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

Zayd Hammoudeh
(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 `if` Expression . . . . . . . . . . 4

2.3 Proving Progress for the `succ` Expression . . . . . . . . . . 5

2.4 Proving Progress for the `pred` Expression . . . . . . . . . . 5

2.5 Proving Progress for the `iszero` Expression . . . . . . . . . . 6

3 Preservation . . . . . . . . . . 7

3.1 Proving Preservation for Boolean and Integer Values . . . . . . . . . . 7

3.2 Proving Preservation for the `if` Expression . . . . . . . . . . 7

# 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 `if` Expression . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

6 Proof of Progress for the `succ` Expression . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

7 Proof of Progress for the `pred` Expression . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

8 Proof of Progress for the `iszero` Expression . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

9 Formal Definition of the Preservation Theorem . . . . . . . . . . . . . . . . . . . . . . . . . . 7

10 Proof of Preservation for the `if` Expression . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

# 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.

| | | |
|---|---|---|
| $e$ ::= | | *Expressions* |
| | `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 |
| $v$ ::= | | *Values* |
| | $i$ | integer values |
| | $b$ | boolean values |
| $T$ ::= | | *Types* |
| | `Bool` | Boolean Type |
| | `Int` | Integer Type |

**Figure 1:** The Arith language

**Evaluation Rules:** $\boxed{e \rightarrow e'}$

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

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

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

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

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

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

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

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

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

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

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

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

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

$$[\text{T-F}\textsc{alse}] \qquad \texttt{false} : \texttt{Bool}$$

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

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

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

$$[\text{T-I}\textsc{s}\text{Z}\textsc{ero}] \qquad \frac{e_1 : \texttt{Int}}{\texttt{iszero}\ (e_1) : \texttt{Bool}}$$

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

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

## 2 Progress

In 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: $e \to e'$.

**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:

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

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 $e_1 : \texttt{Bool}$.
2. Otherwise, $e_1 \to e_1'$ which means that [E-If-Ctxt] applies (as shown below) which by induction holds.

$$\succ e_1 \to \texttt{if}\ (e_1')\ \texttt{then}\ (e_2)\ \texttt{else}\ (e_3)$$

**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: $e \to i'$ where $i' = i + 1$ by [E-SUCC] as $e_1 : \texttt{Int}$.
2. Otherwise, then: $e \to e'$. Hence, [E-SUCC-CTXT] applies (as shown below) which holds by induction.

$$e \to \texttt{succ}\ (e')$$

**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.

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 $i' = i - 1$ by [E-PRED] as $e_1 : \texttt{Int}$.
2. Otherwise, then: $e \to e'$. Hence, [E-PRED-CTXT] applies (as shown below) which holds by induction.

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

**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.

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. $i$), then either rule [E-IsZero-Z] or [E-IsZero-NZ] applies as $e_1 : \texttt{Int}$.
2. Otherwise, then: $e \to e'$. Hence, [E-IsZero-Ctxt] applies (as shown below) which holds by induction.

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

**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 $e \to e'$, 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:

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

Then:

$$e' : T$$

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 $e_1 \to e_1'$ by induction $e_1 : \texttt{Bool}$ (this can be assumed by induction since $e_1$ is a subcase of $e$). Furthermore, by induction using [T-If], then:

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

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