1 INTRODUCTION

ffur compiler does not yet
the environment of the compi

ffe supply many ffaskell
$ffffffffffRprogs contains ma
contains the version of the pr
contains a number of useful l
$ffffffffffRdocQ

ffhe one incompatibility b
monadic ffRff system we use
the features are not yet impl
officially adoptedQ ffe provide
TQUO but stream ffRff is not

In addition to program execution, the various interfaces provide the following basic services:

- Compilation of Haskell programs and interfaces. Two basic compilation modes are used: fast compilation to interpreted Lisp code for immediate execution, and slower compilation to native code which is both written to output files and available for immediate execution. These are described as

WQ ffhe compilation unit being edited is loaded into the ffaskell system if necessaryQ

XQ ffhe expression is evaluated or run in the context of the module containing the cursor there may be more than one module in a fileLQ

YQ ffhe MhaskellM buffer pops up to display the resultQ

ffs well as receiving diagnostic and error messages and ffRff operations on stdin and stdoutO the MhaskellM buffer acts as a listener window for the ffaskell command interpreterQ ffou can interact directly with the command interpreter in this buffer if you chooseQ ffee section X for more information about the command interfaceQ

ffhe mode line for the MhaskellM buffer includes information about the status of the inferior ffaskell processQ ffhe values areff ready the process is waiting for a commandLO busy the process is busy processing a commandLO and input the process is waiting for input on stdinLQ ffote thatO since the ffaskell process runs asynchronouslyO you can do other things in ffmacs and queue up more commands to be executed even when the ffaskell process is busyQ

fl.fl flvaluation flontext

ffhe c

haskellPload ffPc lL

ffhis command loads the compilation unit currently being edited into the ffaskell processO without evaluating any of its contentsQ ffhis is used to check for compilation errorsQ

haskellPcompile ffPc cL

ffhis is similar to haskellPloadO but also compiles the ffisp code generated by the ffale ffaskell compilerQ ffince this is fairly slowO you probably don't want to do this frequently during program developmentQ ffommands such as ffPc e and ffPc r do not compile the code generated or write any compiled code into filesQ ffhen you are ready to create fully compiled code and save it in output filesO use the ffPc c commandQ ffhis command recursively compiles all imported compilation unitsQ

haskellPrunPfile ffPc ffPrL

ffoads the current compilation unit and then runs the dialogue named mainQ

haskellPgetPpad ffPc pL

ffops up the buffer containing the scratch pad for the current moduleO and makes it the current bufferQ

haskellPprinters ffPc ffPpL

ffhis command pops up a buffer that lets you set ffaskell compiler print options from a menuQ ffse ff to get further helpQ

ffou can specify printer settings for a particular program within its unit fileQ ffee section YQ

haskellPoptimizers ffPc ffPoL

ffhis command pops up a buffer that lets you set ffaskell compiler optimizer settings from a menuQ ffse ff to get further helpQ

ffou can specify optimizer settings for a particular program within its unit fileQ ffee section YQ

haskellPexit ffPc qL

ffhis command terminates the ffaskell subprocessQ fft leaves all buffers openQ

haskellPswitch ffPc hL

ffops up the MhaskellM buffer and makes it the current bufferQ

haskellPkill ffPc ffPkL

ffelete the current contents of the MhaskellM bufferQ

haskellPinterrupt ffPc iL

ffhis command sends an interrupt to the ffaskell subprocessQ ffou can use it to terminate execution of a running program for exampleO if it gets stuck in an infinite loopL and return to the command loopQ

haskellPcommand ffPc ffL

ffou can submit commands directly to the command interface with this commandQ ffee section X for more information about the command interfaceQ

haskell

fftarts up a ffaskell subprocessO popping up the MhaskellM bufferQ ffou normally don't need to do this explicitly since the default behavior for any of the commands that cause code to be evaluated or compiled is to start up a ffaskell subprocess if there isn't already one runningQ

haskellPmode

ffuts the current buffer into the ffaskell editing modeQ

haskellPtutorial
    fftarts the online ffaskell tutorialQ

ffn additionO the minibuffer commands ffPp and ffPn allow you to search back and forth through the history of the last VS expressions you have typed in when ffaskell prompts you for something to evaluateQ

fl.fl flustomization flooks

ffou can customize some aspects of the ffmacs ffaskell interface by changing the values of these variables eQgQO in your QemacsLQ

haskellPprogramPname
    ffhis is the name of the executable that is run as the inferior ffaskell processQ fft defaults to the value of the environment variable ffffffffffffffffffQ

haskellPautoPcreatePprocess
    fff this variable is true the defaultLO then any of the ffaskell commands that interact with the inferior ffaskell process automatically start up such a process if one doesn't already existQ fff falseO all of these commands just refuse to do anything usefulQ

haskellPautoPswitchPinput
    fff this variable is true the defaultLO then ffmacs automatically pops up the MhaskellM buffer and makes it the current buffer whenever the inferior ffaskell process issues a request to read from stdinQ ffemember that input requests arrive asynchronously!

haskellPaskPbeforePsaving
    fff this variable is true the defaultLO then ffmacs explicitly asks you if you want to save modified buffers containing ffaskell source files before evaluating or running anythingQ fff it's falseO all of these buffers are simply saved quietlyQ

haskellPinitialPprinters
    ffhis variable contains the printers to set when starting a new ffaskell processO specified as a list of stringsQ ffome people prefer different default printers for the ffmacs interface than when running ffaskell using the vanilla command interfaceQL

4 .sing Haskell on the .acintosh

ffale ffaskell runs on the ffacintosh under ffacintosh ffommon fflisp ffffflLQ ffhe default user interPface on the ffacintosh is layered on top of ffffff's builtPin ffmacsPlike editorO ffredQ

ffhis section assumes that you have some familiarity with the ffacintosh eQgQO that you know how to manipulate windows and about the standard ffacintosh text editing facilities and file selection dialogsLQ ffhe items on the ffile and ffdit menus are fairly standard in this respectQ

fl.1 flnteraction with the flaskell Process

ffhere are two basic editor commands for interacting with ffaskellff evaluating and printing an expression ffPc eL and running a dialogue ffPc rLQ ffhey operate as followsff

TQ ffou are prompted for a ffaskell expressionQ ffor ffPc eO the value of the expression must be in class ffextQ ffor ffPc rO the value must be of type ffffLQ

UQ ffll buffers containing ffaskell source files are savedQ

VQ ffhe compilation unit being edited is loaded into the ffaskell system if necessaryQ

WQ ffhe expression is evaluated or run in the context of the module containing the cursor there may be more than one module in a fileLQ

XQ ffhe ffistener window pops up to display the resultQ

ffou can interact with ffaskell by issuing ffred commandsO or by selecting commands from the ffaskell menuQ ffhe basic ffaskell command interpreterO described in section XO runs in the ffistener windowQ ffhis window also receives ffRff operations on stdin and stdout and is used to display various status and diagnostic messagesQ

ffhe mode line for the ffistener window includes information about the status of ffaskell's command loopQ ffhe values areff ready the process is waiting for a commandLO busy the process is busy processing a commandLO and input the process is waiting for input on stdinLQ ffote thatO since the ffaskell command loop process runs asynchronouslyO you can do other things in the editor and queue up more commands to be executed even when the ffaskell process is busyQ

fl.fl flvaluation flontext

ffhe c

ffoad ffile ffPc lL
: ffhis command loads the compilation unit currently being edited into ffaskellO without evaluPating any of its contentsQ ffhis is used to check for compilation errorsQ

ffun ffile ffPc ffPrL
: ffoads the current compilation unit and then runs the dialogue named mainQ

ffompile ffile ffPc cL
: ffhis is similar to ffoad ffileO but also compiles the ffisp code generated by the ffale ffaskell compilerQ ffince this is fairly slowO you probably don't want to do this during program dePvelopmentQ ffommands such as ffPc e and ffPc r do not compile the code generated or write any compiled code into filesQ ffhen you are ready to create fully compiled code and save it in output filesO use the ffPc c commandQ ffhis command recursively compiles all imported compilation unitsQ

ffcratch ffad ffPc pL
: ffops up the buffer containing the scratch pad for the current moduleO and makes it the current bufferQ

ffrinters ffPc ffPpL
: ffhis command pops up a dialog box that lets you set ffaskell compiler print optionsQ

  ffou can specify printer settings for a particular program within its unit fileQ ffee section YQ

ffptimizers ffPc ffPoL
: ffhis command pops up a dialog box that lets you set ffaskell compiler optimizer settingsQ

  ffou can specify optimizer settings for a particular program within its unit fileQ ffee section YQ

ffutorial
: ffhis loads the onPline ffaskell tutorialQ ffhe tutorial is split into pagesO each of which contains some small example programs that you can runQ ffithin the tutorial windowO you can use two additional editor commandsff ffPc ffPf to go forward to the next page of the tutorialO and ffPc ffPb to go backward to the previous pageQ

ffbort ffPc iL
: ffhis command interrupts ffaskell and causes it to return to the command loopQ ffou can use it to terminate execution of a running program for exampleO if it gets stuck in an infinite loopL and return to the command loopQ

ffote that you can get a complete list of ffred key bindings by doing ffPffQ

fl.fl flustomization flooks

ffou can customize some aspects of the ffaskell user interface by changing the value of these variables for exampleO in your initialization fileLQ

MmacPautoPswitchPinputM
: fff this variable is true the defaultLO then ffaskell automatically pops up the ffistener window and selects it as the active window whenever ffaskell issues a request to read from stdinQ

MmacPaskPbeforePsavingM
: fff this variable is true the defaultLO then ffaskell explicitly asks you if you want to save modified buffers before evaluating or running anythingQ

5 .he Command Interface

ffhen you run the ffaskell executable $ffffffffffffffffffff directlyO it puts you into a command loop that allows you to compileO loadO and run programsQ ffhis section gives brief descriptions of all commands recognized by the command interfaceQ

ffhe command interface maintains a single scratch pad for ffaskell source codeQ ffny line starting with ff is assumed to be a commandQ ffther lines are added to the current contents of the scratch padQ ffhere is only one pad and it is associated with the current module that the system has focused onO as selected by the ffmodule command or loading a ffaskell programQ ffhis module name is used as the promptQ ffhenever an error is encountered during compilation of the padO the pad reverts to whatever value last compiled successfullyQ ffhe command interface scratch pad is not intended for serious program development but rather serves as a place to easily save a few definitions while debugging a programQ

ffommand arguments are separated by whitespaceQ ffll commands may be abbreviated using the shortest unique prefix of the command nameQ ffere are brief descriptions of the commandsQ

ffff `e`

ffinput ≰

ffhen a module is brought into a unit through only a Qhi fileO no cross module optimization can be performed on references to values in the imported moduleQ ffn the other handO when the implementation corresponding to the interface changesO this does not require recompilation unless the Qhi file is also changedQ ffhis also allows mutually recursive modules to be separately compiledO something that cannot be done without Qhi filesQ

ffhen both the implementation and user supplied interface for a module are part of a compilation unitO the system checks for agreement between the two interfaces and notifies you of any inconsisP tencies between the module and its interfaceQ ffny program which uses Qhi files should include the Qhi interface in any unit where the corresponding implementations are definedQ

ffn practiceO you may want to start out by using Qhi files to define the interfaces between compiP lation units as you are writing and debugging your programO and switch to importing Qhu files once the code has stabilized to increase the amount of cross module optimizationQ

ffere is an example showing how you can set up your Qhu files for each of these approachesQ ffuppose you have a program p that consists of two source files a and b that contain mutually recursive referencesO and a library l that is used by both a and bQ

ffhen debugging your codeO you could begin by creating a Qhi file for each of aO bO and lO and structuring the Qhu files like thisff

```
pQhuff
    lQhu
    aQhu
    bQhu

lQhuff
    lQhs
    lQhi

aQhuff
    aQhs
    aQhi
    bQhi
    lQhi

bQhuff
    bQhs
    bQhi
    aQhi
    lQhi
```

ffhenO once you are no longer making changes to the code and want to get it to compile more efficientlyO you can restructure the Qhu files like thisff

```
pQhuff
    lQhu
    aQhs
    aQhi
    bQhs
    bQhi
```

lQhuff
    lQhs
    lQhi

ffote thatO in both casesO we have been careful to include the Qhi files in the same compilation unit that contains the corresponding implementationsO so that the compiler can automatically detect inconsistencies between the twoQ

fl.fl flompilation Units for floreign flunctions

ffnterface files that define an interface to code written in ffispO ffO or other languages are different from ordinary interface files in that they generate some amount of executable code to perform conversions on function arguments and resultsLQ ffou f ffff ffoo a Pff L

will allow the recursive call to be polymorphic and avoid a type errorQ ffn explicit type signature is required to make use of polymorphic recursionQ

ffhe ffaskell TQV committee is seriously considering dropping N patterns from the languageQ ffhese are still in the ffale system but should be avoidedQ

8 Annotations

ffhe ffale compiler recognizes a number of annotations in the source programQ ffnnotations are used for strictness propertiesO optimizer controlO and the interface to the ffisp environmentQ

fl.1 flnnotation Syntax

ffnnotations are a special form of commentsQ ffhey begin with ffPff and end with ffP based on interface files which describe ffisp code instead of ffaskell functionsQ ffull details are found in a technical report included in the standard distribution of the ffale systemQ

10 Dynamic .yping

ffale ffaskell has been extended with an experimental dynamic typing systemQ ffynamic typing has been used to implement derived instances in our compilerQ ffome minor extensions to the language have been madeO namely the use of patterns in the form ffrue N ffalse

is good coding practice and eases debuggingQ fft is also good coding practice to avoid nonPdisjoint patterns whenever possible such patterns also tend to translate into less efficient code than patterns written disjointlyLQ

results in a message indicating that ffloor is not in class ffumQ ffuch messages usually result from ordinary type mismatchesQ

ffsing ffnt arithmetic can lead to serious runtime errors when overflows occurQ ffnly unsafe ffnt operations are currently providedQ fff you run into memory protection problemsO illegal instrucPtionsO or other mysterious errors when the program executesO check for potential overflow problemsQ ffhanging ffnt to ffnteger should fix this problemQ

ffnother source of subtle type errors is overloaded numeric constantsQ ffn integer constant has the typing ffum a fff aQ ffhen integers are used unexpectedlyO the type unification algorithm complains that the parameter type is not in class ffumQ ffor exampleO inff

length V

fff you think your program is in an infinite loopO use ffPc i from the editor interface or ffff from the command interface to interrupt the ffaskell processQ

the type error generated indicates that lists e case statements and function definitionsff this

11.fl isp flrrors

ffugs in the ffale ffaskell implementation may cause an error to be reported by the underlying ffisp systemQ ffisp errors may also occur as a result of executing code you have loaded into ffaskell through the foreignPfunction interfaceQ fff this happensO there is a handler in place that normally causes control to be returned to the main command loopO rather than entering the ffisp debuggerQ

fff you want to be able to mess around with the ffisp debuggerO you should issue the ffdebug on commandO or put this in your ffaskell initialization fileff

```
setf MhaskellPdebugPinPlispM 'fftL
```

12 .erformance Issues

ffany different factors affect the speed of the compiled ffaskell code and the compilation timeQ ffhis section deals with both of these issuesQ

1fl.1 flenerating flaster flode

ffhe code executed under the ffaskell system may be either compiled using ffcompile or ffPc cL or interpretedQ fff your program is running too slowO ensure that all components are compiled firstQ

ffsing strict structures often greatly speeds program executionQ ffhile care must be taken to avoid losing laziness that a program depends onO attaching strictness annotations to your data declarations can have a dramatic effect of execution time and memory usageQ

ffrithmetic operations can often be improved by paying careful attention to the type systemQ ffirstO ffnt arithmetic is much faster than ffnteger arithmeticQ ffomplex and rational numbers are not implemented in a particularly efficient mannerQ ffrithmetic primitives implemented by the prelude instead of imported as primitives are often very slowQ ffverloaded arithmetic operators are particularly slowQ ffou should use type signatures or the ffpecialize annotation to avoid calling overloaded functions at run timeQ

ffther aspects of the performance reflect the ffaskell compiler itselfQ ffhe compiler performs many optimizationsO including strictness analysisO inliningO function specializationO deforestationO and othersQ ffou may wish to look at the code generated by the compiler to observe the results of these optimizationsQ ffhe code generator is able to perform many more optimizations on firstPorder function calls than on higher order callsQ ffhe ffnline declaration and optimizer can help turn higher order calls into first order onesQ

fff you have functions that iterate over listsO try to write them using primitives such as foldr and map so that you can take advantage of the deforestation optimizations performed by the compilerQ

1fl.fl flvoiding Stack Overflow

fff you have problems with stack overflowO you should first check for infinite loops in your programQ fff the overflow is not caused by an infinite loopO you have few optionsQ

fff you have built ffale ffaskell from sources using fffff ffispO you are likely to run into problems because the stack size in the standard fffff distribution is really too small for running ffaskellQ ffhanging the stack size involves rebuilding fffff ffisp from sourcesQ ffe have already done this for the precompiled fffffPbased ffaskell system we distributeQ

ffther ffisp systems have various procedures for allocating larger stacksQ ffou will have to look into the documentation of your ffisp system to find out what to doQ

1fl.fl Optimizer Options

ffy defaultO compiling code to a file makes use of all optimizations possibleQ ffnless you need to improve compilation timeO changing the optimization settings should not be necessaryQ ffere is a list of the various optimizers and what they doQ

inline fff this option is enabledO the optimizer marks functions that have “trivial” definitions as inlinableO and also inlines references to both variables and functions that have already been noted as inlinableQ fff it’s disabledO the optimizer is much more conservative about inliningQ

constant ffhis option controls whether the optimizer attempts to detect function applications that evaluate to constant valuesO specifically applications of data constructors to constant arguPmentsQ fff enabledO the optimizer hoists such applications to top level to avoid repeated evaluPationsQ

foldr fff this optimization is enabledO

```
setenv ffffffffffff ffdirectory containing ffaskellff
source $ffffffffffffRhaskellPsetup
```

ffemember to reload your Qcshrc before you run ffaskell for the first timeQ

ffsers of the ffmacs ffaskell mode need to make a few changed to their Qemacs fileQ

- ffdd the ffaskell ffmacs tools directory to the search pathff

```
   setq loadPpath
         cons concat getenv ”ffffffffffff”L ”RemacsPtools”L
               loadPpathLL
```

- ffhange the value of the ffmacs variable autoPmodePalistff

```
   setq autoPmodePalist
         append ”““Qhs$” Q haskellPmodeL ”““Qlhs$” Q haskellPmodeL
                 autoPmodePalistLL
```

  ffhis causes ffaskell mode to be selected whenever a ffaskell source file is editedQ

- ffake ffmacs load the ffaskell mode from the startff

```
   loadPlibrary ”haskell”L
```

1fl.fl ffnit fliles

fft is possible to customize the ffaskell environment using init files which are read to the ffaskell system at startupQ ffhese can be used to control the underlying ffisp environmentQ ffhe files $ffffffffffffRQyhaskell and

1fl.fl flnstalling Yale flaskell on the Mac

ffhe ffacintosh version of ffale ffaskell is distributed in a selfPextracting ffompact ffro archiveff just doublePclick on it to extract the filesQ ffhis archive contains both the full source distribution and a readyPtoPrun executableQ ffou can also rebuild ffaskell on the ffacintosh from scratch using the normal tarPformat ffnix source distributionQ

ffpple's licensing restrictions prevent us from distributing an executable that contains the fffff compilerQ ffhis means thatO in the executable we provideO the ffompile ffile menu command signals an error instead of doing anything usefulQ fff you want to be able to compile ffaskell programs to object codeO you must buy a copy of fffff and rebuild ffale ffaskell from the sourcesQ ffee the file $ffffffffffffRcomRmclRffffffffff for instructions on how to do thisQ

ffe have set the default size of the executable we distribute so that it will run comfortably on a ffacintosh with TUffff of memoryQ fff your machine has less memoryO we recommend that you enable virtual memory rather than trying to make ffaskell run in a smaller partitionQ ffou can change the amount of memory ffaskell uses from the finder using the ffet ffnfo menu itemff virtual memory is enabled from the ffemory control panelQ ffake sure you also enable VUPbit addressing from the ffemory control panelQL

ffebuilding ffaskell from sources requires somewhat more memoryff we have been installing fffff in a USffff partition to do thisQ

ffhe ffacintosh version of ffale ffaskell does

.eferences

[T] ffhuck ffonselQ ffast strictness analysis via symbolic fixpoint iterationQ ffechnical ffeport ffffffffRffffffRffffPffYffO ffale ffniversity ffepartment of ffomputer ffcienceO ffeptember TffffTQ

[U] ffndrew ffillO ffohn ffaunchburyO and ffimon ff ffeyton ffonesQ ff short cut to deforestationQ ffn *ff*