Oat v. 1 Language Specification

CIS341 – Steve Zdancewic

March 1, 2018

1 Grammar

The following grammar defines the Oat syntax. All binary operations are *left associative* with precedence levels indicated numerically. Higher precedence operators bind tighter than lower precedence ones.

prog	::=		prog
	|	$decl_1 \,..\, decl_i$	
decl	::=		global declarations
	|	*gdecl*	
	|	*fdecl*	
gdecl	::=		global variable declarations
	|	`global` *id* = *gexp* `;`	
arg	::=		arg
	|	*t id*	
args	::=		args
	|	$arg_1, \,..\,, arg_i$	
fdecl	::=		function declaration
	|	*t id*(*args*) *block*	
block	::=		blocks
	|	`{`$stmt_1 \,..\, stmt_i$`}`	
t	::=		types
	|	`int`	
	|	`bool`	
	|	*ref*	

ref	::=		reference types
	|	`string`	
	|	*t* `[]`	
	|	*fty*	

fty	::=		function types
	|	$(t_0, .., t_i) \rightarrow retty$	
retty	::=		return types
	|	`void`	
	|	t	
bop	::=		(left associative) binary operations
	|	`*`	precedence 100
	|	$+$	precedence 90
	|	`-`	precedence 90
	|	`<<`	precedence 80
	|	`>>`	precedence 80
	|	`>>>`	precedence 80
	|	`<`	precedence 70
	|	`<=`	precedence 70
	|	`>`	precedence 70
	|	`>=`	precedence 70
	|	`==`	precedence 60
	|	`!=`	precedence 60
	|	`&`	precedence 50
	|	`|`	precedence 40
	|	`[&]`	precedence 30
	|	`[|]`	precedence 20
uop	::=		unary operations
	|	`-`	
	|	`!`	
	|	`~`	
gexp	::=		global initializers
	|	n	
	|	s	
	|	t `null`	
	|	`true`	
	|	`false`	
	|	t`[]`$\{gexp_1, .., gexp_i\}$	
lhs	::=		lhs expressions
	|	id	
	|	$exp1[exp2]$	

exp	::=		expressions
	|	*id*	
	|	*n*	
	|	*s*	
	|	*t* `null`	
	|	`true`	
	|	`false`	
	|	exp_1 `[`exp_2`]`	
	|	id`(`exp_1`, .. ,`exp_i`)`	
	|	`new` *t* `[]{`exp_1`, .. ,`exp_i`}`	
	|	`new` *t* `[`exp_1`]`	
	|	exp_1 *bop* exp_2	
	|	*uop exp*	
	|	*lhs*	
	|	*gexp*	
	|	`(`*exp*`)`	
vdecl	::=		local declarations
	|	`var` *id* `=` *exp*	
vdecls	::=		decl list
	|	$vdecl_1$`, .. ,`$vdecl_i$	
stmt	::=		statements
	|	*lhs* `=` *exp* `;`	
	|	*vdecl* `;`	
	|	`return` *exp* `;`	
	|	`return ;`	
	|	id`(`exp_1`, .. ,`exp_i`);`	
	|	*if_stmt*	
	|	`for(`*vdecls* `;` *exp_opt* `;` *stmt_opt*`)` *block*	
	|	`while(`*exp*`)` *block*	
if_stmt	::=		if statements
	|	`if(`*exp*`)` *block else_stmt*	
else_stmt	::=		else
	|	ϵ	
	|	`else` *block*	
	|	`else` *if_stmt*	