CISC 672 – Advanced Compiler Construction

Timo Kötzing

September 9, 2008

Disclaimer

The following does not describe the cool-language in depth. It is not designed to be used as a syntax reference, but rather as an introduction into programming with cool, and also into object oriented programming in general.

For the purpose of writing your own cool-compiler, please read the cool-manual carefully.

What is a COOL-Program?

- a cool-program is a list of cool-classes
- classes may be spread over several files
- one of the classes has to be named "Main"
- this class has to contain a method named "main"
- executing a cool-program is equivalent to evaluating this "Main.main()" function

What is a COOL-Program?

- a cool-program is a list of cool-classes
- classes may be spread over several files
- one of the classes has to be named "Main"
- this class has to contain a method named "main"
- executing a cool-program is equivalent to evaluating this "Main.main()" function

What is a COOL-Program?

- a cool-program is a list of cool-classes
- classes may be spread over several files
- one of the classes has to be named "Main"
- this class has to contain a method named "main"
- executing a cool-program is equivalent to evaluating this "Main.main()" function

What is a COOL-Program?

- a cool-program is a list of cool-classes
- classes may be spread over several files
- one of the classes has to be named "Main"
- this class has to contain a method named "main"
- executing a cool-program is equivalent to evaluating this "Main.main()" function

What is a COOL-Program?

- a cool-program is a list of cool-classes
- classes may be spread over several files
- one of the classes has to be named "Main"
- this class has to contain a method named "main"
- executing a cool-program is equivalent to evaluating this "Main.main()" function

What is a COOL-Program?

- a cool-program is a list of cool-classes
- classes may be spread over several files
- one of the classes has to be named “Main”
- this class has to contain a method named “main”
- executing a cool-program is equivalent to evaluating this “Main.main()” function

What is a COOL-Class?

- a cool-class is a list of features
- features are either attributes or methods
- attributes are local variables (with scope of the class)
- methods are global functions, addressed by "$< functionName > . < methodName > (...)$"
- attributes have to have a type and may be initialized
- methods have a (possibly empty) list of formal parameters, a return type and a body

What is a COOL-Class?

- a cool-class is a list of features
- features are either attributes or methods
- attributes are local variables (with scope of the class)
- methods are global functions, addressed by "$<$ *functionName* $>$. $<$ *methodName* $>$ (...)"
- attributes have to have a type and may be initialized
- methods have a (possibly empty) list of formal parameters, a return type and a body

What is a COOL-Class?

- a cool-class is a list of features
- features are either attributes or methods
- attributes are local variables (with scope of the class)
- methods are global functions, addressed by "$< functionName > . < methodName > (...)$"
- attributes have to have a type and may be initialized
- methods have a (possibly empty) list of formal parameters, a return type and a body

What is a COOL-Class?

- a cool-class is a list of features
- features are either attributes or methods
- attributes are local variables (with scope of the class)
- methods are global functions, addressed by "$< functionName > . < methodName > (...)$"
- attributes have to have a type and may be initialized
- methods have a (possibly empty) list of formal parameters, a return type and a body

What is a COOL-Class?

- a cool-class is a list of features
- features are either attributes or methods
- attributes are local variables (with scope of the class)
- methods are global functions, addressed by "$< functionName > . < methodName > (...)$"
- attributes have to have a type and may be initialized
- methods have a (possibly empty) list of formal parameters, a return type and a body

What is a COOL-Class?

- a cool-class is a list of features
- features are either attributes or methods
- attributes are local variables (with scope of the class)
- methods are global functions, addressed by "$< functionName > . < methodName > (\ldots)$"
- attributes have to have a type and may be initialized
- methods have a (possibly empty) list of formal parameters, a return type and a body

What is a COOL-Class?

- a cool-class is a list of features
- features are either attributes or methods
- attributes are local variables (with scope of the class)
- methods are global functions, addressed by "$< functionName > . < methodName > (...)$"
- attributes have to have a type and may be initialized
- methods have a (possibly empty) list of formal parameters, a return type and a body

What are Types in COOL?

- every class is a type
- the basic types are the *classes* "Object", "IO", "Int", "String" and "Bool"
- all classes but Object have to be inherited from exactly one other class (be a child of this class)
- every class that does not specify a class to inherit from is inherited from Object
- the "is child of"-relation has to be a tree, rooted at Object
- basic classes provide several basic functions
- "Int", "String" and "Bool" may not be inherited from

What are Types in COOL?

- every class is a type
- the basic types are the *classes* “Object”, “IO”, “Int”, “String” and “Bool”
- all classes but Object have to be inherited from exactly one other class (be a child of this class)
- every class that does not specify a class to inherit from is inherited from Object
- the “is child of”-relation has to be a tree, rooted at Object
- basic classes provide several basic functions
- “Int”, “String” and “Bool” may not be inherited from

What are Types in COOL?

- every class is a type
- the basic types are the *classes* “Object”, “IO”, “Int”, “String” and “Bool”
- all classes but Object have to be inherited from exactly one other class (be a child of this class)
- every class that does not specify a class to inherit from is inherited from Object
- the “is child of”-relation has to be a tree, rooted at Object
- basic classes provide several basic functions
- “Int”, “String” and “Bool” may not be inherited from

What are Types in COOL?

- every class is a type
- the basic types are the *classes* “Object”, “IO”, “Int”, “String” and “Bool”
- all classes but Object have to be inherited from exactly one other class (be a child of this class)
- every class that does not specify a class to inherit from is inherited from Object
- the “is child of”-relation has to be a tree, rooted at Object
- basic classes provide several basic functions
- “Int”, “String” and “Bool” may not be inherited from

What are Types in COOL?

- every class is a type
- the basic types are the *classes* “Object”, “IO”, “Int”, “String” and “Bool”
- all classes but Object have to be inherited from exactly one other class (be a child of this class)
- every class that does not specify a class to inherit from is inherited from Object
- the “is child of”-relation has to be a tree, rooted at Object
- basic classes provide several basic functions
- “Int”, “String” and “Bool” may not be inherited from

What are Types in COOL?

- every class is a type
- the basic types are the *classes* “Object”, “IO”, “Int”, “String” and “Bool”
- all classes but Object have to be inherited from exactly one other class (be a child of this class)
- every class that does not specify a class to inherit from is inherited from Object
- the “is child of”-relation has to be a tree, rooted at Object
- basic classes provide several basic functions
- “Int”, “String” and “Bool” may not be inherited from

What are Types in COOL?

- every class is a type
- the basic types are the *classes* “Object”, “IO”, “Int”, “String” and “Bool”
- all classes but Object have to be inherited from exactly one other class (be a child of this class)
- every class that does not specify a class to inherit from is inherited from Object
- the “is child of”-relation has to be a tree, rooted at Object
- basic classes provide several basic functions
- “Int”, “String” and “Bool” may not be inherited from

What are Types in COOL?

- every class is a type
- the basic types are the *classes* “Object”, “IO”, “Int”, “String” and “Bool”
- all classes but Object have to be inherited from exactly one other class (be a child of this class)
- every class that does not specify a class to inherit from is inherited from Object
- the “is child of”-relation has to be a tree, rooted at Object
- basic classes provide several basic functions
- “Int”, “String” and “Bool” may not be inherited from

What are Types in COOL?

- Int-constants: $5, 3, 221, \ldots$
- String-constants: "Hello World!", "\t Hi\n",...
- Bool-constants: true, false

What are Types in COOL?

- Int-constants: $5, 3, 221, \ldots$
- String-constants: "Hello World!", "\t Hi\n",...
- Bool-constants: true, false

What are Types in COOL?

- Int-constants: $5, 3, 221, \dots$
- String-constants: "Hello World!", "\t Hi\n",...
- Bool-constants: true, false

What are Types in COOL?

- Int-constants: $5, 3, 221, \ldots$
- String-constants: "Hello World!", "\t Hi\n",...
- Bool-constants: true, false

Simple Example

```
 class Main {
     main() :  Int { 0 };
};
```

IO-Example

```
class Main {
     myIO : IO <- new IO;
     myInput : Int;

     main() : Int { {
          myIO.out_string(''How many? '');
          myInput <- myIO.in_int();
          while 0 < myInput loop
               myIO.out_string(''Hello world!'')
          pool;
          0;
     }};
};
```

Sally

```
class Silly {
    f() : Int { 5 };
};

class Sally inherits Silly { };

class Main {
    x : Int <- (new Sally).f();

    main() : Int { x };
};
```

Inheritance

- all features are inherited
- no attribute may be redefined
- methods may be overridden, if the number and all types of the formal parameters match, as well as the return type
- static dispatch possible with $< object > @ < type > . < methodName > (\dots)$

Inheritance

- all features are inherited
- no attribute may be redefined
- methods may be overridden, if the number and all types of the formal parameters match, as well as the return type
- static dispatch possible with $< object > @ < type > . < methodName > (\ldots)$

Inheritance

- all features are inherited
- no attribute may be redefined
- methods may be overridden, if the number and all types of the formal parameters match, as well as the return type
- static dispatch possible with $< object > @ < type > . < methodName > (\ldots)$

Inheritance

- all features are inherited
- no attribute may be redefined
- methods may be overridden, if the number and all types of the formal parameters match, as well as the return type
- static dispatch possible with $< object > @ < type > . < methodName > (\ldots)$

Inheritance

- all features are inherited
- no attribute may be redefined
- methods may be overridden, if the number and all types of the formal parameters match, as well as the return type
- static dispatch possible with
  $< \textit{object} >$ @ $< \textit{type} > . < \textit{methodName} > (\ldots)$

Sally, revised

```
class Silly { f() :  Int { 5 }; };

class Sally inherits Silly {
      f() :  Int { 7 }; };

class Main {
      mySally :  Sally <- new Sally;

      main() :  Int {
           mySally.f()
      };

      alternative() :  Int {
           mySally@Silly.f()
      };
};
```

The COOL-Manual

Read the cool-manual on if-then-else, case-statements, let, arithmetic operations and so forth.

The cool-manual will be your main reference when working on any of the phases of your cool-compiler.

The COOL-Manual

Read the cool-manual on if-then-else, case-statements, let, arithmetic operations and so forth.
The cool-manual will be your main reference when working on any of the phases of your cool-compiler.