Small step semantics for arithmetic expressions in Lean

Sonja Joost

In this document we will define small step rules for the datatype defining arithmetic expressions in `Test.lean`. After that, we will show that the `eval` function defined in Lean corresponds to the defined small step rules.

1 Rules

The datatype is defined as

$$e ::= \mathrm{const}(q) \mid \mathrm{var}(x) \mid \mathrm{add}(e_1, e_2) \mid \mathrm{sub}(e_1, e_2) \mid \mathrm{mul}(e_1, e_2) \mid \mathrm{div}(e_1, e_2), \qquad q \in \mathbb{Q},\ x \in \mathrm{String}.$$

In the following we abbreviate constants with values. A value is a constant: $v ::= \mathrm{const}(q)$.
An *environment* $\Gamma : \mathrm{Var} \rightharpoonup \mathbb{Q}$ maps variables to rationals. We write $\Gamma(x) = q$ if the variable x is bound to the value q and $\Gamma(x) = \bot$ if unbound.
Small-step now depends on Γ: $\Gamma \vdash e \to e'$. Evaluation order is left-to-right (eager evaluation).
Congruence (context) rules:

$$\frac{\Gamma \vdash e_1 \to e_1'}{\Gamma \vdash \mathrm{add}(e_1, e_2) \to \mathrm{add}(e_1', e_2)}\ \textsc{E-AddL} \qquad \frac{v \text{ value} \quad \Gamma \vdash e_2 \to e_2'}{\Gamma \vdash \mathrm{add}(v, e_2) \to \mathrm{add}(v, e_2')}\ \textsc{E-AddR}$$

$$\frac{\Gamma \vdash e_1 \to e_1'}{\Gamma \vdash \mathrm{sub}(e_1, e_2) \to \mathrm{sub}(e_1', e_2)}\ \textsc{E-SubL} \qquad \frac{v \text{ value} \quad \Gamma \vdash e_2 \to e_2'}{\Gamma \vdash \mathrm{sub}(v, e_2) \to \mathrm{sub}(v, e_2')}\ \textsc{E-SubR}$$

$$\frac{\Gamma \vdash e_1 \to e_1'}{\Gamma \vdash \mathrm{mul}(e_1, e_2) \to \mathrm{mul}(e_1', e_2)}\ \textsc{E-MulL} \qquad \frac{v \text{ value} \quad \Gamma \vdash e_2 \to e_2'}{\Gamma \vdash \mathrm{mul}(v, e_2) \to \mathrm{mul}(v, e_2')}\ \textsc{E-MulR}$$

$$\frac{\Gamma \vdash e_1 \to e_1'}{\Gamma \vdash \mathrm{div}(e_1, e_2) \to \mathrm{div}(e_1', e_2)}\ \textsc{E-DivL} \qquad \frac{v \text{ value} \quad \Gamma \vdash e_2 \to e_2'}{\Gamma \vdash \mathrm{div}(v, e_2) \to \mathrm{div}(v, e_2')}\ \textsc{E-DivR}$$

Lookup and computation rules:

$$\frac{\Gamma(x) = q}{\Gamma \vdash \mathrm{var}(x) \to \mathrm{const}(q)}\ \textsc{E-Var}$$

$$\frac{}{\Gamma \vdash \mathrm{add}(\mathrm{const}(q_1), \mathrm{const}(q_2)) \to \mathrm{const}(q_1 + q_2)}\ \textsc{E-Add}$$

$$\frac{}{\Gamma \vdash \mathrm{sub}(\mathrm{const}(q_1), \mathrm{const}(q_2)) \to \mathrm{const}(q_1 - q_2)}\ \textsc{E-Sub}$$

$$\frac{}{\Gamma \vdash \mathrm{mul}(\mathrm{const}(q_1), \mathrm{const}(q_2)) \to \mathrm{const}(q_1 \cdot q_2)}\ \textsc{E-Mul}$$

$$\frac{q_2 \neq 0}{\Gamma \vdash \mathrm{div}(\mathrm{const}(q_1), \mathrm{const}(q_2)) \to \mathrm{const}\left(\frac{q_1}{q_2}\right)}\ \textsc{E-Div}$$

No rule applies for division by zero or an unbound variable, so those forms are stuck. This corresponds to the evaluation function that would return none.
Write $\Gamma \vdash e \to^* e'$ for the reflexive–transitive closure.

2 Helper lemmas

Define $e \downarrow_\Gamma q :\iff \mathrm{eval}(e, \Gamma) = \mathrm{some}\ q$.

Lemma 1 (Canonical forms)**.** *If v is a value then $v = \mathrm{const}(q)$.*

Lemma 2 (Terminal Normal forms)**.** *If e is Γ-normal (no e' with $\Gamma \vdash e \to e'$) and a value or* cannot reach a compound expression through any steps*, then exactly one of:*

a) $e = \mathrm{const}(q)$*;*

b) $e = \mathrm{div}(\mathrm{const}(q_1), \mathrm{const}(0))$*;*

c) $e = \mathrm{var}(x)$ *with* $\Gamma(x) = \bot$.

These are the terminal *normal forms. Note that stuck expressions may also have compound structure (e.g.,* $\mathrm{add}(n, e)$ *where n is a terminal stuck form).*

Proof. We must show both directions: if e is normal then it's one of these forms, and conversely.
($\Rightarrow$) If e is Γ-normal then e is one of (a), (b), (c).
By structural induction on e:

Const: $e = \mathrm{const}(q)$. This is form (a).

Var: $e = \mathrm{var}(x)$. If $\Gamma(x) = q$, then E-Var applies, so e is not normal. Thus $\Gamma(x) = \bot$, giving form (c).

Add: $e = \mathrm{add}(e_1, e_2)$. For e to be normal:

- E-AddL requires e_1 to step, so e_1 must be normal.
- If e_1 is a value (i.e., $e_1 = \mathrm{const}(r)$), then E-AddR would apply if e_2 steps, so e_2 must be normal.
- If both $e_1 = \mathrm{const}(r_1)$ and $e_2 = \mathrm{const}(r_2)$, then E-Add applies, contradiction.

So we need e_1 to be a value and e_2 to be a normal non-value, or e_1 to be a normal non-value. By IH, normal forms are only (a), (b), (c). Forms (b) and (c) are not values. But if e has a compound structure like $\mathrm{add}(\ldots)$, it cannot match any of (a), (b), (c), which are all atomic. This is a contradiction. Therefore, e cannot have the form $\mathrm{add}(e_1, e_2)$ and be a normal form.

Sub/Mul: Similar reasoning: compound expressions cannot be normal forms.

Div: $e = \mathrm{div}(e_1, e_2)$. For e to be normal, e_1 cannot step (else E-DivL applies). If $e_1 = \mathrm{const}(r_1)$, then e_2 cannot step (else E-DivR applies). If $e_2 = \mathrm{const}(r_2)$ with $r_2 \neq 0$, then E-Div applies. So the only way $\mathrm{div}(\mathrm{const}(r_1), \mathrm{const}(r_2))$ is normal is if $r_2 = 0$, giving form (b). Any other structure for $\mathrm{div}(e_1, e_2)$ would be compound and not match (a), (b), or (c).

($\Leftarrow$) Each of (a), (b), (c) is Γ-normal.

- Form (a): $e = \mathrm{const}(q)$. No rule has a constant as the source of a step.
- Form (b): $e = \mathrm{div}(\mathrm{const}(q_1), \mathrm{const}(0))$. The only applicable rule would be E-Div, but it requires $q_2 \neq 0$, which fails.
- Form (c): $e = \mathrm{var}(x)$ with $\Gamma(x) = \bot$. E-Var requires $\Gamma(x) = q$, which fails.

Thus the three forms are exactly the normal forms. □

Auxiliary multi-step lemmas

We use the reflexive–transitive closure $\Gamma \vdash e \to^* e'$ (written in Lean as the inductive predicate 'Steps env e e''). The following lemmas mirror the Lean helper lemmas.

Lemma 3 (Concatenation (Lean: `Steps.append`).)**.** *If $\Gamma \vdash e \to^* e_1$ and $\Gamma \vdash e_1 \to^* e_2$ then $\Gamma \vdash e \to^* e_2$.*

Proof. Let $h_1 : \Gamma \vdash e \to^* e_1$ and $h_2 : \Gamma \vdash e_1 \to^* e_2$. We proceed by induction on h_2.
Base case: h_2 is the reflexive rule, so $e_1 = e_2$. Then $\Gamma \vdash e \to^* e_1 = e_2$ by h_1.
Inductive case: h_2 is of the form $\Gamma \vdash e_1 \to^* e'$ followed by a single step $\Gamma \vdash e' \to e_2$ for some intermediate expression e'.
By the induction hypothesis applied to h_1 and $\Gamma \vdash e_1 \to^* e'$, we obtain $\Gamma \vdash e \to^* e'$.
Now we apply the trans constructor to combine $\Gamma \vdash e \to^* e'$ with the single step $\Gamma \vdash e' \to e_2$ to get $\Gamma \vdash e \to^* e_2$. □

Lemma 4 (Context lifting (Lean: `Steps.liftCtx`).)**.** *Let $C[\_]$ be a one-hole context formed by adding a fixed surrounding constructor (e.g. $C[t] = \mathrm{add}(t, e_2)$). Suppose every single step lifts through C: whenever $\Gamma \vdash t \to t'$ then $\Gamma \vdash C[t] \to C[t']$. Then if $\Gamma \vdash e \to^* e'$ we have $\Gamma \vdash C[e] \to^* C[e']$.*

Proof. Let $h : \Gamma \vdash e \to^* e'$ and assume the lifting property: for all x, y, if $\Gamma \vdash x \to y$ then $\Gamma \vdash C[x] \to C[y]$.
We proceed by induction on h.
Base case: h is the reflexive rule, so $e = e'$. Then $C[e] = C[e']$ and $\Gamma \vdash C[e] \to^* C[e']$ by reflexivity.
Inductive case: h is of the form $\Gamma \vdash e \to^* e''$ followed by a single step $\Gamma \vdash e'' \to e'$ for some intermediate expression e''.
By the induction hypothesis applied to $\Gamma \vdash e \to^* e''$, we obtain $\Gamma \vdash C[e] \to^* C[e'']$.
By the lifting property applied to the single step $\Gamma \vdash e'' \to e'$, we obtain $\Gamma \vdash C[e''] \to C[e']$.
We apply the trans constructor to combine $\Gamma \vdash C[e] \to^* C[e'']$ with $\Gamma \vdash C[e''] \to C[e']$ to get $\Gamma \vdash C[e] \to^* C[e']$. □

Lemma 5 (Reduction lemmas (Lean: `Steps.reduceAdd`, `Steps.reduceSub`, `Steps.reduceMul`, `Steps.reduceDiv`.).)**.** *For each arithmetic operator we derive a multi-step reduction once its operands are reduced to constants:*

$$
\begin{aligned}
&\textit{Add: } \Gamma \vdash e_1 \to^* \mathrm{const}(r_1),\ \Gamma \vdash e_2 \to^* \mathrm{const}(r_2) \Rightarrow \Gamma \vdash \mathrm{add}(e_1, e_2) \to^* \mathrm{const}(r_1 + r_2).\\
&\textit{Sub: } \cdots \Rightarrow \Gamma \vdash \mathrm{sub}(e_1, e_2) \to^* \mathrm{const}(r_1 - r_2).\\
&\textit{Mul: } \cdots \Rightarrow \Gamma \vdash \mathrm{mul}(e_1, e_2) \to^* \mathrm{const}(r_1 \cdot r_2).\\
&\textit{Div: } r_2 \neq 0,\ \cdots \Rightarrow \Gamma \vdash \mathrm{div}(e_1, e_2) \to^* \mathrm{const}(r_1 / r_2).
\end{aligned}
$$

Proof (Add). Let $s_1 : \Gamma \vdash e_1 \to^* \mathrm{const}(r_1)$ and $s_2 : \Gamma \vdash e_2 \to^* \mathrm{const}(r_2)$.
We apply context lifting to s_1 using $C_L[t] = \mathrm{add}(t, e_2)$ with lifting property E-AddL to obtain:

$$L : \Gamma \vdash \mathrm{add}(e_1, e_2) \to^* \mathrm{add}(\mathrm{const}(r_1), e_2).$$

We apply context lifting to s_2 using $C_R[t] = \mathrm{add}(\mathrm{const}(r_1), t)$ with lifting property E-AddR to obtain:

$$R : \Gamma \vdash \mathrm{add}(\mathrm{const}(r_1), e_2) \to^* \mathrm{add}(\mathrm{const}(r_1), \mathrm{const}(r_2)).$$

By rule E-Add, we have $\Gamma \vdash \mathrm{add}(\mathrm{const}(r_1), \mathrm{const}(r_2)) \to \mathrm{const}(r_1 + r_2)$. We extend R with this step using trans to get:

$$R' : \Gamma \vdash \mathrm{add}(\mathrm{const}(r_1), e_2) \to^* \mathrm{const}(r_1 + r_2).$$

Applying concatenation to L and R' yields $\Gamma \vdash \mathrm{add}(e_1, e_2) \to^* \mathrm{const}(r_1 + r_2)$. □

Proof (Sub/Mul). Identical reasoning to addition using E-SubL/E-SubR/E-Sub and E-MulL/E-MulR/E-Mul respectively.

Proof (Div). Let $nz : r_2 \neq 0$, $s_1 : \Gamma \vdash e_1 \to^* \mathrm{const}(r_1)$, and $s_2 : \Gamma \vdash e_2 \to^* \mathrm{const}(r_2)$.
We apply context lifting to s_1 using $C_L[t] = \mathrm{div}(t, e_2)$ with lifting property E-DivL to obtain:

$$L : \Gamma \vdash \mathrm{div}(e_1, e_2) \to^* \mathrm{div}(\mathrm{const}(r_1), e_2).$$

We apply context lifting to s_2 using $C_R[t] = \mathrm{div}(\mathrm{const}(r_1), t)$ with lifting property E-DivR to obtain:

$$R : \Gamma \vdash \mathrm{div}(\mathrm{const}(r_1), e_2) \to^* \mathrm{div}(\mathrm{const}(r_1), \mathrm{const}(r_2)).$$

By rule E-Div with premise $r_2 \neq 0$, we have $\Gamma \vdash \mathrm{div}(\mathrm{const}(r_1), \mathrm{const}(r_2)) \to \mathrm{const}(r_1/r_2)$. We extend R with this step using trans to get:

$$R' : \Gamma \vdash \mathrm{div}(\mathrm{const}(r_1), e_2) \to^* \mathrm{const}(r_1/r_2).$$

Applying concatenation to L and R' yields $\Gamma \vdash \mathrm{div}(e_1, e_2) \to^* \mathrm{const}(r_1/r_2)$. $\square$

3 Proofs

Equivalence

To show that the evaluation function and the small step semantics agree, we have to show that

$$e \downarrow_\Gamma q \iff \Gamma \vdash e \to^* \mathrm{const}(q) \qquad \text{and} \qquad e \Downarrow_\Gamma \iff \Gamma \vdash e \to^* n \text{ stuck as above.}$$

Evaluation preservation

Lemma 6 (Single-step preserves evaluation (Lean: `step_preserves_eval`))**.** *If* $\Gamma \vdash e \to e'$ *then* $\mathrm{eval}(e, \Gamma) = \mathrm{eval}(e', \Gamma)$.

Proof. We do (structural) induction on $\Gamma \vdash e \to^* e'$. Cases:

E-Var. Premise: $\Gamma(x) = q$. Then

$$\begin{aligned}\mathrm{eval}(\mathrm{var}(x), \Gamma) &= \Gamma[x]? \\ &= \mathrm{some}\ q \\ &= \mathrm{eval}(\mathrm{const}(q), \Gamma).\end{aligned}$$

E-AddL. Premise: $\Gamma \vdash e_1 \to e_1'$ and IH gives $\mathrm{eval}(e_1, \Gamma) = \mathrm{eval}(e_1', \Gamma)$. Then

$$\begin{aligned}\mathrm{eval}(\mathrm{add}(e_1, e_2), \Gamma) &= \mathrm{do}\ (\leftarrow \mathrm{eval}(e_1, \Gamma)) + (\leftarrow \mathrm{eval}(e_2, \Gamma)) \\ &= \mathrm{do}\ (\leftarrow \mathrm{eval}(e_1', \Gamma)) + (\leftarrow \mathrm{eval}(e_2, \Gamma)) \\ &= \mathrm{eval}(\mathrm{add}(e_1', e_2), \Gamma).\end{aligned}$$

E-AddR. Premises: v value and $\Gamma \vdash e_2 \to e_2'$. By canonical forms for values, $v = \mathrm{const}(r)$, hence $\mathrm{eval}(v, \Gamma) = \mathrm{some}\ r$. IH: $\mathrm{eval}(e_2, \Gamma) = \mathrm{eval}(e_2', \Gamma)$. Then

$$\begin{aligned}\mathrm{eval}(\mathrm{add}(v, e_2), \Gamma) &= \mathrm{do}\ (\leftarrow \mathrm{eval}(v, \Gamma)) + (\leftarrow \mathrm{eval}(e_2, \Gamma)) \\ &= \mathrm{do}\ (\leftarrow \mathrm{some}\ r) + (\leftarrow \mathrm{eval}(e_2, \Gamma)) \\ &= \mathrm{do}\ r + (\leftarrow \mathrm{eval}(e_2, \Gamma)) \\ &= \mathrm{do}\ r + (\leftarrow \mathrm{eval}(e_2', \Gamma)) \\ &= \mathrm{eval}(\mathrm{add}(v, e_2'), \Gamma).\end{aligned}$$

E-Add. Both sides constants. Then

$$\begin{aligned}\mathrm{eval}(\mathrm{add}(\mathrm{const}(q_1), \mathrm{const}(q_2)), \Gamma) &= \mathrm{do}\ (\leftarrow \mathrm{some}\ q_1) + (\leftarrow \mathrm{some}\ q_2) \\ &= \mathrm{some}\ (q_1 + q_2) \\ &= \mathrm{eval}(\mathrm{const}(q_1 + q_2), \Gamma).\end{aligned}$$

E-SubL/E-SubR/E-Sub. Identical equational reasoning with $-$ in place of $+$.

E-MulL/E-MulR/E-Mul. Identical equational reasoning with $*$ in place of $+$.

E-DivL. Premise: $\Gamma \vdash e_1 \to e_1'$ and IH gives $\mathrm{eval}(e_1, \Gamma) = \mathrm{eval}(e_1', \Gamma)$. Then

$$\begin{aligned}\mathrm{eval}(\mathrm{div}(e_1, e_2), \Gamma) &= \mathrm{do}\ r_b \leftarrow \mathrm{eval}(e_2, \Gamma);\ \textbf{if}\ r_b = 0\ \textbf{then}\ \mathrm{none} \\ &\qquad \textbf{else}\ r_a \leftarrow \mathrm{eval}(e_1, \Gamma);\ \mathrm{some}\ (r_a / r_b) \\ &= \mathrm{do}\ r_b \leftarrow \mathrm{eval}(e_2, \Gamma);\ \textbf{if}\ r_b = 0\ \textbf{then}\ \mathrm{none} \\ &\qquad \textbf{else}\ r_a \leftarrow \mathrm{eval}(e_1', \Gamma);\ \mathrm{some}\ (r_a / r_b) \\ &= \mathrm{eval}(\mathrm{div}(e_1', e_2), \Gamma).\end{aligned}$$

E-DivR. Premises: v value so $v = \mathrm{const}(r)$ and $\mathrm{eval}(v, \Gamma) = \mathrm{some}\ r$, and $\Gamma \vdash e_2 \to e_2'$ with IH $\mathrm{eval}(e_2, \Gamma) = \mathrm{eval}(e_2', \Gamma)$. Then

$$\begin{aligned}\mathrm{eval}(\mathrm{div}(v, e_2), \Gamma) &= \mathrm{do}\ r_b \leftarrow \mathrm{eval}(e_2, \Gamma);\ \textbf{if}\ r_b = 0\ \textbf{then}\ \mathrm{none} \\ &\qquad \textbf{else}\ r_a \leftarrow \mathrm{eval}(v, \Gamma);\ \mathrm{some}\ (r_a / r_b) \\ &= \mathrm{do}\ r_b \leftarrow \mathrm{eval}(e_2', \Gamma);\ \textbf{if}\ r_b = 0\ \textbf{then}\ \mathrm{none} \\ &\qquad \textbf{else}\ r_a \leftarrow \mathrm{some}\ r;\ \mathrm{some}\ (r_a / r_b) \\ &= \mathrm{eval}(\mathrm{div}(v, e_2'), \Gamma).\end{aligned}$$

E-Div. Premise: $q_2 \neq 0$. Then

$$\begin{aligned}
\text{eval}(\text{div}(\text{const}(q_1), \text{const}(q_2)), \Gamma) &= \text{do } r_b \leftarrow \text{some } q_2; \\
&\quad \textbf{if } r_b = 0 \textbf{ then } \text{none} \ \ \textbf{else } r_a \leftarrow \text{some } q_1;\ \text{some } (r_a/r_b) \\
&= \text{some } (q_1/q_2) \quad (\text{since } q_2 \neq 0) \\
&= \text{eval}(\text{const}(q_1/q_2), \Gamma).
\end{aligned}$$

□

Lemma 7 (Multi-step preserves evaluation (Lean: `steps_preserve_eval`))**.** *If* $\Gamma \vdash e \to^* e'$ *then* $\text{eval}(e, \Gamma) = \text{eval}(e', \Gamma)$.

Proof. Induction on the length of the small step derivation. The reflexive case trivial. In the inductive case, by definition of $\to^*$ we have that $e \to^* e''$ and $e'' \to e$. Using the induction hypothesis on $e \to^* e''$ we have $\text{eval}(e, \Gamma) = \text{eval}(e'', \Gamma)$. By using Lemma 1 on $e'' \to e$ we have $\text{eval}(e'', \Gamma) = \text{eval}(e, \Gamma)$. If we combine both, we get $\text{eval}(e, \Gamma) = \text{eval}(e', \Gamma)$. □

Soundness

Theorem 1 (Soundness (Lean: `small_step_soundness`))**.** *If* $\Gamma \vdash e \to^* \text{const}(q)$ *then* $e \downarrow_\Gamma q$.

Proof. By multi-step preservation,

$$\begin{aligned}
\text{eval}(e, \Gamma) &= \text{eval}(\text{const}(q), \Gamma) \\
&= \text{some } q
\end{aligned}$$

by the first clause of the definition of `eval`. Hence $e \downarrow_\Gamma q$. □

Completeness

As a reminder for the upcoming proof, those are the definitions for the evaluation function from `Test.lean`:

$$
\begin{aligned}
&\mathrm{eval}(\mathrm{const}(q), \Gamma) = \mathrm{some}\ q,\\
&\mathrm{eval}(\mathrm{var}(x), \Gamma) = \Gamma[x]?,\\
&\mathrm{eval}(\mathrm{add}(a,b), \Gamma) = \mathrm{do}\ (\leftarrow \mathrm{eval}(a,\Gamma)) + (\leftarrow \mathrm{eval}(b,\Gamma)),\\
&\mathrm{eval}(\mathrm{sub}(a,b), \Gamma) = \mathrm{do}\ (\leftarrow \mathrm{eval}(a,\Gamma)) - (\leftarrow \mathrm{eval}(b,\Gamma)),\\
&\mathrm{eval}(\mathrm{mul}(a,b), \Gamma) = \mathrm{do}\ (\leftarrow \mathrm{eval}(a,\Gamma)) * (\leftarrow \mathrm{eval}(b,\Gamma)),\\
&\mathrm{eval}(\mathrm{div}(a,b), \Gamma) = \mathrm{do}\ r_b \leftarrow \mathrm{eval}(b,\Gamma);\ \textbf{if}\ r_b = 0\ \textbf{then}\ \mathrm{none}\ \textbf{else}\ r_a \leftarrow \mathrm{eval}(a,\Gamma);\ \mathrm{some}\ (r_a/r_b).
\end{aligned}
$$

Theorem 2 (Completeness (Lean: `eval_some_implies_steps_to_const`)). *If* $e \downarrow_\Gamma \mathrm{some}\, q$ *then* $\Gamma \vdash e \to^* \mathrm{const}(q)$.

Proof. By structural induction on e:

Const. We have $e = \mathrm{const}(r)$ for some $r \in \mathbb{Q}$. Then, $\mathrm{eval}(\mathrm{const}(r), \Gamma) = \mathrm{some}\ q$. By definition of eval, we have $r = q$. By taking zero steps we have $\Gamma \vdash \mathrm{const}(r) \to^* \mathrm{const}(q)$.

Var. We have $e = \mathrm{var}(x)$ for some $x \in \mathrm{String}$. Then, $\mathrm{eval}(\mathrm{var}(x), \Gamma) = \mathrm{some}\ q$. By definition of eval we have $\Gamma[x]? = \mathrm{some}\ q$ and thus $\Gamma(x) = q$. By applying E-Var once we get $\Gamma \vdash \mathrm{var}(x) \to^* \mathrm{const}\ q$.

Add. We have $e = \mathrm{add}(e_1, e_2)$.

Our assumption

$$\mathrm{eval}(\mathrm{add}(e_1, e_2), \Gamma) = \mathrm{some}\ q$$

unfolds to

$$\mathrm{do}\ (\leftarrow \mathrm{eval}(e_1,\Gamma)) + (\leftarrow \mathrm{eval}(e_2,\Gamma)) = \mathrm{some}\ q,$$

which is possible iff there exist $r_1, r_2 \in \mathbb{Q}$ with

$$\mathrm{eval}(e_1,\Gamma) = \mathrm{some}\ r_1, \quad \mathrm{eval}(e_2,\Gamma) = \mathrm{some}\ r_2, \quad q = r_1 + r_2.$$

We apply the induction hypothesis to both eval-expressions to obtain $\Gamma \vdash e_1 \to^* \mathrm{const}(r_1)$ and $\Gamma \vdash e_2 \to^* \mathrm{const}(r_2)$. We apply the reduction lemma to obtain $\Gamma \vdash \mathrm{add}(e_1, e_2) \to^* \mathrm{const}(r_1 + r_2)$.

Sub/Mul. Identical unpacking of the monadic definition with $-$ or $*$, yielding $q = r_1 - r_2$ or $q = r_1 * r_2$ by applying the respective reduction lemma.

Div. We have $e = \mathrm{div}(e_1, e_2)$.

Our assumption

$$\mathrm{eval}(\mathrm{div}(e_1, e_2), \Gamma) = \mathrm{some}\ q$$

unfolds to

$$\mathrm{do}\ r_2 \leftarrow \mathrm{eval}(e_2,\Gamma);\ \textbf{if}\ r_2 = 0\ \textbf{then}\ \mathrm{none}\ \textbf{else}\ r_1 \leftarrow \mathrm{eval}(e_1,\Gamma);\ \mathrm{some}\ (r_1/r_2) = \mathrm{some}\ q,$$

which is possible iff there exist r_1, r_2 with

$$\mathrm{eval}(e_2,\Gamma) = \mathrm{some}\ r_2,\ r_2 \neq 0,\ \mathrm{eval}(e_1,\Gamma) = \mathrm{some}\ r_1,\ q = r_1/r_2.$$

We apply our induction hypothesis to both evaluations to obtain $\Gamma \vdash e_1 \to^* \mathrm{const}(r_1)$ and $\Gamma \vdash e_2 \to^* \mathrm{const}(r_2)$, then the division reduction lemma (using $r_2 \neq 0$) yields $\mathrm{div}(e_1, e_2) \to^* \mathrm{const}(r_1/r_2)$.

□

Failure and stuckness

We say an expression e is *stuck* if it cannot take a step: there is no e' with $\Gamma \vdash e \to e'$. This includes both terminal stuck forms and compound expressions containing them (e.g., $\mathrm{add}(\mathrm{var}(x), e_2)$ where $\Gamma(x) = \bot$).
An expression e *reaches a terminal stuck form* iff $\Gamma \vdash e \to^* n$, where n is one of the two terminal stuck forms: $\mathrm{div}(\mathrm{const}(q_1), \mathrm{const}(0))$ or $\mathrm{var}(x)$ with $\Gamma(x) = \bot$.

Lemma 8. $\mathrm{eval}(e, \Gamma) =$ *none iff* $\Gamma \vdash e \to^* n$ *where* n *is stuck.*

Proof. We prove both directions.

(⇒) If $\mathrm{eval}(e, \Gamma) =$ **none then** $\Gamma \vdash e \to^* n$ **where** n **is stuck.** By structural induction on e.

Const. We have $e = \mathrm{const}(q)$. Then $\mathrm{eval}(\mathrm{const}(q), \Gamma) = \mathrm{some}\ q \neq \mathrm{none}$, contradiction.

Var. We have $e = \mathrm{var}(x)$. Then $\mathrm{eval}(\mathrm{var}(x), \Gamma) = \Gamma[x]? = \mathrm{none}$ means $\Gamma(x) = \bot$. We have $\Gamma \vdash e \to^* e$ by zero steps (reflexivity). The expression $\mathrm{var}(x)$ with $\Gamma(x) = \bot$ is stuck: no rule applies since E-Var requires $\Gamma(x) = q$ for some q.

Add. We have $e = \mathrm{add}(e_1, e_2)$. The monadic definition gives

$$\mathrm{eval}(\mathrm{add}(e_1, e_2), \Gamma) = \mathrm{do}\ (\leftarrow \mathrm{eval}(e_1, \Gamma)) + (\leftarrow \mathrm{eval}(e_2, \Gamma)) = \mathrm{none}.$$

This is possible iff $\mathrm{eval}(e_1, \Gamma) = \mathrm{none}$ or $\mathrm{eval}(e_2, \Gamma) = \mathrm{none}$ (or both).

Subcase $\mathrm{eval}(e_1, \Gamma) = \mathrm{none}$. By IH, $\Gamma \vdash e_1 \to^* n_1$ where n_1 is stuck. By context lifting (using E-AddL repeatedly), $\Gamma \vdash \mathrm{add}(e_1, e_2) \to^* \mathrm{add}(n_1, e_2)$. This compound expression is stuck: E-AddL cannot apply (since n_1 is stuck), and E-AddR cannot apply (since n_1 is not a value—it's either $\mathrm{div}(\mathrm{const}(q_1), \mathrm{const}(0))$ or $\mathrm{var}(x)$ with $\Gamma(x) = \bot$, neither of which is a value). E-Add cannot apply since n_1 is not a constant.

Subcase $\mathrm{eval}(e_1, \Gamma) = \mathrm{some}\ r_1$ and $\mathrm{eval}(e_2, \Gamma) = \mathrm{none}$. By completeness, $\Gamma \vdash e_1 \to^* \mathrm{const}(r_1)$. By IH on e_2, $\Gamma \vdash e_2 \to^* n_2$ where n_2 is stuck. Context lifting gives $\Gamma \vdash \mathrm{add}(e_1, e_2) \to^* \mathrm{add}(\mathrm{const}(r_1), e_2) \to^* \mathrm{add}(\mathrm{const}(r_1), n_2)$. This is stuck: E-Add requires both arguments to be constants, but n_2 is not a constant; E-AddR requires n_2 to step, but n_2 is stuck.

Sub/Mul. We have $e = \mathrm{sub}(e_1, e_2)$, $e = \mathrm{mul}(e_1, e_2)$. Identical reasoning to addition using E-SubL/E-SubR/E-Sub and E-MulL/E-MulR/E-Mul respectively.

Div. We have $e = \mathrm{div}(e_1, e_2)$. The definition is

$$\mathrm{eval}(\mathrm{div}(e_1, e_2), \Gamma) = \mathrm{do}\ r_2 \leftarrow \mathrm{eval}(e_2, \Gamma);\ \textbf{if}\ r_2 = 0\ \textbf{then}\ \mathrm{none}\ \textbf{else}\ r_1 \leftarrow \mathrm{eval}(e_1, \Gamma);\ \mathrm{some}\ (r_1/r_2).$$

This equals none iff:

- $\mathrm{eval}(e_2, \Gamma) = \mathrm{none}$, or
- $\mathrm{eval}(e_2, \Gamma) = \mathrm{some}\ r_2$ with $r_2 = 0$ and any outcome for e_1, or
- $\mathrm{eval}(e_2, \Gamma) = \mathrm{some}\ r_2$ with $r_2 \neq 0$ and $\mathrm{eval}(e_1, \Gamma) = \mathrm{none}$.

Subcase $\mathrm{eval}(e_2, \Gamma) = \mathrm{none}$. By IH, $\Gamma \vdash e_2 \to^* n_2$ stuck. Context lifting gives $\Gamma \vdash \mathrm{div}(e_1, e_2) \to^* \mathrm{div}(e_1, n_2)$, which is stuck by similar reasoning to addition.

Subcase $\mathrm{eval}(e_2, \Gamma) = \mathrm{some}\ 0$ and $\mathrm{eval}(e_1, \Gamma) = \mathrm{some}\ r_1$. By completeness, $\Gamma \vdash e_1 \to^* \mathrm{const}(r_1)$ and $\Gamma \vdash e_2 \to^* \mathrm{const}(0)$. Context lifting yields $\Gamma \vdash \mathrm{div}(e_1, e_2) \to^* \mathrm{div}(\mathrm{const}(r_1), \mathrm{const}(0))$. This is stuck: E-Div requires $q_2 \neq 0$, which fails here.

Subcase $\mathrm{eval}(e_2, \Gamma) = \mathrm{some}\ 0$ and $\mathrm{eval}(e_1, \Gamma) = \mathrm{none}$. By IH, $\Gamma \vdash e_1 \to^* n_1$ stuck. By completeness on e_2, $\Gamma \vdash e_2 \to^* \mathrm{const}(0)$. We reach $\Gamma \vdash \mathrm{div}(e_1, e_2) \to^* \mathrm{div}(n_1, \mathrm{const}(0))$, which is stuck.

Subcase $\mathrm{eval}(e_2, \Gamma) = \mathrm{some}\ r_2$ with $r_2 \neq 0$ and $\mathrm{eval}(e_1, \Gamma) = \mathrm{none}$. By completeness on e_2 and IH on e_1, we reach $\Gamma \vdash \mathrm{div}(e_1, e_2) \to^* \mathrm{div}(n_1, \mathrm{const}(r_2))$, which is stuck.

($\Leftarrow$) If $\Gamma \vdash e \to^* n$ where n is stuck and not a value, then $\text{eval}(e, \Gamma) =$ **none.** By multi-step preservation, $\text{eval}(e, \Gamma) = \text{eval}(n, \Gamma)$. We show $\text{eval}(n, \Gamma) = \text{none}$ by structural induction on n.
Terminal stuck forms:
Case $n = \text{var}(x)$ with $\Gamma(x) = \bot$. Then $\text{eval}(\text{var}(x), \Gamma) = \Gamma[x]? = \text{none}$ by definition.
Case $n = \text{div}(\text{const}(q_1), \text{const}(0))$. Then

$$\text{eval}(n, \Gamma) = \text{do } r_2 \leftarrow \text{some } 0;\ \textbf{if } r_2 = 0 \textbf{ then } \text{none} \textbf{ else } \cdots = \text{none}.$$

Compound stuck forms:
Case $n = \text{const}(q)$. Constants are values, contradicting the assumption that n is not a value.
Case $n = \text{add}(n_1, n_2)$ stuck. Since n is stuck, E-AddL cannot apply, so n_1 must be stuck. By IH, $\text{eval}(n_1, \Gamma) = \text{none}$. Then

$$\text{eval}(\text{add}(n_1, n_2), \Gamma) = \text{do } (\leftarrow \text{eval}(n_1, \Gamma)) + (\leftarrow \text{eval}(n_2, \Gamma)) = \text{do } (\leftarrow \text{none}) + \cdots = \text{none}.$$

Case $n = \text{sub}(n_1, n_2)$ or $n = \text{mul}(n_1, n_2)$ stuck. Identical reasoning: the left operand must be stuck, so evaluation returns none.
Case $n = \text{div}(n_1, n_2)$ stuck. Since n is stuck, either:

- n_2 is stuck (E-DivL fails), so by IH $\text{eval}(n_2, \Gamma) = \text{none}$, giving

  $$\text{eval}(\text{div}(n_1, n_2), \Gamma) = \text{do } r_2 \leftarrow \text{eval}(n_2, \Gamma); \cdots = \text{do } r_2 \leftarrow \text{none}; \cdots = \text{none}.$$

- $n_2 = \text{const}(0)$ and n_1 stuck or $n_1 = \text{const}(r_1)$. If n_1 stuck, then $\text{eval}(n_1, \Gamma) = \text{none}$ by IH, and evaluation fails. If $n_1 = \text{const}(r_1)$, then

  $$\text{eval}(\text{div}(\text{const}(r_1), \text{const}(0)), \Gamma) = \text{do } r_2 \leftarrow \text{some } 0;\ \textbf{if } r_2 = 0 \textbf{ then } \text{none} \cdots = \text{none}.$$

Thus every stuck non-value evaluates to none. $\square$