Oat_0 Scoping Rules

February 18, 2013

1 Syntactic Definitions

exp	::=		expressions
	|	x	
	|	$int32$	
	|	$exp_1\ bop\ exp_2$	
	|	`0` M	
stmt	::=		statements
	|	$x = exp;$	
	|	`if` $(exp) stmt_1$ `else` $stmt_2$	
	|	$\{block\}$	
stmts	::=		
	|	*stmt stmts*	
	|		
vdecl	::=		variable declaration
	|	`int` $x = exp;$	
vdecls	::=		vdecls
	|	*vdecl vdecls*	
	|		
block	::=		blocks
	|	*vdecls stmts*	
prog	::=		programs
	|	*block* `return` $(exp);$	

2 Scope Checking

$\mathtt{G}$	::=		contexts
	|	$[]$	
	|	x::$\mathtt{G}$	
Scoping	::=		
	|	$\mathtt{G} \vdash exp$	Expressions
	|	$\mathtt{G} \vdash stmt$	Statements
	|	$\mathtt{G} \vdash stmts$	Sequences
	|	$\mathtt{G}_1 \vdash vdecls \Rightarrow \mathtt{G}_2$	Declaration lists
	|	$\mathtt{G}_1 \vdash block \Rightarrow \mathtt{G}_2$	Blocks
	|	$\vdash prog$	Programs

2.1 Inference Rules

$\boxed{\mathtt{G} \vdash exp}$ Expressions

$$\frac{}{\mathtt{G} \vdash int32} \textsc{Int}$$

$$\frac{x \in \mathtt{G}}{\mathtt{G} \vdash x} \textsc{Var}$$

$$\frac{\mathtt{G} \vdash exp_1 \quad \mathtt{G} \vdash exp_2}{\mathtt{G} \vdash exp_1\ bop\ exp_2} \textsc{Bop}$$

$\boxed{\mathtt{G} \vdash stmt}$ Statements

$$\frac{x \in \mathtt{G} \quad \mathtt{G} \vdash exp}{\mathtt{G} \vdash x = exp;} \textsc{Assign}$$

$$\frac{\mathtt{G} \vdash exp \quad \mathtt{G} \vdash stmt_1 \quad \mathtt{G} \vdash stmt_2}{\mathtt{G} \vdash \mathtt{if}\,(exp)stmt_1\,\mathtt{else}\,stmt_2} \textsc{IfElse}$$

$$\frac{\mathtt{G}_1 \vdash block \Rightarrow \mathtt{G}_2}{\mathtt{G}_1 \vdash \{block\}} \textsc{SBlock}$$

$\boxed{\mathtt{G} \vdash stmts}$ Sequences

$$\frac{}{\mathtt{G} \vdash} \textsc{empty}$$

$$\frac{\mathtt{G} \vdash stmt \quad \mathtt{G} \vdash stmts}{\mathtt{G} \vdash stmt\ stmts} \textsc{Seq}$$

$\boxed{\mathtt{G}_1 \vdash vdecls \Rightarrow \mathtt{G}_2}$ Declaration lists

$$\frac{}{\mathtt{G} \vdash \Rightarrow \mathtt{G}} \textsc{Done}$$

$$\frac{\mathtt{G}_1 \vdash exp \quad x::\mathtt{G}_1 \vdash vdecls \Rightarrow \mathtt{G}_2}{\mathtt{G}_1 \vdash \mathtt{int}\,x = exp; vdecls \Rightarrow \mathtt{G}_2} \textsc{Decls}$$

$\boxed{\mathtt{G}_1 \vdash block \Rightarrow \mathtt{G}_2}$ Blocks

$$\frac{\mathtt{G}_1 \vdash \mathit{vdecls} \Rightarrow \mathtt{G}_2 \quad \mathtt{G}_2 \vdash \mathit{stmts}}{\mathtt{G}_1 \vdash \mathit{vdecls}\ \mathit{stmts} \Rightarrow \mathtt{G}_2}\ \textsc{Block}$$

$\boxed{\vdash \mathit{prog}}$ Programs

$$\frac{[] \vdash \mathit{block} \Rightarrow \mathtt{G} \quad \mathtt{G} \vdash \mathit{exp}}{\vdash \mathit{block}\ \mathtt{return}\,(\mathit{exp});}\ \textsc{Prog}$$

Example of a successful scope checking derivation. Note how the structure of this is isomorphic to the structure of the recursive calls in the equivalent OCaml implementation.

$$\dfrac{\dfrac{}{[] \vdash \mathtt{0}} \quad \dfrac{\dfrac{\dfrac{\dfrac{x_1 \in x_1::[]}{x_1::[] \vdash x_1} \quad \dfrac{x_1 \in x_1::[]}{x_1::[] \vdash x_1}}{x_1::[] \vdash x_1 + x_1} \quad \dfrac{}{x_2::x_1::[] \vdash\Rightarrow x_2::x_1::[]}}{x_1::[] \vdash \mathtt{int}\ x_2 = x_1 + x_1;\ \Rightarrow x_2::x_1::[]}}{[] \vdash \mathtt{int}\ x_1 = \mathtt{0}; \mathtt{int}\ x_2 = x_1 + x_1;\ \Rightarrow x_2::x_1::[]} \quad \dfrac{\dfrac{x_1 \in x_2::x_1::[]}{x_2::x_1::[] \vdash x_1} \quad \dfrac{\dfrac{x_1 \in x_2::x_1::[]}{x_2::x_1::[] \vdash x_1} \quad \dfrac{x_2 \in x_2::x_1::[]}{x_2::x_1::[] \vdash x_2}}{x_2::x_1::[] \vdash x_1 - x_2} \quad \dfrac{x_1 \in x_2::x_1::[]}{x_2::x_1::[] \vdash x_1}}{x_2::x_1::[] \vdash x_1 = x_1 - x_2;}}{\vdash \mathtt{int}\ x_1 = \mathtt{0}; \mathtt{int}\ x_2 = x_1 + x_1;\ x_1 = x_1 - x_2;\ \mathtt{return}\,(x_1);}$$

Example of an unsuccessful scope checking derivation—the check $x_1 \in []$ fails, which corresponds to the `lookup` function raising an exception:

$$\dfrac{\dfrac{\dfrac{\dfrac{x_1 \in [] \quad !!FAILS}{[] \vdash x_1}}{[] \vdash x_1 + x_1}}{[] \vdash \mathtt{int}\ x_2 = x_1 + x_1;\ \Rightarrow x_2::[]}}{\vdash \mathtt{int}\ x_2 = x_1 + x_1;\ \mathtt{return}\,(x_2);}$$