Small step semantics for arithmetic expressions in Lean

In this document we define small-step rules for the datatype in `math-expr.lean`. We then show that the `eval` function defined in Lean corresponds to these small-step rules.

1 Rules

The datatype is defined as

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

An *environment* $\Gamma : \text{Var} \rightharpoonup \mathbb{Q}$ 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. The small-step semantics now depend on Γ. We write $\Gamma \vdash e \to e'$ if an expression e steps (in one step) to e' in environment Γ.
We first define the rules that result in a constant:

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

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

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

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

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

In addition we need the context rules. Evaluation order is left-to-right (i.e., the right operand reduces only after the left operand becomes a constant).

$$\frac{\Gamma \vdash e_1 \to e_1'}{\Gamma \vdash \text{add}(e_1, e_2) \to \text{add}(e_1', e_2)}\ \text{E-AddL} \qquad \frac{\Gamma \vdash e_2 \to e_2'}{\Gamma \vdash \text{add}(\text{const}(q), e_2) \to \text{add}(\text{const}(q), e_2')}\ \text{E-AddR}$$

$$\frac{\Gamma \vdash e_1 \to e_1'}{\Gamma \vdash \text{sub}(e_1, e_2) \to \text{sub}(e_1', e_2)}\ \text{E-SubL} \qquad \frac{\Gamma \vdash e_2 \to e_2'}{\Gamma \vdash \text{sub}(\text{const}(q), e_2) \to \text{sub}(\text{const}(q), e_2')}\ \text{E-SubR}$$

$$\frac{\Gamma \vdash e_1 \to e_1'}{\Gamma \vdash \text{mul}(e_1, e_2) \to \text{mul}(e_1', e_2)}\ \text{E-MulL} \qquad \frac{\Gamma \vdash e_2 \to e_2'}{\Gamma \vdash \text{mul}(\text{const}(q), e_2) \to \text{mul}(\text{const}(q), e_2')}\ \text{E-MulR}$$

$$\frac{\Gamma \vdash e_1 \to e_1'}{\Gamma \vdash \text{div}(e_1, e_2) \to \text{div}(e_1', e_2)}\ \text{E-DivL} \qquad \frac{\Gamma \vdash e_2 \to e_2'}{\Gamma \vdash \text{div}(\text{const}(q), e_2) \to \text{div}(\text{const}(q), e_2')}\ \text{E-DivR}$$

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.
We write $\Gamma \vdash e \to^* e'$ for the reflexive-transitive closure of $\to$. Formally, $\Gamma \vdash e \to^* e'$ iff either $e = e'$ (reflexive) or there exists e'' with $\Gamma \vdash e \to^* e''$ and $\Gamma \vdash e'' \to e'$ (transitive).

2 Helper lemmas

Lemma 1 (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 $\Gamma \vdash e \to^* e'$. By definition of $\to^*$ we can 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 2 (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*: for all expressions* e_1, e_2 *whenever* $\Gamma \vdash e_1 \to e_2$ *then* $\Gamma \vdash C[e_1] \to C[e_2]$*. 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. 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 $\Gamma \vdash C[e''] \to C[e']$.
By definition of $\to^*$ we can 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 3 (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:*

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

Proof. We proceed left-to-right by lifting each operand's multi-step reduction through its evaluation context (Lemma 2) and then append the computation step using concatenation (Lemma 1).

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 the context lifting lemma (Lemma 2) 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 the context lifting again 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 by the definition of $\to^*$ to get:

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

Applying concatenation lemma (Lemma 1) to L and R' yields $\Gamma \vdash \mathrm{add}(e_1, e_2) \to^* \mathrm{const}(r_1 + r_2)$.

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

Div. Let $s_1 : \Gamma \vdash e_1 \to^* \text{const}(r_1)$ and $s_2 : \Gamma \vdash e_2 \to^* \text{const}(r_2)$ with $r_2 \neq 0$. Using context lifting with $C_L[t] = \text{div}(t, e_2)$ (E-DivL) we obtain

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

Then lifting s_2 with $C_R[t] = \text{div}(\text{const}(r_1), t)$ (E-DivR) yields

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

Finally apply E-Div (using $r_2 \neq 0$) to extend R, and concatenate with L to derive $\Gamma \vdash \text{div}(e_1, e_2) \to^* \text{const}(r_1/r_2)$. *Remark:* Our small-step semantics enforces a left-to-right discipline: we may only reduce the right operand of a division once the left has become a constant (rule E-DivR requires a constant on the left). In contrast, the Lean `eval` function inspects the denominator first to short-circuit on division by zero. These orders therefore differ operationally; a right-first reduction sequence is *not* derivable from the stated small-step rules. This does not affect extensional equivalence: whenever both operands evaluate successfully to nonzero constants, there exists a left-first small-step derivation to the same result; and if evaluation returns none (due to a stuck subexpression or zero denominator), the left-first semantics still reaches a stuck term. We rely only on existence of some derivation, not mirroring `eval`'s order.

□

3 Proofs

For the upcoming proofs, recall the definitions of the evaluation function from `math-expr.lean`:

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

Equivalence

We say an expression e is *stuck* iff it is not a value (not of the form $\text{const}(q)$ for any $q \in \mathbb{Q}$) and it cannot take a step: there is no e' with $\Gamma \vdash e \to e'$. This includes both terminal stuck forms and compound expressions that contain them (e.g., $\text{add}(\text{var}(x), e_2)$ where $\Gamma(x) = \bot$).
To show that the evaluation function and the small step semantics agree, we have to show that

$$\text{eval}(e, \Gamma) = \text{some}\, q \iff \Gamma \vdash e \to^* \text{const}(q)$$

and

$$\text{eval}(e, \Gamma) = \text{none} \iff \Gamma \vdash e \to^* n \text{ stuck as above.}$$

We next prove that single steps preserve the result of `eval`, and lift this to multi-step preservation. From multi-step preservation we can then conclude soundness (small-step reducing to a constant implies that the evaluation function evaluates to the same value). Then we establish completeness, showing any successful evaluation yields a multi-step reduction to the corresponding constant. Finally, we analyze the failure cases to complete the second equivalence, proving that evaluation yields none if and only if the small-step semantics reduces the expression to a stuck term.

Evaluation preservation

Lemma 4 (Single-step preserves evaluation (Lean: `step_preserves_eval`)**.** *If* $\Gamma \vdash e \to e'$ *then* $\text{eval}(e, \Gamma) = \text{eval}(e', \Gamma)$.

Proof. We do induction on $\Gamma \vdash e \to e'$. Cases:

E-Var. By definition of the rule we have $\Gamma(x) = q$. Then

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

E-AddL. By definition of the rule we have $\Gamma \vdash e_1 \to e_1'$. Applying the induction hypothesis, this gives $\text{eval}(e_1, \Gamma) = \text{eval}(e_1', \Gamma)$. Then

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

E-AddR. By definition of the rule we have that the first operand is of the form $\text{const}(q)$ for some q and that $\Gamma \vdash e_2 \to e_2'$. Applying the induction hypothesis, this gives $\text{eval}(e_2, \Gamma) = \text{eval}(e_2', \Gamma)$. Then

$$\begin{aligned} \text{eval}(\text{add}(\text{const}(q), e_2), \Gamma) &= \text{do } (\leftarrow \text{eval}(\text{const}(q), \Gamma)) + (\leftarrow \text{eval}(e_2, \Gamma)) && \text{(definition of eval)} \\ &= \text{do } (\leftarrow \text{eval}(\text{const}(q), \Gamma)) + (\leftarrow \text{eval}(e_2', \Gamma)) \\ & && (\text{eval}(e_2, \Gamma) = \text{eval}(e_2', \Gamma)) \\ &= \text{eval}(\text{add}(\text{const}(q), e_2'), \Gamma). && \text{(definition of eval)} \end{aligned}$$

E-Add. By definition of the rule we have that both operands are of the form $\text{const}(q)$ for some q. Then

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

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. By definition of the rule we have $\Gamma \vdash e_1 \to e_1'$ and applying the induction hypothesis gives $\text{eval}(e_1, \Gamma) = \text{eval}(e_1', \Gamma)$. Then

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

E-DivR. By definition of the rule we have that the first operand is of the form $\text{const}(q)$ for some q and that $\Gamma \vdash e_2 \to e_2'$. Applying the induction hypothesis gives $\text{eval}(e_2, \Gamma) = \text{eval}(e_2', \Gamma)$. Then

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

E-Div. By definition of the rule we have that both operands are of the form $\text{const}(q)$ for some q and that the second constant is nonzero. Then

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

Thus, case analysis over all possible derivation rules shows in every instance that $\text{eval}(e, \Gamma) = \text{eval}(e', \Gamma)$. Hence single-step reduction preserves evaluation, completing the proof. □

Lemma 5 (Multi-step preserves evaluation (Lean: `steps_preserve_eval`)). *If $\Gamma \vdash e \to^* e'$ then $\text{eval}(e, \Gamma) = \text{eval}(e', \Gamma)$.*

Proof. We proceed by induction on the derivation of $\Gamma \vdash e \to^* e'$.
Base case (reflexive): The derivation is the reflexive rule, so $e = e'$. Then $\text{eval}(e, \Gamma) = \text{eval}(e', \Gamma)$ holds trivially.
Inductive case (transitive): By definition of $\to^*$, the derivation has 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 $\text{eval}(e, \Gamma) = \text{eval}(e'', \Gamma)$.
By the single-step preservation lemma (Lemma 4) applied to $\Gamma \vdash e'' \to e'$, we obtain $\text{eval}(e'', \Gamma) = \text{eval}(e', \Gamma)$.
By transitivity of equality, combining these two results yields $\text{eval}(e, \Gamma) = \text{eval}(e', \Gamma)$.
Thus multi-step reduction preserves evaluation. □

Soundness

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

Proof. By multi-step preservation and the first clause of the definition of `eval`,

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

□

Completeness

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

Proof. By functional induction on the evaluation function:

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

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

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

Our assumption

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

unfolds to

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

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

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

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

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 $e = \text{div}(e_1, e_2)$.

Our assumption

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

unfolds (right operand evaluated first) to

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

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

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

We apply our induction hypothesis to both evaluations to obtain $\Gamma \vdash e_1 \to^* \text{const}(r_1)$ and $\Gamma \vdash e_2 \to^* \text{const}(r_2)$; the division reduction lemma (using $r_2 \neq 0$) yields $\Gamma \vdash \text{div}(e_1, e_2) \to^* \text{const}(r_1/r_2)$. *Remark:* Here we construct a left-first small-step derivation even though `eval` inspects e_2 first. Matching the evaluation order is unnecessary; the rules guarantee a deterministic left-first path whenever both sides succeed, and any failure detected early by `eval` corresponds to a stuck term reachable under left-first reduction.

□

Failure and stuckness

Theorem 3. $\text{eval}(e, \Gamma) = \textit{none iff } \Gamma \vdash e \to^* n \textit{ where } n \textit{ is stuck.}$

Proof. We prove both directions.

(⇒) If $\text{eval}(e, \Gamma) = \textbf{none then } \Gamma \vdash e \to^* n \textbf{ where } n \textbf{ is stuck.}$ By functional induction on the evaluation function.

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

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

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

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

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

Subcase $\text{eval}(e_1, \Gamma) = \text{none}$. By induction hypothesis, $\Gamma \vdash e_1 \to^* n_1$ where n_1 is stuck. By context lifting (using E-AddL repeatedly), $\Gamma \vdash \text{add}(e_1, e_2) \to^* \text{add}(n_1, e_2)$. Since n_1 is stuck and thus cannot be of the form $\text{const}(q)$ for any $q \in \mathbb{Q}$ both E-AddR and E-Add cannot apply. Since n_1 is stuck and thus $\nexists e'. \Gamma \vdash n_1 \to e'$, E-AddL also cannot apply.

Subcase $\text{eval}(e_1, \Gamma) = \text{some } r_1$ and $\text{eval}(e_2, \Gamma) = \text{none}$. By completeness (Theorem 2), $\Gamma \vdash e_1 \to^* \text{const}(r_1)$. By induction hypothesis on e_2, $\Gamma \vdash e_2 \to^* n_2$ where n_2 is stuck. Context lifting gives $\Gamma \vdash \text{add}(e_1, e_2) \to^* \text{add}(\text{const}(r_1), e_2) \to^* \text{add}(\text{const}(r_1), n_2)$. 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 $e = \text{sub}(e_1, e_2)$, $e = \text{mul}(e_1, e_2)$. Identical reasoning to addition using E-SubL/E-SubR/E-Sub and E-MulL/E-MulR/E-Mul respectively.

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

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

This equals none iff (noting right operand evaluated first for early zero / none detection):

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

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

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

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

Subcase $\text{eval}(e_2, \Gamma) = \text{some } r_2$ with $r_2 \neq 0$ and $\text{eval}(e_1, \Gamma) = \text{none}$. By completeness on e_2 and induction hypothesis on e_1, we reach $\Gamma \vdash \text{div}(e_1, e_2) \to^* \text{div}(n_1, \text{const}(r_2))$, which is stuck. *Note:* If e_1 is stuck already (non-constant with no step), evaluation would also yield none after inspecting e_2 (whether e_2 produces none or a nonzero value); the multi-step argument treats all branches uniformly by reducing each operand (when possible) before forming a stuck division term.

($\Leftarrow$) If $\Gamma \vdash e \to^* n$ where n is stuck, then $\text{eval}(e, \Gamma) = \textbf{none}$. By multi-step preservation, $\text{eval}(e, \Gamma) = \text{eval}(n, \Gamma)$. We show $\text{eval}(n, \Gamma) = \text{none}$ by functional induction on the evaluation function.

Const. We have $n = \text{const}(q)$. By definition they are not stuck.

Var. We have $n = \text{var}(x)$ with $\Gamma(x) = \bot$. Then $\text{eval}(\text{var}(x), \Gamma) = \Gamma[x]? = \text{none}$ by definition.

Add. We have $n = \text{add}(n_1, n_2)$ stuck. This occurs (under left-to-right rules) in exactly two mutually exclusive ways:

(a) n_1 is stuck (non-constant). Then by IH $\text{eval}(n_1, \Gamma) = \text{none}$ and

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

(b) $n_1 = \text{const}(r_1)$ and n_2 is stuck (non-constant). Then $\text{eval}(n_1, \Gamma) = \text{some } r_1$ while by IH $\text{eval}(n_2, \Gamma) = \text{none}$, so

$$\text{eval}(\text{add}(\text{const}(r_1), n_2), \Gamma) = \text{do } (\leftarrow \text{some } r_1) + (\leftarrow \text{none}) = \text{none}.$$

In either subcase the evaluation yields none.

Sub. $n = \text{sub}(n_1, n_2)$ stuck. Exactly analogous dichotomy:

(a) n_1 stuck non-constant $\Rightarrow \text{eval}(n_1, \Gamma) = \text{none}$ and the subtraction monad short-circuits to none.

(b) $n_1 = \text{const}(r_1)$ and n_2 stuck non-constant $\Rightarrow \text{eval}(n_2, \Gamma) = \text{none}$ and the whole evaluates to none.

Mul. $n = \text{mul}(n_1, n_2)$ stuck. Same two cases:

(a) n_1 stuck non-constant $\Rightarrow$ left evaluation none, product none.

(b) $n_1 = \text{const}(r_1)$ and n_2 stuck non-constant $\Rightarrow$ right evaluation none, product none.

Div. We have $n = \text{div}(n_1, n_2)$ stuck. For n to be stuck we need E-DivL, E-DivR, and E-Div all to fail. E-DivL fails means n_1 cannot step, so either n_1 is stuck (and not a constant) or $n_1 = \text{const}(r_1)$ for some r_1.

Case 1: n_1 is stuck (not a constant). Then by induction hypothesis $\text{eval}(n_1, \Gamma) = \text{none}$, giving

$$\text{eval}(\text{div}(n_1, n_2), \Gamma) = \text{do } r_2 \leftarrow \text{eval}(n_2, \Gamma);\ \textbf{if } r_2 = 0 \textbf{ then none else } r_1 \leftarrow \text{none}; \cdots = \text{none}.$$

Case 2: $n_1 = \text{const}(r_1)$. Now E-DivR and E-Div must also fail. E-DivR fails means n_2 cannot step, so either n_2 is stuck or $n_2 = \text{const}(r_2)$.

- If n_2 is stuck (not a constant), then by induction hypothesis $\text{eval}(n_2, \Gamma) = \text{none}$, giving

$$\text{eval}(\text{div}(\text{const}(r_1), n_2), \Gamma) = \text{do } r_2 \leftarrow \text{none}; \cdots = \text{none}.$$

- If $n_2 = \text{const}(r_2)$, then for E-Div to fail we need $r_2 = 0$. Then

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

Thus every stuck (non-value) expression evaluates to none. □

Theorem 4 (Equivalence Summary)**.** *For all environments Γ and expressions e: By Theorem 1 and 2:* $\text{eval}(e, \Gamma) = \text{some } q \iff \Gamma \vdash e \to^* \text{const}(q)$ *and by Theorem 3:* $\text{eval}(e, \Gamma) = \text{none} \iff \exists n.\ \Gamma \vdash e \to^* n$ *and n is stuck.*