1 A new backend for Scala.NET to emit C# sources

1.1 Motivation

In corporate software development, company-wide standards limit the choice of programming language. Scala.NET can overcome those barriers by providing a new backend phase (dubbed `GenCSharp`) to emit C# 3.0 source files, that otherwise does a similar job as the pipeline `GenICode` → `GenMSIL`.

Additionally, a C#-backend is relevant from an engineering perspective (e.g., to quantify performance across different compiler pipelines, and as a bridge towards the Spec# system for program verification).

1.2 Project milestones

Making `GenCSharp` a reality involves solving a number of problems. The good news is that the required transformations can be grouped into milestones, with a number of advantages:

1. each milestone delivers a fully-functional compiler, thus making possible to test work-in-progress against the compiler test suite;
2. all but the last milestone are platform-independent (and in fact, a correctness criteria is that they also work in `forJVM` mode);
3. milestones 1 and 2 perform intra-method rewritings only, and milestone 3 is desirable but not essential for `GenCSharp`
4. the transformations in milestones 1 to 3 are independent of each other. This means that:
   (a) repeatedly applying the transformations in a given phase results in a fixpoint (no help from other milestone is needed, no constructs are introduced that another milestone would have to reduce),
   (b) these transformations can be applied in any order.

1.2.1 Milestone 1: `goto`-elimination

Scala ASTs may contain jumps that straddle block nesting, and not just from inner-to-outer blocks as supported by C#'s `goto` statement. A *GOTO elimination*[1] technique has been devised, to rephrase those trees in a semantics-preserving manner such that the resulting trees exhibit structured control flow only, with extra boolean variables to pick the intended execution path.

1.2.2 Milestone 2: Lowerings after `CleanUp`

This milestone *was* about transforming ASTs from an expression language into an imperative, statement-based language. In the meantime, that has been accomplished as described in *Moving Scala ASTs one step closer to C#*[2].

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

[2] `http://lamp.epfl.ch/~magarcia/ScalaCompilerCornerReloaded/2011Q2/Moving3A.pdf`

1.2.3 Milestone 3: Factor out initialization semantics

The following rewritings (dealing with initialization semantics) are performed in both `GenJVM` and `GenMSIL`. Common wisdom calls for performing them only once, e.g. as part of post-`CleanUp` transformations. Otherwise they need to be carried out in `GenCSharp` anyway, so better to hoist them.

- Sec. 1, *Desugaring of module initialization*[3]
- Sec. 1.9, *Adding static constructors*[4]

1.2.4 Milestone 4: Fire off your C# compilers

With the previous functionality in place, we can now get serious about emitting C# sources. For that, there's one more sub-problem to solve: early defs do require "bytecode inlining" (which is possible in C#, see `ILInline`[5]). The write-up *How the constructors phase works*[6] covers how early defs end up before a super-constructor call.

(As a sidenote, at least one Java compiler allows inlining bytecode, but that's *another* story[7]).

1.3 Additional resources

1. *The Scala Compiler Corner*
2. *scala-internals* Google group
3. In addition to a `goto`-elimination protoype, there's an *unparser*[8] to turn after-`typer` ASTs back into Scala source files. Unparsing is way easier by the time `GenCSharp` would run (thus most of the special cases in that plugin won't be needed by `GenCSharp`) but anyway it provides details about the connection between tree shapes and surface syntax.

[3] `http://lamp.epfl.ch/~magarcia/ScalaCompilerCornerReloaded/2010Q2/i2i.pdf`
[4] `http://lamp.epfl.ch/~magarcia/ScalaCompilerCornerReloaded/2010Q4/ikvmify3.pdf`
[5] `http://blogs.msdn.com/jmstall/archive/2005/02/21/377806.aspx`
[6] `http://lamp.epfl.ch/~magarcia/ScalaCompilerCornerReloaded/2011Q2/ConstrPhase.pdf`
[7] `http://www.program-transformation.org/Stratego/TheDryadCompiler`
[8] "Unparsing" at `http://lamp.epfl.ch/~magarcia/ScalaCompilerCornerReloaded/`