

Progress and Preservation Proofs for the  
Expressions “**iszero**” and “**pred**” in the Arith Language

Zayd Hammoudeh  
([zayd.hammoudeh@sjsu.edu](mailto:zayd.hammoudeh@sjsu.edu))

April 14, 2016

# Contents

List of Figures . . . . .	ii
1 Arith Language . . . . .	1
2 Progress . . . . .	4
2.1 Proving Progress for Boolean and Integer Values . . . . .	4
2.2 Proving Progress for the <b>if</b> Expression . . . . .	4
2.3 Proving Progress for the <b>succ</b> Expression . . . . .	5
2.4 Proving Progress for the <b>pred</b> Expression . . . . .	5
2.5 Proving Progress for the <b>iszero</b> Expression . . . . .	6
3 Preservation . . . . .	7
3.1 Proving Preservation for Boolean and Integer Values . . . . .	7
3.2 Proving Preservation for the <b>if</b> Expression . . . . .	7
3.3 Proving Preservation for the <b>succ</b> Expression . . . . .	8
3.4 Proving Preservation for the <b>pred</b> Expression . . . . .	8
3.5 Proving Preservation for the <b>iszero</b> Expression . . . . .	8

# List of Figures

1	The Arith language . . . . .	1
2	Small-Step, Evaluation Order Semantics Semantics for the Arith Language . . . . .	2
3	Type Rules for the Arith Language . . . . .	3
4	Formal Definition of the Progress Theorem . . . . .	4
5	Proof of Progress for the <b>if</b> Expression . . . . .	4
6	Proof of Progress for the <b>succ</b> Expression . . . . .	5
7	Proof of Progress for the <b>pred</b> Expression . . . . .	5
8	Proof of Progress for the <b>iszero</b> Expression . . . . .	6
9	Formal Definition of the Preservation Theorem . . . . .	7
10	Proof of Preservation for the <b>if</b> Expression . . . . .	7
11	Proof of Preservation for the <b>succ</b> Expression . . . . .	8
12	Definition of Lemma Int . . . . .	8
13	Proof of Preservation for the <b>pred</b> Expression . . . . .	9
14	Proof of Preservation for the <b>iszero</b> Expression . . . . .	9

# 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 while Arith's type rules are enumerated in figure 3.

$e ::=$	<i>Expressions</i>
true	Boolean True
false	Boolean False
0	Integer Value 0
succ (e)	Successor Expressions
pred (e)	Predecessor Expressions
iszero (e)	Zero Value Check Expressions
if (e) then (e) else (e)	Conditional Expressions
$v ::=$	<i>Values</i>
i	integer values
b	boolean values
$T ::=$	<i>Types</i>
Bool	Boolean Type
Int	Integer Type

**Figure 1:** The Arith language

**Evaluation Rules:**

$$e \rightarrow e'$$

$$[\text{E-SUCC-CTXT}] \quad \frac{e_1 \rightarrow e'_1}{\text{succ}(e_1) \rightarrow \text{succ}(e'_1)}$$

$$[\text{E-SUCC}] \quad \frac{i' = i + 1}{\text{succ}(i) \rightarrow i'}$$

$$[\text{E-PRED-CTXT}] \quad \frac{e_1 \rightarrow e'_1}{\text{pred}(e_1) \rightarrow \text{pred}(e'_1)}$$

$$[\text{E-PRED}] \quad \frac{i' = i - 1}{\text{pred}(i) \rightarrow i'}$$

$$[\text{E-ISZERO-CTXT}] \quad \frac{e_1 \rightarrow e'_1}{\text{iszero}(e_1) \rightarrow \text{iszero}(e'_1)}$$

$$[\text{E-ISZERO-Z}] \quad \text{iszero}(0) \rightarrow \text{true}$$

$$[\text{E-ISZERO-NZ}] \quad \frac{i \neq 0}{\text{iszero}(i) \rightarrow \text{false}}$$

$$[\text{E-IF-CTXT}] \quad \frac{e_1 \rightarrow e'_1}{\text{if}(e_1) \text{ then } (e_2) \text{ else } (e_3) \rightarrow \text{if}(e'_1) \text{ then } (e_2) \text{ else } (e_3)}$$

$$[\text{E-IF-TRUE}] \quad \text{if}(\text{true}) \text{ then } (e_2) \text{ else } (e_3) \rightarrow e_2$$

$$[\text{E-IF-FALSE}] \quad \text{if}(\text{false}) \text{ then } (e_2) \text{ else } (e_3) \rightarrow e_3$$

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

**Type Rules:**

$e : T$

[T-TRUE]       $\text{true} : \text{Bool}$

[T-FALSE]       $\text{false} : \text{Bool}$

[T-INT]       $i : \text{Int}$

[T-SUCC]      
$$\frac{e_1 : \text{Int}}{\text{succ}(e_1) : \text{Int}}$$

[T-PRED]      
$$\frac{e_1 : \text{Int}}{\text{pred}(e_1) : \text{Int}}$$

[T-ISZERO]      
$$\frac{e_1 : \text{Int}}{\text{iszero}(e_1) : \text{Bool}}$$

[T-IF]      
$$\frac{e_1 : \text{Bool}, \quad e_2 : T, \quad e_3 : T}{\text{if}(e_1) \text{ then } (e_2) \text{ else } (e_3) : T}$$

**Figure 3:** Type Rules for the Arith Language

## 2 Progress

In semantics context, “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:  $e \rightarrow e'$ .

**Figure 4:** Formal Definition of the Progress Theorem

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

### 2.1 Proving Progress for Boolean and Integer Values

Figure 4 establish that progress is achieved if an expression  $e$  is a value. Hence, by this criterion, type rules [T-TRUE] , [T-FALSE] , and [T-INT] all hold.

### 2.2 Proving Progress for the if Expression

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

Given:

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

Then:

$$e : T$$

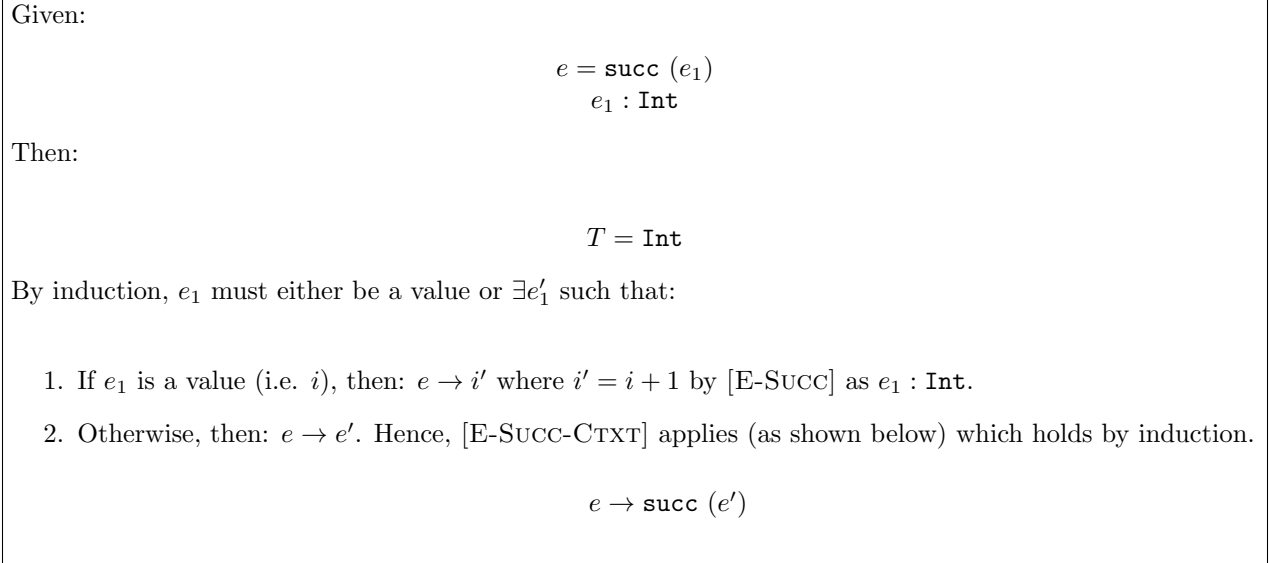
By induction, an expression  $e_1$  must be a value or  $\exists e_1$  such that:

1. If  $e_1$  is a value, then either [E-IF-TRUE] or [E-IF-FALSE] applies since  $e_1 : \text{Bool}$ .
2. Otherwise,  $e_1 \rightarrow e'_1$  which means that [E-IF-CTXT] applies which by induction holds.

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

## 2.3 Proving Progress for the succ Expression

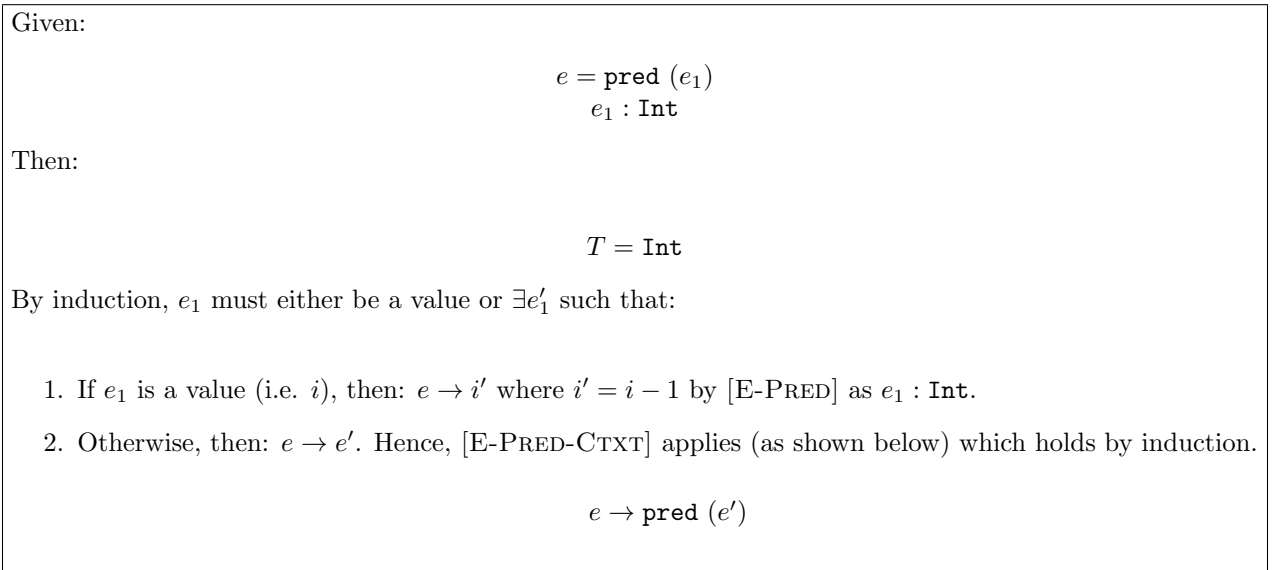
Figure 3 shows the type rule for the **succ** expression. The proof of progress for this expression is shown in figure 6.



**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.

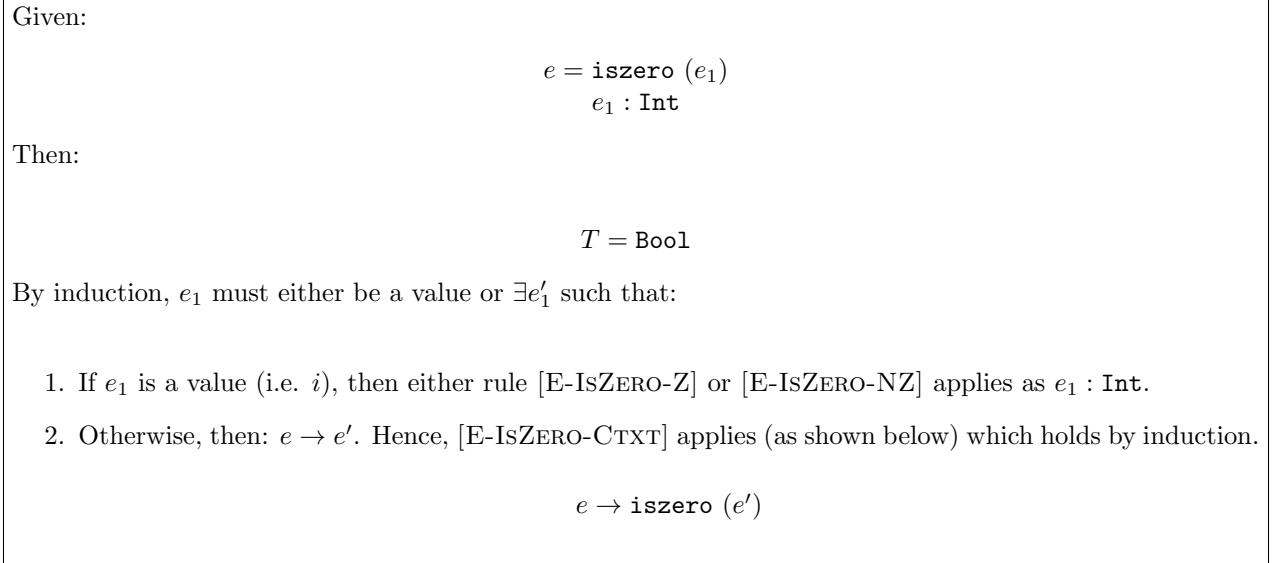


**Figure 7:** Proof of Progress for the **pred** Expression



## 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 7.



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

### 3 Preservation

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  $e \rightarrow e'$ , 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:

$$\begin{aligned} e &= \text{if } (e_1) \text{ then } (e_2) \text{ else } (e_3) \\ e_1 &: \text{Bool}, \quad e_2 : T, \quad e_3 : T \end{aligned}$$

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

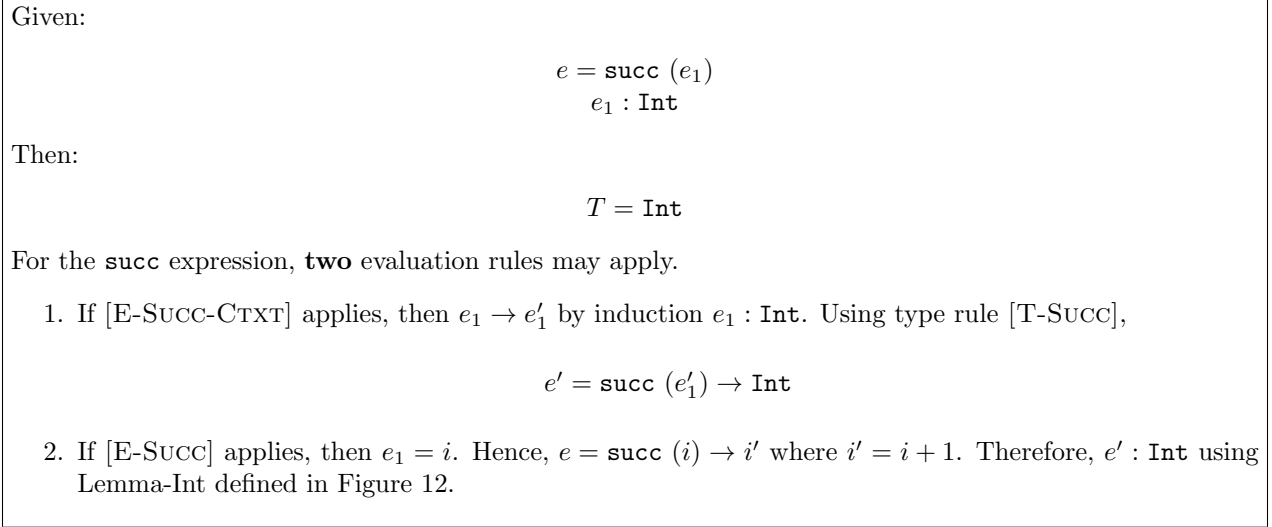
1. If [E-IFTRUE] applies, then  $e_1 = \text{true}$ . Hence,  $e' = e_2$ . This proof holds since by definition  $e_2 : T$ .
2. If [E-IFFALSE] applies, then  $e_1 = \text{false}$ . Hence,  $e' = e_3$ . This proof holds since by definition  $e_3 : T$ .
3. If [E-IFCXTT] applies, then  $e_1 \rightarrow e'_1$  by induction  $e_1 : \text{Bool}$  (this can be assumed by induction since  $e_1$  is a subcase of  $e$ ). Then using [T-IF]:

$$e' = \text{if } (e'_1) \text{ then } (e_2) \text{ else } (e_3) \rightarrow T$$

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

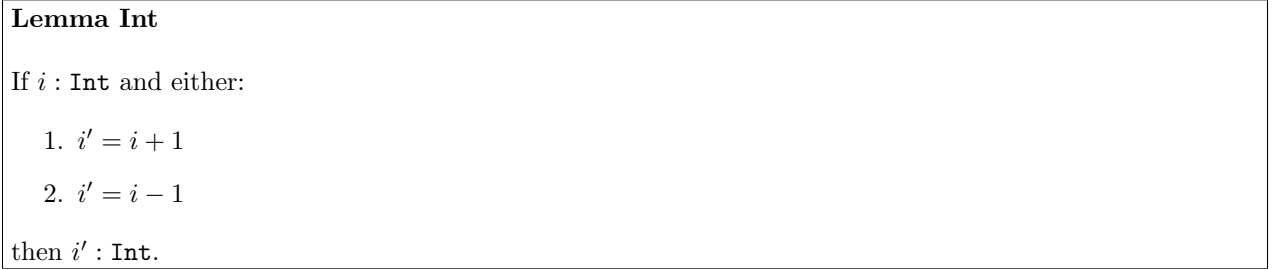
### 3.3 Proving Preservation for the succ Expression

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



**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.



**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:

$$\begin{aligned} e &= \text{pred } (e_1) \\ e_1 &: \text{Int} \end{aligned}$$

Then:

$$T = \text{Int}$$

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

1. If [E-PRED-CTXT] applies, then  $e_1 \rightarrow e'_1$  by induction  $e_1 : \text{Int}$ . Using type rule [T-PRED],

$$e' = \text{pred } (e'_1) \rightarrow \text{Int}$$

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

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

Given:

$$\begin{aligned} e &= \text{iszero } (e_1) \\ e_1 &: \text{Int} \end{aligned}$$

Then:

$$T = \text{Bool}$$

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

1. If [E-ISZERO-Z] applies, then  $e_1 = 0$ . Hence,  $e' = \text{true}$  which holds using rule [T-TRUE].
2. If [E-ISZERO-NZ] applies, then  $e_1$  is a value such that  $e_1 \neq 0$ . Hence,  $e' = \text{false}$  which holds using rule [T-FALSE].
3. If [E-ISZERO-CTXT] applies, then  $e_1 \rightarrow e'_1$  by induction  $e_1 : \text{Bool}$  (this can be assumed by induction since  $e_1$  is a subcase of  $e$ ). Then using [T-ISZERO]:

$$e' = \text{iszero } (e'_1) \rightarrow \text{Bool}$$

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