Dafny: An Automatic Program Verifier for Functional Correctness


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics.Contracts;
using Bpl = Microsoft.Boogie;

namespace Microsoft.Dafny
{
  public class DafnyOptions : Bpl.CommandLineOptions
  {
    private ErrorReporter errorReporter;

    public DafnyOptions(ErrorReporter errorReporter = null)
      : base("Dafny", "Dafny program verifier") {
        this.errorReporter = errorReporter;
        SetZ3ExecutableName();
    }

    public override string VersionNumber {
      get {
        return System.Diagnostics.FileVersionInfo.GetVersionInfo(System.Reflection.Assembl
#if ENABLE_IRONDAFNY
          + "[IronDafny]"
#endif
          ;
      }
    }
    public override string VersionSuffix {
      get {
        return " version " + VersionNumber + ", Copyright (c) 2003-2015, Microsoft.";
      }
    }

    private static DafnyOptions clo;
    public static DafnyOptions O {
      get { return clo; }
    }

    public static void Install(DafnyOptions options) {
      Contract.Requires(options != null);
      clo = options;
      Bpl.CommandLineOptions.Install(options);
    }

    public bool UnicodeOutput = false;
    public bool DisallowSoundnessCheating = false;
    public bool Dafnycc = false;
    public int Induction = 3;
    public int InductionHeuristic = 6;
    public string DafnyPrelude = null;
    public string DafnyPrintFile = null;
    public enum PrintModes { Everything, NoIncludes, NoGhost };
    public PrintModes PrintMode = PrintModes.Everything;  // Default to printing everythi
```

```
    public PrintModes PrintMode = PrintModes.Everything;  // default to printing everythi
    public bool DafnyVerify = true;
    public string DafnyPrintResolvedFile = null;
    public bool Compile = true;
    public bool ForceCompile = false;
    public bool RunAfterCompile = false;
    public bool SpillTargetCode = false;
    public bool DisallowIncludes = false;
    public bool DisableNLarith = false;
    public string AutoReqPrintFile = null;
    public bool ignoreAutoReq = false;
    public bool AllowGlobals = false;
    public bool CountVerificationErrors = true;
    public bool Optimize = false;
    public bool AutoTriggers = false;
    public bool RewriteFocalPredicates = true;
    public bool PrintTooltips = false;
    public bool PrintStats = false;
    public bool PrintFunctionCallGraph = false;
    public bool WarnShadowing = false;
    public bool IronDafny =
#if ENABLE_IRONDAFNY
      true
#else
      false
#endif
    ;

    protected override bool ParseOption(string name, Bpl.CommandLineOptionEngine.CommandLi
      var args = ps.args;  // convenient synonym

      switch (name) {
        case "dprelude":
          if (ps.ConfirmArgumentCount(1)) {
            DafnyPrelude = args[ps.i];
          }
          return true;

        case "dprint":
          if (ps.ConfirmArgumentCount(1)) {
            DafnyPrintFile = args[ps.i];
          }
          return true;

        case "printMode":
          if (ps.ConfirmArgumentCount(1)) {
            if (args[ps.i].Equals("Everything")) {
              PrintMode = PrintModes.Everything;
            }
            else if (args[ps.i].Equals("NoIncludes"))
            {
                PrintMode = PrintModes.NoIncludes;
            }
            else if (args[ps.i].Equals("NoGhost"))
            {
                PrintMode = PrintModes.NoGhost;
            }
            else
            {
                throw new Exception("Invalid value for printMode");
            }
          }
          return true;

        case "rprint":
          if (ps.ConfirmArgumentCount(1)) {
            DafnyPrintResolvedFile = args[ps.i];
          }
          return true;
```

```
            case "compile": {
                int compile = 0;
                if (ps.GetNumericArgument(ref compile, 4)) {
                  // convert option to two booleans
                  Compile = compile != 0;
                  ForceCompile = compile == 2;
                  RunAfterCompile = compile == 3;
                }
                return true;
              }

            case "dafnyVerify":
                {
                    int verify = 0;
                    if (ps.GetNumericArgument(ref verify, 2)) {
                        DafnyVerify = verify != 0; // convert to boolean
                    }
                    return true;
                }

            case "spillTargetCode": {
                int spill = 0;
                if (ps.GetNumericArgument(ref spill, 2)) {
                  SpillTargetCode = spill != 0;  // convert to a boolean
                }
                return true;
              }

            case "dafnycc":
              Dafnycc = true;
              Induction = 0;
              Compile = false;
              UseAbstractInterpretation = false; // /noinfer
              return true;

            case "noCheating": {
                int cheat = 0; // 0 is default, allows cheating
                if (ps.GetNumericArgument(ref cheat, 2)) {
                  DisallowSoundnessCheating = cheat == 1;
                }
                return true;
              }

            case "induction":
              ps.GetNumericArgument(ref Induction, 4);
              return true;

            case "inductionHeuristic":
              ps.GetNumericArgument(ref InductionHeuristic, 7);
              return true;

            case "noIncludes":
              DisallowIncludes = true;
              return true;

            case "noNLarith":
              DisableNLarith = true;
              this.AddZ3Option("smt.arith.nl=false");
              return true;

            case "autoReqPrint":
              if (ps.ConfirmArgumentCount(1)) {
                  AutoReqPrintFile = args[ps.i];
              }
              return true;

            case "noAutoReq":
              ignoreAutoReq = true;
```

```
            IgnoreAbstract = true;
            return true;

          case "allowGlobals":
            AllowGlobals = true;
            return true;

          case "stats":
            PrintStats = true;
            return true;

          case "funcCallGraph":
            PrintFunctionCallGraph = true;
            return true;

          case "warnShadowing":
            WarnShadowing = true;
            return true;

          case "countVerificationErrors": {
            int countErrors = 1; // defaults to reporting verification errors
            if (ps.GetNumericArgument(ref countErrors, 2)) {
              CountVerificationErrors = countErrors == 1;
            }
            return true;
          }

          case "printTooltips":
            PrintTooltips = true;
            return true;

          case "autoTriggers": {
              int autoTriggers = 0;
              if (ps.GetNumericArgument(ref autoTriggers, 2)) {
                AutoTriggers = autoTriggers == 1;
              }
              return true;
            }

          case "rewriteFocalPredicates": {
              int rewriteFocalPredicates = 0;
              if (ps.GetNumericArgument(ref rewriteFocalPredicates, 2)) {
                RewriteFocalPredicates = rewriteFocalPredicates == 1;
              }
              return true;
            }

          case "optimize": {
              Optimize = true;
              return true;
          }

          case "noIronDafny": {
              IronDafny = false;
              return true;
          }

          case "ironDafny": {
              IronDafny = true;
              return true;
          }

          default:
            break;
        }
        // not a Dafny-specific option, so defer to superclass
        return base.ParseOption(name, ps);
      }

      public override void ApplyDefaultOptions() {
```

```
  public override void ApplyDefaultOptions() {
    base.ApplyDefaultOptions();

    // expand macros in filenames, now that LogPrefix is fully determined
    ExpandFilename(ref DafnyPrelude, LogPrefix, FileTimestamp);
    ExpandFilename(ref DafnyPrintFile, LogPrefix, FileTimestamp);
  }

  public override void AttributeUsage() {
    // TODO: provide attribute help here
  }


  /// <summary>
  /// Dafny comes with it's own copy of z3, to save new users the trouble of having to i
  /// For this to work, Dafny makes the Z3ExecutablePath point to the path were Z3 is pu
  /// For developers though (and people getting this from source), it's convenient to be
  /// so we vendor a Windows version.
  /// </summary>
  private void SetZ3ExecutableName() {
    var platform = (int)System.Environment.OSVersion.Platform;

    // http://www.mono-project.com/docs/faq/technical/
    var isUnix = platform == 4 || platform == 128;

    var z3binName = isUnix ? "z3" : "z3.exe";
    var dafnyBinDir = System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExec
    var z3BinDir = System.IO.Path.Combine(dafnyBinDir, "z3", "bin");
    var z3BinPath = System.IO.Path.Combine(z3BinDir, z3binName);

    if (!System.IO.File.Exists(z3BinPath) && !isUnix) {
      // This is most likely a Windows user running from source without downloading z3
      // separately; this is ok, since we vendor z3.exe.
      z3BinPath = System.IO.Path.Combine(dafnyBinDir, z3binName);
    }

    if (!System.IO.File.Exists(z3BinPath) && errorReporter != null) {
      var tok = new Bpl.Token(1, 1) { filename = "*** " };
      errorReporter.Warning(MessageSource.Other, tok, "Could not find '{0}' in '{1}'.{2}
        z3binName, z3BinDir, System.Environment.NewLine);
    } else {
```

Browsing changes in default ▼ tip as of **commit bdf96e1e6347**, 2 days ago

DafnyOptions.cs Compare with other versions: Select version ▼

```
    Console.WriteLine(@"  ---- Dafny options ----------------------------------------

Multiple .dfy files supplied on the command line are concatenated into one
Dafny program.

/dprelude:<file>
              choose Dafny prelude file
/dprint:<file>
              print Dafny program after parsing it
              (use - as <file> to print to console)
/printMode:<Everything|NoIncludes|NoGhost>
              Everything is the default.
              NoIncludes disables printing of {:verify false} methods incorporated via t
              include mechanism, as well as datatypes and fields included from other fil
              NoGhost disables printing of functions, ghost methods, and proof statement
              implementation methods.  It also disables anything NoIncludes disables.
/rprint:<file>
              print Dafny program after resolving it
              (use - as <file> to print to console)
/dafnyVerify:<n>
              0 - stop after typechecking
              1 - continue on to translation, verification, and compilation
/compile:<n>  0 - do not compile Dafny program
```

```
                1 (default) - upon successful verification of the Dafny
                    program, compile Dafny program to .NET assembly
                    Program.exe (if the program has a Main method) or
                    Program.dll (othewise), where Program.dfy is the name
                    of the last .dfy file on the command line
                2 - always attempt to compile Dafny program to C# program
                    out.cs, regardless of verification outcome
                3 - if there is a Main method and there are no verification
                    errors, compiles program in memory (i.e., does not write
                    an output file) and runs it
/spillTargetCode:<n>
                0 (default) - don't write the compiled Dafny program (but
                    still compile it, if /compile indicates to do so)
                1 - write the compiled Dafny program as a .cs file
/dafnycc        Disable features not supported by DafnyCC
/noCheating:<n>
                0 (default) - allow assume statements and free invariants
                1 - treat all assumptions as asserts, and drop free.
/induction:<n>
                0 - never do induction, not even when attributes request it
                1 - only apply induction when attributes request it
                2 - apply induction as requested (by attributes) and also
                    for heuristically chosen quantifiers
                3 (default) - apply induction as requested, and for
                    heuristically chosen quantifiers and lemmas
/inductionHeuristic:<n>
                0 - least discriminating induction heuristic (that is, lean
                    toward applying induction more often)
                1,2,3,4,5 - levels in between, ordered as follows as far as
                    how discriminating they are:  0 < 1 < 2 < (3,4) < 5 < 6
                6 (default) - most discriminating
/noIncludes     Ignore include directives
/noNLarith      Reduce Z3's knowledge of non-linear arithmetic (*,/,%).
                Results in more manual work, but also produces more predictable behavior.
/autoReqPrint:<file>
                Print out requirements that were automatically generated by autoReq.
/noAutoReq      Ignore autoReq attributes
/allowGlobals Allow the implicit class '_default' to contain fields, instance functions,
                and instance methods.  These class members are declared at the module scop
                outside of explicit classes.  This command-line option is provided to simp
                a transition from the behavior in the language prior to version 1.9.3, fro
                which point onward all functions and methods declared at the module scope
                implicitly static and fields declarations are not allowed at the module sc
/countVerificationErrors:<n>
                0 - If preprocessing succeeds, set exit code to 0 regardless of the number
                    of verification errors.
                1 (default) - If preprocessing succeeds, set exit code to the number of
                              verification errors.
/autoTriggers:<n>
                0 (default) - Do not generate {:trigger} annotations for user-level quanti
                1 - Add a {:trigger} to each user-level quantifier. Existing
                              annotations are preserved.
/rewriteFocalPredicates:<n>
                0 - Don't rewrite predicates in the body of prefix lemmas.
                1 (default) - In the body of prefix lemmas, rewrite any use of a focal pre
                              P to P#[_k-1].
/optimize       Produce optimized C# code, meaning:
                  - selects optimized C# prelude by passing
                    /define:DAFNY_USE_SYSTEM_COLLECTIONS_IMMUTABLE to csc.exe (requires
                    System.Collections.Immutable.dll in the source directory to successful
                    compile).
                  - passes /optimize flag to csc.exe.
/stats          Print interesting statistics about the Dafny files supplied.
/funcCallGraph Print out the function call graph.  Format is: func,mod=callee*
/warnShadowing  Emits a warning if the name of a declared variable caused another variab
                to be shadowed
/ironDafny      Enable experimental features needed to support Ironclad/Ironfleet. Use of
                these features may cause your code to become incompatible with future
                releases of Dafny.
```

```
                    releases of Dafny.
      /noIronDafny  Disable Ironclad/Ironfleet features, if enabled by default.
      /printTooltips
                    Dump additional positional information (displayed as mouse-over tooltips b
                    the VS plugin) to stdout as 'Info' messages.
    ");
          base.Usage();  // also print the Boogie options
        }
      }
    }
```