Progress and Preservation Proofs for the Expressions "`iszero`" and "`pred`" in the Arith Language

Zayd Hammoudeh
(zayd.hammoudeh@sjsu.edu)

May 22, 2016

Contents

List of Figures

1 Arith Language

Arith is a basic language; its expressions, values, and types are enumerated in figure 1. Arith's small-step, evaluation order semantics are defined in figure 2, and Arith's type rules are enumerated in figure 3.

$e ::=$		*Expressions*
	`true`	Boolean True
	`false`	Boolean False
	i	Integer Value
	`succ` (e)	Successor Expressions
	`pred` (e)	Predecessor Expressions
	`iszero` (e)	Zero Value Check Expressions
	`if` (e) `then` (e) `else` (e)	Conditional Expressions
$v ::=$		*Values*
	i	integer values
	b	boolean values
$T ::=$		*Types*
	`Bool`	Boolean Type
	`Int`	Integer Type

Figure 1: The Arith language

Evaluation Rules: $\boxed{e \to e'}$

[E-Succ-Ctxt] $$\frac{e_1 \to e_1'}{\texttt{succ}\ (e_1) \to \texttt{succ}\ (e_1')}$$

[E-Succ] $$\frac{i' = i + 1}{\texttt{succ}\ (i) \to i'}$$

[E-Pred-Ctxt] $$\frac{e_1 \to e_1'}{\texttt{pred}\ (e_1) \to \texttt{pred}\ (e_1')}$$

[E-Pred] $$\frac{i' = i - 1}{\texttt{pred}\ (i) \to i'}$$

[E-IsZero-Ctxt] $$\frac{e_1 \to e_1'}{\texttt{iszero}\ (e_1) \to \texttt{iszero}\ (e_1')}$$

[E-IsZero-Z] $$\texttt{iszero}\ (0) \to \texttt{true}$$

[E-IsZero-NZ] $$\frac{i \neq 0}{\texttt{iszero}\ (i) \to \texttt{false}}$$

[E-If-Ctxt] $$\frac{e_1 \to e_1'}{\texttt{if}\ (e_1)\ \texttt{then}\ (e_2)\ \texttt{else}\ (e_3) \to \texttt{if}\ (e_1')\ \texttt{then}\ (e_2)\ \texttt{else}\ (e_3)}$$

[E-If-True] $$\texttt{if}\ (\texttt{true})\ \texttt{then}\ (e_2)\ \texttt{else}\ (e_3) \to e_2$$

[E-If-False] $$\texttt{if}\ (\texttt{false})\ \texttt{then}\ (e_2)\ \texttt{else}\ (e_3) \to e_3$$

Figure 2: Small-Step, Evaluation Order Semantics Rules for the Arith Language

Type Rules: $\boxed{e : T}$

$$[\textsc{T-True}] \qquad \texttt{true} : \texttt{Bool}$$

$$[\textsc{T-False}] \qquad \texttt{false} : \texttt{Bool}$$

$$[\textsc{T-Int}] \qquad i : \texttt{Int}$$

$$[\textsc{T-Succ}] \qquad \frac{e_1 : \texttt{Int}}{\texttt{succ}\ (e_1) : \texttt{Int}}$$

$$[\textsc{T-Pred}] \qquad \frac{e_1 : \texttt{Int}}{\texttt{pred}\ (e_1) : \texttt{Int}}$$

$$[\textsc{T-IsZero}] \qquad \frac{e_1 : \texttt{Int}}{\texttt{iszero}\ (e_1) : \texttt{Bool}}$$

$$[\textsc{T-If}] \qquad \frac{e_1 : \texttt{Bool}, \quad e_2 : T, \quad e_3 : T}{\texttt{if}\ (e_1)\ \texttt{then}\ (e_2)\ \texttt{else}\ (e_3) : T}$$

Figure 3: Type Rules for the Arith Language

2 Progress in the Arith Language

In a type system, "progress" entails that a well-type expression will not "get stuck." Figure 4 is the formal, theoretical definition of progress.

Given $e : T$, then either:

1. e is a value.
2. There exists an e' such that: $e \to e'$.

Figure 4: Formal Definition of the Progress Theorem

The following subsections are the formal progress proofs for the type rules in figure 3.

2.1 Proving Progress for Boolean and Integer Values

Figure 4 establishes that progress is achieved if an expression "e" is a value. Hence, by this criterion, type rules [T-True], [T-False], and [T-Int] are all valid for the progress theorem.

2.2 Proving Progress for the `if` Expression

Figure 5 is the proof of progress for the `if` expression in the Arith Language.

Given:

$$e = \texttt{if } (e_1) \texttt{ then } (e_2) \texttt{ else } (e_3)$$
$$e_1 : \texttt{Bool}, \quad e_2 : T, \quad e_3 : T$$

Then:

$$e : T$$

By induction, an expression e_1 must be a value or $\exists e_1$ such that:

1. If e_1 is a value, then either [E-If-True] or [E-If-False] applies since e_1 : `Bool`. Hence, an expression e' exists for both cases.
2. Otherwise, $e_1 \to e_1'$ which means that [E-If-Ctxt] applies. This satisfies by induction the progress theorem.

Figure 5: Proof of Progress for the `if` Expression

2.3 Proving Progress for the `succ` Expression

Figure 3 shows the type rule for the `succ` expression. The proof of progress for this expression is shown in figure 6.

Given:

$$e = \texttt{succ}\ (e_1)$$
$$e_1 : \texttt{Int}$$

Then:

$$T = \texttt{Int}$$

By induction, e_1 must either be a value or $\exists e_1'$ such that:

1. If e_1 is a value (i.e. "i"), then: $e \rightarrow i'$ where $i' = i + 1$ using the small-step rule [E-SUCC] since it was given that $e_1 : \texttt{Int}$.
2. Otherwise, $e_1 \rightarrow e_1'$ in which case [E-SUCC-CTXT] applies (as shown below). Hence, progress holds by induction.

$$e \rightarrow \texttt{succ}\ (e_1')$$

Figure 6: Proof of Progress for the `succ` Expression

2.4 Proving Progress for the `pred` Expression

Figure 3 shows the type rule for the `pred` expression. The proof of progress for this expression is shown in figure 7.

2.5 Proving Progress for the `iszero` Expression

Figure 3 shows the type rule for the `iszero` expression. The proof of progress for this expression is shown in figure 8.

Given:

$$e = \texttt{pred}\ (e_1)$$
$$e_1 : \texttt{Int}$$

Then:

$$T = \texttt{Int}$$

By induction, e_1 must either be a value or $\exists e_1'$ such that:

1. If e_1 is a value (i.e. "i"), then: $e \to i'$ where $i' = i - 1$ using the small-step rule [E-Pred] since it was given that $e_1 : \texttt{Int}$.
2. Otherwise, $e_1 \to e_1'$ in which case [E-Pred-Ctxt] applies (as shown below). Hence, progress holds by induction.

$$e \to \texttt{pred}\ (e_1')$$

Figure 7: Proof of Progress for the `pred` Expression

Given:

$$e = \texttt{iszero}\ (e_1)$$
$$e_1 : \texttt{Int}$$

Then:

$$T = \texttt{Bool}$$

By induction, e_1 must either be a value or $\exists e_1'$ such that:

1. If e_1 is a value (i.e. "i") in which case either rule [E-IsZero-Z] or [E-IsZero-NZ] applies since $e_1 : \texttt{Int}$.
2. Otherwise, $e_1 \to e_1'$ in which case [E-IsZero-Ctxt] applies (as shown below). Hence, progress holds by induction.

$$e \to \texttt{iszero}\ (e_1')$$

Figure 8: Proof of Progress for the `iszero` Expression

3 Preservation in the Arith Language

Preservation entails that a well-typed expression will not change its type during evaluation. Figure 9 is the formal, theoretical definition of preservation.

Given $e : T$ and that $e \to e'$,
then $e' : T$.

Figure 9: Formal Definition of the Preservation Theorem

The following subsections are the formal proofs of preservation for the type rules in figure 3.

3.1 Proving Preservation for Boolean and Integer Values

For type rules [T-True], [T-False], and [T-Int], preservation holds vacuously as it is not possible to evaluate these expressions given that they are in normal form.

3.2 Proving Preservation for the `if` Expression

Figure 10 is the proof of preservation for the `if` expression in the Arith Language.

Given:

$$e = \texttt{if}\ (e_1)\ \texttt{then}\ (e_2)\ \texttt{else}\ (e_3)$$
$$e_1 : \texttt{Bool},\quad e_2 : T,\quad e_3 : T$$

For the `if` expression, **three** evaluation rules may apply.

1. If [E-IfTrue] applies, then $e_1 = \texttt{true}$. Hence, $e' = e_2$. This proof holds since by definition $e_2 : T$.
2. If [E-IfFalse] applies, then $e_1 = \texttt{false}$. Hence, $e' = e_3$. This proof holds since by definition $e_3 : T$.
3. If [E-If-Ctxt] applies, then $e_1 \to e_1'$. What is more, by induction $e_1' : \texttt{Bool}$ (this can be assumed by induction since e_1 is a subcase of e). Then using [T-If]:

$$e' = \texttt{if}\ (e_1')\ \texttt{then}\ (e_2)\ \texttt{else}\ (e_3) : T$$

Figure 10: Proof of Preservation for the `if` Expression

3.3 Proving Preservation for the `succ` Expression

Figure 11 is the proof of preservation for the `succ` expression in the Arith Language.

Given:

$$e = \texttt{succ}\ (e_1)$$
$$e_1 : \texttt{Int}$$

Then:

$$T = \texttt{Int}$$

For the `succ` expression, **two** evaluation rules may apply.

1. If [E-SUCC-CTXT] applies, then $e_1 \to e_1'$. What is more, by induction $e_1' : \texttt{Int}$. Using type rule [T-SUCC],

$$e' = \texttt{succ}\ (e_1') \to \texttt{Int}$$

2. If [E-SUCC] applies, then $e_1 = i$. Hence, $e = \texttt{succ}\ (i) \to i'$ where $i' = i + 1$. Therefore, $e' : \texttt{Int}$ using Lemma-Int defined in Figure 12.

Figure 11: Proof of Preservation for the `succ` Expression

The preservation proof for successor relies on a lemma defining integer arithmetic; it is shown in Figure 12.

Lemma Int

If $i : \texttt{Int}$ and either:

1. $i' = i + 1$
2. $i' = i - 1$

then $i' : \texttt{Int}$.

Figure 12: Definition of Lemma Int

3.4 Proving Preservation for the `pred` Expression

Figure 13 is the proof of preservation for the `pred` expression in the Arith Language.

3.5 Proving Preservation for the `iszero` Expression

Figure 14 is the proof of preservation for the `iszero` expression in the Arith Language.

Given:

$$e = \texttt{pred}\ (e_1)$$
$$e_1 : \texttt{Int}$$

Then:

$$T = \texttt{Int}$$

For the `pred` expression, **two** evaluation rules may apply.

1. If [E-PRED-CTXT] applies, then $e_1 \to e_1'$ since by induction $e_1' : \texttt{Int}$. Using type rule [T-PRED],

$$e' = \texttt{pred}\ (e_1') \to \texttt{Int}$$

2. If [E-PRED] applies, then $e_1 = i$. Hence, $e = \texttt{pred}\ (i) \to i'$ where $i' = i - 1$. Therefore, $e' : \texttt{Int}$ using Lemma-Int defined in Figure 12.

Figure 13: Proof of Preservation for the `pred` Expression

Given:

$$e = \texttt{iszero}\ (e_1)$$
$$e_1 : \texttt{Int}$$

Then:

$$T = \texttt{Bool}$$

For the `iszero` expression, **three** evaluation rules may apply.

1. If [E-ISZERO-Z] applies, then $e_1 = 0$. Hence, $e' = \texttt{true}$ which holds using type rule [T-TRUE].
2. If [E-ISZERO-NZ] applies, then e_1 is an integer value such that $e_1 \neq 0$. Hence, $e' = \texttt{false}$ which holds using type rule [T-FALSE].
3. If [E-ISZERO-CTXT] applies, then $e_1 \to e_1'$ since by induction $e_1' : \texttt{Bool}$ (this can be assumed by induction since e_1 is a subcase of e). Then using [T-ISZERO]:

$$e' = \texttt{iszero}\ (e_1') \to \texttt{Bool}$$

Figure 14: Proof of Preservation for the `iszero` Expression