G54FOP: Lecture 2

Abstract Syntax and Induction on Terms

Henrik Nilsson

University of Nottingham, UK

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`

Example: MiniTriangle CFG (2)

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

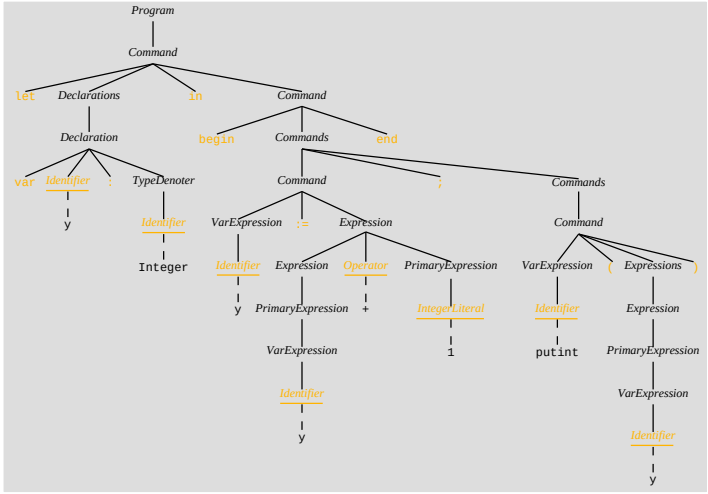
Example: MiniTriangle CFG (3)

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

A MiniTriangle Program

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


Parse Tree for the Program



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* `else` *Command*	CmdIf
	|	`while` *Expression* `do` *Command*	CmdWhile
	|	`let` *Declaration** `in` *Command*	CmdLet

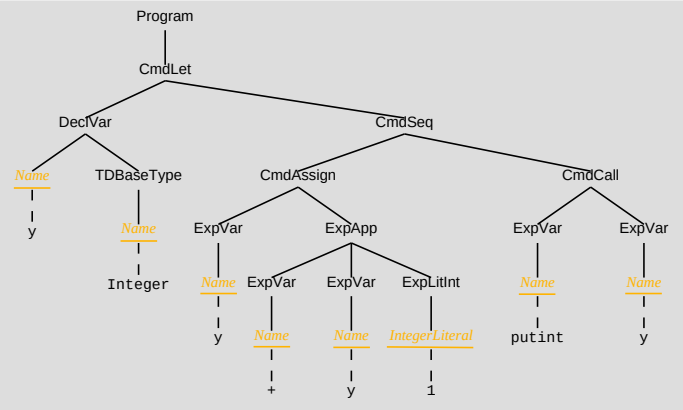
MiniTriangle Abstract Syntax (2)

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

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

Identifier $\subseteq$ *Name*, *Operator* $\subseteq$ *Name*

Abstract Syntax Tree for the Program



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

Abstract Syntax for a Small Language

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

Exercise: State some terms according to this grammar?

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.

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***.

Inductive Definition of Terms (2)

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

$$\texttt{true} \in \tau \qquad \texttt{false} \in \tau \qquad \texttt{0} \in \tau$$

$$\frac{t_1 \in \tau}{\texttt{succ}\ t_1 \in \tau} \qquad \frac{t_1 \in \tau}{\texttt{pred}\ t_1 \in \tau} \qquad \frac{t_1 \in \tau}{\texttt{iszero}\ t_1 \in \tau}$$

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

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 $\mathrm{depth}(r) < \mathrm{depth}(s)$ 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: $\mathrm{size}(r) < \mathrm{size}(s)$.