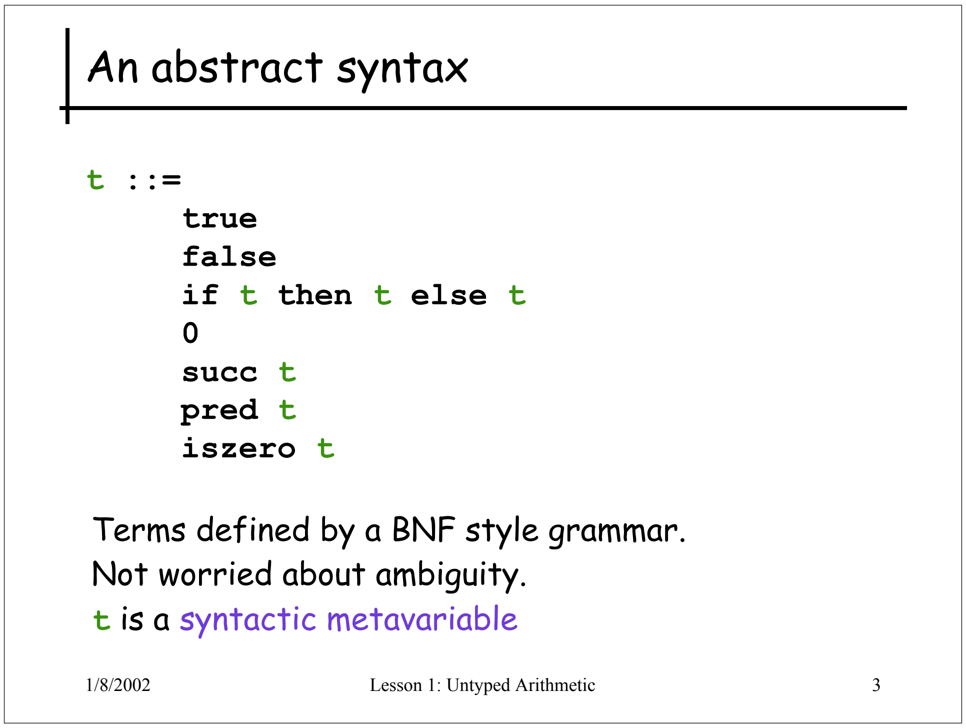
Lesson 1
Untyped Arithmetic Expressions

1/8/2002

Topics

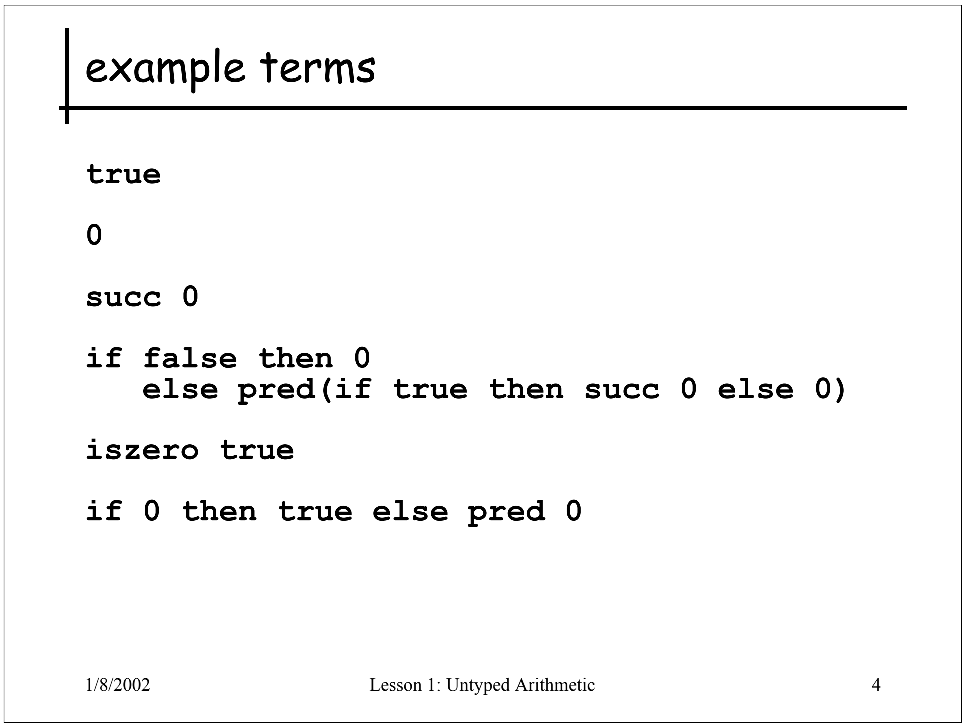
- abstract syntax
- inductive definitions and proofs
- evaluation
- modeling runtime errors



An abstract syntax

```
t ::=
     true
     false
     if t then t else t
     0
     succ t
     pred t
     iszero t
```

Terms defined by a BNF style grammar.
Not worried about ambiguity.
t is a syntactic metavariable



example terms

```
true

0

succ 0

if false then 0
   else pred(if true then succ 0 else 0)

iszero true

if 0 then true else pred 0
```

Inductive defn of terms

Defn: The set of terms is the smallest set $\mathcal{T}$ such that

1. {true, false, 0} $\subseteq \mathcal{T}$
2. if t1 $\in \mathcal{T}$, then {**`succ`** t, **`pred`** t, **`iszero`** t} $\subseteq \mathcal{T}$
3. if t1, t2, t3 $\in \mathcal{T}$, then **`if`** t1 **`then`** t2 **`else`** t3 $\in \mathcal{T}$

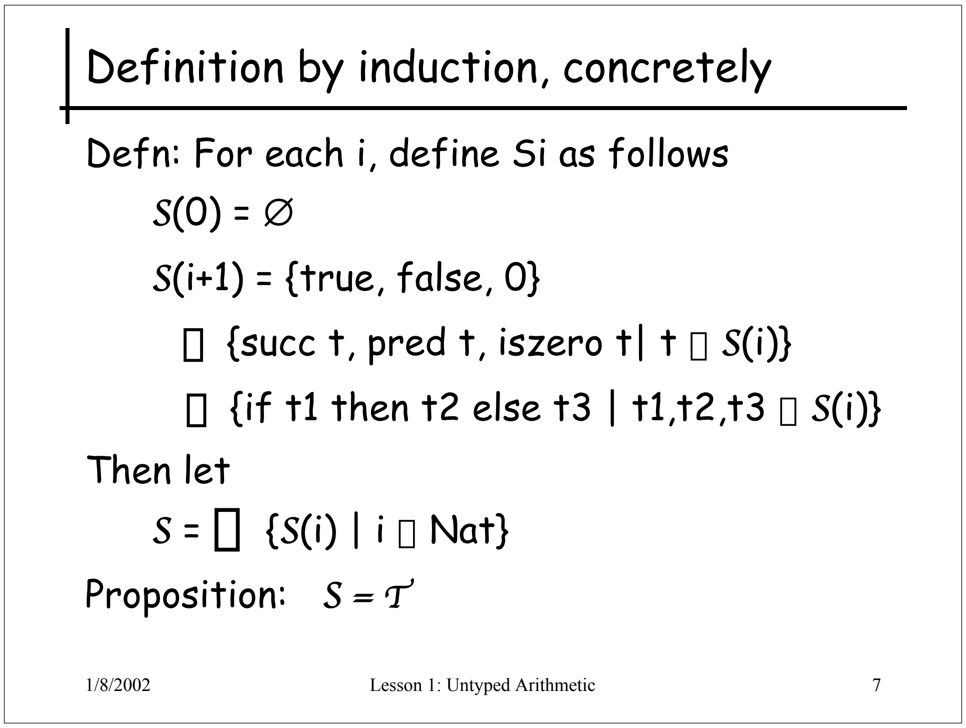
Terms defined using inference rules

Defn: The set of terms is defined by the following rules:

$$\texttt{true} \in \mathcal{T} \qquad \texttt{false} \in \mathcal{T} \qquad 0 \in \mathcal{T}$$

$$\frac{t \in \mathcal{T}}{\texttt{succ t} \in \mathcal{T}} \qquad \frac{t \in \mathcal{T}}{\texttt{pred t} \in \mathcal{T}} \qquad \frac{t \in \mathcal{T}}{\texttt{iszero t} \in \mathcal{T}}$$

$$\frac{t1 \in \mathcal{T} \qquad t2 \in \mathcal{T} \qquad t3 \in \mathcal{T}}{\text{if } t1 \text{ then } t2 \text{ else } t3 \in \mathcal{T}}$$



Definition by induction, concretely

Defn: For each i, define Si as follows

$\mathcal{S}(0) = \varnothing$

$\mathcal{S}(i+1) = \{\text{true, false, } 0\}$

$\cup \; \{\text{succ } t, \text{pred } t, \text{iszero } t \mid t \in \mathcal{S}(i)\}$

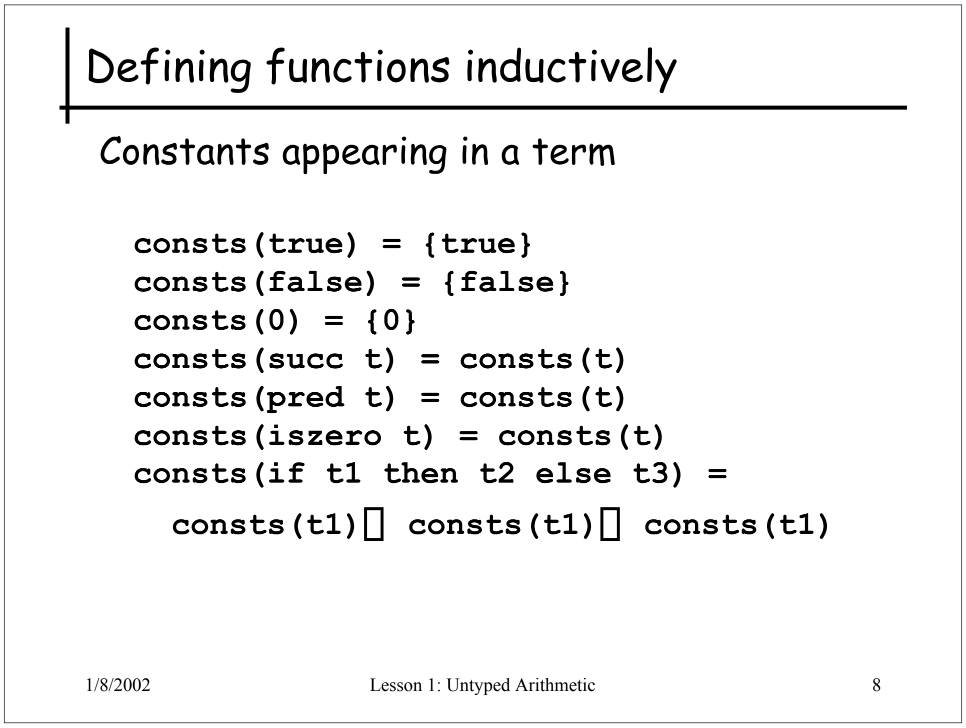
$\cup \; \{\text{if } t1 \text{ then } t2 \text{ else } t3 \mid t1,t2,t3 \in \mathcal{S}(i)\}$

Then let

$\mathcal{S} = \bigcup \; \{\mathcal{S}(i) \mid i \in \text{Nat}\}$

Proposition: $\mathcal{S} = \mathcal{T}$

1/8/2002 Lesson 1: Untyped Arithmetic 7

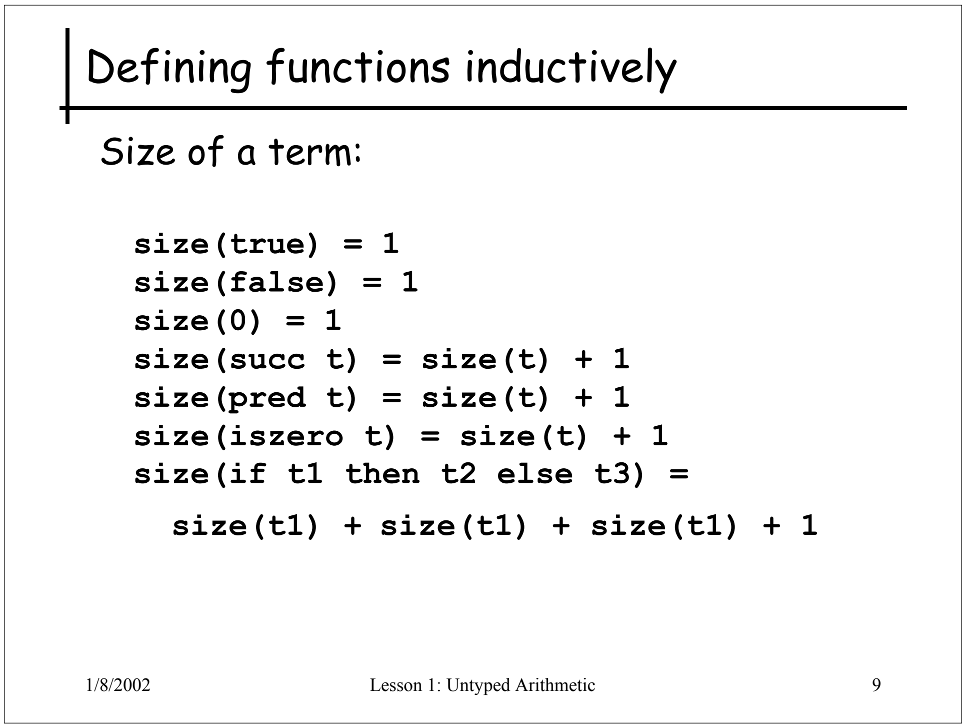


Defining functions inductively

Constants appearing in a term

```
consts(true) = {true}
consts(false) = {false}
consts(0) = {0}
consts(succ t) = consts(t)
consts(pred t) = consts(t)
consts(iszero t) = consts(t)
consts(if t1 then t2 else t3) =
  consts(t1)∪ consts(t1)∪ consts(t1)
```

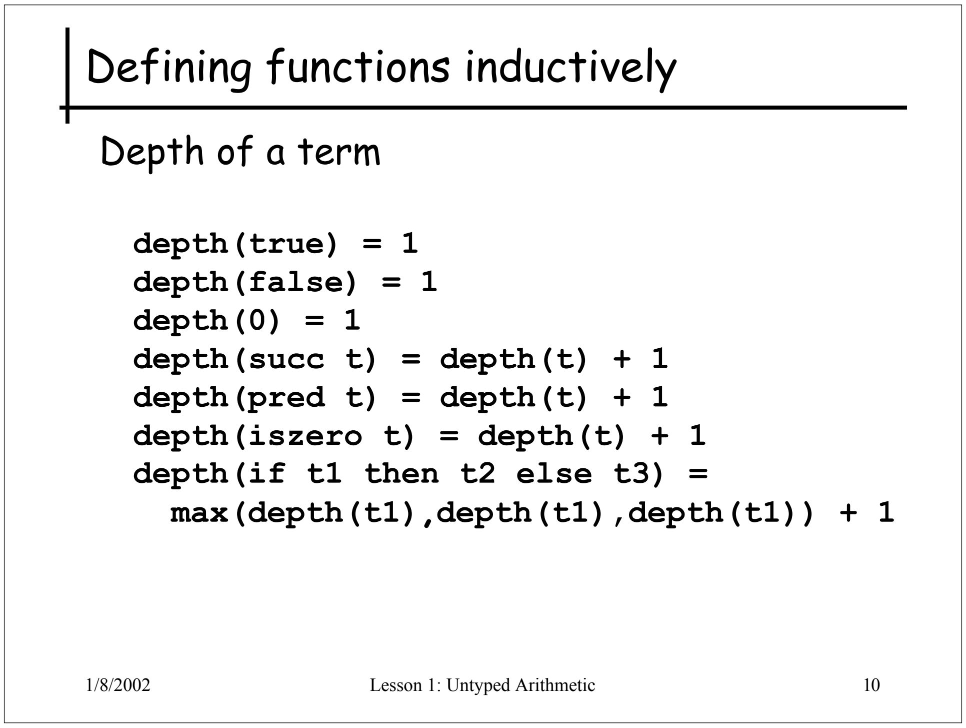
1/8/2002 Lesson 1: Untyped Arithmetic 8



Defining functions inductively

Size of a term:

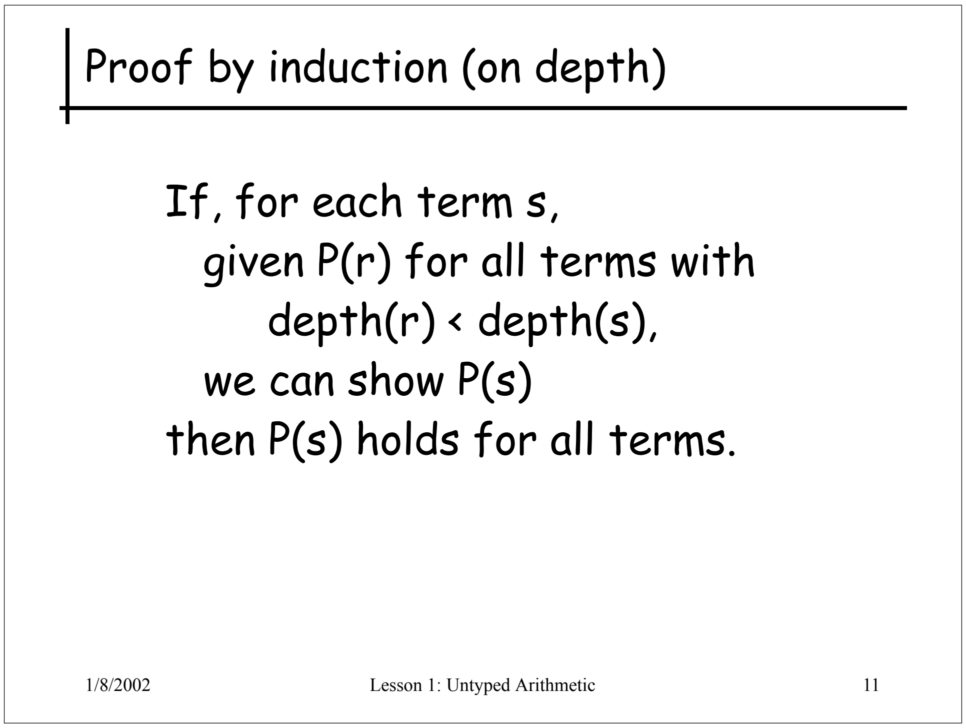
```
size(true) = 1
size(false) = 1
size(0) = 1
size(succ t) = size(t) + 1
size(pred t) = size(t) + 1
size(iszero t) = size(t) + 1
size(if t1 then t2 else t3) =
  size(t1) + size(t1) + size(t1) + 1
```



Defining functions inductively

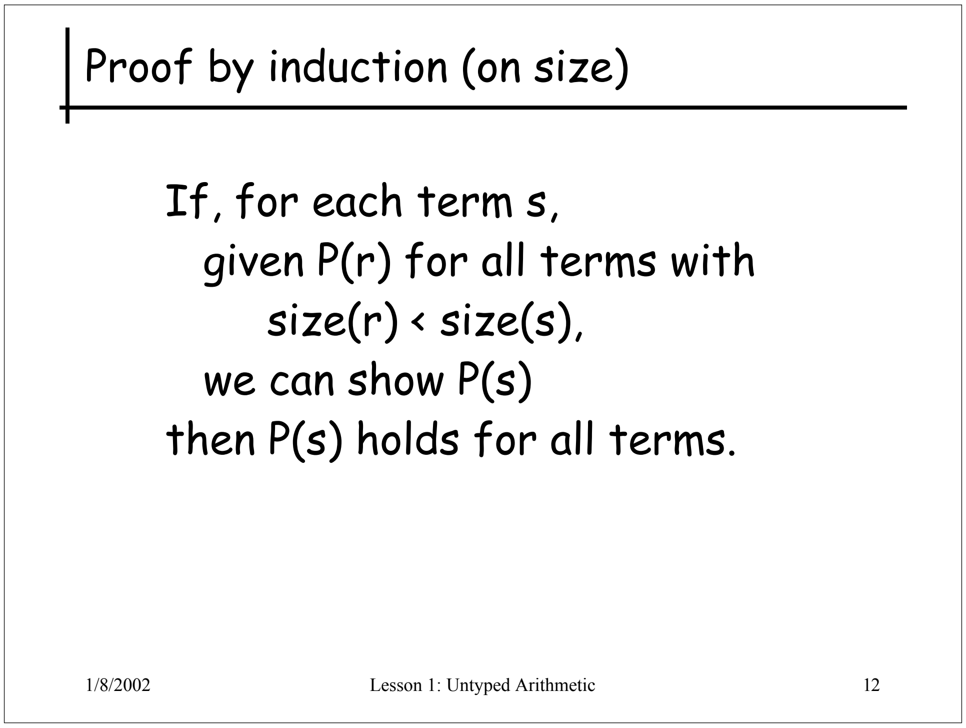
Depth of a term

```
depth(true) = 1
depth(false) = 1
depth(0) = 1
depth(succ t) = depth(t) + 1
depth(pred t) = depth(t) + 1
depth(iszero t) = depth(t) + 1
depth(if t1 then t2 else t3) =
  max(depth(t1),depth(t1),depth(t1)) + 1
```



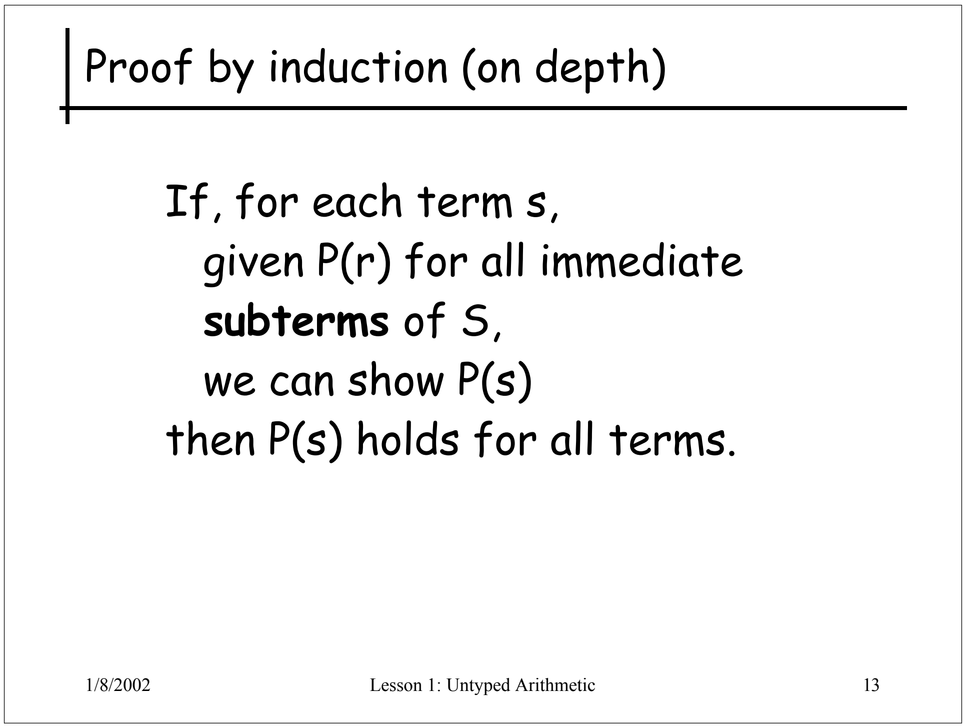
Proof by induction (on depth)

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



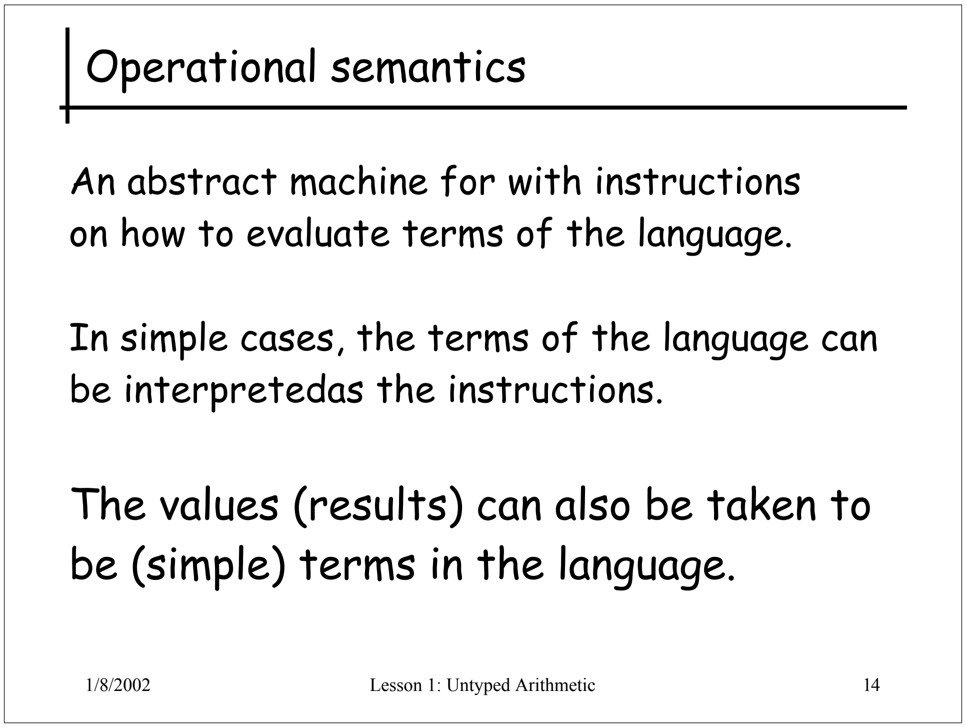
Proof by induction (on size)

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



Proof by induction (on depth)

If, for each term s,
given P(r) for all immediate
subterms of S,
we can show P(s)
then P(s) holds for all terms.

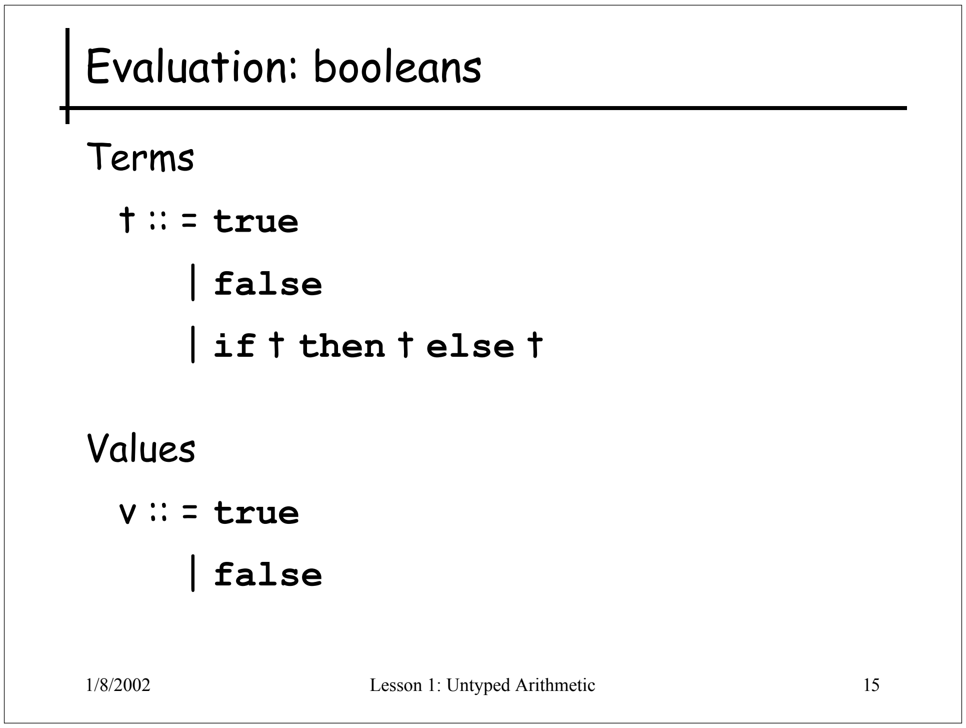


Operational semantics

An abstract machine for with instructions
on how to evaluate terms of the language.

In simple cases, the terms of the language can
be interpretedas the instructions.

The values (results) can also be taken to
be (simple) terms in the language.



Evaluation: booleans

Terms

t ::= **`true`**

  | **`false`**

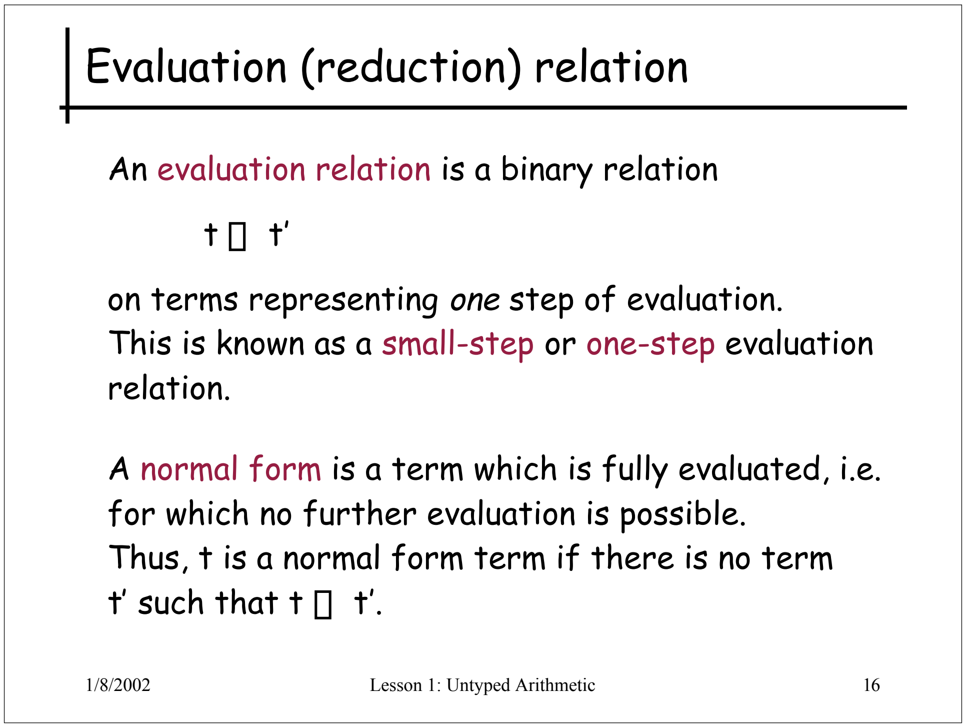
  | **`if`** t **`then`** t **`else`** t

Values

v ::= **`true`**

  | **`false`**

1/8/2002 Lesson 1: Untyped Arithmetic 15



Evaluation (reduction) relation

An evaluation relation is a binary relation

t □ t'

on terms representing *one* step of evaluation.
This is known as a small-step or one-step evaluation relation.

A normal form is a term which is fully evaluated, i.e. for which no further evaluation is possible.
Thus, t is a normal form term if there is no term t' such that t □ t'.

1/8/2002 Lesson 1: Untyped Arithmetic 16

Evaluation rules for boolean terms

The evaluation releation t □ t′ is the least relation satisfying the following rules.

```
if true then t2 else t3 □  t2

if false then t2 else t3 □  t3

                 t1 □  t1′
--------------------------------------------
if t1 then t2 else t3 □
   if t1′ then t2 else t3
```

Evaluation strategy

Evaluation rules can determine an evaluation strategy that limits where evaluation takes place.

Example:

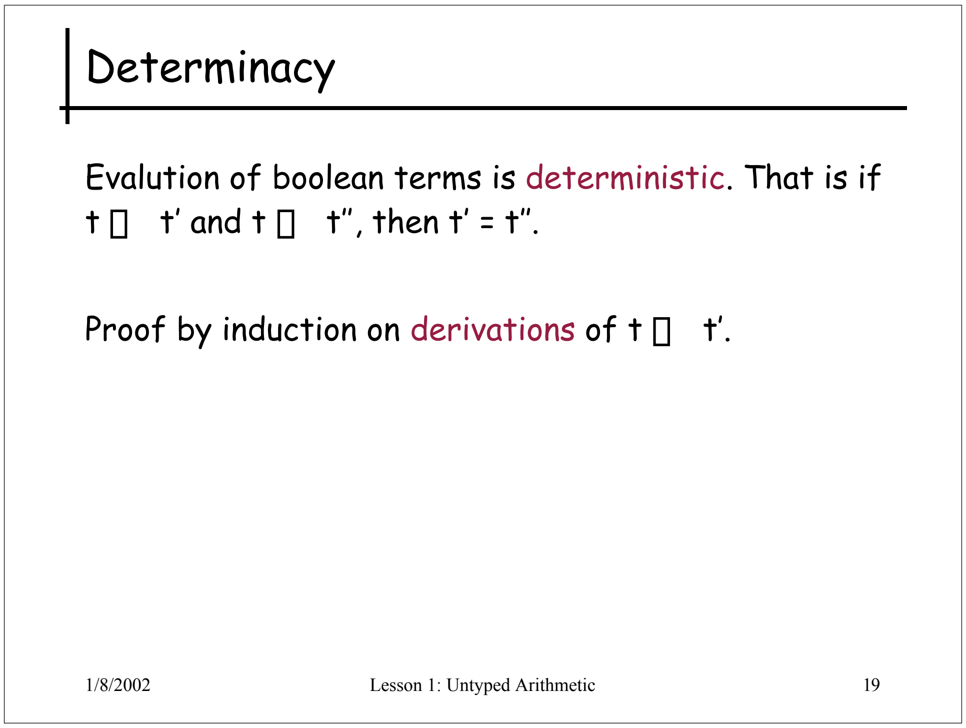
if true then (if false then false else true) else true

□ if false then false else true

But not

if true then (if false then false else true) else true

□ if true then true else true

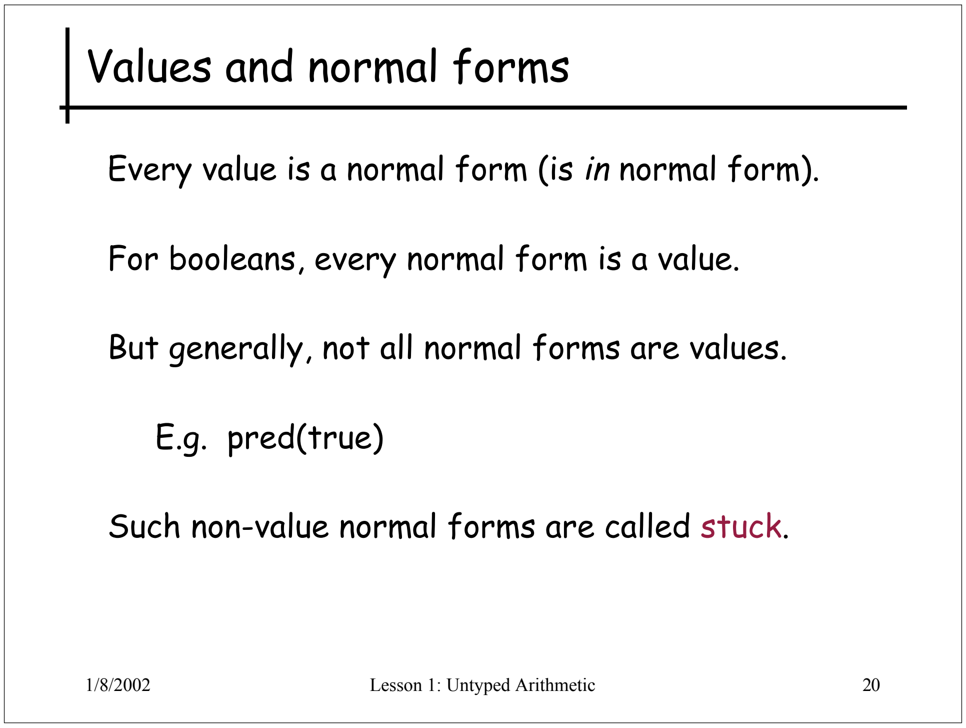


Determinacy

Evalution of boolean terms is deterministic. That is if $t \to t'$ and $t \to t''$, then $t' = t''$.

Proof by induction on derivations of $t \to t'$.

1/8/2002 Lesson 1: Untyped Arithmetic 19



Values and normal forms

Every value is a normal form (is *in* normal form).

For booleans, every normal form is a value.

But generally, not all normal forms are values.

E.g. pred(true)

Such non-value normal forms are called stuck.

1/8/2002 Lesson 1: Untyped Arithmetic 20

Multistep evaluation

Defn: Let $\rightarrow^*$ be the reflexive, transitive closure of $\rightarrow$. I.e $\rightarrow^*$ is the least reln such that

(1) if $t \rightarrow t'$ then $t \rightarrow^* t'$

(2) $t \rightarrow^* t$

(3) if $t \rightarrow^* t'$ and $t' \rightarrow^* t''$ then $t \rightarrow^* t''$

Boolean normal forms

Uniqueness of normal forms

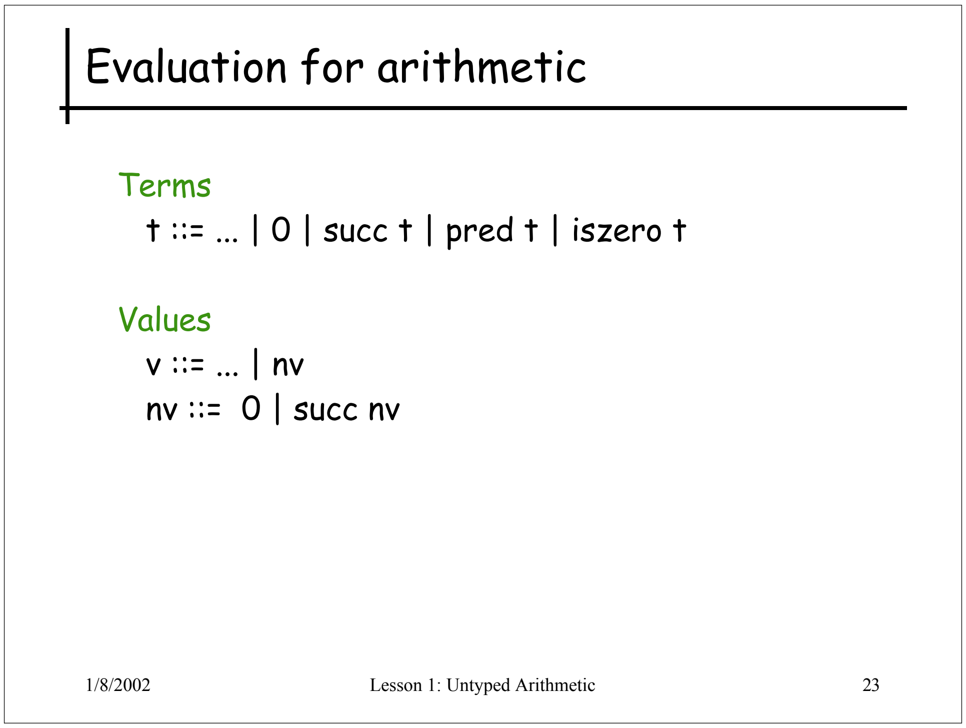
Theorem: If $t \rightarrow^* u$ and $t \rightarrow^* u'$ where u and u' are normal forms, then u = u'.

Proof: determinacy of $\rightarrow$

Existence of normal forms

Theorem: For any term t, there is a normal form u such that $t \rightarrow^* u$.

Proof: If $t \rightarrow t'$, then t' is smaller than t, i.e. size(t') < size(t).



Evaluation for arithmetic

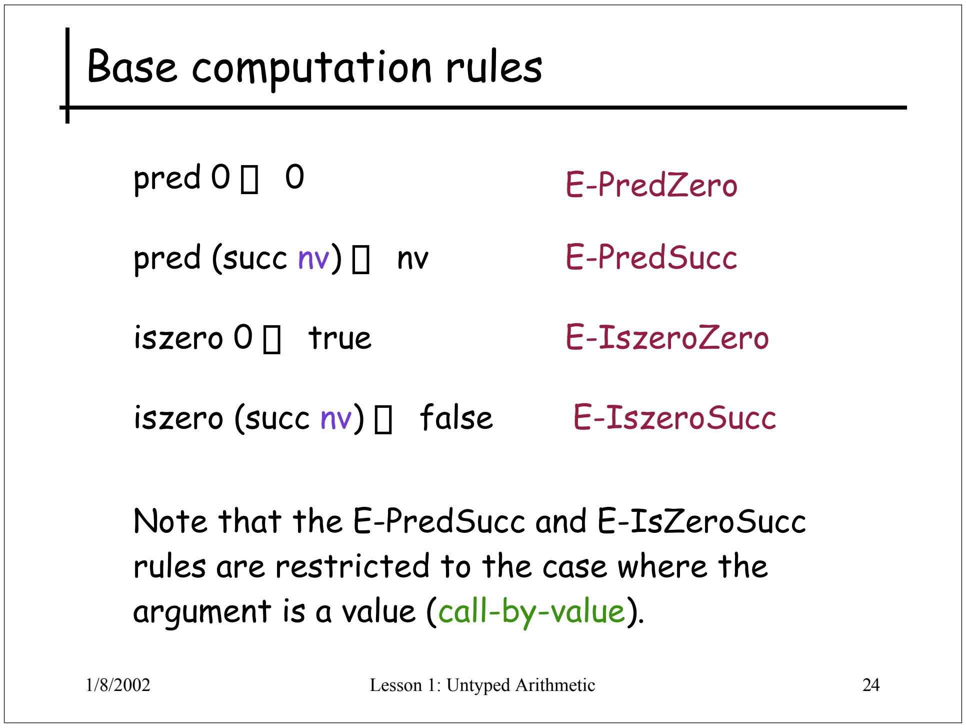
Terms

t ::= ... | 0 | succ t | pred t | iszero t

Values

v ::= ... | nv

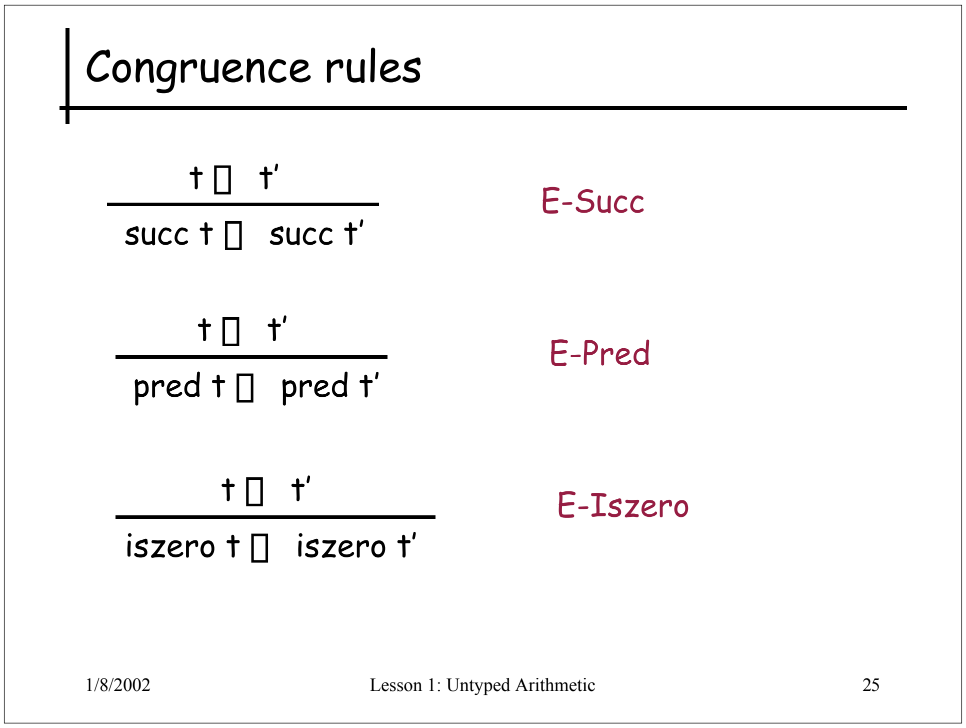
nv ::= 0 | succ nv



Base computation rules

pred 0 $\rightarrow$ 0	E-PredZero
pred (succ nv) $\rightarrow$ nv	E-PredSucc
iszero 0 $\rightarrow$ true	E-IszeroZero
iszero (succ nv) $\rightarrow$ false	E-IszeroSucc

Note that the E-PredSucc and E-IsZeroSucc rules are restricted to the case where the argument is a value (call-by-value).



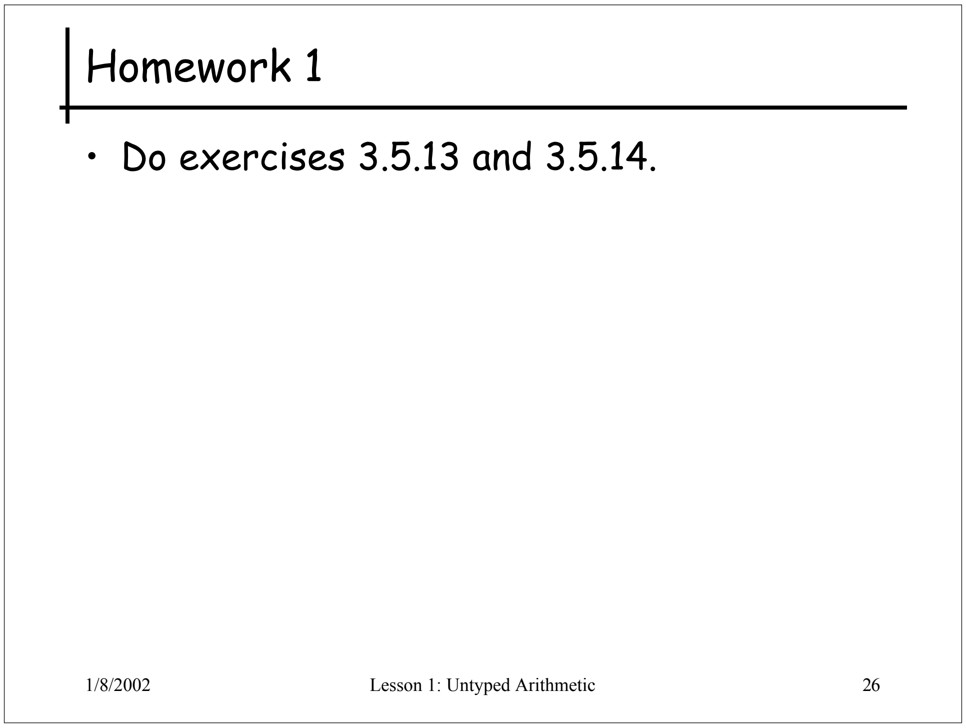
Congruence rules

$$\frac{t \longrightarrow t'}{\text{succ } t \longrightarrow \text{succ } t'} \qquad \text{E-Succ}$$

$$\frac{t \longrightarrow t'}{\text{pred } t \longrightarrow \text{pred } t'} \qquad \text{E-Pred}$$

$$\frac{t \longrightarrow t'}{\text{iszero } t \longrightarrow \text{iszero } t'} \qquad \text{E-Iszero}$$

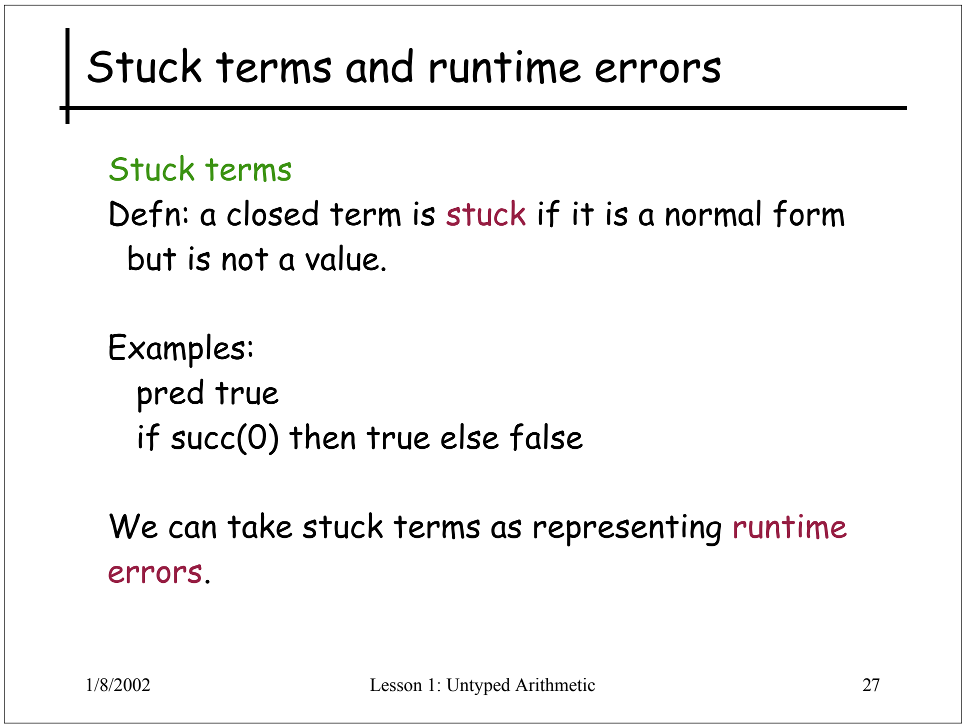
1/8/2002 Lesson 1: Untyped Arithmetic 25



Homework 1

- Do exercises 3.5.13 and 3.5.14.

1/8/2002 Lesson 1: Untyped Arithmetic 26



Stuck terms and runtime errors

Stuck terms

Defn: a closed term is stuck if it is a normal form but is not a value.

Examples:

pred true

if succ(0) then true else false

We can take stuck terms as representing runtime errors.