

G54FOP: Lecture 2

Abstract Syntax and Induction on Terms

Henrik Nilsson
University of Nottingham, UK

G54FOP: Lecture 2 – p.1/14

Example: MiniTriangle CFG (1)

Concrete syntax for MiniTriangle:

```

Program   → Command
Commands → Command
          | Command ; Commands
Command  → VarExpression := Expression
          | VarExpression ( Expressions )
          | if Expression then Command else Command
          | while Expression do Command
          | let Declarations in Command
          | begin Commands end
    
```

G54FOP: Lecture 2 – p.2/14

Example: MiniTriangle CFG (2)

```

Expressions → Expression
            | Expression , Expressions
Expression  → PrimaryExpression
            | Expression Operator PrimaryExpression
PrimaryExpression → IntegerLiteral
                 | VarExpression
                 | Operator PrimaryExpression
                 | ( Expression )
VarExpression  → Identifier
    
```

G54FOP: Lecture 2 – p.3/14

Example: MiniTriangle CFG (3)

```

Declarations → Declaration
            | Declaration ; Declarations
Declaration  → const Identifier : TypeDenoter = Expression
            | var Identifier : TypeDenoter
            | var Identifier : TypeDenoter := Expression
TypeDenoter  → Identifier
    
```

G54FOP: Lecture 2 – p.4/14

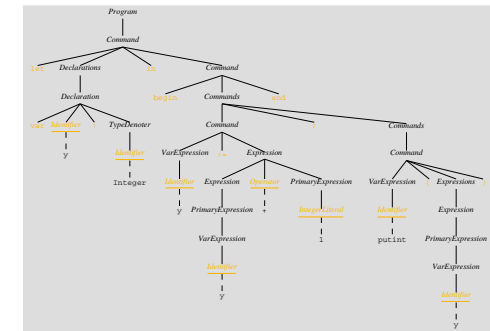
A MiniTriangle Program

```

let
  var y: Integer
in
  begin
    y := y + 1;
    putint(y)
  end
    
```

G54FOP: Lecture 2 – p.5/14

Parse Tree for the Program



G54FOP: Lecture 2 – p.6/14

MiniTriangle Abstract Syntax (1)

The details of the concrete syntax often obscure the essence of the structure of a program. In contrast, **abstract syntax** describe this directly:

Program	→ Command	Program
Command	→ Expression := Expression	CmdAssign
	Expression (Expression*)	CmdCall
	Command*	CmdSeq
	if Expression then Command	CmdIf
	else Command	
	while Expression do Command	CmdWhile
	let Declaration* in Command	CmdLet

G54FOP: Lecture 2 – p.7/14

MiniTriangle Abstract Syntax (2)

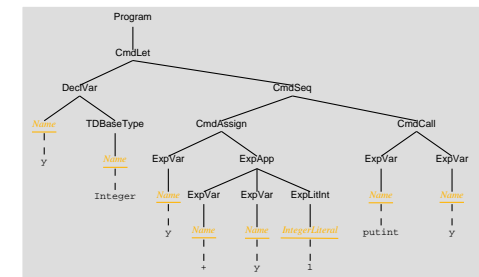
Expression	→ IntegerLiteral	ExpLitInt
	Name	ExpVar
	Expression (Expression*)	ExpApp
Declaration	→ const Name : TypeDenoter	DeclConst
	= Expression	
	var Name : TypeDenoter	DeclVar
	(:= Expression e)	
TypeDenoter	→ Name	TDBaseType

Note: Keywords and other fixed-spelling terminals serve only to make the connection with the concrete syntax clear.

Identifier ⊆ *Name*, *Operator* ⊆ *Name*

G54FOP: Lecture 2 – p.8/14

Abstract Syntax Tree for the Program



Key Point: The abstract syntax specifies **trees**, not strings.

G54FOP: Lecture 2 – p.9/14

Abstract Syntax for a Small Language

$t \rightarrow$	true	terms:	constant true
	false		constant false
	if t then t else t		conditional
	0		constant zero
	succ t		successor
	pred t		predecessor
	iszero t		zero test

Exercise: State some terms according to this grammar?

054FOP: Lecture 2 - p.10/14

Inductive Definition of Terms (2)

We will often use **inference rules** to state inductive definitions:

$$\mathbf{true} \in \tau \quad \mathbf{false} \in \tau \quad \mathbf{0} \in \tau$$

$$\frac{t_1 \in \tau}{\mathbf{succ } t_1 \in \tau} \quad \frac{t_1 \in \tau}{\mathbf{pred } t_1 \in \tau} \quad \frac{t_1 \in \tau}{\mathbf{iszero } t_1 \in \tau}$$

$$\frac{t_1 \in \tau \quad t_2 \in \tau \quad t_3 \in \tau}{\mathbf{if } t_1 \mathbf{ then } t_2 \mathbf{ else } t_3 \in \tau}$$

054FOP: Lecture 2 - p.13/14

Some Terms

Examples of terms:

- **true**
- **if true then 0 else succ 0**
- **if (iszero 0) then 0 else (succ (succ 0))**

Note: For **convenience**, terms will often be written as strings, with **parentheses where necessary** to make the structure clear. However, they **are** trees!

Exercise: Draw the last term above as a tree.

054FOP: Lecture 2 - p.10/14

Principles of Induction on Terms

1. Structural induction:

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

2. Induction on depth:

If, for each term s , given $P(r)$ for all r such that $\text{depth}(r) < \text{depth}(s)$ we can show $P(s)$, then $P(s)$ holds for all terms s .

3. Induction on size: similar to induction on depth, but using a suitable notion of size instead: $\text{size}(r) < \text{size}(s)$.

054FOP: Lecture 2 - p.14/14

Inductive Definition of Terms (1)

The set of trees described by the abstract syntax can be characterized by an **inductive definition**:

The set of terms is the **smallest** set τ such that:

1. $\{\mathbf{true}, \mathbf{false}, \mathbf{0}\} \in \tau$
2. if $t_1 \in \tau$, then $\{\mathbf{succ } t_1, \mathbf{pred } t_1, \mathbf{iszero } t_1\} \subseteq \tau$
3. if $t_1 \in \tau, t_2 \in \tau, t_3 \in \tau$, then $\{\mathbf{if } t_1 \mathbf{ then } t_2 \mathbf{ else } t_3\} \subseteq \tau$

This formulation makes it clear that the terms are an inductively defined set of **trees**.

054FOP: Lecture 2 - p.12/14