

Small step semantics for arithmetic expressions in Lean

In this document we will define small step rules for the datatype defined in `math-expr.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

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

An *environment* $\Gamma : \text{Var} \rightarrow \mathbb{Q}$ maps variables to rationals. We write $\Gamma(x) = q$ if the variable x is bound to the value q and $\Gamma(x) = \perp$ if unbound. The small-step semantics now depend on Γ . We write $\Gamma \vdash e \rightarrow e'$ if an expression e steps (in one step) to e' in environment Γ .

We first define the rules that result in a constant:

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

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

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

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

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

In addition we need the context rules. Evaluation order is left-to-right.

$$\frac{\Gamma \vdash e_1 \rightarrow e'_1}{\Gamma \vdash \text{add}(e_1, e_2) \rightarrow \text{add}(e'_1, e_2)} \text{E-ADDL}$$

$$\frac{\Gamma \vdash e_2 \rightarrow e'_2}{\Gamma \vdash \text{add}(\text{const}(q), e_2) \rightarrow \text{add}(\text{const}(q), e'_2)} \text{E-ADDR}$$

$$\frac{\Gamma \vdash e_1 \rightarrow e'_1}{\Gamma \vdash \text{sub}(e_1, e_2) \rightarrow \text{sub}(e'_1, e_2)} \text{E-SUBL}$$

$$\frac{\Gamma \vdash e_2 \rightarrow e'_2}{\Gamma \vdash \text{sub}(\text{const}(q), e_2) \rightarrow \text{sub}(\text{const}(q), e'_2)} \text{E-SUBR}$$

$$\frac{\Gamma \vdash e_1 \rightarrow e'_1}{\Gamma \vdash \text{mul}(e_1, e_2) \rightarrow \text{mul}(e'_1, e_2)} \text{E-MULL}$$

$$\frac{\Gamma \vdash e_2 \rightarrow e'_2}{\Gamma \vdash \text{mul}(\text{const}(q), e_2) \rightarrow \text{mul}(\text{const}(q), e'_2)} \text{E-MULR}$$

$$\frac{\Gamma \vdash e_1 \rightarrow e'_1}{\Gamma \vdash \text{div}(e_1, e_2) \rightarrow \text{div}(e'_1, e_2)} \text{E-DIVL}$$

$$\frac{\Gamma \vdash e_2 \rightarrow e'_2}{\Gamma \vdash \text{div}(\text{const}(q), e_2) \rightarrow \text{div}(\text{const}(q), e'_2)} \text{E-DIVR}$$

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 $\Gamma \vdash e \rightarrow^* e'$ for the reflexive-transitive closure of \rightarrow . To be more exact, $\Gamma \vdash e \rightarrow^* e'$ if either $e = e'$ (the reflexive case) or there exists an e'' such that $\Gamma \vdash e \rightarrow^* e''$ and $\Gamma \vdash e'' \rightarrow e'$ (the transitive case).

2 Helper lemmas

Lemma 1 (Concatenation, Lean: `Steps.append`). *If $\Gamma \vdash e \rightarrow^* e_1$ and $\Gamma \vdash e_1 \rightarrow^* e_2$ then $\Gamma \vdash e \rightarrow^* e_2$.*

Proof. Let $h_1 : \Gamma \vdash e \rightarrow^* e_1$ and $h_2 : \Gamma \vdash e_1 \rightarrow^* e_2$. We proceed by induction on h_2 .

Base case: h_2 is the reflexive rule, so $e_1 = e_2$. Then $\Gamma \vdash e \rightarrow^* e_1 = e_2$ by h_1 .

Inductive case: h_2 is of the form $\Gamma \vdash e_1 \rightarrow^* e'$ followed by a single step $\Gamma \vdash e' \rightarrow e_2$ for some intermediate expression e' .

By the induction hypothesis applied to h_1 and $\Gamma \vdash e_1 \rightarrow^* e'$, we obtain $\Gamma \vdash e \rightarrow^* e'$. By definition of \rightarrow^* we can combine $\Gamma \vdash e \rightarrow^* e'$ with the single step $\Gamma \vdash e' \rightarrow e_2$ to get $\Gamma \vdash e \rightarrow^* e_2$. \square

Lemma 2 (Context lifting, Lean: `Steps.liftCtx`). *Let $C[_]$ be a one-hole context formed by adding a fixed surrounding constructor (e.g. $C[t] = \text{add}(t, e_2)$). Suppose every single step lifts through C : for all expressions e_1, e_2 whenever $\Gamma \vdash e_1 \rightarrow e_2$ then $\Gamma \vdash C[e_1] \rightarrow C[e_2]$. Then if $\Gamma \vdash e \rightarrow^* e'$ we have $\Gamma \vdash C[e] \rightarrow^* C[e']$.*

Proof. Let $h : \Gamma \vdash e \rightarrow^* e'$ 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 $\Gamma \vdash C[e] \rightarrow^* C[e']$ by reflexivity.

Inductive case: h is of the form $\Gamma \vdash e \rightarrow^* e''$ followed by a single step $\Gamma \vdash e'' \rightarrow e'$ for some intermediate expression e'' .

By the induction hypothesis applied to $\Gamma \vdash e \rightarrow^* e''$, we obtain $\Gamma \vdash C[e] \rightarrow^* C[e'']$.

By the lifting property applied to the single step $\Gamma \vdash e'' \rightarrow e'$, we obtain $\Gamma \vdash C[e''] \rightarrow C[e']$.

By definition of \rightarrow^* we can combine $\Gamma \vdash C[e] \rightarrow^* C[e'']$ with $\Gamma \vdash C[e''] \rightarrow C[e']$ to get $\Gamma \vdash C[e] \rightarrow^* C[e']$. \square

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: $\Gamma \vdash e_1 \rightarrow^* \text{const}(r_1), \Gamma \vdash e_2 \rightarrow^* \text{const}(r_2) \Rightarrow \Gamma \vdash \text{add}(e_1, e_2) \rightarrow^* \text{const}(r_1 + r_2)$.

Sub: $\Gamma \vdash e_1 \rightarrow^* \text{const}(r_1), \Gamma \vdash e_2 \rightarrow^* \text{const}(r_2) \Rightarrow \Gamma \vdash \text{sub}(e_1, e_2) \rightarrow^* \text{const}(r_1 - r_2)$.

Mul: $\Gamma \vdash e_1 \rightarrow^* \text{const}(r_1), \Gamma \vdash e_2 \rightarrow^* \text{const}(r_2) \Rightarrow \Gamma \vdash \text{mul}(e_1, e_2) \rightarrow^* \text{const}(r_1 \cdot r_2)$.

Div: $r_2 \neq 0, \Gamma \vdash e_1 \rightarrow^* \text{const}(r_1), \Gamma \vdash e_2 \rightarrow^* \text{const}(r_2) \Rightarrow \Gamma \vdash \text{div}(e_1, e_2) \rightarrow^* \text{const}(r_1/r_2)$.

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 $s_1 : \Gamma \vdash e_1 \rightarrow^* \text{const}(r_1)$ and $s_2 : \Gamma \vdash e_2 \rightarrow^* \text{const}(r_2)$.

We apply the context lifting lemma (Lemma 2) to s_1 using $C_L[t] = \text{add}(t, e_2)$ with lifting property E-AddL to obtain:

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

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

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

By rule E-Add, we have $\Gamma \vdash \text{add}(\text{const}(r_1), \text{const}(r_2)) \rightarrow \text{const}(r_1 + r_2)$. We extend R with this step by the definition of \rightarrow^* to get:

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

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

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

Div. Identical reasoning to addition using E-DivL/E-DivR, with the premise $r_2 \neq 0$ required to apply the E-Div computation rule in the final step. \square

3 Proofs

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

```

eval(const(q), Γ) = some q,
eval(var(x), Γ) = Γ[x]?,
eval(add(a, b), Γ) = do (← eval(a, Γ)) + (← eval(b, Γ)),
eval(sub(a, b), Γ) = do (← eval(a, Γ)) - (← eval(b, Γ)),
eval(mul(a, b), Γ) = do (← eval(a, Γ)) * (← eval(b, Γ)),
eval(div(a, b), Γ) = do r_b ← eval(b, Γ); if r_b = 0 then none else r_a ← eval(a, Γ); some (r_a/r_b).

```

Equivalence

We say an expression e is *stuck* if it is not of the form $\text{const}(q)$ for some $q \in \mathbb{Q}$ and it cannot take a step: there is no e' with $\Gamma \vdash e \rightarrow e'$. This includes both terminal stuck forms and compound expressions containing them (e.g., $\text{add}(\text{var}(x), e_2)$ where $\Gamma(x) = \perp$).

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

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

and

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

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 $\Gamma \vdash e \rightarrow e'$ then $\text{eval}(e, \Gamma) = \text{eval}(e', \Gamma)$.*

Proof. We do induction on $\Gamma \vdash e \rightarrow e'$. 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 $\Gamma \vdash e_1 \rightarrow e'_1$. 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 $\Gamma \vdash e_2 \rightarrow e'_2$. 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 $\Gamma \vdash e_1 \rightarrow e'_1$ and applying the induction hypothesis gives $\text{eval}(e_1, \Gamma) = \text{eval}(e'_1, \Gamma)$. Then

$$\begin{aligned} \text{eval}(\text{div}(e_1, e_2), \Gamma) &= \text{do } r_b \leftarrow \text{eval}(e_2, \Gamma); \text{ if } r_b = 0 \text{ then none} && \text{(definition of eval)} \\ &\quad \text{else } r_a \leftarrow \text{eval}(e_1, \Gamma); \text{ some } (r_a/r_b) \\ &= \text{do } r_b \leftarrow \text{eval}(e_2, \Gamma); \text{ if } r_b = 0 \text{ then none} \\ &\quad \text{else } r_a \leftarrow \text{eval}(e'_1, \Gamma); \text{ some } (r_a/r_b) && (\text{eval}(e_1, \Gamma) = \text{eval}(e'_1, \Gamma)) \\ &= \text{eval}(\text{div}(e'_1, e_2), \Gamma). && \text{(definition of eval)} \end{aligned}$$

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 $\Gamma \vdash e_2 \rightarrow e'_2$. Applying the induction hypothesis gives $\text{eval}(e_2, \Gamma) = \text{eval}(e'_2, \Gamma)$. Then

$$\begin{aligned} \text{eval}(\text{div}(\text{const}(q), e_2), \Gamma) &= \text{do } r_b \leftarrow \text{eval}(e_2, \Gamma); \text{ if } r_b = 0 \text{ then none} \\ &\quad \text{else } r_a \leftarrow \text{eval}(\text{const}(q), \Gamma); \text{ some } (r_a/r_b) && \text{(definition of eval)} \\ &= \text{do } r_b \leftarrow \text{eval}(e'_2, \Gamma); \text{ if } r_b = 0 \text{ then none} \\ &\quad \text{else } r_a \leftarrow \text{eval}(\text{const}(q), \Gamma); \text{ some } (r_a/r_b) && (\text{eval}(e_2, \Gamma) = \text{eval}(e'_2, \Gamma)) \\ &= \text{eval}(\text{div}(\text{const}(q), e'_2), \Gamma). && \text{(definition of eval)} \end{aligned}$$

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

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

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. \square

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

Proof. We proceed by induction on the derivation of $\Gamma \vdash e \rightarrow^* e'$.

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 \rightarrow^* , the derivation has the form $\Gamma \vdash e \rightarrow^* e''$ followed by a single step $\Gamma \vdash e'' \rightarrow e'$ for some intermediate expression e'' .

By the induction hypothesis applied to $\Gamma \vdash e \rightarrow^* e''$, we obtain $\text{eval}(e, \Gamma) = \text{eval}(e'', \Gamma)$.

By the single-step preservation lemma (Lemma 4) applied to $\Gamma \vdash e'' \rightarrow e'$, 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. \square

Soundness

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

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 $\Gamma \vdash \text{const}(r) \rightarrow^* \text{const}(q)$.

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 $\Gamma(x) = q$. By applying E-Var once we get $\Gamma \vdash \text{var}(x) \rightarrow^* \text{const } q$.

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

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 to

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

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

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

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

□

Failure and stuckness

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

Proof. We prove both directions.

(\Rightarrow) **If $\text{eval}(e, \Gamma) = \text{none}$ then $\Gamma \vdash e \rightarrow^* n$ where n 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 $\Gamma(x) = \perp$. We have $\Gamma \vdash e \rightarrow^* e$ by zero steps (reflexivity). The expression $\text{var}(x)$ with $\Gamma(x) = \perp$ 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, $\Gamma \vdash e_1 \rightarrow^* n_1$ where n_1 is stuck. By context lifting (using E-AddL repeatedly), $\Gamma \vdash \text{add}(e_1, e_2) \rightarrow^* \text{add}(n_1, e_2)$. 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 $\nexists e'. \Gamma \vdash n_1 \rightarrow e'$, 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), $\Gamma \vdash e_1 \rightarrow^* \text{const}(r_1)$. By induction hypothesis on e_2 , $\Gamma \vdash e_2 \rightarrow^* n_2$ where n_2 is stuck. Context lifting gives $\Gamma \vdash \text{add}(e_1, e_2) \rightarrow^* \text{add}(\text{const}(r_1), e_2) \rightarrow^* \text{add}(\text{const}(r_1), n_2)$. 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

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

This equals none iff:

- $\text{eval}(e_2, \Gamma) = \text{none}$, 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, $\Gamma \vdash e_2 \rightarrow^* n_2$ stuck. Context lifting gives $\Gamma \vdash \text{div}(e_1, e_2) \rightarrow^* \text{div}(e_1, n_2)$, 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, $\Gamma \vdash e_1 \rightarrow^* \text{const}(r_1)$ and $\Gamma \vdash e_2 \rightarrow^* \text{const}(0)$. Context lifting yields $\Gamma \vdash \text{div}(e_1, e_2) \rightarrow^* \text{div}(\text{const}(r_1), \text{const}(0))$. 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, $\Gamma \vdash e_1 \rightarrow^* n_1$ stuck. By completeness on e_2 , $\Gamma \vdash e_2 \rightarrow^* \text{const}(0)$. We reach $\Gamma \vdash \text{div}(e_1, e_2) \rightarrow^* \text{div}(n_1, \text{const}(0))$, 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 $\Gamma \vdash \text{div}(e_1, e_2) \rightarrow^* \text{div}(n_1, \text{const}(r_2))$, which is stuck.

(\Leftarrow) **If $\Gamma \vdash e \rightarrow^* n$ where n is stuck, then $\text{eval}(e, \Gamma) = \text{none}$.** 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 $\Gamma(x) = \perp$. Then $\text{eval}(\text{var}(x), \Gamma) = \Gamma[x]? = \text{none}$ by definition.

Add. We have $n = \text{add}(n_1, n_2)$ stuck. Since n is stuck, E-AddL cannot apply, so n_1 must be stuck. By induction hypothesis, $\text{eval}(n_1, \Gamma) = \text{none}$. Then

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

Sub. / Mul. We have either $n = \text{sub}(n_1, n_2)$ or $n = \text{mul}(n_1, n_2)$ stuck. Identical reasoning: the left operand must be stuck, so evaluation returns none.

Div. We have $n = \text{div}(n_1, n_2)$ stuck. Since evaluation is left-to-right, 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

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

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

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

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

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

Thus every stuck expression evaluates to none. □