A few additional AST lowerings that can simplify the development of new backends

Miguel Garcia
http://lamp.epfl.ch/~magarcia

LAMP, EPFL

2011-02-22

Outline

- There is a way to reduce the effort it takes to develop a new backend for Scala
- How? By factoring out those AST lowerings that are shared across different backends (otherwise, each backend developer must handle them anew)
- Moreover, developers of non-backend compiler plugins also stand to benefit from this approach.

A brief history (so far) of non-VM backends:

- `CSharpGen`[1]
  - Goal: emitting C# 3.0 source files,
    acting as a drop-in replacement of `GenICode` → `GenMSIL`
  - Input: `BlockExpr`-free, OO code ("structured GOTO" allowed.)
  - Status: Assigned to community
- Geoffrey Reedy's *Scala to LLVM*[2]
  - Goal: emitting LLVM IR
  - Input: SSA (procedural 3-Address instr with phi-nodes in CFGs)
  - Status: Under development
- The mythical *Scala to Java* translator
  - Goal: emitting Java 1.5 source files
  - Input: GOTO-less, `BlockExpr`-free, OO code.
  - Status: Abandoned

[1] `http://lamp.epfl.ch/~magarcia/ScalaNET/2011Q1/CSharpGen.pdf`
[2] `http://greedy.github.com/scala/`

What-if the following post-CleanUp transformations were available:

1. GOTO elimination[3] *(implemented 50%)*
   rephrases ASTs containing arbitrary jumps into ASTs containing
   - structured control flow constructs, plus
   - additional boolean variables to pick the original execution paths.

2. `BlockExpr` desugaring:
   flatten block expressions, assign value to variable and use later.

3. Explicit eval order:
   again by assigning to temporary variables. Useful to rule out wrong-order in “source-language backends". For example:
   “`x() + y() * z()`” vs. “`x() + (y() * z())`”

Taken together, they bring the AST into GOTO-less 3-Address form.

[3] `http://lamp.epfl.ch/~magarcia/ScalaCompilerCornerReloaded/2011Q1/JumpsRemover.pdf`

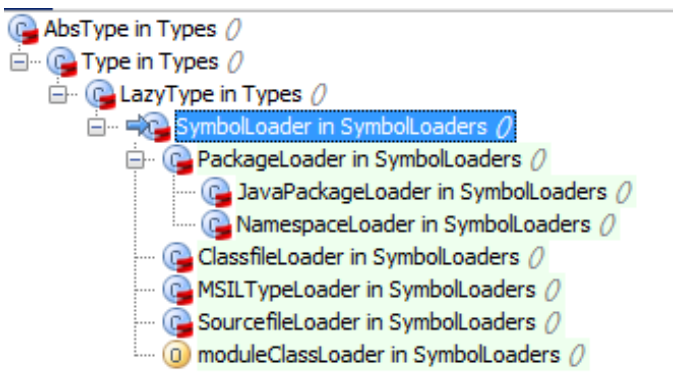
What about SSA (details on backup slide)

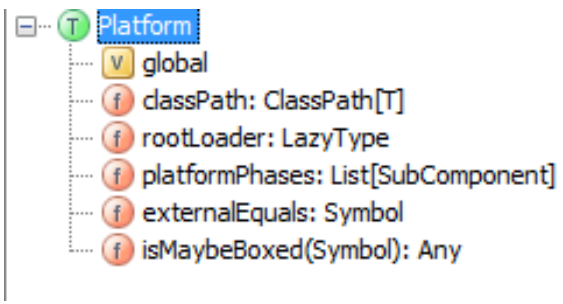
- `Tree` nodes can't represent SSA *explicitly*, so we stop rewriting.
  In any case, SSA from 3-Address not harder than from bytecode:
  SSA-functional connection "at hand" after previous rewritings.

A straightforward way to check whether lowerings preserve semantics

- let `GenICode` → `GenJVM` run as usual afterwards

Granted, some unavoidable (genuine) platform adapting still needed
(a barrier that platform gurus can tackle with far less compiler mojo)





Platform assumptions (SLS Ch. 12)
TODO: *Porting Guide of* `scalac`

A small number of (post-CleanUp) AST lowerings:

1. GOTO elimination
2. `BlockExpr` flattening
3. making eval order explicit

pave the way to a large number of new backends:

- C# 3.0, Java 1.5, “low-level JavaScript” (ditto for ActionScript)
- program verification backends (in particular, Spec#)
- LLVM, Google Native Web Client
- Android NDK (Native Development Kit), Apple iOS
- other “embedded” VMs (tablets, etc.)
- Google @ Game DevCon: `google.com/events/gdc/2011/agenda.html`

and that’s without counting the SSA-functional connection! :-)

Appendix (1 of 2): Bibliograpy for slide 6 (SSA)

- A. Gal, Ch. W. Probst, and M. Franz
  Structural Encoding of Static Single Assignment Form[4]
- Navindra Umanee: *SOOT Shimple: An Investigation of SSA*[5]
- W. Amme, N. Dalton, J. von Ronne, and M. Franz
  SafeTSA: A Type Safe and Referentially Secure Mobile-Code Representation Based on Static Single Assignment Form
- Section on “functional" in this[6] SSA bibliography
- Marius Nita’s *A Functional Intermediate Form for Soot*[7]

[4] http://dx.doi.org/10.1016/j.entcs.2005.02.045

[5] http://www.sable.mcgill.ca/publications/thesis/masters-navindra/sable-thesis-2006-masters-navindra-double-sided.pdf

[6] http://www.cs.man.ac.uk/~jsinger/ssa.html

[7] http://web.cecs.pdx.edu/~marius/files/hw/grad_compilers/results.pdf

Appendix (2 of 2): Bibliography for slide 7 (native compilers)

- Unlike for Android, native development only with partner-license:
  - Windows Phone 7
  - Silverlight
  - XNA (Xbox 360). Note: the Xbox "1.0" had a C++ XDK[8].
- What defines those platforms instead is the *base framework*, e.g. for XNA it's a subset of the *.NET Compact Framework*.
- Assemblies relying on a platform's base framework can run on it:
  - F# for Game Development[9]
  - Using XNA Game Studio with other programming languages[10]
- NGEN can compile those assemblies into (platform-specific) executable images[11], but JITting is on par except for startup time.

[8] http://en.wikipedia.org/wiki/Xbox_Development_Kit
[9] http://sharp-gamedev.blogspot.com/
[10] http://forums.create.msdn.com/forums/p/1464/7267.aspx
[11] http://en.wikipedia.org/wiki/Native_Image_Generator