1 Making BUGS Open

BUGS is a long running software project aiming to make modern MCMC techniques based on graphical models available to applied statisticians in an easy to use package. This talk will give an overview of the structure of OpenBUGS the open source version of the BUGS software and the tools used in its creation and maintenance. Interfacing BUGS to R will also be discussed in particular the possibilities for closer coupling than currently available in the BRugs package.

2 Adopt a module

Unique once in a life time opportunity. You can choose which OpenBUGS module you would like to care for. Adopt a module and you can take a precious piece of software home with you to read and play with! Hundreds of modules to choose from but many thousands of BUGS users. Hurry while stocks last!

3 Happy modules

Would an OpenBUGS module be happy with you? Some questions to ask yourself: what is a module? what is a class? what is an object? what is a factory object? what is an interface? what is the difference between client and extension interfaces? why are concrete classes hidden? what is metaprogramming? what do I do with blue diamonds? why are trap messages helpful? what is the Hollywood principle of programming? can I program in C?

4 Myth I

BUGS is one big scary monster

Reality

BUGS is lots of friendly little bits
Myth II

BUGS is written in a strange complicated language

Reality

Component Pascal is a very simple powerful language

Myth III

BUGS uses strange development tools

Reality

BlackBox tools are very simple to use (they are also free...)

Myth IV

Open source software must be developed in C/C++ using GNU tools

Reality

Open source software must be developed in C/C++ using GNU tools

Myth V

Software technology has not changed in the past 20 years

Reality

There is Microsoft, there is Linux, there is the Intel based PC (and the Mac...)
Create lots of objects, wire them together and then get the objects to talk to each other.

Need a plan of how to do this

Bayesian graphical models
The type of plan BUGS understands is called a bayesian graphical model.

Bayesian graphical models describe conditional independence assumptions

Give factorization of joint probability distribution

Graphs as (formal) language

```
model
{
for( i in 1 : N ) {
for( j in 1 : T ) {
    Y[i, j] ~ dnorm(mu[i, j], tau.c)
}
    alpha[i] ~ dnorm(alpha.c, alpha.tau)
    beta[i] ~ dnorm(beta.c, beta.tau)
}
}
tau.c ~ dgamma(0.001, 0.001)
sigma <- 1 / sqrt(tau.c)
alpha.c ~ dnorm(0.0, 1.0E-6)
alpha.tau ~ dgamma(0.001, 0.001)
beta.c ~ dnorm(0.0, 1.0E-6)
```
13 **Compilation**

Turning description in one language into equivalent description in another language

Can add extra information

New description can be executable

14 **An analogy**

Science program written in Fortran

Compiled to assembly language

Assembly language - low level instruction that cause the CPU to do things

15 **An analogy continued**

Statistical model written as a graph

Compiled into inference algorithm

Inference algorithm executed

16 **Inference algorithms**

Many possibilities

Want good natured algorithm not fussy about what it is asked to do.

MCMC simulation good choice
MCMC simulation

Generate lots of random numbers
After a bit calculate averages of these random numbers
Also can calculate quartiles, kernel densities etc.

MCMC simulation continued

<table>
<thead>
<tr>
<th>iteration</th>
<th>alpha.Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.5</td>
</tr>
<tr>
<td>5000</td>
<td>0.5</td>
</tr>
<tr>
<td>10000</td>
<td>1.0</td>
</tr>
<tr>
<td>15000</td>
<td>1.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mean</th>
<th>sd</th>
<th>MC_error</th>
<th>2.5pc</th>
<th>median</th>
<th>97.5pc</th>
<th>start</th>
<th>sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8934</td>
<td>0.1388</td>
<td>0.006738</td>
<td>0.6265</td>
<td>0.8891</td>
<td>1.173</td>
<td>1001</td>
<td>10000</td>
</tr>
</tbody>
</table>

The BUGS software

BUGS has to do lots of tasks.
This does not make BUGS one big scary monster.
BUGS contains subsystems to perform specific tasks
Each subsystem consists of a number of modules
Each module is small and easy to understand

Tasks and subsystems

Describing graphical models
Doodle subsystem
Compiling graphical model
Bugs subsystem
Probability calculations
Graph subsystem
MCMC simulation
Updater subsystem
Summary statistics
Samples, Summary, Ranks, Deviance subsystems
Subsystems and modules I

Subsystems can consist of a small number of modules or a large number.

Samples subsystem is small

Subsystems and modules II

BUGS consists of 15 subsystems
   5 Windows only subsystems

BUGS consists of 263 modules
   63 windows only modules

BUGS is a small system

Graph subsystem is large

GraphRules GraphNodes GraphLogical GraphStochastic
GraphScalar GraphVector GraphUnivariate
GraphMultivariate GraphConjugateMV GraphChain
GraphConstant GraphSimple GraphStack GraphMixture
GraphFlat GraphGeneric GraphBlock
GraphCopulaGraphFunction GraphIvevent GraphGammap
GraphIvevent GraphInverse GraphHuslerGraphIvevent
GraphLogdet GraphLogit GraphPoisson GraphProduct
GraphRanks GraphPValue GraphSum GraphTable
GraphFunctional GraphODEmath GraphODElang
GraphPredictive GraphBern GraphNormal GraphCat
GraphUniform GraphGEV
GraphHypergeometric GraphMandel GraphNegbin
GraphPoisson GraphRasch GraphMultinom
GraphBeta GraphChiGraphIvevent GraphExp GraphF
GraphGamma GraphGengamma GraphLogistic
GraphLognorm GraphNormal GraphRangeto
GraphMarginal GraphNormal GraphPivot
GraphRfree GraphRfree GraphIvevent
GraphMVNormal GraphMVT GraphRENormal
GraphStochtrend GraphWishart

Styles of modules in BUGS

Many different styles of module in BUGS both within and between subsystems.

Example the BugsNames modules contains 78 statements, the BugsInterpreter module contains 20 statements

In general modules either implement algorithms or they establish concepts.
What is a module (in CP)

A module is a package of source code with a well defined interface
A module is a unit of compilation
A module is a unit of loading
A module knows what services it provides (syntactically)
A module can make "use" of other modules
Under the "use" relation modules are arranged in a DAG

Languages other than CP

Weird languages have weird ideas (they just happened)
Header files
Include
Name spaces
Only classes
main{}
Dynamic link libraries

Software development tools

These tools are never free
These tools are often very complex
These tools often do tasks that are not needed (or should not be needed)
Best to use commonly used tools (it's nice to be in a crowd)

Object oriented software I

Objects are the easy bit
Designing the classes is the hard bit
(Code) Inheritance is evil
Methods should not be extended
Composition is good
Object orientated software II

Large class hierarchies can be a nightmare
Multiple inheritance deepens that nightmare
Need tools that control the complexity of the hierarchy
Need IDL to describe software

IDL Interface Definition Language

DEFINITION GraphNodes;
TYPE
Factory = POINTER TO ABSTRACT RECORD
(f: Factory) New (option: INTEGER) Node, NEW, ABSTRACT
END;

List = POINTER TO ABSTRACT RECORD
node: Node,
next: List
END;

Node = POINTER TO ABSTRACT RECORD
props: SET,
(node: Node) AddParent (VAR list: List), NEW;
(node: Node) Check (): SET, NEW, ABSTRACT;
(node: Node) Children (): List, NEW, ABSTRACT;
(node: Node) Factory (f: Factory): Factory, NEW, ABSTRACT;
(node: Node) Set (IN args: Args; OUT res: SET), NEW, ABSTRACT;
(node: Node) Value (): REAL, NEW, ABSTRACT
END;

Vector = POINTER TO ARRAY OF Node;

PROCEDURE SetFactory (f: Factory);
END GraphNodes.

More IDL

DEFINITION GraphLogical;
IMPORT GraphNodes;

TYPE
List = POINTER TO ABSTRACT RECORD
node-: Node;
next-: List
END;

Node = POINTER TO ABSTRACT RECORD
(GraphNodes.Node)
level-: INTEGER;
parents-: List;
(node: Node) AddToList (VAR list: List), NEW;
(node: Node) CalculateLevel, NEW;
(node: Node) ClassFunction (parent: GraphNodes.Node): INTEGER, NEW,
ABSTRACT;
(node: Node) ClearLevel, NEW;
(node: Node) HandleMsg (msg: INTEGER), NEW, EMPTY;
(node: Node) Optimize (parent: GraphNodes.Node), NEW, EMPTY;
(node: Node) StoreParents, NEW
END;

Vector = POINTER TO ARRAY OF Node;

Arg = ABSTRACT RECORD
node-: Node,
numConsts, numOps, numScalars, numVectors: INTEGER;
consts: ARRAY 50 OF REAL;
scalars: ARRAY 50 OF GraphNodes.Node;
ops: ARRAY 100 OF INTEGER;

PROCEDURE Ancestors (node: GraphNodes.Node): List;
END GraphLogical.

Metaprogramming

Self awareness for software
Software can ask itself questions
Is there an item called FooBar?
What sort of item is FooBar?
Do this with item FooBar
Metaprogramming cont

Can ask if a module of given name is loaded and if so to return its metadata

Can ask to load a module of given name and return its metadata

Can ask a module’s metadata if it contains an item with a given name and type

Modules metadata limited to what is in the modules interface

---

Metaprogramming example

DEFINITION GraphT;

IMPORT GraphNodes;

VAR fact-: GraphNodes.Factory;

PROCEDURE Install;

END GraphT.

Can ask to load module GraphT and return its metadata

Can ask metadata if there is a procedure called Install (with signature no arguments)

Can ask to execute the Install procedure

---

Metaprogramming and BUGS

Used in many places

To support the BUGS language

To load sampling algorithms

To load data reading algorithms

To construct GUI interface

To implement scripting

To interface to R

---

BUGS language support

Grammar file (snippet)

<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbern</td>
<td>&quot;GraphBern.Install&quot;</td>
</tr>
<tr>
<td>dbeta</td>
<td>&quot;GraphBeta.Install&quot;</td>
</tr>
<tr>
<td>dbin</td>
<td>&quot;GraphBinomial.Install&quot;</td>
</tr>
<tr>
<td>dcat</td>
<td>&quot;GraphCat.Install&quot;</td>
</tr>
<tr>
<td>dchisqr</td>
<td>&quot;GraphChisqr.Install&quot;</td>
</tr>
<tr>
<td>dexp</td>
<td>&quot;GraphDbexp.Install&quot;</td>
</tr>
<tr>
<td>dnorm</td>
<td>&quot;GraphNormal.Install&quot;</td>
</tr>
<tr>
<td>dt</td>
<td>&quot;GraphT.Install&quot;</td>
</tr>
</tbody>
</table>

---
### GUI and BUGS

```pascal
Definition BugsCmds;

TYPE
  UpdateDialog = POINTER TO RECORD
    iteration-, refresh, thin, updates: INTEGER;
    isAdapting-, overRelax: BOOLEAN
  END;

VAR updateDialog: UpdateDialog;

PROCEDURE Update;
  PROCEDURE UpdateGuardWin (VAR par: Dialog.Par);

modelCheck(^0) --->
  "BugsCmds.SetFilePath('^0'); BugsCmds.ParseGuard;
  BugsCmds.ParseFile"

modelUpdate(^0) -->
  "BugsCmds.updateDialog.updates := ^0;
  BugsCmds.UpdateGuard; BugsCmds.Update"

samplesStats(^0) -->
  "SamplesCmds.SetVariable('^0'); SamplesCmds.StatsGuard;
  SamplesCmds.Stats"
```

### Scripting and BUGS

```plaintext
R and BUGS makes BRugs

- R has a nice paste function
- R can talk to dynamic link libraries
- Dynamic link library can use metaprogramming to talk to BUGS
- Few technical problems: heap management most difficult
- BRugs package up on CRAN thanks to Uwe and Sibyle
```

### R and BUGS's Guts

```plaintext
New OpenBUGS module BugsRobjects to help R see and interact with OpenBUGS internal data structures

- Can set node values
- Can look at node values
- Can program higher level algorithms in R
- Can load data / inits from R objects (no need for textual representation)
```
Adopt a module (please)

BUGS has become OpenBUGS

Open source software needs active developers

OpenBUGS is small and friendly
(just don’t mention C)

You can make a difference --
take a module home today!