XS Type Checking

1. Notation

- Γ is a type environment mapping XS identifiers to types.
- $\Gamma \vdash E : T$ means that an expression E has type T in Γ (read as Γ yields E of type T)
- $\Gamma \vdash S$ means that a statement S is soundly typed in Γ (read as Γ yields S)
- $$(\texttt{xsTcCase}) \quad \frac{C_1 \quad C_2}{S_1}$$

  is read as $C_1 \wedge C_2 \implies S_1$

2. Type Checking For Expressions

2.1. Literals

let L denote a literal

$$(\texttt{xsTcInt}) \quad \frac{L \ \in \{x | -999,999,999 \leq x \leq 999,999,999 \wedge x \in \mathbb{Z}\}}{\Gamma \vdash L : \texttt{int}}$$

(XS `int` literals may not be more than 9 digits long. yES)

$$(\texttt{xsTcStr}) \quad \frac{L \ \texttt{is a " delimited sequence of characters}}{\Gamma \vdash L : \texttt{string}}$$

$$(\texttt{xsTcFloat}) \quad \frac{L \ \in \mathbb{R}}{\Gamma \vdash L : \texttt{float}}$$

$$(\texttt{xsTcBool}) \quad \frac{L \ \in \{\texttt{true, false}\}}{\Gamma \vdash L : \texttt{bool}}$$

$$(\texttt{xsTcVec}) \quad \frac{L \mid L \coloneqq \texttt{vector}(x, y, z) \quad \Gamma \vdash x : \texttt{int} \mid \texttt{float} \quad \Gamma \vdash y : \texttt{int} \mid \texttt{float} \quad \Gamma \vdash z : \texttt{int} \mid \texttt{float}}{\Gamma \vdash L : \texttt{vector}}$$

Note: x, y, and z must also be literals

2.2. Identifier

let X be an identifier

$$(\texttt{xsTcId}) \quad \frac{(X, T) \ \in \Gamma}{\Gamma \vdash X : T}$$

2.3. Parenthesis

$$(\texttt{xsTcParen}) \quad \frac{\Gamma \vdash E : T}{\Gamma \vdash (E) : T}$$

2.4. Function Call (Expression)

$$(\texttt{xsTcFncExpr})\quad \frac{(\texttt{fnName}, (T_1, ..., T_n) \to T_r) \in \Gamma \quad \Gamma \vdash E_i : T_i \mid \texttt{void} \quad n \leq 12}{\Gamma \vdash \texttt{fnName}(\texttt{E}_1, ..., \texttt{E}_\texttt{n}) : T_r}$$

Note: XS can have functions of arity $\in [0, 12]$ but every function must define default values for each parameter, which means it is possible to omit any number of its arguments which are initialised with defaults in that case. Hence the `T_n | void` type for parameters. There is no currying in XS :(

2.5. Operations

$$(\texttt{xsTcArithInt})\quad \frac{\Gamma \vdash E_1 : \texttt{int} \quad \Gamma \vdash E_2 : \texttt{int} \mid \texttt{float} \quad \text{OP } \in \{+,\ -,\ *,\ /,\ \%\}}{\Gamma \vdash E_1 \text{ OP } E_2 : \texttt{int}}$$

(an `int op float` is an `int` in XS… yES.)

$$(\texttt{xsTcArithFloat})\quad \frac{\Gamma \vdash E_1 : \texttt{float} \quad \Gamma \vdash E_2 : \texttt{int} \mid \texttt{float} \quad \text{OP } \in \{+,\ -,\ *,\ /,\ \%\}}{\Gamma \vdash E_1 \text{ OP } E_2 : \texttt{float}}$$

$$(\texttt{xsTcStrConc1})\quad \frac{\Gamma \vdash E_1 : \texttt{string} \quad \Gamma \vdash E_2 : \texttt{int} \mid \texttt{float} \mid \texttt{bool} \mid \texttt{string} \mid \texttt{vector}}{\Gamma \vdash E_1 + E_2 : \texttt{string}}$$

$$(\texttt{xsTcStrConc2})\quad \frac{\Gamma \vdash E_1 : \texttt{int} \mid \texttt{float} \mid \texttt{bool} \mid \texttt{string} \mid \texttt{vector} \quad \Gamma \vdash E_2 : \texttt{string}}{\Gamma \vdash E_1 + E_2 : \texttt{string}}$$

$$(\texttt{xsTcRelnNum})\quad \frac{\Gamma \vdash E_1 : \texttt{int} \mid \texttt{float} \quad \Gamma \vdash E_2 : \texttt{int} \mid \texttt{float} \quad \text{OP } \in \{<,\ <=,\ >,\ >=,\ ==,\ !=\}}{\Gamma \vdash E_1 \text{ OP } E_2 : \texttt{bool}}$$

$$(\texttt{xsTcRelnStr})\quad \frac{\Gamma \vdash E_1 : \texttt{string} \quad \Gamma \vdash E_2 : \texttt{string} \quad \text{OP } \in \{<,\ <=,\ >,\ >=,\ ==,\ !=\}}{\Gamma \vdash E_1 \text{ OP } E_2 : \texttt{bool}}$$

Note: strings are compared lexically

$$(\texttt{xsTcEqVec})\quad \frac{\Gamma \vdash E_1 : \texttt{vector} \quad \Gamma \vdash E_2 : \texttt{vector} \quad \text{OP } \in \{==,\ !=\}}{\Gamma \vdash E_1 \text{ OP } E_2 : \texttt{bool}}$$

$$(\texttt{xsTcEqBool})\quad \frac{\Gamma \vdash E_1 : \texttt{bool} \quad \Gamma \vdash E_2 : \texttt{bool} \quad \text{OP } \in \{==,\ !=\}}{\Gamma \vdash E_1 \text{ OP } E_2 : \texttt{bool}}$$

Note: Trying relational operators on vectors and booleans passes the in game type checker, but will cause a silent XS crash.

$$(\texttt{xsTcLogical})\quad \frac{\Gamma \vdash E_1 : \texttt{bool} \quad \Gamma \vdash E_2 : \texttt{bool} \quad \text{OP } \in \{\&\&,\ \texttt{||}\}}{\Gamma \vdash E_1 \text{ OP } E_2 : \texttt{bool}}$$

3. Type Checking For Statements

3.1. Sequence

$$(\texttt{xsTcSeq})\quad \frac{\Gamma \vdash S \quad \Gamma \vdash \bar{S}}{\Gamma \vdash S\bar{S}}$$

3.2. Include

$$(\texttt{xsTcInc}) \quad \frac{\Gamma \vdash S : \texttt{string}}{\Gamma \vdash \texttt{include}\ S;}$$

3.3. Var Def/Assign

$$(\texttt{xsTcAssign}) \quad \frac{\Gamma \vdash X : T \quad \Gamma \vdash E : T}{\Gamma \vdash X\ =\ E;}$$

Note: floats may be assigned to ints - they follow the expected casting rules.

3.4. If Else

$$(\texttt{xsTcIf}) \quad \frac{\Gamma \vdash E_c : \texttt{bool} \quad \Gamma \vdash \bar{S_1} \quad \Gamma \vdash \bar{S_2}}{\Gamma \vdash \texttt{if}\ (E_c)\ \{\ \bar{S_1}\ \}\ \texttt{else}\ \{\ \bar{S_2}\ \}}$$

3.5. While

$$(\texttt{xsTcWhile}) \quad \frac{\Gamma \vdash E_c : \texttt{bool} \quad \Gamma \vdash \bar{S}}{\Gamma \vdash \texttt{while}\ (E_c)\ \{\ \bar{S}\ \}}$$

3.6. For

$$(\texttt{xsTcFor}) \quad \frac{\Gamma \vdash E_1 : \texttt{int} \mid \texttt{float} \mid \texttt{bool} \quad \Gamma \vdash E_2 : \texttt{int} \mid \texttt{float} \quad \Gamma \vdash \bar{S} \quad \mathrm{OP}\ \in \{<,\ <=,\ >,\ >=\}}{\Gamma \vdash \texttt{for}\ (X\ =\ E_1;\ \mathrm{OP}\ E_2)\ \{\ \bar{S}\ \}}$$

Note: Floats may be used in E_1 or E_2 they will be cast to int before being used by the loop. Bools in the initializer are cast to int, bools in the conditional pass the in game type checker but will cause a silent XS crash.

3.7. Switch

$$(\texttt{xsTcSwitch}) \quad \frac{\Gamma \vdash E_c : \texttt{int} \mid \texttt{float} \mid \texttt{bool} \quad \Gamma \vdash E_i : \texttt{int} \mid \texttt{float} \quad \Gamma \vdash \bar{S_i} \quad \Gamma \vdash \bar{S_d}}{\Gamma \vdash \texttt{switch}\ (E_c)\ \{\ \texttt{case}\ E_1\ \texttt{:}\ \ \{\ \bar{S_1}\ \}\ ...\ \texttt{case}\ E_n\ \texttt{:}\ \ \{\ \bar{S_n}\ \}\ \texttt{default}\ \ \texttt{:}\ \ \{\ \bar{S_d}\ \}\ \}}$$

Note: Floats may be used in E_c or E_n they will be cast to int before being used by the cases. Bools in the expression are cast to int, bools in a case's expression pass the in game type checker but will cause a silent XS crash.

3.8. Break, Continue, Break Point

$$\Gamma \vdash \texttt{break;}$$

$$\Gamma \vdash \texttt{continue;}$$

$$\Gamma \vdash \texttt{breakpoint;}$$

3.9. Function Definitions

$$(\texttt{xsTcFn}) \quad \frac{T_r \in \{\texttt{int},\ \texttt{float},\ \texttt{bool},\ \texttt{string},\ \texttt{vector},\ \texttt{void}\} \quad \Gamma \vdash E_i : T_i \quad n \leq 12 \quad \Gamma \vdash \bar{S} \quad \texttt{return}\ \mathtt{E_r};\in \bar{S} \quad \Gamma \vdash E_r : T}{\Gamma \vdash T_r\ \texttt{fnName}(T_1\ id_1\ =\ E_1,\ ...,\ T_n\ id_n\ =\ E_n)\ \{\ \bar{S}\ \}}$$

If the return type of a function is `void`, the return statement may be omitted

3.10 Rule Definitions

$$(\texttt{xsTcFn}) \quad \frac{\Gamma \vdash \bar{S}}{\Gamma \vdash \texttt{rule}\ \texttt{ruleName}\ \texttt{ruleOpts}\ \ \{\ \bar{S}\ \}}$$

3.11. Postfix

$$(\texttt{xsTcPost}) \quad \frac{(\texttt{id},\ \texttt{int | float}) \in \Gamma \quad \#\# \ \in \{++,\ --\}}{\Gamma \vdash \text{id}\#\#;}$$

Postfixes are ~~expressions~~ statements in XS. yES

3.12. Label, Goto

$$\Gamma \vdash \texttt{label id};$$

$$\Gamma \vdash \texttt{goto id};$$

3.13. Function Call (Statement)

($\texttt{xsTcFncStmt}$) same as 2.4. Function Call (Expression) with a terminating semicolon.

3.14. Class Definition

$$(\texttt{xsTcClsDef}) \quad \frac{\Gamma \vdash E_i : T_i}{\Gamma \vdash \texttt{class clsName}\ \{\ T_1\ id_1\ =\ E_1;\ ...\ T_n\ id_n\ =\ E_n;\ \};}$$

Classes are unused in XS and can't be instantiated afaik. This exists purely for completeness' sake.