CS 365 — Programming Language Concepts

Pointers, Arrays and Structs

Mar 3, 2008

PoiJay Concrete Syntax (abreviated)

$$
\begin{array}{rcl}
\mathit{Type} & \rightarrow & \mathsf{int} \mid \mathsf{boolean} \mid \text{`}\ast\text{'}\ \mathit{Type} \\
\mathit{Factor} & \rightarrow & \mathit{Identifier} \mid \mathit{Literal} \mid (\ \mathit{Expression}\ ) \mid \\
 & & \&\ \mathit{Identifier} \mid \text{`}\ast\text{'}\ \mathit{Expression} \\
\mathit{Assignment} & \rightarrow & \text{`}\ast\text{'}\ \ast\ \mathit{Identifier} = \mathit{Expression}
\end{array}
$$

'RayJay Concrete Syntax (corrected)

Program	$\rightarrow$	*Declarations* { *Method* }* void main () '{' *Declarations Statements* '}'
Method	$\rightarrow$	*Type Identifier* ([*Parameters*]) '{' *Declarations Statements* '}'
Parameters	$\rightarrow$	*Parameter* { , *Parameter* }
Parameter	$\rightarrow$	*Type Identifier*
Declarations	$\rightarrow$	{ *Declaration* }*
Declaration	$\rightarrow$	*Type Identifiers;*
Type	$\rightarrow$	void | *PrimitiveType* | *ArrayType*
PrimitiveType	$\rightarrow$	int | boolean
ArrayType	$\rightarrow$	*PrimitiveType* { [] } +
Identifiers	$\rightarrow$	*Identifer* { , *Identifier* }*
Statements	$\rightarrow$	{ *Statement* }*
Statement	$\rightarrow$	; | *Block* | *Assignment* | *IfStatement* | *WhileStatement* | *PrintStatement* | *CallStatement* | *ReturnStatement*
Block	$\rightarrow$	'{' *Declarations Statements* '}'
Assignment	$\rightarrow$	*LeftHandSide* = *Expression* ;
LeftHandSide	$\rightarrow$	*Identifier* { [*Expression*] } +
IfStatement	$\rightarrow$	if (*Expression*) *Statement* { else *Statement* }$_{opt}$
WhileStatement	$\rightarrow$	while (*Experssion*) *Statement*
PrintStatement	$\rightarrow$	System.out.println (*Expression*) ;

'RayJay Concrete Syntax, continued

CallStatement → *Identifier* ([*Arguments*]);
Arguments → *Expression* { , *Expression* }*
ReturnStatement → return *Expression* ;
Expression → *Conjunction* { || *Conjunction* }*
Conjunction → *Relation* { && *Relation* }*
Relation → *Addition* { [| < | <= | > | >= | == | ! =] *Addition* }$_{opt}$
Addition → *Term* { [+ | -] *Term* }*
Term → *Negation* { ['*' | /] *Negation* }*
Negation → { ! }$_{opt}$ *Factor*
Factor → *Identifier* | *Literal* | (*Expression*) | *Call* |
IndexedVariable | *Creation*
Call → *Identifier* ([*Arguments*])
IndexedVariable → *Identifier* { [*Expression*] } +
Creation → new *PrimitiveType* { [*Expression*] } + { [] } *

'RayJay Abstract Syntax

Program	$\longrightarrow$	*Declaration*∗ *Method*∗ *Block*
Method	$\longrightarrow$	**Type Identifier** *Parameter*∗ *Block*
Parameter	$\longrightarrow$	**Type Identifier**
Declaration	$\longrightarrow$	**Type Identifier**∗
Type	$\longrightarrow$	*PrimitiveType* | *ArrayType*
PrimitiveType	$\longrightarrow$	**PrimType**
ArrayType	$\longrightarrow$	**PrimType Dimensions**
Statement	$\longrightarrow$	*Skip* | *Block* | *Assignment* | *Conditional* | *Loop* | *Print* | *CallStmt* | *Return*
Skip	$\longrightarrow$	
Block	$\longrightarrow$	*Declaration*∗ *Statement*∗
Assignment	$\longrightarrow$	*LeftHandSide* *Expression*
LeftHandSide	$\longrightarrow$	**Identifier** *Expression*∗
Conditional	$\longrightarrow$	*Expression* *Statement* *Statement*
Loop	$\longrightarrow$	*Expression* *Statement*
Print	$\longrightarrow$	*Expression*
CallStmt	$\longrightarrow$	**Identifier** *Expression*∗
Return	$\longrightarrow$	*Expression*

'RayJay Abstract Syntax, continued

Expression	→	*Variable* | *IntLitExpr* | *BoolLitExpr* | *BinaryExpr* | *UnaryExpr* | *Call* | *IndexedVariable* | *Creation*
Variable	→	**Identifier**
IntLitExpr	→	**IntLiteral**
BoolLitExpr	→	**BoolLiteral**
BinaryExpr	→	*Expression* **Operator** *Expression*
UnaryExpr	→	**Operator** *Expression*
Call	→	**Identifier** *Expression*∗
IndexedVariable	→	**Identifier** *Expression*∗
Creation	→	**PrimType** *Expression*∗ **IntLiteral**

RecJay Concrete Syntax

Program	$\longrightarrow$	{ *Class* }* *Declarations* { *Method* }*
		void main () '{' *Declarations Statements* '}'
Class	$\longrightarrow$	class *Identifier* '{' *Declarations* '}'
Method	$\longrightarrow$	*Type Identifier* ([*Parameters*]) '{' *Declarations Statements* '}'
Parameters	$\longrightarrow$	*Parameter* { , *Parameter* }
Parameter	$\longrightarrow$	*Type Identifier*
Declarations	$\longrightarrow$	{ *Declaration* }*
Declaration	$\longrightarrow$	*Type Identifiers;*
Type	$\longrightarrow$	void | int | boolean | *Identifier*
Identifiers	$\longrightarrow$	*Identifer* { , *Identifier* }*
Statements	$\longrightarrow$	{ *Statement* }*
Statement	$\longrightarrow$	; | *Block* | *Assignment* | *IfStatement* | *WhileStatement* | *PrintStatement* |
		CallStatement | *ReturnStatement*
Block	$\longrightarrow$	'{' *Declarations Statements* '}'
Assignment	$\longrightarrow$	*QualifiedVariable* = *Expression* ;
IfStatement	$\longrightarrow$	if (*Expression*) *Statement* { else *Statement* }$_{opt}$
WhileStatement	$\longrightarrow$	while (*Experssion*) *Statement*
PrintStatement	$\longrightarrow$	System.out.println (*Expression*) ;

RecJay Concrete Syntax (continued)

CallStatement	$\longrightarrow$	*Identifier* (*Arguments*);
Arguments	$\longrightarrow$	*Expression* { , *Expression* }*
ReturnStatement	$\longrightarrow$	return *Expression* ;
Expression	$\longrightarrow$	*Conjunction* { || *Conjunction* }*
Conjunction	$\longrightarrow$	*Relation* { && *Relation* }*
Relation	$\longrightarrow$	*Addition* { [| < | <= | > | >= | == | ! =] *Addition* }$_{opt}$
Addition	$\longrightarrow$	*Term* { [+ | -] *Term* }*
Term	$\longrightarrow$	*Negation* { ['*' | /] *Negation* }*
Negation	$\longrightarrow$	{ ! }$_{opt}$ *Factor*
Factor	$\longrightarrow$	*Identifier* | *Literal* | (*Expression*) | *Call* | *QualifiedVariable* | *Instantiation*
Call	$\longrightarrow$	*Identifier* (*Arguments*)
QualifiedVariable	$\longrightarrow$	{ *Identifier* . } * *Identifier*
Instantiation	$\longrightarrow$	new *Identifier* ()

RecJay Abstract Syntax

Program	→	*Class** *Declaration** *Method** *Block*
Class	→	*Identifier* *Declaration* *
Method	→	**Type Identifier** *Parameter** *Block*
Parameter	→	**Type Identifier**
Declaration	→	**Type Identifier***
Statement	→	*Skip* | *Block* | *Assignment* | *Conditional* | *Loop* | *Print* | *CallStmt* | *Return*
Skip	→	
Block	→	*Declaration** *Statement**
Assignment	→	**Identifier* Identifier** *Expression*
Conditional	→	*Expression* *Statement* *Statement*
Loop	→	*Expression* *Statement*
Print	→	*Expression*
CallStmt	→	**Identifier** *Expression**
Return	→	*Expression*

RecJay Abstract Syntax (continued)

Expression	$\longrightarrow$	*Variable* | *IntLitExpr* | *BoolLitExpr* | *BinaryExpr* | *UnaryExpr* | *Call* | *QualifiedVariable* | *Instantiation*
Variable	$\longrightarrow$	**Identifier**
IntLitExpr	$\longrightarrow$	**IntLiteral**
BoolLitExpr	$\longrightarrow$	**BoolLiteral**
BinaryExpr	$\longrightarrow$	*Expression* **Operator** *Expression*
UnaryExpr	$\longrightarrow$	**Operator** *Expression*
Call	$\longrightarrow$	**Identifier** *Expression*$*$
QualifiedVariable	$\longrightarrow$	**Identifier**$*$ **Identifier**
Instantiation	$\longrightarrow$	**Identifier**