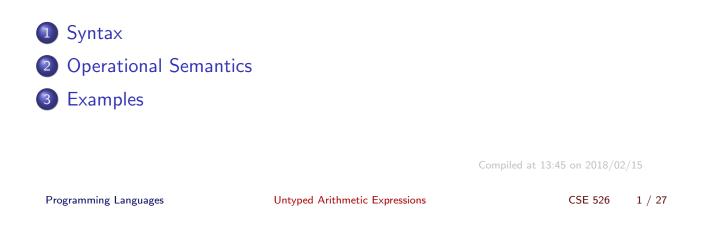
## Untyped Arithmetic Expressions

Principles of Programming Languages

CSE 526



## Formal Description of Programming Languages

- Formal Definition of Syntax
  - Grammars to define the set of *strings* that define a syntactically valid program
  - Inductive definitions of *abstract syntax trees*.
- Formal Definition of Semantics
  - Structural operational semantics

Example: A language of untyped arithmetic expressions

t

```
::= true \\ | false \\ if(t, t, t) \\ | 0 \\ | succ t \\ pred t \\ iszero t
```

**Inductive Definition:** The set T of *terms* is the smallest set such that:

- **2** if  $t_1 \in \mathcal{T}$  then {succ  $t_1$ , pred  $t_1$ , iszero  $t_1$ }  $\subseteq \mathcal{T}$ .
- **3** if  $t_1, t_2, t_3 \in \mathcal{T}$  then  $if(t_1, t_2, t_3) \in \mathcal{T}$ .

Programming Languages

Untyped Arithmetic Expressions

CSE 526 3 / 27

Syntax

### Alternative Definitions of Terms

**Inductive Definition:** The set T of *terms* is the smallest set such that:

- **2** if  $t_1 \in \mathcal{T}$  then {succ  $t_1$ , pred  $t_1$ , iszero  $t_1$ }  $\subseteq \mathcal{T}$ .
- **3** if  $t_1, t_2, t_3 \in \mathcal{T}$  then  $if(t_1, t_2, t_3) \in \mathcal{T}$ .

**Inference Rules:** The set  $\mathcal{T}$  is defined by the following rules:

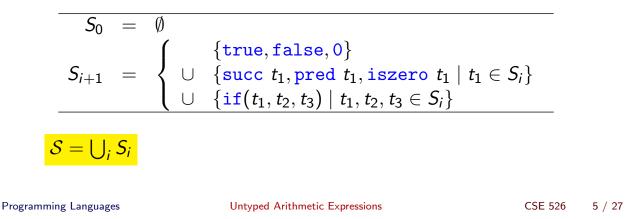
$\texttt{true} \in \mathcal{T}$	$\texttt{false} \in \mathcal{T}$	$0\in\mathcal{T}$
$\frac{t_1 \in \mathcal{T}}{\texttt{succ } t_1 \in \mathcal{T}}$	$\frac{t_1 \in \mathcal{T}}{\texttt{pred} \ t_1 \in \mathcal{T}}$	$rac{t_1 \in \mathcal{T}}{ ext{iszero } t_1 \in \mathcal{T}}$
	$rac{t_1,t_2,t_3\in\mathcal{T}}{ extsf{if}(t_1,t_2,t_3)\in\mathcal{T}}$	

## Alternative Definitions of Terms (contd.)

**Inductive Definition:** The set T of *terms* is the smallest set such that:

- $\texttt{1} \ \{\texttt{true}, \texttt{false}, \texttt{0}\} \subseteq \mathcal{T}$
- **2** if  $t_1 \in \mathcal{T}$  then {succ  $t_1$ , pred  $t_1$ , iszero  $t_1$ }  $\subseteq \mathcal{T}$ .
- 3 if  $t_1, t_2, t_3 \in \mathcal{T}$  then  $if(t_1, t_2, t_3) \in \mathcal{T}$ .

**Constructive Definition:** For each natural number *i* define set  $S_i$  as follows:



Syntax

Alternative Definitions of Terms (contd.)

Properties:

- The sets  $S_i$  are cumulative, i.e.,  $\forall i \ S_i \subseteq S_{i+1}$
- $\mathcal{T} = \mathcal{S}$ 
  - $\textcircled{0} \ \mathcal{S} \ \text{satisfies the conditions on } \mathcal{T}$
  - 2 Let S' be a set that satisfies the conditions on  $\mathcal{T}$ . Then  $S \subseteq S'$ .

### Equivalence of ${\mathcal S}$ and ${\mathcal T}$

$$\begin{array}{rcl} S_0 &=& \emptyset \\ S_{i+1} &=& \left\{ \begin{array}{cc} \{\texttt{true}, \texttt{false}, 0\} \\ \cup & \{\texttt{succ } t_1, \texttt{pred } t_1, \texttt{iszero } t_1 \mid t_1 \in S_i\} \\ \cup & \{\texttt{if}(t_1, t_2, t_3) \mid t_1, t_2, t_3 \in S_i\} \end{array} \right. \end{array}$$

### **1.** $\forall i \ S_i \subseteq S_{i+1}$

Proof is by *ordinary* induction on *i*: P(0) and  $\forall k.P(k) \implies P(k+1)$ , where  $P(i) : S_i \subseteq S_{i+1}$ 

P(0):  $S_0$  is empty, and hence is a subset of  $S_1$ .

 $P(k) \implies P(k+1)$ : We'll show that every  $t \in S_{k+1}$  is also  $\in S_{k+2}$ .

Consider  $t \in S_{k+1}$ . Then t is of one of the following forms:

- 1.  $t \in \{\texttt{true}, \texttt{false}, 0\}$ . Then  $t \in S_{k+2}$  by definition.
- 2.  $t = \operatorname{succ}(t_1)$  for some  $t_1 \in S_k$ . By ind. hyp.,  $t_1 \in S_{k+1}$  and hence  $t \in S_{k+2}$ .
- 3-5. proof steps for terms of the form  $pred(t_1)$  etc. are similar to case 2.

Programming Languages

Untyped Arithmetic Expressions

```
CSE 526 7 / 27
```

Syntax

# Equivalence of ${\mathcal S}$ and ${\mathcal T}$

$$\mathcal{T} \text{ is the smallest set such that}$$

$$\{ \text{true, false, 0} \} \subseteq \mathcal{T}$$

$$\text{if } t_1 \in \mathcal{T} \text{ then } \{ \text{succ } t_1, \\ \text{pred } t_1, \text{ iszero } t_1 \} \subseteq \mathcal{T}.$$

$$\text{if } t_1, t_2, t_3 \in \mathcal{T} \text{ then} \\ \text{if } (t_1, t_2, t_3) \in \mathcal{T}.$$

$$S_{i+1} = \begin{cases} \{ \text{true, false, 0} \} \\ \cup \{ \text{succ } t_1, \text{pred } t_1, \\ \text{iszero } t_1 \mid t_1 \in S_i \} \\ \cup \{ \text{if}(t_1, t_2, t_3) \mid t_1, t_2, t_3 \in S_i \} \end{cases}$$

$$S = \bigcup_{i \ge 0} S_i$$

$$S = \bigcup_{i \ge 0} S_i$$

- ${\tt true, false, 0}$  are in  $S_1$  and hence in S.
- ② If  $t_1 \in S$  then  $t_1 \in S_k$  for some  $k \ge 0$ . Hence, {succ( $t_1$ ), pred( $t_1$ ), iszero( $t_1$ )} ⊆  $S_{k+1}$  and consequently ⊆ S.

◎ If  $t_1, t_2, t_3 \in S$  then there are  $k_1, k_2, k_3$  such that  $t_1 \in S_{k_1}, t_2 \in S_{k_2}$ , and  $t_3 \in S_{k_3}$ . From Lemma 1,  $t_1, t_2, t_3$  are all  $\in S_k$  for  $k \ge \max(k_1, k_2, k_3)$ . Hence  $if(t_1, t_2, t_3) \in S_{k+1}$  and consequently  $\in S$ .

# Equivalence of $\mathcal{S}$ and $\mathcal{T}$ (Contd.)

$$\mathcal{T} \text{ is the smallest set such that}$$

$$\{ \text{true, false, 0} \} \subseteq \mathcal{T}$$

$$\text{if } t_1 \in \mathcal{T} \text{ then } \{ \text{succ } t_1, \\ \text{pred } t_1, \text{ iszero } t_1 \} \subseteq \mathcal{T}.$$

$$\text{if } t_1, t_2, t_3 \in \mathcal{T} \text{ then} \\ \text{if } (t_1, t_2, t_3) \in \mathcal{T}.$$

**2b.** If S' satisfies the conditions 1, 2, and 3 on T, then  $S \subseteq S'$ . We will show this by proving that  $S_i$  is a subset of S' by *complete* induction:  $(\forall j < i P(j)) \implies P(i).$ 

We get two cases from the definition of  $S_i$ :

$$i = 0$$
:  $S_0 \subseteq S'$ .

 $\exists j. \ i = j + 1$ : Every  $t \in S_{j+1}$  is also  $\in S'$ .

- 1. {true, false, 0} are in S' by condition 1.
- 2. If  $t = \operatorname{succ}(t_1) \in S_{j+1}$  for  $t_1 \in S_j$ , then by ind. hyp.  $t_1 \in S'$ , and  $t \in S'$  by condition 2.
- 3–5. Proof steps are similar to case 2 for  $t = \text{pred}(t_1)$  etc.

Programming Languages

Untyped Arithmetic Expressions

```
CSE 526 9 / 27
```

Syntax

### Inductive Definitions

The following recursive definition is "well-defined" since the function on a term is defined based on that on *smaller* terms.

$$Const(true) = \{true\}$$

$$Const(false) = \{false\}$$

$$Const(0) = \{0\}$$

$$Const(succ t_1) = Const(t_1)$$

$$Const(pred t_1) = Const(t_1)$$

$$Const(iszero t_1) = Const(t_1)$$

$$Const(if(t_1, t_2, t_3)) = Const(t_1) \cup Const(t_2) \cup Const(t_3)$$

## Inductive Definitions (contd.)

The size of a term is also defined inductively:

$$size(true) = 1$$
  
 $size(false) = 1$   
 $size(0) = 1$   
 $size(succ t_1) = size(t_1) + 1$   
 $size(pred t_1) = size(t_1) + 1$   
 $size(iszero t_1) = size(t_1) + 1$   
 $size(if(t_1, t_2, t_3)) = size(t_1) + size(t_2) + size(t_3) + 1$ 

The *depth* of terms can be defined similarly.

Programming Languages Untyped Arithmetic Expressions CSE 526 11 / 27
Syntax

### Induction on terms

### • Structural Induction:

- If, for each term s, given P(r) for all immediate sub-terms r of swe can show P(s)then P(s) holds for all s.
- Induction on size:

If, for each term s, given P(r) for all terms r such that size(r) < size(s)we can show P(s)then P(s) holds for all s.

# **Operational Semantics**

**Example:** A language of untyped boolean expressions  $\mathcal{B}$ :

t	::=	<b>true</b>   false   if( $t, t, t$ )	Terms
V	::=	true false	<b>Values</b>

Evaluation:

$$\begin{aligned} & \texttt{if}(\texttt{true}, t_2, t_3) \rightarrow t_2 & \text{E-IFTRUE} \\ & \texttt{if}(\texttt{false}, t_2, t_3) \rightarrow t_3 & \text{E-IFFALSE} \\ & \underline{t_1 \rightarrow t_1'} \\ & \underline{\texttt{if}(t_1, t_2, t_3) \rightarrow \texttt{if}(t_1', t_2, t_3)} & \text{E-IF} \end{aligned}$$

Programming Languages	Untyped Arithmetic Expressions	CSE 526	13 / 27

**Operational Semantics** 

### The Inference Rule Notation



- Inference rules without premises are called *axioms*.
- Inference rules (more precisely *rule schema*) may have meta-variables.
   E.g., t<sub>1</sub>, t<sub>2</sub>, t<sub>3</sub>, t'<sub>1</sub> in:

$$rac{t_1 
ightarrow t_1'}{ extsf{if}(t_1,t_2,t_3) 
ightarrow extsf{if}(t_1',t_2,t_3)} \quad extsf{E-IF}$$

A *rule instance* is obtained by consistently replacing each meta-variable by the same term in the premises as well as the conclusion.

#### **Operational Semantics**

### **Operational Semantics of Boolean Expressions**

$$\begin{split} & \texttt{if}(\texttt{true}, t_2, t_3) \rightarrow t_2 \qquad \text{E-IFTRUE} \\ & \texttt{if}(\texttt{false}, t_2, t_3) \rightarrow t_3 \qquad \text{E-IFFALSE} \\ & \underline{t_1 \rightarrow t_1'} \\ & \underline{\texttt{if}(t_1, t_2, t_3) \rightarrow \texttt{if}(t_1', t_2, t_3)} \quad \text{E-IF} \end{split}$$

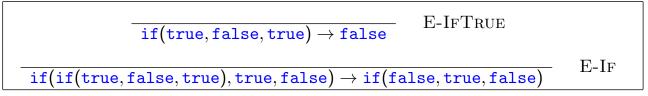
- The *one-step evaluation relation* is the smallest relation "→" on terms satisfying the above rules.
- When (t, t') is in the evaluation relation, we say that the evaluation statement (or judgment)  $t \rightarrow t'$  is derivable.
- **Determinacy:** If  $t \to t'$  and  $t \to t''$  then t' = t''.

Programming Languages	Untyped Arithmetic Expressions	CSE 526	15 / 27

**Operational Semantics** 

### Example Evaluations





Step 2:

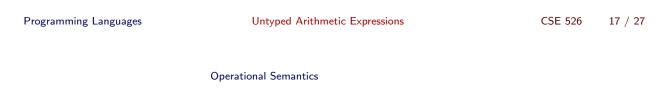
 $if(false, true, false) \rightarrow false$  E-IFFALSE

if(if(true, false, true), true, false) $\rightarrow if(false, true, false)$  $\rightarrow false$ 

### **Small-Step Semantics**

# Properties of (previously defined) operational semantics

- **Determinacy:** If  $t \to t'$  and  $t \to t''$  then t' = t''.
- Proof: by induction on the derivation of  $t \rightarrow t'$ .
- This proof is also identical to induction on structure of t
- The operational semantics defined previously is said to be "*Structural Operational Semantics* (SOS)", where the evaluation derivation follows the structure of the term being reduced.



### Proof of Determinacy

$$\begin{array}{c|c} \text{if}(\texttt{true}, t_2, t_3) \rightarrow t_2 & \text{E-IFTRUE} \\ \text{if}(\texttt{false}, t_2, t_3) \rightarrow t_3 & \text{E-IFFALSE} \end{array} \qquad \begin{array}{c} t_1 \rightarrow t_1' \\ \hline \texttt{if}(t_1, t_2, t_3) \rightarrow \texttt{if}(t_1', t_2, t_3) \end{array} \qquad \text{E-IF} \end{array}$$

If  $t \to t'$  and  $t \to t''$  then t' = t''.

Consider the last rule used in the derivation of  $t \rightarrow t'$ .

- E-IFTRUE: Then  $t = if(true, t_2, t_3)$  for some terms  $t_2$  and  $t_3$ , and  $t' = t_2$ . Consider the derivation  $t \to t''$ . The last rule used here cannot be E-IFFALSE (does not match) or E-IF (premise does not hold). Hence the last rule used in  $t \to t''$  must be E-IFTRUE, and  $t'' = t_2 = t'$ .
- E-IFFALSE: Similar to above case.
- E-IF: Then  $t = if(t_1, t_2, t_3)$ ,  $t_1 \rightarrow t'_1$  (premise), and  $t' = if(t'_1, t_2, t_3)$ . The last rule used in derivation of  $t \rightarrow t''$  must also be E-IF, with premise  $t_1 \rightarrow t''_1$ , resulting in  $t'' = if(t''_1, t_2, t_3)$ . By induction hypotheses, we know  $t'_1 = t''_1$ .

## Normal Form

- A term t is in normal form if there is no t' such that  $t \rightarrow t'$ .
- Every *value* is in normal form.
- If t is in normal form, then t is a value.
- Let "→\*" relation be the reflexive, transitive closure of "→" relation in the following:
- Uniqueness: If  $t \rightarrow^* u$  and  $t \rightarrow^* u'$  where u and u' are normal forms, then u = u'.

If  $t \to^* u$ , and u is in normal form, we say u is the normal form of t.

• **Termination:** For every term t, there is some normal form t' such that  $t \rightarrow^* t'$ .

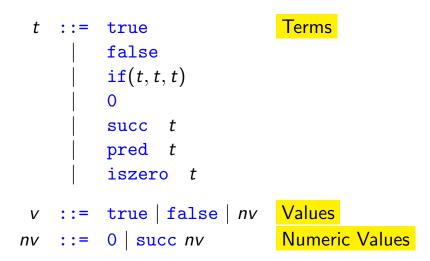
Programming Languages

Untyped Arithmetic Expressions

CSE 526 19 / 27

**Operational Semantics** 

Untyped Arithmetic Expressions



## **Operational Semantics of Untyped Arithmetic Expressions**

$$\frac{t_1 \rightarrow t'_1}{\text{succ } t_1 \rightarrow \text{succ } t'_1} \quad \text{E-Succ}$$

$$pred \ 0 \rightarrow 0 \quad \text{E-PREDZERO}$$

$$pred \ \text{succ } nv_1 \rightarrow nv_1 \quad \text{E-PREDSUCC}$$

$$\frac{t_1 \rightarrow t'_1}{\text{pred } t_1 \rightarrow \text{pred } t'_1} \quad \text{E-PRED}$$

$$\text{iszero } 0 \rightarrow \text{true} \quad \text{E-IsZEROZERO}$$

$$\frac{t_1 \rightarrow t'_1}{\text{iszero } t_1 \rightarrow \text{false}} \quad \text{E-IsZEROSUCC}$$

$$\frac{t_1 \rightarrow t'_1}{\text{iszero } t_1 \rightarrow \text{iszero } t'_1} \quad \text{E-IsZERO}$$

Programming Languages

Untyped Arithmetic Expressions

CSE 526 21 / 27

**Operational Semantics** 

### Properties of the operational semantics (prev. slide)

- **Determinacy**
- Uniqueness of normal forms
- Termination
- Not all normal forms are values!

A term is stuck if it is in normal form but not a value.

Stuck terms correspond to "run-time errors".

Examples

### Reflexive Transitive Closure

Let  $R \subseteq D \times D$  be a binary relation. The reflexive transitive closure  $R^*$  of R is the smallest relation such that

- $\forall d \in D$   $(d,d) \in R^*$
- $R \subseteq R^*$
- $\forall x, y, z \in D$   $(x, y) \in R^* \land (y, z) \in R^* \Rightarrow (x, z) \in R^*$

Inference rules for  $\rightarrow$ , the small-step transition relation (Ex. 3.5.10):

$$\begin{array}{c} t \rightarrow^{*} t \\ \\ \frac{t \rightarrow t'}{t \rightarrow^{*} t'} \\ \\ \frac{t \rightarrow^{*} t' \quad t' \rightarrow^{*} t''}{t \rightarrow^{*} t''} \end{array}$$

Programming Languages

Untyped Arithmetic Expressions

CSE 526 23 / 27

Examples

## **Big-Step Semantics**

Small-Step Semantics for **B**:

$$\begin{aligned} & \text{if}(\texttt{true}, t_2, t_3) \to t_2 & \text{E-IFTRUE} \\ & \text{if}(\texttt{false}, t_2, t_3) \to t_3 & \text{E-IFFALSE} \\ & \frac{t_1 \to t_1'}{\texttt{if}(t_1, t_2, t_3) \to \texttt{if}(t_1', t_2, t_3)} & \text{E-IF} \end{aligned}$$

$$\forall t \exists v. t \rightarrow^* v \quad (\text{Uniqueness of N.F. \& Termination})$$

Big-Step Semantics for **B**:

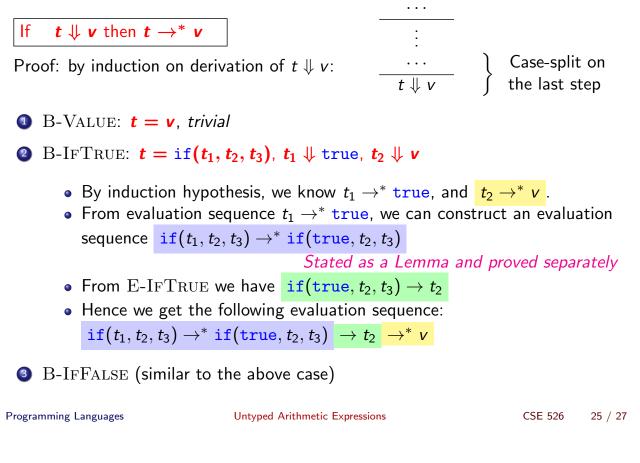
$$v \Downarrow v$$
 B-VALUE

$$\frac{t_1 \Downarrow \text{true} \quad t_2 \Downarrow v_2}{\text{if}(t_1, t_2, t_3) \Downarrow v_2} \quad \text{B-IFTRUE}$$

$$\frac{t_1 \Downarrow \texttt{false} \quad t_3 \Downarrow v_3}{\texttt{if}(t_1, t_2, t_3) \Downarrow v_3} \quad \text{B-IFFALSE}$$

Examples

# Soundness of Big-Step Semantics



Examples

# Soundness of Big-Step Semantics (contd.)

Lemma needed for soundness proof:  $\begin{array}{c|c} \text{If} \quad t_1 \rightarrow^* t'_1 \quad \text{then} \quad \text{if}(t_1, t_2, t_3) \rightarrow^* \text{if}(t'_1, t_2, t_3) \end{array}$ 

Proof: by induction on the length of evaluation sequence  $t_1 \rightarrow^* t'_1$ .

- $t_1 = t'_1$  (i.e zero-length evaluation sequence): trivial.
- $t_1 \rightarrow \hat{t_1} \rightarrow^* t'_1$ : Then  $t_1$  is not a value (by defn of small-step semantics)
  - By E-IF,  $if(t_1, t_2, t_3) 
    ightarrow if(\hat{t_1}, t_2, t_3)$
  - By induction hypothesis,  $\hat{t_1} \rightarrow^* t'_1$  means  $if(\hat{t_1}, t_2, t_3) \rightarrow^* if(t'_1, t_2, t_3)$
  - Hence  $if(t_1, t_2, t_3) \rightarrow if(\hat{t_1}, t_2, t_3) \rightarrow^* if(t'_1, t_2, t_3)$

Examples

# Completeness of Big-Step Semantics If $t \rightarrow^* v$ then $t \Downarrow v$

Proof: by induction on length of evaluation of  $t \rightarrow^* v$ 

• t = v:  $v \Downarrow v$  by B-VALUE.

2 
$$t \rightarrow \hat{t} \rightarrow^* v$$
: then  $t = if(t_1, t_2, t_3)$ .

Use the following lemma:

 $\begin{array}{l} \text{If } \texttt{if}(t_1,t_2,t_3) \to^* v \text{ then} \\ t_1 \to^* \texttt{true} \text{ and } t_2 \to^* v \text{, or} \\ t_1 \to^* \texttt{false} \text{ and } t_3 \to^* v \end{array}$ 

and the evaluation sequences for  $t_1$  and  $t_2$  or  $t_3$  are strictly shorter than the given evaluation sequence.

If  $t_1 \rightarrow^* \text{true}$ , then by induction hypothesis,  $t_1 \Downarrow \text{true}$  and  $t_2 \Downarrow v$ Hence by applying B-IFTRUE, we get  $if(t_1, t_2, t_3) \Downarrow v$ . Proof if  $t_1 \rightarrow^* \text{false}$  is similar.

Programming Languages

Untyped Arithmetic Expressions

CSE 526 27 / 27