# Appendix

## A EVALUATION RULES FOR SECTION 3

Figure 1 shows selected evaluation rules for the core language. Evaluation is based on the reduction relation $\rightsquigarrow$. The (unlisted) congruence rules, which evaluate a sub-expression within a larger expression, are all standard. Rule RECORD FIELD reduces a record to the expression associated with the applied label. Rule VARIANT CASE evaluates a case expression $\texttt{case}\ \texttt{l}_\texttt{k}\ \texttt{e}\ \texttt{of}\ \overrightarrow{\texttt{l}_\texttt{i}\ \texttt{x}_\texttt{i} \to \texttt{e}_\texttt{i}}$ over a variant by matching $\texttt{l}_\texttt{k}\ \texttt{e}$ with the corresponding case $\texttt{l}_\texttt{k}\ \texttt{x}_\texttt{k}$, returning $\texttt{e}_\texttt{k}[\texttt{x}_\texttt{k} = \texttt{e}]$. Rules BETA BOP, BETA UOP, and BETA EQ evaluate the built-in operators. Since the language is both confluent and strongly normalizing, the order in which we apply the evaluation rules does not matter and evaluation always terminates. We write $[\![\texttt{e}]\!]$ to represent the unique term obtained by exhaustive application of the evaluation rules to e. We write $\rightsquigarrow_*$ for the reflexive-transitive closure of $\rightsquigarrow$.

$$\frac{}{\texttt{if true then e}_1\ \texttt{else e}_2 \rightsquigarrow \texttt{e}_1}\ \text{IF TRUE} \qquad \frac{}{\texttt{if false then e}_1\ \texttt{else e}_2 \rightsquigarrow \texttt{e}_2}\ \text{IF FALSE}$$

$$\frac{}{\texttt{case l}_\texttt{k}\ \texttt{e of}\ \overrightarrow{\texttt{l}_\texttt{i}\texttt{x}_\texttt{i} \to \texttt{e}'_\texttt{i}} \rightsquigarrow \texttt{e}'_\texttt{k}[\texttt{x}_\texttt{k} = \texttt{e}]}\ \text{CASE VARIANT} \qquad \frac{}{\{\overrightarrow{\texttt{l}_\texttt{i} = \texttt{e}_\texttt{i}}\}.\texttt{l}_\texttt{j} \rightsquigarrow \texttt{e}_\texttt{j}}\ \text{RECORD FIELD} \qquad \frac{\texttt{e} \rightsquigarrow \texttt{e}'}{\lambda\texttt{x}.\ \texttt{e} \rightsquigarrow \lambda\texttt{x}.\ \texttt{e}'}\ \text{CONG}$$

$$\frac{}{(\lambda\texttt{x}.\ \texttt{e}_1)\ \texttt{e}_2 \rightsquigarrow \texttt{e}_1[\texttt{x} = \texttt{e}_2]}\ \text{BETA} \qquad \frac{}{\texttt{bop b}_1\ \texttt{b}_2 \rightsquigarrow \texttt{bop}(\texttt{b}_1, \texttt{b}_2)}\ \text{BETA BOP} \qquad \frac{}{\texttt{uop b} \rightsquigarrow \texttt{uop}(\texttt{b})}\ \text{BETA UOP} \qquad \frac{}{\texttt{eq b}_1\ \texttt{b}_2 \rightsquigarrow (\texttt{b}_1 == \texttt{b}_2)}\ \text{BETA EQ}$$

Fig. 1. Selected evaluation rules for our core language.

## B PROOFS FOR SECTION 3

PROOF OF THEOREM 3.10. We proceed by induction on the input sequence *is*. The base case [] is clear. For the inductive step, assume an input sequence of the form $i : is$. We have the chain of equalities below:

$$\begin{aligned}
&\mathcal{M}_{(c_1, s_1)}(i : is) \\
&=^{(1)} o : \mathcal{M}_{(c_1, s'_1)}(is) \\
&=^{(2)} o : \mathcal{M}_{(c_2, s'_2)}(is) \\
&=^{(3)} \mathcal{M}_{(c_2, s_2)}(i : is).
\end{aligned}$$

Step (1) follows by definition of $\mathcal{M}$, where $(s'_1, o) = [\![c_1\ (s_1, i)]\!]$, and step (2) from the induction hypothesis, where $s'_2 = [\![p\ s'_1]\!]$. Step (3) follows from the definition of $\mathcal{M}$, where we have the chain

of equalities below:

$$\begin{aligned}
&[\![c_2\ (s_2, i)]\!] \\
=^{(1)}\ &[\![c_2\ (p\ s_1, i)]\!] \\
=^{(2)}\ &[\![pc_2\ (s_1, i)]\!] \\
=^{(3)}\ &[\![c_1 p\ (s_1, i)]\!] \\
=^{(4)}\ &[\![(s_2', o)]\!].
\end{aligned}$$

Here step (1) follows from the assumption $p\ s_1 \simeq s_2$, which means $[\![p\ s_1]\!] = [\![s_2]\!]$; step (2) from the definition of $pc_2$; step (3) from the assumption $c_1 p \simeq pc_2$; and step (4) from the definition of $c_1 p$. □

Proof of Theorem 3.7. Given circuits $(c, s_c) : \big((S, I) \to (S, O), S\big)$ and $(obs, s_o) : \big((S_o, O) \to (S_o, O_o), S_o\big)$, we need to show that for any circuit $(leak, s_l) : \big((S_l, I) \to (S_l, O_l), S_l\big)$, the two conditions below are equivalent:

(1) There exists a simulator $(sim, s_s) : \big((S_s, O_l) \to (S_s, O_o), S_s\big)$ such that

$$(c, s_c) \circ (obs, s_o) =_{\mathcal{M}} (leak, s_l) \circ (sim, s_s).$$

(2) For all input sequences $is, is' : [I]$, we have that

$$\mathcal{M}_{(leak, s_l)}(is) = \mathcal{M}_{(leak, s_l)}(is') \text{ implies } \mathcal{M}_{(c, s_c) \circ (obs, s_o)}(is) = \mathcal{M}_{(c, s_c) \circ (obs, s_o)}(is').$$

Without loss of generality, we can treat the composition $(c, s_c) \circ (obs, s_o)$ as a monolithic circuit $(c, s_c) : \big((S, I) \to (S, O_o), S\big)$.

*(1) implies (2):* The condition $(c, s_c) =_{\mathcal{M}} (leak, s_l) \circ (sim, s_s)$ directly translates to

$$(c, s_c) =_{\mathcal{M}} \big(leak \circ sim, (s_l, s_s)\big). \tag{$\star_1$}$$

Fix input sequences $is, is' : [I]$ such that

$$\mathcal{M}_{(leak, s_l)}(is') = \mathcal{M}_{(leak, s_l)}(is'). \tag{$\star_2$}$$

We have the chain of equalities below:

$$\begin{aligned}
&\mathcal{M}_{(c, s_c)}(is) \\
=^{(1)}\ &\mathcal{M}_{(leak \circ sim, (s_l, s_s))}(is) \\
=^{(3)}\ &\mathcal{M}_{(sim, s_s)}\big(\mathcal{M}_{(leak, s_l)}(is')\big) \\
=^{(3)}\ &\mathcal{M}_{(sim, s_s)}\big(\mathcal{M}_{(leak, s_l)}(is')\big) \\
=^{(4)}\ &\mathcal{M}_{(leak \circ sim, (s_l, s_s))}(is') \\
=^{(5)}\ &\mathcal{M}_{(c, s_c)}(is')
\end{aligned}$$

Steps (1) and (5) follow from ($\star_1$); steps (2) and (4) from Lemma B.1, and step (3) from ($\star_2$).

*(2) implies (1):* Without loss of generality, we can assume that the states $s_c, s_l$ are already in normal form. We construct a simulator as follows. Fix a term $o_\star : O_o$. For any core type $\mathsf{s}$, let $Terms[\mathsf{s}]$ denote the set of closed terms $\mathsf{e} : \mathsf{s}$ in normal form (that is, such that $[\![\mathsf{e}]\!] = \mathsf{e}$). Crucially, this set is finite. To define a simulator, we need to give the internal state type $S_s$, the initial state $s_s : S_s$, and the tick function $sim : (S_s, O_l) \to (S_s, O_o)$.

**The Internal State Type.** Let $\mathcal{P}(-)$ denote the power set operator on sets. Let

$$N + 1 = \big|\mathcal{P}\big(Terms[(S_l, S)]\big)\big|$$

be the cardinality of the set of all subsets of closed terms of type $(S_l, S)$ in normal form, and let

$$\phi : \{0, \ldots, N\} \to \mathcal{P}\big(\mathit{Terms}[(S_l, S)]\big)$$

be its enumeration. The state type is the record type

$$S_s \coloneqq \big\{\mathtt{proj}[0] : (), \ldots, \mathtt{proj}[N] : ()\big\}$$

That is, the internal state type of the simulator encodes all possible subsets of closed terms of type $(S_l, S)$ in normal form. Informally, each subset represents the possible configuration of states that we reach after simultaneously running the circuits $(\mathit{leak}, s_l)$ and $(c, s_c)$ on the same input sequence.

**The Initial State.** The initial state $s_s \coloneqq \mathtt{proj}\big[\phi^{-1}\big(\{(s_l,\ s_c)\}\big)\big]$ corresponds to the singleton subset consisting of the initial configuration $(s_l, s_c)$.

**The Tick Function.** For a core type $\mathtt{s}$, let $=_\mathtt{s} : \mathtt{s} \to \mathtt{s} \to \mathtt{Bool}$ be the Boolean comparison operator that internalizes syntactic equality, *i.e.*, such that $[\![\mathtt{e}_1 =_\mathtt{s} \mathtt{e}_2]\!] = \mathtt{true}$ if and only if $[\![\mathtt{e}_1]\!] = [\![\mathtt{e}_1]\!]$. We define $=_\mathtt{s}$ by induction on $\mathtt{s}$ as follows:

- If $\mathtt{s} = \mathtt{w}$ is a base type, we use the corresponding built-in comparison operator. For example, if $\mathtt{w} = \mathtt{Word8}$, we define $=_\mathtt{s}$ as the function $\lambda\,\mathtt{x}.\ \lambda\,\mathtt{y}.\ \mathtt{x} =_8 \mathtt{y}$.
- If $\mathtt{s} = \mathtt{Bool}$ is the type of Booleans, we define $=_\mathtt{s}$ as the function

$$\begin{aligned}&\lambda\,\mathtt{x}.\ \lambda\,\mathtt{y}.\ \mathtt{if}\ \mathtt{x}\ \mathtt{then}\ \mathtt{if}\ \mathtt{y}\ \mathtt{then}\ \mathtt{true}\ \mathtt{else}\ \mathtt{false}\\ &\qquad\qquad\mathtt{else}\ \mathtt{if}\ \mathtt{y}\ \mathtt{then}\ \mathtt{false}\ \mathtt{else}\ \mathtt{true}\end{aligned}$$

- If $\mathtt{s} = \{\mathtt{l}_1 : \mathtt{s}_1, \ldots, \mathtt{l}_n : \mathtt{s}_n\}$ is a record type, we define $=_\mathtt{s}$ as the function

$$\begin{aligned}&\lambda\,\mathtt{x}.\ \lambda\,\mathtt{y}.\ \mathtt{if}\ \mathtt{x}.\mathtt{l}_1 =_{\mathtt{s}_1} \mathtt{y}.\mathtt{l}_1\\ &\qquad\mathtt{then}\ \ldots\ \mathtt{if}\ \mathtt{x}.\mathtt{l}_n =_{\mathtt{s}_n} \mathtt{y}.\mathtt{l}_n\\ &\qquad\qquad\mathtt{then}\ \mathtt{true}\\ &\qquad\qquad\mathtt{else}\ \mathtt{false}\\ &\qquad\mathtt{else}\ \mathtt{false}.\end{aligned}$$

- If $\mathtt{s} = \langle\{\mathtt{l}_1 : \mathtt{s}_1, \ldots, \mathtt{l}_n : \mathtt{s}_n\}\rangle$ is a variant type, we define $=_\mathtt{s}$ as the function

$$\begin{aligned}&\lambda\,\mathtt{x}.\ \lambda\,\mathtt{y}.\ \mathtt{case}\ \mathtt{x}\ \mathtt{of}\\ &\qquad\ldots\\ &\qquad\mathtt{l}_\mathtt{i}\ \mathtt{x}_\mathtt{i} \to\ \mathtt{case}\ \mathtt{y}\ \mathtt{of}\\ &\qquad\qquad\ldots\\ &\qquad\qquad\mathtt{l}_\mathtt{j}\ \mathtt{y}_\mathtt{j} \to \begin{cases}\mathtt{x}_\mathtt{i} =_{\mathtt{s}_\mathtt{i}} \mathtt{y}_\mathtt{j} & \text{if } i = j\\ \mathtt{false} & \text{if } i \neq j\end{cases}\\ &\qquad\qquad\ldots\\ &\qquad\ldots.\end{aligned}$$

Let

$$M + 1 = \big|\mathit{Terms}[((S_l, S), I)]\big|$$

be the cardinality of the set of closed terms of type $((S_l, S), I)$ in normal form, and let

$$\psi : \{0, \ldots, M\} \to \mathit{Terms}[((S_l, S), I)]$$

be its enumeration. We now define a function $\mathit{Tick}(-, -, -)$, whose first argument is a number $m \in \{0, \ldots, M + 1\}$, the second is a set $X \subseteq \mathit{Terms}[((S_l, S), I)]$, and the third is a set $Y \subseteq \mathit{Terms}[(S_l, S)]$. The result will be a term $\mathtt{x} : O_l \vdash \mathit{Tick}(m, X, Y) : (S_s, O_o)$.

Given a set $X \subseteq \mathit{Terms}[((S_l, S), I)]$, we define sets $\mathit{RunSt}(X) \subseteq \mathit{Terms}[(S_l, S)]$ and $\mathit{RunOut}(X) \subseteq \mathit{Terms}[(O_l, O_o)]$ as follows:

$$\mathit{RunSt}(X) \coloneqq \big\{\big([\![\texttt{fst}\ (\mathit{leak}\ (s_1, i))]\!], [\![\texttt{fst}\ (c\ (s_2, i))]\!]\big) \mid ((s_1, s_2), i) \in X\big\},$$
$$\mathit{RunOut}(X) \coloneqq \big\{\big([\![\texttt{snd}\ (\mathit{leak}\ (s_1, i))]\!], [\![\texttt{snd}\ (c\ (s_2, i))]\!]\big) \mid ((s_1, s_2), i) \in X\big\}.$$

With these definitions in place, we put:

$$\mathit{Tick}(m, X, Y) \coloneqq \begin{cases} - \ \big(\texttt{proj}[\phi^{-1}(\emptyset)], o_\star\big) \\ \quad \text{if } m = 0 \text{ and } X = \emptyset \\ - \ \texttt{let}\ (\_, \texttt{o}) = c\ (s_2, i)\ \texttt{in}\ \big(\texttt{proj}[\phi^{-1}(\mathit{RunSt}(X))], \texttt{o}\big) \\ \quad \text{if } m = 0 \text{ and } X \neq \emptyset \text{ and } \psi(m') = ((s_1, s_2), i) \text{ and } m' \text{ smallest s.t. } \psi(m') \in X \\ - \ \mathit{Tick}(m', X, Y) \\ \quad \text{if } m = m' + 1 \text{ and } \psi(m') = ((s_1, s_2), i) \text{ and } (s_1, s_2) \notin Y \\ - \ \texttt{let}\ (\_, \texttt{o}_1) = \mathit{leak}\ (s_1, i)\ \texttt{in} \\ \quad \texttt{if}\ \texttt{o}_1 =_{O_l} \texttt{x}\ \texttt{then}\ \mathit{Tick}\big(m', X \cup \{((s_1, s_2), i)\}, Y\big)\ \texttt{else}\ \mathit{Tick}(m', X, Y) \\ \quad \text{if } m = m' + 1 \text{ and } \psi(m') = ((s_1, s_2), i) \text{ and } (s_1, s_2) \in Y \end{cases}$$

We define the tick function $\mathit{sim} : (S_s, O_l) \to (S_s, O_o)$ as follows:

$$\mathit{sim} \coloneqq \lambda\,(\texttt{s}_\texttt{s}, \texttt{x}).\ \texttt{case}\ \texttt{s}_\texttt{s}\ \texttt{of}\ \overrightarrow{\texttt{proj[n]} \to \mathit{Tick}\big(M + 1, \emptyset, \phi(\texttt{n})\big)}.$$

**Simulator Correctness.** Given a set $X \subseteq \mathit{Terms}[((S_l, S), I)]$, we define a set $X.1 \subseteq \mathit{Terms}[(S_l, S)]$ as follows:

$$X.1 \coloneqq \{(s_1, s_2) \mid ((s_1, s_2), i) \in X\}.$$

We call a set $Y \subseteq \mathit{Terms}[(S_l, S)]$ *coherent* if the following condition holds:

- For all input sequences $is, is' : [I]$ and all $(s_1, s_2), (s_1', s_2') \in Y$, we have that
$$\mathcal{M}_{(\mathit{leak}, s_1)}(is) = \mathcal{M}_{(\mathit{leak}, s_1')}(is') \text{ implies } \mathcal{M}_{(c, s_2)}(is) = \mathcal{M}_{(c, s_2')}(is').$$

To prove the Mealy equivalence $(c, s_c) =_{\mathcal{M}} \big(\mathit{leak} \circ \mathit{sim}, (s_l, s_s)\big)$, we show the following generalization:

- If $\phi(n)$ is coherent and $(s_1, s_2) \in \phi(n)$, then for any input sequence $is : [I]$, we have
$$\mathcal{M}_{(c, s_2)}(is) = \mathcal{M}_{(\mathit{leak} \circ \mathit{sim}, (s_1, \texttt{proj}[n]))}(is).$$

Since the singleton set $\{(s_l, s_c)\}$ is coherent by assumption (this is precisely condition (2)), the instantiation of the above generalization with $n = \phi^{-1}\big(\{(s_l, s_c)\}\big)$ and $s_1 = s_l$, $s_2 = s_c$ gives us the desired Mealy equivalence.

To prove the generalization, we proceed by induction on the input sequence $is$. The base case [] is clear. For the inductive step, assume an input sequence of the form $i : is$. Let $(t_1, o_l) = [\![\mathit{leak}\ (s_1, i)]\!]$ and $(t_2, o) = [\![c\ (s_2, i)]\!]$. We now establish the following invariant: for any $m \in \{0, \ldots, M + 1\}$ and sets $X \subseteq \mathit{Terms}[((S_l, S), I)]$, $Y \subseteq \mathit{Terms}[(S_l, S)]$, if

(a) $Y$ is coherent,
(b) $(s_1, s_2) \in Y$,
(c) $X.1 \subseteq Y$,
(d) $\mathit{RunOut}(X) \subseteq \{(o_l, o)\}$, and
(e) either $\big((s_1, s_2), [\![i]\!]\big) \in X$ or there exists an $m' < m$ such that $\psi(m') = \big((s_1, s_2), [\![i]\!]\big)$,

then there exists an $n' \in \{0, \ldots, N\}$ such that

(f) $[\![\mathit{Tick}(m, X, Y)[\texttt{x} \coloneqq o_l]]\!] = (\texttt{proj}[n'], o)$,

(g) $\phi(n')$ is coherent, and
(h) $(t_1, t_2) \in \phi(n')$.

To prove this invariant, we proceed by induction on $m$. We consider four cases:

**Case 1.** If $m = 0$ and $X = \emptyset$, then assumption (e) cannot be satisfied, so we are done.

**Case 2.** If $m = 0$ and $X \neq \emptyset$, let $n' = \phi^{-1}(\mathit{RunSt}(X))$. To prove (f), let $m'$ be the smallest natural number such that $\psi(m') \in X$, and let $((s_1', s_2'), i') = \psi(m')$. Let $(t_1', o_l') = [\![\mathit{leak}\ (s_1', i')]\!]$ and $(t_2', o') = [\![c\ (s_2', i')]\!]$. By assumption, $((s_1', s_2'), i') \in X$, therefore $(o_l', o') \in \mathit{RunOut}(X)$. By assumption (d), we must have $o_l' = o_l$ and $o' = o$. By definition, we have

$$\begin{aligned}
&[\![\mathit{Tick}(m, X, Y)[\mathtt{x} \coloneqq o_l]]\!] \\
&= \mathtt{let}\ (_, \mathtt{o}) = c\ (s_2, i)\ \mathtt{in}\ \big(\mathtt{proj}[\phi^{-1}(\mathit{RunSt}(X))], \mathtt{o}\big) \\
&= \big(\mathtt{proj}[\phi^{-1}(\mathit{RunSt}(X))], o\big).
\end{aligned}$$

Thus (f) holds. To prove (g), we need to show that the set $\mathit{RunSt}(X)$ is coherent. To this end, fix sequences $is', is'' : [I]$ and terms $((s_1', s_2'), i'), ((s_1'', s_2''), i'') \in X$. Let $(t_1', o_l') = [\![\mathit{leak}\ (s_1', i')]\!]$ and $(t_2', o') = [\![c\ (s_2', i')]\!]$. Let $(t_1'', o_l'') = [\![\mathit{leak}\ (s_1'', i'')]\!]$ and $(t_2'', o'') = [\![c\ (s_2', i')]\!]$. We want to show that

$$\mathcal{M}_{(\mathit{leak}, t_1')}(is') = \mathcal{M}_{(\mathit{leak}, t_1'')}(is'') \text{ implies } \mathcal{M}_{(c, t_2')}(is') = \mathcal{M}_{(c, t_2'')}(is'').$$

Since by assumption $((s_1', s_2'), i'), ((s_1'', s_2''), i'') \in X$, we must have $(o_l', o'), (o_l'', o'') \in \mathit{RunOut}(X)$ and $(s_1', s_2'), (s_1'', s_2'') \in X$.1. Now by assumption (d), we must have $o_l', o_l'' = o_l$ and $o', o'' = o$. And by assumption (c), we have $(s_1', s_2'), (s_1'', s_2'') \in Y$. Now assume $\mathcal{M}_{(\mathit{leak}, t_1')}(is') = \mathcal{M}_{(\mathit{leak}, t_1'')}(is'')$ and consider the two larger sequences $i' : is'$, $i'' : is''$. By assumption (a), $Y$ is coherent. Since $(s_1', s_2'), (s_1'', s_2'') \in Y$, we have that

$$\mathcal{M}_{(\mathit{leak}, s_1')}(i' : is') = \mathcal{M}_{(\mathit{leak}, s_1'')}(i'' : is'') \text{ implies } \mathcal{M}_{(c, s_2')}(i' : is') = \mathcal{M}_{(c, s_2'')}(i'' : is'').$$

Now we have the chain of equalities below:

$$\begin{aligned}
&\mathcal{M}_{(\mathit{leak}, s_1')}(i' : is') \\
&=^{(1)} o_l : \mathcal{M}_{(\mathit{leak}, t_1')}(is') \\
&=^{(2)} o_l : \mathcal{M}_{(\mathit{leak}, t_1'')}(is'') \\
&=^{(3)} \mathcal{M}_{(\mathit{leak}, s_1'')}(i'' : is'').
\end{aligned}$$

Steps (1) and (3) follow by definition of $\mathcal{M}$, and step (2) is an earlier assumption. Therefore, $\mathcal{M}_{(c, s_2')}(i' : is') = \mathcal{M}_{(c, s_2'')}(i'' : is'')$. This gives us the chain of equalities below:

$$\begin{aligned}
&o : \mathcal{M}_{(c, t_2')}(is') \\
&=^{(1)} \mathcal{M}_{(c, s_2')}(i' : is') \\
&=^{(2)} \mathcal{M}_{(c, s_2'')}(i'' : is'') \\
&=^{(3)} o : \mathcal{M}_{(c, t_2'')}(is'').
\end{aligned}$$

Steps (1) and (3) again follow by definition of $\mathcal{M}$, and step (2) is the observation above. Thus $\mathcal{M}_{(c, t_2')}(is') = \mathcal{M}_{(c, t_2'')}(is'')$ as desired, which proves (g). To prove (h), we note that by assumption (e), we must have $((s_1, s_2), [\![i]\!]) \in X$. Therefore $(t_1, t_2) \in \mathit{RunSt}(X)$ as desired.

**Case 3.** If $m = m' + 1$ and $\psi(m') = ((s_1', s_2'), i')$ such that $(s_1', s_2') \notin Y$, $\mathit{Tick}(m, X, Y) = \mathit{Tick}(m', X, Y)$. The existence of an $n'$ satisfying (f)–(h) follows directly from the induction hypothesis once we establish that assumption (e) also holds for $m'$ in place of $m$. From the corresponding assumption for $m$, we know that either $\big((s_1, s_2), [\![i]\!]\big) \in X$ or there exists an $m'' < m$ such that $\psi(m'') = \big((s_1, s_2), [\![i]\!]\big)$.

In the first case, we are done. In the second case, we have two possibilities: either $m'' < m'$ or $m'' = m'$. If $m'' < m'$, then we are done. If $m'' = m'$, then $\psi(m') = \big((s_1, s_2), [\![i]\!]\big)$. From an earlier assumption, we know that $\psi(m') = ((s_1', s_2'), i')$ and $(s_1', s_2') \notin Y$. Thus $s_1' = s_1$, $s_2' = s_2$, $i' = [\![i]\!]$. But this means $(s_1, s_2) \notin Y$, which contradicts assumption (b), showing that this case is impossible.

**Case 4.** If $m = m' + 1$ and $\psi(m') = ((s_1', s_2'), i')$ such that $(s_1', s_2') \in Y$, let $(t_1', o_l') = [\![leak\ (s_1', i')]\!]$ and $(t_2', o') = [\![c\ (s_2', i')]\!]$. If $o_l' \neq o_l$, then we have

$$
\begin{aligned}
&[\![Tick(m, X, Y)[\mathsf{x} := o_l]]\!] \\
&= \Big[\!\Big[\mathtt{let}\ (\_, \mathsf{o}_1) = leak\ (s_1', i')\ \mathtt{in} \\
&\quad \mathtt{if}\ \mathsf{o}_1 =_{O_l} o_l \\
&\qquad \mathtt{then}\ Tick\big(m', X \cup \{((s_1', s_2'), i')\}, Y\big)[\mathsf{x} := o_l] \\
&\qquad \mathtt{else}\ Tick(m', X, Y)[\mathsf{x} := o_l]\Big]\!\Big] \\
&= [\![Tick(m', X, Y)[\mathsf{x} := o_l]]\!].
\end{aligned}
$$

The existence of an $n'$ satisfying (f)–(h) now follows directly from the induction hypothesis once we establish that assumption (e) also holds for $m'$ in place of $m$. From the corresponding assumption for $m$, we know that either $\big((s_1, s_2), [\![i]\!]\big) \in X$ or there exists an $m'' < m$ such that $\psi(m'') = \big((s_1, s_2), [\![i]\!]\big)$. In the first case, we are done. In the second case, we have two possibilities: either $m'' < m'$ or $m'' = m'$. If $m'' < m'$, then we are done. If $m'' = m'$, then $\psi(m') = \big((s_1, s_2), [\![i]\!]\big)$. From an earlier assumption, we know that $\psi(m') = ((s_1', s_2'), i')$. Thus $s_1' = s_1$, $s_2' = s_2$, $i' = [\![i]\!]$. But this means $t_1' = t_1$, $o_l' = o_l$, which contradicts the assumption that $o_l' \neq o_l$, showing that this case is impossible.
On the other hand, if $o_l' = o_l$, then we have

$$
\begin{aligned}
&[\![Tick(m, X, Y)[\mathsf{x} := o_l]]\!] \\
&= \Big[\!\Big[\mathtt{let}\ (\_, \mathsf{o}_1) = leak\ (s_1', i')\ \mathtt{in} \\
&\quad \mathtt{if}\ \mathsf{o}_1 =_{O_l} o_l \\
&\qquad \mathtt{then}\ Tick\big(m', X \cup \{((s_1', s_2'), i')\}, Y\big)[\mathsf{x} := o_l] \\
&\qquad \mathtt{else}\ Tick(m', X, Y)[\mathsf{x} := o_l]\Big]\!\Big] \\
&= [\![Tick\big(m', X \cup \{((s_1', s_2'), i')\}, Y\big)[\mathsf{x} := o_l]]\!].
\end{aligned}
$$

The existence of an $n'$ satisfying (f)–(h) now follows directly from the induction hypothesis once we establish that assumptions (c)–(e) also hold for $m'$ and $X \cup \{((s_1', s_2'), i')\}$ in place of $m$ and $X$. To prove (c), we need to show that $(s_1', s_2') \in Y$. But this is precisely what we assumed earlier. To prove (d), we need to show that $o_l' = o_l$ and $o' = o$. Since we assumed that $o_l' = o_l$ in the first place, we only need to show $o' = o$. By assumption we have $(s_1', s_2') \in Y$. Furthermore, by assumption (b) we also have $(s_1, s_2) \in Y$. Consider the two singleton sequences $[i]$, $[i']$. Since $Y$ is coherent by assumption (a), we have that

$$\mathcal{M}_{(leak, s_1)}([i]) = \mathcal{M}_{(leak, s_1')}([i']) \text{ implies } \mathcal{M}_{(c, s_2)}([i]) = \mathcal{M}_{(c, s_2')}([i']).$$

Now

$$\mathcal{M}_{(leak, s_1)}([i]) = [o_l] = [o_l'] = \mathcal{M}_{(leak, s_1')}([i']).$$

Therefore, we must have

$$[o] = \mathcal{M}_{(c, s_2)}([i]) = \mathcal{M}_{(c, s_2')}([i']) = [o'],$$

and hence $o' = o$, which proves (d). To prove (e), we conclude from the corresponding assumption for $m$ and $X$ that we either have $\big((s_1, s_2), [\![i]\!]\big) \in X$ or there exists an $m'' < m$ such that $\psi(m'') = \big((s_1, s_2), [\![i]\!]\big)$. In the first case, we are done. In the second case, we have two possibilities: either $m'' < m'$ or $m'' = m'$. If $m'' < m'$, then we are done. If $m'' = m'$, then $\psi(m') = \big((s_1, s_2), [\![i]\!]\big)$. From an earlier assumption, we know that $\psi(m') = ((s_1', s_2'), i')$. Thus $s_1' = s_1$, $s_2' = s_2$, $i' = [\![i]\!]$. But this means $\big((s_1, s_2), [\![i]\!]\big) \in X \cup \{((s_1', s_2'), i')\}$ as desired, which proves (e). This finishes the proof of the invariant.

We now have the following:

$$\begin{aligned}
&[\![sim\ (\texttt{proj[n]}, o_l)]\!] \\
&= \Big[\!\Big[\texttt{case proj[n] of } \overrightarrow{\texttt{proj[n]} \to Tick\big(M+1, \emptyset, \phi(\texttt{n})\big)[\texttt{x} := o_l]}\Big]\!\Big] \\
&= [\![Tick\big(M+1, \emptyset, \phi(n)\big)[\texttt{x} := o_l]]\!].
\end{aligned}$$

Using the invariant that we have just established, we conclude that there exists an $n' \in \{0, \ldots, N\}$ such that

$$[\![Tick\big(M+1, \emptyset, \phi(n)\big)[\texttt{x} := o_l]]\!] = (\texttt{proj}[n'], o),$$

with $\phi(n')$ coherent and $(t_1, t_2) \in \phi(n')$, as assumptions (a)–(e) are clearly satisfied for $m = M+1$, $X = \emptyset$, $Y = \phi(n)$. We thus have

$$[\![leak \circ sim\ \big((s_1, \texttt{proj[n]}), i\big)]\!] = \big((t_1, \texttt{proj}[n']), o\big),$$

which gives us the chain of equalities below:

$$\begin{aligned}
&\mathcal{M}_{(c, s_2)}(i : is) \\
&=^{(1)} o : \mathcal{M}_{(c, t_2)}(is) \\
&=^{(2)} o : \mathcal{M}_{(leak \circ sim,\ (t_1, \texttt{proj}[n']))}(is) \\
&=^{(3)} \mathcal{M}_{(leak \circ sim,\ (s_1, \texttt{proj}[n]))}(i : is).
\end{aligned}$$

Step (1) follows by definition of $\mathcal{M}$; step (2) from the induction hypothesis, using that $\phi(n')$ is coherent and $(t_1, t_2) \in \phi(n')$; and step (3) from what we have just shown. This finishes the proof. □

Proof of Lemma 3.11. Given circuits $(c, s_c) : \big((S, I) \to (S, O), S\big)$ and $(obs, s_o) : \big((S_o, O) \to (S_o, O_o), S_o\big)$, we need to show that for any circuit $(leak, s_l) : \big((S_l, I) \to (S_l, O_l), S_l\big)$ and function $p : (S, S_o) \to (S_l, S_s)$, the two conditions below are equivalent:

(1) There exists a simulator $(sim, s_s) : \big((S_s, O_l) \to (S_s, O_o), S_s\big)$ such that

$$(c, s_c) \circ (obs, s_o) \hookrightarrow (leak, s_l) \circ (sim, s_s)$$

via the state projection function $p$.

(2) The following non-interference properties hold:

(2.1) **Initial State Correspondence**: We have $[\![\texttt{fst}\ (p\ (s_c, s_o))]\!] = [\![s_l]\!]$.

(2.2) **Tick State Correspondence**: For all $((s_1, s_2), i) : ((S, S_o), I)$, we have

$$\begin{aligned}
&[\![\texttt{let}\ ((\texttt{t}_1, \texttt{t}_2), _) = c \circ obs\ ((s_1, s_2), i)\ \texttt{in let}\ (\texttt{v}_1, _) = p\ (\texttt{t}_1, \texttt{t}_2)\ \texttt{in}\ \texttt{v}_1]\!] = \\
&[\![\texttt{let}\ (\texttt{u}_1, _) = p\ (s_1, s_2)\ \texttt{in let}\ (\texttt{v}_1, _) = leak\ (\texttt{u}_1, i)\ \texttt{in}\ \texttt{v}_1]\!].
\end{aligned}$$

(2.3) **Projection Coherence**: For all $((s_1, s_2), i), ((s_1', s_2'), i') : ((S, S_o), I)$, if

$$\begin{aligned}
&[\![\texttt{let}\ (\texttt{u}_1, \texttt{u}_2) = p\ (s_1, s_2)\ \texttt{in let}\ (_, \texttt{o}_1) = leak\ (\texttt{u}_1, i)\ \texttt{in}\ (\texttt{u}_2, \texttt{o}_1)]\!] = \\
&[\![\texttt{let}\ (\texttt{u}_1', \texttt{u}_2') = p\ (s_1', s_2')\ \texttt{in let}\ (_, \texttt{o}_1') = leak\ (\texttt{u}_1', i')\ \texttt{in}\ (\texttt{u}_2', \texttt{o}_1')]\!]
\end{aligned}$$

then we have

$$[\![\texttt{let}\ ((\mathsf{t}_1, \mathsf{t}_2), \mathsf{o}_o) = c \circ \mathit{obs}\ ((s_1, s_2), i)\ \texttt{in let}\ (\_, \mathsf{v}_2) = p\ (\mathsf{t}_1, \mathsf{t}_2)\ \texttt{in}\ (\mathsf{v}_2, \mathsf{o}_o)]\!] =$$
$$[\![\texttt{let}\ ((\mathsf{t}'_1, \mathsf{t}'_2), \mathsf{o}'_o) = c \circ \mathit{obs}\ ((s'_1, s'_2), i)\ \texttt{in let}\ (\_, \mathsf{v}'_2) = p\ (\mathsf{t}'_1, \mathsf{t}'_2)\ \texttt{in}\ (\mathsf{v}'_2, \mathsf{o}'_o)]\!].$$

Without loss of generality, we can treat the composition $(c, s_c) \circ (\mathit{obs}, s_o)$ as a monolithic circuit $(c, s_c) : \big((S, I) \to (S, O_o), S\big)$.

*(1) implies (2):* By assumption, $p$ is the state projection function witnessing the reduction $(c, s_c) \hookrightarrow (\mathit{leak}, s_l) \circ (\mathit{sim}, s_s)$. Recall the definitions

$$c_1 p = \lambda\,(\mathsf{s}, \mathsf{i}).\ \texttt{let}\ (\mathsf{t}, \mathsf{o}) = c\ (\mathsf{s}, \mathsf{i})\ \texttt{in}\ (p\ \mathsf{t}, \mathsf{o}),$$
$$pc_2 = \lambda\,(\mathsf{s}, \mathsf{i}).\ \mathit{leak} \circ \mathit{sim}\ (p\ \mathsf{s}, \mathsf{i}).$$

Then we have $p\ s_c \simeq (s_l, s_s)$ and $c_1 p \simeq pc_2$. To prove (2.1), we need to show $[\![\texttt{fst}\ (p\ s_c)]\!] = [\![s_l]\!]$. As $p\ s_c \simeq (s_l, s_s)$, we have $[\![p\ s_c]\!] = [\![(s_l, s_s)]\!] = \big([\![s_l]\!], [\![s_s]\!]\big)$. Therefore, $[\![\texttt{fst}\ (p\ s_c)]\!] = [\![s_l]\!]$ as desired. To show (2.2), fix term $(s, i) : (S, I)$. Let $(t, o) = [\![c\ (s, i)]\!]$, $(u_1, u_2) = [\![p\ s]\!]$, $(v_1, v_2) = [\![p\ t]\!]$. Let $(w_1, o_l) = [\![\mathit{leak}\ (u_1, i)]\!]$ and $(w_2, o_s) = [\![\mathit{sim}\ (u_2, o_l)]\!]$. The condition

$$[\![\texttt{let}\ (\mathsf{t}, \_) = c\ (s, i)\ \texttt{in let}\ (\mathsf{v}_1, \_) = p\ \mathsf{t}\ \texttt{in}\ \mathsf{v}_1]\!] =$$
$$[\![\texttt{let}\ (\mathsf{u}_1, \_) = p\ s\ \texttt{in let}\ (\mathsf{v}_1, \_) = \mathit{leak}\ (\mathsf{u}_1, i)\ \texttt{in}\ \mathsf{v}_1]\!]$$

reduces to $v_1 = w_1$. Since $c_1 p \simeq pc_2$, we have $[\![c_1 p\ (s, i)]\!] = [\![pc_2\ (s, i)]\!]$. This implies

$$((v_1, v_2), o) = [\![c_1 p\ (s, i)]\!] = [\![pc_2\ (s, i)]\!] = ((w_1, w_2), o_s).$$

Therefore $w_1 = v_1$ as desired. To prove (2.3), fix terms $(s, i), (s', i') : (S, I)$. Let $(t, o) = [\![c\ (s, i)]\!]$ and $(t', o') = [\![c\ (s', i')]\!]$. Let $(u_1, u_2) = [\![p\ s]\!]$ and $(u'_1, u'_2) = [\![p\ s']\!]$. Let $(v_1, v_2) = [\![p\ t]\!]$ and $(v'_1, v'_2) = [\![p\ t']\!]$. Let $(w_1, o_l) = [\![\mathit{leak}\ (u_1, i)]\!]$ and $(w'_1, o'_l) = [\![\mathit{leak}\ (u'_1, i')]\!]$. Let $(w_2, o_s) = [\![\mathit{sim}\ (u_2, o_l)]\!]$ and $(w'_2, o'_s) = [\![\mathit{sim}\ (u'_2, o'_l)]\!]$. Assume that

$$[\![\texttt{let}\ (\mathsf{u}_1, \mathsf{u}_2) = p\ s\ \texttt{in let}\ (\_, \mathsf{o}_1) = \mathit{leak}\ (\mathsf{u}_1, i)\ \texttt{in}\ (\mathsf{u}_2, \mathsf{o}_1)]\!] =$$
$$[\![\texttt{let}\ (\mathsf{u}'_1, \mathsf{u}'_2) = p\ s'\ \texttt{in let}\ (\_, \mathsf{o}'_1) = \mathit{leak}\ (\mathsf{u}'_1, i')\ \texttt{in}\ (\mathsf{u}'_2, \mathsf{o}'_1)]\!].$$

The above condition reduces to $(u_2, o_l) = (u'_2, o'_l)$, which implies $(w'_2, o'_s) = (w_2, o_s)$. We want to show that

$$[\![\texttt{let}\ (\mathsf{t}, \mathsf{o}) = c\ (s, i)\ \texttt{in let}\ (\_, \mathsf{v}_2) = p\ \mathsf{t}\ \texttt{in}\ (\mathsf{v}_2, \mathsf{o})]\!] =$$
$$[\![\texttt{let}\ (\mathsf{t}', \mathsf{o}') = c\ (s', i')\ \texttt{in let}\ (\_, \mathsf{v}'_2) = p\ \mathsf{t}\ \texttt{in}\ (\mathsf{v}'_2, \mathsf{o}')]\!].$$

The above condition reduces to $(v_2, o) = (v'_2, o')$. Since $c_1 p \simeq pc_2$, we have $[\![c_1 p\ (s, i)]\!] = [\![pc_2\ (s, i)]\!]$ and $[\![c_1 p\ (s', i')]\!] = [\![pc_2\ (s', i')]\!]$. This implies

$$((v_1, v_2), o) = [\![c_1 p\ (s, i)]\!] = [\![pc_2\ (s, i)]\!] = ((w_1, w_2), o_s),$$
$$((v'_1, v'_2), o') = [\![c_1 p\ (s', i')]\!] = [\![pc_2\ (s', i')]\!] = ((w'_1, w'_2), o'_s).$$

We thus have $v'_2 = w'_2 = w_2 = v_2$ and $o' = o'_s = o_s = o$ as desired.

*(2) implies (1):* Fix a term $o_\star : O_o$. As in the proof of Theorem 3.7, let $\mathit{Terms}[\mathsf{s}]$, where $\mathsf{s}$ is a core type, denote the set of closed terms $\mathsf{e} : \mathsf{s}$ in normal form. To define a simulator, we need to give the initial state $s_s : S_s$ and the tick function $\mathit{sim} : (S_s, O_l) \to (S_s, O_o)$.

**The Initial State.** The initial state $s_s := \texttt{snd}\ (p\ s_c)$ is the simulator part of the projected initial state $s_c$ of the original circuit.

**The Tick Function.** For a core type $\mathsf{s}$, let $=_{\mathsf{s}} : \mathsf{s} \to \mathsf{s} \to \mathrm{Bool}$ be the comparison operator that internalizes syntactic equality, as defined in the proof of Theorem 3.7. Let $M + 1 = |\mathit{Terms}[(S, I)]|$ be the cardinality of the set of closed terms of type $(S, I)$ in normal form, and let $\psi : \{0, \ldots, M\} \to$

$Terms[(S, I)]$ be its enumeration. We now define a function $Tick(-)$, whose argument is a number $m \in \{0, \ldots, M+1\}$. The result will be a term $\mathsf{x} : S_s, \mathsf{y} : O_l \vdash Tick(m) : (S_s, O_o)$.

$$Tick(m) \coloneqq \begin{cases} -\ (\mathsf{x}, o_\star) \quad \text{if } m = 0 \\ -\ \mathtt{let}\ (\mathsf{u}_1, \mathsf{u}_2) = p\ s\ \mathtt{in}\ \mathtt{let}\ (\_, \mathsf{o}_1) = leak\ (\mathsf{u}_1, i)\ \mathtt{in} \\ \quad \mathtt{if}\ (\mathsf{u}_2, \mathsf{o}_1) =_{(S_s, O_l)} (\mathsf{x}, \mathsf{y}) \\ \qquad \mathtt{then}\ \mathtt{let}\ (\mathsf{t}, \mathsf{o}) = c\ (s, i)\ \mathtt{in}\ \mathtt{let}\ (\_, \mathsf{v}_2) = p\ \mathsf{t}\ \mathtt{in}\ (\mathsf{v}_2, \mathsf{o}) \\ \qquad \mathtt{else}\ Tick(m', X, Y) \\ \quad \text{if } m = m' + 1 \text{ and } \psi(m') = (s, i) \end{cases}$$

We define the tick function $sim : (S_s, O_l) \to (S_s, O_o)$ as $sim \coloneqq \lambda\,(\mathsf{x}, \mathsf{y}).\ Tick(M+1)$.

**Simulator Correctness.** Recall the definitions

$$c_1 p = \lambda\,(\mathsf{s}, \mathsf{i}).\ \mathtt{let}\ (\mathsf{t}, \mathsf{o}) = c\ (\mathsf{s}, \mathsf{i})\ \mathtt{in}\ (p\ \mathsf{t}, \mathsf{o}),$$
$$p c_2 = \lambda\,(\mathsf{s}, \mathsf{i}).\ leak \circ sim\ (p\ \mathsf{s}, \mathsf{i}).$$

To show that $(c, s_c) \hookrightarrow (leak, s_l) \circ (sim, s_s)$ via the state projection function $p$, we need to show that $p\ s_c \simeq (s_l, s_s)$ and $c_1 p \simeq p c_2$. To establish that $p\ s_c \simeq (s_l, s_s)$, we need to show that $[\![p\ s_c]\!] = [\![(s_l, s_s)]\!]$. Let $(u_1, u_2) = [\![p\ s_c]\!]$. From condition (2.1), we know that $[\![\mathtt{fst}\ (p\ s_c)]\!] = [\![s_l]\!]$. We thus have

$$[\![p\ s_c]\!] = (u_1, u_2) = \left([\![\mathtt{fst}\ (p\ s_c)]\!], [\![\mathtt{snd}\ (p\ s_c)]\!]\right) = \left([\![s_l]\!], [\![s_s]\!]\right) = [\![(s_l, s_s)]\!]$$

as desired. To show $c_1 p \simeq p c_2$, it suffices to prove that for any term $(s, i) : Terms[(S, I])$, we have $[\![c_1 p\ (s, i)]\!] = [\![p c_2\ (s, i)]\!]$. To this end, let $(t, o) = [\![c\ (s, i)]\!]$, $(u_1, u_2) = [\![p\ s]\!]$, $(v_1, v_2) = [\![p\ t]\!]$, and $(w_1, o_l) = [\![leak\ (u_1, i)]\!]$. We now establish the following invariant: for any $m \in \{0, \ldots, M+1\}$, if

- there exists an $m' < m$ such that $\psi(m') = (s, i)$,

then we have

- $[\![Tick(m)[\mathsf{x} \coloneqq u_2, \mathsf{y} \coloneqq o_l]]\!] = (v_2, o)$.

To prove this invariant, we proceed by induction on $m$.

**Case 1.** If $m = 0$, then the assumption cannot be satisfied, so we are done.

**Case 2.** If $m = m' + 1$ and $\psi(m') = (s', i')$, let $(t', o') = [\![c\ (s', i')]\!]$, $(u_1', u_2') = [\![p\ s']\!]$, $(v_1', v_2') = [\![p\ t']\!]$, and $(w_1', o_l') = [\![leak\ (u_1', i')]\!]$. If $(u_2', o_l') \neq (u_2, o_l)$, then we have

$$\begin{aligned} & [\![Tick(m)[\mathsf{x} \coloneqq u_2, \mathsf{y} \coloneqq o_l]]\!] \\ &= \Big[\!\Big[\mathtt{let}\ (\mathsf{u}_1, \mathsf{u}_2) = p\ s'\ \mathtt{in}\ \mathtt{let}\ (\_, \mathsf{o}_1) = leak\ (\mathsf{u}_1, i')\ \mathtt{in} \\ &\quad \mathtt{if}\ (\mathsf{u}_2, \mathsf{o}_1) =_{(S_s, O_l)} (u_2, o_l) \\ &\qquad \mathtt{then}\ \mathtt{let}\ (\mathsf{t}, \mathsf{o}) = c\ (s', i')\ \mathtt{in}\ \mathtt{let}\ (\_, \mathsf{v}_2) = p\ \mathsf{t}\ \mathtt{in}\ (\mathsf{v}_2, \mathsf{o}) \\ &\qquad \mathtt{else}\ Tick(m')[\mathsf{x} \coloneqq u_2, \mathsf{y} \coloneqq o_l]\Big]\!\Big] \\ &= [\![Tick(m')[\mathsf{x} \coloneqq u_2, \mathsf{y} \coloneqq o_l]]\!]. \end{aligned}$$

The conclusion now follows directly from the induction hypothesis once we establish that the assumption also holds for $m'$ in place of $m$. From the corresponding assumption for $m$, we know that there exists an $m'' < m$ such that $\psi(m'') = (s, i)$. Now either $m'' < m'$ or $m'' = m'$. If $m'' < m'$, then we are done. If $m'' = m'$, then $\psi(m') = (s, i)$. From an earlier assumption, we know that $\psi(m') = (s', i')$. Thus $s' = s$, $i' = i$. But this means $u_1' = u_1$, $u_2' = u_2$ and $w_1' = w_1$, $o_l' = o_l$, which contradicts the assumption that $(u_2', o_l') \neq (u_2, o_l)$, showing that this case is impossible.

On the other hand, if $(u_2', o_l') = (u_2, o_l)$, then we have

$$
\begin{aligned}
&[\![\mathit{Tick}(m)[\mathsf{x} := u_2, \mathsf{y} := o_l]]\!] \\
&= \Big[\!\Big[\texttt{let } (\mathsf{u}_1, \mathsf{u}_2) = p\ s' \texttt{ in let } (\_, \mathsf{o}_1) = \mathit{leak}\ (\mathsf{u}_1, i') \texttt{ in} \\
&\quad \texttt{if } (\mathsf{u}_2, \mathsf{o}_1) =_{(S_s, O_l)} (u_2, o_l) \\
&\qquad \texttt{then let } (\mathsf{t}, \mathsf{o}) = c\ (s', i') \texttt{ in let } (\_, \mathsf{v}_2) = p\ \mathsf{t} \texttt{ in } (\mathsf{v}_2, \mathsf{o}) \\
&\qquad \texttt{else } \mathit{Tick}(m')[\mathsf{x} := u_2, \mathsf{y} := o_l]\Big]\!\Big] \\
&= (v_2', o').
\end{aligned}
$$

From condition (2.3), we know that if

$$
\begin{aligned}
&[\![\texttt{let } (\mathsf{u}_1, \mathsf{u}_2) = p\ s \texttt{ in let } (\_, \mathsf{o}_1) = \mathit{leak}\ (\mathsf{u}_1, i) \texttt{ in } (\mathsf{u}_2, \mathsf{o}_1)]\!] = \\
&[\![\texttt{let } (\mathsf{u}_1', \mathsf{u}_2') = p\ s' \texttt{ in let } (\_, \mathsf{o}_1') = \mathit{leak}\ (\mathsf{u}_1', i') \texttt{ in } (\mathsf{u}_2', \mathsf{o}_1')]\!]
\end{aligned}
$$

then we have

$$
\begin{aligned}
&[\![\texttt{let } (\mathsf{t}, \mathsf{o}) = c\ (s, i) \texttt{ in let } (\_, \mathsf{v}_2) = p\ \mathsf{t} \texttt{ in } (\mathsf{v}_2, \mathsf{o})]\!] = \\
&[\![\texttt{let } (\mathsf{t}', \mathsf{o}') = c\ (s', i') \texttt{ in let } (\_, \mathsf{v}_2') = p\ \mathsf{t}' \texttt{ in } (\mathsf{v}_2', \mathsf{o}')]\!].
\end{aligned}
$$

In other words, $(u_2, o_l) = (u_2', o_l')$ implies $(v_2', o') = (v_2, o)$. Since we assumed that $(u_2', o_l') = (u_2, o_l)$ in the first place, we have $(v_2', o') = (v_2, o)$, which finishes the proof of the invariant.

Using the invariant that we have just established, we conclude that

$$[\![\mathit{sim}\ (u_2, o_l)]\!] = [\![\mathit{Tick}(M+1)[\mathsf{x} := u_2, \mathsf{y} := o_l]]\!] = (v_2, o),$$

as the invariant assumption is clearly satisfied for $m = M + 1$. We thus have

$$[\![\mathit{leak} \circ \mathit{sim}\ ((u_1, u_2), i)]\!] = ((w_1, v_2), o).$$

The condition $[\![c_1 p\ (s, i)]\!] = [\![pc_2\ (s, i)]\!]$ now reduces to showing that $((v_1, v_2), o) = ((w_1, v_2), o)$. To show $v_1 = w_1$, we recall that from condition (2.2), we know that

$$
\begin{aligned}
&[\![\texttt{let } (\mathsf{t}, \_) = c\ (s, i) \texttt{ in let } (\mathsf{v}_1, \_) = p\ \mathsf{t} \texttt{ in } \mathsf{v}_1]\!] = \\
&[\![\texttt{let } (\mathsf{u}_1, \_) = p\ s \texttt{ in let } (\mathsf{v}_1, \_) = \mathit{leak}\ (\mathsf{u}_1, i) \texttt{ in } \mathsf{v}_1]\!].
\end{aligned}
$$

The above condition reduces precisely to $v_1 = w_1$, which finishes the proof. □

***Lemma* B.1.** *For circuits* $(c_1, s_1) : \big((S_1, I_1) \to (S_1, O_1), S_1\big)$ *and* $(c_2, s_2) : \big((S_2, O_1) \to (S_2, O_2), S_2\big)$, *and an input sequence is* $: [I]$, *we have*

$$\mathcal{M}_{(c_1, s_1) \circ (c_2, s_2)}(is) = \mathcal{M}_{(c_2, s_2)}\big(\mathcal{M}_{(c_1, s_1)}(is)\big).$$

## C COMPLETE PROCESSOR CODE FOR ??

**Fetch Stage.** The fetch stage (Listing 2) decodes the raw instruction and updates the program counter based on the previous instruction's execution. It stores the decoded instruction in the state's `exInstr` field and the current program counter (needed when executing a `Beq` instruction) in `exPC`; these become state inputs to execute stage in the next cycle. If a jump occured in the previous cycle (i.e., `jmp` is a `Just`-value), the instruction is discarded and is replaced with a no-op (`Add 0`). Otherwise, the program counter is incremented by 1.

**Case Study Processor.** The full processor (Listing 1) is composed of all the pipeline stages. The pipeline executes stages in reverse order—writeback, then execute, then fetch—to prevent overwriting stage state inputs before they are used.

```
proc :: (State, Word16) -> (State, (Maybe Word32, Word8))

proc (state0, rawInstr) = do
  let (state1, out) = writeback (state0, ())
  let (state2, jmp) = execute (state1', ())
  let (state3, _) = fetch (state2, (rawInstr, jmp))
  (state3, (out, pc state3))
```

Listing 1. The Pipelined Processor.

```
fetch :: (State, (Word16, Maybe Word8)) -> State
fetch (state@State { fePC }, (rawInstr, jump)) = do
  let (exInstr, fePC') = case jump of
        Just jmpPC -> (Add 0, jmpPC) -- Add 0 == no-op
        Nothing -> (decode rawInstr, fePC + 1)
  state { exPC = fePC , exInstr, fePC = fePC' }
```

Listing 2. Fetch Stage.

```
writeback :: (State, ()) -> (State, Maybe Word32)
writeback (state@State { reg, wbRes, wbOut }, _) = do
  let reg' = case wbRes of Just value -> value; Nothing -> reg
  (state { reg = reg' }, wbOut)
```

Listing 3. Writeback Stage.

```
sim :: (SState, LInstr) -> (SState, Word8)
sim (SState { fePC, exPC }, leakInstr) = do
  let fePC' = case leakInstr of
        LJmp addr -> addr
        LBr off -> exPC + off
        LOther -> fePC + 1
  (SState { exPC = fePC, fePC = fePC' }, fePC')
```

Listing 4. Simulator.

**Writeback Stage.** The writeback stage (Listing 3) updates the register value in the processor state and returns the output.

**Simulator.** The simulator executes the leakage instructions and outputs the current program counter. The leakage instructions only specify when they are branching, for the remaining instructions we simply increment the program counter.

## D EXAMPLE: STATE PROJECTION RULE

*Example (Counterexample to Completeness of State Projection).* Let the implementation circuit have trivial state, input, and output: $S = ()$, $I = ()$, $O = ()$.

```
impl :: ((), ()) -> ((), ())
```

```
impl (_, _) = ((), ())
```

Let the leakage circuit alternate between two internal states:

$$S_\ell = \{s_1, s_2\}, \quad I_\ell = (), \quad O_\ell = ().$$

```
leak :: (S_l, ())-> (S_l, ())
leak (s1, _)= (s2, ())
leak (s2, _)= (s1, ())
```

Any state projection $p : S \to S_\ell$ must fix one state: `p () = s1` and hence remains constant over time. However, `leak` switches between `s1` and `s2` on each step. Thus, no $p$ satisfies the state projection rule, even though their Mealy transducers are I/O-equivalent.

## E REPLICATED RESULTS FROM LEAVE.

| Benchmark Time (min.) | Replicated | Reported |
|---|---|---|
| RE | 0.5 | 1.5 |
| DarkRISCV-2 | 3.5 | 7.2 |
| DarkRISCV-3 | 10.0 | 11.1 |
| Sodor-2 | 31.4 | 97.8 |
| Ibex-small | 825.3 | 1479.4 |
| Ibex-cache | 1030.1 | 1396.7 |
| Ibex-mult-div | 673.4 | 1291.9 |

Table 1. Replicated runtimes from LeaVe.

Table 1 only presents the replicated benchmark results of LeaVe directly used for our comparison. We present the full set of results from LeaVe in Table 1.