Bootstrapping a Smalltalk

Preprint of the Smalltalks 2011 International Workshop

G. Casaccio[1,*], S. Ducasse[1,*], L. Fabresse[2,*], J-B Arnaud[1,*], B. van Ryseghem[1,*]

Abstract

Smalltalk is a reflective system. It means that it is defined in itself in a causally connected way. Traditionally, Smalltalk systems evolved by modifying and cloning what is called an image (a chunk of memory containing all the objects at a given point in time). During the evolution of the system, objects representing it are modified. However, such an image modification and cloning poses several problems: (1) There is no operational machine-executable algorithm that allows one to build a system from scratch. A system object may be modified but it may be difficult to reproduce its exact state before the changes. Therefore it is difficult to get a reproducible process. (2) As a consequence, certain classes may not have been initialized since years. (3) Finally, since the system acts as a living system, it is not simple to evolve the kernel for introducing new abstractions without performing some kind of brain surgery on oneself. There is a need to have a step by step process to build Smalltalk kernels from scratch. In this paper, after an analysis of past and current practices to mutate or generate kernels, we describe a kernel bootstrap process step-by-step. First the illusion of the existence of a kernel is created via stubs objects. Second the classes and meta-classes hierarchy are generated. Code is compiled and finally information needed by the virtual machine and execution are generated and installed.

1. Introduction

Smalltalk is a reflective system. It means that it is defined in itself in a causally connected way. Objects and their meta-representation are synchronized, hence editing a class is automatically reflected in the object structure that represents it. The definition of the complete environment is expressed as Smalltalk expressions. This leads to the expected chicken and egg problem: how can we define the system since it needs the system to be defined. Such question is answered as we will show later, by pretending that a version of the system already exists in some form and using such version to express the full blown version of it or its next iteration.

Traditionally Smalltalk systems were not bootstrapped declaratively (by declaratively we mean following an operational machine-executable algorithm) but evolved by cloning what is called an image (a chunk of memory containing all the objects and in particular the objects representing the kernel at a given point in time). A Smalltalk image is a powerful concept, it stores all object states. When the image is restarted, its state is the same as it was at the last snapshot. It is possible to perform some changes and snapshot the image with another name. Some tools such as the SystemTracer in Squeak [BDN+07] can produce a new image by applying certain transformations (like pointer representation modification) to the objects.

However, such an image cloning poses several problems:

1. While we can produce a new image from an existing one, we have to apply all the sequences of modifications one after the other one. In addition, it may be difficult to get the system to a specific state (*e.g.,* processes) before applying certain update. There is no operational machine-executable algorithm step that allows one to build a system *from scratch.*

2. Certain classes have not been initialized since years. Code may rot because not systematically exercised. For example, in old versions of Squeak some initializing methods where referring to fonts

*Corresponding author

Email addresses: gwenael.casaccio@inria.fr (G. Casaccio), stephane.ducasse@inria.fr (S. Ducasse), luc.fabresse@mines-douai.fr (L. Fabresse), jbarnaud@inria.fr (J-B Arnaud), benjamin.vanryseghem@gmail.com (B. van Ryseghem)

[1]RMoD Project-Team, Inria Lille–Nord Europe / Université de Lille 1.

[2]Université Lille Nord de France, Ecole des Mines de Douai.

stored on hard drive. Such a situation clearly showed that the system was not initialized from its own description and that these initialization methods were absolutely not executed since a couple of years.

3. Since the system acts as a living system, it is not simple to evolve the kernel for introducing new abstractions. We have to pay attention and migrate existing objects (changing class representation for example). Some radical changes (*e.g.,* changing the header size of objects) cannot be achieved by simple object changes (because objects with modified object format cannot co-exist in the same image) but require to use disc storage to create a new modified version of the system.

4. Since the system is not rebuilt using a process that does not have to execute all the modification stream, it is hard to produce a system with only wanted parts. Current implementations often rely on image shrinkers which remove unnecessary parts of the systems. However, this process is tedious because of the dynamic nature of Smalltalk and the use of reflective features which breaks static analysis [LWL05].

The contributions of this paper are:

1. A comparison of existing bootstrapping approaches for Smalltalk. Through this comparison we also did our best to document related work (mainly software) because most of them have never been published in any ways and certainly not in scientific venues, did not run anymore and do not provide documentation or an obsolete one;

2. The description of CorGen: a process and the steps required to bootstrap a Smalltalk kernel. Our solution uses the GNU Smalltalk infrastructure but the approach can be adapted to use another execution engine (such as a binary loader, or another Smalltalk implementation). The solution presented is fully working and the code snippets are extracted from the actual implementation.

The rest of the paper is structured as follows. In Section 2, we present the key aspects of reflective systems by presenting some definitions. We explain the importance of bootstrapping a Smalltalk Kernel. Section 3 describes other solutions. Section 4 presents CorGen our approach. Section 5 discusses some issues. The subsequent Section presents related work and conclude.

2. Reflective System and Bootstrap

Before going any further, we present some definitions that characterize reflective systems.

2.1. Definitions

P. Maes has proposed in the first chapter of her thesis [Mae87], precise definitions to clearly characterize reflective programming. We refer here to these definitions:

- A *computational system* is something that *reasons* about and *acts* upon some part of the world, called the *domain* of the system.
- A computational system may also be *causally connected* to its domain. This means that the system and its domain are linked in such a way that if one of the two changes, this leads to an effect upon the other.
- A *meta-system* is a computational system that has as its domain another computational system, called its *object-system*. [...] A meta-system has a representation of its object-system in its data. Its program specifies *meta-computation* about the object-system and is therefore called a *meta-program*.
- *Reflection* is the process of reasoning about and/or acting upon oneself (see Figure 1).

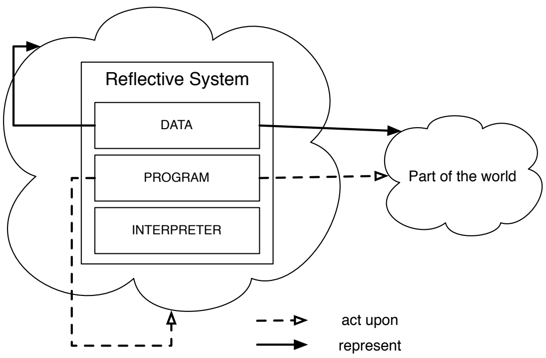

Figure 1: A reflexive system.

- A *reflective system* is a causally connected meta-system that has as object-system itself. The data of a reflective system contain, besides the representation of some part of the external world, also a causally connected representation of itself, called *self-representation* of the system. [...] When a system is reasoning or acting upon itself, we speak of *reflective computation*.

Bootstrapping a kernel is the process that builds the minimal structure of a language that is reusable to define this language itself. The idea is to use as early as possible the benefits of the resulting language by implementing a minimal core whose only goal is to be able to build the full system. As an example of a possible bootstrap: we write in C the minimal structures to represent and execute objects, and we then write with this core the full system. This avoids to have to write the full system (full compiler in C for example). In ObjVLisp [Coi87], the class Class is first defined using low level API, then Object is created, then Class is fully reimplemented using the first one.

2.2. Why bootstrapping is important?

Bootstrapping a system may be perceived as an academic exercise but it has strong software engineering positive properties:

Agile and explicit process. Having a bootstrap is important to be sure that the system can always be built from the ground. It makes sure that initialization of key parts of the system (character, classes, ...) is exercised each time the system is built. It limits broken code; this is especially important in Smalltalk since classes are initialized at load time, but can evolve afterwards. It also makes sure that there is no hidden dependency. This supports the idea of software construction by selecting elements.

Warranty of initial state. Currently, the evolution of a Smalltalk image is done by mutating objects in sequence: a stream of changes can bring an image from a state A to a state B. It is not always possible that a change bringing the system to a state C can be applied from state A. It may happen that B has to be reached, and then only C can be applied. Some changes may not be interchangeable and it may be difficult to identify the exact state of the system (for example in terms of running processes). Using a bootstrap process to initialize the kernel, we get the warranty to have a consistent initial state.

Explicit malleability and evolution support. Having an explicit and operational machine executable process to build a kernel is also important to support evolution of the kernel. The evolution can be performed by defining new entities and their relation with existing ones. There is no need to build transition paths from an existing system to the next version. This is particularly important for radical changes where migration would be too complex.

Minimal self reference. The self referential aspect of a bootstrap supports the identification of the minimal subset necessary to express itself. It forces hidden information to be made explicit (format of objects...). From that perspective, it supports better portability of the code basis to other systems.

2.3. Minimal Infrastructural Requirements

Figure 2 depicts the main parts and steps to bootstrap a Smalltalk system (and probably other languages) *i.e.,* generate a new runtime kernel.

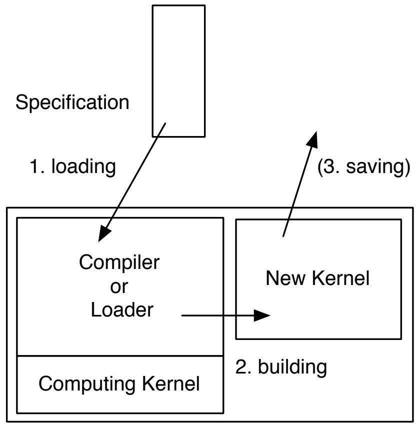

Figure 2: Bootstrap elementary parts.

The most important elements in a bootstrap process are:

Specification. A textual or binary description of a kernel. It can be an execution description (Smalltalk expressions) or results of the execution (objects and compiled methods in our case).

Loader. The loader has an important role because it executes the specification and should lead to the creation of a new kernel (been it a binary or textual one). A loader is either a binary *object loader* or *compiler* that transforms the specification from a given format to another one that can be executed.

Computing kernel. The computing kernel is the setup required to execute the loader and create the infrastructure of the newly created kernel. The computing kernel does not have to be the same as the newly created kernel. For example a computing kernel can be a C application executing objects. For example in Resilient [ABG+04], the compiler and the kernel were defined in Java and the kernel was executing Smalltalk bytecode and objects. When the loader is expressed within the same system than the bootstrapped system, the computing kernel can be the same as the resulting kernel. The loader can be expressed either in C or Smalltalk such as Fuel [DPDA11].

One of the key points is whether the loader is expressed in the implementation language (C for example) or within the language that is bootstrapped. In the former case, the infrastructure work has to be done with the implementation language. This can be tedious. For example in GNU Smalltalk [Gok10] the compiler is written in C, therefore changing the syntax of the language takes much more time than just simply modifying a compiler written in Smalltalk.

2.4. *Key Challenges*

Several challenges occur when bootstrapping a new kernel.

Multiple meta dependencies. Similarly to the chicken-egg problem between a class, its metaclass and Object class, there are more complex circular dependencies in a kernel: for example, Array, String and all the literal objects are used to represent the internal structure of classes and they should be described as classes.

Controlling references. One of the problems is how to deal with existing kernel code and the dependencies between existing packages. Using the current Smalltalk kernel as the skeleton for a new one is an unstable solution. Indeed, during the class and metaclass creation step, we follow the kernel object graph. During this pass we could escape (*i.e.,* refer to objects or classes not belonging to the new kernel) the new kernel boundaries. This way we may end up having kind of reference leaks and referencing to the full system when accessing existing class variables, processes or pool variables [vR11]. It is possible to flag and filter escaping objects or objects that should not be part of new kernel; however, it's hard to decide if a shared pool variable should be excluded or not, since excluding it may produce an unworkable system.

And for such reasons building a kernel from scratch offers a good property because of the explicit control of what belongs to the new kernel. This control of all the information added to the kernel comes at the price of their specification.

Supporting deep changes. Bootstrapping a new kernel should support deep model changes such as: change of CompiledMethod class, new bytecode definition, new object format, new object model (introduction of traits for example), new scheduling or process implementation or semantics. It should be possible to create restricted kernels with no reference to any other objects.

3. Existing Approaches

Bootstrapping a system is the process and steps to produce a (minimal) system able to fully work. As such, generating a Smalltalk image can be seen as the result of the bootstrap process for Smalltalk. This is why traditionally people proposed processes to be able to generate new kernels based on an existing one. We present such solutions now. These solutions can be classified in two categories: *execution-based* (*i.e.,* the system is executed and a trace is used to identify objects that will be part of a new image) or *static-based* (*i.e.,* programs specify all the steps to create a new kernel) approaches.

The simplest way to produce a new image is to save the image with another name. However, the state of the objects is not always in a state that is satisfactory as explained previously. In addition, for certain evolution such as changing object pointers or object headers encoding requires to adapt objects and such different kind of objects cannot coexist in the same image by construction because they require deep changes in the virtual machine.

To support such evolutions, SystemTracer (available in Squeak/Pharo [BDN+09]) is a tool that iterates over all the objects contained in an image. For each visited objects a function is applied and the result is written into a new image file. While SystemTracer is useful to support virtual machine changes that should be reflected at the image level, it addresses a specific scenario and not a bootstrap in itself. SystemTracer can be used to save the resulting kernel.

The approaches that generate new images can be roughly categorized as illustrated in Figure 3:

Execution-based approaches. The first category (Spoon [Lat], Chácharas [Rei]) relies on program execution. The first one starts from a minimal object kernel and copy to this system, methods and classes that are leading to an error at runtime. The second one does the inverse: it copies the objects reached during execution.

Static-based approaches. The second category is based on a static description of a kernel. The difference between static approaches is about their level of explicitness. Some approaches (like MicroSqueak [Mal]) create in a separate namespace a new kernel and use image serialization (SystemTracer) to generate the resulting core. Other approaches, such as the one presented in this paper, follow a more thorough approach where all the steps are explicitly described. Indeed, the serialization is a shortcut that avoids the description of the object format and other implicit information.

Again, since none of these approaches is documented or sometimes even described, we are doing our best to describe them but we may be wrong. For example, the description of Chácharas on the Web describes ideas that we could not identify in the implementation and its documentation mentioned that it may be obsolete.

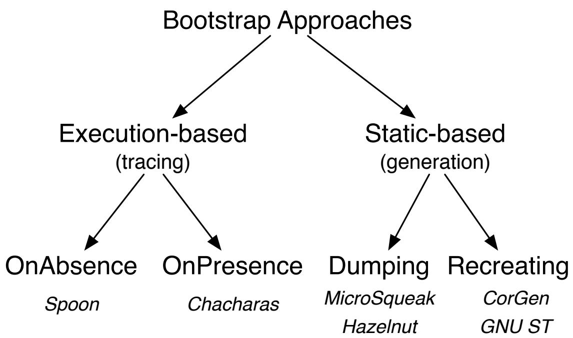

Figure 3: Taxonomy of image generation and bootstrap approaches.

3.1. *Execution-based approaches*

The idea behind the execution-based approaches is to generate specialized images. For example, Chácharas was used to create specific images for a 3D clothing engine.

As shown in Figure 4, we can use a client/server metaphor. Moreover, different versions of the processes exist. However, we use both systems as an illustration of possible solutions based on execution.

Regarding the client/server metaphor, Chácharas creates a new kernel by copying on the “client-side” the objects reached during an execution on the “server-side”. Inversely, Spoon creates a new kernel by importing from the “server-side” the objects that are missing during an execution on the “client-side”.

The approach of Spoon is based on a minimal image and a full image running side-by-side. When an object is needed but not present in the client, the server is asked for it. Thus execution-based approaches are done with a client server communication style between two virtual machines (or potentially two namespaces - however, the fact that the Virtual Machine requires a special object array representing its knowledge about the objects that it can manipulate can be a problem since we cannot have two special arrays in the same image).

The client virtual machine in both Chácharas and Spoon are updated for handling transport of objects between images. The server virtual machine has a full running image and it is used for distributing remote objects. The client virtual machine starts with a minimal kernel, when an object is needed the client virtual machine asks the server to send it. After a certain amount of time errors become less frequent and the image is populated with objects. This process terminates when either the server or the client virtual machine decides to stop the communication.

The question of the creation of the client image is unclear to us: in Spoon, it seems that the minimal image was reached by try and error by its author. Such an image should be minimal but must contain enough functionality to be able to request, and install new objects. We believe that a declarative bootstrap could be used to generate a small client image that integrates such functionalities.

The problem of the fixed point. This process raises the question of the state of the resulting system in case of incomplete scenario. As any dynamic analysis (*i.e.,* based on program execution), the coverage of the execution has an impact on the result [RD99]. The advantage of such approaches is their dynamicity and the way to cope with new entities. There is no predefined description.

3.2. *Static-based Approaches*

Static-based approaches use a kernel source definition and generate a new kernel from the kernel sources. The creation of the kernel is often divided in four steps:

1. Stub objects generation: Generation of objects needed for the class generation, like symbol table, characters, true, false, nil, or Smalltalk namespace. Here stubs are used to make the system believe that they exist but they are only used for reference and their definition is filled up later.

2. Classes and metaclasses generation: Create all the classes and metaclasses and fill their fields (name, superclass, instance variables, format);

3. Compilation: All the methods are compiled and added to method dictionaries. Literals within method literal frames should refer to stub objects;

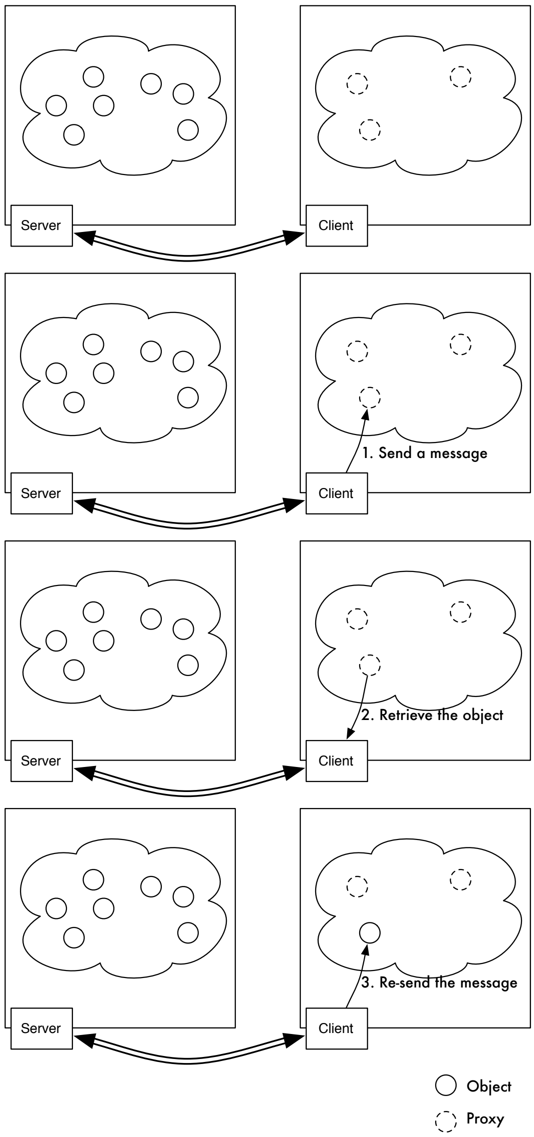

Figure 4: Lazily populating a small core.

4. Stub objects are replaced by real ones. A first process is created and ready to execute and the special object array is filled.

3.2.1. MicroSqueak

In MicroSqueak, a kernel is loaded from files into a namespace - class names are decorated with a prefix - and the generator ensures that the references are self-contained to the namespace [Mal]. The MicroSqueak kernel has a regular set of classes, with their compiled methods, shared pools and class variables.

Kernel classes are visited and added to a dictionary, which is then used by the generator: when the kernel object graph is visited, if a reference to an object class doesn't belong to the generated dictionary, the generation is stopped. During the process some globals are excluded by the visitor like Process, LinkedList or ClassOrganizer. The generator follows the graph of the MicroSqueak kernel objects and fixes the object references. If the referenced class doesn't belong to the MicroSqueak namespace, it's excluded from the generated kernel.

All the external references like nil and the metaclasses are fixed to point to their corresponding entity in the new kernel. Next, an initial process is installed that initializes the image. Finally, the special object array, an array storing objects known by the virtual machine, is installed.

The last step is the serialization of the image, all the objects in the MicroSqueak kernel are visited by the serializer. If the object class doesn't belong to one of the kernel classes the process is stopped. That condition prevents the serializer to escape the kernel and save the full image.

This approach relies on a SystemTracer transformation to change the object format of the compiled method or the class. The introduction of new format cannot be done at the specification level since the computing kernel cannot handle different format.

3.2.2. An Hybrid Approach: Hazelnut

Hazelnut [vR11] is an hybrid bootstrapping approach built in Pharo. At the time of this writing, Hazelnut is not fully bootstrapping a Pharo kernel. It is similar to MicroSqueak but it does not rely on a specific list of classes that are manually edited. Hazelnut takes a list of classes as inputs and recursively copy classes (paying attention to cut certain dependencies when the starting system is not correctly layered) into a new namespace and uses SystemTracer to save a new image. Some of the steps of Hazelnuts are similar to the one described later in the paper. However, the main difference is that Hazelnut does not provide a declarative bootstrap, but extract a kernel from an existing one by recursively visiting a selected set of classes. Hazelnut does not support the explicit specification of handling a different object format. It is only possible using a dedicated SystemTracer.

3.2.3. The C GNU Smalltalk bootstrap

GNU Smalltalk is the only Smalltalk that is able to recreate a new image from scratch. The GNU Smalltalk virtual machine, written in C, performs this task. The bootstrap function in the virtual machine creates some objects like true, false, nil, the characters, a symbol table, the Smalltalk namespace and a processor scheduler. Next, the class and metaclass hierarchy is created. For each class, there is a C struct that stores all its information like its shape (determining that it is an immediate value, a class or a regular object), its name and its instance variables. Finally, an entry is added in the global symbol table for each class. Next, the kernel source files are loaded file by file and are executed as a regular Smalltalk execution, if the class is not present it is created. All the methods in the files are compiled and added to classes. Once all the files are created the classes are initialized.

The main advantage of the GNU Smalltalk approach is to produce a clean image. It recreates all the classes and recompile all the methods. Unfortunately all the process is defined in C as part of the virtual machine level. Therefore it is tedious to change and we cannot take advantage of using Smalltalk to specify it.

3.3. Comparing the Approaches

Both approaches have their advantages and disadvantages; the tracing methods perform well for image migration for instance when the object header is updated or if the image needs to be migrated to 64 bits virtual machine. But they fail when the object graph needs to be controlled and restricted.

Dynamic approaches like Spoon or Chácharas allows one to create kernels with an evaluated portion of code. This is the most dynamic approach, only the needed objects are copied. But it opens multiple questions: when a system is considered as stable, what happens if the server objects are changed too. The server should be in a stable state during the client population. Moreover if the application is an interactive program - a website, a program with an user interface - all the user interactions should be executed.

A step by step approach constructs a new system from scratch. There changing the kernel has no impact on our living system and allows one to experiment with the image. It is easier to distribute a program with a clean environment, without the development tools and unwanted packages. Changing the compiled method or the class format is easy to do with a declarative model.

4. CorGen Overview

In this section, we describe CorGen, a bootstrap developed in GNU Smalltalk to bootstrap new Smalltalk kernels and images from scratch. CorGen uses a step by step machine executable approach following the steps mentioned before more precisely the bootstrap process creation is done in five steps see Figure 6:

1. Creation of the stub objects for literal objects: nil, true, false, characters ;
2. Definition of classes and metaclasses ;
3. Method compilation;
4. Creation of process and special object array;
5. Image serialization.

We will go over these steps and illustrate them using code snippets taken from CorGen. The full code of CorGen is given in Appendix A. Figure 5 shows the declaration of the Bootstrap class and its instance variables that hold essential information such as the literals objects: nil, true, false, characters, symbols. Note that we concatenate 'Gst' to name variables[3] to make sure that we can compile the code (since we cannot have variable named nil, true, false...). In addition, instead of manually listing all the kernel classes, expressing their inheritance relationship and instance variables, we use files saved in a specific directory. Each file only contains the class definition with its instance variables. This way we can modify the list of classes without having to change the bootstrap as illustrated in the bootstrapKernel method.

```
Object subclass: Bootstrap [
  | nilGst trueGst falseGst smalltalkGst
    charactersGst symbolTableGst files ... |

  Bootstrap class>>bootstrapKernel [
    <category: 'bootstrapping'>
    self new
      files: self kernelSourceFiles;
      bootstrap ]
  "..." ]
```

Figure 5: Defining some variables, getting the list of all the core classes and launching the bootstrap.

Figure 6 describes the main steps of the bootstrap. We first create and initialize stub objects (*nil*, ...). We then import sources of kernel classes and create stub classes for them. We process each stub class to fix its internals and add methods. Then, literal objects are created. Finally, some special objects are created (*Processor*, ...) and the image is saved.

[3] Note that naming conventions are slightly different in the source code given in Appendix A. We changed it here to be more understable.

```
Bootstrap>>bootstrap [
  <category: 'bootstrapping'>
  self
    instantiateSmalltalkObjects;
    importClassesFromSources;
    processClasses;
    setupSmalltalkObjects;
    saveImage]
```

Figure 6: Bootstrap process code.

4.1. Creation of the stub for literal objects: nil, true, false, characters

A set of initial objects are created, see Figure 7. nil, true, false, and characters are created but since their respective classes are not created at this stage, their are instances of the class GstObject. The class field of these objects will be later correctly filled when their respective class has been created and compiled. The symbol table is created and then used when symbols are created. Also the characters and the Smalltalk namespace are created too and like the other objects their class isn't set.

```
Bootstrap>>instantiateSmalltalkObjects [
  <category: 'instantiate'>
  self
    instantiateNilGst;
    instantiateTrueGst;
    instantiateFalseGst;
    instantiateCharactersTable; "build all the characters"
    instantiateEnvironment "create System Dictionary"]
```

Figure 7: stubs creation source code

The created objects are instances of GstObject, this is the root class of all the objects knows by the bootstrapper. With GstObject»gstClass:, GstObject»gstClass messages the developer has access to the class of the object. The instance variables are accessible with the GstObject»slotAt:put:, GstObject»slotAt:.

4.2. Classes and metaclasses creation

Since the different stubs objects are created, classes and metaclasses can be created (see Figure 8). All the information for their creation, can be stored in files, or in a model. We create the different classes and metaclasses with the meta-information imported from files. But the method dictionary is for now not yet initialized. We set the name, and since it's a symbol it's added to the symbol table. We set the superclass, the set of subclasses but since the Set class isn't yet defined we have to set it after. And the same is done for the instance variables, category, environment, shared pools and class variables. The metaclass is linked to its class. Now we've setted up the class and metaclass hierarchy, the previously unset classes of nil, false, true, characters, string, and symbol are set.

```
Bootstrap>>processClasses [
  "fill the class stubs with real classes"
  self
    "create classes and add them to System Dictionary"
    createClasses;
    "compile and install CompileMethods"
    compileMethods ]
```

Figure 8: Proccess classes source code.

4.3. Method Compilation

Now classes and metaclasses are correctly filled, they can be used by the compiler to generate the methods. The methods source are taken from the model or the kernel source file. The compiler used here may be a dedicated one, so that the bytecode set or other optimization may be changed. The compiler is parametrized by an environment and a symbol table; the symbol table is used to store new symbols and the globals lookup is achieved via this environment. When the method is compiled it is installed in the method dictionary of the class.

4.4. Creation of process and special object array

The kernel classes are created and initialized, but this is not enough to have a runnable image: some objects are missing such as the Processor. An idle process is created, it will be activated when no other processes are running in the image (see Figure 9). Another process is created, it initializes the system by calling all the classes initialize methods. Next, the ProcessorScheduler is created and initialized with the different processes. Finally the special object array is created, this array contains all the objects known by the virtual machine. It stores the Message class, doesNotUnderstand: symbol, the ProcessScheduler, true, false, nil. This object is specific to the virtual machine. The bootstrapper has to populate it with the needed objects. The system is complete and ready to be saved in an image file.

The method createInitContext creates a method context object that points to the method that will get executed when the image will start.

```
Bootstrap>>setupSmalltalkObjects [
  <category: 'bootstrap'>
  self
    setupCharacter; "insert references to the Character table"
    setupSymbol; "insert references to the Symbol table"
    setupProcessor "create Processor and install it" ]
Bootstrap>>setupProcessor [
  | processorGst |
  processGst := self createProcess.
  processorGst := GstProcessorScheduler new.
  processorGst
    scheduler: nilGst;
    processes: self buildProcessList;
    activeProcess: processGst;
    idleTasks: nilGst. ]
Bootstrap>>createProcess [
  | processGst |
  (processGst := GstProcess new)
    nextLink: nilGst;
    suspendedContext: self createInitContext;
    priority: 4;
    myList: nilGst;
    name: GstString new;
    interrupts: nilGst;
    interruptLock: nilGst ]
```

Figure 9: Final set up of specific objects.

4.5. Image serialization

The serialization is a classical object graph transversal. We follow the object graph; writing them one by one in a stream and adding them in an identity dictionary to avoid serializing twice the same object. There is nothing special here, the responsibility of the serialization is let to the object; and the shape writer if the object has a special shape like the compiled method or byte array. The image header is written, the special object array and all the objects are saved on disk.

4.6. Resulting Kernel

The kernel used for image generation is a little kernel with few classes: only about 54 classes; those classes enable to generate a Smalltalk system with reflection. This kernel is not certainly not minimal. We think that it is possible to generate a smaller kernel; for instance the Array class can replace the method dictionary class.

Kernel (15 classes): Behavior, BlockClosure, BlockContext, Boolean, Class, ClassDescription, ContextPart, False, Metaclass, MethodContext, MethodInfo, Object, ProcessorScheduler, True, UndefinedObject.

Collection (27 classes): Array, ArrayedCollection, Bag, BindingDictionary, ByteArray, CharacterArray, Collection, CompiledBlock, CompiledCode, CompiledMethod, Dictionary, HashedCollection, IdentityDictionary, Iterable, Link, LinkedList, LookupTable, MethodDictionary, OrderedCollection, Process, Semaphore, SequenceableCollection, Set, String, Symbol, SystemDictionary, WeakSet.

Magnitude (12 classes): Association, Character, Float, Fraction, Integer, LookupKey, Magnitude, MethodInfo, Number, SmallInteger, VariableBinding, HomedAssociation.

5. Discussion

Our approach is similar in the way Common Lisp is bootstrapped [Rho08]. Lisp like Smalltalk has the concept of image, and for generating new images they migrate their current image. In that paper they describe their approach for generating a new virtual machine and new image. First a cross compiler is compiled inside the host. A special namespace beginning with SB is used by the cross-compiler. The cross compiler is then used for generating lisp object files. Those files are loaded inside a pseudo image, which is simply a byte array. Once the image is built, the virtual machine uses it for the initialization of the image.

5.1. Ruby and Python

For Ruby, the Ruby kernel is loaded, bootstrapped and initialized. The process is different than our bootstrap process: the initialization mixed Ruby initialization and virtual machine initialization in C. The process is divided in multiple module initialization; all the modules are initialized from the virtual machine side. Once all the modules are initialized the virtual machine can evaluate some Ruby code.

At the beginning, some virtual machine modules are initialized and some stubs objects are created like the symbol table and it interns some symbols. The classes BasicObject, Object, Module, Class, True, False, Nil are created and the hierarchy is correctly set. The Kernel module is created. Then, few primitives methods are inserted in the method dictionaries of these initial classes. Other modules like threads are initialized and the virtual machine is operational. And the Ruby virtual machine is ready to execute code.

Python kernel bootstrapping is really close to the Ruby's one; in C, the Python virtual machine is initialized. Some classes stubs are created and initialized in the virtual machine, some strings are interned. After the modules creation and initialization, the interpreter is ready to execute some code.

5.2. *Static vs. Execution-based approaches*

It's easier to control the result of a static generation in a reflective and dynamic language such Smalltalk; with a kernel we can reproduce all the steps to generate an image from the stub object generation to the method compilation and image writing. It's easy to change the object format or the byte code with a new compiler.

Execution-based approaches are dynamic. On the one hand, they are more risky because it is difficult to carefully control the set objects that will be selected for the bootstrapped image by following an object graph. For example, bootstrapping by tracing the objects used by a Browser will probably end up at cloning the image because it uses reflection and would imply marking all objects as used. This approach is also not suitable for interactive programs. On the other hand, this dynamic approach is interesting to see the minimal runtime required by a program and unit testing can help to see if it behaves well. But the tracing stage is crucial to deliver a reliable image, it should be done by taking a maximum of the execution paths. Ideally, all possible execution paths should be traced.

5.3. *Process and parallel evolution*

Our experience working on Hazelnut while the core of Pharo itself was heavily evolving shows us that a declarative bootstrap can be tedious because we should pay attention of the parallel evolution of the declarative bootstrap class definition and the classes currently modified in the system. Bootstrapping an existing system where dependencies are not layered is tedious. Hazelnut took the process to not be based on a declarative specification but to use the current image as input and to be traced [vR11]. Our conclusion is that a declarative bootstrap as the one of CorGen is clearly a good solution for the static core of the system but depending on the life cycle of a project it worth starting with an execution-based (tracing) approach as an intermediate solution.

6. Conclusion

Bootstrapping a reflective language is the last step towards full auto description. For long time, Smalltalks evolved by cloning their image instead of using a step by step executable process starting from scratch. In this paper we presented the bootstrap process we implemented in GNU Smalltalk. It opens a wide range of applications such as supporting multiple minimal kernels and new generation of kernel as well as the co-evolution of kernel and Virtual Machines.

Acknowledgements

This work was supported by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the Contrat de Projets Etat Region (CPER) 2007-2013.

[ABG+04] Jakob R. Andersen, Lars Bak, Steffen Grarup, Kasper V. Lund, Toke Eskildsen, Klaus Marius Hansen, and Mads Torgersen. Design, implementation, and evaluation of the resilient smalltalk embedded platform. In *Proceedings of ESUG International Smalltalk Conference 2004*, September 2004.

[BDN+07] Andrew Black, Stéphane Ducasse, Oscar Nierstrasz, Damien Pollet, Damien Cassou, and Marcus Denker. *Squeak by Example*. Square Bracket Associates, 2007.

[BDN+09] Andrew P. Black, Stéphane Ducasse, Oscar Nierstrasz, Damien Pollet, Damien Cassou, and Marcus Denker. *Pharo by Example*. Square Bracket Associates, 2009.

[Coi87] Pierre Cointe. Metaclasses are first class: the ObjVlisp model. In *Proceedings OOPSLA '87, ACM SIGPLAN Notices*, volume 22, pages 156–167, December 1987.

[DPDA11] Martin Dias, Mariano Martinez Peck, Stéphane Ducasse, and Gabriela Arévalo. Clustered serialization with fuel. In *Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST 2011)*, Edinburgh, Scotland, 2011.

[Gok10] Cano Gokel. *Computer Programming using GNU Smalltalk*. http://www.canol.info/books/computer_programming_using_gnu_smalltalk/, 2010.

[Lat] Craig Latta. Spoon. http://netjam.org/projects/spoon/.

[LWL05] Benjamin Livshits, John Whaley, and Monica S. Lam. Reflection analysis for java. In *Proceedings of Asian Symposium on Programming Languages and Systems*, 2005.

[Mae87] Pattie Maes. *Computational Reflection*. PhD thesis, Laboratory for Artificial Intelligence, Vrije Universiteit Brussel, Brussels Belgium, January 1987.

[Mal] John Maloney. Microsqueak. http://web.media.mit.edu/~jmaloney/microsqueak/.

[RD99] Tamar Richner and Stéphane Ducasse. Recovering high-level views of object-oriented applications from static and dynamic information. In Hongji Yang and Lee White, editors, *Proceedings of 15th IEEE International Conference on Software Maintenance (ICSM'99)*, pages 13–22, Los Alamitos CA, September 1999. IEEE Computer Society Press.

[Rei] Alejandro Reimondo. Image gestation project. http://alereimondo.no-ip.org/ImageGestation.

[Rho08] Christophe Rhodes. Sbcl: A sanely-bootstrappable common lisp. In *International Workshop on Self Sustainable Systems (S3)*, pages 74–86, 2008.

[vR11] Benjamin van Ryseghem. Hazelnut: dynamically creating a kernel in a reflective language, 2011. http://rmod.lille.inria.fr/web/pier/software/Seed.

Appendix A. GNU ST source code of CoreGen

```
"*************************************************************
 bootstrap/Bootstrap.st
 *************************************************************"
" Copyright 2010 GST.
  Written by Gwenael Casaccio

  This file is part of GST.

  GST is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GST is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GST.  If not, see <http://www.gnu.org/licenses/>. "

Object subclass: Bootstrap [

    Bootstrap class >> bootstrapKernel [
        <category: 'bootstrapping'>

        self new
            files: self kernelSourceFiles;
            bootstrap;
                yourself
    ]

        Bootstrap class >> kernelSourceFiles [
        <category: 'bootstrapping'>
                | array |
                array := OrderedCollection new.
                '../kernel' asFile allFilesMatching: '*.st' do: [ :each | array
add: each ].
                ^array
        ]

    | behaviorSize classDescSize classSize nilOOP trueOOP falseOOP fakeSmalltalk
 smalltalkOOP charsOOP symbolTableOOP classOOP files |

        bootstrap [
        <category: 'bootstrapping class'>

        self
            instantiateSmalltalkObjects; "creates stubs for nil, true, false, ..
."
            importClassesFromSources;
                processClasses;
            setupSmalltalkObjects; "fix Nil, creates Character table and Symbols
 Table"
            saveImage
    ]

    classOOP [
        <category: 'accessing'>

        ^ classOOP ifNil: [ classOOP := Dictionary new ]
    ]

    charsOOP [
        <category: 'accessing'>

        ^ charsOOP
    ]

    symbolTableOOP [
        <category: 'accessing'>

        ^ symbolTableOOP ifNil: [ symbolTableOOP := Dictionary new ]
    ]

    classSize: anOOP [

        | size oop |

        oop := anOOP.
        size := 0.
        [ oop parsedClass superclass class name = #ProxyNilClass ] whileFalse: [
            size := size + oop parsedClass instVarNames size.
            oop := self classOOP at: oop parsedClass superclass name ].

        size := size + oop parsedClass instVarNames size.

        ^ size
    ]

    files: anArray [
        <category: 'accessing'>

        files := anArray
    ]

    importClassesFromSources [
        <category: 'model'>

        files do: [ :file |
            (STInST.GSTFileInParser parseSmalltalkStream: file readStream with:
(STInST.STClassLoader new)) do: [ :class |
                    class isClass ifTrue: [ self fillClassOOP: class ] ] ].
    ]

    fillClassOOP: aClass [
        <category: 'bootstrapping class'>

        | oop |
        oop := ClassOOP new
                    parsedClass: aClass;
                    yourself.
        self classOOP at: aClass name put: oop
    ]

    instantiateSmalltalkObjects [
        <category: 'instantiate'>

        self
                    instantiateNilOOP;
            instantiateTrueOOP;
            instantiateFalseOOP;
            instantiateCharactersTable;
            instantiateEnvironment
    ]

    setupSmalltalkObjects [
        <category: 'bootstrap'>

        self
            setupCharacter;
            setupSymbol;
                setupNil;
            setupProcessor
    ]

    processClasses [
        <category: 'bootstrap'>

        self
            createClassOOPs;
            populateEnvironmentOOP;
            compileMethods
    ]

    instantiateOOP [
        <category: 'instantiate'>

        ^ OOP new
```

```
    ]

    instantiateNilOOP [
        <category: 'instantiate'>

        nilOOP := UndefinedObjectOOP new.
        OOP nilOOP: nilOOP.
    ]

    instantiateTrueOOP [
        <category: 'instantiate'>

        trueOOP := self instantiate: #True
    ]

    instantiateFalseOOP [
        <category: 'instantiate'>

        falseOOP := self instantiate: #False
    ]

    instantiateCharactersTable [
        <category: 'instantiate'>

        charsOOP := self buildCharsOOP
    ]

    instantiateEnvironment [
        <category: 'instantiate'>

        smalltalkOOP := self initialize: SystemDictionaryOOP new class: SystemDi
ctionaryOOP name
    ]

    instantiateCharacter [
        <category: 'instantiate'>

        ^ self initialize: CharacterOOP new class: CharacterOOP name
    ]

    buildCharsOOP [
        <category: 'characters'>

        | chars |
        chars := self instantiateArray.

        1 to: 256 do: [ :i |
            chars oopAdd: (self instantiateCharacter
                                            value: i;
                                            yourself) ].

        ^ chars
    ]

    populateEnvironmentOOP [
        <category: 'populate environment'>

        | arrayOOP |

        fakeSmalltalk := Dictionary new.

        STInST.STSymbolTable environmentOOP: fakeSmalltalk.
        STInST.STSymbolTable bootstrap: self.

        smalltalkOOP
            array: (arrayOOP := self instantiateArray);
            size: self classOOP size.

        self classOOP do: [ :oop | | variable |
            variable := arrayOOP oopAdd: (self initialize: VariableBindingOOP ne
w class: VariableBindingOOP name).

            fakeSmalltalk at: oop parsedClass name put: variable.

            variable
```

CorGenCode

```
                key: oop name;
                value: oop;
                environment: smalltalkOOP ]
    ]

    compileMethods [
        STInST.STCompiler
                    symbolOOP: self symbolTableOOP;
                    bootstrap: self.

        self classOOP keysAndValuesDo: [ :name :oop |
            self
                importMethodDictionary: oop parsedClass methodDictionary inside:
 oop methodDictionary classOOP: oop parsedClass: oop parsedClass;
                importMethodDictionary: oop parsedClass asMetaclass methodDictio
nary inside: oop oopClass methodDictionary classOOP: oop parsedClass: oop parsed
Class asMetaclass ]
    ]

    createClassOOPs [
        <category: 'bootstrapping class'>

        self initializeClassOOPs.

        self classOOP do: [ :oop |
            self
                fillEmptyClassOOP: oop;
                fillEmptyMetaclassOOP: oop ]
    ]

    initializeClassOOPs [
        <category: 'bootstrapping class'>

        self initializeBasicSize.

        self classOOP do: [ :oop |
            self
                fixSuperclass: oop;
                fixClassSize: oop ].

        (self classOOP at: #Object) parsedClass asMetaclass superclass: (self cl
assOOP at: #Class) parsedClass
    ]

    initializeBasicSize [
        <category: 'bootstrapping class'>

        behaviorSize := (self classOOP at: #Behavior) parsedClass instVarNames s
ize.
        classDescSize := (self classOOP at: #ClassDescription) parsedClass instV
arNames size.
        classSize := (self classOOP at: #Class) parsedClass instVarNames size.
    ]

    fixSuperclass: anOOP [
        <category: 'bootstrapping class'>

        anOOP parsedClass superclass class name = #ProxyNilClass ifFalse: [
            ((self classOOP at: anOOP parsedClass superclass name) parsedClass c
lass name) =  #ProxyNilClass
                    ifFalse: [ anOOP parsedClass superclass: ((self classOOP at:
 anOOP parsedClass superclass name) parsedClass).
                               anOOP parsedClass asMetaclass superclass: ((self
classOOP at: anOOP parsedClass superclass name) parsedClass asMetaclass) ] ]
    ]

    fixClassSize: anOOP [
        <category: 'bootstrapping class'>

        anOOP
            oopInstVarSize: behaviorSize + classDescSize + classSize + (anOOP pa
rsedClass asMetaclass instVarNames size);
            oopClass: (self initialize: MetaclassOOP new class: MetaclassOOP nam
e)
    ]
```

CorGenCode

```
    fillEmptyClassOOP: anOOP [
        <category: 'bootstrapping class'>

        | binding |
        anOOP parsedClass name printNl.
        anOOP
            superClass: (anOOP parsedClass superclass class name = #ProxyNilClas
s
                                                            ifTrue: [ nilOOP ]
                                                            ifFalse: [ self classOOP
 at: anOOP parsedClass superclass name ]);
            methodDictionary: self instantiateMethodDic;
            instanceSpec: (self classSize: anOOP);
            subClasses: self instantiateSet;
            instanceVariables: self instantiateArray;
            name: (self importSymbol: anOOP parsedClass name);
            comment: (anOOP parsedClass comment ifNil: [ nilOOP ] ifNotNil: [ se
lf instantiateString ]);
            category: (anOOP parsedClass category ifNil: [ nilOOP ] ifNotNil: [
self instantiateString ]);
            environment: smalltalkOOP;
            classVariables: (anOOP parsedClass asMetaclass instVarNames isEmpty
ifTrue: [ nilOOP ] ifFalse: [ self instantiateDictionary ]);
            sharedPools: (anOOP parsedClass sharedPools isEmpty ifTrue: [ nilOOP
 ] ifFalse: [ self instantiateDictionary ]);
            pragmaHandlers: nilOOP.

        self importSubclasses: anOOP parsedClass inside: anOOP subClasses.
        self importInstVarNames: anOOP parsedClass instVarNames inside: anOOP in
stanceVariables.

        anOOP parsedClass comment ifNotNil: [ :cmt | self importString: cmt insi
de: anOOP comment ].
        anOOP parsedClass category ifNotNil: [ :cat | self importString: cat ins
ide: anOOP category ].
    ]

    fillEmptyMetaclassOOP: anOOP [
        <category: 'bootstrapping class'>


        | metaOOP |
        metaOOP := anOOP oopClass.

        metaOOP oopClass: (self classOOP at: #Metaclass).

        metaOOP
            superClass: (anOOP parsedClass superclass class name = #ProxyNilClas
s
                                                            ifTrue: [ self classOOP
at: #Class ]
                                                            ifFalse: [ (self classOO
P at: anOOP parsedClass superclass name) oopClass ]);
            methodDictionary: self instantiateMethodDic;
            instanceSpec: 0;
            subClasses: self instantiateSet;
            instanceVariables: self instantiateArray;
            instanceClass: anOOP.

        self importMetaSubclasses: anOOP parsedClass insideClass: anOOP oopClass
 subClasses.
        self importInstVarNames: anOOP parsedClass asMetaclass instVarNames insi
de: metaOOP instanceVariables.
    ]

    importMethodDictionary: aMethodDictionary inside: anOOP classOOP: aClassOOP
parsedClass: aPClass [
        <category: 'converting'>

        | i array size |
        i := 0.
        array := anOOP oopInstVarAt: 1 put: self instantiateArray.
        array oopArray: (size := aMethodDictionary size).
        aMethodDictionary do: [ :each |
```

CorGenCode

```
            | assoc methodOOP methodInfoOOP method |

            i := i + 1.
            method := STInST.STCompiler compile: each node asMethodOf: aPClass "
aClassOOP parsedClass" classified: nil parser: (STInST.RBParser new) environment
: smalltalkOOP.
            (methodOOP := self initialize: CompiledMethodOOP new class: Compiled
MethodOOP name)
                literals: (self extractLiterals: method method: methodOOP);
                stackDepth: method stackDepth;
                numTemporaries: method numTemporaries;
                numArgs: method numArgs;
                primitive: method primitive;
                methodInfo: (methodInfoOOP := self buildMethodInfoOOP: aClassOOP
 method: method).

            method do: [ :bc | methodOOP oopAdd: bc ].

            (assoc := self instantiate: #Association)
                oopInstVarAt: 1 put: (methodInfoOOP oopInstVarAt: 4);
                oopInstVarAt: 2 put: methodOOP.

            self hashString: (methodInfoOOP oopInstVarAt: 4) add: assoc into: ar
ray ].
        anOOP oopInstVarAt: 2 put: i.
    ]

    importSubclasses: aClass inside: aSetOOP subclasses: aOneArgBlock [

        | size array |
        size := 0.
        array := aSetOOP oopInstVarAt: 1 put: (self instantiateArray).
        self classOOP do: [ :subclass |
            subclass parsedClass superclass class name = #ProxyNilClass ifFalse:
 [
            subclass parsedClass superclass name = aClass name ifTrue: [ size :=
 size + 1 ] ] ].

        array oopArray: size.
        self classOOP do: [ :subclass |
            subclass parsedClass superclass class name = #ProxyNilClass ifFalse:
 [
            subclass parsedClass superclass name = aClass name ifTrue: [
                self hashString: (subclass oopInstVarAt: 6) add: (aOneArgBlock v
alue: subclass) into: array ] ] ].
        aSetOOP oopInstVarAt: 2 put: size
    ]

    importMetaSubclasses: aClass insideClass: aSetOOP [

        self importSubclasses: aClass inside: aSetOOP subclasses: [ :subclass |
subclass oopClass ]
    ]

    importSubclasses: aClass inside: aSetOOP [

        self importSubclasses: aClass inside: aSetOOP subclasses: [ :subclass |
subclass ]
    ]

    importInstVarNames: anArray inside: anArrayOOP [
        anArray do: [ :var | anArrayOOP oopAdd: (self importSymbol: var) ].
    ]

    importString: aString [
        <category: 'converting'>

        ^ self importString: aString inside: self instantiateString
    ]

    importString: aString inside: anOOP [
        <category: 'converting'>

        aString do: [ :each |
            anOOP oopAdd: (self charsOOP oopAt: each asInteger) ].
```

CorGenCode

```
        ^ anOOP
    ]

    importSymbol: aString [
        <category: 'converting'>

        ^ self symbolTableOOP at: aString ifAbsentPut: [ self pimportSymbol: aSt
ring inside: self instantiateSymbol ]
    ]

    pimportSymbol: aString inside: anOOP [
        <category: 'converting'>

        self importString: aString inside: anOOP.
        ^ self symbolTableOOP at: aString asSymbol put: anOOP
    ]

    importBlock: aFakeBlock [
        <category: 'converting'>

        | blockOOP literals |

        (blockOOP := self initialize: CompiledBlockOOP new class: CompiledBlockO
OP name)
            literals: nilOOP;
            stackDepth: aFakeBlock stackDepth;
            numTemporaries: aFakeBlock numTemporaries;
            numArgs: aFakeBlock numArgs.

        aFakeBlock do: [ :bc | blockOOP oopAdd: bc ].
        ^ blockOOP
    ]

    importPIC: aFakePIC [
        <category: 'converting'>

        | picOOP |
        (picOOP := self instantiate: #PolymorphicInlineCaching)
            oopArray: 8;
            oopInstVarAt: 1 put: (self importSymbol: aFakePIC selector).
        ^ picOOP
    ]

    instantiateMethodDic [

        | oop |

        ^ self instantiateOOP
                oopInstVarSize: (self classSize: (self classOOP at: #MethodDicti
onary));
                oopClass: (self classOOP at: #MethodDictionary);
                yourself
    ]

    instantiateSet [

        ^ self instantiate: #Set
    ]

    instantiateArray [

        ^ self initialize: ArrayOOP new class: ArrayOOP name
    ]

    instantiateSymbol [

        ^ self instantiate: #Symbol
    ]

    instantiateString [


        ^ self instantiate: #String
    ]
```

CorGenCode

```
    instantiateDictionary [

        ^ self instantiate: #Dictionary
    ]

    instantiateBlockClosure [

        ^ self instantiate: #BlockClosure
    ]

    instantiate: aSymbol [

        ^ self initialize: self instantiateOOP class: aSymbol
    ]

    initialize: anOOP class: aSymbol [

        ^ anOOP
            oopInstVarSize: (self classSize: (self classOOP at: aSymbol));
            oopClass: (self classOOP at: aSymbol);
            yourself
    ]

    setupCharacter [

        (self classOOP at: #Character) oopInstVarAt: 13 put: self charsOOP
    ]

    setupSymbol [

        | array table size |
        (table := self instantiateSet)
                  oopInstVarAt: 1 put: (array := self instantiateArray);
                  oopInstVarAt: 2 put: self symbolTableOOP size.

        array oopArray: (size := self symbolTableOOP size).

        self symbolTableOOP do: [ :each | self hashString: each add: each into:
array ].

        (self classOOP at: #Symbol) oopInstVarAt: 13 put: table
    ]

    hashAdd: anOOP to: anArray from: anInteger [
        | i |
        i := anInteger + 1.
        [ i <= anArray oopArray size ] whileTrue: [ (anArray oopAt: i) = nilOOP
ifTrue: [ ^ anArray oopAt: i put: anOOP ].
            i := i + 1 ].
        i := 1.
        [ i < anInteger ] whileTrue: [ (anArray oopAt: i) = nilOOP ifTrue: [ ^ a
nArray oopAt: i put: anOOP ].
            i := i + 1 ].
        self error: 'Impossible to add the item'
    ]

    setupProcessor [

        | processorOOP processOOP |
        processOOP := self setupProcess.
        processorOOP := self initialize: ProcessorSchedulerOOP new class: Proces
sorSchedulerOOP name.
        processorOOP
            scheduler: nilOOP;
            processes: processOOP;
            activeProcess: processOOP;
            idleTasks: nilOOP.
        (self classOOP at: #ProcessorScheduler)
                            oopInstVarAt: 13 put: processorOOP

    ]

    setupNil [

        nilOOP
```

```
            oopInstVarSize: 0;
            oopClass: (self classOOP at: #UndefinedObject)
    ]

    setupProcess [

        | processOOP |
        (processOOP := self initialize: ProcessOOP new class: ProcessOOP name)
            nextLink: nilOOP;
            suspendedContext: self setupBootstrapContext;
            priority: 4;
            myList: nilOOP;
            name: (self instantiate: #String);
            interrupts: nilOOP;
            interruptLock: nilOOP.

        self importString: 'Bootstrap' inside: processOOP name.

        ^ processOOP
    ]

    setupBootstrapContext [

        | find |
    (contextOOP := self initialize: MethodContextOOP new class: MethodContextOOP name)
                                parent: nilOOP;
                                ip: 1;
                                sp: 1;
                                receiver: (self classOOP at: #Bootstrap);
                                method: nilOOP;
                                flags: 0.

        find := false.
        (((self classOOP at: #Bootstrap) oopClass oopInstVarAt: 2) oopInstVarAt: 1) oopDo: [ :assoc |
            | cmpMth |
            cmpMth := assoc oopInstVarAt: 2.
            (self equal: ((cmpMth oopInstVarAt: 6) oopInstVarAt: 4) and: 'initialize') ifTrue: [
                find := true.
                contextOOP oopInstVarAt: 5 put: cmpMth.
                (cmpMth oopInstVarAt: 3) timesRepeat: [ contextOOP oopAdd: 0 ]. " Temps "
                (cmpMth oopInstVarAt: 2) timesRepeat: [ contextOOP oopAdd: 0 ] ] " Stack depth " ].

        contextOOP oopAdd: 0.
        contextOOP oopAdd: 0.
        contextOOP oopAdd: 0.
        contextOOP oopAdd: 0.
        contextOOP oopAdd: 0.
        contextOOP oopAdd: 0.
        contextOOP oopAdd: 0.

        find ifFalse: [ self error: 'Boostrap class not found' ].
        ^ contextOOP
    ]

    equal: anOOP and: aString [

        aString size = anOOP oopArray size ifFalse: [ ^ false ].
        1 to: aString size do: [ :i |
            (aString at: i) value = ((anOOP oopAt: i) oopInstVarAt: 1)  ifFalse: [ ^ false ] ].
        ^ true
    ]

    buildMethodInfoOOP: aClassOOP method: aFakeCompiledMethod [
        <category: 'method'>

            | methodInfoOOP |
            (methodInfoOOP := self initialize: MethodInfoOOP new class: MethodInfoOOP name)
                sourceCode: nilOOP;
                category: self instantiateString;
                classOOP: aClassOOP;
                selector: (self importSymbol: aFakeCompiledMethod selector).
            self importString: aFakeCompiledMethod methodCategory inside: (methodInfoOOP oopInstVarAt: 2).
            ^ methodInfoOOP
    ]

    extractLiterals: aFakeCompiledMethod method: aMethodOOP[
        <category: 'method'>

        | literals |
        literals := self instantiateArray.
        aFakeCompiledMethod literals
                    do: [ :each | | oop |
                        oop := literals oopAdd: (each asGSTOop: self).
                        each isFakeBlock ifTrue: [ oop
                                                    oopInstVarAt: 1 put: literals;
                                                    oopInstVarAt: 5 put: aMethodOOP ].
                        each isFakePIC ifTrue: [oop oopInstVarAt: 2 put: aMethodOOP ] ].
        ^ literals
    ]

    hashString: anOOP [
        <category: 'oop hash'>

        | sum |
        sum := 0.
        anOOP oopDo: [ :each |
            sum := sum + (each oopInstVarAt: 1) ].
        ^ sum bitAnd: SmallInteger largest
    ]

    hashString: aStringOOP add: anOOP into: anArrayOOP [
        <category: 'oop hash'>

        | pos |
        pos := ((self hashString: aStringOOP) \\ anArrayOOP oopArray size) + 1.

        (anArrayOOP oopAt: pos) = nilOOP
            ifTrue: [ anArrayOOP oopAt: pos put: anOOP ]
            ifFalse: [ self hashAdd: anOOP to: anArrayOOP from: pos ]
    ]

    saveImage [

        (GSTImage save: smalltalkOOP named: 'foo.im')
                        platform: GSTia64;
                        nilOOP: nilOOP;
                        falseOOP: falseOOP;
                        trueOOP: trueOOP;
                        save
    ]

    nilOOP [
        <category: 'accessing'>

        ^ nilOOP
    ]

    trueOOP [
        <category: 'accessing'>

        ^ trueOOP
    ]

    falseOOP [
        <category: 'accessing'>

        ^ falseOOP
    ]
```

CorGenCode

CorGenCode

```
    smalltalkOOP [
        <category: 'accessing'>

        ^ smalltalkOOP
    ]
]

"************************************************************
 bootstrap/Compiler.st
 ************************************************************"
STInST.RBProgramNode subclass: Compiler [

    Compiler class >> compile: aMethodNode environment: anEnvironment [

        ^ self new
                environment: anEnvironment;
                methodNode: aMethodNode;
                compile;
                yourself
    ]

    Compiler class >> compile: aMethodNode [

        ^ self compile: aMethodNode environment: Smalltalk
    ]

    Compiler class >> new [

        ^ self basicNew
                initialize;
                yourself
    ]

    | behavior bytecode compiledMethod environment literals methodNode stackDept
h symbol |

    initialize [

        stackDepth := 0.
        bytecode := OrderedCollection new.
        literals := OrderedCollection new.
        symbol := OrderedCollection new.
        environment := (self class environment selectSuperspaces: [ : each | eac
h superspace isNil ]) asArray first
    ]

    environment: anEnvironmnent [

        environment := anEnvironmnent
    ]

    methodNode: aMethodNode [

        methodNode := aMethodNode
    ]

    compile [

        methodNode acceptVisitor: self
    ]

    compiledMethod [

        ^ compiledMethod
    ]

    createCompiledMethod [

        compiledMethod := (environment at: #CompiledMethod) new: bytecode size.
        compiledMethod header: 0 literals: literals.
        1 to: bytecode size do: [ :i |
            compiledMethod at: i put: (bytecode at: i) ]
    ]

    acceptMethodNode: aMethodNode [
```

CorGenCode

```
        symbol addFirst: aMethodNode argumentNames.
        aMethodNode body acceptVisitor: self.
        self createCompiledMethod
    ]

    acceptSequenceNode: aSequenceNode [

        symbol addFirst: aSequenceNode temporaryNames.
        aSequenceNode statements do: [ :each |
            each acceptVisitor: self ]
    ]

    acceptReturnNode: aReturnNode [

        aReturnNode value acceptVisitor: self.
        self bytecode: GSTByteCode.ReturnContextStackTop
    ]

    acceptLiteralNode: aLiteralNode [

        literals addLast: aLiteralNode value.
        self pushLastLiteral
    ]

    pushLastLiteral [

        self pushLiteral: literals size
    ]

    pushLiteral: anInteger [

        stackDepth := stackDepth + 1.
        self bytecode: GSTByteCode.PushLitConstant with: anInteger
    ]

    bytecode: aByteCode with: anInteger [

        bytecode addLast: aByteCode bytecode.
        bytecode addLast: anInteger
    ]

    bytecode: aByteCode [

        bytecode addLast: aByteCode bytecode
    ]

    lookup: aRBVariableNode [

        | node found |
        node := aRBVariableNode parent.
        found := false.
        [ node isNil or: [ found ] ] whileFalse: [
            (found := self tempvarLookup: aRBVariableNode from: node) ifTrue: [
^ true ].
            (found := self argLookup: aRBVariableNode from: node) ifTrue: [ ^ tr
ue ].

            node := node parent ].
        node isNil ifTrue: [ ^ self argLookup: aRBVariableNode ]
    ]

    tempvarLookup: aRBVariableNode from: aNode [

        (aNode isSequence) ifFalse: [ ^ false ].
        (aNode temporaryNames includes: aRBVariableNode name) ifFalse: [ ^ false
 ].
        ^ true
    ]

    argLookup: aRBVariableNode from: aNode [

        (aNode isBlock) ifFalse: [ ^ false ].
        (aNode argumentNames includes: aRBVariableNode name) ifFalse: [ ^ false
].
```

```
        ^ true
    ]

    argLookup: aRBVariableNode [

        (methodNode argumentNames includes: aRBVariableNode name) ifTrue: [ ^ tr
ue ].
        ^ self ivarLookup: aRBVariableNode
    ]

    ivarLookup: aRBVariableNode [

        behavior ifNil: [ ^ false ].
        behavior indexOfInstVar: aRBVariableNode name ifAbsent: [ ^ self classLo
okup: aRBVariableNode ]
    ]

    classLookup: aRBVariableNode [

        | namespace |
        namespace := environment.
        (aRBVariableNode value subStrings: $.) do: [ :each |
            namespace := namespace at: each asSymbol ifAbsent: [ ^ self error: '
lookup is impossible' ] ].
        ^ true
    ]
]

"**************************************************************
 bootstrap/Extends.st
 **************************************************************"
Smalltalk.Object extend [
    isFakeBlock [
        <category: 'testing'>

        ^ false ]

    isFakePIC [
        <category: 'testing'>

        ^ false ]

    asGSTOop: aBootstrap [ ^ self error: 'convertion missed' ]
]

Smalltalk.Symbol extend [
    asGSTOop: aBootstrap [ ^ aBootstrap importSymbol: self ]
]

Smalltalk.UndefinedObject extend [
    asGSTOop: aBootstrap [ ^ aBootstrap nilOOP ]
]

Smalltalk.True extend [
    asGSTOop: aBootstrap [ ^ aBootstrap trueOOP ]
]

Smalltalk.False extend [
    asGSTOop: aBootstrap [ ^ aBootstrap falseOOP ]
]

Smalltalk.String extend [
    asGSTOop: aBootstrap [ ^ aBootstrap importString: self ]
]

Smalltalk.SmallInteger extend [
    asGSTOop: aBootstrap [ ^ self ]
]

STInST.STCompiler class extend [
    | specialIdentifiers symbolOOP bootstrap |

    symbolOOP: aSymbolTable [
        symbolOOP := aSymbolTable
```

CorGenCode

```
    ]

    symbolOOP [
        ^ symbolOOP
    ]

    bootstrap: aBootstrap [
        bootstrap := aBootstrap
    ]

    bootstrap [
        ^ bootstrap
    ]

    specialIdentifiers [
      ^ specialIdentifiers ifNil: [
        specialIdentifiers := (LookupTable new: 8)
                at: 'super' put: [ :c | c compileError: 'invalid occurrence of s
uper' ];
                at: 'self'  put: [ :c | c compileByte: GSTByteCode.PushSelf ];
                at: 'nil'   put: [ :c | c compileByte: GSTByteCode.PushSpecial a
rg: VMOtherConstants.NilIndex ];
                at: 'true'  put: [ :c | c compileByte: GSTByteCode.PushSpecial a
rg: VMOtherConstants.TrueIndex ];
                at: 'false' put: [ :c | c compileByte: GSTByteCode.PushSpecial a
rg: VMOtherConstants.FalseIndex ];
                at: 'thisContext' put: [ :c | c
                                                pushLiteralVariable: #{ContextPa
rt};
                                                compileByte: GSTByteCode.SendImm
ediate
                                                arg: VMOtherConstants.ThisContex
tSpecial ];
                yourself ]
    ]
]

STInST.STCompiler extend [

    | fakeMethod |

    acceptBlockNode: aNode [
    "STBlockNode has a variable that contains a string for each parameter,
     and one that contains a list of statements. Here is how STBlockNodes
     are compiled:

     push BlockClosure or CompiledBlock literal
     make dirty block                    <--- only if pushed CompiledBlock

     Statements are put in a separate CompiledBlock object that is referenced
     by the BlockClosure that the sequence above pushes or creates.

     compileStatements: creates the bytecodes.  It is this method that is
     called by STCompiler>>bytecodesFor: and STCompiler>>bytecodesFor:append:"

    <category: 'visiting RBBlockNodes'>
    | bc depth block clean |
    depth := self depthSet: aNode arguments size + aNode body temporaries size.
    aNode body statements isEmpty
        ifTrue: [aNode body addNode: (RBLiteralNode value: nil)].
    bc := self insideNewScopeDo:
            [self bytecodesFor: aNode
                atEndDo:
                [aNode body lastIsReturn ifFalse: [self compileByte: GSTByteCode
.ReturnContextStackTop ] ] ].
    block := Bootstrap.FakeCompiledBlock
            literals: symTable literals
            numArgs: aNode arguments size
            numTemps: aNode body temporaries size
            attributes: #()
            bytecodes: bc
            depth: self maxDepth.
    block method: fakeMethod.
      block class printNl.
        self pushLiteral: block.
```

CorGenCode

```
        self compileByte: GSTByteCode.MakeDirtyBlock
    ]

    acceptCascadeNode: aNode [
        "RBCascadeNode holds a collection with one item per message."

        <category: 'visiting RBCascadeNodes'>
        | messages first |
        messages := aNode messages.
        first := messages at: 1.
        first receiver = SuperVariable
            ifTrue:
                [aNode messages do: [:each | self compileSendToSuper: each]
                    separatedBy:
                        [self
                            depthDecr: 1;
                            compileByte: GSTByteCode.PopStackTop].
                ^aNode].
        first receiver acceptVisitor: self.
        self
            depthIncr;
            compileByte: GSTByteCode.DupStackTop.
        self compileMessage: first.
        messages
            from: 2
            to: messages size - 1
            do:
                [:each |
                self
                    compileByte: GSTByteCode.PopStackTop;
                    compileByte: GSTByteCode.DupStackTop.
                self compileMessage: each].
        self
            depthDecr: 1;
            compileByte: GSTByteCode.PopStackTop.
        self compileMessage: messages last
    ]

    acceptMethodNode: node [
    <category: 'visiting RBMethodNodes'>
    | statements attributes |
    fakeMethod := Bootstrap.FakeCompiledMethod new.
    node body addSelfReturn.
    depth := maxDepth := 0.
    self declareArgumentsAndTemporaries: node.
    self compileStatements: node body.
    self undeclareArgumentsAndTemporaries: node.
    symTable finish.
    attributes := self compileMethodAttributes: node primitiveSources.
    "method := Bootstrap.FakeCompiledMethod "
    fakeMethod
            literals: symTable literals
            numArgs: node arguments size
            numTemps: node body temporaries size
            attributes: attributes
            bytecodes: bytecodes contents
            depth: maxDepth + node body temporaries size + node arguments size +
 1.
    "(method descriptor)"
    (fakeMethod descriptor)
        setSourceCode: node source asSourceCode;
        methodClass: symTable environment;
        selector: node selector.
    attributes do: [ :ann | | handler error |
            "ann selector = #primitive: ifTrue: [ method primitive: ann argument
s contents first ]"
            ann selector = #primitive: ifTrue: [ fakeMethod primitive: ann argum
ents contents first ] ].
    ^fakeMethod
    ]

    acceptVariableNode: aNode [
        <category: 'visiting RBVariableNodes'>
        | locationType definition |
        self depthIncr.
```

CorGenCode

```
        self class specialIdentifiers at: aNode name
            ifPresent:
                [:block |
                block value: self.
                ^aNode].
        definition := self lookupName: aNode name.
        (symTable isTemporary: aNode name)
            ifTrue:  [ ^ self compilePushTemporary: definition scopes: (symTable
 outerScopes: aNode name) ].
        (symTable isReceiver: aNode name)
            ifTrue:
                [ self compileByte: GSTByteCode.PushReceiverVariable arg: defini
tion.
                  ^ aNode ].
        self compileByte: GSTByteCode.PushLiteralVariable arg: definition
    ]

    class: aBehavior parser: aParser [
        <category: 'private'>
        destClass := aBehavior.
        symTable := STSymbolTable new.
        parser := aParser.
        bytecodes := WriteStream on: (Array new: 240).
        isInsideBlock := 0.
        symTable declareEnvironment: aBehavior
    ]

    bytecodesFor: aBlockNode atEndDo: aBlock [
        <category: 'accessing'>
        | saveBytecodes result |
        saveBytecodes := bytecodes.
        bytecodes := WriteStream on: (Array new: 240).
        self declareArgumentsAndTemporaries: aBlockNode.
        self compileStatements: aBlockNode body.
        self undeclareArgumentsAndTemporaries: aBlockNode.
        aBlock value.
        result := bytecodes contents.
        bytecodes := saveBytecodes.
        ^result
    ]

    compileByte: aByte arg: arg [
        <category: 'accessing'>
        bytecodes
            nextPut: aByte bytecode + (arg bitShift: 8)
    ]

    compileByte: aByte arg: arg1 arg: arg2 [
        <category: 'accessing'>
        bytecodes
            nextPut: aByte bytecode + (arg1 bitShift: 8) + (arg2 bitShift: 16)
    ]

    compileAssignmentFor: aNode [
        "RBVariableNode has one instance variable, the name of the variable
         that it represents."

        <category: 'visiting RBVariableNodes'>
        | definition |
        self checkStore: aNode name.
        definition := self lookupName: aNode name.
        (symTable isTemporary: aNode name)
            ifTrue:  [ ^ self compileStoreTemporary: definition scopes: (symTabl
e outerScopes: aNode name)].
        (symTable isReceiver: aNode name)
            ifTrue: [ ^ self compileByte: GSTByteCode.StoreReceiverVariable arg:
 definition ].
        self compileByte: GSTByteCode.StoreLitVariable arg: definition.
        self compileByte: GSTByteCode.PopStackTop.
        self compileByte: GSTByteCode.PushLitVariable arg: definition
    ]

    compileBoolean: aNode longBranch: bc1 returns: ret1 shortBranch: bc2 longIfT
rue: longIfTrue [
        <category: 'compiling'>
```

CorGenCode

```
        self compileJump: bc1 size + (ret1 ifTrue: [0] ifFalse: [2])
            if: longIfTrue not.
        self nextPutAll: bc1.
        ret1 ifFalse: [self compileByte: GSTByteCode.Jump arg: bc2 size].
        self nextPutAll: bc2.
        ^true
    ]

    compileIfFalse: bcFalse returns: bcFalseReturns ifTrue: bcTrue [
        <category: 'compiling'>
        | falseSize |
        falseSize := bcFalseReturns
                    ifTrue: [bcFalse size]
                    ifFalse: [bcFalse size + (self sizeOfJump: bcTrue size)].
        self compileJump: falseSize if: true.
        self nextPutAll: bcFalse.
        bcFalseReturns ifFalse: [self compileByte: GSTByteCode.Jump arg: bcTrue
size].
        self nextPutAll: bcTrue.
        ^true
    ]

    compileIfTrue: bcTrue returns: bcTrueReturns ifFalse: bcFalse [
        <category: 'compiling'>
        | trueSize |
        trueSize := bcTrueReturns
                    ifTrue: [bcTrue size]
                    ifFalse: [bcTrue size + (self sizeOfJump: bcFalse size)].
        self compileJump: trueSize if: false.
        self nextPutAll: bcTrue.
        bcTrueReturns ifFalse: [self compileByte: GSTByteCode.Jump arg: bcFalse
size].
        self nextPutAll: bcFalse.
        ^true
    ]

    compileJump: displacement if: jmpCondition [
        <category: 'accessing'>
        displacement < 0
            ifTrue:
                ["Should not happen"

                ^self error: 'Cannot compile backwards conditional jumps'].
        self depthDecr: 1.
        jmpCondition
            ifFalse: [self compileByte: GSTByteCode.PopJumpFalse arg: displaceme
nt]
            ifTrue: [self compileByte: GSTByteCode.PopJumpTrue arg: displacement
]
    ]

    compileMessage: aNode [
    "RBMessageNode contains a message send. Its instance variable are
     a receiver, selector, and arguments.  The receiver has already
     been compiled."

    <category: 'compiling'>
    | args litIndex |
    aNode arguments do: [:each | each acceptVisitor: self].
    litIndex := self addLiteral: (Bootstrap.FakeCompiledPIC selector: aNode sele
ctor).
    self
        compileByte: GSTByteCode.Send arg: litIndex arg: aNode arguments size
    ]

    compilePushTemporary: number scopes: outerScopes [
        <category: 'visiting RBVariableNodes'>
        outerScopes = 0
            ifFalse:
                [self
                    compileByte: GSTByteCode.PushOuterVariable
                    arg: number
                    arg: outerScopes.
                ^self].
        self compileByte: GSTByteCode.PushTemporaryVariable arg: number
```

CorGenCode

```
    ]

    compileSendToSuper: aNode [
        <category: 'compiling'>
        | litIndex args |
        self
            depthIncr;
            compileByte: GSTByteCode.PushSelf.
        aNode arguments do: [:each | each acceptVisitor: self].
        litIndex := self addLiteral: aNode selector.
        args := aNode arguments size.
        self
            compileByte: GSTByteCode.SendSuper
            arg: litIndex
            arg: args.
        self depthDecr: aNode arguments size
    ]

    compileStoreTemporary: number scopes: outerScopes [
        <category: 'visiting RBVariableNodes'>
        outerScopes = 0
            ifFalse:
                [self
                    compileByte: GSTByteCode.StoreOuterTemporary
                    arg: number
                    arg: outerScopes.
                ^self].
        self compileByte: GSTByteCode.StoreTemporaryVariable arg: number
    ]

    compileMethodAttributes: attributes [
        <category: 'compiling method attributes'>

        ^ attributes asArray collect: [:each | self compileAttribute: (RBScanner
 on: each readStream)].
    ]

   compileAttribute: scanner [
        <category: 'compiling method attributes'>
        | currentToken selectorBuilder selector arguments argParser node |
        currentToken := self scanTokenFrom: scanner.
        (currentToken isBinary and: [currentToken value == #<])
            ifFalse: [^self compileError: 'method attributes must begin with ''<
'''].
        selectorBuilder := WriteStream on: String new.
        arguments := WriteStream on: Array new.
        currentToken := self scanTokenFrom: scanner.
        [currentToken isBinary and: [currentToken value == #>]] whileFalse:
            [currentToken isKeyword
                ifFalse: [^self compileError: 'keyword expected in method attrib
ute'].
        selectorBuilder nextPutAll: currentToken value.
        argParser := RBParser new.
        argParser errorBlock: parser errorBlock.
        argParser scanner: scanner.
        node := argParser parseBinaryMessageNoGreater.
        node := RBSequenceNode statements: {node}.
        arguments nextPut: (node statements first isLiteral
                                ifTrue: [ self convertLiteral: node statements f
irst value ]
                                ifFalse: [ self convertLiteral: (Primitives.Prim
itive numberFor: node statements first name asSymbol) ]).
        currentToken := argParser currentToken].
        selector := selectorBuilder contents asSymbol.
        ^ Message selector: selector arguments: arguments
    ]

    compileStatements: aNode [
        <category: 'visiting RBBlockNodes'>
        aNode statements keysAndValuesDo:
                [:index :each |
                index = 1
                    ifFalse:
                        [self
                            depthDecr: 1;
```

CorGenCode

```
                        compileByte: GSTByteCode.PopStackTop].
                each acceptVisitor: self].
        aNode statements isEmpty
            ifTrue:
                [self
                    depthIncr;
                    compileByte: GSTByteCode.PushSpecial arg: NilIndex]
    ]

    acceptReturnNode: aNode [
        <category: 'compiling'>

        aNode value acceptVisitor: self.
        self isInsideBlock
            ifTrue: [ self compileByte: GSTByteCode.ReturnMethodStackTop ]
            ifFalse: [ self compileByte: GSTByteCode.ReturnContextStackTop ]
    ]

    pushLiteral: value [
        <category: 'accessing'>
        | definition |
        definition := self addLiteral: value.
        self compileByte: GSTByteCode.PushLitConstant arg: definition
    ]

    addLiteral: literal [
        <category: 'accessing'>

        " Convert literal as OOP "
        ^ symTable addLiteral: literal
    ]

    convertLiteral: aLiteral [
        aLiteral isSymbol ifTrue: [ ^ self class bootstrap importSymbol: aLitera
l ].
        aLiteral isNil ifTrue: [ ^ self class bootstrap nilOOP ].
        aLiteral = true ifTrue: [ ^ self class bootstrap trueOOP ].
        aLiteral = false ifTrue: [ ^ self class bootstrap falseOOP ].
        aLiteral isString ifTrue: [ ^ self class bootstrap importString: aLitera
l ].
        aLiteral isInteger ifTrue: [ ^ aLiteral ].
        ^ aLiteral
    ]
]

STInST.STSymbolTable class extend [
    | envOOP bootstrap |

    environmentOOP: aDictionary [
        envOOP := aDictionary
    ]

    environmentOOP [
        ^ envOOP
    ]

    bootstrap: aBootstrap [
        bootstrap := aBootstrap
    ]

    bootstrap [
        ^ bootstrap
    ]
]

STInST.STLiteralsTable extend [
    | pos |
    initialize: aSize [
        <category: 'private'>

        array := OrderedCollection new: aSize.
        pos := -1
    ]

    addLiteral: anObject [
```

CorGenCode

```
        <category: 'accessing'>

        pos := pos + 1.
        array addLast: anObject.
        ^ pos
    ]

    literals [
        <category: 'accessing'>
        ^ array
    ]

    trim [
        <category: 'accessing'>
    ]
]

STInST.STSymbolTable extend [

    lookupPoolsFor: symbol [
        <category: 'accessing'>

        symbol = #Smalltalk ifTrue: [ ^ self class bootstrap smalltalkOOP ].
        ^ pools at: symbol ifAbsent: [ nil ]
    ]

    addPool: poolDictionary [
        <category: 'declaring'>
    ]

    declareGlobals [
        <category: 'declaring'>
        pools := self class environmentOOP.
    ]
    declareEnvironment: aBehavior [
    <category: 'declaring'>
    | i |
    environment := aBehavior.
    i := -1.
    aBehavior withAllSuperclasses reverseDo:
        [ :class |
        class instVarNames do:
            [:iv |
            instVars at: iv asSymbol
                put: (STVariable
                    id: (i := i + 1)
                    scope: 0
                    canStore: true )]].
    self declareGlobals
    ]
]

STInST.STClassLoaderObjects.ProxyNilClass extend [
    superclass [
        ^ nil
    ]

    instVarNames [
        ^ #()
    ]

    allSharedPoolDictionariesDo: aBlock [
        "Answer the shared pools visible from methods in the metaclass,
         in the correct search order."

    ]
]

STInST.STClassLoaderObjects.LoadedBehavior extend [

    superclass: anObject [
        <category: 'accessing'>

        superclass := anObject
    ]
```

```
    allSuperclasses [
    "Answer all the receiver's superclasses in a collection"

    <category: 'accessing class hierarchy'>
    | supers |
    supers := OrderedCollection new.
    self allSuperclassesDo: [:superclass | supers addLast: superclass].
    ^supers
    ]

    withAllSuperclasses [
    "Answer the receiver and all of its superclasses in a collection"

    <category: 'accessing class hierarchy'>
    | supers |
    supers := OrderedCollection with: self.
    self allSuperclassesDo: [:superclass | supers addLast: superclass].
    ^supers
    ]

    allSharedPoolDictionariesDo: aBlock [
        "Answer the shared pools visible from methods in the metaclass,
         in the correct search order."

        self superclass allSharedPoolDictionariesDo: aBlock
    ]

    poolResolution [
        "Answer a PoolResolution class to be used for resolving pool
         variables while compiling methods on this class."

        <category: 'compiling methods'>
        ^ STInST.PoolResolution current
    ]

    addSelector: aSymbol withMethod: aCompiledMethod [

        ^ aSymbol->aCompiledMethod
    ]

    pragmaHandlerFor: aSymbol [
        ^ [ :x :y | ]
    ]

    printOn: aStream in: aNamespace [
    "Answer the class name when the class is referenced from aNamespace
     - a dummy one, since Behavior does not support names."

        <category: 'support for lightweight classes'>
        aStream nextPutAll: (self nameIn: aNamespace)
    ]
]

"**************************************************************
 bootstrap/FakeCompiledBlock.st
 **************************************************************"
" Copyright 2010 GST.
  Written by Gwenael Casaccio

  This file is part of GST.

  GST is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GST is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GST.  If not, see <http://www.gnu.org/licenses/>. "
```

CorGenCode

```
Object subclass: FakeCompiledBlock [
    | bc literals descriptor method numArgs numTemps stackDepth |

    <shape: #pointer>

    FakeCompiledBlock class >> literals: lits numArgs: numArg numTemps: numTemp
attributes: attrArray bytecodes: bytecodes depth: depth [
        <category: 'instance creation'>

        ^ (self basicNew: bytecodes size)
            literals: lits numArgs: numArg numTemps: numTemp attributes: attrArr
ay bytecodes: bytecodes depth: depth;
            yourself
    ]

    literals: lits numArgs: anInteger numTemps: numTemp attributes: attrArray by
tecodes: bytecodes depth: depth [
        (1 to: bytecodes size) do: [ :i |
            self add: (bytecodes at: i) ].
        numArgs := anInteger.
        stackDepth := depth.
        literals := lits.
        numTemps := numTemp.
    ]

    bc [
        <category: 'accessing'>

        ^ bc ifNil: [ bc := OrderedCollection new ]
    ]

    add: aByte [

        self bc add: aByte
    ]

    at: anInteger [
        ^ self bc at: anInteger
    ]

    size [
        ^ self bc size
    ]

    isFakeBlock [
        <category: 'testing'>

        ^ true
    ]

    asGSTOop: aBootstraper [
        <category: 'converting'>

        ^ aBootstraper importBlock: self
    ]

    method: aFakeCompiledMethod [
        <category: 'accessing'>

        method := aFakeCompiledMethod
    ]

    method [
        <category: 'accessing'>

        ^ method
    ]

    literals [
        <category: 'accessing'>

        ^ literals
    ]
```

```
    numArgs [
        <category: 'accessing'>

        ^ numArgs
    ]

    numTemporaries [
        <category: 'accessing'>

        ^ numTemps
    ]

    stackDepth [
        <category: 'accessing'>

        ^ stackDepth
    ]

    do: aOneArgBlock [
        1 to: self size do: [ :i | aOneArgBlock value: (self at: i) ]
    ]

    sendTo: anObject [
        "anObject inspect"
    ]

    needToMakeDirty [
        self do: [ :bc |
            (bc bitAnd: 255) printNl.
            ({GSTByteCode.PushOuterVariable bytecode. GSTByteCode.StoreOuterTemp
orary bytecode. GSTByteCode.PushSelf bytecode} includes: (bc bitAnd: 255)) ifTru
e: [ ^ true ] ].
        ^ false
    ]
]


"**************************************************************
 bootstrap/FakeCompiledMethod.st
 **************************************************************"
" Copyright 2010 GST.
  Written by Gwenael Casaccio

  This file is part of GST.

  GST is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GST is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GST.  If not, see <http://www.gnu.org/licenses/>. "

Object subclass: FakeCompiledMethod [
    | bc literals descriptor numTemps stackDepth primitive numArgs |

    <shape: #pointer>

    FakeCompiledMethod class >> literals: lits numArgs: numArg numTemps: numTemp
 attributes: attrArray bytecodes: bytecodes depth: depth [
        <category: 'instance creation'>

        ^ (self basicNew: bytecodes size)
            literals: lits numArgs: numArg numTemps: numTemp attributes: attrArr
ay bytecodes: bytecodes depth: depth;
            yourself
    ]

    literals: lits numArgs: anInteger numTemps: numTemp attributes: attrArray by
tecodes: bytecodes depth: depth [
```

CorGenCode

```
        bc := OrderedCollection new.
        (1 to: bytecodes size) do: [ :i |
            self add: (bytecodes at: i) ].
        numArgs := anInteger.
        stackDepth := depth.
        literals := lits.
        numTemps := numTemp.
        primitive := 0.
    ]

    bc [
        <category: 'accessing'>

        ^ bc ifNil: [ bc := OrderedCollection new ]
    ]

    add: aByte [

        self bc add: aByte
    ]

    at: anInteger [
        ^ self bc at: anInteger
    ]

    size [
        ^ self bc size
    ]

    descriptor [
        ^ descriptor ifNil: [ descriptor := MethodInfo new ]
    ]

    literals [
        ^ literals
    ]

    methodCategory [
        ^ self descriptor category
    ]

    selector [
        ^ self descriptor selector
    ]

    numArgs [
        <category: 'accessing'>

        ^ numArgs
    ]

    numTemporaries [
        <category: 'accessing'>

        ^ numTemps
    ]

    primitive: anInteger [
        <category: 'accessing'>

        primitive := anInteger
    ]

    primitive [
        <category: 'accessing'>

        ^ primitive
    ]

    stackDepth [
        <category: 'accessing'>

        ^ stackDepth
    ]
```

CorGenCode

```
    do: aOneArgBlock [
        1 to: self size do: [ :i | aOneArgBlock value: (self at: i) ]
    ]

    sendTo: anObject [
        "anObject inspect"
    ]
]


"************************************************************
 bootstrap/FakeCompiledPIC.st
 ************************************************************"
Object subclass: FakeCompiledPIC [

    FakeCompiledPIC class >> selector: aSymbol [
        <category: ’instance creation’>

        ^ self new
                selector: aSymbol;
                yourself
    ]

    | selector |

    isFakePIC [
        <category: ’testing’>

        ^ true
    ]

    selector: aSymbol [
        <category: ’accessing’>

        selector := aSymbol
    ]

    selector [
        <category: ’accessing’>

        ^ selector
    ]

    asGSTOop: aBootstraper [
        <category: ’converting’>

        ^ aBootstraper importPIC: self
    ]
]


"************************************************************
 bootstrap/GSTImage.st
 ************************************************************"
" Copyright 2010 GST.
  Written by Gwenael Casaccio

  This file is part of GST.

  GST is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GST is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GST.  If not, see <http://www.gnu.org/licenses/>. "

Object subclass: GSTImage [
    | name rootOOP nilOOP trueOOP falseOOP toSave visitedItems platform position
 os |
```

CorGenCode

```
    GSTImage class >> header [
        <category: ’accessing’>

        ^ ’GST’
    ]

    GSTImage class >> version [
        <category: ’accessing’>

        ^ ’0.0.0’
    ]

    GSTImage class >> load: aString [
        <category: ’instance creation’>

        ^ self new
            initialize;
            name: aString;
            yourself
    ]

    GSTImage class >> save: aSmalltalkOOP named: aString [
        <category: ’instance creation’>

        ^ self new
            initialize;
            rootOOP: aSmalltalkOOP;
            name: aString;
            yourself
    ]

    initialize [
        <category: ’initialization’>

        position := 0.
        visitedItems := Dictionary new.
        toSave := OrderedCollection new
    ]

    rootOOP: anOOP [
        <category: ’accessing’>

        rootOOP := anOOP
    ]

    name: aString [
        <category: ’accessing’>

        name := aString
    ]

    nilOOP: anOOP [
        <category: ’accessing’>

        nilOOP := anOOP
    ]

    falseOOP: anOOP [
        <category: ’accessing’>

        falseOOP := anOOP
    ]

    trueOOP: anOOP [
        <category: ’accessing’>

        trueOOP := anOOP
    ]

    save [
        <category: ’image saving’>

        | stream |
```

```
        stream := name asFile writeStream.
        self
            writeHeader: stream.
        position := stream position + (self sizeOf: nilOOP) + (self sizeOf: trueOOP) + (self sizeOf: falseOOP) + (self sizeOf: rootOOP).
        self
            writeSpecialObjects: stream.

        [ toSave isEmpty ] whileFalse: [
            self writeOOP: toSave removeFirst in: stream ].
        stream close.
    ]

    writeHeader: aStream [
        <category: 'image saving'>

        aStream nextPutAll: self class header.
        aStream nextPutByte: self class version size.
        aStream nextPutAll: self class version.
    ]

    readHeader: aStream [
        <category: 'image loading'>

        | length |
        (aStream nextByteArray: 3) asString = self class header ifFalse: [ ^ self error: 'Bad header' ].
        length := aStream nextByte.
        (aStream nextByteArray: length) asString = self class version ifFalse: [ ^ self error: 'Bad version' ].

        position := aStream position
    ]


    writeSpecialObjects: aStream [

        visitedItems at: nilOOP put: aStream position.
        self writeOOP: nilOOP in: aStream.
        visitedItems at: trueOOP put: aStream position.
        self writeOOP: trueOOP in: aStream.
        visitedItems at: falseOOP put: aStream position.
        self writeOOP: falseOOP in: aStream.
        visitedItems at: rootOOP put: aStream position.
        self writeOOP: rootOOP in: aStream.
    ]

    sizeOf: anOOP [
        <category: 'accessing'>

        ^ platform wordSize * 4 + (anOOP oopInstVarSize * platform wordSize) + (anOOP oopArray size * platform wordSize)
    ]

    itemToBeVisited: anOOP [

        | writePos |
        writePos := position.
        toSave addLast: anOOP.
        position := position + (self sizeOf: anOOP).
        ^ writePos
    ]

    writeOOP: anOOP in: aStream [

        aStream
            nextPutInt64: anOOP oopInstVarSize;
            nextPutInt64: anOOP oopArray size.

        (1 to: anOOP oopInstVarSize) do: [ :i |
            (anOOP oopInstVarAt: i) isInteger
                ifTrue: [ self writeInt64: (anOOP oopInstVarAt: i) in: aStream ]
                ifFalse: [ aStream nextPutInt64: (visitedItems at: (anOOP oopInstVarAt: i) ifAbsentPut: [ self itemToBeVisited: (anOOP oopInstVarAt: i) ]) ] ].
        anOOP oopDo: [ :each |
```

CorGenCode

```
            each isInteger
                ifTrue: [ self writeInt64: each in: aStream ]
                ifFalse: [ aStream nextPutInt64: (visitedItems at: each ifAbsentPut: [ self itemToBeVisited: each ]) ] ].

    ]

    readOOP: aStream [

        ^ self read: OOP new in: aStream
    ]

    read: anOOP in: aStream [
        <category: 'oop reading'>

        | oopClass instVarSize arraySize |
        oopClass := visitedItems at: aStream nextUint64 ifAbsentPut: [ toSave add: OOP new ].
        instVarSize := aStream nextByte.
        arraySize := aStream nextByte.

        anOOP instVarSize: instVarSize arraySize: arraySize.

        1 to: anOOP oopInstVarSize do: [ :i |
            (self isOOPInteger: aStream)
                ifTrue: [ anOOP instVarAt: i put: (self readInt64: aStream) ]
                ifFalse: [ anOOP instVarAt: i put: (visitedItems at: (self readInt64: aStream) ifAbsentPut: [ toSave add: OOP new ]) ] ].
        1 to: anOOP oopSize do: [ :i |
            (self isOOPInteger: aStream)
                ifTrue: [ anOOP oopAt: i put: (self readInt64: aStream) ]
                ifFalse: [ anOOP oopAt: i put: (visitedItems at: (self readInt64: aStream) ifAbsentPut: [ toSave add: OOP new ]) ] ].
        ^ anOOP
    ]

    isOOPInteger: aStream [
        <category: 'testing'>

        | res |
        aStream position: (aStream position + 7).
        res := (aStream nextByte bitAnd: 1) = 1.
        aStream position: (aStream position - 8).
        ^ res
    ]

    writeInt64: anInteger in: aStream [

        | n |
        n := 8 * 7.
        7 timesRepeat: [
            aStream nextPutByte: ((anInteger bitShift: (1 - n)) bitAnd: 255).
            n := n - 8 ].
        aStream nextPutByte: (((anInteger bitShift: 1) bitAnd: 255) + 1)
    ]

    readInt64: aStream [

        | byte n number |
        n := 7 * 8.
        number := 0.
        7 timesRepeat: [ | byte |
            byte := aStream nextByte.
            number := number + (byte bitShift: n + 1).
            n := n - 8 ].
        byte := aStream nextByte.
        ^ number + (byte bitShift: -1)
    ]

    platform: aGSTPlatform [
        <category: 'accessing'>

        platform := aGSTPlatform
    ]
```

```
    visitedItems [
        <category: 'accessing'>

        ^ visitedItems
    ]
]


"**************************************************************
 bootstrap/GSTia64.st
 **************************************************************"
Object subclass: GSTia64 [

    GSTia64 class >> wordSize [
        <category: 'accessing'>

        ^ 8
    ]
]


"**************************************************************
 bootstrap/OOP.st
 **************************************************************"
" Copyright 2010 GST.
  Written by Gwenael Casaccio

  This file is part of GST.

  GST is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GST is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GST.  If not, see <http://www.gnu.org/licenses/>. "

Object subclass: OOP [
    | NilOOP |

    OOP class >> nilOOP: anOOP [
        NilOOP := anOOP
    ]

    OOP class >> nilOOP [
        ^ NilOOP
    ]

    OOP class >> name [
        <category: 'accessing'>

        ^ self subclassResponsibility
    ]

    OOP class >> model [
        <category: 'accessing'>

        ^ #()
    ]

    OOP class >> generateModel [
        <category: 'accessing'>

        self allSubclasses do: [ :each | | i |
            i := 1.
            each model do: [ :selector |
                each
                    compile: (selector, ' [ ^ self oopInstVarAt: ', i asString,
' ]');
                    compile: (selector, ': anOOP [ self oopInstVarAt: ', i asStr
```

CorGenCode

```
ing, ' put: anOOP ]').
                i := i + 1 ] ]
    ]

    | oopClass oopInstVarSize oopInstVar oopArraySize oopArray |

    asGSTOop: aBootstrap [
        ^ self
    ]

    oopClass: anObject [
        <category: 'accessing'>

        oopClass := anObject
    ]

    oopClass [
        <category: 'accessing'>

        ^ oopClass
    ]

    oopInstVarSize: aByte [
        <category: 'accessing'>

        oopInstVarSize := aByte
    ]

    oopInstVarSize [
        <category: 'accessing'>

        ^ oopInstVarSize
    ]

    oopInstVar [
        <category: 'instance variables accessing'>

        ^ oopInstVar ifNil: [ oopInstVar := Array new: self oopInstVarSize withA
ll: self class nilOOP ]
    ]

    oopInstVarAt: anIndex [
        <category: 'instance variables accessing'>

        ^ self oopInstVar at: anIndex
    ]

    oopInstVarAt: anIndex put: anObject [
        <category: 'instance variables accessing'>

        ^ self oopInstVar at: anIndex put: anObject
    ]

    oopArray [
        <category: 'array accessing'>

        ^ oopArray ifNil: [ oopArray := OrderedCollection new ]
    ]

    oopArray: anInteger [
        <category: 'array accessing'>

        ^ oopArray := Array new: anInteger withAll: self class nilOOP
    ]

    oopAdd: anObject [
        <category: 'array accessing'>

        ^ self oopArray add: anObject
    ]

    oopAt: anIndex [
        <category: 'array accessing'>

        ^ self oopArray at: anIndex
```

CorGenCode

```
    ]

    oopAt: anIndex put: anObject [
        <category: 'array accessing'>

        ^ self oopArray at: anIndex put: anObject
    ]

    oopDo: aOneArgBlock [
        <category: 'enumerating'>

        (1 to: self oopArray size) do: [ :i | aOneArgBlock value: (self oopAt: i
) ]
    ]

    equalTo: aString [

        aString size = self oopArray size ifFalse: [ ^ false ].
        1 to: aString size do: [ :i |
            (aString at: i) value = ((self oopAt: i) oopInstVarAt: 1)  ifFalse:
 [ ^ false ] ].
        ^ true
    ]

    printOOPString [
        <category: 'debugging'>

        self oopDo: [ :each |
            Transcript show: (each oopInstVarAt: 1) asCharacter asString ].
        Transcript show: (Character nl) asString
    ]

    asSTString [

        | s |
        s := String new.
        self oopDo: [ :each |
            s := s, (each oopInstVarAt: 1) asCharacter asString ].
        ^ s
    ]
]

"************************************************************
 bootstrap/STCompiler.st
 ************************************************************"
"======================================================================
|
|   Smalltalk in Smalltalk compiler
|
|
 ======================================================================"

"======================================================================
|
| Copyright 1999,2000,2001,2002,2003,2006,2007,2009 Free Software Foundation, In
c.
| Written by Paolo Bonzini.
|
| This file is part of GNU Smalltalk.
|
| GNU Smalltalk is free software; you can redistribute it and/or modify it
| under the terms of the GNU General Public License as published by the Free
| Software Foundation; either version 2, or (at your option) any later version.
|
| GNU Smalltalk is distributed in the hope that it will be useful, but WITHOUT
| ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
| FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
| details.
|
| You should have received a copy of the GNU General Public License along with
| GNU Smalltalk; see the file COPYING.  If not, write to the Free Software
| Foundation, 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
|
 ======================================================================"
```

CorGenCode

```
RBProgramNodeVisitor subclass: STCompiler [
    | node destClass symTable parser bytecodes depth maxDepth isInsideBlock |

    OneNode := nil.
    TrueNode := nil.
    FalseNode := nil.
    NilNode := nil.
    SuperVariable := nil.
    SelfVariable := nil.
    ThisContextVariable := nil.

    STCompiler class >> initialize [
        <category: 'initialize'>
        OneNode := RBLiteralNode value: 1.
        TrueNode := RBLiteralNode value: true.
        FalseNode := RBLiteralNode value: false.
        NilNode := RBLiteralNode value: nil.
        SelfVariable := RBVariableNode named: 'self'.
        SuperVariable := RBVariableNode named: 'super'.
        ThisContextVariable := RBVariableNode named: 'thisContext'
    ]

    STCompiler class >> evaluate: aSequenceNode parser: aParser [
        <category: 'evaluation'>
        | cm methodNode |
        aSequenceNode addReturn.
        methodNode := (RBMethodNode new)
                    arguments: #();
                    body: aSequenceNode;
                    selector: #Doit;
                    source: nil;
                    yourself.
        cm := self
                    compile: methodNode
                    asMethodOf: UndefinedObject
                    classified: nil
                    parser: aParser
                    environment: Namespace current.
        ^nil perform: cm
    ]

    STCompiler class >> canCompile: code [
        "Answer whether I know how to compile the given code directly, on
         behalf of a Behavior."

        <category: 'compilation'>
        ^(code isKindOf: RBProgramNode) and: [code isMethod]
    ]

    STCompiler class >> compile: methodNode for: aBehavior classified: aString p
arser: aParser [
        <category: 'compilation'>
        ^aBehavior addSelector: methodNode selector
            withMethod: (self
                    compile: methodNode
                    asMethodOf: aBehavior
                    classified: aString
                    parser: aParser)
    ]

    STCompiler class >> compile: methodNode asMethodOf: aBehavior classified: aS
tring parser: aParser [
        <category: 'compilation'>
        ^self
            compile: methodNode
            asMethodOf: aBehavior
            classified: aString
            parser: aParser
            environment: nil
    ]

    STCompiler class >> compile: methodNode asMethodOf: aBehavior classified: aS
tring parser: aParser environment: aNamespace [
        <category: 'compilation'>
```

CorGenCode

```
        | compiler method |
        compiler := self new.
        compiler class: aBehavior parser: aParser.
        aNamespace isNil ifFalse: [compiler addPool: aNamespace].
        method := compiler visitNode: methodNode.
        aString isNil ifFalse: [ method methodCategory: aString ].
        ^method
    ]

    class: aBehavior parser: aParser [
        <category: 'private'>
        destClass := aBehavior.
        symTable := STSymbolTable new.
        parser := aParser.
        bytecodes := WriteStream on: (ByteArray new: 240).
        isInsideBlock := 0.
        symTable declareEnvironment: aBehavior
    ]

    addLiteral: literal [
        <category: 'accessing'>
        ^symTable addLiteral: literal
    ]

    addPool: aNamespace [
        <category: 'accessing'>
        ^symTable addPool: aNamespace
    ]

    bytecodesFor: aBlockNode [
        <category: 'accessing'>
        ^self bytecodesFor: aBlockNode atEndDo: []
    ]

    bytecodesFor: aBlockNode atEndDo: aBlock [
        <category: 'accessing'>
        | saveBytecodes result |
        saveBytecodes := bytecodes.
        bytecodes := WriteStream on: (ByteArray new: 240).
        self declareArgumentsAndTemporaries: aBlockNode.
        self compileStatements: aBlockNode body.
        self undeclareArgumentsAndTemporaries: aBlockNode.
        aBlock value.
        result := bytecodes contents.
        bytecodes := saveBytecodes.
        ^result
    ]

    checkStore: aVariableName [
        <category: 'accessing'>
        (symTable canStore: aVariableName)
            ifFalse: [self compileError: 'cannot store in argument ' , aVariable
Name]
    ]

    compileError: aString [
        <category: 'accessing'>
        parser parserError: aString
    ]

    compileBackJump: displacement [
        <category: 'accessing'>
        | jumpLen |
        jumpLen := displacement + 2.
        jumpLen := displacement + (self sizeOfJump: jumpLen).
        jumpLen := displacement + (self sizeOfJump: jumpLen).
        self compileByte: JumpBack arg: displacement
    ]

    compileJump: displacement if: jmpCondition [
        <category: 'accessing'>
        displacement < 0
            ifTrue:
                ["Should not happen"
```

CorGenCode

```
                ^self error: 'Cannot compile backwards conditional jumps'].
        self depthDecr: 1.
        jmpCondition
            ifFalse: [self compileByte: PopJumpFalse arg: displacement]
            ifTrue: [self compileByte: PopJumpTrue arg: displacement]
    ]

    compileWarning: aString [
        <category: 'accessing'>
        parser parserWarning: aString
    ]

    declareTemporaries: node [
        <category: 'accessing'>
        node temporaries do:
                [:aTemp |
                symTable
                    declareTemporary: aTemp name
                    canStore: true
                    for: self]
    ]

    declareArgumentsAndTemporaries: node [
        <category: 'accessing'>
        node arguments do:
                [:anArg |
                symTable
                    declareTemporary: anArg name
                    canStore: false
                    for: self].
        self declareTemporaries: node body
    ]

    maxDepth [
        <category: 'accessing'>
        ^maxDepth
    ]

    depthDecr: n [
        <category: 'accessing'>
        depth := depth - n
    ]

    depthIncr [
        <category: 'accessing'>
        depth = maxDepth
            ifTrue:
                [depth := depth + 1.
                maxDepth := maxDepth + 1]
            ifFalse: [depth := depth + 1]
    ]

    depthSet: n [
        "n can be an integer, or a previously returned value (in which case the
         exact status at the moment of the previous call is remembered)"

        <category: 'accessing'>
        | oldDepth |
        oldDepth := n -> maxDepth.
        n isInteger
            ifTrue: [depth := maxDepth := n]
            ifFalse:
                [depth := n key.
                maxDepth := n value].
        ^oldDepth
    ]

    literals [
        <category: 'accessing'>
        ^symTable literals
    ]

    lookupName: variable [
        <category: 'accessing'>
        | definition |
```

CorGenCode

```
        definition := symTable lookupName: variable for: self.
        definition isNil
            ifTrue:
                ["Might want to declare this puppy as a local and go on
                 notwithstanding the error"

                self
                    compileError: 'Undefined variable ' , variable printString ,
    ' referenced.'].
        ^definition
    ]

    compileByte: aByte [
        <category: 'accessing'>
        self compileByte: aByte arg: 0
    ]

    compileByte: aByte arg: arg [
        <category: 'accessing'>
        | n |
        n := 0.
        [(arg bitShift: n) > 255] whileTrue: [n := n - 8].
        n to: -8
            by: 8
            do:
                [:shift |
                bytecodes
                    nextPut: ExtByte;
                    nextPut: ((arg bitShift: shift) bitAnd: 255)].
        bytecodes
            nextPut: aByte;
            nextPut: (arg bitAnd: 255)
    ]

    compileByte: aByte arg: arg1 arg: arg2 [
        <category: 'accessing'>
        self compileByte: aByte arg: (arg1 bitShift: 8) + arg2
    ]

    nextPutAll: aByteArray [
        <category: 'accessing'>
        bytecodes nextPutAll: aByteArray
    ]

    isInsideBlock [
        <category: 'accessing'>
        ^isInsideBlock > 0
    ]

    pushLiteral: value [
        <category: 'accessing'>
        | definition |
        (value isInteger and: [value >= 0 and: [value <= 1073741823]])
            ifTrue:
                [self compileByte: PushInteger arg: value.
                ^self].
        definition := self addLiteral: value.
        self compileByte: PushLitConstant arg: definition
    ]

    pushLiteralVariable: value [
        <category: 'accessing'>
        | definition |
        definition := self addLiteral: value.
        self compileByte: PushLitVariable arg: definition
    ]

    sizeOfJump: distance [
        <category: 'accessing'>
        distance < 256 ifTrue: [^2].
        distance < 65536 ifTrue: [^4].
        distance < 16777216 ifTrue: [^6].
        ^8
    ]
```

CorGenCode

```
    displacementsToJumpAround: jumpAroundOfs and: initialCondLen [
        <category: 'accessing'>
        | jumpAroundLen oldJumpAroundLen finalJumpOfs finalJumpLen |
        jumpAroundLen := oldJumpAroundLen := 0.

        [finalJumpOfs := initialCondLen + oldJumpAroundLen + jumpAroundOfs.
        finalJumpLen := self sizeOfJump: finalJumpOfs.
        jumpAroundLen := self sizeOfJump: jumpAroundOfs + finalJumpLen.
        oldJumpAroundLen = jumpAroundLen]
                whileFalse: [oldJumpAroundLen := jumpAroundLen].
        ^finalJumpLen + finalJumpOfs -> (jumpAroundOfs + finalJumpLen)
    ]

    insideNewScopeDo: aBlock [
        <category: 'accessing'>
        | result |
        isInsideBlock := isInsideBlock + 1.
        symTable scopeEnter.
        result := aBlock value.
        symTable scopeLeave.
        isInsideBlock := isInsideBlock - 1.
        ^result
    ]

    bindingOf: anOrderedCollection [
        <category: 'accessing'>
        | binding |
        binding := symTable bindingOf: anOrderedCollection for: self.
        binding isNil
            ifTrue:
                [self
                    compileError: 'Undefined variable binding'
                            , anOrderedCollection asArray printString , 'referen
ced.'].
        ^binding
    ]

    undeclareTemporaries: aNode [
        <category: 'accessing'>
        aNode temporaries do: [:each | symTable undeclareTemporary: each name]
    ]

    undeclareArgumentsAndTemporaries: aNode [
        <category: 'accessing'>
        self undeclareTemporaries: aNode body.
        aNode arguments do: [:each | symTable undeclareTemporary: each name]
    ]

    acceptSequenceNode: node [
        <category: 'visiting RBSequenceNodes'>
        | statements method |
        node addSelfReturn.
        depth := maxDepth := 0.
        self declareTemporaries: node.
        self compileStatements: node.
        self undeclareTemporaries: node.
        symTable finish.
        method := CompiledMethod
                    literals: symTable literals
                    numArgs: 0
                    numTemps: symTable numTemps
                    attributes: #()
                    bytecodes: bytecodes contents
                    depth: maxDepth + symTable numTemps.
        (method descriptor)
            setSourceCode: node source asSourceCode;
            methodClass: UndefinedObject;
            selector: #executeStatements.
        ^method
    ]

    acceptMethodNode: node [
        <category: 'visiting RBMethodNodes'>
        | statements method attributes |
        node body addSelfReturn.
```

CorGenCode

```
        depth := maxDepth := 0.
        self declareArgumentsAndTemporaries: node.
        self compileStatements: node body.
        self undeclareArgumentsAndTemporaries: node.
        symTable finish.
        attributes := self compileMethodAttributes: node primitiveSources.
        method := CompiledMethod
                    literals: symTable literals
                    numArgs: node arguments size
                    numTemps: node body temporaries size
                    attributes: attributes
                    bytecodes: bytecodes contents
                    depth: maxDepth + node body temporaries size + node argument
s size.
        (method descriptor)
            setSourceCode: node source asSourceCode;
            methodClass: symTable environment;
            selector: node selector.
        method attributesDo:
                [:ann |
                | handler error |
                handler := symTable environment pragmaHandlerFor: ann selector.
                handler notNil
                    ifTrue:
                        [error := handler value: method value: ann.
                        error notNil ifTrue: [self compileError: error]]].
        ^method
    ]

    acceptArrayConstructorNode: aNode [
        "STArrayNode is the parse node class for {...} style array constructors.
         It is compiled like a normal inlined block, but with the statements
         preceded by (Array new: <size of the array>) and with each statement
         followed with a <pop into instance variable of new stack top>
         instead of a simple pop."

        <category: 'visiting RBArrayConstructorNodes'>
        self
            depthIncr;
            pushLiteralVariable: (Smalltalk associationAt: #Array);
            depthIncr;
            compileByte: PushInteger arg: aNode body statements size;
            depthDecr: 1;
            compileByte: SendImmediate arg: NewColonSpecial.
        aNode body statements keysAndValuesDo:
                [:index :each |
                each acceptVisitor: self.
                self
                    depthDecr: 1;
                    compileByte: PopStoreIntoArray arg: index - 1]
    ]

    acceptBlockNode: aNode [
        "STBlockNode has a variable that contains a string for each parameter,
         and one that contains a list of statements. Here is how STBlockNodes
         are compiled:

         push BlockClosure or CompiledBlock literal
         make dirty block                    <--- only if pushed CompiledBlock

         Statements are put in a separate CompiledBlock object that is reference
d
         by the BlockClosure that the sequence above pushes or creates.

         compileStatements: creates the bytecodes.  It is this method that is
         called by STCompiler>>bytecodesFor: and STCompiler>>bytecodesFor:append
:"

        <category: 'visiting RBBlockNodes'>
        | bc depth block clean |
        depth := self depthSet: aNode arguments size + aNode body temporaries si
ze.
        aNode body statements isEmpty
            ifTrue: [aNode body addNode: (RBLiteralNode value: nil)].
        bc := self insideNewScopeDo:
```

CorGenCode

```
                    [self bytecodesFor: aNode
                        atEndDo:
                            [aNode body lastIsReturn ifFalse: [self compileB
yte: ReturnContextStackTop]]].
        block := CompiledBlock
                    numArgs: aNode arguments size
                    numTemps: aNode body temporaries size
                    bytecodes: bc
                    depth: self maxDepth
                    literals: self literals.
        self depthSet: depth.
        clean := block flags.
        clean == 0
            ifTrue:
                [self
                    pushLiteral: (BlockClosure block: block receiver: symTable e
nvironment).
                ^aNode].
        self pushLiteral: block.
        self compileByte: MakeDirtyBlock
    ]

    compileStatements: aNode [
        <category: 'visiting RBBlockNodes'>
        aNode statements keysAndValuesDo:
                [:index :each |
                index = 1
                    ifFalse:
                        [self
                            depthDecr: 1;
                            compileByte: PopStackTop].
                each acceptVisitor: self].
        aNode statements isEmpty
            ifTrue:
                [self
                    depthIncr;
                    compileByte: PushSpecial arg: NilIndex]
    ]

    acceptCascadeNode: aNode [
        "RBCascadeNode holds a collection with one item per message."

        <category: 'visiting RBCascadeNodes'>
        | messages first |
        messages := aNode messages.
        first := messages at: 1.
        first receiver = SuperVariable
            ifTrue:
                [aNode messages do: [:each | self compileSendToSuper: each]
                    separatedBy:
                        [self
                            depthDecr: 1;
                            compileByte: PopStackTop].
                ^aNode].
        first receiver acceptVisitor: self.
        self
            depthIncr;
            compileByte: DupStackTop.
        self compileMessage: first.
        messages
            from: 2
            to: messages size - 1
            do:
                [:each |
                self
                    compileByte: PopStackTop;
                    compileByte: DupStackTop.
                self compileMessage: each].
        self
            depthDecr: 1;
            compileByte: PopStackTop.
        self compileMessage: messages last
    ]

    acceptOptimizedNode: aNode [
```

CorGenCode

```
        <category: 'visiting RBOptimizedNodes'>
        self depthIncr.
        self pushLiteral: (self class evaluate: aNode body parser: parser)
    ]

    acceptLiteralNode: aNode [
        "STLiteralNode has one instance variable, the token for the literal
         it represents."

        <category: 'visiting RBLiteralNodes'>
        self depthIncr.
        aNode compiler: self.
        self pushLiteral: aNode value
    ]

    acceptAssignmentNode: aNode [
        "First compile the assigned, then the assignment to the assignee..."

        <category: 'visiting RBAssignmentNodes'>
        aNode value acceptVisitor: self.
        (VMSpecialIdentifiers includesKey: aNode variable name)
            ifTrue: [self compileError: 'cannot assign to ' , aNode variable nam
e].
        self compileAssignmentFor: aNode variable
    ]

    acceptMessageNode: aNode [
        "RBMessageNode contains a message send. Its instance variable are
         a receiver, selector, and arguments."

        <category: 'compiling'>
        | specialSelector |
        aNode receiver = SuperVariable
            ifTrue:
                [self compileSendToSuper: aNode.
                ^true].
        specialSelector := VMSpecialMethods at: aNode selector ifAbsent: [nil].
        specialSelector isNil
            ifFalse: [(self perform: specialSelector with: aNode) ifTrue: [^fals
e]].
        aNode receiver acceptVisitor: self.
        self compileMessage: aNode
    ]

    compileMessage: aNode [
        "RBMessageNode contains a message send. Its instance variable are
         a receiver, selector, and arguments.  The receiver has already
         been compiled."

        <category: 'compiling'>
        | args litIndex |
        aNode arguments do: [:each | each acceptVisitor: self].
        VMSpecialSelectors at: aNode selector
            ifPresent:
                [:idx |
                idx <= LastImmediateSend
                    ifTrue: [self compileByte: idx arg: 0]
                    ifFalse: [self compileByte: SendImmediate arg: idx].
                ^aNode].
        args := aNode arguments size.
        litIndex := self addLiteral: aNode selector.
        self
            compileByte: Send
            arg: litIndex
            arg: args
    ]

    compileWhileLoop: aNode [
        "Answer whether the while loop can be optimized (that is,
         whether the only parameter is a STBlockNode)"

        <category: 'compiling'>
        | whileBytecodes argBytecodes jumpOffsets |
        aNode receiver isBlock ifFalse: [^false].
        (aNode receiver arguments isEmpty
```

```
            and: [aNode receiver body temporaries isEmpty]) ifFalse: [^false].
        argBytecodes := #().
        aNode arguments do:
                [:onlyArgument |
                onlyArgument isBlock ifFalse: [^false].
                (onlyArgument arguments isEmpty
                    and: [onlyArgument body temporaries isEmpty]) ifFalse: [^fal
se].
                argBytecodes := self bytecodesFor: onlyArgument
                            atEndDo:
                                [self
                                    compileByte: PopStackTop;
                                    depthDecr: 1]].
        whileBytecodes := self bytecodesFor: aNode receiver.
        self nextPutAll: whileBytecodes.
        aNode selector == #repeat
            ifFalse:
                [jumpOffsets := self displacementsToJumpAround: argBytecodes siz
e
                            and: whileBytecodes size + 2.       "for jump around
 jump"

                "The if: clause means: if selector is whileFalse:, compile
                 a 'pop/jump if true'; else compile a 'pop/jump if false'"
                self compileJump: (self sizeOfJump: jumpOffsets value)
                    if: (aNode selector == #whileTrue or: [aNode selector == #wh
ileTrue:]).
                self compileByte: Jump arg: jumpOffsets value.
                argBytecodes isNil ifFalse: [self nextPutAll: argBytecodes].
                self compileByte: JumpBack arg: jumpOffsets key]
            ifTrue: [self compileBackJump: whileBytecodes size].

        "Somebody might want to use the return value of #whileTrue:
         and #whileFalse:"
        self
            depthIncr;
            compileByte: PushSpecial arg: NilIndex.
        ^true
    ]

    compileSendToSuper: aNode [
        <category: 'compiling'>
        | litIndex args |
        self
            depthIncr;
            compileByte: PushSelf.
        aNode arguments do: [:each | each acceptVisitor: self].
        self pushLiteral: destClass superclass.
        VMSpecialSelectors at: aNode selector
            ifPresent:
                [:idx |
                self compileByte: SendImmediateSuper arg: idx.
                ^aNode].
        litIndex := self addLiteral: aNode selector.
        args := aNode arguments size.
        self
            compileByte: SendSuper
            arg: litIndex
            arg: args.
        self depthDecr: aNode arguments size
    ]

    compileTimesRepeat: aNode [
        <category: 'compiling'>
        "aNode receiver acceptVisitor: self."

        | block |
        block := aNode arguments first.
        (block arguments isEmpty and: [block body temporaries isEmpty])
            ifFalse: [^false].
        ^false
    ]

    compileLoop: aNode [
        <category: 'compiling'>
```

CorGenCode

```
        "aNode receiver acceptVisitor: self."

        | stop step block |
        aNode arguments do:
                [:each |
                stop := step.   "to:"
                step := block.  "by:"
                block := each   "do:"].
        (block arguments size = 1 and: [block body temporaries isEmpty])
            ifFalse: [^false].
        stop isNil
            ifTrue:
                [stop := step.
                step := OneNode "#to:do:"]
            ifFalse: [step isImmediate ifFalse: [^false]].
        ^false
    ]

    compileBoolean: aNode [
        <category: 'compiling'>
        | bc1 ret1 bc2 selector |
        aNode arguments do:
                [:each |
                (each arguments isEmpty and: [each body temporaries isEmpty])
                    ifFalse: [^false].
                bc1 isNil
                    ifTrue:
                        [bc1 := self bytecodesFor: each.
                        ret1 := each isReturn]
                    ifFalse: [bc2 := self bytecodesFor: each]].
        aNode receiver acceptVisitor: self.
        selector := aNode selector.
        bc2 isNil
            ifTrue:
                ["Transform everything into #ifTrue:ifFalse: or #ifFalse:ifTrue:
"

                selector == #ifTrue:
                    ifTrue:
                        [selector := #ifTrue:ifFalse:.
                        bc2 := NilIndex "Push nil"].
                selector == #ifFalse:
                    ifTrue:
                        [selector := #ifFalse:ifTrue:.
                        bc2 := NilIndex "Push nil"].
                selector == #and:
                    ifTrue:
                        [selector := #ifTrue:ifFalse:.
                        bc2 := FalseIndex       "Push false"].
                selector == #or:
                    ifTrue:
                        [selector := #ifFalse:ifTrue:.
                        bc2 := TrueIndex        "Push true"].
                bc2 :=
                        {PushSpecial.
                        bc2}.
                ^self
                    compileBoolean: aNode
                    longBranch: bc1
                    returns: ret1
                    shortBranch: bc2
                    longIfTrue: selector == #ifTrue:ifFalse:].
        selector == #ifTrue:ifFalse:
            ifTrue:
                [^self
                    compileIfTrue: bc1
                    returns: ret1
                    ifFalse: bc2].
        selector == #ifFalse:ifTrue:
            ifTrue:
                [^self
                    compileIfFalse: bc1
                    returns: ret1
                    ifTrue: bc2].
        ^self error: 'bad boolean message selector'
```

CorGenCode

```
    ]

    compileBoolean: aNode longBranch: bc1 returns: ret1 shortBranch: bc2 longIfT
rue: longIfTrue [
        <category: 'compiling'>
        self compileJump: bc1 size + (ret1 ifTrue: [0] ifFalse: [2])
            if: longIfTrue not.
        self nextPutAll: bc1.
        ret1 ifFalse: [self compileByte: Jump arg: bc2 size].
        self nextPutAll: bc2.
        ^true
    ]

    compileIfTrue: bcTrue returns: bcTrueReturns ifFalse: bcFalse [
        <category: 'compiling'>
        | trueSize |
        trueSize := bcTrueReturns
                    ifTrue: [bcTrue size]
                    ifFalse: [bcTrue size + (self sizeOfJump: bcFalse size)].
        self compileJump: trueSize if: false.
        self nextPutAll: bcTrue.
        bcTrueReturns ifFalse: [self compileByte: Jump arg: bcFalse size].
        self nextPutAll: bcFalse.
        ^true
    ]

    compileIfFalse: bcFalse returns: bcFalseReturns ifTrue: bcTrue [
        <category: 'compiling'>
        | falseSize |
        falseSize := bcFalseReturns
                    ifTrue: [bcFalse size]
                    ifFalse: [bcFalse size + (self sizeOfJump: bcTrue size)].
        self compileJump: falseSize if: true.
        self nextPutAll: bcFalse.
        bcFalseReturns ifFalse: [self compileByte: Jump arg: bcTrue size].
        self nextPutAll: bcTrue.
        ^true
    ]

    acceptReturnNode: aNode [
        <category: 'compiling'>
        aNode value acceptVisitor: self.
        self isInsideBlock
            ifTrue: [self compileByte: ReturnMethodStackTop]
            ifFalse: [self compileByte: ReturnContextStackTop]
    ]

    compileAssignmentFor: aNode [
        "RBVariableNode has one instance variable, the name of the variable
         that it represents."

        <category: 'visiting RBVariableNodes'>
        | definition |
        self checkStore: aNode name.
        definition := self lookupName: aNode name.
        (symTable isTemporary: aNode name)
            ifTrue:
                [^self compileStoreTemporary: definition
                    scopes: (symTable outerScopes: aNode name)].
        (symTable isReceiver: aNode name)
            ifTrue: [^self compileByte: StoreReceiverVariable arg: definition].
        self compileByte: StoreLitVariable arg: definition.
        self compileByte: PopStackTop.
        self compileByte: PushLitVariable arg: definition
    ]

    acceptVariableNode: aNode [
        <category: 'visiting RBVariableNodes'>
        | locationType definition |
        self depthIncr.
        VMSpecialIdentifiers at: aNode name
            ifPresent:
                [:block |
                block value: self.
                ^aNode].
```

CorGenCode

```
        definition := self lookupName: aNode name.
        (symTable isTemporary: aNode name)
            ifTrue:
                [^self compilePushTemporary: definition
                    scopes: (symTable outerScopes: aNode name)].
        (symTable isReceiver: aNode name)
            ifTrue:
                [self compileByte: PushReceiverVariable arg: definition.
                ^aNode].
        self compileByte: PushLitVariable arg: definition
    ]

    compilePushTemporary: number scopes: outerScopes [
        <category: 'visiting RBVariableNodes'>
        outerScopes = 0
            ifFalse:
                [self
                    compileByte: PushOuterVariable
                    arg: number
                    arg: outerScopes.
                ^self].
        self compileByte: PushTemporaryVariable arg: number
    ]

    compileStoreTemporary: number scopes: outerScopes [
        <category: 'visiting RBVariableNodes'>
        outerScopes = 0
            ifFalse:
                [self
                    compileByte: StoreOuterVariable
                    arg: number
                    arg: outerScopes.
                ^self].
        self compileByte: StoreTemporaryVariable arg: number
    ]

    compileMethodAttributes: attributes [
        <category: 'compiling method attributes'>
        ^attributes asArray
            collect: [:each | self compileAttribute: (RBScanner on: each readStr
eam)]
    ]

    scanTokenFrom: scanner [
        <category: 'compiling method attributes'>
        scanner atEnd
            ifTrue: [^self compileError: 'method attributes must end with ''>'''
].
        ^scanner next
    ]

    compileAttribute: scanner [
        <category: 'compiling method attributes'>
        | currentToken selectorBuilder selector arguments argParser node |
        currentToken := self scanTokenFrom: scanner.
        (currentToken isBinary and: [currentToken value == #<])
            ifFalse: [^self compileError: 'method attributes must begin with ''<
'''].
        selectorBuilder := WriteStream on: String new.
        arguments := WriteStream on: Array new.
        currentToken := self scanTokenFrom: scanner.
        [currentToken isBinary and: [currentToken value == #>]] whileFalse:
                [currentToken isKeyword
                    ifFalse: [^self compileError: 'keyword expected in method at
tribute'].
                selectorBuilder nextPutAll: currentToken value.
                argParser := RBParser new.
                argParser errorBlock: parser errorBlock.
                argParser scanner: scanner.
                node := argParser parseBinaryMessageNoGreater.
                node := RBSequenceNode statements: {node}.
                arguments nextPut: (self class evaluate: node parser: argParser)
.
                currentToken := argParser currentToken].
        selector := selectorBuilder contents asSymbol.
```

```
            ^Message selector: selector arguments: arguments contents
        ]
    ]
```

CorGenCode

```
Eval [
    STCompiler initialize
]

"************************************************************
 bootstrap/oopModel/ArrayOOP.st
 ************************************************************"
OOP subclass: ArrayOOP [

    ArrayOOP class >> name [
        <category: 'accessing'>

        ^ #Array
    ]
]


"************************************************************
 bootstrap/oopModel/BehaviorOOP.st
 ************************************************************"
OOP subclass: BehaviorOOP [

    BehaviorOOP class >> name [
        <category: 'accessing'>

        ^ #Behavior
    ]

    BehaviorOOP class >> model [
        <category: 'accessing'>

        ^ super model, #(#superClass #methodDictionary #instanceSpec #subClasses
 #instanceVariables)
    ]
]


"************************************************************
 bootstrap/oopModel/CharacterOOP.st
 ************************************************************"
OOP subclass: CharacterOOP [

    CharacterOOP class >> name [
        <category: 'accessing'>

        ^ #Character
    ]

    CharacterOOP class >> model [
        <category: 'accessing'>

        ^ super model, #(#value)
    ]
]

"************************************************************
 bootstrap/oopModel/ClassOOP.st
 ************************************************************"
BehaviorOOP subclass: ClassOOP [

    ClassOOP class >> name [
        <category: 'accessing'>

        ^ #Class
    ]

    ClassOOP class >> model [
        <category: 'accessing'>

        ^ super model, #(#name #comment #category #environment #classVariables #
sharedPools #pragmaHandlers)
    ]

    | parsedClass |
```

```
    parsedClass: aClass [
        <category: 'accessing'>

        parsedClass := aClass
    ]

    parsedClass [
        <category: 'accessing'>

        ^ parsedClass
    ]
]


"************************************************************
 bootstrap/oopModel/CompiledBlockOOP.st
 ************************************************************"
CompiledCodeOOP subclass: CompiledBlockOOP [

    CompiledBlockOOP class >> name [
        <category: 'accessing'>

        ^ #CompiledBlock
    ]

    CompiledBlockOOP class >> model [
        <category: 'accessing'>

        ^ super model, #(#method)
    ]
]


"************************************************************
 bootstrap/oopModel/CompiledCodeOOP.st
 ************************************************************"
OOP subclass: CompiledCodeOOP [

    CompiledCodeOOP class >> name [
        <category: 'accessing'>

        ^ #CompiledCode
    ]

    CompiledCodeOOP class >> model [
        <category: 'accessing'>

        ^ super model, #(#literals #stackDepth #numTemporaries #numArgs)
    ]
]


"************************************************************
 bootstrap/oopModel/CompiledMethodOOP.st
 ************************************************************"
CompiledCodeOOP subclass: CompiledMethodOOP [

    CompiledMethodOOP class >> name [
        <category: 'accessing'>

        ^ #CompiledMethod
    ]

    CompiledMethodOOP class >> model [
        <category: 'accessing'>

        ^ super model, #(#primitive #methodInfo)
    ]
]


"************************************************************
 bootstrap/oopModel/MetaclassOOP.st
 ************************************************************"
```

CorGenCode

```
BehaviorOOP subclass: MetaclassOOP [

    MetaclassOOP class >> name [
        <category: 'accessing'>

        ^ #Metaclass
    ]

    MetaclassOOP class >> model [
        <category: 'accessing'>

        ^ super model, #(#instanceClass)
    ]
]


"************************************************************
 bootstrap/oopModel/MethodContextOOP.st
 ************************************************************"
OOP subclass: MethodContextOOP [

    MethodContextOOP class >> name [
        <category: 'accessing'>

        ^ #MethodContext
    ]

    MethodContextOOP class >> model [
        <category: 'accessing'>

        ^ super model, #(#parent #ip #sp #receiver #method #flags)
    ]
]


"************************************************************
 bootstrap/oopModel/MethodInfoOOP.st
 ************************************************************"
OOP subclass: MethodInfoOOP [

    MethodInfoOOP class >> name [
        <category: 'accessing'>

        ^ #MethodInfo
    ]

    MethodInfoOOP class >> model [
        <category: 'accessing'>

        ^ super model, #(#sourceCode #category #classOOP #selector)
    ]
]


"************************************************************
 bootstrap/oopModel/ProcessOOP.st
 ************************************************************"
OOP subclass: ProcessOOP [

    ProcessOOP class >> name [
        <category: 'accessing'>

        ^ #Process
    ]

    ProcessOOP class >> model [
        <category: 'accessing'>

        ^ super model, #(#nextLink #suspendedContext #priority #myList #name #in
terrupts #interruptLock)
    ]
]


"************************************************************
```

```
 bootstrap/oopModel/ProcessorSchedulerOOP.st
 ************************************************************"
OOP subclass: ProcessorSchedulerOOP [

    ProcessorSchedulerOOP class >> name [
        <category: 'accessing'>

        ^ #ProcessorScheduler
    ]

    ProcessorSchedulerOOP class >> model [
        <category: 'accessing'>

        ^ super model, #(#scheduler #processes #activeProcess #idleTasks)
    ]
]


"************************************************************
 bootstrap/oopModel/SystemDictionaryOOP.st
 ************************************************************"
OOP subclass: SystemDictionaryOOP [

    SystemDictionaryOOP class >> name [
        <category: 'accessing'>

        ^ #SystemDictionary
    ]

    SystemDictionaryOOP class >> model [
        <category: 'accessing'>

        ^ super model, #(#array #size)
    ]
]


"************************************************************
 bootstrap/oopModel/UndefinedObjectOOP.st
 ************************************************************"
OOP subclass: UndefinedObjectOOP [

    UndefinedObjectOOP class >> name [
        <category: 'accessing'>

        ^ #UndefinedObject
    ]
]


"************************************************************
 bootstrap/oopModel/VariableBindingOOP.st
 ************************************************************"
OOP subclass: VariableBindingOOP [

    VariableBindingOOP class >> name [
        <category: 'accessing'>

        ^ #VariableBinding
    ]

    VariableBindingOOP class >> model [
        <category: 'accessing'>

        ^ super model, #(#key #value #environment)
    ]
]


"************************************************************
 kernel/ActiveVariable.st
 ************************************************************"
Object subclass: ActiveVariable [
    | behavior name offset afterChanges beforeChanges |
    behavior: aBehavior [
        <category: 'accessing'>

        behavior := aBehavior
    ]

    named: aString [
        <category: 'accessing'>

        name := aString
    ]

    offset: anInteger [
        <category: 'accessing'>

        offset := anInteger
    ]

    valueFor: anObject [
        <category: 'accessing'>

        ^ anObject instVarAt: offset
    ]

]


"************************************************************
 kernel/Behavior.st
 ************************************************************"
Object subclass: Behavior [
    | superClass methodDictionary instanceSpec subClasses instanceVariables |

    <category: 'Language-Implementation'>

    new [
        <primitive: VMPrimitiveBehaviorNew>
    ]

    new: anInteger [
        <primitive: VMPrimitiveBehaviorNewColon>
    ]
]


"************************************************************
 kernel/BlockClosure.st
 ************************************************************"
Object subclass: BlockClosure [
    | outerContext block receiver |

    value [
        <primitive: VMPrimitiveValue>
    ]
]


"************************************************************
 kernel/BlockContext.st
 ************************************************************"
ContextPart subclass: BlockContext [
    | outerContext |
]


"************************************************************
 kernel/Boolean.st
 ************************************************************"
Object subclass: Boolean [
]


"************************************************************
 kernel/Bootstrap.st
 ************************************************************"
```

CorGenCode

```
Object subclass: Bootstrap [
    | a |

    Bootstrap class >> initialize [
        <category: 'initialization'>

        | obj obj2 |
        self primitivePrint: self displayVersion.
        obj := Array new: 10.
        obj at: 1 put: 'hello'.
        self primitivePrint: (obj at: 1).
        1 < 10
            ifTrue: [ self primitivePrint: 'GOOD' ]
            ifFalse: [ self primitivePrint: 'BAD' ].
        1 > 10
            ifTrue: [ self primitivePrint: 'BAD' ]
            ifFalse: [ self primitivePrint: 'GOOD' ].
        obj2 := Array new: 10.
        obj == obj2 ifFalse: [ self primitivePrint: 'GOOD' ].
        obj changeClassTo: Array.
        obj class == Array ifTrue: [ self primitivePrint: 'GOOD' ].
        self primitivePrint: (obj class instVarAt: 6).
        obj := self new.
        obj instVarAt: 1 put: 'Plouf'.
        self primitivePrint: (obj instVarAt: 1).
        [ self primitivePrint: 'hello' ] value
    ]

    Bootstrap class >> displayVersion [
        <category: 'initialization'>

        ^ 'GST 0.1.0'
    ]

    Bootstrap class >> doesNotUnderstand: aMessage[
        <category: 'initialization'>


    ]

    Bootstrap class >> primitivePrint: aString [
        <primitive: VMPrimitivePrint>
    ]
]


"************************************************************
 kernel/Class.st
 ************************************************************"
ClassDescription subclass: Class [
    | name comment category environment classVariables sharedPools pragmaHandler
s |

    <category: 'Language-Implementation'>
    <comment: 'I am THE class object.  My instances are the classes of the syste
m.
I provide information commonly attributed to classes: namely, the
class name, class comment (you wouldn''t be reading this if it
weren''t for me), a list of the instance variables of the class, and
the class category.'>

    Class class >> initialize [
    "Perform the special initialization of root classes."

    <category: 'initialize'>
    ]

    name [
    "Answer the class name"

    <category: 'accessing instances and variables'>
    ^name
    ]

    comment [
```

CorGenCode

```
    "Answer the class comment"

    <category: 'accessing instances and variables'>
    ^comment
    ]

    comment: aString [
    "Change the class name"

    <category: 'accessing instances and variables'>
    comment := aString
    ]

    environment [
    <category: 'accessing instances and variables'>
    ^environment
    ]

    environment: aNamespace [
    "Set the receiver's environment to aNamespace and recompile everything"

    <category: 'accessing instances and variables'>
    ]

    category [
    "Answer the class category"

    <category: 'accessing instances and variables'>
    ^category
    ]

    category: aString [
    "Change the class category to aString"

    <category: 'accessing instances and variables'>
    category := aString
    ]

    superclass: aClass [
    "Set the receiver's superclass."
    <category: 'accessing instances and variables'>
    ]

    addClassVarName: aString [
    "Add a class variable with the given name to the class pool dictionary."

    <category: 'accessing instances and variables'>
    | sym |
    ]

    addClassVarName: aString value: valueBlock [
    "Add a class variable with the given name to the class pool dictionary,
     and evaluate valueBlock as its initializer."

    <category: 'accessing instances and variables'>
    ]

    bindingFor: aString [
    "Answer the variable binding for the class variable with the
     given name"

    <category: 'accessing instances and variables'>
    | sym |
    ]

    removeClassVarName: aString [
    "Removes the class variable from the class, error if not present, or
     still in use."

    <category: 'accessing instances and variables'>
    ]

    classPool [
    "Answer the class pool dictionary"
```

CorGenCode

```
    <category: 'accessing instances and variables'>
    ]

    classVarNames [
    "Answer the names of the variables in the class pool dictionary"

    <category: 'accessing instances and variables'>
    ]

    allClassVarNames [
    "Answer the names of the variables in the receiver's class pool dictionary
     and in each of the superclasses' class pool dictionaries"

    <category: 'accessing instances and variables'>
    ]

    addSharedPool: aDictionary [
    "Add the given shared pool to the list of the class' pool dictionaries"

    <category: 'accessing instances and variables'>
    ]

    removeSharedPool: aDictionary [
    "Remove the given dictionary to the list of the class' pool dictionaries"

    <category: 'accessing instances and variables'>
    ]

    sharedPools [
    "Return the names of the shared pools defined by the class"

    <category: 'accessing instances and variables'>
    ]

    classPragmas [
    "Return the pragmas that are written in the file-out of this class."

    <category: 'accessing instances and variables'>
    ]

    initializeAsRootClass [
    "Perform special initialization reserved to root classes."

    <category: 'accessing instances and variables'>
    ]

    initialize [
    "redefined in children (?)"

    <category: 'accessing instances and variables'>
    ^self
    ]

    = aClass [
    "Returns true if the two class objects are to be considered equal."

    "^(aClass isKindOf: Class) and: [name = aClass name]"

    <category: 'testing'>
    ^self == aClass
    ]

    categoriesFor: method are: categories [
    "Don't use this, it is only present to file in from IBM Smalltalk"

    <category: 'instance creation - alternative'>
    ]

    inheritShape [
    "Answer whether subclasses will have by default the same shape as
     this class.  The default is false."
    <category: 'instance creation'>
    ]

    subclass: classNameString [
```

CorGenCode

```
    "Define a subclass of the receiver with the given name.  If the class
     is already defined, don't modify its instance or class variables
     but still, if necessary, recompile everything needed."

    <category: 'instance creation'>
    ]

    subclass: classNameString instanceVariableNames: stringInstVarNames classVar
iableNames: stringOfClassVarNames poolDictionaries: stringOfPoolNames category:
categoryNameString [
    "Define a fixed subclass of the receiver with the given name, instance
     variables, class variables, pool dictionaries and category. If the
     class is already defined, if necessary, recompile everything needed."

    <category: 'instance creation'>
    ]

    variableSubclass: classNameString instanceVariableNames: stringInstVarNames
classVariableNames: stringOfClassVarNames poolDictionaries: stringOfPoolNames ca
tegory: categoryNameString [
    "Define a variable pointer subclass of the receiver with the given
     name, instance variables, class variables, pool dictionaries and
     category. If the class is already defined, if necessary, recompile
     everything needed."

    <category: 'instance creation'>
    ]

    variable: shape subclass: classNameString instanceVariableNames: stringInstV
arNames classVariableNames: stringOfClassVarNames poolDictionaries: stringOfPool
Names category: categoryNameString [
    "Define a variable subclass of the receiver with the given name,
     shape, instance variables, class variables, pool dictionaries and
     category. If the class is already defined, if necessary, recompile
     everything needed.  The shape can be one of #byte #int8 #character
     #short #ushort #int #uint #int64 #uint64 #utf32 #float #double or
     #pointer."

    <category: 'instance creation'>
    ]

    variableWordSubclass: classNameString instanceVariableNames: stringInstVarNa
mes classVariableNames: stringOfClassVarNames poolDictionaries: stringOfPoolName
s category: categoryNameString [
    "Define a word variable subclass of the receiver with the given
     name, instance variables (must be ''), class variables, pool
     dictionaries and category. If the class is already defined, if
     necessary, recompile everything needed."

    <category: 'instance creation'>
    ]

    variableByteSubclass: classNameString instanceVariableNames: stringInstVarNa
mes classVariableNames: stringOfClassVarNames poolDictionaries: stringOfPoolName
s category: categoryNameString [
    "Define a byte variable subclass of the receiver with the given
     name, instance variables (must be ''), class variables, pool
     dictionaries and category. If the class is already defined, if
     necessary, recompile everything needed."

    <category: 'instance creation'>
    ]

    article [
    "Answer an article ('a' or 'an') which is ok for the receiver's name"

    <category: 'printing'>
    ]

    printOn: aStream [
    "Print a representation of the receiver on aStream"

    <category: 'printing'>
    ]
```

```
    storeOn: aStream [
    "Store Smalltalk code compiling to the receiver on aStream"

    <category: 'printing'>
    ]

    registerHandler: aBlock forPragma: pragma [
    "While compiling methods, on every encounter of the pragma
     with the given name, call aBlock with the CompiledMethod and
     an array of pragma argument values."
    <category: 'pragmas'>
    ]

    pragmaHandlerFor: aSymbol [
    "Answer the (possibly inherited) registered handler for pragma
     aSymbol, or nil if not found."
    <category: 'pragmas'>
    ]

    classInstanceVariableNames: stringClassInstVarNames [
    <category: 'private'>
    ]

    sharedPoolDictionaries [
    "Return the shared pools (not the names!) defined by the class"

    <category: 'private'>
    ]

    allSharedPoolDictionariesDo: aBlock [
    ]

    allLocalSharedPoolDictionariesExcept: inWhite do: aBlock [
    ]

    metaclassFor: classNameString [
    ]

    asClass [
    <category: 'testing functionality'>
    ^self
    ]

    isClass [
    <category: 'testing functionality'>
    ]

]

"************************************************************
 kernel/ClassDescription.st
 ************************************************************"
Behavior subclass: ClassDescription [

    printOn: aStream in: aNamespace [
    "Print on aStream the class name when the class is referenced from
     aNamespace"

    <category: 'printing'>
    ]

    classVariableString [
    <category: 'printing'>
    ]

    instanceVariableString [
    "Answer a string containing the name of the receiver's instance variables."

    <category: 'printing'>
    ]

    sharedVariableString [
    <category: 'printing'>
    ]
```

CorGenCode

```
    binding [
    "Answer a VariableBinding object whose value is the receiver"

    <category: 'conversion'>
    ]

    asClass [
    <category: 'conversion'>
    ]

    asMetaclass [
    "Answer the metaclass associated to the receiver"

    <category: 'conversion'>
    ]

    addSharedPool: aDictionary [
    "Add the given shared pool to the list of the class' pool dictionaries"

    <category: 'parsing class declarations'>
    self subclassResponsibility
    ]

]


"************************************************************
 kernel/ContextPart.st
 ************************************************************"
Object subclass: ContextPart [
     | parent ip sp receiver method |
]


"************************************************************
 kernel/False.st
 ************************************************************"
Boolean subclass: False [
]


"************************************************************
 kernel/InlineCachingEntry.st
 ************************************************************"
Object subclass: InlineCachingEntry [
    | class cmpMethod counter |
]


"************************************************************
 kernel/Metaclass.st
 ************************************************************"
ClassDescription subclass: Metaclass [
     | instanceClass |

    <category: 'Language-Implementation'>
    <comment: 'I am the root of the class hierarchy.  My instances are metaclass
es, one for
each real class.  My instances have a single instance, which they hold
onto, which is the class that they are the metaclass of.  I provide methods
for creation of actual class objects from metaclass object, and the creation
of metaclass objects, which are my instances.  If this is confusing to you,
it should be...the Smalltalk metaclass system is strange and complex.'>

    addClassVarName: aString [
    "Add a class variable with the given name to the class pool dictionary"

    <category: 'delegation'>
    ^self instanceClass addClassVarName: aString
    ]

    removeClassVarName: aString [
    "Removes the class variable from the class, error if not present, or
     still in use."
```

CorGenCode

```

    <category: 'delegation'>
    ^self instanceClass removeClassVarName: aString
    ]

    name [
    "Answer the class name - it has none, actually"

    <category: 'delegation'>
    ^nil
    ]

    allSharedPoolDictionariesDo: aBlock [
    "Answer the shared pools visible from methods in the metaclass,
     in the correct search order."

    <category: 'delegation'>
    self asClass allSharedPoolDictionariesDo: aBlock
    ]

    category [
    "Answer the class category"

    <category: 'delegation'>
    ^self asClass category
    ]

    comment [
    "Answer the class comment"

    <category: 'delegation'>
    ^self asClass comment
    ]

    environment [
    "Answer the namespace in which the receiver is implemented"

    <category: 'delegation'>
    ^self asClass environment
    ]

    classPool [
    "Answer the class pool dictionary"

    <category: 'delegation'>
    ^self instanceClass classPool
    ]

    classVarNames [
    "Answer the names of the variables in the class pool dictionary"

    <category: 'delegation'>
    ^self instanceClass classVarNames
    ]

    allClassVarNames [
    "Answer the names of the variables in the receiver's class pool dictionary
     and in each of the superclasses' class pool dictionaries"

    <category: 'delegation'>
    ^self instanceClass allClassVarNames
    ]

    addSharedPool: aDictionary [
    "Add the given shared pool to the list of the class' pool dictionaries"

    <category: 'delegation'>
    ^self instanceClass addSharedPool: aDictionary
    ]

    removeSharedPool: aDictionary [
    "Remove the given dictionary to the list of the class' pool dictionaries"

    <category: 'delegation'>
    ^self instanceClass removeSharedPool: aDictionary
```

CorGenCode

```
    ]

    sharedPools [
    "Return the names of the shared pools defined by the class"

    <category: 'delegation'>
    ^self instanceClass sharedPools
    ]

    allSharedPools [
    "Return the names of the shared pools defined by the class and any of
     its superclasses"

    <category: 'delegation'>
    ^self instanceClass allSharedPools
    ]

    pragmaHandlerFor: aSymbol [
    "Answer the (possibly inherited) registered handler for pragma
     aSymbol, or nil if not found."
    <category: 'delegation'>
    ^self instanceClass pragmaHandlerFor: aSymbol
    ]

    instanceClass [
    "Answer the only instance of the metaclass"

    <category: 'accessing'>
    ^instanceClass
    ]

    printOn: aStream in: aNamespace [
    "Print on aStream the class name when the class is referenced from
     aNamespace."

    <category: 'printing'>
    ]

    printOn: aStream [
    "Print a represention of the receiver on aStream"

    <category: 'printing'>
    ]

    storeOn: aStream [
    "Store Smalltalk code compiling to the receiver on aStream"

    <category: 'printing'>
    ]

    asClass [
    <category: 'testing functionality'>
    ^instanceClass
    ]

    isMetaclass [
    <category: 'testing functionality'>
    ^true
    ]

]


"************************************************************
 kernel/MethodContext.st
 ************************************************************"
ContextPart subclass: MethodContext [
    | flags |
]


"************************************************************
 kernel/MethodInfo.st
 ************************************************************"
Object subclass: MethodInfo [
```

CorGenCode

```
    | sourceCode category class selector |
]


"*************************************************************
 kernel/Object.st
 *************************************************************"
nil subclass: Object [

    == anObject [
        <category: 'testing'>

        <primitive: VMPrimitiveObjectEq>
    ]

    = anObject [
        <category: 'testing'>

        <primitive: VMPrimitiveObjectEq>
    ]

    ~= anObject [
        <category: 'testing'>

        ^ (self = anObject) == false
    ]

    ~~ anObject [
        <category: 'testing'>

        ^ (self == anObject) == false
    ]

    at: anIndex [
        <category: 'accessing'>

        <primitive: VMPrimitiveObjectAt>
    ]

    basicAt: anIndex [
        <category: 'accessing'>

        <primitive: VMPrimitiveObjectAt>
    ]

    at: anIndex put: value [
        <category: 'accessing'>

        <primitive: VMPrimitiveObjectAtPut>
    ]

    basicAt: anIndex put: value [
        <category: 'accessing'>

        <primitive: VMPrimitiveObjectAtPut>
    ]

    size [
        <category: 'accessing'>

        <primitive: VMPrimitiveObjectSize>
    ]

    basicSize [
        <category: 'accessing'>

        <primitive: VMPrimitiveObjectSize>
    ]

    becomeForward: otherObject [
        <category: 'accessing'>

        <primitive: VMPrimitiveObjectBecomeForward>
    ]
```

CorGenCode

```
    become: otherObject [
        <category: 'accessing'>

        <primitive: VMPrimitiveObjectBecome>
    ]

    instVarAt: index [
        <category: 'accessing'>

        <primitive: VMPrimitiveObjectInstVarAt>
    ]

    instVarAt: index put: value [
        <category: 'accessing'>

        <primitive: VMPrimitiveObjectInstVarAtPut>
    ]

    class [
        <category: 'accessing'>

        <primitive: VMPrimitiveObjectClass>
    ]

    changeClassTo: aClass [
        <category: 'accessing'>

        <primitive: VMPrimitiveObjectChangeClassTo>
    ]

    isNil [
        <category: 'testing functionality'>

        ^ false
    ]

    notNil [
        <category: 'testing functionality'>

        ^ true
    ]

    ifNil: nilBlock [
        <category: 'testing functionality'>

        ^ self
    ]

    copy [
        <category: 'copying'>

        ^ self shallowCopy postCopy
    ]

    shallowCopy [
        <category: 'copying'>

        self error
    ]

    postCopy [
        <category: 'copying'>

        ^ self
    ]

    deepCopy [
        <category: 'copying'>

    ]

    yourself [
        <category: 'class type methods'>

        ^ self
```

```
    ]

    identityHash [
        <category: 'hash'>

        <primitive: VMPrimitiveObjectHash>
    ]

    hash [
        <category: 'hash'>

        <primitive: VMPrimitiveObjectHash>
    ]

    allOwners [
        <category: 'reflection'>

        <primitive: VMPrimitiveObjectOwners>
    ]

    nextInstance [
        <category: 'reflection'>

        <primitive: VMPrimitiveObjectNextInstance>
    ]

    doesNotUnderstand: aMessage [
        <category: 'error handling'>
    ]

    perform: selectorOrMessageOrMethod [
    ]

    perform: selectorOrMethod with: arg1 [
    ]

    perform: selectorOrMethod with: arg1 with: arg2 [
    ]

    perform: selectorOrMethod with: arg1 with: arg2 with: arg3 [
    ]

    perform: selectorOrMethod with: arg1 with: arg2 with: arg3 with: arg4 [
    ]

    perform: selectorOrMethod withArguments: argumentsArray [
    ]
]


"************************************************************
 kernel/PolymorphicInlineCaching.st
 ************************************************************"
Object subclass: PolymorphicInlineCaching [
    | selector cmpCode |
]


"************************************************************
 kernel/ProcessMemorySpace.st
 ************************************************************"
ObjectMemorySpace subclass: ProcessMemorySpace [
]

"************************************************************
 kernel/ProcessorScheduler.st
 ************************************************************"
Object subclass: ProcessorScheduler [
    ProcessorScheduler class [ | uniqueInstance | ]

    | scheduler processLists activeProcess idleTasks processTimeslice gcSemaphor
e gcArray |
]
```



```
"************************************************************
 kernel/True.st
 ************************************************************"
Boolean subclass: True [
]


"************************************************************
 kernel/UndefinedObject.st
 ************************************************************"
Object subclass: UndefinedObject [
]


"************************************************************
 kernel/Collections/Array.st
 ************************************************************"
ArrayedCollection subclass: Array [
]


"************************************************************
 kernel/Collections/ArrayedCollection.st
 ************************************************************"
SequenceableCollection subclass: ArrayedCollection [
]


"************************************************************
 kernel/Collections/Bag.st
 ************************************************************"
Collection subclass: Bag [
]


"************************************************************
 kernel/Collections/BindingDictionary.st
 ************************************************************"
Dictionary subclass: BindingDictionary [
]


"************************************************************
 kernel/Collections/ByteArray.st
 ************************************************************"
Array subclass: ByteArray [
]


"************************************************************
 kernel/Collections/CharacterArray.st
 ************************************************************"
Array subclass: CharacterArray [
]


"************************************************************
 kernel/Collections/Collection.st
 ************************************************************"
Iterable subclass: Collection [
]


"************************************************************
 kernel/Collections/CompiledBlock.st
 ************************************************************"
CompiledCode subclass: CompiledBlock [
    | method |
]


"************************************************************
 kernel/Collections/CompiledCode.st
 ************************************************************"
ArrayedCollection subclass: CompiledCode [
```

CorGenCode

```
    | literals stackDepth numTemps numArgs |
]


"************************************************************
 kernel/Collections/CompiledMethod.st
 ************************************************************"
CompiledCode subclass: CompiledMethod [
    | primitive descriptor |
]


"************************************************************
 kernel/Collections/Dictionary.st
 ************************************************************"
HashedCollection subclass: Dictionary [
]


"************************************************************
 kernel/Collections/HashedCollection.st
 ************************************************************"
Collection subclass: HashedCollection [
    | tally array |

]


"************************************************************
 kernel/Collections/IdentityDictionary.st
 ************************************************************"
LookupTable subclass: IdentityDictionary [
]


"************************************************************
 kernel/Collections/Iterable.st
 ************************************************************"
Object subclass: Iterable [
]


"************************************************************
 kernel/Collections/Link.st
 ************************************************************"
Object subclass: Link [
    | nextLink |
]


"************************************************************
 kernel/Collections/LinkedList.st
 ************************************************************"
SequenceableCollection subclass: LinkedList [
    | firstLink lastLink |
]


"************************************************************
 kernel/Collections/LookupTable.st
 ************************************************************"
Dictionary subclass: LookupTable [
]


"************************************************************
 kernel/Collections/MethodDictionary.st
 ************************************************************"
LookupTable subclass: MethodDictionary [
]


"************************************************************
 kernel/Collections/OrderedCollection.st
 ************************************************************"
```

CorGenCode

```
SequenceableCollection subclass: OrderedCollection [
]


"************************************************************
 kernel/Collections/Process.st
 ************************************************************"
Link subclass: Process [
    | suspendedContext priority myList name environment interrupts interruptLock
 |
]


"************************************************************
 kernel/Collections/Semaphore.st
 ************************************************************"
LinkedList subclass: Semaphore [
    | signals name |
]


"************************************************************
 kernel/Collections/SequenceableCollection.st
 ************************************************************"
Collection subclass: SequenceableCollection [
]


"************************************************************
 kernel/Collections/Set.st
 ************************************************************"
HashedCollection subclass: Set [
]


"************************************************************
 kernel/Collections/String.st
 ************************************************************"
CharacterArray subclass: String [
]


"************************************************************
 kernel/Collections/Symbol.st
 ************************************************************"
String subclass: Symbol [

    Symbol class [ | SymbolTable | ]

    Symbol class >> new [
        <category: 'instance creation'>

        self shouldNotImplement
    ]
]


"************************************************************
 kernel/Collections/SystemDictionary.st
 ************************************************************"
BindingDictionary subclass: SystemDictionary [
]


"************************************************************
 kernel/Collections/WeakSet.st
 ************************************************************"
Set subclass: WeakSet [
]


"************************************************************
 kernel/Magnitude/Association.st
 ************************************************************"
LookupKey subclass: Association [
```

CorGenCode

```
    | value |
]

"***********************************************************
 kernel/Magnitude/Character.st
 ***********************************************************"
Magnitude subclass: Character [
    Character class [ | CharacterTable | ]

    | value |
]

"***********************************************************
 kernel/Magnitude/Float.st
 ***********************************************************"
Number subclass: Float [
]

"***********************************************************
 kernel/Magnitude/Fraction.st
 ***********************************************************"
Number subclass: Fraction [
]

"***********************************************************
 kernel/Magnitude/HomedAssociation.st
 ***********************************************************"
Association subclass: HomedAssociation [
    | environment |
]

"***********************************************************
 kernel/Magnitude/Integer.st
 ***********************************************************"
Number subclass: Integer [
]

"***********************************************************
 kernel/Magnitude/LookupKey.st
 ***********************************************************"
Magnitude subclass: LookupKey [
    | key |
]

"***********************************************************
 kernel/Magnitude/Magnitude.st
 ***********************************************************"
Object subclass: Magnitude [
]

"***********************************************************
 kernel/Magnitude/MethodInfo.st
 ***********************************************************"
Object subclass: MethodInfo [
]

"***********************************************************
 kernel/Magnitude/Number.st
 ***********************************************************"
Magnitude subclass: Number [
]

"***********************************************************
 kernel/Magnitude/SmallInteger.st
 ***********************************************************"
```

CorGenCode

```
Integer subclass: SmallInteger [

    = anObject [
        <category: 'testing'>

        <primitive: VMPrimitiveIntegerEq>
    ]

    < anObject [
        <category: 'testing'>

        <primitive: VMPrimitiveIntegerLt>
    ]

    > anObject [
        <category: 'testing'>

        <primitive: VMPrimitiveIntegerGt>
    ]

]

"***********************************************************
 kernel/Magnitude/VariableBinding.st
 ***********************************************************"
HomedAssociation subclass: VariableBinding [
]

"***********************************************************
 kernel/primitives/Primitive.st
 ***********************************************************"
Object subclass: Primitive [

    Primitive class [ | numbers | ]

    Primitive class >> description [
        <category: 'accessing'>

        self subclassResponsibility
    ]

    Primitive class >> initializeNumber [
        <category: 'initialization'>

        numbers ifNil: [ | i |
            i := 1.
            numbers := Dictionary new.
            (self subclasses asSortedCollection: [ :a :b | a name < b name ]) do
: [ :each |
                numbers at: each name put: i.
                i := i + 1 ] ].
    ]

    Primitive class >> number [
        <category: 'accessing'>

        ^ self numberFor: self name
    ]

    Primitive class >> numberFor: aSymbol [
        <category: 'accessing'>

        self initializeNumber.
        ^ numbers at: aSymbol
    ]

    Primitive class >> at: anInteger [
        <category: 'accessing'>

        | name |
        self initializeNumber.
        name := (numbers keyAtValue: anInteger ifAbsent: [ self error: 'Primitiv
e ', anInteger, ' not found' ]).
```

```
        ^ self environment at: name
    ]

    Primitive class >> doIt: anInterpret [
        <category: 'accessing'>

        self subclassResponsibility
    ]
]


"****************************************************************
 kernel/primitives/VMPrimitiveBehaviorNew.st
 ****************************************************************"
Primitive subclass: VMPrimitiveBehaviorNew [

    VMPrimitiveBehaviorNew class >> description [
        <category: 'accessing'>

        self subclassResponsibility
    ]

    VMPrimitiveBehaviorNew class >> doIt: anInterpret [
        <category: 'accessing'>

        | oop |
        oop := anInterpret instantiate: anInterpret top sized: 0.
        anInterpret top: oop
    ]
]


"****************************************************************
 kernel/primitives/VMPrimitiveBehaviorNewColon.st
 ****************************************************************"
Primitive subclass: VMPrimitiveBehaviorNewColon [

    VMPrimitiveBehaviorNewColon class >> doIt: anInterpret [
        <category: 'accessing'>

        | oop size |
        size := anInterpret pop.
        oop := anInterpret instantiate: anInterpret top sized: size.
        anInterpret top: oop
    ]
]


"****************************************************************
 kernel/primitives/VMPrimitiveIntegerEq.st
 ****************************************************************"
Primitive subclass: VMPrimitiveIntegerEq [

    VMPrimitiveIntegerEq class >> description [
        <category: 'accessing'>

        ^ 'Compare two integer numbers'
    ]

    VMPrimitiveIntegerEq class >> doIt: anInterpret [

        | x y |
        (x := anInterpret pop) isInteger ifFalse: [ self error: 'receiver is not
 an integer' ].
        (y := anInterpret pop) isInteger ifFalse: [ self error: 'argument is not
 an integer : ', (anInterpret top) ].
        "anInterpret receiver = anInterpret pop ifTrue: [ anInterpret pushTrueOO
P ] ifFalse: [ anInterpret pushFalseOOP ]."
        anInterpret receiver = anInterpret pop ifTrue: [ anInterpret pushTrueOOP
 ] ifFalse: [ anInterpret pushFalseOOP ].
        "anInterpret returnToTop"
    ]
]
```

CorGenCode

```
"****************************************************************
 kernel/primitives/VMPrimitiveIntegerGe.st
 ****************************************************************"
Primitive subclass: VMPrimitiveIntegerGe [

    VMPrimitiveIntegerGe class >> description [
        <category: 'accessing'>

        ^ 'Compare two integer numbers'
    ]

    VMPrimitiveIntegerGe class >> doIt: anInterpret [

        | x y |
        (x := anInterpret pop) isInteger ifFalse: [ self error: 'receiver is not
 an integer' ].
        (y := anInterpret pop) isInteger ifFalse: [ self error: 'argument is not
 an integer : ', (anInterpret top) ].
        "anInterpret receiver = anInterpret pop ifTrue: [ anInterpret pushTrueOO
P ] ifFalse: [ anInterpret pushFalseOOP ]."
        anInterpret receiver = anInterpret pop ifTrue: [ anInterpret pushTrueOOP
 ] ifFalse: [ anInterpret pushFalseOOP ].
        "anInterpret returnToTop"
    ]
]


"****************************************************************
 kernel/primitives/VMPrimitiveIntegerGt.st
 ****************************************************************"
Primitive subclass: VMPrimitiveIntegerGt [

    VMPrimitiveIntegerGt class >> doIt: anInterpret [

        | i j |
        (j := anInterpret pop) isInteger ifFalse: [ self error: 'receiver is not
 an integer' ].
        (i := anInterpret pop) isInteger ifFalse: [ self error: 'argument is not
 an integer' ].
        i > j
            ifTrue: [ anInterpret pushTrueOOP ]
            ifFalse: [ anInterpret pushFalseOOP ]
    ]
]


"****************************************************************
 kernel/primitives/VMPrimitiveIntegerLe.st
 ****************************************************************"
Primitive subclass: VMPrimitiveIntegerLe [

    VMPrimitiveIntegerLe class >> description [
        <category: 'accessing'>

        ^ 'Compare two integer numbers'
    ]

    VMPrimitiveIntegerLe class >> doIt: anInterpret [

        | x y |
        (x := anInterpret pop) isInteger ifFalse: [ self error: 'receiver is not
 an integer' ].
        (y := anInterpret pop) isInteger ifFalse: [ self error: 'argument is not
 an integer : ', (anInterpret top) ].
        "anInterpret receiver = anInterpret pop ifTrue: [ anInterpret pushTrueOO
P ] ifFalse: [ anInterpret pushFalseOOP ]."
        anInterpret receiver = anInterpret pop ifTrue: [ anInterpret pushTrueOOP
 ] ifFalse: [ anInterpret pushFalseOOP ].
        "anInterpret returnToTop"
    ]
]


"****************************************************************
 kernel/primitives/VMPrimitiveIntegerLt.st
```

```
 ***************************************************************"
Primitive subclass: VMPrimitiveIntegerLt [

    VMPrimitiveIntegerLt class >> doIt: anInterpret [

        | i j |
        (j := anInterpret pop) isInteger ifFalse: [ self error: 'argument is not
 an integer' ].
        (i := anInterpret pop) isInteger ifFalse: [ self error: 'argument is not
 an integer' ].
        i < j
            ifTrue: [ anInterpret pushTrueOOP ]
            ifFalse: [ anInterpret pushFalseOOP ]
    ]
]

"**************************************************************
 kernel/primitives/VMPrimitiveNew.st
 ***************************************************************"
Primitive subclass: VMPrimitiveNew [
    VMPrimitiveNew class >> description [

        ^ 'Instantiate a new object'
    ]
]

"**************************************************************
 kernel/primitives/VMPrimitiveObjectAt.st
 ***************************************************************"
Primitive subclass: VMPrimitiveObjectAt [

    VMPrimitiveObjectAt class >> doIt: anInterpret [

        | index |
        index := anInterpret pop.
        anInterpret top: (anInterpret top oopAt: index)
    ]
]

"**************************************************************
 kernel/primitives/VMPrimitiveObjectAtPut.st
 ***************************************************************"
Primitive subclass: VMPrimitiveObjectAtPut [

    VMPrimitiveObjectAtPut class >> doIt: anInterpret [

        | index oop |
        oop := anInterpret pop.
        index := anInterpret pop.
        anInterpret top: (anInterpret top oopAt: index put: oop)
    ]
]

"**************************************************************
 kernel/primitives/VMPrimitiveObjectBecome.st
 ***************************************************************"
Primitive subclass: VMPrimitiveObjectBecome [
]

"**************************************************************
 kernel/primitives/VMPrimitiveObjectBecomeForward.st
 ***************************************************************"
Primitive subclass: VMPrimitiveObjectBecomeForward [
]

"**************************************************************
 kernel/primitives/VMPrimitiveObjectChangeClassTo.st
 ***************************************************************"
Primitive subclass: VMPrimitiveObjectChangeClassTo [
```

CorGenCode

```

    VMPrimitiveObjectChangeClassTo class >> doIt: anInterpreter [

        | class |
        class := anInterpreter pop.
        anInterpreter top isInteger ifTrue: [ self error: 'cannot change class o
f integer' ].
        anInterpreter top: (anInterpreter top oopClass: class)
    ]

]

"**************************************************************
 kernel/primitives/VMPrimitiveObjectClass.st
 ***************************************************************"
Primitive subclass: VMPrimitiveObjectClass [

    VMPrimitiveObjectClass class >> doIt: anInterpreter [

        anInterpreter top: (anInterpreter top isInteger
                                            ifTrue: [ anInterpreter smallInteger
 ]
                                            ifFalse: [ anInterpreter top oopClas
s ])
    ]
]

"**************************************************************
 kernel/primitives/VMPrimitiveObjectEq.st
 ***************************************************************"
Primitive subclass: VMPrimitiveObjectEq [

    VMPrimitiveObjectEq class >> doIt: anInterpreter [

        | oop |
        oop := anInterpreter pop.
        anInterpreter top: (anInterpreter top == oop)
    ]
]
"**************************************************************
 kernel/primitives/VMPrimitiveObjectHash.st
 ***************************************************************"
Primitive subclass: VMPrimitiveObjectHash [
]

"**************************************************************
 kernel/primitives/VMPrimitiveObjectInstVarAt.st
 ***************************************************************"
Primitive subclass: VMPrimitiveObjectInstVarAt [

    VMPrimitiveObjectInstVarAt class >> doIt: anInterpreter [

        | index |
        anInterpreter top isInteger ifFalse: [ ^ self error: 'should be an integ
er' ].
        index := anInterpreter pop.
        anInterpreter top: (anInterpreter top oopInstVarAt: index)
    ]

]

"**************************************************************
 kernel/primitives/VMPrimitiveObjectInstVarAtPut.st
 ***************************************************************"
Primitive subclass: VMPrimitiveObjectInstVarAtPut [

    VMPrimitiveObjectInstVarAtPut class >> doIt: anInterpreter [

        | index oop |
        oop := anInterpreter pop.
```

```
        anInterpreter top isInteger ifFalse: [ ^ self error: 'should be an integ
er' ].
        index := anInterpreter pop.
        anInterpreter top: (anInterpreter top oopInstVarAt: index put: oop)
    ]
]

"*************************************************************
 kernel/primitives/VMPrimitiveObjectNextInstance.st
 *************************************************************"
Primitive subclass: VMPrimitiveObjectNextInstance [
]

"*************************************************************
 kernel/primitives/VMPrimitiveObjectOwners.st
 *************************************************************"
Primitive subclass: VMPrimitiveObjectOwners [
]

"*************************************************************
 kernel/primitives/VMPrimitiveObjectSize.st
 *************************************************************"
Primitive subclass: VMPrimitiveObjectSize [

    VMPrimitiveObjectSize class >> doIt: anInterpret [

        anInterpret top: (anInterpret top oopSize)
    ]
]

"*************************************************************
 kernel/primitives/VMPrimitivePrint.st
 *************************************************************"
Primitive subclass: VMPrimitivePrint [

    VMPrimitivePrint class >> description [
        <category: 'accessing'>

        ^ 'Output string on the display, used for the bootstrap step'
    ]

    VMPrimitivePrint class >> doIt: anInterpret [

        anInterpret top printOOPString
    ]
]

"*************************************************************
 kernel/primitives/VMPrimitiveValue.st
 *************************************************************"
Primitive subclass: VMPrimitiveValue [

    VMPrimitiveValue class >> description [
        <category: 'accessing'>

        ^ 'BlockClosure value'
    ]

    VMPrimitiveValue class >> doIt: anInterpret [

        anInterpret blockValue
    ]
]

"*************************************************************
 kernel/shape/ByteShape.st
 *************************************************************"
Shape subclass: ByteShape [
```

CorGenCode

```
]

"*************************************************************
 kernel/shape/DoubleWord.st
 *************************************************************"
Shape subclass: DoubleWord [
]

"*************************************************************
 kernel/shape/QuadWord.st
 *************************************************************"
Shape subclass: QuadWord [
]

"*************************************************************
 kernel/shape/Shape.st
 *************************************************************"
Object subclass: Shape [
]

"*************************************************************
 kernel/shape/WordShape.st
 *************************************************************"
Shape subclass: WordShape [
]
```