DAG-Calculus: A Calculus for Parallel Computation

Technical Appendix

The proofs of translations to the DAG calculus presented in this appendix, involve one key difficulty, related to administrative reduction steps, i.e. to the fact that one reduction step in the source language may correpond to several reduction steps in the target language. Most compiler proofs deal with administrative reduction steps using well-known proofs techniques, typically based on simulation diagrams. However, we have found that these techniques were not directly applicable to the parallel semantics of a language such as that the fork-join language considered here.

When the target program takes a reduction step, this step corresponds either to an administrative step, or to a real step from the source program. Consider the latter case. With a sequential semantics, when the target program takes a real step, the target program then typically corresponds exactly to the translation of the source program. However, with a parallel semantics, this might not be the case. For example, since the two branches of a parallel pair may reduce independently, one branch may take a real step while the other branch is in the middle of performing a sequence of administrative steps. Although the target program takes a real step, it is not, at this point, the translation of any source program. Thus, we cannot easily close a simulation diagram.

To address this challenge, we introduce an *instrumented* programming language, similar to the source programming language, except that each parallel construct gets annotated with information about identities associated with its representation in the target language: number of administrative steps already performed, thread identifiers, locations for the results, etc. We then set up a two-layer simulation diagram: the first layer relates the source program with the instrumented program, while the second layer relates the instrumented program with the target program. We are able to reason about both layers independently, using conventional simulation diagrams, and then conclude by relating the source and the target programs.

A The DAG Calculus, λ^{DC}

Here we present the syntax and semantics of the DAG calculus, which we denote in short λ^{DC}. These constructs and rules are described in detail in Section 3 of the paper.

Note that while the syntax of the calculus is identical to the one presented in the paper, the formulation of the reduction rules is slightly different: the DAG-modification constructs are assumed to always appear in the context of a let-binding or sequential composition, and the reduction reduces them together with the binding/composition.

$$
\begin{aligned}
e &::= v \mid e \oplus e \mid e \otimes e \mid \langle e, e \rangle \mid \pi_i\ e \mid\ e\ e \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid \\
&\quad \texttt{alloc} \mid e := e \mid\ !e \mid \texttt{capture}\ e \mid \texttt{abort}\ e \mid \\
&\quad \texttt{newTd}\ e \mid \texttt{release}\ e \mid \texttt{newEdge}\ e\ e \mid \texttt{transfer}\ e \\
v &::= x \mid \ell \mid \texttt{t} \mid \texttt{n} \mid \texttt{()} \mid (v, v) \mid \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end} \\
K &::= \bullet \mid \texttt{let}\ x = K\ \texttt{in}\ e \mid K := e \mid v := K \mid\ !\,K \mid \texttt{release}\ K \\
&\quad \mid \texttt{newEdge}\ K\ e \mid \texttt{newEdge}\ v\ K \mid \texttt{transfer}\ K \mid \ldots
\end{aligned}
$$

Figure 1: Abstract syntax for DAG-Calculus.

$$
\frac{}{\sigma, \texttt{fst}\ (v_1, v_2) \to v_1, \sigma}\ \textsc{Fst} \qquad \frac{}{\sigma, \texttt{snd}\ (v_1, v_2) \to v_2, \sigma}\ \textsc{Snd} \qquad \frac{l \notin \mathrm{dom}(\sigma)}{\sigma, \texttt{alloc} \to l, \sigma[l \mapsto \texttt{()}]}\ \textsc{Alloc}
$$

$$
\frac{\sigma(l) = v}{\sigma, !\,l \to v, \sigma}\ \textsc{Deref} \qquad \frac{l \in \mathrm{dom}(\sigma)}{\sigma,\ (l := v) \to \texttt{()}, \sigma[l \mapsto v]}\ \textsc{Assign}
$$

$$
\frac{}{\sigma, ((\texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end})\ v) \to e[f \mapsto \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end}][x \mapsto v], \sigma}\ \textsc{Apply}
$$

$$
\frac{k\ =\ \texttt{fun}\ f\ x\ \texttt{is abort}\ (K[x])\ \texttt{end}}{\sigma, K[\texttt{capture}\ (\lambda x.\ e)] \to e[x \mapsto k], \sigma}\ \textsc{Capture} \qquad \frac{}{\sigma, K[\texttt{abort}\ e] \to e, \sigma}\ \textsc{Abort}
$$

$$
\frac{\sigma, e \to e', \sigma' \qquad \forall K', e''.\ e \notin \{K'[\texttt{capture}\ e''], K'[\texttt{abort}\ e'']\}}{\sigma, K[e] \to K[e'], \sigma'}\ \textsc{Context}
$$

$$
\frac{V(t) = (e_1, \mathsf{X}) \qquad \sigma_1, e_1 \to e_2, \sigma_2}{V, E, \sigma_1 \twoheadrightarrow V[t \mapsto (e_2, \mathsf{X})], E, \sigma_2}\ \textsc{Step} \qquad \frac{V(t) = (v, \mathsf{X}) \qquad E' = E \setminus \{(t, t') \mid t' \in \mathrm{dom}(V)\}}{V, E, \sigma \twoheadrightarrow V[t \mapsto (\texttt{()}, \mathsf{F})], E', \sigma}\ \textsc{Stop}
$$

$$
\frac{V(t) = (K[\texttt{let}\ x = \texttt{newTd}\ e\ \texttt{in}\ e'], \mathsf{X}) \qquad t'\ \ \mathsf{fresh}}{V, E, \sigma \twoheadrightarrow V[t \mapsto (K[e'[x \mapsto t']], \mathsf{X})][t' \mapsto (e, \mathsf{N})], E, \sigma}\ \textsc{NewTd} \qquad \frac{V(t) = (e, \mathsf{R}) \qquad \{t' \mid (t', t) \in E\} = \emptyset}{V, E, \sigma \twoheadrightarrow V[t \mapsto (e, \mathsf{X})], E, \sigma}\ \textsc{Start}
$$

$$
\frac{V(t) = (K[\texttt{release}\ t'; e], \mathsf{X}) \qquad V(t') = (e', \mathsf{N})}{V, E, \sigma \twoheadrightarrow V[t \mapsto (K[e], \mathsf{X})][t' \mapsto (e', \mathsf{R})], E, \sigma}\ \textsc{Release}
$$

$$
\frac{\begin{array}{c} V(t) = (K[\texttt{newEdge}\ t_1\ t_2; e], \mathsf{X}) \qquad t_1, t_2 \in \mathrm{dom}(V) \qquad \mathrm{status}(V(t_2)) \in \{\mathsf{N}, \mathsf{R}\} \\ E'\ =\ E \uplus (\text{if } \mathrm{status}(V(t_1)) = \mathsf{F} \text{ then } \emptyset \text{ else } \{(t_1, t_2)\}) \qquad E' \text{ cycle-free} \end{array}}{V, E, \sigma \twoheadrightarrow V[t \mapsto (K[e], \mathsf{X})], E', \sigma}\ \textsc{NewEdge}
$$

$$
\frac{\begin{array}{c} V(t) = (K[\texttt{transfer}\ t'; e], \mathsf{X}) \qquad \mathrm{status}(V(t')) = \mathsf{N} \qquad \{t'' \mid (t', t'') \in E\} = \emptyset \\ E'\ =\ E \setminus \{(t, t'') \mid t'' \in \mathrm{dom}(V)\} \uplus \{(t', t'') \mid (t, t'') \in E\} \qquad E' \text{ cycle-free} \end{array}}{V, E, \sigma \twoheadrightarrow V[t \mapsto (K[e], \mathsf{X})], E', \sigma}\ \textsc{Transfer}
$$

Figure 2: Dynamic semantics for λ^{DC}. N stands for thread status *new*, R for *released*, X for *executing* and F for *finished*. The operation status($V(t)$) denotes the status of thread t, i.e. the second component of $V(t)$.

This approach is clearly equivalent to the presentation in the paper, but it significantly reduces the number of otherwise not interesting intermediate steps that would be possible to encounter in the translations, thus shortening the translations and the proofs significantly.

Definition 1. A *computation DAG* is a pair (V, E), where the *vertex-map* V is a finite map from threads to a pair consisting of expressions, and status and *edges* E is a edges (edges) between threads, more precisely:

$$\begin{array}{lcl} s & ::= & \mathsf{N} \mid \mathsf{R} \mid \mathsf{X} \mid \mathsf{F} \\ V & : & \mathcal{T} \rightharpoonup_{\mathit{fin}} (e \times s) \\ E & \subseteq & \mathrm{dom}(V) \times \mathrm{dom}(V). \end{array}$$

Definition 2. The operational semantics of λ^{DC} is a relation $\twoheadrightarrow$ between states made of a computation DAG and a store mapping locations to values. It is given in Figure 2. We take a subset of the transitions Start and Stop as *scheduler* transitions, written $\twoheadrightarrow_{\mathrm{s}}$.

Lemma 1. *For any DAG* (V, E) *and any state* σ, $V, E, \sigma \not\twoheadrightarrow_{\mathrm{s}}^{\infty}$.

Proof. Give weight to a dag by taking for each vertex weight according to its status:

$$w(s) = \begin{cases} 0 & \text{if } s = \mathsf{N} \text{ or } s = \mathsf{F} \\ 1 & \text{if } s = \mathsf{X} \\ 2 & \text{if } s = \mathsf{R} \end{cases}$$

Take the weight of the DAG as the sum of the weights of its vertices, then proceed by induction on the weight of the DAG. Observe that each scheduler transition decreases the weight of the node it operates on, and hence the whole DAG, which ends the proof. □

B Fork-Join

B.1 Syntax and Operational Semantics

The syntax and semantics in this section are precisely the same as in the paper, and repeated here for convenience.

$$\begin{array}{rcl} e & ::= & v \mid \mathtt{fst}\ v \mid \mathtt{snd}\ v \mid \mathtt{let}\ x = e\ \mathtt{in}\ e \mid v\ v \mid \ \ e \parallel e \\ v & ::= & x \mid n \mid (v, v) \mid \mathtt{fun}\ f\ x\ \mathtt{is}\ e\ \mathtt{end} \\ K & ::= & \bullet \mid \mathtt{let}\ x = K\ \mathtt{in}\ e \mid \ K \parallel e \ \mid \ e \parallel K \end{array}$$

$$\begin{array}{rcl} K[\mathtt{fst}\ (v_1, v_2)] & \to & K[v_1] \\ K[\mathtt{snd}\ (v_1, v_2)] & \to & K[v_2] \\ K[\mathtt{let}\ x = v\ \mathtt{in}\ e] & \to & K[e[x \mapsto v]] \\ K[\mathtt{fun}\ f\ x\ \mathtt{is}\ e\ \mathtt{end}\ v] & \to & K[e[x \mapsto v, f \mapsto \mathtt{fun}\ f\ x\ \mathtt{is}\ e\ \mathtt{end}]] \\ K[\ v_1 \parallel v_2\ ] & \to & K[(v_1, v_2)] \end{array}$$

B.2 Instrumented Syntax and Operational Semantics

The general pattern of the instrumentation, both for fork-join parallelism and for the other constructs, is to annotate the parallel constructs, which need to be compiled in the translation to λ^{DC}, with additional information. The crucial part of these annotations is a number that describes how far the evaluation of the compiled code has progressed. Intuitively, 0 always corresponds to an expression, whose compiled equivalent has not yet started execution, and each consecutive step of compiled execution takes the number one step higher. If in the evaluation process new information is obtained, such as identities of relevant threads, locations, etc., these are also included in the annotation (cf. the grammar of a_p, the annotations on parallel compositions, below)

Since in λ^{DC} the program parts are stored in the vertices of the DAG, parts of the program that correspond to separate vertices also need to be annotated. In case that values are passed through the store, like in the compiled version of fork-join, these also need to be present in the annotation, and some additional steps may be possible (cf. the grammar for a_t, the annotations of expressions that correspond to the λ^{DC} threads, below). In the particular case of fork-join, 0 means that the expression has not yet had a thread allocated to it and released, $(1, l, t)$ — that the expression is running in thread t and will write its result to location l, and $(2, l)$ — that the expression has finished the evaluation, and the result *has been written* to l. The extra step required to perform the write action is why we increment the number in the annotation here.

$$
\begin{aligned}
e &::= v \mid \texttt{fst}\ v \mid \texttt{snd}\ v \mid v\ v \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid (\, e^{a_t} \parallel e^{a_t}\,)^{a_p} \\
v &::= x \mid n \mid (v, v) \mid \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end} \\
L &::= \bullet \mid \texttt{let}\ x = L\ \texttt{in}\ e \\
K &::= \bullet \mid L^{a_t}[(\, K \parallel e^{a_t}\,)^{a_p}] \mid L^{a_t}[(\, e^{a_t} \parallel K\,)^{a_p}] \\
a_t &::= 0 \mid (1, l, t) \mid (2, l) \\
a_p &::= 0 \mid 1 \mid (2, l) \mid (3, l, l) \mid (4, l, l, t) \mid (5, l, l, t, t) \mid (n, l, l, t, t, t) \text{ where } 6 \le n \le 10 \\
&\quad \mid (11, l, l, t, t) \mid (n, l, l, t) \text{ where } 12 \le n \le 14
\end{aligned}
$$

A *surface* expression is one where all annotations have value 0. Note that these are isomorphic to the expressions of the source language. We define well-formed expressions as follows:

$$
\mathrm{wf}(e) \equiv \begin{cases}
a_1 = 0 \wedge a_2 = 0 \wedge \mathrm{surf}(e_1) \wedge \mathrm{surf}(e_2) \wedge \mathrm{surf}(L) & \text{if } e = L[\, e_1^{a_1} \parallel e_2^{a_2}\ {}^{a}] \text{ and } \mathrm{num}(a) \le 10 \\
\mathrm{loc}(a_1) = l_1 \wedge a_2 = 0 \wedge \mathrm{wf}(e_1) \wedge \mathrm{surf}(e_2) \wedge \mathrm{surf}(L) & \text{if } e = L[\, e_1^{a_1} \parallel e_2^{a_2\ (11, l_1, l_2, t)}] \\
\mathrm{loc}(a_1) = l_1 \wedge \mathrm{loc}(a_2) = l_2 \wedge \mathrm{wf}(e_1) \wedge \mathrm{wf}(e_2) \wedge \mathrm{surf}(L) & \text{if } e = L[\, e_1^{a_1} \parallel e_2^{a_2\ (12, l_1, l_2, t)}] \\
a_1 = (2, l_1) \wedge a_2 = (2, l_2) \wedge \mathrm{surf}(e_1) \wedge \mathrm{surf}(e_2) \wedge \mathrm{surf}(L) & \text{if } e = L[\, e_1^{a_1} \parallel e_2^{a_2\ (n, l_1, l_2, t)}] \text{ and } 13 \le n \\
\mathrm{surf}(e) & \text{otherwise}
\end{cases}
$$

We extend well-formedness to annotated expressions e^{a_t} by setting $\mathrm{wf}(e^a) \equiv a \neq 0$, and to parallel contexts, by setting $\mathrm{wf}(K) \equiv \forall e.\ \mathrm{wf}(e) \implies \mathrm{wf}(K[e])$.

Operational semantics is defined on annotated, well-formed expressions, e^{a_t}. First,

we define some of the transitions between the parallel composition annotations.

$$0 \rightsquigarrow_p 1 \qquad 1 \rightsquigarrow_p (2, l_1) \text{ where } l_1 \text{ fresh} \qquad (2, l_1) \rightsquigarrow_p (3, l_1, l_2) \text{ where } l_2 \text{ fresh}$$
$$(3, l_1, l_2) \rightsquigarrow_p (4, l_1, l_2, t_1) \text{ where } t_1 \text{ fresh} \qquad (4, l_1, l_2, t_1) \rightsquigarrow_p (5, l_1, l_2, t_1, t_2) \text{ where } t_2 \text{ fresh}$$
$$(5, l_1, l_2, t_1, t_2) \rightsquigarrow_p (6, l_1, l_2, t_1, t_2, t) \text{ where } t \text{ fresh}$$
$$(n, l_1, l_2, t_1, t_2, t) \rightsquigarrow_p (n+1, l_1, l_2, t_1, t_2, t) \text{ for } 6 \le n \le 10 \qquad (13, l_1, l_2, t) \rightsquigarrow_p (14, l_1, l_2, t)$$

These transitions can be used to make simple administrative reductions on the parallel composition form, as seen in the operational semantics below. Since the transitions through the remaining three steps are not entirely local, they are defined directly in the rules.

$$\texttt{fst}\ (v_1, v_2) \bigcirc\!\!\!\!\rightarrow v_1 \qquad \texttt{snd}\ (v_1, v_2) \bigcirc\!\!\!\!\rightarrow v_2 \qquad \texttt{let}\ x = v\ \texttt{in}\ e \bigcirc\!\!\!\!\rightarrow e[x \mapsto v]$$
$$\texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end}\ v \bigcirc\!\!\!\!\rightarrow e[x \mapsto v, f \mapsto \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end}]$$

$$\frac{a \rightsquigarrow_p a'}{(\, e_1^{a_1} \parallel e_2^{a_2}\,)^{a} \bigcirc\!\!\!\!\rightarrow (\, e_1^{a_1} \parallel e_2^{a_2}\,)^{a'}} \qquad (\, e_1^{(0)} \parallel e_2^{(0)}\,)^{(10,l_1,l_2,t_1,t_2,t)} \bigcirc\!\!\!\!\rightarrow (\, e_1^{(1,l_1,t_1)} \parallel e_2^{(0)}\,)^{(11,l_1,l_2,t_2,t)}$$
$$(\, e_1^{a_1} \parallel e_2^{(0)}\,)^{(11,l_1,l_2,t_2,t)} \bigcirc\!\!\!\!\rightarrow (\, e_1^{a_1} \parallel e_2^{(1,l_2,t_2)}\,)^{(12,l_1,l_2,t)}$$
$$(\, v_1^{(2,l_1)} \parallel v_2^{(2,l_2)}\,)^{(12,l_1,l_2,t)} \bigcirc\!\!\!\!\rightarrow (\, v_1^{(2,l_1)} \parallel v_2^{(2,l_2)}\,)^{(13,l_1,l_2,t)}$$

$$\frac{\mathrm{wf}(K) \qquad e \bigcirc\!\!\!\!\rightarrow e'}{K[(L[e])^{(1,l,t)}] \to K[(L[e'])^{(1,l,t)}]} \qquad \frac{\mathrm{wf}(K)}{K[v^{(1,l,t)}] \to K[v^{(2,l)}]}$$

$$\frac{\mathrm{wf}(K)}{K[(L[(\, v_1^{(2,l_1)} \parallel v_2^{(2,l_2)}\,)^{(14,l_1,l_2,t)}])^{(1,l,t')}] \to K[(L[(v_1, v_2)])^{(1,l,t)}]}$$

B.3 Connecting the instrumented and source languages

We define a map $\lfloor - \rfloor : \mathcal{E}_I \to \mathcal{E}_O$ that removes annotations from the instrumented expressions. We use it to connect the instrumented language to the source language. Note that a map that adds an annotation 0 to parallel compositions is the right inverse of $\lfloor - \rfloor$. For *surface* term, it is also the left inverse.

Lemma 2 (Instr-Step)**.** *For any two expressions $e, e' \in \mathcal{E}_I$ and annotations $a, a' \in \mathcal{A}_t$, if $e^a \to_I e'^{a'}$, then either $\lfloor e \rfloor = \lfloor e' \rfloor$ or $\lfloor e \rfloor \to_O \lfloor e' \rfloor$.*

Proof. By cases on the reduction. If the reduction is one that changes the annotation on a parallel pair or a thread, the erasure is trivially preserved. In the other cases, we can simply pick the corresponding reduction rule to match the erasure of the right-hand side. □

Definition 3. An *administrative* reduction in the instrumented language is a reduction $e^a \to_I e'$ such that $\lfloor e \rfloor = \lfloor e' \rfloor$. We write $e^a \to_a e'^{a'}$ for administrative steps.

Lemma 3 (Admin-Fin)**.** *For well-formed annotated expressions e^a, there are no infinite sequence of administrative reductions, i.e., if* wf e^a *then $e^a \not\to_a^\infty$.*

Proof. Assign a weight to the expression based on the count of its annotations in evaluation positions. Take num(a) to be the natural number in the annotation. Then define

$$w(e) = \begin{cases} 18 - \mathrm{num}(a) - \mathrm{num}(a_1) - \mathrm{num}(a_2) + w(e_1) + w(e_2) & \text{if } e = (\,(e_1)^{a_1} \parallel (e_2)^{a_2}\,) \\ w(e_1) & \text{if } e = \texttt{let } x = e_1 \texttt{ in } e_2 \\ 0 & \text{otherwise} \end{cases}$$

and extend it to annotated threads as $w(e^a) \equiv 2 - \mathrm{num}(a) + w(e)$. Observe that all administrative reductions decrease the weight of the expression. Thus, by induction on the weight we can conclude that the lemma holds. □

Lemma 4 (Instrument)**.** *For any expression $e \in \mathcal{E}_I$, value $v \in \mathcal{V}_I$ annotation $a \in \mathcal{A}_t$ and location l, if $e^a \to_I^* v^{(2,l)}$ then $\lfloor e \rfloor \to_O^* \lfloor v \rfloor$. Moreover, if $e^a \to_I^\infty$, then $\lfloor e \rfloor \to_O^\infty$.*

Proof. First part, by induction on the reduction sequence and Lemma 2.

Similarly, for the second part we proceed by coinduction: since $e \to_I^\infty$, by Lemmas 3 and 2 there exists e' such that $e \to_I^* e'$, $\lfloor e \rfloor \to_O \lfloor e' \rfloor$ and $e' \to_I^\infty$. Hence, $\lfloor e' \rfloor \to_O^\infty$ and finally $\lfloor e \rfloor \to_O^\infty$. □

B.4 Connecting instrumented language to λ^{DC}

Translation of surface expressions into λ^{DC} First, let us define the translation for surface expressions. We define a map $[\![-]\!]^S : \mathcal{E}_I \to \mathcal{E}_\lambda^{DC}$, as follows. Note that the map extends in a simple way to sequential evaluation contexts L that are surface contexts. We write the latter one $[\![-]\!]^L : \mathcal{L}_I \to \mathcal{K}_\lambda^{DC}$.

$$\begin{aligned} [\![\texttt{let } x = e_1 \texttt{ in } e_2]\!]^S &= \texttt{let } x = [\![e_1]\!]^S \texttt{ in } [\![e_2]\!]^S \\ [\![\texttt{fun } f\ x \texttt{ is } e \texttt{ end}]\!]^S &= \texttt{fun } f\ x \texttt{ is } [\![e]\!]^S \texttt{ end} \\ [\![v_1\ v_2]\!]^S &= [\![v_1]\!]^S\ [\![v_2]\!]^S \end{aligned}$$

For the parallel pair, we have

```
1  [[(e1^(0) || e2^(0))^0]]^S =
2    capture (fun _ k is
3      let l1 = alloc
4          l2 = alloc
5          t1 = newTd (l1 := [[e1]]^S)
6          t2 = newTd (l2 := [[e2]]^S)
7          t  = newTd (k (!l1, !l2))
8      in newEdge (t1, t); newEdge (t2, t); transfer t;
9          release t; release t1; release t2; ())
```

Translation of well-formed expressions into λ^{DC} Now we define the translation for well-formed expressions and annotated expressions. For well-formed, annotated expressions we take an annotated expression and a set of its dependencies:

$$[\![e^{(1,l,t)}]\!](T) = [\![e]\!](l, t, T) \qquad [\![v^{(2,l)}]\!](T) = \emptyset, \emptyset, [l \mapsto v]$$

For well-formed expressions we have $[\![-]\!] : \mathcal{E}_I \to \mathcal{L} \times \mathrm{TId} \times \mathcal{P}_{\mathrm{fin}}(\mathrm{TId}) \to \mathcal{S}$, where $\mathcal{S}$ denotes the DAG states. We have:

$$[\![e]\!](l,t,T) =$$
$$\quad [t \mapsto (l := [\![e]\!]^S, \mathsf{R})], \{(t,t') \mid t' \in T\}, [l \mapsto ()] \quad \text{if } \mathrm{surf}(e)$$
$$[\![L[\,(e_1)^0 \parallel (e_2)^0\,)^1]\,]\!](l,t,T) =$$
$$\quad [t \mapsto (\mathtt{par1}([\![e_1]\!]^S, [\![e_2]\!]^S, l := [\![L]\!]^L), \mathsf{R})], \{(t,t') \mid t' \in T\}, [l \mapsto ()]$$
$$[\![L[\,(e_1)^0 \parallel (e_2)^0\,)^{(2,l_1)}]\,]\!](l,t,T) =$$
$$\quad [t \mapsto (\mathtt{par2}([\![e_1]\!]^S, [\![e_2]\!]^S, l := [\![L]\!]^L, l_1), \mathsf{R})], \{(t,t') \mid t' \in T\}, [l \mapsto (), l_1 \mapsto ()]$$
$$[\![L[\,(e_1)^0 \parallel (e_2)^0\,)^{(3,l_1,l_2)}]\,]\!](l,t,T) =$$
$$\quad [t \mapsto (\mathtt{par3}([\![e_1]\!]^S, [\![e_2]\!]^S, l := [\![L]\!]^L, l_1, l_2), \mathsf{R})], \{(t,t') \mid t' \in T\}, [l \mapsto (), l_1 \mapsto (), l_2 \mapsto ()]$$
$$[\![L[\,(e_1)^0 \parallel (e_2)^0\,)^{(4,l_1,l_2,t_1)}]\,]\!](l,t,T) =$$
$$\quad [t \mapsto (\mathtt{par4}([\![e_2]\!]^S, l := [\![L]\!]^L, l_1, l_2, t_1), \mathsf{R}), t_1 \mapsto (l_1 := [\![e_1]\!]^S, \mathsf{N})], \{(t,t') \mid t' \in T\}, [l \mapsto (), l_1 \mapsto (), l_2 \mapsto ()]$$
$$[\![L[\,(e_1)^0 \parallel (e_2)^0\,)^{(5,l_1,l_2,t_1,t_2)}]\,]\!](l,t,T) =$$
$$\quad [t \mapsto (\mathtt{par5}(l := [\![L]\!]^L, l_1, l_2, t_1, t_2), \mathsf{R}), t_1 \mapsto (l_1 := [\![e_1]\!]^S, \mathsf{N}), t_2 \mapsto (l_2 := [\![e_2]\!]^S, \mathsf{N})],$$
$$\quad \{(t,t') \mid t' \in T\}, [l \mapsto (), l_1 \mapsto (), l_2 \mapsto ()]$$
$$[\![L[\,(e_1)^0 \parallel (e_2)^0\,)^{(6,l_1,l_2,t_1,t_2,t')}]\,]\!](l,t) =$$
$$\quad [t \mapsto (\mathtt{par6}(t_1, t_2, t'), \mathsf{R}), t_1 \mapsto (l_1 := [\![e_1]\!]^S, \mathsf{N}), t_2 \mapsto (l_2 := [\![e_2]\!]^S, \mathsf{N}), t' \mapsto (\mathtt{join}(l, L, l_1, l_2), \mathsf{N})],$$
$$\quad \{(t,t'') \mid t'' \in T\}, [l \mapsto (), l_1 \mapsto (), l_2 \mapsto ()]$$
$$[\![L[\,(e_1)^0 \parallel (e_2)^0\,)^{(7,l_1,l_2,t_1,t_2,t')}]\,]\!](l,t) =$$
$$\quad [t \mapsto (\mathtt{par7}(t_1, t_2, t'), \mathsf{R}), t_1 \mapsto (l_1 := [\![e_1]\!]^S, \mathsf{N}), t_2 \mapsto (l_2 := [\![e_2]\!]^S, \mathsf{N}), t' \mapsto (\mathtt{join}(l, L, l_1, l_2), \mathsf{N})],$$
$$\quad \{(t,t'') \mid t'' \in T\} \cup \{(t_1,t')\}, [l \mapsto (), l_1 \mapsto (), l_2 \mapsto ()]$$
$$[\![L[\,(e_1)^0 \parallel (e_2)^0\,)^{(8,l_1,l_2,t_1,t_2,t')}]\,]\!](l,t) =$$
$$\quad [t \mapsto (\mathtt{par8}(t_1, t_2, t'), \mathsf{R}), t_1 \mapsto (l_1 := [\![e_1]\!]^S, \mathsf{N}), t_2 \mapsto (l_2 := [\![e_2]\!]^S, \mathsf{N}), t' \mapsto (\mathtt{join}(l, L, l_1, l_2), \mathsf{N})],$$
$$\quad \{(t,t'') \mid t'' \in T\} \cup \{(t_1,t'), (t_2,t')\}, [l \mapsto (), l_1 \mapsto (), l_2 \mapsto ()]$$
$$[\![L[\,(e_1)^0 \parallel (e_2)^0\,)^{(9,l_1,l_2,t_1,t_2,t')}]\,]\!](l,t) =$$
$$\quad [t \mapsto (\mathtt{par9}(t_1, t_2, t'), \mathsf{R}), t_1 \mapsto (l_1 := [\![e_1]\!]^S, \mathsf{N}), t_2 \mapsto (l_2 := [\![e_2]\!]^S, \mathsf{N}), t' \mapsto (\mathtt{join}(l, L, l_1, l_2), \mathsf{N})],$$
$$\quad \{(t',t'') \mid t'' \in T\} \cup \{(t_1,t'), (t_2,t')\}, [l \mapsto (), l_1 \mapsto (), l_2 \mapsto ()]$$
$$[\![L[\,(e_1)^0 \parallel (e_2)^0\,)^{(10,l_1,l_2,t_1,t_2,t')}]\,]\!](l,t) =$$
$$\quad [t \mapsto (\mathtt{par10}(t_1, t_2), \mathsf{R}), t_1 \mapsto (l_1 := [\![e_1]\!]^S, \mathsf{N}), t_2 \mapsto (l_2 := [\![e_2]\!]^S, \mathsf{N}), t' \mapsto (\mathtt{join}(l, L, l_1, l_2), \mathsf{R})],$$
$$\quad \{(t',t'') \mid t'' \in T\} \cup \{(t_1,t'), (t_2,t')\}, [l \mapsto (), l_1 \mapsto (), l_2 \mapsto ()]$$
$$[\![L[\,(e_1)^{a_1} \parallel (e_2)^0\,)^{(11,l_1,l_2,t_2,t')}]\,]\!](l,t) =$$
$$\quad [\![(e_1)^{a_1}]\!](l_1, t_1, \{t'\}) \uplus ([t \mapsto (\mathtt{par11}(t_2), \mathsf{R}), t_2 \mapsto (l_2 := [\![e_2]\!]^S, \mathsf{N}), t' \mapsto (\mathtt{join}(l, L, l_1, l_2), \mathsf{R})],$$
$$\quad \{(t',t'') \mid t'' \in T\} \cup \{(t_2,t')\}, [l \mapsto (), l_2 \mapsto ()])$$
$$[\![L[\,(e_1)^{a_1} \parallel (e_2)^{a_2}\,)^{(12,l_1,l_2,t')}]\,]\!](l,t) =$$
$$\quad [\![(e_1)^{a_1}]\!](l_1, t_1, \{t'\}) \uplus [\![(e_2)^{a_2}]\!](l_2, t_2, \{t'\}) \uplus ([t \mapsto ((), \mathsf{R}), t' \mapsto (\mathtt{join}(l, L, l_1, l_2), \mathsf{R})], \{(t',t'') \mid t'' \in T\}, [l \mapsto ()])$$
$$[\![L[\,(v_1)^{(2,l_1)} \parallel (v_2)^{(2,l_2)}\,)^{(13,l_1,l_2,t')}]\,]\!](l,t) =$$
$$\quad [t \mapsto ((), \mathsf{R}), t' \mapsto ((\mathtt{fun}\ f\ x\ \mathtt{is}\ L[x]\ \mathtt{end})\ ([\![v_1]\!]^S, !\, l_2), \mathsf{R})], \{(t',t'') \mid t'' \in T\}, [l \mapsto (), l_1 \mapsto [\![v_1]\!]^S, l_2 \mapsto [\![v_2]\!]^S]$$
$$[\![L[\,(v_1)^{(2,l_1)} \parallel (v_2)^{(2,l_2)}\,)^{(14,l_1,l_2,t')}]\,]\!](l,t) =$$
$$\quad [t \mapsto ((), \mathsf{R}), t' \mapsto ((\mathtt{fun}\ f\ x\ \mathtt{is}\ L[x]\ \mathtt{end})\ ([\![v_1]\!]^S, [\![v_2]\!]^S), \mathsf{R})], \{(t',t'') \mid t'' \in T\}, [l \mapsto (), l_1 \mapsto [\![v_1]\!]^S, l_2 \mapsto [\![v_2]\!]^S]$$

The macrodefinitions are partial evaluations of the translation of the parallel composition, as well as the code for the join task. These are as follows:

```
1  par1(e1, e2, K) =
2    let l1 = alloc
3        l2 = alloc
```

```
4       t1 = newTd (l1 := e1)
5       t2 = newTd (l2 := e2)
6       t  = newTd ((fun _ z => K[z]) (!l1, !l2))
7     in newEdge (t1, t); newEdge (t2, t); transfer t;
8       release t; release t1; release t2; ()
9
10  par2(e1, e2, K, l1) =
11    let l2 = alloc
12      t1 = newTd (l1 := e1)
13      t2 = newTd (l2 := e2)
14      t  = newTd ((fun _ z => K[z]) (!l1, !l2))
15    in newEdge (t1, t); newEdge (t2, t); transfer t;
16      release t; release t1; release t2; ()
17
18  par3(e1, e2, K, l1, l2) =
19    let t1 = newTd (l1 := e1)
20      t2 = newTd (l2 := e2)
21      t  = newTd ((fun _ z => K[z]) (!l1, !l2))
22    in newEdge (t1, t); newEdge (t2, t); transfer t;
23      release t; release t1; release t2; ()
24
25  par4(e2, K, l1, l2, t1) =
26    let t2 = newTd (l2 := e2)
27      t  = newTd ((fun _ z => K[z]) (!l1, !l2))
28    in newEdge (t1, t); newEdge (t2, t); transfer t;
29      release t; release t1; release t2; ()
30
31  par5(K, l1, l2, t1, t2) =
32    let t  = newTd ((fun _ z => K[z]) (!l1, !l2))
33    in newEdge (t1, t); newEdge (t2, t); transfer t;
34      release t; release t1; release t2; ()
35
36  par6(t1, t2, t) =
37    newEdge (t1, t); newEdge (t2, t); transfer t;
38    release t; release t1; release t2; ()
39
40  par7(t1, t2, t) =
41    newEdge (t2, t); transfer t;
42    release t; release t1; release t2; ()
43
44  par8(t1, t2, t) =
45    transfer t; release t; release t1; release t2; ()
46
47  par9(t1, t2, t) =
48    release t; release t1; release t2; ()
49
50  par10(t1, t2) =
51    release t1; release t2; ()
52
53  par11(t2) =
54    release t2; ()
55
56  join(l, K, l1, l2) =
57    (fun _ z => l := K[z]) (!l1, !l2)
```

Correctness With the translations defined, we can connect the annotated expressions with the DAG calculus states (through the definition of *matching* states $\propto$). Then,

Lemma 5 allows us to locate any executing thread as a subterm of our computation. This is of crucial importance, since this is what restricts the reduction steps that the DAG calculus can take in that thread: the code must come from the translation of that subcomputation. Using this property, we can prove Lemma 6, which states that every step on the DAG calculus side is either a scheduler step, or can be matched on the side of the instrumented expressions. Since we know the precise subexpression where the reduction happens, we can generally restrict ourselves to ensuring that the administrative reductions on the instrumented expression side match, step for step, the execution of the translation of the parallel pair — which is how the translation was designed in the first place. Finally, Lemma 7 gives us the backwards simulation of λ^{DC} with the instrumented semantics, and the Correctness Theorem composes it with the instrumentation lemma, Lemma 4. The proof process is much the same for the other translations.

Definition 4. We say that a well-formed annotated instrumented expression e^a such that $\mathrm{wf}(e^a)$ *matches* a λ^{DC} state V, E, σ, written $e^a \propto V, E, \sigma$ if there exists a DAG state V', E', σ' such that $\sigma' \sqsubseteq s$ and $[\![e^a]\!](\emptyset) = V', E, \sigma'$, and a finite map of vertices V'' such that $\forall t \in \mathrm{dom}(V'').\ V''(t) = (\mathsf{()}, \mathsf{R})$, and that

$$V' \uplus V'', E', \sigma \twoheadrightarrow_{\mathrm{s}}^{*} V, E, \sigma.$$

Lemma 5. *For any e^a, V, E, σ, t if $\mathrm{wf}(e^a)$, $e^a \propto V, E, \sigma$, $t \in \mathrm{dom}(V)$ and $\mathrm{status}(V(t)) = \mathsf{X}$, then either*

1. *$V(t) = (\mathsf{()}, \mathsf{X})$,*
2. *there exist K, e', l such that $e^a = K[e'^{(1,l,t)}]$,*
3. *there exist K, v_1, v_2, l_1, l_2, a', a'', L such that $e^a = K[(L[(\, v_1^{(2,l_1)} \parallel v_2^{(2,l_2)}\,)^{a'}])^{a''}]$ and $\mathrm{num}(a') >= 12$.*

Lemma 6. *For any e^a, V, V', E, E', σ, σ' if $\mathrm{wf}(e^a)$, $e^a \propto V, E, \sigma$ and $V, E, \sigma \twoheadrightarrow V', E', \sigma'$ then either $e^a \propto V', E', \sigma'$ or there exists $e'^{a'}$ such that $e^a \to e'^{a'}$ and $e'^{a'} \propto V', E', \sigma'$.*

Lemma 7. *For any e, a, V, V', E, σ, σ', if $\mathrm{wf}(e^a)$, $e^a \propto V, E, \sigma$ and $\forall t \in \mathrm{dom}(V').\ V'(t) = (\mathsf{()}, \mathsf{F})$, then the following simulation holds:*

$$\begin{array}{rcl} V, E, \sigma \twoheadrightarrow^{*} V', \emptyset, \sigma' & \Rightarrow & \exists v.\ e^a \to^{*} v^{(2,\mathrm{loc}(a))} \wedge \sigma'(l) = [\![v]\!]^S \\ V, E, \sigma \twoheadrightarrow^{\infty} & \Rightarrow & e^a \to^{\infty} \end{array}$$

Proof. Follows from Lemmas 6 and 1 by the same inductive and coinductive argument as used in the proof of Lemma 4. □

Theorem 1 (Correctness)**.** *Let t be the identifier of the main thread, $e \in \mathcal{E}_O$ be the source program stored in this thread, and l be a designated location in which to store the final result. Let $e_I \in \mathcal{E}_I$ be e annotated with the 0 annotations on all parallel compositions. For any integer result n, final state σ such that $\sigma(l) = n$, and final set of vertices V, assuming that all threads t' in V are finished (i.e. $\mathrm{status}\,(V(t')) = \mathsf{F}$), we have:*

$$[t \mapsto (l := [\![e_I]\!]^S, \mathsf{R})], \emptyset, [l \mapsto ()] \twoheadrightarrow^{*} V, \emptyset, \sigma \quad \Rightarrow \quad e \to_O^{*} n.$$

Furthermore, divergence in the DAG calculus entails divergence in the source language:

$$[t \mapsto (l := [\![e_I]\!]^S, \mathsf{R})], \emptyset, [l \mapsto ()] \twoheadrightarrow^{\infty} \quad \Rightarrow \quad e \to_O^{\infty} .$$

Proof. This theorem follows from the composition of the two simulation diagrams, given by Lemmas 4 and 7. Clearly, $\lfloor e_I \rfloor = e$. Moreover, since $\mathrm{surf}(e_I)$, we also have $[\![e_I^{(1,l,t)}]\!](\emptyset) = [t \mapsto (l := [\![e_I]\!]^S, \mathsf{R})], \emptyset, [l \mapsto ()]$, which gives us $e_I^{(1,l,t)} \propto [t \mapsto (l := [\![e_I]\!]^S, \mathsf{R})], \emptyset, [l \mapsto ()]$.

For the termination case, by Lemma 7 we obtain that there exists a value v such that $e_I^{(1,l,t)} \to_I^* v^{(2,l)}$ and $\sigma'(l) = [\![v]\!]^S$. However, since $\sigma'(l) = n$, we can conclude that $v = n$. By Lemma 4 we can now conclude that $e \to_O^* \lfloor n \rfloor$, which ends the proof, as $\lfloor n \rfloor = n$.

For the nontermination case, from Lemma 7 conclude that $e_I^{(1,l,t)} \to_I^{\infty}$, which, by Lemma 4 implies that $e \to_O^{\infty}$. This ends the proof. □

C Futures

C.1 Syntax and Operational Semantics

The syntax and semantics in this section are the same as in the paper and repeated here for convenience.

$$\begin{aligned} e &::= x \mid n \mid f \mid (v, v) \mid \texttt{fst}\ v \mid \texttt{snd}\ v \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid v\ v \\ &\quad \mid \texttt{fun}\ g\ x\ \texttt{is}\ e\ \texttt{end} \mid \texttt{future}(e) \mid \texttt{force}(v) \\ v &::= x \mid n \mid f \mid (v, v) \mid \texttt{fun}\ g\ x\ \texttt{is}\ e\ \texttt{end} \\ K &::= \bullet \mid \texttt{let}\ x = K\ \texttt{in}\ e \end{aligned}$$

We take the set of configurations $\mathcal{M} \equiv \mathcal{F} \rightharpoonup_{fin} \mathcal{E}$, where $\mathcal{F}$ is the set of future identities and $\mathcal{E}$ — the set of expressions, and write $M \in \mathcal{M}$ for these configurations. We also pick an identifier of the main program, f_0: thus, the initial state is $[f_0 \mapsto e]$ for some program e. The evaluation reaches a terminal state when all the futures in M have values associated with them.

$$\texttt{fst}\ (v_1, v_2) \circ\!\!\to v_1 \qquad \texttt{snd}\ (v_1, v_2) \circ\!\!\to v_2 \qquad \texttt{let}\ x = v\ \texttt{in}\ e \circ\!\!\to e[x \mapsto v]$$
$$\texttt{fun}\ g\ x\ \texttt{is}\ e\ \texttt{end}\ v \circ\!\!\to e[x \mapsto v, g \mapsto \texttt{fun}\ g\ x\ \texttt{is}\ e\ \texttt{end}]$$

$$\frac{M(f) = K[e] \qquad e \circ\!\!\to e'}{M \to M[t \mapsto K[e']]}$$
$$\frac{M(f) = K[\texttt{future}(e)] \qquad f'\ \text{fresh}}{M \to M[f \mapsto K[f'], f' \mapsto e]}$$
$$\frac{M(f) = K[\texttt{force}(f')] \qquad M(f') = v}{M \to M[f \mapsto K[v]]}$$

C.2 Instrumented Syntax and Operational Semantics

The instrumented language is presented below. The input terms can only be annotated with 0. When there is a chance of confusion between the instrumented and original language, we annotate the constructs of the instrumented language with a subscript I, and the constructs of original language with O. The instrumentations are a_s for future creation, a_f for future access, and a_w for future evaluation.

$$
\begin{aligned}
e ::= & x \mid n \mid t \mid (v, v) \mid \texttt{fst}\ v \mid \texttt{snd}\ v \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid v\ v \\
& \mid \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end} \mid \texttt{future}(e)^{a_s} \mid \texttt{force}(v)^{a_f} \\
v ::= & x \mid n \mid t \mid (v, v) \mid \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end} \\
a_s ::= & 0 \mid (1, l) \mid (2, l, t) \mid (n, l, t, t) \text{where } 3 \leq n \leq 5 \\
a_f ::= & n \text{where } 0 \leq n \leq 2 \mid (n, t) \text{where } 3 \leq n \leq 6 \\
a_w ::= & 0 \mid 1 \\
K ::= & \bullet \mid \texttt{let}\ x = K\ \texttt{in}\ e
\end{aligned}
$$

The annotations a_s and a_f can *progress*, written $a \rightsquigarrow_s a'$ (and, respectively $a \rightsquigarrow_f a'$), if the step of a' is one greater than the step of a. All the other parts of a' present in a have to agree, while those absent in a are globally fresh, as follows (for a_f):

$$
\begin{gathered}
0 \rightsquigarrow_f 1 \qquad 1 \rightsquigarrow_f 2 \qquad 2 \rightsquigarrow_f (3, t) \text{ where } t \text{ is fresh} \\
(n, t) \rightsquigarrow_f (n + 1, t) \text{ for } 3 \leq n \leq 5
\end{gathered}
$$

The definition for a_s is analogous:

$$
\begin{gathered}
0 \rightsquigarrow_s (1, l) \text{ where } l \text{ is fresh} \qquad (1, l) \rightsquigarrow_s (2, l, t) \text{ where } t \text{ is fresh} \\
(2, l, t) \rightsquigarrow_s (3, l, t, t') \text{ where } t' \text{ is fresh} \qquad (n, l, t, t') \rightsquigarrow_s (n + 1, l, t, t') \text{ for } 3 \leq n \leq 4
\end{gathered}
$$

Sequential evaluation within a future proceeds as follows:

$$
\begin{gathered}
\texttt{fst}\ (v_1, v_2) \bigcirc\!\!\!\rightarrow v_1 \qquad \texttt{snd}\ (v_1, v_2) \bigcirc\!\!\!\rightarrow v_2 \qquad \texttt{let}\ x = v\ \texttt{in}\ e \bigcirc\!\!\!\rightarrow e[x \mapsto v] \\
\texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end}\ v \bigcirc\!\!\!\rightarrow e[x \mapsto v, f \mapsto \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end}] \\
\frac{a \rightsquigarrow_f a'}{\texttt{future}(e)^{a} \bigcirc\!\!\!\rightarrow \texttt{future}(e)^{a'}} \qquad \frac{a \rightsquigarrow_s a'}{\texttt{force}(e)^{a} \bigcirc\!\!\!\rightarrow \texttt{force}(e)^{a'}}
\end{gathered}
$$

The configurations are defined as $\mathcal{M} = \mathcal{F} \rightharpoonup_{\mathit{fin}} (\mathcal{L} \times \mathrm{TId} \times \mathrm{TId} \times \mathcal{E} \times a_w)$, and well-formed configurations have all the locations and thread id's disjoint. The special starting future, f_0 has a special first thread id, t_0. The evaluation then proceeds as

follows.

$$\frac{M(f) = (l, t_1, t_2, K[e], 0) \qquad K[e] \bigcirc\!\!\!\rightarrow K[e']}{M \to M[t \mapsto (l, t_1, t_2, K[e'], 0)]}$$

$$\frac{M(f) = (l, t_1, t_2, K[\texttt{future}(e)^{(5,l',t_1',t_2')}], 0) \qquad f' \text{ fresh}}{M \to M[f \mapsto (l, t_1, t_2, K[f'], 0), f' \mapsto (l', t_1', t_2', e, 0)]}$$

$$\frac{M(f) = (l, t_1, t_2, K[\texttt{force}(f')^{(6,t_3)}], 0) \qquad M(f') = (l', t_1', t_2', v, 1)}{M \to M[f \mapsto (l, t_1, t_3, K[v], 0)]}$$

$$\frac{M(f) = (l, t_1, t_2, v, 0)}{M \to M[f \mapsto (l, t_1, t_2, v, 1)]}$$

An expression is a *surface* expression, written $\text{surf}(e)$ if all its annotations are 0. Note that these are isomorphic to the terms of the original language. An term is *well-formed*, written $\text{wf}(e)$, if it has at most one non-zero annotation, in an evaluation position, i.e.,

$$\text{wf}(e) = \begin{cases} \text{wf}(e_1) \wedge \text{surf}(e_2) & \text{if } e = \texttt{let } x = e_1 \texttt{ in } e_2 \\ \text{surf}(e') & \text{if } e = \texttt{future}(e')^a \\ \text{surf}(v) & \text{if } e = \texttt{force}(v)^a \\ \text{surf}(e) & \text{otherwise} \end{cases}$$

We write $\text{wf}(M)$ to mean that each of the expressions associated with the futures is well-formed.

Lemma 8. *Evaluation preserves well-formedness, i.e., if* $\text{wf}(e)$ *and* $e \bigcirc\!\!\!\rightarrow e'$ *then* $\text{wf}(e')$. *Similarly, if* $\text{wf}(M)$ *and* $M \to M'$, *then* $\text{wf}(M')$.

Proof. The first part proceeds by simply checking the reductions. The second follows by simple induction on the structure of the expression associated with the future that reduces. □

C.3 Connecting the instrumented and source languages

We define a map $\lfloor - \rfloor : \mathcal{E}_I \to \mathcal{E}_O$ that removes annotations from the instrumented expressions. This map extends to the configurations, by also removing the additional information in the futures. We use it to connect the instrumented language to the source language. Note that a map that adds an annotation 0 to `future` and `force` is a right inverse of $\lfloor - \rfloor$. For *surface* term, it is also the left inverse.

Lemma 9 (Instr-Step)**.** *For any configurations* $M, M' \in \mathcal{M}_I$, *if* $M \to_I M'$, *then either* $\lfloor M \rfloor = \lfloor M' \rfloor$ *or* $\lfloor M \rfloor \to_O \lfloor M' \rfloor$.

Proof. By cases on the reduction. In cases of spawning a future, forcing a future of finishing a future, trivially true, since $\lfloor K[e] \rfloor = \lfloor K \rfloor [\lfloor e \rfloor]$. For the same reason, suffices to check the same statement for any $e \bigcirc\!\!\!\rightarrow e'$.

These split into two forms: the administrative reductions of `future` and `force`, which give the same erasure for both the redex and the reduct, and the reductions of sequential constructs, which can be trivially matched. □

Definition 5. An *administrative* reduction in the instrumented language is a reduction $M \to M'$ such that $\lfloor M \rfloor = \lfloor M' \rfloor$. We write $M \to_a M'$ for administrative steps.

Lemma 10 (Admin-Fin)**.** *For well-formed configurations M, there are no infinite sequence of administrative reductions, i.e., for any $M \in \mathcal{M}_I$, if* wf M *then* $M \not\to_a^\infty$.

Proof. Assign as a weight to each future based on the count of their annotations in evaluation positions. Take the $\mathrm{cnt}(a)$ to be the natural number in the annotation. Then define

$$w(e) = \begin{cases} 5 - \mathrm{cnt}(a) & \text{if } e = \texttt{future}(e')^a \\ 6 - \mathrm{cnt}(a) & \text{if } e = \texttt{force}(e')^a \\ w(e_1) & \text{if } e = \texttt{let } x = e_1 \texttt{ in } e_2 \\ 0 & \text{otherwise} \end{cases},$$

$w(l, t_1, t_2, e, n) = w(e) + 1 - n$, and $w(M) = \sum_{f \in \mathrm{dom}(M)} w(M(f))$. From there, the proof is analogous to the proof of Lemma 1: by simple induction, each administrative reduction decreases the weight, and since the weight of a given configuration is finite, there is no infinite chain of reductions administrative. □

Lemma 11 (Instrument)**.** *For any configurations M, M' such that M' is final and $M \to_I^* M'$, $\lfloor M \rfloor \to_O^* \lfloor M' \rfloor$ and $\lfloor M' \rfloor$ is final. Moreover, if $M \to_I^\infty$, then $\lfloor M \rfloor \to_O^\infty$.*

Proof. First part, by induction on the reduction sequence and Lemma 9. Since final configurations of the instrumented language map to final configurations of the source language, this ends the proof.

Similarly, for the second part we proceed by coinduction: since $M \to_I^\infty$, by Lemmas 10 and 9 there exists an M' such that $M \to_I^* M'$, $\lfloor M \rfloor \to_O \lfloor M' \rfloor$ and $M' \to_I^\infty$. By coinduction, we obtain that $\lfloor M' \rfloor \to_O^\infty$, and so $\lfloor M \rfloor \to_O^\infty$. □

C.4 Connecting the instrumented language and λ^{DC}

Translation of surface expressions into λ^{DC} Note that the translation trivially extends to the evaluation contexts of the instrumented language. We define a translation $[\![-]\!]^S : \mathcal{E}_I \to \mathcal{M}_I \to \mathcal{E}_{\lambda^{DC}}$.

$$[\![\texttt{let } x = e_1 \texttt{ in } e_2]\!]^S(M) = \texttt{let } x = [\![e_1]\!]^S(M) \texttt{ in } [\![e_2]\!]^S(M)$$
$$[\![\texttt{fun } f\ x \texttt{ is } e \texttt{ end}]\!]^S(M) = \texttt{fun } f\ x \texttt{ is } [\![e]\!]^S(M) \texttt{ end}$$
$$[\![v_1\ v_2]\!]^S(M) = [\![v_1]\!]^S(M)\ [\![v_2]\!]^S(M)$$

The translations for the pairs, variables and numbers are analgous. For future id's we use the state M to map them to the thread-location pairs:

$$[\![f]\!]^S(M) = (t, l) \quad \text{where M(f) = (l, t, \_, \_, \_).}$$

For the other constructs, we have:

```
1  ⟦future(e)^0⟧^S =
2    let l = alloc
3        t = newTd (())
4        tb = newTd (l := ⟦e⟧^S)
5    in newEdge (tb, t); release t; release tb; (t, l)
```

and

```
⟦force(v)^0⟧^S =
  let (ft, fl) = ⟦v⟧^S
  in capture (fun _ k is
    let ct = newTd (k (!fl))
    in newEdge (ft, ct); transfer ct; release ct; ())
```

As mentioned, since all expressions in the well-formed evaluation contexts are surface expressions, we can extend this translation to contexts, giving us $[\![-]\!]^K : \mathcal{K}_I \to \mathcal{K}_{\lambda^{DC}}$.

Translation of well-formed futures into λ^{DC} Here, we define a translation $[\![-]\!] : \mathcal{E}_I \to \mathcal{L} \times \mathrm{TId}^2 \times \mathcal{M}_I \to \mathcal{S}$, where $\mathcal{S}$ are the states of λ^{DC}. In case the expression is a surface expression, we simply use the surface translation and add the DAG structure:

$$[\![e]\!](l, t_f, t_b, M) = [t_b \mapsto (l := [\![e]\!]^S, \mathsf{R})], \{(tb, tf)\}, \emptyset.$$

This leaves us with non-surface cases: the creation and reading of futures. The translation of these constructs takes into account the partial evaluation of the code that their annotations denote. For the `future` construct, we have as follows:

$$\begin{aligned}
&[\![K[\texttt{future}(e)^{(1,l)}]]\!](l_f, t_f, t_b, M) = \\
&\quad [t_b \mapsto (l_f := [\![K]\!]^K[\texttt{ft1}(e, l)], \mathsf{R})], \{(t_b, t_f)\}, [l \mapsto ()] \\
&[\![K[\texttt{future}(e)^{(2,l,t)}]]\!](l_f, t_f, t_b, M) = \\
&\quad [t_b \mapsto (l_f := [\![K]\!]^K[\texttt{ft2}(e, l, t)], \mathsf{R}); t \mapsto ((), \mathsf{N})], \{(t_b, t_f)\}, [l \mapsto ()] \\
&[\![K[\texttt{future}(e)^{(3,l,t,t'_b)}]]\!](l_f, t_f, t_b, M) = \\
&\quad [t_b \mapsto (l_f := [\![K]\!]^K[\texttt{ft3}(l, t, t'_b)], \mathsf{R}); t \mapsto ((), \mathsf{N}); t_b \mapsto (l := [\![e]\!]^S, \mathsf{N})], \\
&\quad \{(t_b, t_f)\}, [l \mapsto ()] \\
&[\![K[\texttt{future}(e)^{(4,l,t,t'_b)}]]\!](l_f, t_f, t_b, M) = \\
&\quad [t_b \mapsto (l_f := [\![K]\!]^K[\texttt{ft4}(l, t, t'_b)], \mathsf{R}); t \mapsto ((), \mathsf{N}); t_b \mapsto (l := [\![e]\!]^S, \mathsf{N})], \\
&\quad \{(t_b, t_f), (t'_b, t)\}, [l \mapsto ()] \\
&[\![K[\texttt{future}(e)^{(5,l,t,t'_b)}]]\!](l_f, t_f, t_b, M) = \\
&\quad [t_b \mapsto (l_f := [\![K]\!]^K[\texttt{ft5}(l, t, t'_b)], \mathsf{R}); t \mapsto ((), \mathsf{R}); t_b \mapsto (l := [\![e]\!]^S, \mathsf{N})], \\
&\quad \{(t_b, t_f), (t'_b, t)\}, [l \mapsto ()]
\end{aligned}$$

In the above, the helper macros are defined as partial evaluation states of the surface translation of `future`:

```
ft1(e, l) =
  let t = newTd (())
      tb = newTd (l := ⟦e⟧^S)
  in newEdge (tb, t); release t; release tb; (t, l)

ft2(e, l, t) =
  let tb = newTd (l := ⟦e⟧^S)
  in newEdge (tb, t); release t; release tb; (t, l)
```

```
9
10 ft3(l, t, tb) =
11   newEdge(tb, t); release t; release tb; (t, l)
12
13 ft4(l, t, tb) =
14   release t; release tb; (t, l)
15
16 ft5(l, t, tb) =
17   release tb; (t, l)
```

We tackle the `force` operation in a similar manner:

$$[\![K[\texttt{force}(v)^1]]\!](l_f, t_f, t_b, M) =$$
$$[t_b \mapsto (l_f, [\![K]\!]^K[\texttt{fr1}(t, l)], \mathsf{R})], \{(t_b, t_f)\}, \emptyset$$
$$[\![K[\texttt{force}(v)^2]]\!](l_f, t_f, t_b, M) =$$
$$[t_b \mapsto (\texttt{fr2}(t, l, l_f := [\![K]\!]^K), \mathsf{R})], \{(t_b, t_f)\}, \emptyset$$
$$[\![K[\texttt{force}(v)^{(3,t')}]]\!](l_f, t_f, t_b, M) =$$
$$[t_b \mapsto (\texttt{fr3}(t, t'), \mathsf{R}); t' \mapsto (l_f := [\![K]\!]^K[!\, l], \mathsf{N})], \{(t_b, t_f)\}, \emptyset$$
$$[\![K[\texttt{force}(v)^{(4,t')}]]\!](l_f, t_f, t_b, M) =$$
$$[t_b \mapsto (\texttt{fr4}(t'), \mathsf{R}); t' \mapsto (l_f := [\![K]\!]^K[!\, l], \mathsf{N})], \{(t_b, t_f), (t, t')\}, \emptyset$$
$$[\![K[\texttt{force}(v)^{(5,t')}]]\!](l_f, t_f, t_b, M) =$$
$$[t_b \mapsto (\texttt{fr5}(t'), \mathsf{R}); t' \mapsto (l_f := [\![K]\!]^K[!\, l], \mathsf{N})], \{(t', t_f), (t, t')\}, \emptyset$$
$$[\![K[\texttt{force}(v)^{(6,t')}]]\!](l_f, t_f, t_b, M) =$$
$$[t_b \mapsto ((), \mathsf{R}); t' \mapsto (l_f := [\![K]\!]^K[!\, l], \mathsf{R})], \{(t', t_f), (t, t')\}, \emptyset$$

where throughout the definition $(t, l) = [\![v]\!]^S(M)$. Similar to before, we use helper macros that define partial evaluation of the surface translation of `force`:

```
1  fr1(ft, fl) =
2    in capture (fun _ k is
3      let ct = newTd (k (!fl))
4      in newEdge (ft, ct); transfer ct; release ct; ())
5
6  fr2(ft, fl, K) =
7    let ct = newTd (K[!fl])
8    in newEdge (ft, ct); transfer ct; release ct; ()
9
10 fr3(ft, ct) =
11   newEdge (ft, ct); transfer ct; release ct; ()
12
13 fr4(ct) =
14   transfer ct; release ct; ()
15
16 fr5(ct) = release ct
```

Finally we define the translation for a future — and then as a translation of the complete state we take a disjoint union of the translation of the individual futures. We have:

$$[\![(l, t_f, t_b, e, 0)]\!](M) = \begin{cases} [t_f \mapsto ((), \mathsf{R})], \emptyset, [l \mapsto ()] \uplus [\![e]\!](l, t_f, t_b, M) & \text{if } t_f \neq t_0 \\ \emptyset, \emptyset, [l \mapsto ()] \uplus [\![e]\!](l, t_f, t_b, M) & \text{otherwise} \end{cases}$$

$$[\![(l, t_f, t_b, v, 1)]\!](M) = \begin{cases} [t_f \mapsto ((), \mathsf{R})], \emptyset, [l \mapsto [\![v]\!]^S] & \text{if } t_f \neq t_0 \\ \emptyset, \emptyset, [l \mapsto [\![v]\!]^S] & \text{otherwise} \end{cases}$$

Definition 6. We say that an instrumented configuration M *matches* a DAG Calculus state V, E, σ, written $M \propto V, E, \sigma$ if there exists a DAG state V', E' such that $[\![M]\!] = V', E', \sigma$ and a finite map of threads V'' such that $\forall t \in \mathrm{dom}(V'')$. $V''(t) = ((), \mathsf{R})$ and that

$$V' \uplus V'', E', \sigma \twoheadrightarrow_{\mathrm{s}}^{*} V, E, \sigma.$$

Lemma 12. *For any M, V, E, σ, t if* $\mathrm{wf}(M)$, $M \propto V, E, \sigma$, $t \in \mathrm{dom}(V)$ *and* $\mathrm{status}(V(t)) = \mathsf{X}$, *then either $V(t) = ((), \mathsf{X})$ or there exist f, t_1, t_2, l and e such that $M(f) = (l, t_1, t_2, e, 0)$ and either $t_2 = t$, or $e = K[\texttt{force}(v)^{6,t}]$*

Lemma 13. *For any M, V, V', E, E', σ, σ' if* $\mathrm{wf}(M)$, $M \propto V, E, \sigma$ *and $V, E, \sigma \twoheadrightarrow V', E', \sigma'$ then either $M \propto V', E', \sigma'$ or there exists an M' such that $M \to M'$ and $M' \propto V', E', \sigma'$.*

Proof. Consider the reduction $V, E, \sigma \twoheadrightarrow V', E', \sigma'$. It is either a Start reduction, in which case it is an administrative reduction, and $M \propto V', E', \sigma'$, or it happens in some thread $t \in \mathrm{dom}(V)$ such that $\mathrm{status}(V(t)) = \mathsf{X}$. Thus, by Lemma 12 we either have $V(t) = ((), \mathsf{X})$ or there exists a future f such that $M(f) = (l, t_1, t_2, e, 0)$ and either $t_2 = t$ or $e = K[\texttt{force}(v)^{6,t}]$. In the first case, the only applicable reduction rule is Stop, which is an administrative reduction. This leaves us with the two subcases of the second case. Start with the second.

Let $V_M, E_M, \sigma_M = [\![M]\!]$ and V'' — the additional nodes witnessed by $\propto$. By the definition of the translation we know that $M(v) = (l_v, t_v, tb_v, e_v, n_v)$ and that $V_M(t) = (l := [\![K]\!]^K[!\, l_v], \mathsf{R})$ and $(t_v, t) \in E_M$. Since $\mathrm{status}(V(t)) = \mathsf{X}$, the reduction $V_M, E_M, \sigma_M \twoheadrightarrow V, E, \sigma$ had to include a Start call on t. However, for this rule to apply there can be no incoming edges on t. Thus, there also had to be a Stop call on t_v. Inspecting the translation, we find that this is possible to achieve through administrative reductions only if $n_v = 1$ — and consequently that e_v is a value, v. However, this means in M we can apply the rule for `force` in future f getting $M' = M[f \mapsto (l, t_1, t, K[v], 0)]$. This leaves us with showing that $M' \propto V', E', \sigma'$. Take $V'_M, E'_M, \sigma'_M = [\![M']\!]$. Since $t_2 \notin \mathrm{dom}(V'_M)$, we take as the witness for $\propto$ the map $V''[t_2 \mapsto ((), \mathsf{R})]$. For all the threads other than t, the same administrative reductions applied in the assumption are available. For t we need to observe that the rule applied in the reduction $V, E, \sigma \twoheadrightarrow V', E', \sigma'$ has is Step with a dereference reduction. Thus, we obtain that $V'(t) = (l := [\![K]\!]^K[[\![v]\!]^S], \mathsf{X})$, which can be obtained from $V'_M(t)$ by the administrative Start step.

Let us now turn to the remaining case. We have $M(f) = (l, t_1, t, e, 0)$ and a reduction in thread t. We can decompose e uniquely into K and e' such that $e = K[e']$ and e' is not a let-binding. If e' is neither a value nor a parallel construct, then it's at $[\![e']\!]^S(M)$ that the reduction occurs on the DAG calculus side. Since the translation is entirely structural, we can match it by using the corresponding rule. The case if e' is a value is similar: the redex is the innermost let-binding in K, and can be matched by the let-rule. Finally, for parallel primitives, the location of the redex is based on the annotation: however, the translations of the sequences of annotated primitives are specifically designed so that they correspond to one step of reduction on the λ^{DC} side. Thus, we can localize the

redex, obtain any new threads or locations created on the DAG calculus side, and use them in the matching reduction. □

Lemma 14. *For any M, V, V', E, σ, σ' if* $\mathrm{wf}(M)$, $M \propto V, E, \sigma$ *and* $\forall t \in \mathrm{dom}(V').\ V'(t) = (\langle\rangle, \mathsf{F})$, *then the following simulation holds:*

$$V, E, \sigma \twoheadrightarrow^* V', \emptyset, \sigma' \quad \Rightarrow \quad \exists M'.\ M \to_I^* M' \wedge M' \propto V', \emptyset, \sigma'$$
$$V, E, \sigma \twoheadrightarrow^\infty \quad \Rightarrow \quad M \to_I^\infty$$

Proof. Follows from Lemmas C.4 and 1 by the same inductive and coinductive argument as used in the proof of Lemma 11. □

Theorem 2 (Correctness)**.** *Let t be the identifier of the main thread, e be the source program stored in this thread, and l be a designated location in which e stores its final result. Let e_I to be the result of annotating all instances of* `future` *and* `force` *in e with* 0. *For any number n, final state σ such that $\sigma(l) = n$, and final set of vertices V, assuming that all threads t' in V are finished (i.e.* $\mathrm{status}\,(V(t')) = \mathsf{F}$*), if*

$$[t \mapsto (l := [\![e_I]\!]^S(\emptyset), \mathsf{R})], \emptyset, [l \mapsto ()] \twoheadrightarrow^* V, \emptyset, \sigma$$

then there is a final configuration M such that $[f_0 \mapsto e] \to^* M$ and $M(f_0) = n$. Furthermore, divergence is preserved:

$$[t \mapsto (l := [\![e_I]\!]^S(\emptyset), \mathsf{R})], \emptyset, [l \mapsto ()] \twoheadrightarrow^\infty \quad \Rightarrow \quad [f_0 \mapsto e] \to^\infty .$$

Proof. This theorem follows from the composition of the two simulation diagrams, given by Lemmas 11 and 14. Clearly, $\lfloor e_I \rfloor = e$. Moreover, since $\mathrm{surf}(e_I)$, we also have $[\![[f_0 \mapsto (l, t_0, t, e_I, 0)]]\!](\emptyset) = [t \mapsto (l := [\![e_I]\!]^S(\emptyset), \mathsf{R})], \emptyset, [l \mapsto ()]$, which gives us $[f_0 \mapsto (l, t_0, t, e_I, 0)] \propto [t \mapsto (l := [\![e_I]\!]^S(\emptyset), \mathsf{R})], \emptyset, [l \mapsto ()]$.

For the termination case, by Lemma 14 we obtain that there exists a configuration M such that $[f_0 \mapsto (l, t_0, t, e_I, 0)] \to_I^* M$ and $M \propto V', \emptyset, \sigma'$. Inspecting the translation, we find that this requires that the annotations of all the futures in M are equal to 1, and so M is a final configuration. Since this means that $M(f_0) = (l, t_0, t', v, 1)$ for some t' and v, we learn that $\sigma'(l) = [\![v]\!]^S(M)$, and so, sine $\sigma'(l) = n$, that $v = n$. By Lemma 11 we can now conclude that $[f_0 \mapsto e] \to_O^* \lfloor M \rfloor$, which ends the proof, since $\lfloor M \rfloor(f_0) = n$.

For the nontermination case, from Lemma 14 conclude that $[f_0 \mapsto (l, t_0, t, e_I, 0)] \to_I^\infty$, which, by Lemma 4 implies that $[f_0 \mapsto e] \to_O^\infty$. This ends the proof. □

D Async-Finish

D.1 Syntax and Operational Semantics

$$\begin{aligned}
e &::= v \mid \texttt{fst}\ v \mid \texttt{snd}\ v \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid v\ v \\
&\quad \mid \texttt{alloc} \mid v := v \mid !\,v \mid \texttt{async}(e) \mid \texttt{finish}(S) \\
v &::= x \mid n \mid l \mid (v, v) \mid \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end} \\
S &\equiv \{e_1, e_2, \ldots, e_n\} \\
K &::= L \mid L[\texttt{finish}(\{K\} \uplus S)] \\
L &::= \bullet \mid \texttt{let}\ x = L\ \texttt{in}\ e
\end{aligned}$$

The input programs are restricted to use the form $\texttt{finish}(\{e\})$. The operational semantics is given below. It operates on the store $\mathcal{H} = \mathcal{L} \rightharpoonup_{fin} \mathcal{V}$

$$\frac{}{h, \texttt{fst}\ (v_1, v_2) \to v_1, h} \qquad \frac{}{h, \texttt{snd}\ (v_1, v_2)] \to v_2, h} \qquad \frac{}{h, \texttt{let}\ x = v\ \texttt{in}\ e \to e[x \mapsto v], h}$$

$$\frac{}{h, \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end}\ v \to e[x \mapsto v, f \mapsto \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end}], h}$$

$$\frac{l \notin \mathrm{dom}(h)}{h, \texttt{alloc} \to l, h[l \mapsto ()]} \qquad \frac{h(l) = v}{h, !\, l \to v, h} \qquad \frac{l \in \mathrm{dom}(h)}{h, l := v \to (), h[l \mapsto v]}$$

$$\frac{}{h, \texttt{finish}(\{L[\texttt{async}(e)]\} \uplus S) \to \texttt{finish}(\{L[()], e\} \uplus S), h}$$

$$\frac{\forall e \in S.\ e = ()}{h, \texttt{finish}(S) \to (), h} \qquad \frac{h, e \to e', h'}{h, K[e] \to K[e'], h'}$$

Note the operational semantics is changed slightly wrt. the paper: the finished async threads are not removed one by one, but rather stay as values associated with the enclosing finish block, which finishes when all the async computations terminate. This is done to facilitate the proof, since the removal of a single finished async would correspond to a scheduler transition (Stop), and it is easier to keep the scheduler transitions separate from the others. It is a simple exercise to show this presentation equivalent to the one presented in the paper: since the removals of finished async threads are nondeterministic, they can all happen right before the finish block should reduce. The other direction is even simpler.

The following lemma allows us to only consider programs that write their final value to a predetermined location in the state in the following, since these can simulate the other programs.

Lemma 15. *For any program e and location l and result state h, we have the following properties:*

$$[l \mapsto ()], \texttt{let}\ x = e\ \texttt{in}\ l := x \to^* (), h \quad \Rightarrow \quad \emptyset, e \to^* h(l), h \setminus [l \mapsto h(l)],$$

$$[l \mapsto ()], \texttt{let}\ x = e\ \texttt{in}\ l := x \to^\infty \quad \Rightarrow \quad \emptyset, e \to^\infty .$$

Proof. Simple induction. □

D.2 Instrumented Syntax and Operational Semantics

When there is a chance of confusion between the instrumented and original language, we annotate the constructs of the instrumented language with a subscript I, and the constructs of original language with O. The instrumentations are a_a for async and a_f for finish. Additionally, we annotate roots of expressions with thread identifiers.

$$
\begin{aligned}
e &::= v \mid \texttt{fst}\ v \mid \texttt{snd}\ v \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid v\ v \mid \texttt{alloc} \mid v := v \mid !\, v \\
&\quad \mid \texttt{async}(e)^{a_a} \mid \texttt{finish}(e)^{a_f} \mid \texttt{finish}(S) \\
v &::= x \mid n \mid l \mid (v, v) \mid \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end} \\
S &\equiv \{e_1^{t_1}, e_2^{t_2}, \ldots, e_n^{t_n}\} \\
L &::= \bullet \mid \texttt{let}\ x = L\ \texttt{in}\ e \\
K &::= \bullet \mid (L[\texttt{finish}(\{K\} \uplus S)])^t \\
a_a &::= 0 \mid (1, t) \mid (2, t) \\
a_f &::= 0 \mid 1 \mid (2, t) \mid (n, t, t) \text{ where } 3 \le n \le 6
\end{aligned}
$$

We define the surface expressions, $\text{surf}(e)$ as ones that only contain annotations of 0 and do not contain subexpressions of the form $\texttt{finish}(S)$. This notion extends in a trivial way to the sequential contexts L. With this, we can proceed to define the well-formed expressions, $\text{wf}(e)$:

$$
\text{wf}(e) = \begin{cases}
\text{surf}(e') \wedge \text{surf}\, L & \text{if } e = L[\texttt{async}(e')^a] \\
\text{surf}(e') \wedge \text{surf}\, L & \text{if } e = L[\texttt{finish}(e')^a] \\
(\forall e', t.\ (e'^t) \in S \Rightarrow \text{wf}(e')) \wedge \text{surf}\, L & \text{if } e = L[\texttt{finish}(S)] \\
\text{surf}\, e & \text{otherwise}
\end{cases}
$$

This notion simply extends to contexts K.

For operational semantics, let us first define the allowed transitions on annotations, $\rightsquigarrow_a$ for the async annotations, and $\rightsquigarrow_f$ for the finish ones. The essence is that the associated number can increase by one, and if any new thread identifiers appear, they must be globally fresh. Otherwise, thread ids are preserved.

$$
0 \rightsquigarrow_a (1, t) \text{ where } t \text{ is fresh} \qquad (1, t) \rightsquigarrow_a (2, t)
$$

$$
0 \rightsquigarrow_f 1 \qquad 1 \rightsquigarrow_f (2, t_1) \text{ where } t_1 \text{ is fresh}
$$

$$
(2, t_1) \rightsquigarrow_f (3, t_1, t_2) \text{ where } t_2 \text{ is fresh} \qquad (n, t_1, t_2) \rightsquigarrow_f (n + 1, t_1, t_2) \text{ where } 3 \le n \le 5
$$

Next, we can define the operational semantics for the instrumented language. For the most part, it is the same as the source language, with key differences being the reduction for annotation steps and the separate preparatory finish construct.

$$
\frac{}{h, \texttt{fst}\ (v_1, v_2) \to v_1, h} \qquad \frac{}{h, \texttt{snd}\ (v_1, v_2) \to v_2, h} \qquad \frac{}{h, \texttt{let}\ x = v\ \texttt{in}\ e \to e[x \mapsto v], h}
$$

$$
\frac{}{h, \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end}\ v \to e[x \mapsto v, f \mapsto \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end}], h}
$$

$$
\frac{l \notin \text{dom}(h)}{h, \texttt{alloc} \to l, h[l \mapsto ()]} \qquad \frac{h(l) = v}{h, !\, l \to v, h} \qquad \frac{l \in \text{dom}(h)}{h, l := v \to (), h[l \mapsto v]}
$$

$$
\frac{t \text{ is fresh}}{h, \texttt{async}(e)^0 \to \texttt{async}(e)^{(1,t)}} \qquad \frac{}{h, \texttt{finish}(\{(L[\texttt{async}(e)^{(1,t)}])^{t'}\} \uplus S) \to \texttt{finish}(\{(L[\texttt{async}(e)^{(2,t)}])^{t'}\} \uplus S), h}
$$

$$
\frac{}{h, \texttt{finish}(\{(L[\texttt{async}(e)^{(2,t)}])^{t'}\} \uplus S) \to \texttt{finish}(\{(L[()])^{t'}, e^t\} \uplus S), h}
$$

$$
\frac{a \rightsquigarrow_f a'}{h, \texttt{finish}(e)^a \to \texttt{finish}(e)^{a'}, h} \qquad \frac{}{h, K[(L[\texttt{finish}(e)^{(6,t_1,t_2)}])^t] \to K[(L[\texttt{finish}(\{e^{t_1}\})])^{t_2}], h}
$$

$$
\frac{\forall e, t.\ e^t \in S \Rightarrow e = ()}{h, \texttt{finish}(S) \to (), h} \qquad \frac{h, e \to e', h'}{h, K[(L[e])^t] \to K[(L[e'])^t], h'}
$$

Lemma 16. *Evaluation preserves well-formedness, i.e., for any* e, e', h, h'*, if* $\mathrm{wf}(e)$ *and* $h, e \to e', h$*, then* $\mathrm{wf}(e')$.

D.3 Connecting instrumented language and source language

In order to connect the instrumented language expressions to the source language ones, we introduce an *erasure* function: $\lfloor - \rfloor : \mathcal{E}_I \to \mathcal{E}_O$, that removes annotations from terms and maps $\texttt{finish}(e)^a$ to $\texttt{finish}(\{\lfloor e \rfloor\})$. Erasure function extends in the obvious way to heaps and evaluation contexts.

Lemma 17 (Instr-Step)**.** *For any expressions* $e, e' \in \mathcal{E}_I$ *and heaps* $h, h' \in \mathcal{H}_I$*, if* $h, e \to_I e', h'$*, then either* $\lfloor e \rfloor = \lfloor e' \rfloor$ *and* $h = h'$*, or* $\lfloor h \rfloor, \lfloor e \rfloor \to_O \lfloor e' \rfloor, \lfloor h' \rfloor$.

Proof. By induction on the reduction judgment. These split into three forms: the administrative reductions of `async` and `finish`, which give the same erasure for both the redex and the reduct, the reductions of sequential constructs, which can be trivially matched, and the two rules that spawn an async thread and terminate the finish block — which also can be easily matched. □

Definition 7. An *administrative* reduction in the instrumented language is a reduction $h, e \to e', h'$ such that $\lfloor e \rfloor = \lfloor e' \rfloor$ and $h = h'$. We write $h, e \to_a e', h'$ for administrative steps.

Lemma 18 (Admin-Fin)**.** *There are no infinite sequences of administrative reductions, i.e., for any* $e \in \mathcal{E}_I$ *and* $h \in \mathcal{H}_I$*, if* $\mathrm{wf}(e)$ *then* $h, e \not\to_a^\infty$.

Proof. Assign as a weight of the expression based on the number in each of its annotations in evaluation positions. Take $\mathrm{num}(a)$ to denote the number associated with the annotation a. Then define

$$w(e) = \begin{cases} 2 - \mathrm{num}(a) & \text{if } e = L[\texttt{async}(e')^a] \\ 6 - \mathrm{num}(a) & \text{if } e = L[\texttt{future}(e')^a] \\ \sum_{e' \in S} w(e) & \text{if } e = L[\texttt{future}(S)] \\ 0 & \text{otherwise} \end{cases}.$$

Observe that each administrative reduction decreases the weight. The lemma then holds by simple induction on the weight. □

Lemma 19 (Instrument)**.** *For any expression* $e \in \mathcal{E}_I$ *and heaps* $h, h' \in \mathcal{H}_I$ *such that* $h, e \to_I^* (), h', \lfloor h \rfloor, \lfloor e \rfloor \to_O^* (), \lfloor h' \rfloor$*. Moreover, if* $h, e \to_I^\infty$ *then* $\lfloor h \rfloor, \lfloor e \rfloor \to_O^\infty$.

Proof. First part, by induction on the reduction sequence and Lemma 17.

Similarly, for the second part we proceed by coinduction: since $h, e \to_I^\infty$, by Lemma 10 and 17 there exist e' and h' such that $h, e \to_I^* e', h', \lfloor h \rfloor, \lfloor e \rfloor \to_O \lfloor e' \rfloor, \lfloor h' \rfloor$ and $h', e' \to_I^\infty$. Hence, $\lfloor h' \rfloor, \lfloor e' \rfloor \to_O^\infty$ and finally $\lfloor h \rfloor, \lfloor e \rfloor \to_O^\infty$. □

D.4 Connecting instrumented language and λ^{DC}

Translation of surface expressions into λ^{DC} Note that the translation trivially extends to the evaluation contexts of the instrumented language. We define a translation $[\![- \mid -]\!]^S : \mathcal{V}_{\lambda^{DC}} \times \mathcal{E}_I \to \mathcal{E}_{\lambda^{DC}}$, where the first argument keeps track of the *nearest enclosing finish clock*. The bulk of the translation is structural:

$$[\![t \mid \texttt{let}\ x = e_1\ \texttt{in}\ e_2]\!] = \texttt{let}\ x = [\![t \mid e_1]\!]\ \texttt{in}\ [\![t \mid e_2]\!]$$
$$[\![t \mid \texttt{fun}\ f\ x\ \texttt{is}\ e\ \texttt{end}]\!] = \texttt{fun}\ f\ (x, t')\ \texttt{is}\ [\![t' \mid e]\!]\ \texttt{end}$$
$$[\![t \mid v_1\ v_2]\!] = [\![t \mid v_1]\!]\ ([\![t \mid v_2]\!], t)$$

The translation for parallel primitives is defined as follows:

```
1  [[t | async(e)^0]] =
2    let t' = newTd [[t | e]]
3    in newEdge (t', t); release t'; ()
4
5  [[t | finish(e)^0]] =
6    capture (fun _ k is
7      let t2 = newTd (k ())
8          t1 = newTd ([[t2 | e]])
9      in newEdge (t1, t2); transfer t2; release t2; release t1; ())
```

Note that due to its destination-passing flavour, the translation for values does not depend on the thread identifier. Thus, we can also extend this translation to work on heaps: we denote it $[\![-]\!]^H : \mathcal{H}_I \to \mathcal{H}^{DC}_{\lambda}$.

Translation of the well-formed expressions into λ^{DC} The translation for well-formed, thread-annotated expressions is as follows, defined mutually recursively with the translation for expressions. The latter has the form $[\![-]\!] : \mathcal{E}_I \to \mathrm{TId} \times \mathrm{TId}^{\perp} \to \mathcal{D}$, where $\mathcal{D}$ denotes computation DAGs of the form (V, E). The argument that tracks the continuation vertex is $\perp$ outside the topmost finish block. We define a helper function

$$\mathrm{mkEdge}\,(t, t') = \begin{cases} \emptyset & \text{if } t' = \perp \\ \{(t, t')\} & \text{otherwise} \end{cases}$$

The definition for thread-annotated expressions is just shorthand: the thread-annotation is passed as an argument to the main translation.

$$[\![e^{t'}]\!](t') = [\![e]\!](t, t')$$

$$
\begin{aligned}
&[\![e]\!](t,t') = \\
&\quad [t \mapsto ([\![e]\!]^S, \mathsf{R})], \mathrm{mkEdge}\,(t,t') \quad \text{if } \mathrm{surf}(e) \\
&[\![L[\mathtt{async}(e)^{(1,t_a)}]]\!](t,t') = \\
&\quad [t \mapsto ([\![t' \mid L]\!]^S\,[\mathtt{async1}(t_a,t')], \mathsf{R}), t_a \mapsto ([\![t' \mid e]\!]^S, \mathsf{N})], \mathrm{mkEdge}\,(t,t') \\
&[\![L[\mathtt{async}(e)^{(2,t_a)}]]\!](t,t') = \\
&\quad [t \mapsto ([\![t' \mid L]\!]^S\,[\mathtt{async2}(t_a)], \mathsf{R}), t_a \mapsto ([\![t' \mid e]\!]^S, \mathsf{N})], \mathrm{mkEdge}\,(t,t') \cup \mathrm{mkEdge}\,(t_a,t') \\
&[\![L[\mathtt{finish}(e)^1]]\!](t,t') = \\
&\quad [t \mapsto (\mathtt{finish1}(e, [\![t' \mid L]\!]^S), \mathsf{R})], \mathrm{mkEdge}\,(t,t') \\
&[\![L[\mathtt{finish}(e)^{(2,t_f)}]]\!](t,t') = \\
&\quad [t \mapsto (\mathtt{finish2}([\![t_f \mid e]\!]^S, t_f), \mathsf{R}), t_f \mapsto (\mathtt{cont}([\![t' \mid L]\!]^S), \mathsf{N})], \mathrm{mkEdge}\,(t,t') \\
&[\![L[\mathtt{finish}(e)^{(3,t_f,t_a)}]]\!](t,t') = \\
&\quad [t \mapsto (\mathtt{finish3}(t_a,t_f), \mathsf{R}), t_f \mapsto (\mathtt{cont}([\![t' \mid L]\!]^S), \mathsf{N}), t_a \mapsto ([\![t_f \mid e]\!]^S, \mathsf{N})], \mathrm{mkEdge}\,(t,t') \\
&[\![L[\mathtt{finish}(e)^{(4,t_f,t_a)}]]\!](t,t') = \\
&\quad [t \mapsto (\mathtt{finish4}(t_a,t_f), \mathsf{R}), t_f \mapsto (\mathtt{cont}([\![t' \mid L]\!]^S), \mathsf{N}), t_a \mapsto ([\![t_f \mid e]\!]^S, \mathsf{N})], \mathrm{mkEdge}\,(t,t') \cup \{(t_a,t_f)\} \\
&[\![L[\mathtt{finish}(e)^{(5,t_f,t_a)}]]\!](t,t') = \\
&\quad [t \mapsto (\mathtt{finish5}(t_a,t_f), \mathsf{R}), t_f \mapsto (\mathtt{cont}([\![t' \mid L]\!]^S), \mathsf{N}), t_a \mapsto ([\![t_f \mid e]\!]^S, \mathsf{N})], \mathrm{mkEdge}\,(t_f,t') \cup \{(t_a,t_f)\} \\
&[\![L[\mathtt{finish}(e)^{(6,t_f,t_a)}]]\!](t,t') = \\
&\quad [t \mapsto (\mathtt{finish6}(t_a,t_f), \mathsf{R}), t_f \mapsto (\mathtt{cont}([\![t' \mid L]\!]^S), \mathsf{R}), t_a \mapsto ([\![t_f \mid e]\!]^S, \mathsf{N})], \mathrm{mkEdge}\,(t_f,t') \cup \{(t_a,t_f)\} \\
&[\![L[\mathtt{finish}(S)]]\!](t,t') = \\
&\quad ([t \mapsto (\mathtt{cont}([\![t' \mid L]\!]^S), \mathsf{R})], \mathrm{mkEdge}\,(t,t')) \uplus \biguplus_{e'^e \in S} [\![e'^e]\!](t)
\end{aligned}
$$

The macrodefinitions used above are as follows:

```
1  async1(t, t') = newEdge (t, t'); release t; ()
2
3  async2(t) = release t; ()
4
5  finish1(e*), K) =let t' = newTd ((fun f z => K[z]) ())t = newTd ((*@[|t2 | e|])
6    in newEdge (t, t'); transfer t';
7       release t'; release t1; ()
8
9  finish2(e, t') =
10   let t = newTd (e)
11   in newEdge (t, t'); transfer t'; release t'; release t; ()
12
13 finish3(t, t') =
14   newEdge (t, t'); transfer t'; release t'; release t; ()
15
16 finish4(t, t') =
17   transfer t'; release t'; release t; ()
18
19 finish5(t, t') = release t'; release t; ()
20
21 finish6(t) = release t; ()
22
23 cont(K) = (fun _ z => K[z]) ()
```

Correctness

Definition 8. We say that a configuration of annotated instrumented expression and state e^t, h such that $\mathrm{wf}(e^t)$ and $\mathrm{surf}(h)$ *matches* a λ^{DC} state V, E, σ, written $e^t, h \propto V, E, \sigma$ if $\sigma = [\![h]\!]^H$, and there exists a DAG (V', E') such that $[\![() \mid e^t]\!] = V', E'$ and a finite map of vertices V'' such that $\forall t \in \mathrm{dom}(V'').\ V''(t) = ((), \mathsf{R})$, and that

$$V' \uplus V'', E', \sigma \twoheadrightarrow_{\mathrm{s}}^* V, E, \sigma.$$

Lemma 20. *For any e^{t_0}, h, V, E, σ, t if* $\mathrm{wf}(e)$, *$e^{t_0}, h \propto V, E, \sigma$, $t \in \mathrm{dom}(V)$ and* $\mathrm{status}(V(t)) = \mathsf{X}$, *then either $V(t) = ((), \mathsf{X})$, or there exist K, e' such that $e^{t_0} = K[e'^t]$.*

Lemma 21. *For any e^{t_0}, h, V, V', E, E', σ, σ' if* $\mathrm{wf}(e^{t_0})$, *$e^{t_0}, h \propto V, E, \sigma$ and $V, E, \sigma \twoheadrightarrow V', E', \sigma'$ then either $e^{t_0}, h \propto V', E', \sigma'$ or there exist $e'^{t'_0}$, h' such that $h, e^{t_0} \to e'^{t'_0}, h'$ and $e'^{t'_0}, h' \propto V', E', \sigma'$.*

Lemma 22. *For any e, t, h, V, V', E, σ, σ', if* $\mathrm{wf}(e^t)$, *$e^t, h \propto V, E, \sigma$ and $\forall t' \in \mathrm{dom}(V')$. $V'(t') = ((), \mathsf{F})$, then the following simulation holds:*

$$V, E, \sigma \twoheadrightarrow^* V', \emptyset, \sigma' \quad \Rightarrow \quad \exists v, t', h'.\ h, e^t \to^* v^{t'}, h' \wedge \sigma' = [\![h']\!]^H$$
$$V, E, \sigma \twoheadrightarrow^\infty \quad \Rightarrow \quad h, e^t \to^\infty$$

Proof. Follows from Lemmas 21 and 1 by the same inductive and coinductive argument as used in the proof of Lemma 19. □

Theorem 3 (Correctness)**.** *Let t be the identifier of the main thread, e be the source program stored in this thread, l be a designated location in which to store the final result, and e_I be the result of annotating e with* 0 *on all* `finish` *and* `async` *constructs. For any integer result n, final state σ such that $\sigma(l) = n$, and final set of vertices V, assuming that all threads t' in V are finished (i.e.* $\mathrm{status}\,(V(t')) = \mathsf{F}$*), we have:*

$$[t \mapsto (\texttt{let}\ x = [\![() \mid e_I]\!]^S\ \texttt{in}\ l := x, \mathsf{R})], \emptyset, [l \mapsto ()] \twoheadrightarrow^* V, \emptyset, \sigma \ \Rightarrow\ \exists h.\ \emptyset, e \to_O^* n, h$$

Furthermore, divergence in the DAG calculus entails divergence in the source language:

$$[t \mapsto (\texttt{let}\ x = [\![() \mid e_I]\!]^S\ \texttt{in}\ l := x, \mathsf{R})], \emptyset, [l \mapsto ()] \twoheadrightarrow^\infty \quad \Rightarrow \quad \emptyset, e \to_O^\infty .$$

Proof. This theorem follows from the composition of the two simulation diagrams, given by Lemmas 19 and 22. Clearly, $\lfloor e_I \rfloor = e$. Moreover, since $\mathrm{surf}(e_I)$, we also have $[\![(\texttt{let}\ x = e_I\ \texttt{in}\ l := x)^t]\!](\bot) = [t \mapsto (\texttt{let}\ x = [\![() \mid e_I]\!]^S\ \texttt{in}\ l := x, \mathsf{R})], \emptyset$, which gives us $(\texttt{let}\ x = e_I\ \texttt{in}\ l := x)^t, [l \mapsto ()] \propto [t \mapsto (\texttt{let}\ x = [\![() \mid e_I]\!]^S\ \texttt{in}\ l := x, \mathsf{R})], \emptyset, [l \mapsto ()]$.

For the termination case, by Lemma 22 we obtain that there exist v, t' and h such that $[l \mapsto ()], (\texttt{let}\ x = e_I\ \texttt{in}\ l := x)^t \to_I^* v^{t'}, h$ and $\sigma' = [\![h]\!]^H$ (and hence $h(l) = n$). Since the final command of the program is assignment, we can conclude that $v = ()$. By Lemma 19 we can now conclude that $[l \mapsto ()], \texttt{let}\ x = e\ \texttt{in}\ l := x \to_O^* (), \lfloor h \rfloor$. Since $h(l) = n$, by Lemma 15 this gives us that $\emptyset, e \to_O^* n, \lfloor h \rfloor \setminus [l \mapsto n]$, which ends the proof.

For the nontermination case, from Lemma 22 conclude that $[l \mapsto ()], (\texttt{let}\ x = e_I\ \texttt{in}\ l := x)^t \to_I^\infty$, which, by Lemma 19 implies that $[l \mapsto ()], \texttt{let}\ x = e\ \texttt{in}\ l := x \to_O^\infty$. Finally, by Lemma 15 we obtain $\emptyset, e \to_O^\infty$ which ends the proof. □