The case where c is a jump is impossible by assumption.

□

Proof of Proposition 3. From Proposition 26 and Lemma 31 we get $S_a [\![D [\![v]\!]]\!] = S [\![D [\![v]\!]]\!] = v$. From Proposition 26, Lemma 27, and Lemma 31 we get $D [\![S_a [\![e]\!]]\!] = D [\![S [\![e]\!]]\!] = e$. Finally, from Propositions 22 and 25 we have $S_a [\![D [\![v]\!]]\!] \cong v$ and $D [\![S_a [\![e]\!]]\!] \cong e$. □

F. Proof of Well-Typed Translation (Proposition 4)

To show that the translations between Core and Sequent Core are well-typed, we need to refer to the type system for Core, which is illustrated in Figure 13. Notice that, except for lacking the type for jumps, Core has exactly the same rules for determining the kinds of types as Sequent Core from Figure 4.

We already have that S_a is equivalent to S (Proposition 26). We can make use of this to prove type safety of S_a from S by extending type preservation:

Proposition 32 (Preservation under $=$)**.** *If $\Gamma_1 \vdash v_1 : \tau_1$, $\Gamma_2 \vdash v_2 : \tau_2$, and $v_1 = v_2$, then $\tau_1 \equiv \tau_2$.*

Proof. Uniqueness of types (i.e., the case where $v_1 \equiv v_2$) is obvious, since the typing rules are syntax-directed. Thus if we find v with $v_1 \to^\star v \leftarrow^\star v_2$, we are done, since type preservation (Proposition 2)[3] says that v has the same type as both v_1 and v_2. But confluence gives us exactly such a v. □

Proving type safety of S is now straightforward.

Lemma 33 (Type safety of S)**.**
If $\Gamma \vdash e : \tau$ in Core, then $\Gamma \vdash S [\![e]\!] : \tau$ in Sequent Core.

Proof. An easy induction on the typing derivation. For example, to handle term application, suppose we have:

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \vdots\\ \Gamma \vdash e : \sigma \to \tau\end{array} \quad \begin{array}{c}\mathcal{E}\\ \vdots\\ \Gamma \vdash e' : \sigma\end{array}}{\Gamma \vdash e\, e' : \tau} \to\mathsf{E}$$

By the induction hypothesis, we then have:

$$\begin{array}{c}\mathcal{D}'\\ \vdots\\ \Gamma \vdash S [\![e]\!] : \sigma \to \tau\end{array} \qquad \begin{array}{c}\mathcal{E}'\\ \vdots\\ \Gamma \vdash S [\![e']\!] : \sigma\end{array}$$

Now, noting that

$$S [\![e\, e']\!] = \mu\mathsf{ret}. \left\langle S [\![e]\!] \,\middle\|\, S [\![e']\!] \cdot \mathsf{ret} \right\rangle,$$

we have:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}'\\ \vdots\\ \Gamma \vdash S [\![e]\!] : \sigma \to \tau\end{array} \quad \dfrac{\begin{array}{c}\mathcal{E}'\\ \vdots\\ \Gamma \vdash S [\![e']\!] : \sigma\end{array} \quad \dfrac{}{\Gamma \mid \mathsf{ret} : \tau \vdash \mathsf{ret} : \tau}\,\mathsf{Ret}}{\Gamma \mid S [\![e']\!] \cdot \mathsf{ret} : \sigma \to \tau \vdash \mathsf{ret} : \tau}\to\mathsf{L}}{\left\langle S [\![e]\!] \,\middle\|\, S [\![e']\!] \cdot \mathsf{ret} \right\rangle : (\Gamma \vdash \mathsf{ret} : \tau)}\,\mathsf{Cut}}{\Gamma \vdash \mu\mathsf{ret}. \left\langle S [\![e]\!] \,\middle\|\, S [\![e']\!] \cdot \mathsf{ret} \right\rangle : \tau}\,\mathsf{Act}$$

□

Proving type safety of D hits a snag: while D does not change the type of a term, it *does* change the type of a join point. Namely, if a join point has type $\exists\vec{a}.(\vec{\sigma})$ and its context gives ret the type τ, it will become a function of type $\forall\vec{a}.\vec{\sigma} \to \tau$. Thus we define D_τ on types, homomorphically except for

$$D_\tau [\![\exists\vec{a}.(\vec{\sigma})]\!] \equiv \forall\vec{a}.\vec{\sigma} \to \tau.$$

Then, we have D operate on continuation contexts:

$$D \left[\!\!\left[\overrightarrow{j : \sigma}, \mathsf{ret} : \tau \right]\!\!\right] \equiv D_\tau \left[\!\!\left[\overrightarrow{j : \sigma} \right]\!\!\right] \equiv \overrightarrow{j : D_\tau [\![\sigma]\!]}.$$

Now we can state and prove the general form of type safety for D:

Lemma 34 (Type safety of D)**.** *1. If $\Gamma \vdash v : \tau$, then $\Gamma \vdash D [\![v]\!] : \tau$.*

2. If $\Gamma \mid k : \sigma \vdash \Delta, \mathsf{ret} : \tau$ and $\Gamma \vdash e : \sigma$, then $\Gamma, D_\tau [\![\Delta]\!] \vdash D [\![k]\!] [e] : \tau$.

3. If $c : (\Gamma \vdash \Delta, \mathsf{ret} : \tau)$, then $\Gamma, D_\tau [\![\Delta]\!] \vdash D [\![c]\!] : \tau$.

4. If $bind : (\Gamma \mid \Delta' \vdash \Gamma' \mid \Delta, \mathsf{ret} : \tau)$, then $\Gamma, D_\tau [\![\Delta]\!] \vdash D [\![bind]\!] : \Gamma', D_\tau [\![\Delta']\!]$.

Proof. By mutual induction on the typing derivations. We show a few cases:

- In 2, suppose we have

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \vdots\\ \Gamma \vdash v : \sigma\end{array} \quad \begin{array}{c}\mathcal{E}\\ \vdots\\ \Gamma \mid k : \sigma' \vdash \Delta, \mathsf{ret} : \tau\end{array}}{\Gamma \mid v \cdot k : \sigma \to \sigma' \vdash \Delta, \mathsf{ret} : \tau} \to\mathsf{L}$$

  and also:

$$\begin{array}{c}\mathcal{F}\\ \vdots\\ \Gamma \vdash e : \sigma \to \sigma'\end{array}$$

  By the induction hypothesis, we then have:

$$\begin{array}{c}\mathcal{D}'\\ \vdots\\ \Gamma \vdash D [\![v]\!] : \sigma\end{array} \qquad \begin{array}{c}\Gamma \vdash e\, D [\![v]\!] : \sigma'\\ \vdots\\ \mathcal{E}'\\ \vdots\\ \Gamma, D_\tau [\![\Delta]\!] \vdash D [\![k]\!] [e\, D [\![v]\!]] : \tau\end{array}$$

  Noting that

$$D [\![v \cdot k]\!] [e] \equiv D [\![k]\!] [e\, D [\![v]\!]],$$

  we see that $\mathcal{E}'$ is has the conclusion we require, so long as we can prove its premise. Thus we write:

$$\begin{array}{c}\dfrac{\begin{array}{c}\mathcal{F}\\ \vdots\\ \Gamma \vdash e : \sigma \to \sigma'\end{array} \quad \begin{array}{c}\mathcal{D}'\\ \vdots\\ \Gamma \vdash D [\![v]\!] : \sigma\end{array}}{\Gamma \vdash e\, D [\![v]\!] : \sigma'} \to\mathsf{E}\\ \vdots\\ \mathcal{E}'\\ \vdots\\ \Gamma, D_\tau [\![\Delta]\!] \vdash D [\![k]\!] [e\, D [\![v]\!]] : \tau\end{array}$$

- In 3, suppose we have:

$$\dfrac{\begin{array}{c}\vec{\mathcal{D}}\\ \vdots\\ \overrightarrow{\Gamma \vdash \sigma' : \kappa}\end{array} \quad \begin{array}{c}\vec{\mathcal{E}}\\ \vdots\\ \overrightarrow{\Gamma \vdash v : \sigma\{\overrightarrow{\sigma'/a}\}}\end{array}}{\mathsf{jump}\, j\, \vec{\sigma'}\, \vec{v} : (\Gamma \vdash j : \exists\overrightarrow{a{:}\kappa}.(\vec{\sigma}), \Delta, \mathsf{ret} : \tau)}\,\mathsf{Jump}$$

[3] In fact, we need to extend type preservation from $\mapsto$ to $\to$, but this is trivial since $\to$ adds only the μ_η-rule (easily verified) and compatibility, and our type system is compositional.

$$\Gamma \in \mathit{Environment} ::= \varepsilon \mid \Gamma, x:\tau \mid \Gamma, a:\kappa \mid \Gamma, K:\tau \mid \Gamma, T:\kappa$$

Type kinding: $\boxed{\Gamma \vdash \tau : \kappa}$

$$\frac{}{\Gamma, a:\kappa \vdash a:\kappa}\ \mathsf{TyVar} \qquad \frac{}{\Gamma, T:\kappa \vdash T:\kappa}\ \mathsf{TyCon} \qquad \frac{\Gamma\vdash\sigma:\kappa'\to\kappa \quad \Gamma\vdash\tau:\kappa'}{\Gamma\vdash\sigma\,\tau:\kappa}\ \mathsf{TyApp} \qquad \frac{\Gamma,a:\kappa\vdash\tau:\star}{\Gamma\vdash\forall a{:}\kappa.\tau:\star}\ \forall$$

Expression typing: $\boxed{\Gamma \vdash e : \tau}$

$$\frac{}{\Gamma, x:\tau \vdash x:\tau}\ \mathsf{Var} \qquad \frac{\Gamma\vdash \mathit{bind}:\{\Gamma'\} \quad \Gamma,\Gamma'\vdash e':\sigma}{\Gamma\vdash \textbf{let}\,\mathit{bind}\,\textbf{in}\,e':\sigma}\ \mathsf{Let}$$

$$\frac{\Gamma,x:\tau\vdash e:\sigma}{\Gamma\vdash\lambda x{:}\tau.e:\tau\to\sigma}\ \to\!\mathsf{I} \qquad \frac{\Gamma\vdash e:\tau\to\sigma \quad \Gamma\vdash e':\tau}{\Gamma\vdash e\,e':\sigma}\ \to\!\mathsf{E} \qquad \frac{\Gamma,a:\kappa\vdash e:\tau}{\Gamma\vdash\Lambda a{:}\kappa.e:\forall a{:}\kappa.\tau}\ \forall\mathsf{I} \qquad \frac{\Gamma\vdash e:\forall a{:}\kappa.\tau \quad \Gamma\vdash\sigma:\kappa}{\Gamma\vdash e\,\sigma:\tau\,\{\sigma/a\}}\ \forall\mathsf{E}$$

$$\frac{K:\forall\overrightarrow{a{:}\kappa}.\forall\overrightarrow{b{:}\kappa'}.\overrightarrow{\tau'}\to T\,\vec{a} \quad \overrightarrow{\Gamma\vdash\sigma:\kappa'} \quad \overrightarrow{\Gamma\vdash e:\tau'\overrightarrow{\{\tau/a\}}\overrightarrow{\{\sigma/b\}}}}{\Gamma\vdash K\ \vec{\sigma}\ \vec{e}\,:\,T\,\vec{\tau}}\ \mathsf{TKI} \qquad \frac{\Gamma\vdash e:\tau \quad \overrightarrow{\Gamma\mid\tau\vdash \mathit{alt}:\sigma}}{\Gamma\vdash\textbf{case}\,e\,\textbf{of}\,\overrightarrow{\mathit{alt}}:\sigma}\ \mathsf{Case}$$

Alternative typing: $\boxed{\Gamma \mid \tau \vdash e : \sigma}$

$$\frac{\Gamma,x:\tau\vdash e:\sigma}{\Gamma\mid\tau\vdash x{:}\tau\to e:\sigma}\ \mathsf{Deflt} \qquad \frac{K:\forall\overrightarrow{a{:}\kappa'}.\forall\overrightarrow{b{:}\kappa}.\vec{\sigma}\to T\,\vec{a} \quad \Gamma,\overrightarrow{b:\kappa},\overrightarrow{x:\sigma\overrightarrow{\{\tau/a\}}}\vdash e:\sigma'}{\Gamma\mid T\,\vec{\tau}\vdash K\,\overrightarrow{b{:}\kappa}\,\overrightarrow{x{:}\sigma}\to e:\sigma'}\ \mathsf{TKE}$$

Binding typing: $\boxed{\Gamma \vdash \mathit{bind} : \{\Gamma'\}}$

$$\frac{\Gamma\vdash e:\tau}{\Gamma\vdash x{:}\tau=e:\{x:\tau\}}\ \mathsf{Name} \qquad \frac{\overrightarrow{\Gamma,\overrightarrow{x:\tau}\vdash e:\tau}}{\Gamma\vdash\textbf{rec}\,\{\overrightarrow{x{:}\tau=e}\}:\{\Gamma'\}}\ \mathsf{Rec}$$

Figure 13. Type system for Core

$$\dfrac{\dfrac{\dfrac{}{\Gamma'\vdash j:\forall\overrightarrow{a:\kappa}.\vec{\sigma}\to\tau}\ \mathsf{Var} \quad \begin{array}{c}\vec{\mathcal{D}}\\ \vdots\\ \overrightarrow{\Gamma\vdash\sigma':\kappa}\end{array}}{\Gamma'\vdash j\,\overrightarrow{\sigma'}:\sigma\overrightarrow{\{\sigma'/a\}}\to\tau}\ \overrightarrow{\forall\mathsf{E}} \quad \begin{array}{c}\overrightarrow{\mathcal{E}'}\\ \vdots\\ \overrightarrow{\Gamma\vdash D\,[\![v]\!]:\sigma\overrightarrow{\{\sigma'/a\}}}\end{array}}{\Gamma'\vdash j\,\overrightarrow{\sigma'}\,\overrightarrow{D\,[\![v]\!]}:\tau}\ \overrightarrow{\to\mathsf{E}}$$

Figure 14. Proof of Lemma 34, jump case

By the induction hypothesis, we have:

$$\begin{array}{c}\overrightarrow{\mathcal{E}'}\\ \vdots\\ \overrightarrow{\Gamma\vdash D\,[\![v]\!]:\sigma\overrightarrow{\{\sigma'/a\}}}\end{array}$$

Noting that

$$D\left[\!\!\left[\mathsf{jump}\,j\,\overrightarrow{\sigma'}\,\vec{v}\right]\!\!\right]\equiv j\,\overrightarrow{\sigma'}\,\overrightarrow{D\,[\![v]\!]}$$

and

$$D_\tau\,[\![j:\exists\overrightarrow{a:\kappa}.(\vec{\sigma})]\!]\equiv j:\forall\overrightarrow{a:\kappa}.\vec{\sigma}\to\tau,$$

letting

$$\Gamma'\equiv\Gamma,j:\forall\overrightarrow{a:\kappa}.\vec{\sigma}\to\tau,D_\tau\,[\![\Delta]\!]\,,$$

we then have the derivation in Figure 14[4].

- In 4, suppose we have:

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \vdots\\ c:(\Gamma,\overrightarrow{a:\kappa},\overrightarrow{x:\sigma}\vdash\Delta,\mathsf{ret}:\tau)\end{array}}{(j=\tilde{\mu}[\overrightarrow{a{:}\kappa},\overrightarrow{x{:}\sigma}].c):(\Gamma\mid j:\exists\overrightarrow{a{:}\kappa}.(\vec{\sigma})\vdash\varepsilon\mid\Delta,\mathsf{ret}:\tau)}\ \mathsf{Label}$$

By the induction hypothesis, we have:

$$\begin{array}{c}\mathcal{D}'\\ \vdots\\ \Gamma,\overrightarrow{a:\kappa},\overrightarrow{x:\sigma},D_\tau\,[\![\Delta]\!]\vdash D\,[\![c]\!]:\tau\end{array}$$

Noting that

$$D\,[\![j=\tilde{\mu}[\overrightarrow{a{:}\kappa},\overrightarrow{x{:}\sigma}].c]\!]\equiv\big(j=\Lambda\overrightarrow{a{:}\kappa}.\lambda\overrightarrow{x{:}\sigma}.D\,[\![c]\!]\big)$$

and

$$D_\tau\,[\![j:\exists\overrightarrow{a:\kappa}.(\vec{\sigma})]\!]\equiv j:\forall\overrightarrow{a:\kappa}.\vec{\sigma}\to\tau,$$

we then have:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\mathcal{D}'\\ \vdots\\ \Gamma,\overrightarrow{a{:}\kappa},\overrightarrow{x{:}\sigma},D_\tau\,[\![\Delta]\!]\vdash D\,[\![c]\!]:\tau\end{array}}{\Gamma,\overrightarrow{a{:}\kappa},D_\tau\,[\![\Delta]\!]\vdash\lambda\overrightarrow{x{:}\sigma}.D\,[\![c]\!]:\vec{\sigma}\to\tau}}{\Gamma,D_\tau\,[\![\Delta]\!]\vdash\Lambda\overrightarrow{a{:}\kappa}.\lambda\overrightarrow{x{:}\sigma}.D\,[\![c]\!]:\forall\overrightarrow{a{:}\kappa}.\vec{\sigma}\to\tau}\ \to\!\mathsf{I}}{\Gamma,D_\tau\,[\![\Delta]\!]\vdash(j=\Lambda\overrightarrow{a{:}\kappa}.\lambda\overrightarrow{x{:}\sigma}.D\,[\![c]\!]):(j:\forall\overrightarrow{a{:}\kappa}.\vec{\sigma}\to\tau)}\ \forall\mathsf{I}$$

□

Proof of Proposition 4. Immediate from Lemmas 33 and 34. □

[4] The reader may notice we make implicit use of weakening in this derivation.