**Arguments**

There are four types of arguments:

* Plain string arguments, known at compile time, e.g. “configure”.
* Build parameters, computed at run time, e.g. libraries/bin-package-db.
* Environment parameters, computed by looking up the environment, e.g. binary == 0.7.2.3.
* Complex arguments, built up from several argument types, e.g. --with-ghc=path/to/ghc.exe.

Arguments passed to **ghc-cabal** when building **bin-package-db** with the bootstrapping compiler:

arg “configure”   
argBuildPath (depends on the package)  
argBuildDir (depends on the package & stage)

Positional arguments cannot be reordered, hence **→**

|  |
| --- |
| --constraint → bin-package-db == 0.0.0.0 --constraint → binary == 0.7.2.3 --constraint → Cabal == 1.22.0.0 --constraint → hoopl == 3.10.0.2 --constraint → hpc == 0.6.0.2 --constraint → transformers == 0.4.2.0 |

configure →  
libraries/bin-package-db →  
dist-boot →  
"" →  
--with-ghc=C:/msys/usr/local/bin/ghc.exe  
--with-ghc-pkg=C:/msys/usr/local/bin/ghc-pkg.exe  
--package-db=C:/msys/home/chEEtah/ghc/libraries/bootstrapping.conf  
--enable-library-vanilla  
--enable-library-for-ghci  
--disable-library-profiling  
--disable-shared  
--configure-option=CFLAGS=-fno-stack-protector  
--configure-option=LDFLAGS=  
--configure-option=CPPFLAGS=  
--gcc-options=-fno-stack-protector  
--configure-option=--with-iconv-includes=  
--configure-option=--with-iconv-libraries=  
--configure-option=--with-gmp-includes=  
--configure-option=--with-gmp-libraries=  
--configure-option=--with-cc=C:/msys/usr/local/lib/../mingw/bin/gcc.exe

…  
argWithStagedBuilder Gcc  
notStage Stage0 ? argWithBuilder Ld  
argWithBuilder Ar  
…

stage Stage0 ?  
 argPackageConstraints targetPackages  
  
Note that targetPackages also depends on stage and environment.

These arguments depend on the target ways, which in turn depend on stage and possibly on the environment (e.g., on the platform)

“--constraint” and the value of the constraint are separate arguments and cannot be reordered

--with-gcc=C:/msys/usr/local/lib/../mingw/bin/gcc.exe  
--with-ar=C:/msys/usr/bin/ar.exe  
--with-alex=C:/msys/usr/local/bin/alex.exe  
--with-happy=C:/msys/usr/local/bin/happy.exe

**Manipulating build expressions**

The following build expressions are used in the GHC build system:

* Target packages: a collection of packages to build
* Target ways: which ways to build
* Target directories: where to put build results
* Build settings: collections of arguments to pass to builders

**Example: target ways**

Consider the following collection of target ways:

* **vanilla**, always enabled
* **profiling**, disabled in stage 0
* **dynamic**, enabled if the platform supports shared libraries

The collection is *parameterised*, i.e. it depends on parameters such as stage, environment, etc. We can capture this as the following expression:

targetWays = {vanilla} ∪ {[stage ≠ 0] profiling} ∪ {[platformSupportsSharedLibs] dynamic}

Or, in Haskell:

targetWays :: Ways  
targetWays = msum  
 [ return vanilla  
 , notStage Stage0 ? return profiling  
 , platformSupportsSharedLibs ? return dynamic ]

The predicate **platformSupportsSharedLibs** depends on the contents of configuration files.

platformSupportsSharedLibs :: BuildPredicate  
platformSupportsSharedLibs =  
 not (targetPlatforms [ "powerpc-unknown-linux"  
 , "x86\_64-unknown-mingw32"  
 , "i386-unknown-mingw32" ]  
 ||  
 solarisBrokenShld && targetPlatform "i386-unknown-solaris2")

solarisBrokenShld :: BuildPredicate  
solarisBrokenShld = configYes "solaris-broken-shld"

targetPlatforms :: [String] -> BuildPredicate  
targetPlatforms = configValues "target-platform-full"

targetPlatform :: String -> BuildPredicate  
targetPlatform s = targetPlatforms [s]

In other words, shared libraries are not supported on platforms "powerpc-unknown-linux", "x86\_64-unknown-mingw32", and "i386-unknown-mingw32", and on the platform "i386-unknown-solaris2" if the flag "solaris-broken-shld" is set to Yes. The platform’s name is given by "target-platform-full" key.

Since targetWays is just a value we can print it:

**v** [!StageVariable 0]**p** [!(ConfigVariable "target-platform-full" "powerpc-unknown-linux" \/ ConfigVariable "target-platform-full" "x86\_64-unknown-mingw32" \/ ConfigVariable "target-platform-full" "i386-unknown-mingw32" \/ ConfigVariable "solaris-broken-shld" "YES" /\ ConfigVariable "target-platform-full" "i386-unknown-solaris2")]**dyn**

Where **v**, **p** and **dyn** stand for **vanilla**, **profiling**, and **dynamic** ways, respectively.

**Partial evaluation**

We can use two methods to partially evaluate targetWays, i.e. to set the parameters and find out the resulting set of target ways.

* We can **project** targetWays on a parameter’s value, e.g., by setting parameter stage to Stage1.  
  project Stage1 targetWays :: Ways  
  You can check that stage parameter has been substituted with Stage1 by printing the result:

**v** [True]**p** [!(ConfigVariable "target-platform-full" "powerpc-unknown-linux" \/ ConfigVariable "target-platform-full" "x86\_64-unknown-mingw32" \/ ConfigVariable "target-platform-full" "i386-unknown-mingw32" \/ ConfigVariable "solaris-broken-shld" "YES" /\ ConfigVariable "target-platform-full" "i386-unknown-solaris2")]**dyn**

As expected, condition **!StageVariable 0** has been replaced with **True**.

* We can **resolve** parameters, which depend on the environment.  
  resolve targetWays :: Action Ways   
  This Shake action looks up configuration files and substitutes the corresponding variables in the targetWays expression:

**v** [!StageVariable 0]**p** [!(False \/ True \/ False \/ False /\ False)]**dyn**

Importantly, whenever the corresponding values of the configuration flags change, the build rule that called the resolve function will be rerun (although, one might argue that if any of the **False** values in the above predicate change we shouldn’t initiate the rebuild, since the resulting value is the same – this is an opportunity for further optimisation).

Note that **project** and **resolve** do not perform any simplification of the resulting expression; they only evaluate some of the parameters. To simplify the result, use **simplify** :: Ways -> Ways. For example, by simplifying the result of applying both **project Stage1** and **resolve** to targetWays, we get **v p**.

**Extracting predicates**

Given targetWays how do we determine under which conditions a particular way belongs to the collection? This can be done by function **whenExists** :: v -> BuildExpression v -> BuildPredicate. Note that Ways is just a type synonym for BuildExpression Way.

whenExists vanilla targetWays == true  
whenExists profiling targetWays == notStage Stage0  
whenExists dynamic targetWays == platformSupportsSharedLibs

For example, whenExists profiling targetWays ? arg "--enable-library-profiling" is used in ghc-cabal settings.

**Build predicates**

The following basic predicates can be used in build expressions:

* stage :: Stage -> BuildPredicate  
  Evaluates to true if the current build stage matches the given stage
* package :: Package -> BuildPredicate  
  Evaluates to true if the package that is currently being built matches the given package
* builder :: Builder -> BuildPredicate  
  Evaluates to true if the builder that is currently run matches the given builder
* way :: Way -> BuildPredicate  
  Evaluates to true if the current build way matches the given way
* file :: FilePattern -> BuildPredicate  
  Evaluates to true if the file that is currently being processed matches the given file pattern
* config :: String -> String -> BuildPredicate  
  Evaluates to true if configuration files contain a value for the given key matching the given value

**Combining build expressions**

Use methods provided by the Alternative class:

* **empty** stands for the empty expression
* **<|>** combines two build expressions together
* **msum** folds a list of expressions

Additionally, use **|>** when the order of expressions matters, or **mproduct** to fold a list of ordered expressions.