KOOL-TYPED-STATIC

KOOL | Types | Static Typing Bugs and Traces Proto-Semantics (syntax, checkers/lint) | v.0.0.1 - 0.0.3

Abstract

This is the complete semantics of the core KOOL language. It extends the core semantics of typed SMPL/C with some constructs for the object-oriented extension. After the static semantics of some of the existing SMPL/C constructs (such as the type system and the rules for lvalues and rvalues) are reviewed, the new constructs are introduced. The new constructs include classes, inheritance, and dynamic dispatch. The static semantics of these constructs are defined in terms of a set of checkers, which are written in a variant of the lint DSL. A set of checkers for the existing SMPL/C constructs is also provided, so that the complete set of checkers for KOOL can be generated.

- Each class C defines a heterogeneous type, which can be implicitly used in programs in type variables, and explicitly, using the keyword `class` followed by the class name.
- Each class C may extend another type, so that the type system and the SMPL/C assignment rule (which allows for a subtype to be assigned to a variable of a supertype) can be used with class types.
- Cross-variables and field types (which may be type variables) should also allow such a type substitution. Such a type substitution should be allowed in the assignment rule, and in the rules for function calls, so that a subtype can be used in place of a supertype in these contexts, as in other contexts, to support polymorphism and dynamic dispatch.
- A set identifier is not bound to the level type environment, it will be attached to the current class type environment. Thus, a set identifier can be used in any method of the current class, or any method of a subclass of the current class. But this set is not accessible in a superclass or a sibling class.
- The assignment allows variables to be assigned values of some variable types. The result type of the assignment is the type of the variable on the left-hand side. This allows for a variable to be assigned a value of a subtype of its type.
- Constructors are defined in the SMPL/C as other functions, and calling constructors is like a function. Also, in the SMPL/C, it is up to the user to initialize the type of the destructor location before the type of the complex variable. Access to the destructor location is also allowed in the SMPL/C, and it is up to the user to make sure that the destructor location is initialized before it is accessed. In the KOOL, the rules are the same, but the rules for a class type are that the class is initialized automatically, and the destructor location is not accessible to the user. Instead, the destructor location is accessed by the keyword `this`.
- Class destructors are not allowed to have any values in their normal contexts. Such a value would lead to a memory leak, or to a variable in the heap that is not accessible to the program, or to a dangling pointer.
- Methods overriding other methods should be in the type subtyping relationship with the overridden methods, so cannot be a heterogeneous extension of the overridden method.

```
KOOL-TYPED-STATIC-SYNTAX
```

Syntax

The syntax of typed-KOOL (TYPE) is identical to that of dynamically-typed KOOL. For both languages, a input file (.kool) is processed by the KOOL parser, which produces a parse tree. The parse tree is then processed by a set of checkers, which are written in a variant of the lint DSL. The checkers are specific to the syntax of KOOL-dynamic, so a set of checkers for the syntax of typed-KOOL is provided.

```
KOOL-TYPED-STATIC-SYNTAX
```

Types

```
KOOL-TYPED-STATIC-SYNTAX
Type -> int
        char
        bool
        void
        string
        class IdentifierClass
        class IdentifierClass[]
        Type[]
        Type[Decimal]
KOOL-TYPED-STATIC-SYNTAX
```

Declarations

```
KOOL-TYPED-STATIC-SYNTAX
Decl -> Type Id
        Decl, Decl
KOOL-TYPED-STATIC-SYNTAX
Decl -> Decl = Expr
        Decl = {Expr,Expr}
        Decl = Decl[Decimal]
        Decl = Decl[Decimal,Decimal]
KOOL-TYPED-STATIC-SYNTAX
```

Expressions

```
KOOL-TYPED-STATIC-SYNTAX
Expr -> Id
        Decl
        Expr
        Expr[Expr]
        Expr[Expr,Expr]
        Expr[Expr:Expr]
        Expr[Expr:Expr:Expr]
        Expr[Expr:Expr:Expr:Expr]
        Expr[Expr:Expr:Expr:Expr:Expr]
        Expr[:Expr]
        Expr[:Expr:Expr]
        Expr[:Expr:Expr:Expr]
        Expr[:Expr:Expr:Expr:Expr]
        Expr[Expr:]
        Expr[Expr:Expr:]
        Expr[Expr:Expr:Expr:]
        Expr[Expr:Expr:Expr:Expr:]
        Expr[Expr:Expr:Expr:Expr:Expr]
        Expr[Expr:Expr:Expr:Expr:Expr:Expr]
        Expr[Expr:Expr:Expr:Expr:Expr:Expr:Expr]
        Expr[Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr]
        Expr[Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr]
        Expr[Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr]
        Expr[Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr]
        Expr[Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr]
        Expr[Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr:Expr]
KOOL-TYPED-STATIC-SYNTAX
Expr -> Expr.Id
        Expr.Decl
KOOL-TYPED-STATIC-SYNTAX
```

Statements

```
KOOL-TYPED-STATIC-SYNTAX
Stmt -> Decl
        Expr
KOOL-TYPED-STATIC-SYNTAX
Stmt -> Expr
        Expr,Expr
        Expr,Expr,Expr
        Expr,Expr,Expr,Expr
        Expr,Expr,Expr,Expr,Expr
        Expr,Expr,Expr,Expr,Expr,Expr
        Expr,Expr,Expr,Expr,Expr,Expr,Expr
        Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr
        Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr
        Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr
        Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr
        Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr
        Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr,Expr
KOOL-TYPED-STATIC-SYNTAX
Stmt -> Id = Expr
        Id, Id=Expr
```

Dangling-else rule

```
KOOL-TYPED-STATIC-SYNTAX
Stmt -> if (Expr)
        if (Expr) Stmt else Stmt
        if (Expr) Stmt else Stmt
        if (Expr) Stmt else Stmt
```