Fleet

0.0.9

Generated by Doxygen 1.8.13

Contents

Chapter 1

Fleet - Fast inference in the Language of Thought

1.1 Introduction

Fleet is a C++ library for programming language of thought models. In these models, you will typically specify a grammar of primitive operations which can be composed to form complex hypotheses. These hypotheses are best thought of as programs in a mental programming language, and the job of learners is to observe data (typically inputs and outputs of programs) and infer the most likely program to have generated the outputs from the inputs. This is accomplished in Fleet by using a fully-Bayesian setup, with a prior over programs typically defined thought a Probabilistic Context-Free Grammar (PCFG) and a likelihood model that typically says that the output of a program is observed with some noise.

Fleet is most similar to LOTlib (`https://github.com/piantado/LOTlib3`) but is considerably faster. L↩
OTlib converts grammar productions into python expressions which are then evaled in python; this process is flexible and powerful, but very slow. Fleet avoids this by implementing a lightweight stack-based virtual machine in which programs can be directly evaluated. This is especially advantageous when evaluating stochastic hypotheses (e.g. those using flip() or sample()) in which multiple execution paths must be evaluated. Fleet stores these multiple execution traces of a single program in a priority queue (sorted by probability) and allows you to rapidly explore the space of execution traces.

To accomplish this, Fleet makes heavy use of C++ template metaprogramming. It requires strongly-typed functions and requires you to specify the macro FLEET_GRAMMAR_TYPES in order to tell its virtual machine what kinds of variables must be stored. In addition, Fleet uses a std::tuple named PRIMITIVES in order to help define the grammar. This tuple consists of a collection of Primitive objects, essentially just lambda functions and weights). The input/output types of these primitives are automatically deduced from the lambdas (using templates) and the corresponding functions are added to the grammar. Note that the details of this mechanism may change in future versions in order to make it easier to add grammar types in other ways. In addition, Fleet has a number of built-in operations, which do special things to the virtual machine (including Builtin::Flip, which stores multiple execution traces; Builtin::If which uses short-circuit evaluation; Builtin::Recurse, which handles recursives hypotheses; and Builtin::X which provides the argument to the expression). These are not currently well documented but should be soon. ∗

1.2 Installation

Fleet is based on header-files, and requires no additional dependencies (command line arguments are processed in CL11.hpp, which is included in src/dependencies/).

The easiest way to begin using Fleet is to modify one of the examples. For simple rational-rules style inference, try Models/RationalRules; for an example using stochastic operations, try Models/FormalLanguageTheory-Simple.

Fleet is developed using GCC.

1.3 Installation

Fleet provides a number of simple inference routines to use.

1.3.1 Markov-Chain Monte-Carlo

1.3.2 Search (Monte-Carlo Tree Search)

1.3.3 Enumeration

etc...

1.4 Installation

- Sample things, store in TopN, then evaluate...

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- Bayesable< _t_datum, _t_data > ??
- Bayesable< Args... > ??
  - MCMCable< HYP, Args > ??
  - MCMCable< GrammarHypothesis< HYP, t_datum, t_data >, t_datum, t_data > ??
    - GrammarHypothesis< HYP, t_datum, t_data > ??
  - MCMCable< HYP, _t_datum, _t_data > ??
    - LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data > ??
  - MCMCable< HYP, t_datum > ??
    - Lexicon< HYP, T, t_input, t_output, t_datum > ??
  - MCMCable< InnerHypothesis, default_datum< S, S >, std::vector< default_datum< S, S > > > ??
    - LOTHypothesis< InnerHypothesis, Node, S, S > ??
      - InnerHypothesis ??
  - MCMCable< MyHypothesis, default_datum< Object, bool >, std::vector< default_datum< Object, bool > > > ??
    - LOTHypothesis< MyHypothesis, Node, Object, bool > ??
      - MyHypothesis ??
  - MCMCable< MyHypothesis, default_datum< S, S > > ??
    - Lexicon< MyHypothesis, InnerHypothesis, S, S > ??
      - MyHypothesis ??
  - MCMCable< MyHypothesis, float, std::multiset< float > > ??
    - LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > > ??
      - MyHypothesis ??
- BuiltinPrimitive< t, args > ??
- BuiltinPrimitive< bool > ??
  - Builtin::Flip ??
- BuiltinPrimitive< bool, double > ??
  - Builtin::FlipP ??
- BuiltinPrimitive< t > ??
  - Builtin::X< t > ??
- BuiltinPrimitive< t, bool, t, t > ??
  - Builtin::If< t > ??
- BuiltinPrimitive< t_out, t_in > ??

- LOTHypothesis< MyHypothesis, Node, Object, bool > . . . . **??**
- Searchable< MyHypothesis, S, S > . . . . **??**
  - Lexicon< MyHypothesis, InnerHypothesis, S, S > . . . . **??**
- Fleet::Statistics::StreamingStatistics . . . . **??**
- t_null . . . . **??**
- VirtualMachineState< t_x, t_return >::t_stack< args > . . . . **??**
- VirtualMachineState< t_x, t_return >::t_stack< FLEET_GRAMMAR_TYPES > . . . . **??**
- Fleet::Statistics::TopN< T > . . . . **??**
- true_type
  - is_iterable< T, std::void_t< decltype(std::begin(std::declval< T >())), decltype(std::end(std::declval< T >())) > > . . . . **??**
- type
  - has_operator_lessthan< T, EqualTo > . . . . **??**
- TypeHead< Args > . . . . **??**
- vector
  - Stack< T > . . . . **??**
  - Stack< std::pair< index_t, t_x > > . . . . **??**
  - Stack< t_x > . . . . **??**
- VirtualMachinePool< t_x, t_return > . . . . **??**
- VirtualMachineState< t_x, t_return > . . . . **??**

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 Builtin Namespace Reference

Classes

- struct Flip
- struct FlipP
- struct If
- struct Recurse
- struct SafeMemRecurse
- struct SafeRecurse
- struct X

6.2 Fleet Namespace Reference

Namespaces

- applyVMS
- Statistics

Variables

- std::mutex output_lock

6.2.1 Variable Documentation

6.2.1.1 output_lock

```
std::mutex Fleet::output_lock
```

6.3 Fleet::applyVMS Namespace Reference

Functions

- template<int n, class T , typename V , typename P , typename L >
  vmstatus_t applyToVMS_one (T &p, V ∗vms, P ∗pool, L ∗loader)
- template<class T , typename V , typename P , typename L , size_t... Is>
  vmstatus_t applyToVMS (T &p, int index, V ∗vms, P ∗pool, L ∗loader, std::index_sequence< Is... >)

6.3.1 Function Documentation

6.3.1.1 applyToVMS()

```
template<class T , typename V , typename P , typename L , size_t...  Is>
vmstatus_t Fleet::applyVMS::applyToVMS (
             T & p,
             int index,
             V ∗ vms,
             P ∗ pool,
             L ∗ loader,
             std::index_sequence< Is...  >  )  [inline]
```

6.3.1.2 applyToVMS_one()

```
template<int n, class T , typename V , typename P , typename L >
vmstatus_t Fleet::applyVMS::applyToVMS_one (
             T & p,
             V ∗ vms,
             P ∗ pool,
             L ∗ loader )  [inline]
```

6.4 Fleet::Statistics Namespace Reference

Classes

- class FiniteHistory
- class MedianFAME
- class ReservoirSample
- class StreamingStatistics
- class TopN

Functions

- template<typename HYP >
  void operator<< (std::set< HYP > &s, TopN< HYP > &t)

6.4.1 Function Documentation

6.4.1.1 operator<<()

```
template<typename HYP >
void Fleet::Statistics::operator<< (
             std::set< HYP > & s,
             TopN< HYP > & t )
```

6.5 Proposals Namespace Reference

Functions

- double can_resample (const Node &n)
- std::pair< Node, double > prior_proposal (Grammar ∗grammar, const Node &from)
- std::pair< Node, double > regenerate (Grammar ∗grammar, const Node &from)
- std::pair< Node, double > insert_tree (Grammar ∗grammar, const Node &from)
- std::pair< Node, double > delete_tree (Grammar ∗grammar, const Node &from)

6.5.1 Function Documentation

6.5.1.1 can_resample()

```
double Proposals::can_resample (
             const Node & n )
```

Helper function for whether we can resample from a node (just accesses n.can_resample)

Parameters

n	- what node are we asking about?

Returns

- a double (1.0 or 0.0) depending on whether n can be sampled

6.5.1.2 delete_tree()

```
std::pair<Node, double> Proposals::delete_tree (
             Grammar * grammar,
             const Node & from )
```

backward is we choose the *new* s, then generate everything else, and choose anything equal

6.5.1.3 insert_tree()

```
std::pair<Node, double> Proposals::insert_tree (
            Grammar * grammar,
            const Node & from )
```

backward is we choose t exactly, then we pick anything below that is equal to s

6.5.1.4 prior_proposal()

```
std::pair<Node,double> Proposals::prior_proposal (
            Grammar * grammar,
            const Node & from )
```

6.5.1.5 regenerate()

```
std::pair<Node,double> Proposals::regenerate (
            Grammar * grammar,
            const Node & from )
```

Regenerate with a rational-rules (Goodman et al.) style regeneration proposal: pick a node uniformly and regenerate it from the grammar.

Parameters

grammar	- what grammar to use
from	- what node are we proposing from

Returns

A pair of the new proposed tree and the forward-backward log probability (for use in MCMC)

Chapter 7

Class Documentation

7.1 Bayesable$<$ _t_datum, _t_data $>$ Class Template Reference

```
#include <Bayesable.h>
```

Public Types

- typedef _t_datum t_datum
- typedef _t_data t_data

Public Member Functions

- Bayesable ()
- virtual void clear_bayes ()
- virtual double compute_prior ()=0

  Compute the prior – defaultly not defined.
- virtual double compute_single_likelihood (const t_datum &datum)=0

  Compute the likelihood of a single data point.
- virtual double compute_likelihood (const t_data &data, const double breakout=-infinity)

  Compute the likelihood of a collection of data, by calling compute_single_likelihood on each. This stops if our likelihood falls below breakout.
- virtual double compute_posterior (const t_data &data, const double breakout=-infinity)
- virtual double at_temperature (double t)
- virtual size_t hash () const =0

  Default hash function.
- virtual bool operator$<$ (const Bayesable$<$ t_datum, t_data $>$ &l) const
- virtual std::string string () const =0
- virtual void print (std::string prefix="")

Public Attributes

- double prior
- double likelihood
- double posterior
- uintmax_t born

7.1.1 Detailed Description

**template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
class Bayesable< _t_datum, _t_data >**

Author

steven piantadosi

Date

29/01/20

7.1.2 Member Typedef Documentation

7.1.2.1 t_data

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
typedef _t_data Bayesable< _t_datum, _t_data >::t_data
```

7.1.2.2 t_datum

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
typedef _t_datum Bayesable< _t_datum, _t_data >::t_datum
```

7.1.3 Constructor & Destructor Documentation

7.1.3.1 Bayesable()

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
Bayesable< _t_datum, _t_data >::Bayesable ( )  [inline]
```

7.1.4 Member Function Documentation

7.1.4.1 at_temperature()

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
virtual double Bayesable< _t_datum, _t_data >::at_temperature (
             double t )  [inline], [virtual]
```

Return my posterior score at a given (likelihood) temperature

Parameters

t	

Returns

7.1.4.2 clear_bayes()

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
virtual void Bayesable< _t_datum, _t_data >::clear_bayes ( )  [inline], [virtual]
```

Zero by prior, likelihood, posterior

7.1.4.3 compute_likelihood()

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
virtual double Bayesable< _t_datum, _t_data >::compute_likelihood (
             const t_data & data,
             const double breakout = -infinity )  [inline], [virtual]
```

Compute the likelihood of a collection of data, by calling compute_single_likelihood on each. This stops if our likelihood falls below breakout.

Parameters

data	
breakout	

Returns

Reimplemented in MyHypothesis, GrammarHypothesis< HYP, t_datum, t_data >, and MyHypothesis.

7.1.4.4 compute_posterior()

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
virtual double Bayesable< _t_datum, _t_data >::compute_posterior (
             const t_data & data,
             const double breakout = -infinity )  [inline], [virtual]
```

Compute the posterior, by calling prior and likelihood. This involves only a little bit of fanciness, which is that if our prior is -inf, then we don't both computing the likelihood.

Parameters

data	
breakout	

Returns

7.1.4.5 compute_prior()

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
virtual double Bayesable< _t_datum, _t_data >::compute_prior ( )  [pure virtual]
```

Compute the prior – defaultly not defined.

Implemented in Lexicon< HYP, T, t_input, t_output, t_datum >, Lexicon< MyHypothesis, InnerHypothesis, S, S >, MyHypothesis, GrammarHypothesis< HYP, t_datum, t_data >, LOTHypothesis< HYP, T, t_input, t_output, ↩ _t_datum, _t_data >, LOTHypothesis< InnerHypothesis, Node, S, S >, LOTHypothesis< MyHypothesis, Node, Object, bool >, and LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >.

7.1.4.6 compute_single_likelihood()

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
virtual double Bayesable< _t_datum, _t_data >::compute_single_likelihood (
             const t_datum & datum )  [pure virtual]
```

Compute the likelihood of a single data point.

Parameters

datum	

Implemented in MyHypothesis, GrammarHypothesis< HYP, t_datum, t_data >, LOTHypothesis< InnerHypothesis, Node, S, S >, LOTHypothesis< MyHypothesis, Node, Object, bool >, LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >, and MyHypothesis.

7.1.4.7 hash()

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
virtual size_t Bayesable< _t_datum, _t_data >::hash ( ) const  [pure virtual]
```

Default hash function.

Implemented in GrammarHypothesis< HYP, t_datum, t_data >, LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >, LOTHypothesis< InnerHypothesis, Node, S, S >, LOTHypothesis< MyHypothesis, Node, Object, bool >, LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >, Lexicon< HYP, T, t_input, t_output, t_datum >, and Lexicon< MyHypothesis, InnerHypothesis, S, S >.

7.1.4.8 operator<()

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
virtual bool Bayesable< _t_datum, _t_data >::operator< (
            const Bayesable< t_datum, t_data > & l ) const  [inline], [virtual]
```

Allow sorting of Bayesable hypotheses. We defaultly sort by posterior so that TopN works right. But we also need to be careful because std::set uses this to determine equality, so this also checks priors and then hashes.

Parameters

l	

Returns

7.1.4.9 print()

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
virtual void Bayesable< _t_datum, _t_data >::print (
            std::string prefix = "" )  [inline], [virtual]
```

Default printing of a hypothesis includes its posterior, prior, likelihood, and quoted string version

Parameters

prefix	

Reimplemented in MyHypothesis.

7.1.4.10 string()

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
virtual std::string Bayesable< _t_datum, _t_data >::string ( ) const  [pure virtual]
```

Implemented in GrammarHypothesis< HYP, t_datum, t_data >, LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >, LOTHypothesis< InnerHypothesis, Node, S, S >, LOTHypothesis< MyHypothesis, Node, Object, bool >, LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >, Lexicon< HYP, T, t_input, t_output, t_datum >, and Lexicon< MyHypothesis, InnerHypothesis, S, S >.

7.1.5 Member Data Documentation

7.1.5.1 born

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
uintmax_t Bayesable< _t_datum, _t_data >::born
```

7.1.5.2 likelihood

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
double Bayesable< _t_datum, _t_data >::likelihood
```

7.1.5.3 posterior

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
double Bayesable< _t_datum, _t_data >::posterior
```

7.1.5.4 prior

```
template<typename _t_datum, typename _t_data = std::vector<_t_datum>>
double Bayesable< _t_datum, _t_data >::prior
```

The documentation for this class was generated from the following file:

- src/Hypotheses/Interfaces/Bayesable.h

7.2 BuiltinPrimitive< t, args > Struct Template Reference

```
#include <Builtins.h>
```

Public Member Functions

- template<typename V , typename P , typename L >
  vmstatus_t VMScall (V ∗vms, P ∗pool, L ∗loader)

Public Attributes

- std::string format
- BuiltinOp op
- double p

7.2.1 Member Function Documentation

7.2.1.1 VMScall()

```
template<typename t, typename...  args>
template<typename V , typename P , typename L >
vmstatus_t BuiltinPrimitive< t, args >::VMScall (
             V * vms,
             P * pool,
             L * loader )  [inline]
```

7.2.2 Member Data Documentation

7.2.2.1 format

```
template<typename t, typename...  args>
std::string BuiltinPrimitive< t, args >::format
```

7.2.2.2 op

```
template<typename t, typename...  args>
BuiltinOp BuiltinPrimitive< t, args >::op
```

7.2.2.3 p

```
template<typename t, typename...  args>
double BuiltinPrimitive< t, args >::p
```

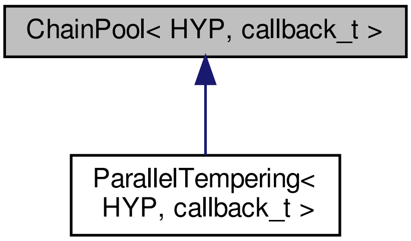
The documentation for this struct was generated from the following file:

- src/VirtualMachine/Builtins.h

7.3 ChainPool< HYP, callback_t > Class Template Reference

```
#include <ChainPool.h>
```

Inheritance diagram for ChainPool< HYP, callback_t >:


ChainPool< HYP, callback_t >
ParallelTempering< HYP, callback_t >

Public Member Functions

- ChainPool ()
- ChainPool (HYP &h0, typename HYP::t_data ∗d, callback_t &cb, size_t n, bool allcallback=true)
- virtual void run (Control ctl)

Static Public Member Functions

- static void __run_helper (std::vector< MCMCChain< HYP, callback_t >> ∗pool, std::vector< bool > ∗running, std::mutex ∗running_mutex, Control ctl)

Public Attributes

- std::vector< MCMCChain< HYP, callback_t > > pool
- std::mutex running_mutex

Static Public Attributes

- static const unsigned long steps_before_change = 0
- static const time_ms time_before_change = 250

7.3.1 Detailed Description

template<typename HYP, typename callback_t>
class ChainPool< HYP, callback_t >

Author

steven piantadosi

Date

29/01/20

7.3.2 Constructor & Destructor Documentation

7.3.2.1 ChainPool() [1/2]

```
template<typename HYP , typename callback_t >
ChainPool< HYP, callback_t >::ChainPool ( )  [inline]
```

7.3.2.2 ChainPool() [2/2]

```
template<typename HYP , typename callback_t >
ChainPool< HYP, callback_t >::ChainPool (
             HYP & h0,
             typename HYP::t_data * d,
             callback_t & cb,
             size_t n,
             bool allcallback = true )  [inline]
```

7.3.3 Member Function Documentation

7.3.3.1 __run_helper()

```
template<typename HYP , typename callback_t >
static void ChainPool< HYP, callback_t >::__run_helper (
             std::vector< MCMCChain< HYP, callback_t >> * pool,
             std::vector< bool > * running,
             std::mutex * running_mutex,
             Control ctl )  [inline], [static]
```

This run helper is called internally by multiple different threads, and runs a given pool.

Parameters

ctl	

7.3.3.2 run()

```
template<typename HYP , typename callback_t >
virtual void ChainPool< HYP, callback_t >::run (
             Control ctl )  [inline], [virtual]
```

Reimplemented in ParallelTempering< HYP, callback_t >.

7.3.4 Member Data Documentation

7.3.4.1 pool

```
template<typename HYP , typename callback_t >
std::vector<MCMCChain<HYP,callback_t> > ChainPool< HYP, callback_t >::pool
```

7.3.4.2 running_mutex

```
template<typename HYP , typename callback_t >
std::mutex ChainPool< HYP, callback_t >::running_mutex
```

7.3.4.3 steps_before_change

```
template<typename HYP , typename callback_t >
const unsigned long ChainPool< HYP, callback_t >::steps_before_change = 0  [static]
```

7.3.4.4 time_before_change

```
template<typename HYP , typename callback_t >
const time_ms ChainPool< HYP, callback_t >::time_before_change = 250  [static]
```

The documentation for this class was generated from the following file:

- src/Inference/ChainPool.h

7.4 Control Class Reference

```
#include <Control.h>
```

Public Member Functions

- Control (unsigned long s=0, time_ms t=0, size_t thr=1)
- void start ()
- bool running ()

Public Attributes

- unsigned long steps
- time_ms time
- size_t threads
- unsigned long burn
- unsigned long thin
- unsigned long restart
- timept start_time
- unsigned long done_steps
- bool break_CTRLC

7.4.1 Detailed Description

Author

steven piantadosi

Date

03/02/20

7.4.2 Constructor & Destructor Documentation

7.4.2.1 Control()

```
Control::Control (
            unsigned long s = 0,
            time_ms t = 0,
            size_t thr = 1 )  [inline]
```

7.4.3 Member Function Documentation

7.4.3.1 running()

```
bool Control::running ( )  [inline]
```

Check if we are currently running.

Returns

7.4.3.2 start()

```
void Control::start ( )  [inline]
```

Start running

7.4.4 Member Data Documentation

7.4.4.1 break_CTRLC

```
bool Control::break_CTRLC
```

7.4.4.2 burn

```
unsigned long Control::burn
```

7.4.4.3 done_steps

```
unsigned long Control::done_steps
```

7.4.4.4 restart

```
unsigned long Control::restart
```

7.4.4.5 start_time

```
timept Control::start_time
```

7.4.4.6 steps

```
unsigned long Control::steps
```

7.4.4.7 thin

```
unsigned long Control::thin
```

7.4.4.8 threads

```
size_t Control::threads
```

7.4.4.9 time

```
time_ms Control::time
```

The documentation for this class was generated from the following file:

- src/Control.h

7.5 default_datum< t_input, t_output > Class Template Reference

```
#include <Datum.h>
```

Public Member Functions

- default_datum ()
- default_datum (const t_input &i, const t_output &o, double r)
- default_datum (const t_input &i, const t_output &o)
- bool operator== (const default_datum &y) const

Public Attributes

- t_input input
- t_output output
- double reliability

7.5.1 Detailed Description

template<typename t_input, typename t_output>
class default_datum< t_input, t_output >

Author

piantado

Date

29/01/20

7.5.2 Constructor & Destructor Documentation

7.5.2.1 default_datum() [1/3]

```
template<typename t_input, typename t_output>
default_datum< t_input, t_output >::default_datum ( )  [inline]
```

7.5.2.2 default_datum() [2/3]

```
template<typename t_input, typename t_output>
default_datum< t_input, t_output >::default_datum (
             const t_input & i,
             const t_output & o,
             double r )  [inline]
```

7.5.2.3 default_datum() [3/3]

```
template<typename t_input, typename t_output>
default_datum< t_input, t_output >::default_datum (
             const t_input & i,
             const t_output & o )  [inline]
```

7.5.3 Member Function Documentation

7.5.3.1 operator==()

```
template<typename t_input, typename t_output>
bool default_datum< t_input, t_output >::operator== (
             const default_datum< t_input, t_output > & y ) const  [inline]
```

7.5.4 Member Data Documentation

7.5.4.1 input

```
template<typename t_input, typename t_output>
t_input default_datum< t_input, t_output >::input
```

7.5.4.2 output

```
template<typename t_input, typename t_output>
t_output default_datum< t_input, t_output >::output
```

7.5.4.3 reliability

```
template<typename t_input, typename t_output>
double default_datum< t_input, t_output >::reliability
```

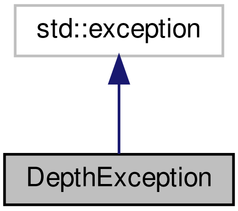
The documentation for this class was generated from the following file:

- src/Hypotheses/Datum.h

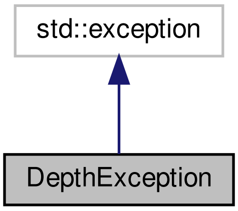
7.6 DepthException Class Reference

```
#include <Grammar.h>
```

Inheritance diagram for DepthException:



Collaboration diagram for DepthException:



The documentation for this class was generated from the following file:

- src/Grammar.h

7.7 DiscreteDistribution< T > Class Template Reference

```
#include <DiscreteDistribution.h>
```

Public Member Functions

- DiscreteDistribution ()
- virtual T argmax () const
- void print (std::ostream &out, unsigned long nprint=0) const
- void print (unsigned long nprint=0) const
- std::string string (unsigned long nprint=0) const
- void addmass (T x, double v)
- const std::map< T, double > & values () const
- void operator<< (const DiscreteDistribution< T > &x)
- std::vector< T > best (size_t N) const
- std::vector< std::pair< T, double > > sorted (bool decreasing=false) const
- size_t count (T x) const
- size_t size () const
- double operator[ ] (T x)
- double at (T x) const

Public Attributes

- std::map< T, double > m

7.7.1 Detailed Description

template<typename T>
class DiscreteDistribution< T >

Author

steven piantadosi

Date

03/02/20

7.7.2 Constructor & Destructor Documentation

7.7.2.1 DiscreteDistribution()

```
template<typename T>
DiscreteDistribution< T >::DiscreteDistribution ( )  [inline]
```

7.7.3 Member Function Documentation

7.7.3.1 addmass()

```
template<typename T>
void DiscreteDistribution< T >::addmass (
             T x,
             double v )  [inline]
```

Add log probability v to type x

Parameters

x	
v	

7.7.3.2 argmax()

```
template<typename T>
virtual T DiscreteDistribution< T >::argmax ( ) const  [inline], [virtual]
```

7.7.3.3 at()

```
template<typename T>
double DiscreteDistribution< T >::at (
             T x ) const  [inline]
```

7.7.3.4 best()

```
template<typename T>
std::vector<T> DiscreteDistribution< T >::best (
             size_t N ) const  [inline]
```

Get the N best from this distribution

Parameters

N	

Returns

7.7.3.5 count()

```
template<typename T>
size_t DiscreteDistribution< T >::count (
             T x ) const  [inline]
```

7.7.3.6 operator<<()

```
template<typename T>
void DiscreteDistribution< T >::operator<< (
             const DiscreteDistribution< T > & x )  [inline]
```

7.7.3.7 operator[]()

```
template<typename T>
double DiscreteDistribution< T >::operator[] (
             T x )  [inline]
```

7.7.3.8 print() [1/2]

```
template<typename T>
void DiscreteDistribution< T >::print (
            std::ostream & out,
            unsigned long nprint = 0 ) const  [inline]
```

7.7.3.9 print() [2/2]

```
template<typename T>
void DiscreteDistribution< T >::print (
            unsigned long nprint = 0 ) const  [inline]
```

7.7.3.10 size()

```
template<typename T>
size_t DiscreteDistribution< T >::size ( ) const  [inline]
```

7.7.3.11 sorted()

```
template<typename T>
std::vector<std::pair<T,double> > DiscreteDistribution< T >::sorted (
            bool decreasing = false ) const  [inline]
```

Get this distribution as a sorted vector of pairs

Parameters

decreasing	

7.7.3.12 string()

```
template<typename T>
std::string DiscreteDistribution< T >::string (
            unsigned long nprint = 0 ) const  [inline]
```

Convert this distribution into a string, printing at most nprint

Parameters

nprint	

Returns

7.7.3.13 values()

```
template<typename T>
const std::map<T,double>& DiscreteDistribution< T >::values ( ) const  [inline]
```

Get all of the values in this distribution

Returns

7.7.4 Member Data Documentation

7.7.4.1 m

```
template<typename T>
std::map<T,double> DiscreteDistribution< T >::m
```

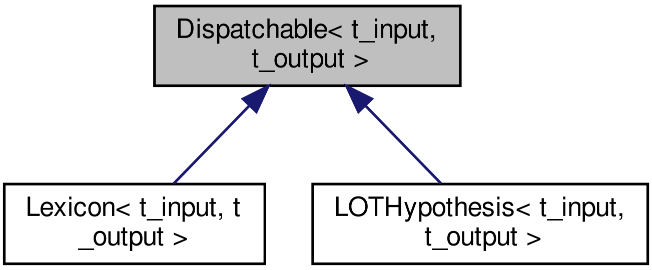
The documentation for this class was generated from the following file:

- src/DiscreteDistribution.h

7.8 Dispatchable< t_input, t_output > Class Template Reference

```
#include <Dispatchable.h>
```

Inheritance diagram for Dispatchable< t_input, t_output >:


Dispatchable< t_input, t_output >
Lexicon< t_input, t _output >
LOTHypothesis< t_input, t_output >

Public Member Functions

- virtual vmstatus_t dispatch_custom (Instruction i, VirtualMachinePool< t_input, t_output > ∗pool, VirtualMachineState< t_input, t_output > ∗vms, Dispatchable< t_input, t_output > ∗loader)=0
- virtual void push_program (Program &, short)=0

7.8.1 Detailed Description

template<typename t_input, typename t_output>
class Dispatchable< t_input, t_output >

Author

steven piantadosi

Date

03/02/20

7.8.2 Member Function Documentation

7.8.2.1 dispatch_custom()

```
template<typename t_input, typename t_output>
virtual vmstatus_t Dispatchable< t_input, t_output >::dispatch_custom (
             Instruction i,
             VirtualMachinePool< t_input, t_output > * pool,
             VirtualMachineState< t_input, t_output > * vms,
             Dispatchable< t_input, t_output > * loader )  [pure virtual]
```

Implemented in Lexicon< HYP, T, t_input, t_output, t_datum >, Lexicon< MyHypothesis, InnerHypothesis, S, S >, InnerHypothesis, LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >, LOTHypothesis< InnerHypothesis, Node, S, S >, LOTHypothesis< MyHypothesis, Node, Object, bool >, LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >, and MyHypothesis.

7.8.2.2 push_program()

```
template<typename t_input, typename t_output>
virtual void Dispatchable< t_input, t_output >::push_program (
             Program & ,
             short )  [pure virtual]
```

Implemented in Lexicon< HYP, T, t_input, t_output, t_datum >, Lexicon< MyHypothesis, InnerHypothesis, S, S >, LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >, LOTHypothesis< InnerHypothesis, Node, S, S >, LOTHypothesis< MyHypothesis, Node, Object, bool >, and LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >.

The documentation for this class was generated from the following file:

- src/Hypotheses/Interfaces/Dispatchable.h

7.9 Fleet::Statistics::FiniteHistory< T > Class Template Reference

```
#include <FiniteHistory.h>
```

Public Member Functions

- FiniteHistory (size_t n)
- FiniteHistory ()
- FiniteHistory (const FiniteHistory &fh)
- FiniteHistory (FiniteHistory &&fh)
- void operator= (const FiniteHistory &fh)
- void operator= (FiniteHistory &&fh)
- void add (T x)
- void operator<< (T x)
- double mean ()

Public Attributes

- std::vector< T > history
- std::atomic< size_t > history_size
- std::atomic< size_t > history_index
- std::atomic< unsigned long > N
- std::mutex mutex

7.9.1 Detailed Description

template<typename T>
class Fleet::Statistics::FiniteHistory< T >

Author

steven piantadosi

Date

03/02/20

7.9.2 Constructor & Destructor Documentation

7.9.2.1 FiniteHistory() [1/4]

```
template<typename T>
Fleet::Statistics::FiniteHistory< T >::FiniteHistory (
            size_t n ) [inline]
```

7.9.2.2 FiniteHistory() [2/4]

```
template<typename T>
Fleet::Statistics::FiniteHistory< T >::FiniteHistory ( )  [inline]
```

7.9.2.3 FiniteHistory() [3/4]

```
template<typename T>
Fleet::Statistics::FiniteHistory< T >::FiniteHistory (
             const FiniteHistory< T > & fh )  [inline]
```

7.9.2.4 FiniteHistory() [4/4]

```
template<typename T>
Fleet::Statistics::FiniteHistory< T >::FiniteHistory (
             FiniteHistory< T > && fh )  [inline]
```

7.9.3 Member Function Documentation

7.9.3.1 add()

```
template<typename T>
void Fleet::Statistics::FiniteHistory< T >::add (
             T x )  [inline]
```

7.9.3.2 mean()

```
template<typename T>
double Fleet::Statistics::FiniteHistory< T >::mean ( )  [inline]
```

7.9.3.3 operator<<()

```
template<typename T>
void Fleet::Statistics::FiniteHistory< T >::operator<< (
             T x )  [inline]
```

7.9.3.4 operator=() [1/2]

```
template<typename T>
void Fleet::Statistics::FiniteHistory< T >::operator= (
            const FiniteHistory< T > & fh ) [inline]
```

7.9.3.5 operator=() [2/2]

```
template<typename T>
void Fleet::Statistics::FiniteHistory< T >::operator= (
            FiniteHistory< T > && fh ) [inline]
```

7.9.4 Member Data Documentation

7.9.4.1 history

```
template<typename T>
std::vector<T> Fleet::Statistics::FiniteHistory< T >::history
```

7.9.4.2 history_index

```
template<typename T>
std::atomic<size_t> Fleet::Statistics::FiniteHistory< T >::history_index
```

7.9.4.3 history_size

```
template<typename T>
std::atomic<size_t> Fleet::Statistics::FiniteHistory< T >::history_size
```

7.9.4.4 mutex

```
template<typename T>
std::mutex Fleet::Statistics::FiniteHistory< T >::mutex [mutable]
```

7.9.4.5 N

```
template<typename T>
std::atomic<unsigned long> Fleet::Statistics::FiniteHistory< T >::N
```

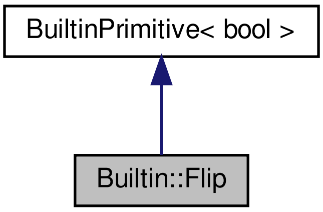
The documentation for this class was generated from the following file:

- src/Statistics/FiniteHistory.h

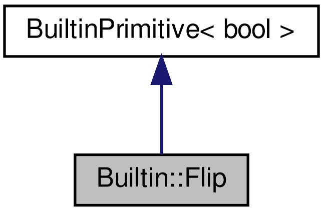
7.10 Builtin::Flip Struct Reference

```
#include <Builtins.h>
```

Inheritance diagram for Builtin::Flip:



Collaboration diagram for Builtin::Flip:



Public Member Functions

- Flip (std::string fmt, double _p=1.0)

Additional Inherited Members

7.10.1 Constructor & Destructor Documentation

7.10.1.1 Flip()

```
Builtin::Flip::Flip (
             std::string fmt,
             double _p = 1.0 )  [inline]
```

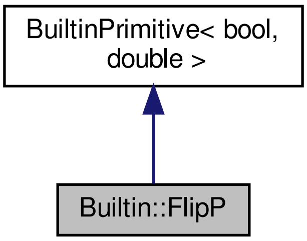
The documentation for this struct was generated from the following file:

- src/VirtualMachine/Builtins.h

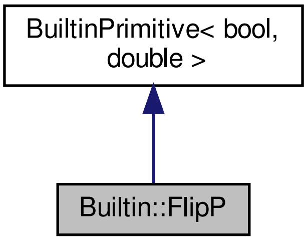
7.11 Builtin::FlipP Struct Reference

```
#include <Builtins.h>
```

Inheritance diagram for Builtin::FlipP:



Collaboration diagram for Builtin::FlipP:



Public Member Functions

- FlipP (std::string fmt, double _p=1.0)

Additional Inherited Members

7.11.1 Constructor & Destructor Documentation

7.11.1.1 FlipP()

```
Builtin::FlipP::FlipP (
             std::string fmt,
             double _p = 1.0 )  [inline]
```

The documentation for this struct was generated from the following file:

- src/VirtualMachine/Builtins.h

7.12 Grammar Class Reference

```
#include <Grammar.h>
```

Public Member Functions

- template< class T >
  constexpr nonterminal_t nt ()
- Grammar ()
- template< typename... T>
  Grammar (std::tuple< T... > tup)
- Grammar (const Grammar &g)=delete
- Grammar (const Grammar &&g)=delete
- size_t count_nonterminals () const
- size_t count_rules (const nonterminal_t nt) const
- size_t count_rules () const
- size_t count_terminals (nonterminal_t nt) const
- size_t count_nonterminals (nonterminal_t nt) const
- size_t count_expansions (const nonterminal_t nt) const
- void show (std::string prefix="# ")
- virtual void add (Rule &&r)
- template< typename... args, size_t... Is>
  void add (std::tuple< args... > t, std::index_sequence< Is... >)
- template< typename T , typename... args>
  void add (Primitive< T, args... > p, const int arg=0)
- template< typename T , typename... args>
  void add (BuiltinPrimitive< T, args... > p, const int arg=0)

- template<typename T , typename... args>
  void add (BuiltinOp o, std::string format, const double p=1.0, const int arg=0)
- template<typename T , typename... args>
  void add (CustomOp o, std::string format, const double p=1.0, const int arg=0)
- size_t get_index_of (const Rule ∗r) const
- virtual Rule ∗ get_rule (const nonterminal_t nt, size_t k) const
- virtual Rule ∗ get_rule (const nonterminal_t nt, const CustomOp o, const int a=0)
- virtual Rule ∗ get_rule (const nonterminal_t nt, const BuiltinOp o, const int a=0)
- virtual Rule ∗ get_rule (const std::string s) const
- double rule_normalizer (const nonterminal_t nt) const
- virtual Rule ∗ sample_rule (const nonterminal_t nt) const
- Node makeNode (const Rule ∗r) const
- Node generate (const nonterminal_t nt, unsigned long depth=0) const
- template<class t >
  Node generate (unsigned long depth=0)
- Node copy_resample (const Node &node, bool f(const Node &n)) const
- std::vector< size_t > get_counts (const Node &node) const
- double log_probability (const Node &n) const
- Node expand_from_names (std::deque< std::string > &q) const
- Node expand_from_names (std::string s) const
- Node expand_from_names (const char ∗c) const
- Node expand_from_integer (nonterminal_t nt, IntegerizedStack &is) const
- Node expand_from_integer (nonterminal_t nt, enumerationidx_t z) const
- enumerationidx_t compute_enumeration_order (const Node &n)
- Node lempel_ziv_full_expand (nonterminal_t nt, enumerationidx_t z, Node ∗root=nullptr) const
- Node lempel_ziv_full_expand (nonterminal_t nt, IntegerizedStack &is, Node ∗root=nullptr) const
- virtual enumerationidx_t count_connected_partial_subtrees (const Node &n) const
- size_t neighbors (const Node &node) const
- void expand_to_neighbor (Node &node, int &which)
- void complete (Node &node)

Protected Attributes

- std::vector< Rule > rules [N_NTs]
- double Z [N_NTs]

7.12.1 Constructor & Destructor Documentation

7.12.1.1 Grammar() [1/4]

```
Grammar::Grammar ( )  [inline]
```

7.12.1.2 Grammar() [2/4]

```
template<typename...  T>
Grammar::Grammar (
            std::tuple< T...  > tup )  [inline]
```

Constructor for grammar that uses a tuple of Primitives.

Parameters

tup	- a tuple of Primitives

7.12.1.3 Grammar() [3/4]

```
Grammar::Grammar (
             const Grammar & g )  [delete]
```

7.12.1.4 Grammar() [4/4]

```
Grammar::Grammar (
             const Grammar && g )  [delete]
```

7.12.2 Member Function Documentation

7.12.2.1 add() [1/6]

```
virtual void Grammar::add (
             Rule && r )  [inline], [virtual]
```

Add a rule

7.12.2.2 add() [2/6]

```
template<typename...  args, size_t...  Is>
void Grammar::add (
             std::tuple< args...  > t,
             std::index_sequence< Is...  >  )  [inline]
```

7.12.2.3 add() [3/6]

```
template<typename T , typename...  args>
void Grammar::add (
             Primitive< T, args...  > p,
             const int arg = 0 )  [inline]
```

7.12.2.4 add() [4/6]

```
template<typename T , typename...  args>
void Grammar::add (
             BuiltinPrimitive< T, args...  > p,
             const int arg = 0 )  [inline]
```

7.12.2.5 add() [5/6]

```
template<typename T , typename...  args>
void Grammar::add (
             BuiltinOp o,
             std::string format,
             const double p = 1.0,
             const int arg = 0 )  [inline]
```

7.12.2.6 add() [6/6]

```
template<typename T , typename...  args>
void Grammar::add (
             CustomOp o,
             std::string format,
             const double p = 1.0,
             const int arg = 0 )  [inline]
```

7.12.2.7 complete()

```
void Grammar::complete (
             Node & node )  [inline]
```

7.12.2.8 compute_enumeration_order()

```
enumerationidx_t Grammar::compute_enumeration_order (
             const Node & n )  [inline]
```

7.12.2.9 copy_resample()

```
Node Grammar::copy_resample (
             const Node & node,
             bool  fconst Node &n ) const  [inline]
```

Make a copy of node where all nodes satisfying f are regenerated from the grammar.

Parameters

node	
f	- a function saying what we should resample

Returns

NOTE: this does NOT allow f to apply to nullptr children (so cannot be used to fill in)

7.12.2.10 count_connected_partial_subtrees()

```
virtual enumerationidx_t Grammar::count_connected_partial_subtrees (
            const Node & n ) const  [inline], [virtual]
```

7.12.2.11 count_expansions()

```
size_t Grammar::count_expansions (
            const nonterminal_t nt ) const  [inline]
```

7.12.2.12 count_nonterminals() [1/2]

```
size_t Grammar::count_nonterminals ( ) const  [inline]
```

How many nonterminals are there in the grammar.

Returns

7.12.2.13 count_nonterminals() [2/2]

```
size_t Grammar::count_nonterminals (
            nonterminal_t nt ) const  [inline]
```

Count th enumber of non-terminal rules of return type nt

Parameters

nt	

Returns

7.12.2.14 count_rules() [1/2]

```
size_t Grammar::count_rules (
            const nonterminal_t nt ) const  [inline]
```

Returns the number of rules of return type nt

Parameters

nt	

Returns

7.12.2.15 count_rules() [2/2]

```
size_t Grammar::count_rules ( ) const  [inline]
```

Total number of rules

Returns

7.12.2.16 count_terminals()

```
size_t Grammar::count_terminals (
            nonterminal_t nt ) const  [inline]
```

Count the number of terminal rules of return type nt

Parameters

nt	

Returns

7.12.2.17 expand_from_integer() [1/2]

```
Node Grammar::expand_from_integer (
             nonterminal_t nt,
             IntegerizedStack & is ) const  [inline]
```

7.12.2.18 expand_from_integer() [2/2]

```
Node Grammar::expand_from_integer (
             nonterminal_t nt,
             enumerationidx_t z ) const  [inline]
```

7.12.2.19 expand_from_names() [1/3]

```
Node Grammar::expand_from_names (
             std::deque< std::string > & q ) const  [inline]
```

Fills an entire tree using the string format prefixes – see get_rule(std::string)

Parameters

q	

Returns

7.12.2.20 expand_from_names() [2/3]

```
Node Grammar::expand_from_names (
             std::string s ) const  [inline]
```

Expand from names where s is delimited by ':'

Parameters

s	

Returns

7.12.2.21 expand_from_names() [3/3]

```
Node Grammar::expand_from_names (
             const char * c ) const  [inline]
```

7.12.2.22 expand_to_neighbor()

```
void Grammar::expand_to_neighbor (
             Node & node,
             int & which )  [inline]
```

7.12.2.23 generate() [1/2]

```
Node Grammar::generate (
             const nonterminal_t nt,
             unsigned long depth = 0 ) const  [inline]
```

Sample an entire tree from this grammar (keeping track of depth in case we recurse too far) of return type nt. This samples a rule, makes them with makeNode, and then recurses.

Parameters

nt	
depth	

Returns

Returns a Node sampled from the grammar.

NOTE: this may throw a DepthException if the grammar recurses too far (usually that means the grammar is improper)

7.12.2.24 generate() [2/2]

```
template<class t >
Node Grammar::generate (
             unsigned long depth = 0 )  [inline]
```

A friendly version of generate that can be called with template by type.

Parameters

depth	

Returns

7.12.2.25 get_counts()

```
std::vector<size_t> Grammar::get_counts (
             const Node & node ) const  [inline]
```

Compute a vector of counts of how often each rule was used, in a *standard* order given by iterating over nts and then iterating over rules

Parameters

node	

Returns

7.12.2.26 get_index_of()

```
size_t Grammar::get_index_of (
             const Rule * r ) const  [inline]
```

Find the index in rules of where r is.

Parameters

r	

Returns

7.12.2.27 get_rule() [1/4]

```
virtual Rule* Grammar::get_rule (
             const nonterminal_t nt,
             size_t k ) const  [inline], [virtual]
```

Get the k'th rule of type nt

Parameters

nt	
k	

Returns

7.12.2.28 get_rule() [2/4]

```
virtual Rule* Grammar::get_rule (
             const nonterminal_t nt,
             const CustomOp o,
             const int a = 0 )  [inline], [virtual]
```

Get rule of type nt with a given CustomOp and argument a

Parameters

nt	
o	
a	

Returns

7.12.2.29 get_rule() [3/4]

```
virtual Rule* Grammar::get_rule (
             const nonterminal_t nt,
             const BuiltinOp o,
             const int a = 0 )  [inline], [virtual]
```

Get rule of type nt with a given BuiltinOp and argument a

Parameters

nt	
o	
a	

Returns

7.12.2.30 get_rule() [4/4]

```
virtual Rule* Grammar::get_rule (
             const std::string s ) const  [inline], [virtual]
```

Return a rule based on s, which must uniquely be a prefix of the rule's format

Parameters

s	

Returns

7.12.2.31 lempel_ziv_full_expand() [1/2]

```
Node Grammar::lempel_ziv_full_expand (
             nonterminal_t nt,
             enumerationidx_t z,
             Node * root = nullptr ) const  [inline]
```

7.12.2.32 lempel_ziv_full_expand() [2/2]

```
Node Grammar::lempel_ziv_full_expand (
             nonterminal_t nt,
             IntegerizedStack & is,
             Node * root = nullptr ) const  [inline]
```

7.12.2.33 log_probability()

```
double Grammar::log_probability (
             const Node & n ) const  [inline]
```

Compute the log probability of a tree according to the grammar

Parameters

n	

Returns

7.12.2.34 makeNode()

```
Node Grammar::makeNode (
             const Rule * r ) const  [inline]
```

Helper function to create a node according to this grammar. This is how nodes get their log probabilities.

Parameters

r	

Returns

7.12.2.35 neighbors()

```
size_t Grammar::neighbors (
             const Node & node ) const  [inline]
```

7.12.2.36 nt()

```
template<class T >
constexpr nonterminal_t Grammar::nt ( )  [inline]
```

template function giving the index of its template argument (index in FLEET_GRAMMAR_TYPES). NOTE: The names here are decayed (meaning that references and base types are the same.

7.12.2.37 rule_normalizer()

```
double Grammar::rule_normalizer (
             const nonterminal_t nt ) const  [inline]
```

Return the normalizing constant (NOT log) for all rules of type nt

Parameters

nt	

Returns

7.12.2.38 sample_rule()

```
virtual Rule* Grammar::sample_rule (
             const nonterminal_t nt ) const  [inline], [virtual]
```

Randomly sample a rule of type nt.

Parameters

nt	

Returns

7.12.2.39 show()

```
void Grammar::show (
             std::string prefix = "# " )  [inline]
```

Show the grammar by printing each rule

7.12.3 Member Data Documentation

7.12.3.1 rules

```
std::vector<Rule> Grammar::rules[N_NTs]  [protected]
```

7.12.3.2 Z

```
double Grammar::Z[N_NTs]  [protected]
```

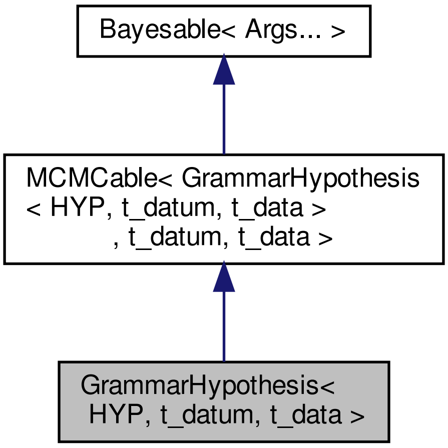
The documentation for this class was generated from the following file:

- src/Grammar.h

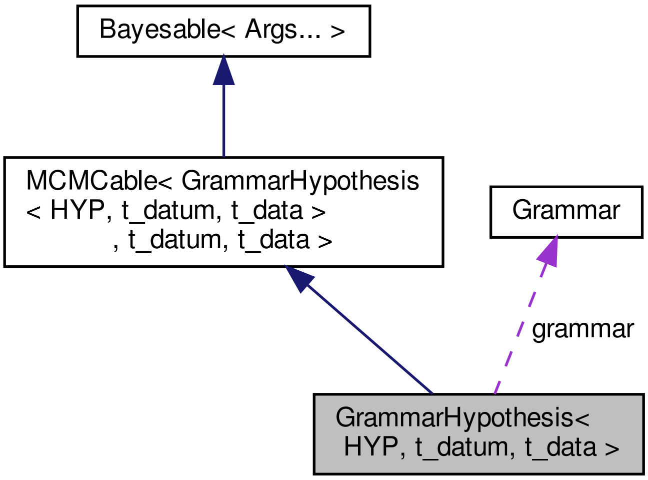
7.13 GrammarHypothesis< HYP, t_datum, t_data > Class Template Reference

```
#include <GrammarHypothesis.h>
```

Inheritance diagram for GrammarHypothesis< HYP, t_datum, t_data >:


Bayesable< Args... >
MCMCable< GrammarHypothesis < HYP, t_datum, t_data > , t_datum, t_data >
GrammarHypothesis< HYP, t_datum, t_data >

Collaboration diagram for GrammarHypothesis< HYP, t_datum, t_data >:


Bayesable< Args... >
MCMCable< GrammarHypothesis < HYP, t_datum, t_data > , t_datum, t_data >
Grammar
grammar
GrammarHypothesis< HYP, t_datum, t_data >

Public Member Functions

- GrammarHypothesis ()
- GrammarHypothesis (Grammar ∗g, Matrix ∗c, Matrix ∗ll, Matrix ∗p)
- Vector & getX ()
- const Vector & getX () const
- float get_baseline () const
- float get_forwardalpha () const
- double compute_prior ()

  Compute the prior – defaultly not defined.
- virtual double compute_single_likelihood (const t_datum &datum)

  Compute the likelihood of a single data point.
- virtual double compute_likelihood (const t_data &data, const double breakout=-infinity)

  Compute the likelihood of a collection of data, by calling compute_single_likelihood on each. This stops if our likelihood falls below breakout.
- virtual GrammarHypothesis restart () const
- virtual std::pair< GrammarHypothesis, double > propose () const
- Vector hypothesis_prior (Matrix &C)
- virtual bool operator== (const GrammarHypothesis< HYP, t_datum, t_data > &h) const
- virtual std::string string () const
- virtual size_t hash () const

  Default hash function.

Public Attributes

- Vector x
- Grammar ∗ grammar
- Matrix ∗ C
- Matrix ∗ LL
- Matrix ∗ P
- float logodds_baseline
- float logodds_forwardalpha

Additional Inherited Members

7.13.1 Constructor & Destructor Documentation

7.13.1.1 GrammarHypothesis() [1/2]

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
GrammarHypothesis< HYP, t_datum, t_data >::GrammarHypothesis ( )  [inline]
```

7.13.1.2 GrammarHypothesis() [2/2]

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
GrammarHypothesis< HYP, t_datum, t_data >::GrammarHypothesis (
             Grammar * g,
             Matrix * c,
             Matrix * ll,
             Matrix * p )  [inline]
```

7.13.2 Member Function Documentation

7.13.2.1 compute_likelihood()

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
virtual double GrammarHypothesis< HYP, t_datum, t_data >::compute_likelihood (
             const t_data & data,
             const double breakout = -infinity )  [inline], [virtual]
```

Compute the likelihood of a collection of data, by calling compute_single_likelihood on each. This stops if our likelihood falls below breakout.

Parameters

data	
breakout	

Returns

Reimplemented from Bayesable< Args... >.

7.13.2.2 compute_prior()

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
double GrammarHypothesis< HYP, t_datum, t_data >::compute_prior ( )  [inline], [virtual]
```

Compute the prior – defaultly not defined.

Implements Bayesable< Args... >.

7.13.2.3 compute_single_likelihood()

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
virtual double GrammarHypothesis< HYP, t_datum, t_data >::compute_single_likelihood (
            const t_datum & datum )  [inline], [virtual]
```

Compute the likelihood of a single data point.

Parameters

datum	

Implements Bayesable< Args... >.

7.13.2.4 get_baseline()

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
float GrammarHypothesis< HYP, t_datum, t_data >::get_baseline ( ) const  [inline]
```

7.13.2.5 get_forwardalpha()

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
float GrammarHypothesis< HYP, t_datum, t_data >::get_forwardalpha ( ) const  [inline]
```

7.13.2.6 getX() [1/2]

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
Vector& GrammarHypothesis< HYP, t_datum, t_data >::getX ( )  [inline]
```

7.13.2.7 getX() [2/2]

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
const Vector& GrammarHypothesis< HYP, t_datum, t_data >::getX ( ) const  [inline]
```

7.13.2.8 hash()

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
virtual size_t GrammarHypothesis< HYP, t_datum, t_data >::hash ( ) const  [inline], [virtual]
```

Default hash function.

Implements Bayesable< Args... >.

7.13.2.9 hypothesis_prior()

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
Vector GrammarHypothesis< HYP, t_datum, t_data >::hypothesis_prior (
             Matrix & C )  [inline]
```

7.13.2.10 operator==()

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
virtual bool GrammarHypothesis< HYP, t_datum, t_data >::operator== (
             const GrammarHypothesis< HYP, t_datum, t_data > & h ) const  [inline], [virtual]
```

Implements MCMCable< GrammarHypothesis< HYP, t_datum, t_data >, t_datum, t_data >.

7.13.2.11 propose()

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
virtual std::pair<GrammarHypothesis,double> GrammarHypothesis< HYP, t_datum, t_data >::propose
( ) const  [inline], [virtual]
```

Implements MCMCable< GrammarHypothesis< HYP, t_datum, t_data >, t_datum, t_data >.

7.13.2.12 restart()

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
virtual GrammarHypothesis GrammarHypothesis< HYP, t_datum, t_data >::restart ( ) const  [inline],
[virtual]
```

Implements MCMCable< GrammarHypothesis< HYP, t_datum, t_data >, t_datum, t_data >.

7.13.2.13 string()

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
virtual std::string GrammarHypothesis< HYP, t_datum, t_data >::string ( ) const  [inline],
[virtual]
```

Implements Bayesable< Args... >.

7.13.3 Member Data Documentation

7.13.3.1 C

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
Matrix* GrammarHypothesis< HYP, t_datum, t_data >::C
```

7.13.3.2 grammar

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
Grammar* GrammarHypothesis< HYP, t_datum, t_data >::grammar
```

7.13.3.3 LL

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
Matrix* GrammarHypothesis< HYP, t_datum, t_data >::LL
```

7.13.3.4 logodds_baseline

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
float GrammarHypothesis< HYP, t_datum, t_data >::logodds_baseline
```

7.13.3.5 logodds_forwardalpha

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
float GrammarHypothesis< HYP, t_datum, t_data >::logodds_forwardalpha
```

7.13.3.6 P

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
Matrix* GrammarHypothesis< HYP, t_datum, t_data >::P
```

7.13.3.7 x

```
template<typename HYP, typename t_datum, typename t_data = std::vector<t_datum>>
Vector GrammarHypothesis< HYP, t_datum, t_data >::x
```

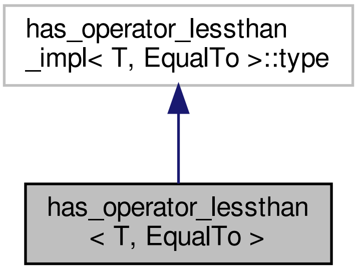
The documentation for this class was generated from the following file:

- src/Hypotheses/GrammarHypothesis.h

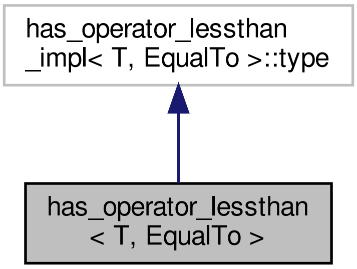
7.14 has_operator_lessthan< T, EqualTo > Struct Template Reference

```
#include <TemplateMagic.h>
```

Inheritance diagram for has_operator_lessthan< T, EqualTo >:



Collaboration diagram for has_operator_lessthan< T, EqualTo >:



The documentation for this struct was generated from the following file:

- src/TemplateMagic.h

7.15 has_operator_lessthan_impl< T, EqualTo > Struct Template Reference

```
#include <TemplateMagic.h>
```

Public Types

- using type = typename std::is_same< bool, decltype(test< T, EqualTo >(0))>::type

Static Public Member Functions

- template<class U , class V >
  static auto test (U ∗) -> decltype(std::declval< U >()< std::declval< V >())
- template<typename , typename >
  static auto test (...) -> std::false_type

7.15.1 Detailed Description

template<class T, class EqualTo>
struct has_operator_lessthan_impl< T, EqualTo >

```
    Find the numerical index (as a nonterminal_t) in a tuple of a given type
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

// When users define the macro FLEET_GRAMMAR_TYPES, as in
// #define FLEET_GRAMMAR_TYPES int,double,char
// then we can use type2int to map each to a unique int. This mapping to ints is
// for example how Fleet stores information in the grammar

// When users define the macro FLEET_GRAMMAR_TYPES, as in
```

```
// #define FLEET_GRAMMAR_TYPES int,double,char
// then we can use type2int to map each to a unique int. This mapping to ints is
// for example how Fleet stores information in the grammar

typedef size_t nonterminal_t;

// from https://stackoverflow.com/questions/42258608/c-constexpr-values-for-types
template <class T, class Tuple>
struct TypeIndex;

template <class T, class... Types>
struct TypeIndex<T, std::tuple<T, Types...>> {
    static const nonterminal_t value = 0;
};

template <class T, class U, class... Types>
struct TypeIndex<T, std::tuple<U, Types...>> {
    static const nonterminal_t value = 1 + TypeIndex<T, std::tuple<Types...>>::value;
};

/*
```

Fancy trick to see if a class implements operator< (for filtering out in op_MEM code so it doesn't give an error if we use t_input that doesn't implement operator< as long as no op_MEM is called ~~~~~~

7.15.2 Member Typedef Documentation

7.15.2.1 type

```
template<class T, class EqualTo>
using has_operator_lessthan_impl< T, EqualTo >::type = typename std::is_same<bool, decltype(test<T,
EqualTo>(0))>::type
```

7.15.3 Member Function Documentation

7.15.3.1 test() [1/2]

```
template<class T, class EqualTo>
template<class U , class V >
static auto has_operator_lessthan_impl< T, EqualTo >::test (
              U * ) -> decltype(std::declval< U >()< std::declval< V >()) [static]
```

7.15.3.2 test() [2/2]

```
template<class T, class EqualTo>
template<typename , typename >
static auto has_operator_lessthan_impl< T, EqualTo >::test (
               ... ) -> std::false_type [static]
```

The documentation for this struct was generated from the following file:

- src/TemplateMagic.h

7.16 HeadIfReferenceElseT< T, args > Struct Template Reference

```
#include <TemplateMagic.h>
```

Public Types

- typedef std::conditional< std::is_reference< typename TypeHead< args... >::type >::value, typename std::decay< typename TypeHead< args... >::type >::type, T >::type type

7.16.1 Member Typedef Documentation

7.16.1.1 type

```
template<class T , class...  args>
typedef std::conditional<std::is_reference<typename TypeHead<args...>::type>::value, typename
std::decay<typename TypeHead<args...>::type>::type, T >::type HeadIfReferenceElseT< T, args
>::type
```

The documentation for this struct was generated from the following file:

- src/TemplateMagic.h

7.17 HeadIfReferenceElseT< T > Struct Template Reference

```
#include <TemplateMagic.h>
```

Public Types

- typedef T type

7.17.1 Member Typedef Documentation

7.17.1.1 type

```
template<class T >
typedef T HeadIfReferenceElseT< T >::type
```

The documentation for this struct was generated from the following file:

- src/TemplateMagic.h

7.18 HumanDatum< t_learnerdatum, t_learnerdata > Struct Template Reference

```
#include <GrammarHypothesis.h>
```

Public Attributes

- size_t cntyes
- size_t cntno
- t_learnerdata given_data
- t_learnerdatum predict_data

7.18.1 Member Data Documentation

7.18.1.1 cntno

```
template<typename t_learnerdatum , typename t_learnerdata = std::vector<t_learnerdatum>>
size_t HumanDatum< t_learnerdatum, t_learnerdata >::cntno
```

7.18.1.2 cntyes

```
template<typename t_learnerdatum , typename t_learnerdata = std::vector<t_learnerdatum>>
size_t HumanDatum< t_learnerdatum, t_learnerdata >::cntyes
```

7.18.1.3 given_data

```
template<typename t_learnerdatum , typename t_learnerdata = std::vector<t_learnerdatum>>
t_learnerdata HumanDatum< t_learnerdatum, t_learnerdata >::given_data
```

7.18.1.4 predict_data

```
template<typename t_learnerdatum , typename t_learnerdata = std::vector<t_learnerdatum>>
t_learnerdatum HumanDatum< t_learnerdatum, t_learnerdata >::predict_data
```

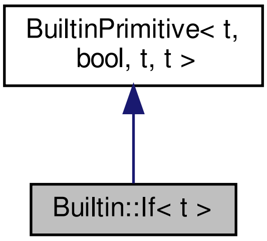
The documentation for this struct was generated from the following file:

- src/Hypotheses/GrammarHypothesis.h

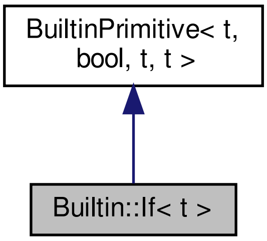
7.19 Builtin::If< t > Struct Template Reference

```
#include <Builtins.h>
```

Inheritance diagram for Builtin::If< t >:



Collaboration diagram for Builtin::If< t >:



Public Member Functions

- If (std::string fmt, double _p=1.0)

Additional Inherited Members

7.19.1 Constructor & Destructor Documentation

7.19.1.1 If()

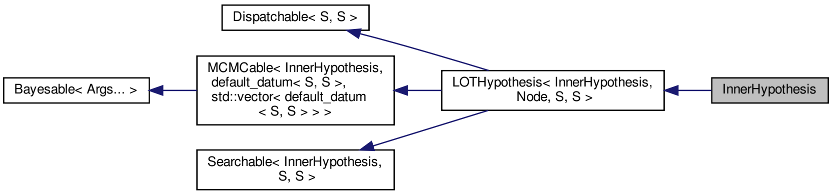
```
template<typename t >
Builtin::If< t >::If (
              std::string fmt,
              double _p = 1.0 )  [inline]
```

The documentation for this struct was generated from the following file:

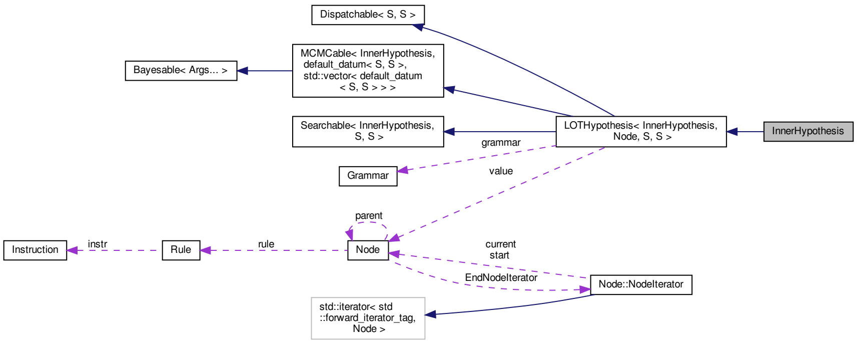
- src/VirtualMachine/Builtins.h

7.20 InnerHypothesis Class Reference

Inheritance diagram for InnerHypothesis:


Dispatchable< S, S >
Bayesable< Args... >
MCMCable< InnerHypothesis, default_datum< S, S >, std::vector< default_datum < S, S > > >
LOTHypothesis< InnerHypothesis, Node, S, S >
InnerHypothesis
Searchable< InnerHypothesis, S, S >

Collaboration diagram for InnerHypothesis:


Dispatchable< S, S >
Bayesable< Args... >
MCMCable< InnerHypothesis, default_datum< S, S >, std::vector< default_datum < S, S > > >
Searchable< InnerHypothesis, S, S >
LOTHypothesis< InnerHypothesis, Node, S, S >
InnerHypothesis
grammar
Grammar
value
parent
Instruction
instr
Rule
rule
Node
current
start
EndNodeIterator
Node::NodeIterator
std::iterator< std ::forward_iterator_tag, Node >

Public Types

- using Super = LOTHypothesis< InnerHypothesis, Node, S, S >

Public Member Functions

- virtual vmstatus_t dispatch_custom (Instruction i, VirtualMachinePool< S, S > ∗pool, VirtualMachineState< S, S > ∗vms, Dispatchable< S, S > ∗loader)
- virtual std::pair< InnerHypothesis, double > propose () const

Additional Inherited Members

7.20.1 Member Typedef Documentation

7.20.1.1 Super

```
using InnerHypothesis::Super = LOTHypothesis<InnerHypothesis,Node,S,S>
```

7.20.2 Member Function Documentation

7.20.2.1 dispatch_custom()

```
virtual vmstatus_t InnerHypothesis::dispatch_custom (
            Instruction i,
            VirtualMachinePool< S, S > * pool,
            VirtualMachineState< S, S > * vms,
            Dispatchable< S, S > * loader )  [inline], [virtual]
```

Reimplemented from LOTHypothesis< InnerHypothesis, Node, S, S >.

7.20.2.2 propose()

```
virtual std::pair<InnerHypothesis,double> InnerHypothesis::propose ( ) const  [inline], [virtual]
```

Reimplemented from LOTHypothesis< InnerHypothesis, Node, S, S >.

The documentation for this class was generated from the following file:

- Models/FormalLanguageTheory-Complex/Main.cpp

7.21 Instruction Class Reference

```
#include <Instruction.h>
```

Public Member Functions

- Instruction ()
- Instruction (BuiltinOp x, int arg_=0x0)
- Instruction (CustomOp x, int arg_=0x0)
- Instruction (PrimitiveOp x, int arg_=0x0)
- template<typename t >
  bool is () const
- template<typename t >
  t as () const
- int getArg () const
- bool operator== (const Instruction &i) const
- template<typename T >
  bool is_a (const T x) const

  compare the instruction types (ignores the arg)
- template<typename T , typename... Ts>
  bool is_a (T x, Ts... args) const

Public Attributes

- std::variant< BuiltinOp, CustomOp, PrimitiveOp > op
- int arg

7.21.1 Detailed Description

Author

piantado

Date

29/01/20

7.21.2 Constructor & Destructor Documentation

7.21.2.1 Instruction() [1/4]

```
Instruction::Instruction ( )  [inline]
```

7.21.2.2 Instruction() [2/4]

```
Instruction::Instruction (
              BuiltinOp x,
              int arg_ = 0x0 )  [inline]
```

7.21.2.3 Instruction() [3/4]

```
Instruction::Instruction (
            CustomOp x,
            int arg_ = 0x0 )  [inline]
```

7.21.2.4 Instruction() [4/4]

```
Instruction::Instruction (
            PrimitiveOp x,
            int arg_ = 0x0 )  [inline]
```

7.21.3 Member Function Documentation

7.21.3.1 as()

```
template<typename t >
t Instruction::as ( ) const  [inline]
```

Get as type t

Returns

7.21.3.2 getArg()

```
int Instruction::getArg ( ) const  [inline]
```

Return the argument (an int)

Returns

7.21.3.3 is()

```
template<typename t >
bool Instruction::is ( ) const  [inline]
```

Template to check if this instruction is holding type t

Returns

7.21.3.4 is_a() [1/2]

```
template<typename T >
bool Instruction::is_a (
             const T x ) const  [inline]
```

compare the instruction types (ignores the arg)

7.21.3.5 is_a() [2/2]

```
template<typename T , typename...  Ts>
bool Instruction::is_a (
             T x,
             Ts...  args ) const  [inline]
```

Variadic checking of whether this is a given op type

Parameters

x	

Returns

7.21.3.6 operator==()

```
bool Instruction::operator== (
             const Instruction & i ) const  [inline]
```

7.21.4 Member Data Documentation

7.21.4.1 arg

```
int Instruction::arg
```

7.21.4.2 op

```
std::variant<BuiltinOp, CustomOp, PrimitiveOp> Instruction::op
```

The documentation for this class was generated from the following file:

- src/VirtualMachine/Instruction.h

7.22 IntegerizedStack Class Reference

```
#include <IntegerizedStack.h>
```

Public Member Functions

- IntegerizedStack (value_t v=0)
- value_t pop ()
- value_t pop (value_t modulus)
- void push (value_t x)
- void push (value_t x, value_t modulus)
- value_t get_value ()
- bool empty () const
- void operator= (value_t z)
- void operator-= (value_t x)
- void operator+= (value_t x)

Protected Attributes

- value_t value

7.22.1 Constructor & Destructor Documentation

7.22.1.1 IntegerizedStack()

```
IntegerizedStack::IntegerizedStack (
            value_t v = 0 )  [inline]
```

7.22.2 Member Function Documentation

7.22.2.1 empty()

```
bool IntegerizedStack::empty ( ) const  [inline]
```

7.22.2.2 get_value()

```
value_t IntegerizedStack::get_value ( )  [inline]
```

7.22.2.3 operator+=()

```
void IntegerizedStack::operator+= (
            value_t x )  [inline]
```

7.22.2.4 operator-=()

```
void IntegerizedStack::operator-= (
            value_t x )  [inline]
```

7.22.2.5 operator=()

```
void IntegerizedStack::operator= (
            value_t z )  [inline]
```

7.22.2.6 pop() [1/2]

```
value_t IntegerizedStack::pop ( )  [inline]
```

7.22.2.7 pop() [2/2]

```
value_t IntegerizedStack::pop (
            value_t modulus )  [inline]
```

7.22.2.8 push() [1/2]

```
void IntegerizedStack::push (
            value_t x )  [inline]
```

7.22.2.9 push() [2/2]

```
void IntegerizedStack::push (
            value_t x,
            value_t modulus )  [inline]
```

7.22.3 Member Data Documentation

7.22.3.1 value

```
value_t IntegerizedStack::value  [protected]
```

The documentation for this class was generated from the following file:

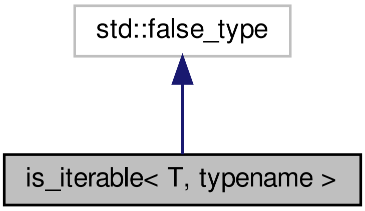
- src/IntegerizedStack.h

7.23 is_iterable< T, typename > Struct Template Reference

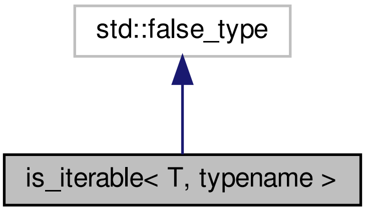
Converts our own time format to ms, which is what Fleet's time utilities use The time format we accept is #+(.#+)[smhd] where shmd specifies seconds, minutes, hours days.

```
#include <Miscellaneous.h>
```

Inheritance diagram for is_iterable< T, typename >:



Collaboration diagram for is_iterable< T, typename >:



7.23.1 Detailed Description

template<typename T, typename = void>
struct is_iterable< T, typename >

Converts our own time format to ms, which is what Fleet's time utilities use The time format we accept is #+(.#+)[smhd] where shmd specifies seconds, minutes, hours days.

```
    Time conversions for fleet
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

time_t convert_time(std::string& s) {
    // specila case of s="0" will be allowed
    if(s == "0") return 0;

    // else we must specify a unit
    double multiplier; // for default multiplier of 1 is seconds
    switch(s.at(s.length()-1)) {
        case 's': multiplier = 1000; break;
        case 'm': multiplier = 60*1000; break;
        case 'h': multiplier = 60*60*1000; break;
        case 'd': multiplier = 60*60*24*1000; break;
        default:
            CERR "*** Unknown time specifier: " << s.at(s.length()-1) << " in " << s << ". Did you
       forget a unit?" ENDL;
            assert(0);
```

```
    }

    double t = std::stod(s.substr(0,s.length()-1)); // all but the last character

    return (unsigned long)(t*multiplier); // note this effectively rounds to the nearest second

}

// Python-like pass statements
#define pass


/*
```

Template magic to check if a class is iterable ∼∼∼

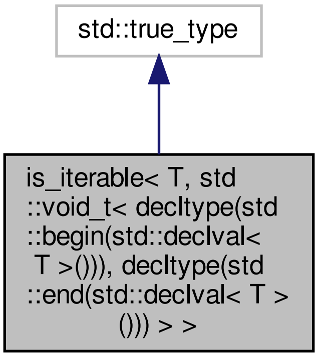
The documentation for this struct was generated from the following file:

- src/Miscellaneous.h

7.24 is_iterable< T, std::void_t< decltype(std::begin(std::declval< T >())), decltype(std::end(std::declval< T >())) > > Struct Template Reference

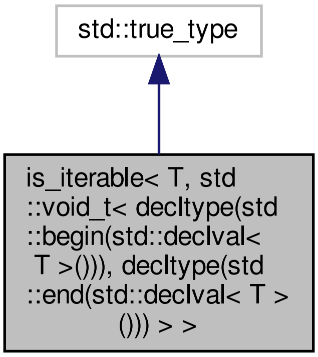
```
#include <Miscellaneous.h>
```

Inheritance diagram for is_iterable< T, std::void_t< decltype(std::begin(std::declval< T >())), decltype(std::end(std::declval< T >())) > >:


std::true_type
is_iterable< T, std ::void_t< decltype(std ::begin(std::declval< T >())), decltype(std ::end(std::declval< T > ())) > >

Collaboration diagram for is_iterable< T, std::void_t< decltype(std::begin(std::declval< T >())), decltype(std:↩

:end(std::declval< T >())) > >:


std::true_type
is_iterable< T, std ::void_t< decltype(std ::begin(std::declval< T >())), decltype(std ::end(std::declval< T > ())) > >

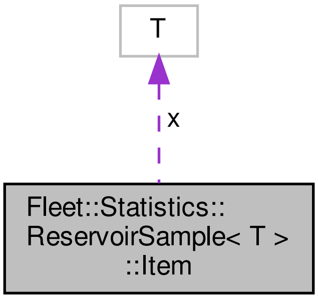
The documentation for this struct was generated from the following file:

- src/Miscellaneous.h

7.25 Fleet::Statistics::ReservoirSample< T >::Item Class Reference

```
#include <ReservoirSample.h>
```

Collaboration diagram for Fleet::Statistics::ReservoirSample< T >::Item:



Public Member Functions

- Item (T x_, double r_, double lw_=0.0)
- bool operator< (const Item &b) const
- bool operator== (const Item &b) const

Public Attributes

- T x
- const double r
- const double lw
- const double lv

7.25.1 Detailed Description

template<typename T>
class Fleet::Statistics::ReservoirSample< T >::Item

Author

piantado

Date

29/01/20

7.25.2 Constructor & Destructor Documentation

7.25.2.1 Item()

```
template<typename T>
Fleet::Statistics::ReservoirSample< T >::Item::Item (
             T x_,
             double r_,
             double lw_ = 0.0 )  [inline]
```

7.25.3 Member Function Documentation

7.25.3.1 operator<()

```
template<typename T>
bool Fleet::Statistics::ReservoirSample< T >::Item::operator< (
             const Item & b ) const  [inline]
```

7.25.3.2 operator==()

```
template<typename T>
bool Fleet::Statistics::ReservoirSample< T >::Item::operator== (
             const Item & b ) const  [inline]
```

7.25.4 Member Data Documentation

7.25.4.1 lv

```
template<typename T>
const double Fleet::Statistics::ReservoirSample< T >::Item::lv
```

7.25.4.2 lw

```
template<typename T>
const double Fleet::Statistics::ReservoirSample< T >::Item::lw
```

7.25.4.3 r

```
template<typename T>
const double Fleet::Statistics::ReservoirSample< T >::Item::r
```

7.25.4.4 x

```
template<typename T>
T Fleet::Statistics::ReservoirSample< T >::Item::x
```

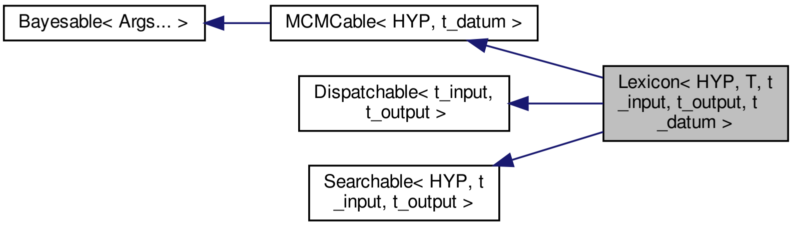
The documentation for this class was generated from the following file:

- src/Statistics/ReservoirSample.h

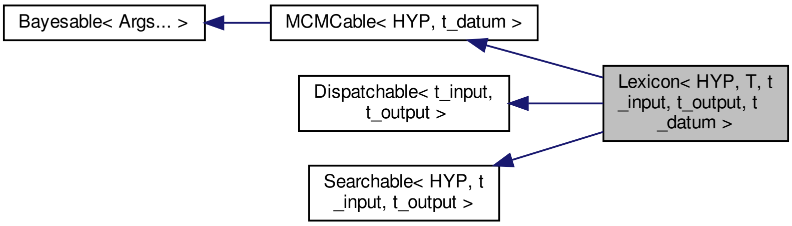
7.26 Lexicon< HYP, T, t_input, t_output, t_datum > Class Template Reference

```
#include <Lexicon.h>
```

Inheritance diagram for Lexicon< HYP, T, t_input, t_output, t_datum >:


Bayesable< Args... >
MCMCable< HYP, t_datum >
Dispatchable< t_input, t_output >
Lexicon< HYP, T, t _input, t_output, t _datum >
Searchable< HYP, t _input, t_output >

Collaboration diagram for Lexicon< HYP, T, t_input, t_output, t_datum >:


Bayesable< Args... >
MCMCable< HYP, t_datum >
Dispatchable< t_input, t_output >
Lexicon< HYP, T, t _input, t_output, t _datum >
Searchable< HYP, t _input, t_output >

Public Member Functions

- Lexicon (size_t n)
- Lexicon ()
- virtual std::string string () const
- virtual std::string parseable () const
- virtual size_t hash () const

    Default hash function.
- virtual bool operator== (const HYP &l) const
- bool has_valid_indices () const
- bool check_reachable () const
- virtual void push_program (Program &s, short j)
- virtual vmstatus_t dispatch_custom (Instruction i, VirtualMachinePool< t_input, t_output > ∗pool, Virtual↩ MachineState< t_input, t_output > ∗vms, Dispatchable< t_input, t_output > ∗loader)
- virtual HYP copy_and_complete () const
- virtual double compute_prior ()

    Compute the prior – defaultly not defined.
- virtual std::pair< HYP, double > propose () const
- virtual HYP restart () const
- int neighbors () const
- HYP make_neighbor (int k) const
- bool is_evaluable () const
- virtual DiscreteDistribution< t_output > call (const t_input x, const t_output err)=0

Public Attributes

- std::vector< T > factors

Additional Inherited Members

7.26.1 Detailed Description

template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum = default_datum<t_input, t_↩ output>>
class Lexicon< HYP, T, t_input, t_output, t_datum >

Author

piantado

Date

29/01/20

7.26.2 Constructor & Destructor Documentation

7.26.2.1 Lexicon() [1/2]

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
Lexicon< HYP, T, t_input, t_output, t_datum >::Lexicon (
             size_t n )  [inline]
```

7.26.2.2 Lexicon() [2/2]

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
Lexicon< HYP, T, t_input, t_output, t_datum >::Lexicon ( )  [inline]
```

7.26.3 Member Function Documentation

7.26.3.1 call()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
virtual DiscreteDistribution<t_output> Lexicon< HYP, T, t_input, t_output, t_datum >::call (
            const t_input x,
            const t_output err )  [pure virtual]
```

Implemented in MyHypothesis.

7.26.3.2 check_reachable()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
bool Lexicon< HYP, T, t_input, t_output, t_datum >::check_reachable ( ) const  [inline]
```

Check if the last factor call everything else transitively (e.g. are we "wasting" factors) We do this by making a graph of what factors call which others and then computing the transitive closure.

Returns

7.26.3.3 compute_prior()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
virtual double Lexicon< HYP, T, t_input, t_output, t_datum >::compute_prior ( )  [inline],
[virtual]
```

Compute the prior – defaultly not defined.

Implements Bayesable< Args... >.

Reimplemented in MyHypothesis.

7.26.3.4 copy_and_complete()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
virtual HYP Lexicon< HYP, T, t_input, t_output, t_datum >::copy_and_complete ( ) const  [inline],
[virtual]
```

7.26.3.5 dispatch_custom()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
virtual vmstatus_t Lexicon< HYP, T, t_input, t_output, t_datum >::dispatch_custom (
             Instruction i,
             VirtualMachinePool< t_input, t_output > * pool,
             VirtualMachineState< t_input, t_output > * vms,
             Dispatchable< t_input, t_output > * loader ) [inline], [virtual]
```

Implements Dispatchable< t_input, t_output >.

7.26.3.6 has_valid_indices()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
bool Lexicon< HYP, T, t_input, t_output, t_datum >::has_valid_indices ( ) const [inline]
```

A lexicon has valid indices if calls to op_RECURSE, op_MEM_RECURSE, op_SAFE_RECURSE, and op_SAF↩ E_MEM_RECURSE all have arguments that are less than the size. (So this places no restrictions on the calling earlier factors)

Returns

7.26.3.7 hash()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
virtual size_t Lexicon< HYP, T, t_input, t_output, t_datum >::hash ( ) const [inline], [virtual]
```

Default hash function.

Hash a Lexicon by hashing each part

Returns

Implements Bayesable< Args... >.

7.26.3.8 is_evaluable()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
bool Lexicon< HYP, T, t_input, t_output, t_datum >::is_evaluable ( ) const  [inline], [virtual]
```

Implements Searchable< HYP, t_input, t_output >.

7.26.3.9 make_neighbor()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
HYP Lexicon< HYP, T, t_input, t_output, t_datum >::make_neighbor (
             int k ) const  [inline], [virtual]
```

Implements Searchable< HYP, t_input, t_output >.

7.26.3.10 neighbors()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
int Lexicon< HYP, T, t_input, t_output, t_datum >::neighbors ( ) const  [inline], [virtual]
```

Implements Searchable< HYP, t_input, t_output >.

7.26.3.11 operator==()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
virtual bool Lexicon< HYP, T, t_input, t_output, t_datum >::operator== (
             const HYP & l ) const  [inline], [virtual]
```

Equality checks equality on each part

Parameters

l	

Returns

Implements MCMCable< HYP, t_datum >.

7.26.3.12 parseable()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
virtual std::string Lexicon< HYP, T, t_input, t_output, t_datum >::parseable ( ) const  [inline],
[virtual]
```

Convert to a parseable format (using a delimiter for each factor)

Returns

7.26.3.13 propose()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
virtual std::pair<HYP,double> Lexicon< HYP, T, t_input, t_output, t_datum >::propose ( )
const  [inline], [virtual]
```

This proposal guarantees that there will be at least one factor that is proposed to. To do this, we draw random numbers on 2∗∗factors.size()-1 and then use the bits of that integer to determine which factors to propose to.

Returns

Implements MCMCable< HYP, t_datum >.

7.26.3.14 push_program()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
virtual void Lexicon< HYP, T, t_input, t_output, t_datum >::push_program (
             Program & s,
             short j )  [inline], [virtual]
```

Put factor j onto program s

Parameters

s	
j	

Implements Dispatchable< t_input, t_output >.

7.26.3.15 restart()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
virtual HYP Lexicon< HYP, T, t_input, t_output, t_datum >::restart ( ) const  [inline], [virtual]
```

Implements MCMCable< HYP, t_datum >.

7.26.3.16 string()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
virtual std::string Lexicon< HYP, T, t_input, t_output, t_datum >::string ( ) const  [inline],
[virtual]
```

AConvert a lexicon to a string – defaultly includes all arguments.

Returns

Implements Bayesable< Args... >.

7.26.4 Member Data Documentation

7.26.4.1 factors

```
template<typename HYP, typename T, typename t_input, typename t_output, typename t_datum =
default_datum<t_input, t_output>>
std::vector<T> Lexicon< HYP, T, t_input, t_output, t_datum >::factors
```

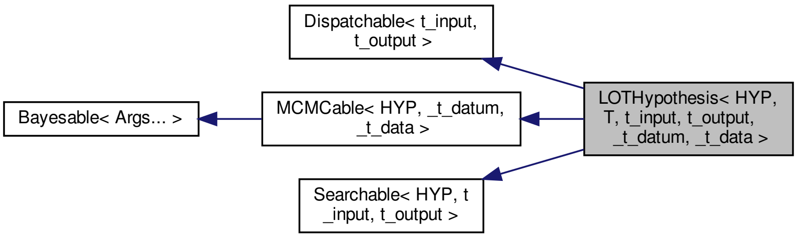
The documentation for this class was generated from the following file:

- src/Hypotheses/Lexicon.h

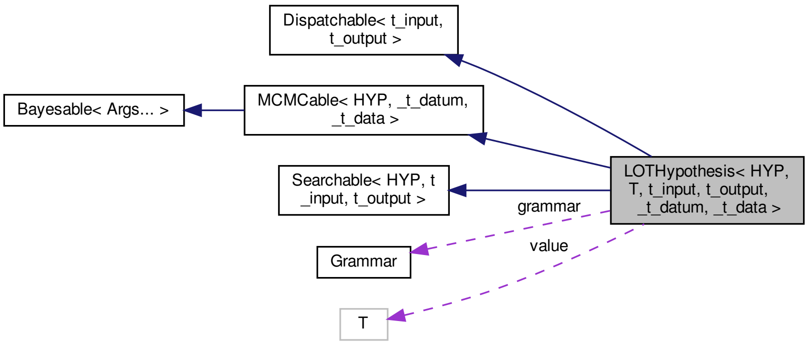
7.27 LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data > Class Template Reference

```
#include <LOTHypothesis.h>
```

Inheritance diagram for LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >:


Dispatchable< t_input, t_output >
Bayesable< Args... >
MCMCable< HYP, _t_datum, _t_data >
LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >
Searchable< HYP, t _input, t_output >

Collaboration diagram for LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >:


Dispatchable< t_input, t_output >
Bayesable< Args... >
MCMCable< HYP, _t_datum, _t_data >
Searchable< HYP, t _input, t_output >
LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >
grammar
value
Grammar
T

Public Types

- typedef Bayesable< _t_datum, _t_data >::t_data t_data
- typedef Bayesable< _t_datum, _t_data >::t_datum t_datum

Public Member Functions

- LOTHypothesis (Grammar ∗g=nullptr)
- LOTHypothesis (Grammar ∗g, T &&x)
- LOTHypothesis (Grammar ∗g, T &x)
- LOTHypothesis (Grammar ∗g, std::string s)
- virtual std::pair< HYP, double > propose () const
- virtual HYP restart () const

- void set_value (T &v)
- void set_value (T &&v)
- virtual double compute_prior ()

    Compute the prior – defaultly not defined.
- virtual double compute_single_likelihood (const t_datum &datum)
- virtual void push_program (Program &s, short k=0)
- virtual DiscreteDistribution< t_output > call (const t_input x, const t_output err, Dispatchable< t_input, t_↩ output > ∗loader, unsigned long max_steps=2048, unsigned long max_outputs=256, double minlp=-10.0)
- virtual DiscreteDistribution< t_output > call (const t_input x, const t_output err)
- auto operator() (const t_input x, const t_output err)
- virtual t_output callOne (const t_input x, const t_output err, Dispatchable< t_input, t_output > ∗loader=nullptr)
- virtual std::string string () const
- virtual std::string parseable () const
- virtual size_t hash () const

    Default hash function.
- virtual bool operator== (const HYP &h) const
- virtual vmstatus_t dispatch_custom (Instruction i, VirtualMachinePool< t_input, t_output > ∗pool, Virtual↩ MachineState< t_input, t_output > ∗vms, Dispatchable< t_input, t_output > ∗loader)
- virtual HYP copy_and_complete () const
- virtual int neighbors () const
- virtual HYP make_neighbor (int k) const
- virtual bool is_evaluable () const

Public Attributes

- Grammar ∗ grammar
- T value

Static Public Attributes

- static const size_t MAX_NODES = 64

7.27.1 Member Typedef Documentation

7.27.1.1 t_data

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
typedef Bayesable<_t_datum,_t_data>::t_data LOTHypothesis< HYP, T, t_input, t_output, _t_↩
datum, _t_data >::t_data
```

7.27.1.2 t_datum

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
typedef Bayesable<_t_datum,_t_data>::t_datum LOTHypothesis< HYP, T, t_input, t_output, _t_↩
datum, _t_data >::t_datum
```

7.27.2 Constructor & Destructor Documentation

7.27.2.1 LOTHypothesis() [1/4]

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::LOTHypothesis (
              Grammar * g = nullptr )  [inline]
```

7.27.2.2 LOTHypothesis() [2/4]

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::LOTHypothesis (
              Grammar * g,
              T && x )  [inline]
```

7.27.2.3 LOTHypothesis() [3/4]

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::LOTHypothesis (
              Grammar * g,
              T & x )  [inline]
```

7.27.2.4 LOTHypothesis() [4/4]

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::LOTHypothesis (
              Grammar * g,
              std::string s )  [inline]
```

7.27.3 Member Function Documentation

7.27.3.1 call() [1/2]

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual DiscreteDistribution<t_output> LOTHypothesis< HYP, T, t_input, t_output, _t_datum, ↩
_t_data >::call (
             const t_input x,
             const t_output err,
             Dispatchable< t_input, t_output > * loader,
             unsigned long max_steps = 2048,
             unsigned long max_outputs = 256,
             double minlp = -10.0 )  [inline], [virtual]
```

7.27.3.2 call() [2/2]

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual DiscreteDistribution<t_output> LOTHypothesis< HYP, T, t_input, t_output, _t_datum, ↩
_t_data >::call (
             const t_input x,
             const t_output err )  [inline], [virtual]
```

7.27.3.3 callOne()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual t_output LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::callOne (
             const t_input x,
             const t_output err,
             Dispatchable< t_input, t_output > * loader = nullptr )  [inline], [virtual]
```

7.27.3.4 compute_prior()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual double LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::compute_prior (
)  [inline], [virtual]
```

Compute the prior – defaultly not defined.

Implements Bayesable< Args... >.

Reimplemented in MyHypothesis.

7.27.3.5 compute_single_likelihood()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual double LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::compute_single↩
_likelihood (
            const t_datum & datum )  [inline], [virtual]
```

Reimplemented in MyHypothesis, and MyHypothesis.

7.27.3.6 copy_and_complete()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual HYP LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::copy_and_complete (
) const  [inline], [virtual]
```

7.27.3.7 dispatch_custom()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual vmstatus_t LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::dispatch_↩
custom (
            Instruction i,
            VirtualMachinePool< t_input, t_output > * pool,
            VirtualMachineState< t_input, t_output > * vms,
            Dispatchable< t_input, t_output > * loader )  [inline], [virtual]
```

Implements Dispatchable< t_input, t_output >.

Reimplemented in InnerHypothesis, and MyHypothesis.

7.27.3.8 hash()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual size_t LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::hash ( ) const
[inline], [virtual]
```

Default hash function.

Implements Bayesable< Args... >.

7.27.3.9 is_evaluable()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual bool LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::is_evaluable ( )
const [inline], [virtual]
```

Implements Searchable< HYP, t_input, t_output >.

7.27.3.10 make_neighbor()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual HYP LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::make_neighbor (
             int k ) const [inline], [virtual]
```

Implements Searchable< HYP, t_input, t_output >.

7.27.3.11 neighbors()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual int LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::neighbors ( ) const
[inline], [virtual]
```

Implements Searchable< HYP, t_input, t_output >.

7.27.3.12 operator()()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
auto LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::operator() (
             const t_input x,
             const t_output err ) [inline]
```

7.27.3.13 operator==()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual bool LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::operator== (
             const HYP & h ) const [inline], [virtual]
```

Implements MCMCable< HYP, _t_datum, _t_data >.

7.27.3.14 parseable()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual std::string LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::parseable (
) const [inline], [virtual]
```

7.27.3.15 propose()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual std::pair<HYP,double> LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >↩
::propose ( ) const [inline], [virtual]
```

Default proposal is rational-rules style regeneration.

Returns

Implements MCMCable< HYP, _t_datum, _t_data >.

Reimplemented in InnerHypothesis.

7.27.3.16 push_program()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual void LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::push_program (
             Program & s,
             short k = 0 ) [inline], [virtual]
```

Implements Dispatchable< t_input, t_output >.

7.27.3.17 restart()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual HYP LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::restart ( ) const
[inline], [virtual]
```

This is used to restart chains, sampling from prior

Returns

Implements MCMCable< HYP, _t_datum, _t_data >.

7.27.3.18 set_value() [1/2]

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
void LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::set_value (
             T & v )  [inline]
```

7.27.3.19 set_value() [2/2]

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
void LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::set_value (
             T && v )  [inline]
```

7.27.3.20 string()

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
virtual std::string LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::string ( )
const  [inline], [virtual]
```

Implements Bayesable< Args... >.

7.27.4 Member Data Documentation

7.27.4.1 grammar

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
Grammar* LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::grammar
```

7.27.4.2 MAX_NODES

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
const size_t LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::MAX_NODES = 64
[static]
```

7.27.4.3 value

```
template<typename HYP, typename T, typename t_input, typename t_output, typename _t_datum =
default_datum<t_input, t_output>, typename _t_data = std::vector<_t_datum>>
T LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >::value
```

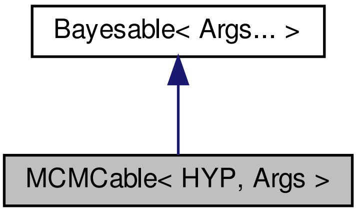
The documentation for this class was generated from the following file:

- src/Hypotheses/LOTHypothesis.h

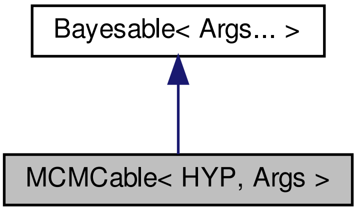
7.28 MCMCable< HYP, Args > Class Template Reference

```
#include <MCMCable.h>
```

Inheritance diagram for MCMCable< HYP, Args >:



Collaboration diagram for MCMCable< HYP, Args >:



Public Member Functions

- MCMCable ()
- virtual std::pair< HYP, double > propose () const =0
- virtual HYP restart () const =0
- virtual bool operator== (const HYP &h) const =0

Additional Inherited Members

7.28.1 Detailed Description

template<typename HYP, typename ... Args>
class MCMCable< HYP, Args >

Author

steven piantadosi

Date

03/02/20

7.28.2 Constructor & Destructor Documentation

7.28.2.1 MCMCable()

```
template<typename HYP, typename ...  Args>
MCMCable< HYP, Args >::MCMCable ( )  [inline]
```

7.28.3 Member Function Documentation

7.28.3.1 operator==()

```
template<typename HYP, typename ...  Args>
virtual bool MCMCable< HYP, Args >::operator== (
             const HYP & h ) const  [pure virtual]
```

Implemented in GrammarHypothesis< HYP, t_datum, t_data >, LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >, LOTHypothesis< InnerHypothesis, Node, S, S >, LOTHypothesis< MyHypothesis, Node, Object, bool >, LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >, Lexicon< HYP, T, t_input, t_output, t_datum >, and Lexicon< MyHypothesis, InnerHypothesis, S, S >.

7.28.3.2 propose()

```
template<typename HYP, typename ...  Args>
virtual std::pair<HYP,double> MCMCable< HYP, Args >::propose ( ) const  [pure virtual]
```

Implemented in Lexicon< HYP, T, t_input, t_output, t_datum >, Lexicon< MyHypothesis, InnerHypothesis, S, S >, GrammarHypothesis< HYP, t_datum, t_data >, InnerHypothesis, LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >, LOTHypothesis< InnerHypothesis, Node, S, S >, LOTHypothesis< MyHypothesis, Node, Object, bool >, and LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >.

7.28.3.3 restart()

```
template<typename HYP, typename ...  Args>
virtual HYP MCMCable< HYP, Args >::restart ( ) const  [pure virtual]
```

Implemented in Lexicon< HYP, T, t_input, t_output, t_datum >, Lexicon< MyHypothesis, InnerHypothesis, S, S >, GrammarHypothesis< HYP, t_datum, t_data >, LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >, LOTHypothesis< InnerHypothesis, Node, S, S >, LOTHypothesis< MyHypothesis, Node, Object, bool >, and LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >.

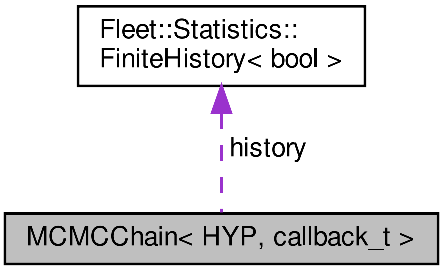
The documentation for this class was generated from the following file:

- src/Hypotheses/Interfaces/MCMCable.h

7.29 MCMCChain< HYP, callback_t > Class Template Reference

```
#include <MCMCChain.h>
```

Collaboration diagram for MCMCChain< HYP, callback_t >:



Public Member Functions

- MCMCChain (HYP &h0, typename HYP::t_data ∗d, callback_t &cb)
- MCMCChain (HYP &&h0, typename HYP::t_data ∗d, callback_t &cb)
- MCMCChain (HYP &h0, typename HYP::t_data ∗d, callback_t ∗cb=nullptr)
- MCMCChain (HYP &&h0, typename HYP::t_data ∗d, callback_t ∗cb=nullptr)
- MCMCChain (const MCMCChain &m)
- MCMCChain (MCMCChain &&m)
- virtual ∼MCMCChain ()
- HYP & getCurrent ()
- void runOnCurrent ()
- const HYP & getMax ()
- void run (Control ctl)
- void run ()
- double acceptance_ratio ()
- double at_temperature (double t)

Public Attributes

- HYP current
- std::mutex current_mutex
- HYP::t_data ∗ data
- HYP themax
- callback_t ∗ callback
- bool returnmax
- unsigned long samples
- unsigned long proposals
- unsigned long acceptances
- unsigned long steps_since_improvement
- std::atomic< double > temperature
- Fleet::Statistics::FiniteHistory< bool > history

7.29.1 Constructor & Destructor Documentation

7.29.1.1 MCMCChain() [1/6]

```
template<typename HYP , typename callback_t >
MCMCChain< HYP, callback_t >::MCMCChain (
            HYP & h0,
            typename HYP::t_data * d,
            callback_t & cb ) [inline]
```

7.29.1.2 MCMCChain() [2/6]

```
template<typename HYP , typename callback_t >
MCMCChain< HYP, callback_t >::MCMCChain (
            HYP && h0,
            typename HYP::t_data * d,
            callback_t & cb ) [inline]
```

7.29.1.3 MCMCChain() [3/6]

```
template<typename HYP , typename callback_t >
MCMCChain< HYP, callback_t >::MCMCChain (
            HYP & h0,
            typename HYP::t_data * d,
            callback_t * cb = nullptr ) [inline]
```

7.29.1.4 MCMCChain() [4/6]

```
template<typename HYP , typename callback_t >
MCMCChain< HYP, callback_t >::MCMCChain (
             HYP && h0,
             typename HYP::t_data * d,
             callback_t * cb = nullptr )  [inline]
```

7.29.1.5 MCMCChain() [5/6]

```
template<typename HYP , typename callback_t >
MCMCChain< HYP, callback_t >::MCMCChain (
             const MCMCChain< HYP, callback_t > & m )  [inline]
```

7.29.1.6 MCMCChain() [6/6]

```
template<typename HYP , typename callback_t >
MCMCChain< HYP, callback_t >::MCMCChain (
             MCMCChain< HYP, callback_t > && m )  [inline]
```

7.29.1.7 ∼MCMCChain()

```
template<typename HYP , typename callback_t >
virtual MCMCChain< HYP, callback_t >::∼MCMCChain ( )  [inline], [virtual]
```

7.29.2 Member Function Documentation

7.29.2.1 acceptance_ratio()

```
template<typename HYP , typename callback_t >
double MCMCChain< HYP, callback_t >::acceptance_ratio ( )  [inline]
```

Get my acceptance ratio

Returns

7.29.2.2 at_temperature()

```
template<typename HYP , typename callback_t >
double MCMCChain< HYP, callback_t >::at_temperature (
             double t )  [inline]
```

Return my current posterior at a given temperature t

Parameters

t	

Returns

7.29.2.3 getCurrent()

```
template<typename HYP , typename callback_t >
HYP& MCMCChain< HYP, callback_t >::getCurrent ( )  [inline]
```

get a reference to the current value

Returns

7.29.2.4 getMax()

```
template<typename HYP , typename callback_t >
const HYP& MCMCChain< HYP, callback_t >::getMax ( )  [inline]
```

7.29.2.5 run() [1/2]

```
template<typename HYP , typename callback_t >
void MCMCChain< HYP, callback_t >::run (
             Control ctl )  [inline]
```

Run MCMC according to the control parameters passed in. NOTE: ctl cannot be passed by reference.

Parameters

ctl	

7.29.2.6 run() [2/2]

```
template<typename HYP , typename callback_t >
void MCMCChain< HYP, callback_t >::run ( )  [inline]
```

Run forever

7.29.2.7 runOnCurrent()

```
template<typename HYP , typename callback_t >
void MCMCChain< HYP, callback_t >::runOnCurrent ( )  [inline]
```

This is called by the constructor to compute the posterior and callback for an initial h0

7.29.3 Member Data Documentation

7.29.3.1 acceptances

```
template<typename HYP , typename callback_t >
unsigned long MCMCChain< HYP, callback_t >::acceptances
```

7.29.3.2 callback

```
template<typename HYP , typename callback_t >
callback_t* MCMCChain< HYP, callback_t >::callback
```

7.29.3.3 current

```
template<typename HYP , typename callback_t >
HYP MCMCChain< HYP, callback_t >::current
```

7.29.3.4 current_mutex

```
template<typename HYP , typename callback_t >
std::mutex MCMCChain< HYP, callback_t >::current_mutex  [mutable]
```

7.29.3.5 data

```
template<typename HYP , typename callback_t >
HYP::t_data* MCMCChain< HYP, callback_t >::data
```

7.29.3.6 history

```
template<typename HYP , typename callback_t >
Fleet::Statistics::FiniteHistory<bool> MCMCChain< HYP, callback_t >::history
```

7.29.3.7 proposals

```
template<typename HYP , typename callback_t >
unsigned long MCMCChain< HYP, callback_t >::proposals
```

7.29.3.8 returnmax

```
template<typename HYP , typename callback_t >
bool MCMCChain< HYP, callback_t >::returnmax
```

7.29.3.9 samples

```
template<typename HYP , typename callback_t >
unsigned long MCMCChain< HYP, callback_t >::samples
```

7.29.3.10 steps_since_improvement

```
template<typename HYP , typename callback_t >
unsigned long MCMCChain< HYP, callback_t >::steps_since_improvement
```

7.29.3.11 temperature

```
template<typename HYP , typename callback_t >
std::atomic<double> MCMCChain< HYP, callback_t >::temperature
```

7.29.3.12 themax

```
template<typename HYP , typename callback_t >
HYP MCMCChain< HYP, callback_t >::themax
```

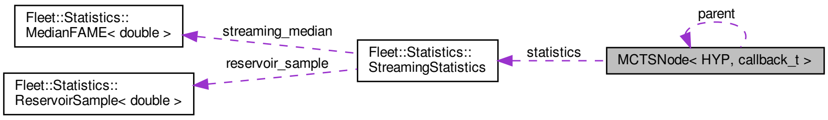
The documentation for this class was generated from the following file:

- src/Inference/MCMCChain.h

7.30 MCTSNode< HYP, callback_t > Class Template Reference

```
#include <MCTS.h>
```

Collaboration diagram for MCTSNode< HYP, callback_t >:



Public Types

- enum ScoringType { ScoringType::SAMPLE, ScoringType::UCBMAX, ScoringType::MEDIAN }

  MCTS Implementation.

Public Member Functions

- MCTSNode (MCTSNode ∗par, HYP &v)
- MCTSNode (double ex, HYP &h0, callback_t ∗cb, typename HYP::t_data ∗d, ScoringType st=ScoringType↩ ::SAMPLE)
- MCTSNode (const MCTSNode &m)=delete
- MCTSNode (MCTSNode &&m)
- size_t size () const
- void initialize ()
- void print (std::ostream &o, const int depth, const bool sort) const
- void print (const bool sort=true) const
- void printerr (const bool sort=true) const
- void print (const char ∗filename, const bool sort=true) const
- double score () const
- size_t open_children () const
- size_t best_child_index () const
- void add_sample (const double v, const size_t num=1)
- void operator<< (double v)
- virtual void playout (Control inner_ctl)
- void add_child_nodes ()
- void search (Control ctl, Control inner_ctl)
- void parallel_search (Control ctl, Control inner_ctl)
- void search_one (Control inner_ctl)

Public Attributes

- std::vector< MCTSNode > children
- unsigned long nvisits
- bool open
- callback_t ∗ callback
- double explore
- std::mutex child_mutex
- std::mutex stats_mutex
- Fleet::Statistics::StreamingStatistics statistics
- HYP::t_data ∗ data
- MCTSNode ∗ parent
- HYP value
- ScoringType scoring_type

7.30.1 Member Enumeration Documentation

7.30.1.1 ScoringType

```
template<typename HYP, typename callback_t>
enum MCTSNode::ScoringType  [strong]
```

MCTS Implementation.

Enumerator

SAMPLE	
UCBMAX	
MEDIAN	

7.30.2 Constructor & Destructor Documentation

7.30.2.1 MCTSNode() [1/4]

```
template<typename HYP, typename callback_t>
MCTSNode< HYP, callback_t >::MCTSNode (
            MCTSNode< HYP, callback_t > * par,
            HYP & v )  [inline]
```

7.30.2.2 MCTSNode() [2/4]

```
template<typename HYP, typename callback_t>
MCTSNode< HYP, callback_t >::MCTSNode (
            double ex,
            HYP & h0,
            callback_t * cb,
            typename HYP::t_data * d,
            ScoringType st = ScoringType::SAMPLE )  [inline]
```

7.30.2.3 MCTSNode() [3/4]

```
template<typename HYP, typename callback_t>
MCTSNode< HYP, callback_t >::MCTSNode (
            const MCTSNode< HYP, callback_t > & m )  [delete]
```

7.30.2.4 MCTSNode() [4/4]

```
template<typename HYP, typename callback_t>
MCTSNode< HYP, callback_t >::MCTSNode (
            MCTSNode< HYP, callback_t > && m )  [inline]
```

7.30.3 Member Function Documentation

7.30.3.1 add_child_nodes()

```
template<typename HYP, typename callback_t>
void MCTSNode< HYP, callback_t >::add_child_nodes ( )  [inline]
```

7.30.3.2 add_sample()

```
template<typename HYP, typename callback_t>
void MCTSNode< HYP, callback_t >::add_sample (
            const double v,
            const size_t num = 1 )  [inline]
```

7.30.3.3 best_child_index()

```
template<typename HYP, typename callback_t>
size_t MCTSNode< HYP, callback_t >::best_child_index ( ) const  [inline]
```

7.30.3.4 initialize()

```
template<typename HYP, typename callback_t>
void MCTSNode< HYP, callback_t >::initialize ( )  [inline]
```

7.30.3.5 open_children()

```
template<typename HYP, typename callback_t>
size_t MCTSNode< HYP, callback_t >::open_children ( ) const  [inline]
```

7.30.3.6 operator<<()

```
template<typename HYP, typename callback_t>
void MCTSNode< HYP, callback_t >::operator<< (
             double v )  [inline]
```

7.30.3.7 parallel_search()

```
template<typename HYP, typename callback_t>
void MCTSNode< HYP, callback_t >::parallel_search (
             Control ctl,
             Control inner_ctl )  [inline]
```

7.30.3.8 playout()

```
template<typename HYP, typename callback_t>
virtual void MCTSNode< HYP, callback_t >::playout (
             Control inner_ctl )  [inline], [virtual]
```

7.30.3.9 print() [1/3]

```
template<typename HYP, typename callback_t>
void MCTSNode< HYP, callback_t >::print (
             std::ostream & o,
             const int depth,
             const bool sort ) const  [inline]
```

7.30.3.10 print() [2/3]

```
template<typename HYP, typename callback_t>
void MCTSNode< HYP, callback_t >::print (
             const bool sort = true ) const  [inline]
```

7.30.3.11 print() [3/3]

```
template<typename HYP, typename callback_t>
void MCTSNode< HYP, callback_t >::print (
             const char * filename,
             const bool sort = true ) const  [inline]
```

7.30.3.12 printerr()

```
template<typename HYP, typename callback_t>
void MCTSNode< HYP, callback_t >::printerr (
             const bool sort = true ) const  [inline]
```

7.30.3.13 score()

```
template<typename HYP, typename callback_t>
double MCTSNode< HYP, callback_t >::score ( ) const  [inline]
```

7.30.3.14 search()

```
template<typename HYP, typename callback_t>
void MCTSNode< HYP, callback_t >::search (
             Control ctl,
             Control inner_ctl )  [inline]
```

7.30.3.15 search_one()

```
template<typename HYP, typename callback_t>
void MCTSNode< HYP, callback_t >::search_one (
             Control inner_ctl )  [inline]
```

7.30.3.16 size()

```
template<typename HYP, typename callback_t>
size_t MCTSNode< HYP, callback_t >::size ( ) const  [inline]
```

7.30.4 Member Data Documentation

7.30.4.1 callback

```
template<typename HYP, typename callback_t>
callback_t* MCTSNode< HYP, callback_t >::callback
```

7.30.4.2 child_mutex

```
template<typename HYP, typename callback_t>
std::mutex MCTSNode< HYP, callback_t >::child_mutex  [mutable]
```

7.30.4.3 children

```
template<typename HYP, typename callback_t>
std::vector<MCTSNode> MCTSNode< HYP, callback_t >::children
```

7.30.4.4 data

```
template<typename HYP, typename callback_t>
HYP::t_data* MCTSNode< HYP, callback_t >::data
```

7.30.4.5 explore

```
template<typename HYP, typename callback_t>
double MCTSNode< HYP, callback_t >::explore
```

7.30.4.6 nvisits

```
template<typename HYP, typename callback_t>
unsigned long MCTSNode< HYP, callback_t >::nvisits
```

7.30.4.7 open

```
template<typename HYP, typename callback_t>
bool MCTSNode< HYP, callback_t >::open
```

7.30.4.8 parent

```
template<typename HYP, typename callback_t>
MCTSNode* MCTSNode< HYP, callback_t >::parent
```

7.30.4.9 scoring_type

```
template<typename HYP, typename callback_t>
ScoringType MCTSNode< HYP, callback_t >::scoring_type
```

7.30.4.10 statistics

```
template<typename HYP, typename callback_t>
Fleet::Statistics::StreamingStatistics MCTSNode< HYP, callback_t >::statistics
```

7.30.4.11 stats_mutex

```
template<typename HYP, typename callback_t>
std::mutex MCTSNode< HYP, callback_t >::stats_mutex  [mutable]
```

7.30.4.12 value

```
template<typename HYP, typename callback_t>
HYP MCTSNode< HYP, callback_t >::value
```

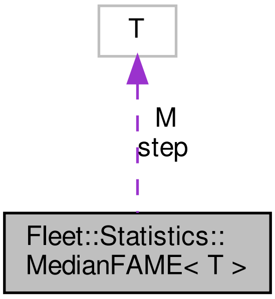
The documentation for this class was generated from the following file:

- src/Inference/MCTS.h

7.31 Fleet::Statistics::MedianFAME< T > Class Template Reference

```
#include <MedianFAME.h>
```

Collaboration diagram for Fleet::Statistics::MedianFAME< T >:



Public Member Functions

- MedianFAME ()
- T median () const
- void add (T x)
- void operator<< (T x)

Public Attributes

- T M
- size_t n
- T step

7.31.1 Detailed Description

template<typename T>
class Fleet::Statistics::MedianFAME< T >

Author

steven piantadosi

Date

03/02/20

7.31.2 Constructor & Destructor Documentation

7.31.2.1 MedianFAME()

```
template<typename T>
Fleet::Statistics::MedianFAME< T >::MedianFAME ( )  [inline]
```

7.31.3 Member Function Documentation

7.31.3.1 add()

```
template<typename T>
void Fleet::Statistics::MedianFAME< T >::add (
             T x )  [inline]
```

7.31.3.2 median()

```
template<typename T>
T Fleet::Statistics::MedianFAME< T >::median ( ) const  [inline]
```

7.31.3.3 operator<<()

```
template<typename T>
void Fleet::Statistics::MedianFAME< T >::operator<< (
             T x )  [inline]
```

7.31.4 Member Data Documentation

7.31.4.1 M

```
template<typename T>
T Fleet::Statistics::MedianFAME< T >::M
```

7.31.4.2 n

```
template<typename T>
size_t Fleet::Statistics::MedianFAME< T >::n
```

7.31.4.3 step

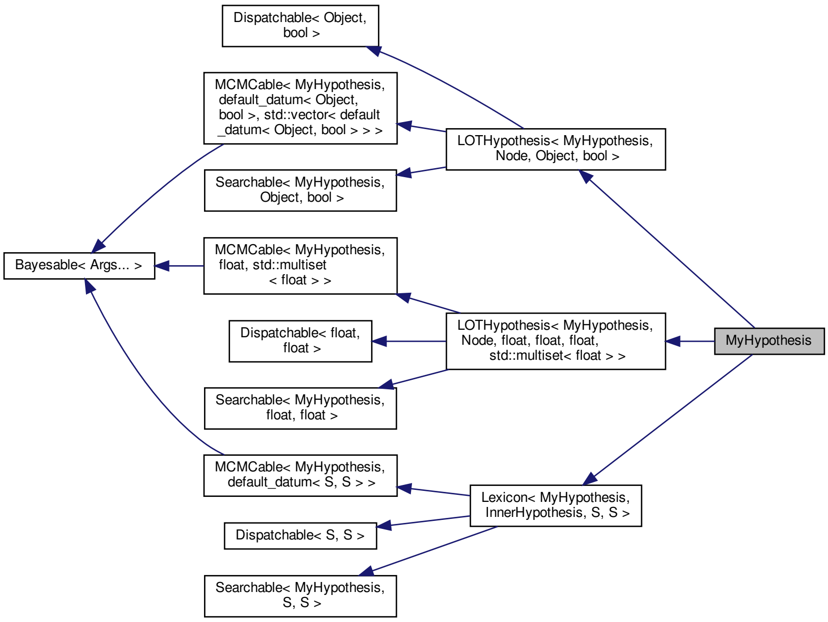
```
template<typename T>
T Fleet::Statistics::MedianFAME< T >::step
```

The documentation for this class was generated from the following file:

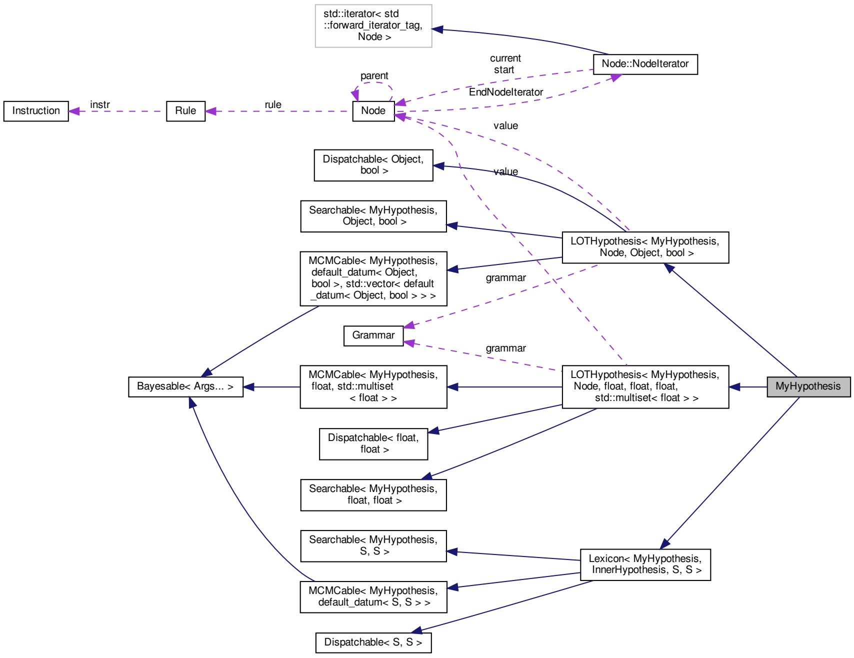
- src/Statistics/MedianFAME.h

7.32 MyHypothesis Class Reference

Inheritance diagram for MyHypothesis:


Dispatchable< Object, bool >
MCMCable< MyHypothesis, default_datum< Object, bool >, std::vector< default _datum< Object, bool > > >
LOTHypothesis< MyHypothesis, Node, Object, bool >
Searchable< MyHypothesis, Object, bool >
Bayesable< Args... >
MCMCable< MyHypothesis, float, std::multiset < float > >
Dispatchable< float, float >
LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >
MyHypothesis
Searchable< MyHypothesis, float, float >
MCMCable< MyHypothesis, default_datum< S, S > >
Lexicon< MyHypothesis, InnerHypothesis, S, S >
Dispatchable< S, S >
Searchable< MyHypothesis, S, S >

Collaboration diagram for MyHypothesis:


std::iterator< std ::forward_iterator_tag, Node >
Node::NodeIterator
current
start
parent
EndNodeIterator
Instruction
instr
Rule
rule
Node
value
Dispatchable< Object, bool >
value
Searchable< MyHypothesis, Object, bool >
LOTHypothesis< MyHypothesis, Node, Object, bool >
MCMCable< MyHypothesis, default_datum< Object, bool >, std::vector< default _datum< Object, bool > > >
grammar
Grammar
grammar
Bayesable< Args... >
MCMCable< MyHypothesis, float, std::multiset < float > >
LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >
MyHypothesis
Dispatchable< float, float >
Searchable< MyHypothesis, float, float >
Searchable< MyHypothesis, S, S >
Lexicon< MyHypothesis, InnerHypothesis, S, S >
MCMCable< MyHypothesis, default_datum< S, S > >
Dispatchable< S, S >

Public Types

- using Super = Lexicon< MyHypothesis, InnerHypothesis, S, S >
- using Super = LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >
- using Super = LOTHypothesis< MyHypothesis, Node, Object, bool >

Public Member Functions

- MyHypothesis ()
- MyHypothesis (const MyHypothesis &h)
- virtual double compute_prior ()

  Compute the prior – defaultly not defined.
- virtual DiscreteDistribution< S > call (const S x, const S err)
- virtual double compute_single_likelihood (const t_datum &datum)

  Compute the likelihood of a single data point.
- double compute_likelihood (const t_data &data, const double breakout=-infinity)

  Compute the likelihood of a collection of data, by calling compute_single_likelihood on each. This stops if our likelihood falls below breakout.
- void print (std::string prefix="")
- virtual double compute_likelihood (const t_data &data, const double breakout=-infinity)

  Compute the likelihood of a collection of data, by calling compute_single_likelihood on each. This stops if our likelihood falls below breakout.
- vmstatus_t dispatch_custom (Instruction i, VirtualMachinePool< float, float > *pool, VirtualMachineState< float, float > *vms, Dispatchable< float, float > *loader)
- double compute_single_likelihood (const t_datum &x)

  Compute the likelihood of a single data point.

Additional Inherited Members

7.32.1 Detailed Description

```
    This is a global variable that provides a convenient way to wrap our primitives
    where we can pair up a function with a name, and pass that as a constructor
    to the grammar. We need a tuple here because Primitive has a bunch of template
    types to handle thee function it has, so each is actually a different type.
    This must be defined before we import Fleet because Fleet does some template
    magic internally
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
```

```
std::tuple PRIMITIVES = {
    Primitive("and(%s,%s)",    +[](bool a, bool b) -> bool { return (a and b); }, 2.0), //
       optional specification of prior weight (default=1.0)
    Primitive("or(%s,%s)",     +[](bool a, bool b) -> bool { return (a or b); }),
    Primitive("not(%s)",       +[](bool a)         -> bool { return (not a); }),
    // that + is really insane, but is needed to convert a lambda to a function pointer

    Primitive("red(%s)",       +[](Object x)       -> bool { return x.
      color == Color::Red; }),
    Primitive("green(%s)",     +[](Object x)       -> bool { return x.
      color == Color::Green; }),
    Primitive("blue(%s)",      +[](Object x)       -> bool { return x.
      color == Color::Blue; }),

    Primitive("square(%s)",    +[](Object x)       -> bool { return x.
      shape == Shape::Square; }),
    Primitive("triangle(%s)",  +[](Object x)       -> bool { return x.
      shape == Shape::Triangle; }),
    Primitive("circle(%s)",    +[](Object x)       -> bool { return x.
      shape == Shape::Circle; }),
```

```
    // but we also have to add a rule for the BuiltinOp that access x, our argument
    Builtin::X<Object>("x", 10.0)
};

// Includes critical files. Also defines some variables (mcts_steps, explore, etc.) that get processed from
       argv


/*
```

Define a class for handling my specific hypotheses and data. Everything is defaultly a PCFG prior and regeneration proposals, but I have to define a likelihood ~~~~~~

7.32.2 Member Typedef Documentation

7.32.2.1 Super [1/3]

```
using MyHypothesis::Super = LOTHypothesis<MyHypothesis,Node,float,float, float, std::multiset<float>
>
```

7.32.2.2 Super [2/3]

```
using MyHypothesis::Super = LOTHypothesis<MyHypothesis,Node,Object,bool>
```

7.32.2.3 Super [3/3]

```
using MyHypothesis::Super = Lexicon<MyHypothesis, InnerHypothesis, S, S>
```

7.32.3 Constructor & Destructor Documentation

7.32.3.1 MyHypothesis() [1/2]

```
MyHypothesis::MyHypothesis ( )  [inline]
```

7.32.3.2 MyHypothesis() [2/2]

```
MyHypothesis::MyHypothesis (
               const MyHypothesis & h )  [inline]
```

7.32.4 Member Function Documentation

7.32.4.1 call()

```
virtual DiscreteDistribution<S> MyHypothesis::call (
             const S x,
             const S err )  [inline], [virtual]
```

Implements Lexicon< MyHypothesis, InnerHypothesis, S, S >.

7.32.4.2 compute_likelihood() [1/2]

```
virtual double MyHypothesis::compute_likelihood (
             const t_data & data,
             const double breakout = -infinity )  [inline], [virtual]
```

Compute the likelihood of a collection of data, by calling compute_single_likelihood on each. This stops if our likelihood falls below breakout.

Parameters

data	
breakout	

Returns

Reimplemented from Bayesable< Args... >.

7.32.4.3 compute_likelihood() [2/2]

```
double MyHypothesis::compute_likelihood (
             const t_data & data,
             const double breakout = -infinity )  [inline], [virtual]
```

Compute the likelihood of a collection of data, by calling compute_single_likelihood on each. This stops if our likelihood falls below breakout.

Parameters

data	
breakout	

Returns

Reimplemented from Bayesable< Args... >.

7.32.4.4 compute_prior()

```
virtual double MyHypothesis::compute_prior ( )  [inline], [virtual]
```

Compute the prior – defaultly not defined.

Reimplemented from Lexicon< MyHypothesis, InnerHypothesis, S, S >.

7.32.4.5 compute_single_likelihood() [1/2]

```
double MyHypothesis::compute_single_likelihood (
             const t_datum & datum )  [inline], [virtual]
```

Compute the likelihood of a single data point.

Parameters

datum	

Reimplemented from LOTHypothesis< MyHypothesis, Node, Object, bool >.

7.32.4.6 compute_single_likelihood() [2/2]

```
virtual double MyHypothesis::compute_single_likelihood (
             const t_datum & datum )  [inline], [virtual]
```

Compute the likelihood of a single data point.

Parameters

datum	

Reimplemented from LOTHypothesis< MyHypothesis, Node, Object, bool >.

7.32.4.7 dispatch_custom()

```
vmstatus_t MyHypothesis::dispatch_custom (
            Instruction i,
            VirtualMachinePool< float, float > * pool,
            VirtualMachineState< float, float > * vms,
            Dispatchable< float, float > * loader ) [inline], [virtual]
```

Reimplemented from LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >.

7.32.4.8 print()

```
void MyHypothesis::print (
            std::string prefix = "" ) [inline], [virtual]
```

Reimplemented from Bayesable< Args... >.

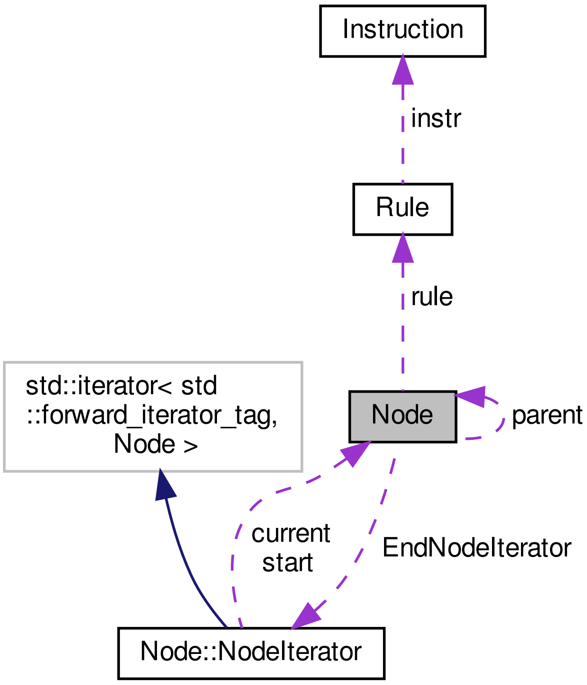
The documentation for this class was generated from the following file:

- Models/FormalLanguageTheory-Complex/Main.cpp

7.33 Node Class Reference

```
#include <Node.h>
```

Collaboration diagram for Node:



Classes

- class NodeIterator

Public Member Functions

- Node (const Rule ∗r=nullptr, double _lp=0.0, bool cr=true)
- Node (const Node &n)
- Node (Node &&n)
- Node & child (const size_t i)
- const Node & child (const size_t i) const
- nonterminal_t type (const size_t i) const
- const size_t nchildren () const
- void fill ()
- void operator= (const Node &n)
- void operator= (Node &&n)
- bool operator< (const Node &n) const
- NodeIterator begin () const
- NodeIterator end () const
- Node ∗ left_descend () const
- void fix_child_info ()
- nonterminal_t nt () const
- Node & operator[ ] (const size_t i)
- const Node & operator[ ] (const size_t i) const
- void set_child (size_t i, Node &n)
- void set_child (size_t i, Node &&n)
- void set_to (Node &n)
- void set_to (Node &&n)
- bool is_null () const
- template< typename T >
  T sum (std::function< T(const Node &)> &f) const
- template< typename T >
  T sum (T(∗f)(const Node &)) const
- void map (const std::function< void(Node &)> &f)
- void map_const (const std::function< void(const Node &)> &f) const
- void rmap_const (const std::function< void(const Node &)> &f) const
- virtual size_t count () const
- virtual size_t count (const Node &n) const
- virtual bool is_root () const
- virtual bool is_terminal () const
- virtual size_t count_terminals () const
- virtual bool is_complete () const
- virtual Node ∗ get_nth (int n, std::function< int(const Node &)> &f)
- virtual Node ∗ get_nth (int n)
- virtual std::string string () const
- virtual std::string parseable () const
- virtual size_t program_size () const
- virtual void linearize (Program &ops) const
- virtual bool operator== (const Node &n) const
- virtual size_t hash (size_t depth=0) const

Public Attributes

- Node ∗ parent
- size_t pi
- const Rule ∗ rule
- double lp
- bool can_resample

Static Public Attributes

- static NodeIterator EndNodeIterator = NodeIterator(nullptr)

Protected Attributes

- std::vector< Node > children

7.33.1 Constructor & Destructor Documentation

7.33.1.1 Node() [1/3]

```
Node::Node (
            const Rule ∗ r = nullptr,
            double _lp = 0.0,
            bool cr = true )  [inline]
```

7.33.1.2 Node() [2/3]

```
Node::Node (
            const Node & n )  [inline]
```

7.33.1.3 Node() [3/3]

```
Node::Node (
            Node && n )  [inline]
```

7.33.2 Member Function Documentation

7.33.2.1 begin()

```
NodeIterator Node::begin ( ) const  [inline]
```

7.33.2.2 child() [1/2]

```
Node& Node::child (
            const size_t i )  [inline]
```

Get a reference to my i'th child

Parameters

i	

Returns

7.33.2.3 child() [2/2]

```
const Node& Node::child (
            const size_t i ) const  [inline]
```

Constant reference to the i'th child

Parameters

i	

Returns

7.33.2.4 count() [1/2]

```
virtual size_t Node::count ( ) const  [inline], [virtual]
```

How many nodes are below me?

Returns

7.33.2.5 count() [2/2]

```
virtual size_t Node::count (
            const Node & n ) const  [inline], [virtual]
```

How many nodes below me are equal to n?

Parameters

n	

Returns

7.33.2.6 count_terminals()

```
virtual size_t Node::count_terminals ( ) const  [inline], [virtual]
```

How many terminals are below me?

Returns

7.33.2.7 end()

```
NodeIterator Node::end ( ) const  [inline]
```

7.33.2.8 fill()

```
void Node::fill ( )  [inline]
```

Fill in all of my immediate children with Null nodes (via NullRule)

7.33.2.9 fix_child_info()

```
void Node::fix_child_info ( )  [inline]
```

Fix my immediate children's pointers to ensure that children's parent pointers and indices are correct.

7.33.2.10 get_nth() [1/2]

```
virtual Node* Node::get_nth (
             int n,
             std::function< int(const Node &)> & f )  [inline], [virtual]
```

Return a pointer to the n'th child satisfying f (f's output is cast to bool)

Parameters

n	
f	

Returns

7.33.2.11 get_nth() [2/2]

```
virtual Node* Node::get_nth (
              int n )  [inline], [virtual]
```

7.33.2.12 hash()

```
virtual size_t Node::hash (
              size_t depth = 0 ) const  [inline], [virtual]
```

Hash a tree by hashing the rule and everything below.

Parameters

depth	

Returns

7.33.2.13 is_complete()

```
virtual bool Node::is_complete ( ) const  [inline], [virtual]
```

A tree is complete if it contains no null nodes below it.

Returns

7.33.2.14 is_null()

```
bool Node::is_null ( ) const  [inline]
```

Am I a null node?

Returns

7.33.2.15 is_root()

```
virtual bool Node::is_root ( ) const  [inline], [virtual]
```

Am I a root node? I am if my parent is nullptr.

Returns

7.33.2.16 is_terminal()

```
virtual bool Node::is_terminal ( ) const  [inline], [virtual]
```

Am I a terminal? I am if I have no children.

Returns

7.33.2.17 left_descend()

```
Node* Node::left_descend ( ) const  [inline]
```

Go down a tree and find the leftmost child

Returns

7.33.2.18 linearize()

```
virtual void Node::linearize (
             Program & ops ) const  [inline], [virtual]
```

convert tree to a linear sequence of operations. To do this, we first linearize the kids, leaving their values as the top on the stack then we compute our value, remove our kids' values to clean up the stack, and push on our return the only fanciness is for if: here we will use the following layout <TOP of="" stack>=""> <bool> op_IF(xsize) X-branch JUMP(ysize) Y-branch

NOTE: Inline here lets gcc inline a few recursions of this function, which ends up speeding us up a bit (otherwise recursive inlining only happens at -O3) This optimization is why we do set max-inline-insns-recursive in Fleet.mk

Parameters

ops	

7.33.2.19 map()

```
void Node::map (
            const std::function< void(Node &)> & f )  [inline]
```

Apply f to me and everything below.

Parameters

f	

7.33.2.20 map_const()

```
void Node::map_const (
            const std::function< void(const Node &)> & f ) const  [inline]
```

7.33.2.21 nchildren()

```
const size_t Node::nchildren ( ) const  [inline]
```

How many children do I have?

Returns

7.33.2.22 nt()

```
nonterminal_t Node::nt ( ) const  [inline]
```

What nonterminal type do I return?

Returns

7.33.2.23 operator<()

```
bool Node::operator< (
             const Node & n ) const  [inline]
```

7.33.2.24 operator=() [1/2]

```
void Node::operator= (
             const Node & n )  [inline]
```

7.33.2.25 operator=() [2/2]

```
void Node::operator= (
             Node && n )  [inline]
```

7.33.2.26 operator==()

```
virtual bool Node::operator== (
             const Node & n ) const  [inline], [virtual]
```

Check equality between notes. Note this compares rule objects.

Parameters

n	

Returns

7.33.2.27 operator[]() [1/2]

```
Node& Node::operator[] (
             const size_t i )  [inline]
```

Index my i'th child

Parameters

i	

Returns

7.33.2.28 operator[]() [2/2]

```
const Node& Node::operator[] (
             const size_t i ) const  [inline]
```

7.33.2.29 parseable()

```
virtual std::string Node::parseable ( ) const  [inline], [virtual]
```

Create a string that can be parsed according to Grammar.expand_from_names

Returns

7.33.2.30 program_size()

```
virtual size_t Node::program_size ( ) const  [inline], [virtual]
```

How big of a program does this correspond to? This is mostly the number of nodes, except that short-circuit evaluation of IF makes things a little more complex.

Returns

7.33.2.31 rmap_const()

```
void Node::rmap_const (
             const std::function< void(const Node &)> & f ) const  [inline]
```

7.33.2.32 set_child() [1/2]

```
void Node::set_child (
             size_t i,
             Node & n )  [inline]
```

Set my child to n. NOTE: This one needs to be used, rather than accessing children directly, because we have to set parent pointers and indices.

Parameters

i	
n	

7.33.2.33 set_child() [2/2]

```
void Node::set_child (
             size_t i,
             Node && n )  [inline]
```

Child setter for move

Parameters

i	

7.33.2.34 set_to() [1/2]

```
void Node::set_to (
             Node & n )  [inline]
```

Set myself to n. This should be used so that parent pointers etc. can be updated.

Parameters

n	

7.33.2.35 set_to() [2/2]

```
void Node::set_to (
             Node && n )  [inline]
```

Set myself to n (move version)

7.33.2.36 string()

```
virtual std::string Node::string ( ) const  [inline], [virtual]
```

Convert a tree to a string, using each node's format.

Returns

7.33.2.37 sum() [1/2]

```
template<typename T >
T Node::sum (
             std::function< T(const Node &)> & f ) const  [inline]
```

Apply f to me and everything below me, adding up the result.

Parameters

f	

Returns

7.33.2.38 sum() [2/2]

```
template<typename T >
T Node::sum (
             T(*)(const Node &) f ) const  [inline]
```

Apply f to me and everything below me, adding up the result.

Parameters

f	

Returns

7.33.2.39 type()

```
nonterminal_t Node::type (
             const size_t i ) const  [inline]
```

Return the type of the i'th child

Parameters

i	

Returns

7.33.3 Member Data Documentation

7.33.3.1 can_resample

```
bool Node::can_resample
```

7.33.3.2 children

```
std::vector<Node> Node::children  [protected]
```

7.33.3.3 EndNodeIterator

```
Node::NodeIterator Node::EndNodeIterator = NodeIterator(nullptr)  [static]
```

7.33.3.4 lp

```
double Node::lp
```

7.33.3.5 parent

```
Node* Node::parent
```

7.33.3.6 pi

```
size_t Node::pi
```

7.33.3.7 rule

```
const Rule* Node::rule
```

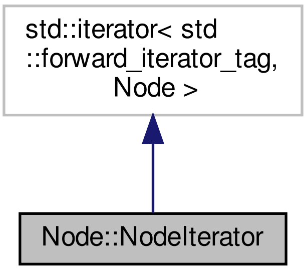
The documentation for this class was generated from the following file:

- src/Node.h

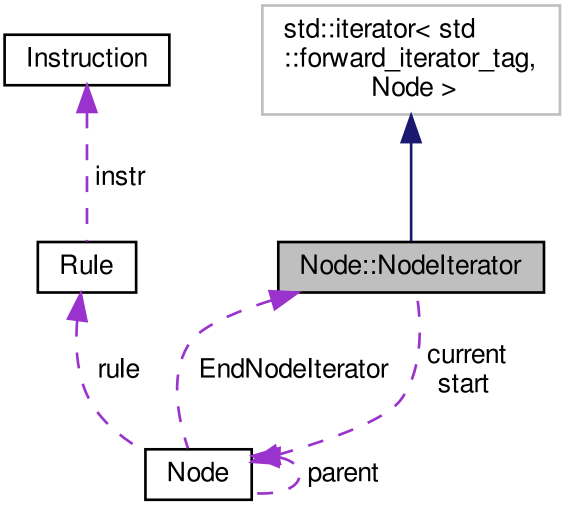
7.34 Node::NodeIterator Class Reference

```
#include <Node.h>
```

Inheritance diagram for Node::NodeIterator:



Collaboration diagram for Node::NodeIterator:



Public Member Functions

- NodeIterator (const Node ∗n)
- Node & operator∗ () const
- Node ∗ operator-> () const
- NodeIterator & operator++ (int blah)
- NodeIterator & operator++ ()
- NodeIterator & operator+ (size_t n)
- bool operator== (const NodeIterator &rhs)
- bool operator!= (const NodeIterator &rhs)

Protected Attributes

- Node ∗ current
- const Node ∗ start

7.34.1 Constructor & Destructor Documentation

7.34.1.1 NodeIterator()

```
Node::NodeIterator::NodeIterator (
             const Node * n )   [inline]
```

7.34.2 Member Function Documentation

7.34.2.1 operator"!=()

```
bool Node::NodeIterator::operator!= (
             const NodeIterator & rhs )   [inline]
```

7.34.2.2 operator∗()

```
Node& Node::NodeIterator::operator* ( ) const   [inline]
```

7.34.2.3 operator+()

```
NodeIterator& Node::NodeIterator::operator+ (
             size_t n )   [inline]
```

7.34.2.4 operator++() [1/2]

```
NodeIterator& Node::NodeIterator::operator++ (
             int blah )  [inline]
```

7.34.2.5 operator++() [2/2]

```
NodeIterator& Node::NodeIterator::operator++ ( )  [inline]
```

7.34.2.6 operator->()

```
Node* Node::NodeIterator::operator-> ( ) const  [inline]
```

7.34.2.7 operator==()

```
bool Node::NodeIterator::operator== (
             const NodeIterator & rhs )  [inline]
```

7.34.3 Member Data Documentation

7.34.3.1 current

```
Node* Node::NodeIterator::current  [protected]
```

7.34.3.2 start

```
const Node* Node::NodeIterator::start  [protected]
```

The documentation for this class was generated from the following file:

- src/Node.h

7.35 Object Struct Reference

Public Attributes

- Color color
- Shape shape

7.35.1 Member Data Documentation

7.35.1.1 color

```
Color Object::color
```

7.35.1.2 shape

```
Shape Object::shape
```

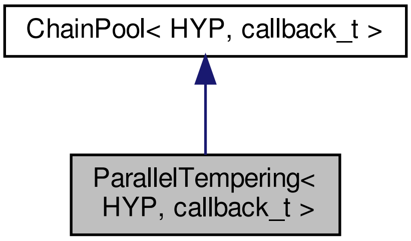
The documentation for this struct was generated from the following file:

- Models/RationalRules/Main.cpp

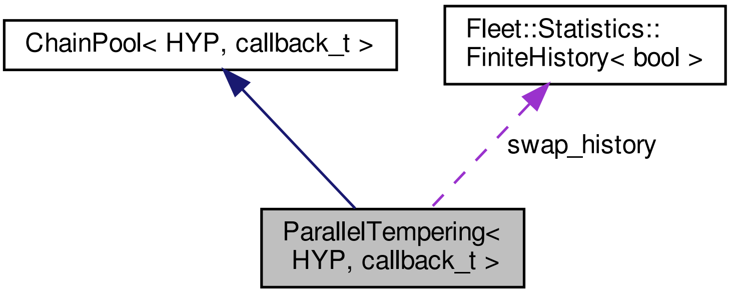
7.36 ParallelTempering< HYP, callback_t > Class Template Reference

```
#include <ParallelTempering.h>
```

Inheritance diagram for ParallelTempering< HYP, callback_t >:


ChainPool< HYP, callback_t >
ParallelTempering< HYP, callback_t >

Collaboration diagram for ParallelTempering< HYP, callback_t >:



Public Member Functions

- ParallelTempering (HYP &h0, typename HYP::t_data ∗d, callback_t &cb, std::initializer_list< double > t, bool allcallback=true)
- ParallelTempering (HYP &h0, typename HYP::t_data ∗d, callback_t &cb, unsigned long n, double maxT, bool allcallback=true)
- ParallelTempering (HYP &h0, std::vector< typename HYP::t_data > &datas, std::vector< callback_t > &cb)
- ∼ParallelTempering ()
- void __swapper_thread (time_ms swap_every)
- void __adapter_thread (time_ms adapt_every)
- virtual void run (Control ctl)
- virtual void run (Control ctl, time_ms swap_every, time_ms adapt_every)
- void show_statistics ()
- double k (unsigned long t, double v, double t0)
- void adapt (double v=3, double t0=1000000)

Public Attributes

- std::vector< double > temperatures
- Fleet::Statistics::FiniteHistory< bool > ∗ swap_history
- bool is_temperature
- std::atomic< bool > terminate

Additional Inherited Members

7.36.1 Constructor & Destructor Documentation

7.36.1.1 ParallelTempering() [1/3]

```
template<typename HYP , typename callback_t >
ParallelTempering< HYP, callback_t >::ParallelTempering (
            HYP & h0,
            typename HYP::t_data * d,
            callback_t & cb,
            std::initializer_list< double > t,
            bool allcallback = true )  [inline]
```

7.36.1.2 ParallelTempering() [2/3]

```
template<typename HYP , typename callback_t >
ParallelTempering< HYP, callback_t >::ParallelTempering (
            HYP & h0,
            typename HYP::t_data * d,
            callback_t & cb,
            unsigned long n,
            double maxT,
            bool allcallback = true )  [inline]
```

7.36.1.3 ParallelTempering() [3/3]

```
template<typename HYP , typename callback_t >
ParallelTempering< HYP, callback_t >::ParallelTempering (
            HYP & h0,
            std::vector< typename HYP::t_data > & datas,
            std::vector< callback_t > & cb )  [inline]
```

7.36.1.4 ∼ParallelTempering()

```
template<typename HYP , typename callback_t >
ParallelTempering< HYP, callback_t >::∼ParallelTempering ( )  [inline]
```

7.36.2 Member Function Documentation

7.36.2.1 __adapter_thread()

```
template<typename HYP , typename callback_t >
void ParallelTempering< HYP, callback_t >::__adapter_thread (
            time_ms adapt_every )  [inline]
```

7.36.2.2 __swapper_thread()

```
template<typename HYP , typename callback_t >
void ParallelTempering< HYP, callback_t >::__swapper_thread (
             time_ms swap_every )  [inline]
```

7.36.2.3 adapt()

```
template<typename HYP , typename callback_t >
void ParallelTempering< HYP, callback_t >::adapt (
             double v = 3,
             double t0 = 1000000 )  [inline]
```

7.36.2.4 k()

```
template<typename HYP , typename callback_t >
double ParallelTempering< HYP, callback_t >::k (
             unsigned long t,
             double v,
             double t0 )  [inline]
```

7.36.2.5 run() [1/2]

```
template<typename HYP , typename callback_t >
virtual void ParallelTempering< HYP, callback_t >::run (
             Control ctl )  [inline], [virtual]
```

Reimplemented from ChainPool< HYP, callback_t >.

7.36.2.6 run() [2/2]

```
template<typename HYP , typename callback_t >
virtual void ParallelTempering< HYP, callback_t >::run (
             Control ctl,
             time_ms swap_every,
             time_ms adapt_every )  [inline], [virtual]
```

7.36.2.7 show_statistics()

```
template<typename HYP , typename callback_t >
void ParallelTempering< HYP, callback_t >::show_statistics ( )  [inline]
```

7.36.3 Member Data Documentation

7.36.3.1 is_temperature

```
template<typename HYP , typename callback_t >
bool ParallelTempering< HYP, callback_t >::is_temperature
```

7.36.3.2 swap_history

```
template<typename HYP , typename callback_t >
Fleet::Statistics::FiniteHistory<bool>* ParallelTempering< HYP, callback_t >::swap_history
```

7.36.3.3 temperatures

```
template<typename HYP , typename callback_t >
std::vector<double> ParallelTempering< HYP, callback_t >::temperatures
```

7.36.3.4 terminate

```
template<typename HYP , typename callback_t >
std::atomic<bool> ParallelTempering< HYP, callback_t >::terminate
```

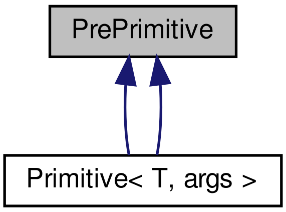
The documentation for this class was generated from the following file:

- src/Inference/ParallelTempering.h

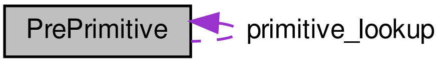
7.37 PrePrimitive Struct Reference

```
#include <Primitives.h>
```

Inheritance diagram for PrePrimitive:



Collaboration diagram for PrePrimitive:



Public Member Functions

- PrePrimitive ()

Public Attributes

- const PrimitiveOp op

Static Public Attributes

- static PrimitiveOp op_counter = 0
- static PrimitiveOp global_primitive_counter = 0
- static const size_t MAX_PRIMITIVES = 256
- static PrePrimitive ∗ primitive_lookup [MAX_PRIMITIVES]

7.37.1 Constructor & Destructor Documentation

7.37.1.1 PrePrimitive()

```
PrePrimitive::PrePrimitive ( )  [inline]
```

7.37.2 Member Data Documentation

7.37.2.1 global_primitive_counter

```
PrimitiveOp PrePrimitive::global_primitive_counter = 0  [static]
```

7.37.2.2 MAX_PRIMITIVES

```
const size_t PrePrimitive::MAX_PRIMITIVES = 256  [static]
```

7.37.2.3 op

```
const PrimitiveOp PrePrimitive::op
```

7.37.2.4 op_counter

```
PrimitiveOp PrePrimitive::op_counter = 0  [static]
```

7.37.2.5 primitive_lookup

```
PrePrimitive* PrePrimitive::primitive_lookup[MAX_PRIMITIVES]  [static]
```

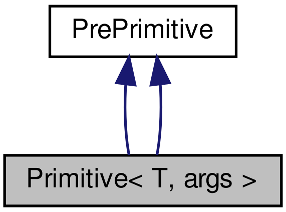
The documentation for this struct was generated from the following files:

- src/VirtualMachine/Primitives.h
- src/VirtualMachine/Primitives2.h

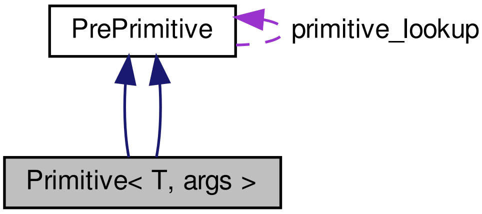
7.38 Primitive< T, args > Struct Template Reference

```
#include <Grammar.h>
```

Inheritance diagram for Primitive< T, args >:



Collaboration diagram for Primitive< T, args >:



Public Types

- typedef std::decay< typename HeadIfReferenceElseT< T, args... >::type >::type GrammarReturnType
- typedef std::decay< typename HeadIfReferenceElseT< T, args... >::type >::type GrammarReturnType

Public Member Functions

- Primitive (std::string fmt, T(∗f)(args...), double _p=1.0)
- template<typename V , typename P , typename L >
  vmstatus_t VMScall (V ∗vms, P ∗pool, L ∗loader)
- Primitive (std::string fmt, T(∗f)(args...), double _p=1.0)
- template<typename V >
  vmstatus_t call (V ∗vms)

Public Attributes

- std::string format
- T(∗ call)(args...)
- PrimitiveOp op
- double p

Additional Inherited Members

7.38.1 Member Typedef Documentation

7.38.1.1 GrammarReturnType [1/2]

```
template<typename T, typename...  args>
typedef std::decay<typename HeadIfReferenceElseT<T,args...>::type>::type Primitive< T, args
>::GrammarReturnType
```

7.38.1.2 GrammarReturnType [2/2]

```
template<typename T, typename...  args>
typedef std::decay<typename HeadIfReferenceElseT<T,args...>::type>::type Primitive< T, args
>::GrammarReturnType
```

7.38.2 Constructor & Destructor Documentation

7.38.2.1 Primitive() [1/2]

```
template<typename T, typename...  args>
Primitive< T, args >::Primitive (
             std::string fmt,
             T(*)(args...)   f,
             double _p = 1.0 )   [inline]
```

7.38.2.2 Primitive() [2/2]

```
template<typename T, typename...  args>
Primitive< T, args >::Primitive (
             std::string fmt,
             T(*)(args...)   f,
             double _p = 1.0 )   [inline]
```

7.38.3 Member Function Documentation

7.38.3.1 call()

```
template<typename T, typename... args>
template<typename V >
vmstatus_t Primitive< T, args >::call (
             V * vms )  [inline]
```

7.38.3.2 VMScall()

```
template<typename T, typename... args>
template<typename V , typename P , typename L >
vmstatus_t Primitive< T, args >::VMScall (
             V * vms,
             P * pool,
             L * loader )  [inline]
```

7.38.4 Member Data Documentation

7.38.4.1 call

```
template<typename T, typename... args>
T(* Primitive< T, args >::call)(args...)
```

7.38.4.2 format

```
template<typename T, typename... args>
std::string Primitive< T, args >::format
```

7.38.4.3 op

```
template<typename T, typename... args>
PrimitiveOp Primitive< T, args >::op
```

7.38.4.4 p

```
template<typename T, typename...  args>
double Primitive< T, args >::p
```

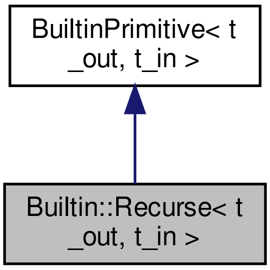
The documentation for this struct was generated from the following files:

- src/Grammar.h
- src/VirtualMachine/Primitives.h
- src/VirtualMachine/Primitives2.h

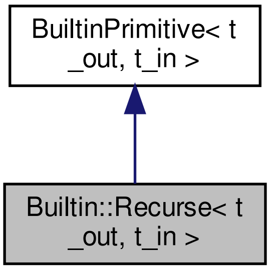
7.39 Builtin::Recurse< t_out, t_in > Struct Template Reference

```
#include <Builtins.h>
```

Inheritance diagram for Builtin::Recurse< t_out, t_in >:



Collaboration diagram for Builtin::Recurse< t_out, t_in >:



Public Member Functions

- Recurse (std::string fmt, double _p=1.0)

Additional Inherited Members

7.39.1 Constructor & Destructor Documentation

7.39.1.1 Recurse()

```
template<typename t_out , typename t_in >
Builtin::Recurse< t_out, t_in >::Recurse (
             std::string fmt,
             double _p = 1.0 )  [inline]
```

The documentation for this struct was generated from the following file:

- src/VirtualMachine/Builtins.h

7.40 Fleet::Statistics::ReservoirSample< T > Class Template Reference

```
#include <ReservoirSample.h>
```

Classes

- class Item

Public Member Functions

- ReservoirSample (size_t n, bool u=false)
- ReservoirSample (bool u=false)
- void set_reservoir_size (const size_t s) const
- size_t size () const
- T max ()
- T min ()
- auto begin ()
- auto end ()
- double best_posterior () const
- void add (T x, double lw=0.0)
- void operator<< (T x)
- T sample () const

Public Attributes

- std::multiset< Item > s
- std::multiset< T > vals
- size_t reservoir_size
- bool unique
- unsigned long N

7.40.1 Detailed Description

template<typename T>
class Fleet::Statistics::ReservoirSample< T >

Author

piantado

Date

29/01/20

7.40.2 Constructor & Destructor Documentation

7.40.2.1 ReservoirSample() [1/2]

```
template<typename T>
Fleet::Statistics::ReservoirSample< T >::ReservoirSample (
            size_t n,
            bool u = false )  [inline]
```

7.40.2.2 ReservoirSample() [2/2]

```
template<typename T>
Fleet::Statistics::ReservoirSample< T >::ReservoirSample (
            bool u = false )  [inline]
```

7.40.3 Member Function Documentation

7.40.3.1 add()

```
template<typename T>
void Fleet::Statistics::ReservoirSample< T >::add (
             T x,
             double lw = 0.0 )  [inline]
```

7.40.3.2 begin()

```
template<typename T>
auto Fleet::Statistics::ReservoirSample< T >::begin ( )  [inline]
```

7.40.3.3 best_posterior()

```
template<typename T>
double Fleet::Statistics::ReservoirSample< T >::best_posterior ( ) const  [inline]
```

What has the best posterior?

Returns

7.40.3.4 end()

```
template<typename T>
auto Fleet::Statistics::ReservoirSample< T >::end ( )  [inline]
```

7.40.3.5 max()

```
template<typename T>
T Fleet::Statistics::ReservoirSample< T >::max ( )  [inline]
```

7.40.3.6 min()

```
template<typename T>
T Fleet::Statistics::ReservoirSample< T >::min ( )  [inline]
```

7.40.3.7 operator<<()

```
template<typename T>
void Fleet::Statistics::ReservoirSample< T >::operator<< (
             T x )  [inline]
```

7.40.3.8 sample()

```
template<typename T>
T Fleet::Statistics::ReservoirSample< T >::sample ( ) const  [inline]
```

Return a sample from my vals (e.g. a sample of the samples I happen to have saved)

Returns

7.40.3.9 set_reservoir_size()

```
template<typename T>
void Fleet::Statistics::ReservoirSample< T >::set_reservoir_size (
             const size_t s ) const  [inline]
```

How big should the reservoir size be?

Parameters

s	

7.40.3.10 size()

```
template<typename T>
size_t Fleet::Statistics::ReservoirSample< T >::size ( ) const  [inline]
```

How many elements are currently stored?

Returns

7.40.4 Member Data Documentation

7.40.4.1 N

```
template<typename T>
unsigned long Fleet::Statistics::ReservoirSample< T >::N
```

7.40.4.2 reservoir_size

```
template<typename T>
size_t Fleet::Statistics::ReservoirSample< T >::reservoir_size
```

7.40.4.3 s

```
template<typename T>
std::multiset<Item> Fleet::Statistics::ReservoirSample< T >::s
```

7.40.4.4 unique

```
template<typename T>
bool Fleet::Statistics::ReservoirSample< T >::unique
```

7.40.4.5 vals

```
template<typename T>
std::multiset<T> Fleet::Statistics::ReservoirSample< T >::vals
```

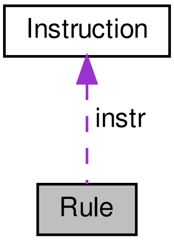
The documentation for this class was generated from the following file:

- src/Statistics/ReservoirSample.h

7.41 Rule Class Reference

```
#include <Rule.h>
```

Collaboration diagram for Rule:



Public Member Functions

- template<typename OPT >
  Rule (const nonterminal_t rt, const OPT o, const std::string fmt, std::initializer_list< nonterminal_t > c, double _p, const int arg=0)
- bool operator< (const Rule &r) const
- bool operator== (const Rule &r) const
- size_t get_hash () const
- bool is_terminal () const
- nonterminal_t type (size_t i) const
- std::string string ()

Public Attributes

- nonterminal_t nt
- Instruction instr
- std::string format
- size_t N
- double p

Protected Attributes

- nonterminal_t child_types [Fleet::MAX_CHILD_SIZE]
- std::size_t my_hash

7.41.1 Constructor & Destructor Documentation

7.41.1.1 Rule()

```
template<typename OPT >
Rule::Rule (
             const nonterminal_t rt,
             const OPT o,
             const std::string fmt,
             std::initializer_list< nonterminal_t > c,
             double _p,
             const int arg = 0 )  [inline]
```

7.41.2 Member Function Documentation

7.41.2.1 get_hash()

```
size_t Rule::get_hash ( ) const  [inline]
```

7.41.2.2 is_terminal()

```
bool Rule::is_terminal ( ) const  [inline]
```

A terminal rule has no children.

Returns

7.41.2.3 operator<()

```
bool Rule::operator< (
             const Rule & r ) const  [inline]
```

We sort rules so that they can be stored in arrays in a standard order. For enumeration, it's actually important that we sort them with the terminals first. So we sort first by terminals and then by probability (higher first).

Parameters

r	

Returns

7.41.2.4 operator==()

```
bool Rule::operator== (
             const Rule & r ) const  [inline]
```

Two rules are equal if they have the same instructions, nonterminal, format, children, and children types

Parameters

r	

Returns

7.41.2.5 string()

```
std::string Rule::string ( )  [inline]
```

7.41.2.6 type()

```
nonterminal_t Rule::type (
             size_t i ) const  [inline]
```

What type is my i'th child?

Parameters

i	

Returns

7.41.3 Member Data Documentation

7.41.3.1 child_types

```
nonterminal_t Rule::child_types[Fleet::MAX_CHILD_SIZE]  [protected]
```

7.41.3.2 format

```
std::string Rule::format
```

7.41.3.3 instr

```
Instruction Rule::instr
```

7.41.3.4 my_hash

```
std::size_t Rule::my_hash  [protected]
```

7.41.3.5 N

```
size_t Rule::N
```

7.41.3.6 nt

```
nonterminal_t Rule::nt
```

7.41.3.7 p

```
double Rule::p
```

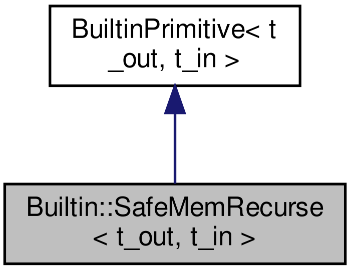
The documentation for this class was generated from the following file:

- src/Rule.h

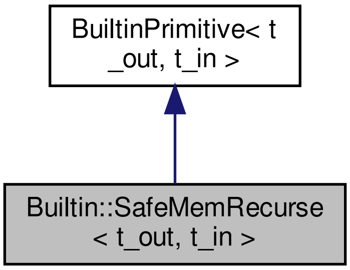
7.42 Builtin::SafeMemRecurse< t_out, t_in > Struct Template Reference

```
#include <Builtins.h>
```

Inheritance diagram for Builtin::SafeMemRecurse< t_out, t_in >:


BuiltinPrimitive< t _out, t_in >
Builtin::SafeMemRecurse < t_out, t_in >

Collaboration diagram for Builtin::SafeMemRecurse< t_out, t_in >:


BuiltinPrimitive< t _out, t_in >
Builtin::SafeMemRecurse < t_out, t_in >

Public Member Functions

- SafeMemRecurse (std::string fmt, double _p=1.0)

Additional Inherited Members

7.42.1 Constructor & Destructor Documentation

7.42.1.1 SafeMemRecurse()

```
template<typename t_out , typename t_in >
Builtin::SafeMemRecurse< t_out, t_in >::SafeMemRecurse (
             std::string fmt,
             double _p = 1.0 )  [inline]
```

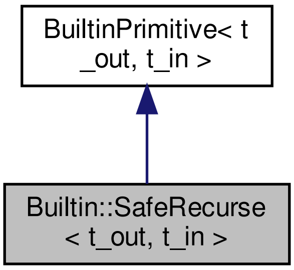
The documentation for this struct was generated from the following file:

- src/VirtualMachine/Builtins.h

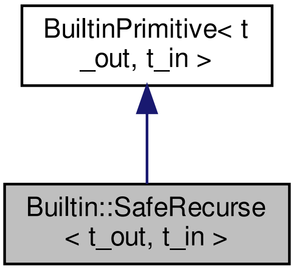
7.43 Builtin::SafeRecurse< t_out, t_in > Struct Template Reference

```
#include <Builtins.h>
```

Inheritance diagram for Builtin::SafeRecurse< t_out, t_in >:



Collaboration diagram for Builtin::SafeRecurse< t_out, t_in >:



Public Member Functions

- SafeRecurse (std::string fmt, double _p=1.0)

Additional Inherited Members

7.43.1 Constructor & Destructor Documentation

7.43.1.1 SafeRecurse()

```
template<typename t_out , typename t_in >
Builtin::SafeRecurse< t_out, t_in >::SafeRecurse (
            std::string fmt,
            double _p = 1.0 )  [inline]
```

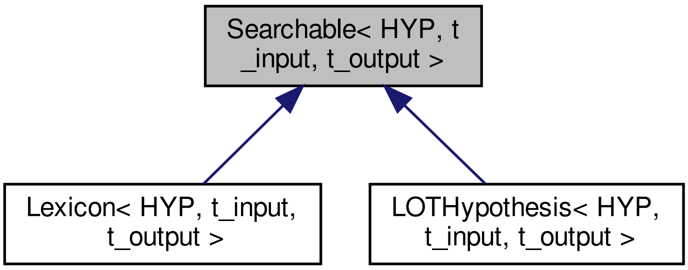
The documentation for this struct was generated from the following file:

- src/VirtualMachine/Builtins.h

7.44 Searchable< HYP, t_input, t_output > Class Template Reference

```
#include <Searchable.h>
```

Inheritance diagram for Searchable< HYP, t_input, t_output >:


Searchable< HYP, t _input, t_output >
Lexicon< HYP, t_input, t_output >
LOTHypothesis< HYP, t_input, t_output >

Public Member Functions

- virtual int neighbors () const =0
- virtual HYP make_neighbor (int k) const =0
- virtual bool is_evaluable () const =0

7.44.1 Detailed Description

**template<typename HYP, typename t_input, typename t_output>
class Searchable< HYP, t_input, t_output >**

Author

steven piantadosi

Date

03/02/20

7.44.2 Member Function Documentation

7.44.2.1 is_evaluable()

```
template<typename HYP, typename t_input, typename t_output>
virtual bool Searchable< HYP, t_input, t_output >::is_evaluable ( ) const  [pure virtual]
```

Implemented in Lexicon< HYP, T, t_input, t_output, t_datum >, Lexicon< MyHypothesis, InnerHypothesis, S, S >, LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >, LOTHypothesis< InnerHypothesis, Node, S, S >, LOTHypothesis< MyHypothesis, Node, Object, bool >, and LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >.

7.44.2.2 make_neighbor()

```
template<typename HYP, typename t_input, typename t_output>
virtual HYP Searchable< HYP, t_input, t_output >::make_neighbor (
             int k ) const  [pure virtual]
```

Implemented in Lexicon< HYP, T, t_input, t_output, t_datum >, Lexicon< MyHypothesis, InnerHypothesis, S, S >, LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >, LOTHypothesis< InnerHypothesis, Node, S, S >, LOTHypothesis< MyHypothesis, Node, Object, bool >, and LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >.

7.44.2.3 neighbors()

```
template<typename HYP, typename t_input, typename t_output>
virtual int Searchable< HYP, t_input, t_output >::neighbors ( ) const  [pure virtual]
```

Implemented in Lexicon< HYP, T, t_input, t_output, t_datum >, Lexicon< MyHypothesis, InnerHypothesis, S, S >, LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >, LOTHypothesis< InnerHypothesis, Node, S, S >, LOTHypothesis< MyHypothesis, Node, Object, bool >, and LOTHypothesis< MyHypothesis, Node, float, float, float, std::multiset< float > >.

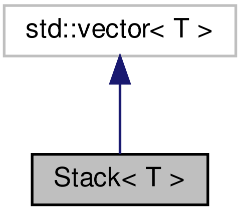
The documentation for this class was generated from the following file:

- src/Hypotheses/Interfaces/Searchable.h

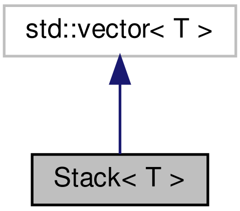
7.45 Stack< T > Class Template Reference

```
#include <Stack.h>
```

Inheritance diagram for Stack< T >:



Collaboration diagram for Stack< T >:



Public Member Functions

- Stack ()
- void push (const T &val)
- void pop ()
- T top ()
- T & topref ()

7.45.1 Constructor & Destructor Documentation

7.45.1.1 Stack()

```
template<typename T>
Stack< T >::Stack ( )  [inline]
```

7.45.2 Member Function Documentation

7.45.2.1 pop()

```
template<typename T>
void Stack< T >::pop ( ) [inline]
```

Remove the top element (returning void)

7.45.2.2 push()

```
template<typename T>
void Stack< T >::push (
             const T & val ) [inline]
```

Push val onto the stack

Parameters

val	

7.45.2.3 top()

```
template<typename T>
T Stack< T >::top ( ) [inline]
```

Return the top element.

Returns

7.45.2.4 topref()

```
template<typename T>
T& Stack< T >::topref ( ) [inline]
```

Get a reference to the top element (allowing in-place modification)

Returns

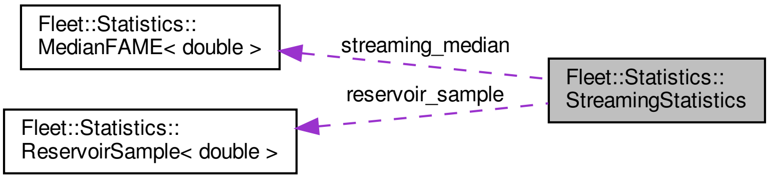
The documentation for this class was generated from the following file:

- src/Stack.h

7.46 Fleet::Statistics::StreamingStatistics Class Reference

```
#include <StreamingStatistics.h>
```

Collaboration diagram for Fleet::Statistics::StreamingStatistics:



Public Member Functions

- StreamingStatistics (size_t rs=1000)
- void operator= (StreamingStatistics &&s)
- void add (double x, double lw=0.0)
- void operator<< (double x)
- double sample () const
- double median () const
- void print () const
- double p_exceeds_median (const StreamingStatistics &q) const

Public Attributes

- double min
- double max
- double sum
- double lse
- double N
- MedianFAME< double > streaming_median
- ReservoirSample< double > reservoir_sample

Protected Attributes

- std::mutex lock

7.46.1 Detailed Description

Author

piantado

Date

29/01/20

7.46.2 Constructor & Destructor Documentation

7.46.2.1 StreamingStatistics()

```
Fleet::Statistics::StreamingStatistics::StreamingStatistics (
             size_t rs = 1000 )  [inline]
```

7.46.3 Member Function Documentation

7.46.3.1 add()

```
void Fleet::Statistics::StreamingStatistics::add (
             double x,
             double lw = 0.0 )  [inline]
```

Add sample x (with log weight lw) to these statistics.

Parameters

x	
lw	

7.46.3.2 median()

```
double Fleet::Statistics::StreamingStatistics::median ( ) const  [inline]
```

Compute the median according to my reservoir sample.

Returns

7.46.3.3 operator<<()

```
void Fleet::Statistics::StreamingStatistics::operator<< (
             double x )  [inline]
```

Add x

Parameters

x	

7.46.3.4 operator=()

```
void Fleet::Statistics::StreamingStatistics::operator= (
             StreamingStatistics && s )  [inline]
```

7.46.3.5 p_exceeds_median()

```
double Fleet::Statistics::StreamingStatistics::p_exceeds_median (
             const StreamingStatistics & q ) const  [inline]
```

What proportion of my samples exceed the median of q?

Parameters

q	

Returns

7.46.3.6 print()

```
void Fleet::Statistics::StreamingStatistics::print ( ) const  [inline]
```

7.46.3.7 sample()

```
double Fleet::Statistics::StreamingStatistics::sample ( ) const  [inline]
```

Treat as a reservoir sample to sample an element.

Returns

7.46.4 Member Data Documentation

7.46.4.1 lock

```
std::mutex Fleet::Statistics::StreamingStatistics::lock  [mutable], [protected]
```

7.46.4.2 lse

```
double Fleet::Statistics::StreamingStatistics::lse
```

7.46.4.3 max

```
double Fleet::Statistics::StreamingStatistics::max
```

7.46.4.4 min

```
double Fleet::Statistics::StreamingStatistics::min
```

7.46.4.5 N

```
double Fleet::Statistics::StreamingStatistics::N
```

7.46.4.6 reservoir_sample

```
ReservoirSample<double> Fleet::Statistics::StreamingStatistics::reservoir_sample
```

7.46.4.7 streaming_median

```
MedianFAME<double> Fleet::Statistics::StreamingStatistics::streaming_median
```

7.46.4.8 sum

```
double Fleet::Statistics::StreamingStatistics::sum
```

The documentation for this class was generated from the following file:

- src/Statistics/StreamingStatistics.h

7.47 t_null Struct Reference

```
#include <Miscellaneous.h>
```

The documentation for this struct was generated from the following file:

- src/Miscellaneous.h

7.48 VirtualMachineState< t_x, t_return >::t_stack< args > Struct Template Reference

```
#include <VirtualMachineState.h>
```

Public Attributes

- std::tuple< Stack< args >... > value

7.48.1 Member Data Documentation

7.48.1.1 value

```
template<typename t_x, typename t_return>
template<typename... args>
std::tuple<Stack<args>...> VirtualMachineState< t_x, t_return >::t_stack< args >::value
```

The documentation for this struct was generated from the following file:

- src/VirtualMachine/VirtualMachineState.h

7.49 Fleet::Statistics::TopN< T > Class Template Reference

```
#include <Top.h>
```

Public Member Functions

- TopN (size_t n=std::numeric_limits< size_t >::max())
- TopN (const TopN< T > &x)
- TopN (TopN< T > &&x)
- void operator= (const TopN< T > &x)
- void operator= (TopN< T > &&x)
- void set_size (size_t n)
- void set_print_best (bool b)
- size_t size () const
- bool empty () const
- std::multiset< T > & values ()
- void add (const T &x, size_t count=1)
- void operator<< (const T &x)
- void add (const TopN< T > &x)
- void operator<< (const TopN< T > &x)
- void operator() (T &x)
- size_t operator[ ] (const T &x)
- const T & best ()
- const T & worst ()
- double best_score ()
- double worst_score ()
- double Z ()
- void print (std::string prefix="")
- void clear ()
- unsigned long count (const T x)
- template<typename t_data >
  TopN compute_posterior (t_data &data)

Public Attributes

- std::map< T, unsigned long > cnt
- std::multiset< T > s
- bool print_best
- std::atomic< size_t > N

7.49.1 Detailed Description

template<class T>
class Fleet::Statistics::TopN< T >

Author

steven piantadosi

Date

03/02/20

7.49.2 Constructor & Destructor Documentation

7.49.2.1 TopN() [1/3]

```
template<class T>
Fleet::Statistics::TopN< T >::TopN (
              size_t n = std::numeric_limits<size_t>::max() )  [inline]
```

7.49.2.2 TopN() [2/3]

```
template<class T>
Fleet::Statistics::TopN< T >::TopN (
              const TopN< T > & x )  [inline]
```

7.49.2.3 TopN() [3/3]

```
template<class T>
Fleet::Statistics::TopN< T >::TopN (
              TopN< T > && x )  [inline]
```

7.49.3 Member Function Documentation

7.49.3.1 add() [1/2]

```
template<class T>
void Fleet::Statistics::TopN< T >::add (
              const T & x,
              size_t count = 1 )  [inline]
```

Add x. If count is set, that will add that many counts. NOTE that we do not add objects x such that x.posterior == -infinity or NaN

Parameters

x	
count	

7.49.3.2 add() [2/2]

```
template<class T>
void Fleet::Statistics::TopN< T >::add (
              const TopN< T > & x )  [inline]
```

Add everything in x to this one.

Parameters

x	

7.49.3.3 best()

```
template<class T>
const T& Fleet::Statistics::TopN< T >::best ( )  [inline]
```

Returns a reference to the best element currently stored

Returns

7.49.3.4 best_score()

```
template<class T>
double Fleet::Statistics::TopN< T >::best_score ( )  [inline]
```

Return the score of the best element (thread-safe)

Returns

7.49.3.5 clear()

```
template<class T>
void Fleet::Statistics::TopN< T >::clear ( )  [inline]
```

Remove everything

7.49.3.6 compute_posterior()

```
template<class T>
template<typename t_data >
TopN Fleet::Statistics::TopN< T >::compute_posterior (
             t_data & data )  [inline]
```

Returns a NEW TopN where each current hypothesis is evaluated on the data. NOTE: If a hypothesis has a new posterior of -inf or NaN, it won't be added.

Parameters

data	

Returns

7.49.3.7 count()

```
template<class T>
unsigned long Fleet::Statistics::TopN< T >::count (
              const T x )  [inline]
```

How many times have we seen x?

Parameters

x	

Returns

7.49.3.8 empty()

```
template<class T>
bool Fleet::Statistics::TopN< T >::empty ( ) const  [inline]
```

Is it empty?

Returns

7.49.3.9 operator()()

```
template<class T>
void Fleet::Statistics::TopN< T >::operator() (
              T & x )  [inline]
```

7.49.3.10 operator<<() [1/2]

```
template<class T>
void Fleet::Statistics::TopN< T >::operator<< (
             const T & x )  [inline]
```

Friendlier syntax for adding.

Parameters

x	

7.49.3.11 operator<<() [2/2]

```
template<class T>
void Fleet::Statistics::TopN< T >::operator<< (
             const TopN< T > & x )  [inline]
```

7.49.3.12 operator=() [1/2]

```
template<class T>
void Fleet::Statistics::TopN< T >::operator= (
             const TopN< T > & x )  [inline]
```

7.49.3.13 operator=() [2/2]

```
template<class T>
void Fleet::Statistics::TopN< T >::operator= (
             TopN< T > && x )  [inline]
```

7.49.3.14 operator[]()

```
template<class T>
size_t Fleet::Statistics::TopN< T >::operator[] (
             const T & x )  [inline]
```

Access the counts for a given element x

Parameters

x	

Returns

7.49.3.15 print()

```
template<class T>
void Fleet::Statistics::TopN< T >::print (
             std::string prefix = "" )  [inline]
```

Sort and print from worst to best

Parameters

prefix	- an optional prefix to print before each line

7.49.3.16 set_print_best()

```
template<class T>
void Fleet::Statistics::TopN< T >::set_print_best (
             bool b )  [inline]
```

As I add things, should I print the best I've seen so far?

Parameters

b	

7.49.3.17 set_size()

```
template<class T>
void Fleet::Statistics::TopN< T >::set_size (
             size_t n )  [inline]
```

Set the size of n that I cna have. NOTE: this does not resize the existing data.

Parameters

n	

7.49.3.18 size()

```
template<class T>
size_t Fleet::Statistics::TopN< T >::size ( ) const  [inline]
```

How many are currently in the set? (NOT the total number allowed)

Returns

7.49.3.19 values()

```
template<class T>
std::multiset<T>& Fleet::Statistics::TopN< T >::values ( ) [inline]
```

Return a multiset of all the values in TopN

Returns

7.49.3.20 worst()

```
template<class T>
const T& Fleet::Statistics::TopN< T >::worst ( ) [inline]
```

Returns a reference to the worst element currently stored

Returns

7.49.3.21 worst_score()

```
template<class T>
double Fleet::Statistics::TopN< T >::worst_score ( ) [inline]
```

Return the score of the worst element

Returns

7.49.3.22 Z()

```
template<class T>
double Fleet::Statistics::TopN< T >::Z ( )  [inline]
```

Compute the logsumexp of all of the elements stored.

Returns

7.49.4 Member Data Documentation

7.49.4.1 cnt

```
template<class T>
std::map<T,unsigned long> Fleet::Statistics::TopN< T >::cnt
```

7.49.4.2 N

```
template<class T>
std::atomic<size_t> Fleet::Statistics::TopN< T >::N
```

7.49.4.3 print_best

```
template<class T>
bool Fleet::Statistics::TopN< T >::print_best
```

7.49.4.4 s

```
template<class T>
std::multiset<T> Fleet::Statistics::TopN< T >::s
```

The documentation for this class was generated from the following file:

- src/Statistics/Top.h

7.50 TypeHead< Args > Struct Template Reference

```
#include <TemplateMagic.h>
```

Public Types

- typedef std::tuple_element< 0, std::tuple< Args... > >::type type

7.50.1 Detailed Description

template<class... Args>
struct TypeHead< Args >

```
    For managing references in lambda arguments
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */


// Count how many reference types
//template <class T, class... Types>
//struct CountReferences;
template <class T, class... Types>
struct CountReferences {
    static const size_t value = std::is_reference<T>::value + CountReferences<Types...>::value;
};
template <class T>
struct CountReferences<T> { static const size_t value = std::is_reference<T>::value; };


// If there ar eany references in the arguments, only the first can be a reference
template <class T, class... Types>
struct CheckReferenceIsFirst {
    static const bool value = (CountReferences<Types...>::value == 0);
};
template <class T>
struct CheckReferenceIsFirst<T> { static const bool value = true; };


/*
```

List operations on args ∼∼

7.50.2 Member Typedef Documentation

7.50.2.1 type

```
template<class... Args>
typedef std::tuple_element<0, std::tuple<Args...> >::type TypeHead< Args >::type
```

The documentation for this struct was generated from the following file:

- src/TemplateMagic.h

7.51 VirtualMachinePool< t_x, t_return > Class Template Reference

```
#include <Dispatchable.h>
```

Public Member Functions

- VirtualMachinePool (unsigned long ms=2048, unsigned long mo=256, double mlp=-20)
- virtual ∼VirtualMachinePool ()
- bool wouldIadd (double lp)
- void push (VMState ∗o)
- template< typename T >
  bool copy_increment_push (const VMState ∗x, T v, double lpinc)
- template< typename T >
  bool increment_push (VMState ∗s, T v, double lpinc)
- DiscreteDistribution< t_return > run (Dispatchable< t_x, t_return > ∗dispatcher, Dispatchable< t_x, t_return > ∗loader)

Public Attributes

- const unsigned long max_steps
- const unsigned long max_outputs
- unsigned long current_steps
- double min_lp
- double worst_lp = infinity
- std::priority_queue< VMState ∗, std::vector< VMState ∗ >, VirtualMachinePool::compare_VMState_prt > Q

7.51.1 Detailed Description

template<typename t_x, typename t_return>
class VirtualMachinePool< t_x, t_return >

Author

piantado

Date

02/02/20

7.51.2 Constructor & Destructor Documentation

7.51.2.1 VirtualMachinePool()

```
template<typename t_x, typename t_return>
VirtualMachinePool< t_x, t_return >::VirtualMachinePool (
            unsigned long ms = 2048,
            unsigned long mo = 256,
            double mlp = -20 )  [inline]
```

7.51.2.2 ∼VirtualMachinePool()

```
template<typename t_x, typename t_return>
virtual VirtualMachinePool< t_x, t_return >::∼VirtualMachinePool ( )  [inline], [virtual]
```

7.51.3 Member Function Documentation

7.51.3.1 copy_increment_push()

```
template<typename t_x, typename t_return>
template<typename T >
bool VirtualMachinePool< t_x, t_return >::copy_increment_push (
            const VMState * x,
            T v,
            double lpinc )  [inline]
```

This is an important opimization where we will make a copy of x, push v into it's stack, and increment its lp by lpinc only if it will be added to the queue, which we check in the pool here. This saves us from having to use the VirtualMachineState constructor (e.g. making a copy, which is expensive if we are copying all the stacks) if the copy won't actually be added to the queue.

Parameters

x	
v	
lpinc	

Returns

7.51.3.2 increment_push()

```
template<typename t_x, typename t_return>
template<typename T >
```

```
bool VirtualMachinePool< t_x, t_return >::increment_push (
            VMState * s,
            T v,
            double lpinc )  [inline]
```

Same as copy_increment_push, but does not make a copy – just add

Parameters

s	
v	
lpinc	

Returns

7.51.3.3 push()

```
template<typename t_x, typename t_return>
void VirtualMachinePool< t_x, t_return >::push (
            VMState * o )  [inline]
```

Add the VMState o to this pool (but again checking if I'd add)

Parameters

o	

7.51.3.4 run()

```
template<typename t_x, typename t_return>
DiscreteDistribution<t_return> VirtualMachinePool< t_x, t_return >::run (
            Dispatchable< t_x, t_return > * dispatcher,
            Dispatchable< t_x, t_return > * loader )  [inline]
```

This runs and adds up the probability mass for everything, returning a dictionary outcomes->log_probabilities. This is the main running loop, which pops frmo the top of our queue, runs, and continues until we've done enough or all. Note that objects lower than min_lp are not ever pushed onto the queue.

Parameters

dispatcher	
loader	

Returns

7.51.3.5 wouldIadd()

```
template<typename t_x, typename t_return>
bool VirtualMachinePool< t_x, t_return >::wouldIadd (
             double lp )  [inline]
```

Returns true if I would add something with this lp, given my max_steps and the stack. This lets us speed up by checking if we would add before copying/constructing a VMS

Parameters

lp	

Returns

7.51.4 Member Data Documentation

7.51.4.1 current_steps

```
template<typename t_x, typename t_return>
unsigned long VirtualMachinePool< t_x, t_return >::current_steps
```

7.51.4.2 max_outputs

```
template<typename t_x, typename t_return>
const unsigned long VirtualMachinePool< t_x, t_return >::max_outputs
```

7.51.4.3 max_steps

```
template<typename t_x, typename t_return>
const unsigned long VirtualMachinePool< t_x, t_return >::max_steps
```

7.51.4.4 min_lp

```
template<typename t_x, typename t_return>
double VirtualMachinePool< t_x, t_return >::min_lp
```

7.51.4.5 Q

```
template<typename t_x, typename t_return>
std::priority_queue<VMState*, std::vector<VMState*>, VirtualMachinePool::compare_VMState_prt>
VirtualMachinePool< t_x, t_return >::Q
```

7.51.4.6 worst_lp

```
template<typename t_x, typename t_return>
double VirtualMachinePool< t_x, t_return >::worst_lp = infinity
```

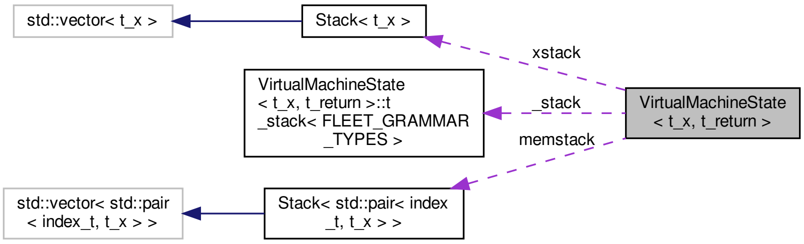
The documentation for this class was generated from the following files:

- src/Hypotheses/Interfaces/Dispatchable.h
- src/VirtualMachine/VirtualMachinePool.h

7.52 VirtualMachineState< t_x, t_return > Class Template Reference

```
#include <Dispatchable.h>
```

Collaboration diagram for VirtualMachineState< t_x, t_return >:


std::vector< t_x >
Stack< t_x >
xstack
VirtualMachineState < t_x, t_return >::t _stack< FLEET_GRAMMAR _TYPES >
_stack
VirtualMachineState < t_x, t_return >
memstack
std::vector< std::pair < index_t, t_x > >
Stack< std::pair< index _t, t_x > >

Classes

- struct t_stack

Public Types

- typedef int index_t

Public Member Functions

- VirtualMachineState (t_x x, t_return e, size_t _recursion_depth=0)
- virtual ∼VirtualMachineState ()
- bool operator< (const VirtualMachineState &m) const
- void increment_lp (double v)
- template<typename T >
  Stack< T > & stack ()
- template<typename T >
  const Stack< T > & stack () const
- template<typename T >
  T getpop ()
- template<typename T >
  T gettop ()
- template<typename T >
  std::conditional< std::is_reference< T >::value, T &, T >::type get ()
- template<typename T >
  bool empty ()
- template<typename T >
  void push (T &x)
- template<typename T >
  void push (T &&x)
- template<typename... args>
  bool any_stacks_empty () const
- template<typename... args>
  bool all_stacks_empty () const
- bool stacks_empty () const
- virtual t_return run (Dispatchable< t_x, t_return > ∗d)
- virtual t_return run (VirtualMachinePool< t_x, t_return > ∗pool, Dispatchable< t_x, t_return > ∗dispatch, Dispatchable< t_x, t_return > ∗loader)

Public Attributes

- Program opstack
- Stack< t_x > xstack
- t_return err
- double lp
- unsigned long recursion_depth
- t_stack< FLEET_GRAMMAR_TYPES > _stack
- std::map< std::pair< index_t, t_x >, t_return > mem
- Stack< std::pair< index_t, t_x > > memstack
- vmstatus_t status

Static Public Attributes

- static const unsigned long MAX_RECURSE = 64

7.52.1 Detailed Description

template<typename t_x, typename t_return>
class VirtualMachineState< t_x, t_return >

Author

piantado

Date

02/02/20

7.52.2 Member Typedef Documentation

7.52.2.1 index_t

```
template<typename t_x, typename t_return>
typedef int VirtualMachineState< t_x, t_return >::index_t
```

7.52.3 Constructor & Destructor Documentation

7.52.3.1 VirtualMachineState()

```
template<typename t_x, typename t_return>
VirtualMachineState< t_x, t_return >::VirtualMachineState (
            t_x x,
            t_return e,
            size_t _recursion_depth = 0 )  [inline]
```

7.52.3.2 ∼VirtualMachineState()

```
template<typename t_x, typename t_return>
virtual VirtualMachineState< t_x, t_return >::∼VirtualMachineState ( )  [inline], [virtual]
```

7.52.4 Member Function Documentation

7.52.4.1 all_stacks_empty()

```
template<typename t_x, typename t_return>
template<typename... args>
bool VirtualMachineState< t_x, t_return >::all_stacks_empty ( ) const  [inline]
```

Check if all of the stacks are empty (should be at the end of evaluation)

Returns

7.52.4.2 any_stacks_empty()

```
template<typename t_x, typename t_return>
template<typename... args>
bool VirtualMachineState< t_x, t_return >::any_stacks_empty ( ) const  [inline]
```

Check if any of the stacks are empty

Returns

7.52.4.3 empty()

```
template<typename t_x, typename t_return>
template<typename T >
bool VirtualMachineState< t_x, t_return >::empty ( )  [inline]
```

Is this stack empty?

Returns

7.52.4.4 get()

```
template<typename t_x, typename t_return>
template<typename T >
std::conditional<std::is_reference<T>::value, T&, T>::type VirtualMachineState< t_x, t_↩
return >::get ( )  [inline]
```

This is some fanciness that will return a reference to the top of the stack if we give it a reference type otherwise it will return the type. This lets us get the top of a stack with a reference in PRIMITIVES as though we were some kind of wizards

Returns

7.52.4.5 getpop()

```
template<typename t_x, typename t_return>
template<typename T >
T VirtualMachineState< t_x, t_return >::getpop ( )  [inline]
```

Retrieves and pops the element of type T from the stack

Returns

7.52.4.6 gettop()

```
template<typename t_x, typename t_return>
template<typename T >
T VirtualMachineState< t_x, t_return >::gettop ( )  [inline]
```

Retrieves the top of the stack as a copy and does *not* remove

Returns

7.52.4.7 increment_lp()

```
template<typename t_x, typename t_return>
void VirtualMachineState< t_x, t_return >::increment_lp (
             double v )  [inline]
```

Add v to my lp

Parameters

v	

7.52.4.8 operator<()

```
template<typename t_x, typename t_return>
bool VirtualMachineState< t_x, t_return >::operator< (
             const VirtualMachineState< t_x, t_return > & m ) const  [inline]
```

These must be sortable by lp so that we can enumerate them from low to high probability in a VirtualMachinePool NOTE: VirtualMachineStates shouldn't be put in a set because they might evaluate to equal!

Parameters

m	

Returns

7.52.4.9 push() [1/2]

```
template<typename t_x, typename t_return>
template<typename T >
void VirtualMachineState< t_x, t_return >::push (
            T & x )  [inline]
```

Push things onto the appropriate stack

Parameters

x	

7.52.4.10 push() [2/2]

```
template<typename t_x, typename t_return>
template<typename T >
void VirtualMachineState< t_x, t_return >::push (
            T && x )  [inline]
```

7.52.4.11 run() [1/2]

```
template<typename t_x, typename t_return>
virtual t_return VirtualMachineState< t_x, t_return >::run (
            Dispatchable< t_x, t_return > * d )  [inline], [virtual]
```

Defaultly run a non-recursive hypothesis

Parameters

d	

Returns

7.52.4.12 run() [2/2]

```
template<typename t_x, typename t_return>
virtual t_return VirtualMachineState< t_x, t_return >::run (
            VirtualMachinePool< t_x, t_return > * pool,
            Dispatchable< t_x, t_return > * dispatch,
            Dispatchable< t_x, t_return > * loader )  [inline], [virtual]
```

Run with a pointer back to pool p. This is required because "flip" may push things onto the pool. Here, dispatch is called to evaluate the function, and loader is called on recursion (allowing us to handle recursion via a lexicon or just via a LOTHypothesis). NOTE that anything NOT built-in is handled via applyToVMS defined in Primitives.h

Parameters

pool	
dispatch	
loader	

Returns

7.52.4.13 stack() [1/2]

```
template<typename t_x, typename t_return>
template<typename T >
Stack<T>& VirtualMachineState< t_x, t_return >::stack ( )  [inline]
```

Returns a reference to the stack (of a given type)

Returns

7.52.4.14 stack() [2/2]

```
template<typename t_x, typename t_return>
template<typename T >
const Stack<T>& VirtualMachineState< t_x, t_return >::stack ( ) const  [inline]
```

Const reference to top of stack

Returns

7.52.4.15 stacks_empty()

```
template<typename t_x, typename t_return>
bool VirtualMachineState< t_x, t_return >::stacks_empty ( ) const  [inline]
```

True if all stacks are empty for the FLEET_GRAMMAR_TYPES

Returns

7.52.5 Member Data Documentation

7.52.5.1 _stack

```
template<typename t_x, typename t_return>
t_stack<FLEET_GRAMMAR_TYPES> VirtualMachineState< t_x, t_return >::_stack
```

7.52.5.2 err

```
template<typename t_x, typename t_return>
t_return VirtualMachineState< t_x, t_return >::err
```

7.52.5.3 lp

```
template<typename t_x, typename t_return>
double VirtualMachineState< t_x, t_return >::lp
```

7.52.5.4 MAX_RECURSE

```
template<typename t_x, typename t_return>
const unsigned long VirtualMachineState< t_x, t_return >::MAX_RECURSE = 64  [static]
```

7.52.5.5 mem

```
template<typename t_x, typename t_return>
std::map<std::pair<index_t, t_x>, t_return> VirtualMachineState< t_x, t_return >::mem
```

7.52.5.6 memstack

```
template<typename t_x, typename t_return>
Stack<std::pair<index_t, t_x> > VirtualMachineState< t_x, t_return >::memstack
```

7.52.5.7 opstack

```
template<typename t_x, typename t_return>
Program VirtualMachineState< t_x, t_return >::opstack
```

7.52.5.8 recursion_depth

```
template<typename t_x, typename t_return>
unsigned long VirtualMachineState< t_x, t_return >::recursion_depth
```

7.52.5.9 status

```
template<typename t_x, typename t_return>
vmstatus_t VirtualMachineState< t_x, t_return >::status
```

7.52.5.10 xstack

```
template<typename t_x, typename t_return>
Stack<t_x> VirtualMachineState< t_x, t_return >::xstack
```

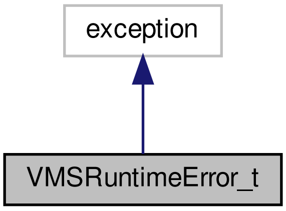
The documentation for this class was generated from the following files:

- src/Hypotheses/Interfaces/Dispatchable.h
- src/VirtualMachine/VirtualMachineState.h

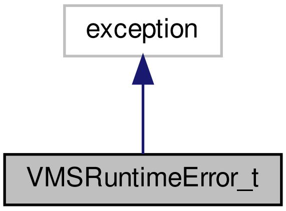
7.53 VMSRuntimeError_t Class Reference

```
#include <Instruction.h>
```

Inheritance diagram for VMSRuntimeError_t:



Collaboration diagram for VMSRuntimeError_t:



7.53.1 Detailed Description

Author

steven piantadosi

Date

03/02/20

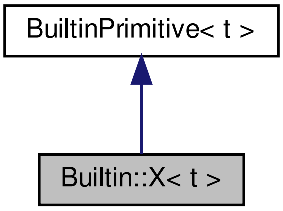
The documentation for this class was generated from the following file:

- src/VirtualMachine/Instruction.h

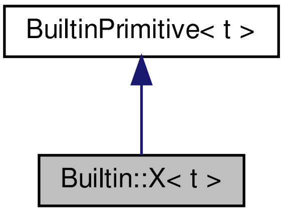
7.54 Builtin::X< t > Struct Template Reference

```
#include <Builtins.h>
```

Inheritance diagram for Builtin::X< t >:



Collaboration diagram for Builtin::X< t >:



Public Member Functions

- X (std::string fmt, double _p=1.0)

Additional Inherited Members

7.54.1 Constructor & Destructor Documentation

7.54.1.1 X()

```
template<typename t >
Builtin::X< t >::X (
            std::string fmt,
            double _p = 1.0 )  [inline]
```

The documentation for this struct was generated from the following file:

- src/VirtualMachine/Builtins.h

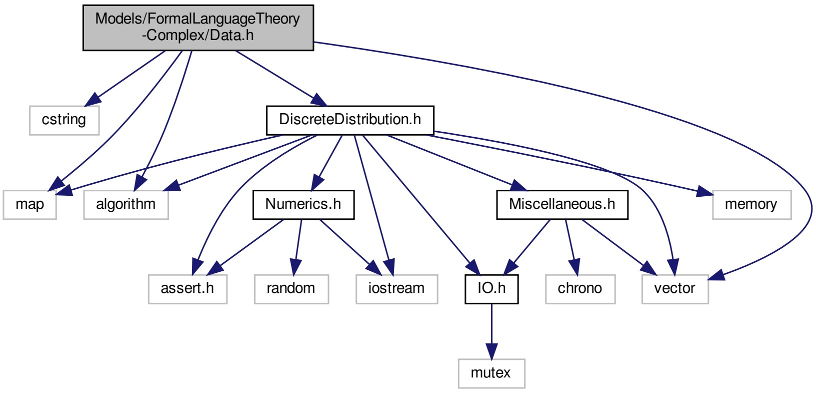
Chapter 8

File Documentation

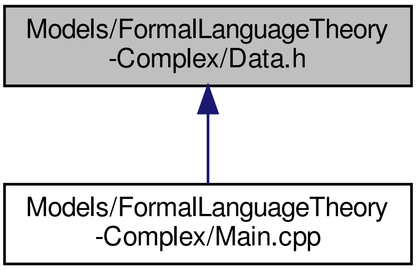
8.1 Models/FormalLanguageTheory-Complex/Data.h File Reference

```
#include <cstring>
#include <vector>
#include <map>
#include <algorithm>
#include "DiscreteDistribution.h"
```

Include dependency graph for Data.h:



This graph shows which files directly or indirectly include this file:



Functions

- template< typename tdata >
  void load_data_file (std::vector< tdata > &data, const char ∗datapath)
- template< typename T , typename TDATA >
  std::map< T, double > highest (const std::vector< TDATA > &m, unsigned long N)
- template< typename TDATA >
  std::pair< double, double > get_precision_and_recall (std::ostream &output, DiscreteDistribution< std::string > &model, std::vector< TDATA > &data, unsigned long N)

8.1.1 Function Documentation

8.1.1.1 get_precision_and_recall()

```
template<typename TDATA >
std::pair<double, double> get_precision_and_recall (
            std::ostream & output,
            DiscreteDistribution< std::string > & model,
            std::vector< TDATA > & data,
            unsigned long N )
```

8.1.1.2 highest()

```
template<typename T , typename TDATA >
std::map<T, double> highest (
            const std::vector< TDATA > & m,
            unsigned long N )
```

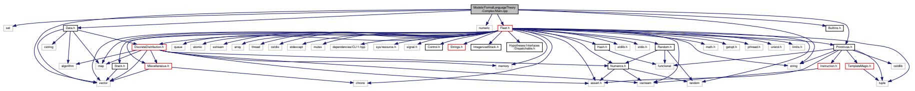
8.1.1.3 load_data_file()

```
template<typename tdata >
void load_data_file (
            std::vector< tdata > & data,
            const char * datapath )
```

8.2 Models/FormalLanguageTheory-Complex/Main.cpp File Reference

```
#include <set>
#include <string>
#include <vector>
#include <numeric>
#include "Data.h"
#include "Primitives.h"
#include "Builtins.h"
#include "Fleet.h"
```

Include dependency graph for Main.cpp:



Classes

- class InnerHypothesis
- class MyHypothesis

Macros

- #define FLEET_GRAMMAR_TYPES S,bool,double,StrSet
- #define CUSTOM_OPS op_UniformSample,op_P

Typedefs

- using S = std::string
- using StrSet = std::set< S >

Functions

- int main (int argc, char **argv)

Variables

- const std::string my_default_input = "data/SimpleEnglish"
- S alphabet ="nvadt"
- size_t max_length = 256
- size_t max_setsize = 64
- size_t nfactors = 2
- const size_t PREC_REC_N = 25
- const size_t MAX_LINES = 1000000
- const size_t MAX_PR_LINES = 1000000
- const size_t NTEMPS = 10
- const size_t MAXTEMP = 1000.0
- std::vector$<$ S $>$ data_amounts ={"50000"}
- const double MIN_LP = -25.0
- unsigned long MAX_STEPS_PER_FACTOR = 2048

  NOTE: IF YOU CHANGE, CHANGE BELOW TOO ∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼.
- unsigned long MAX_OUTPUTS_PER_FACTOR = 512
- const unsigned long PRINT_STRINGS = 128

  NOTE: IF YOU CHANGE, CHANGE BELOW TOO ∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼.
- std::tuple PRIMITIVES
- std::string prdata_path = ""
- MyHypothesis::t_data prdata
- S current_data = ""

8.2.1 Macro Definition Documentation

8.2.1.1 CUSTOM_OPS

```
#define CUSTOM_OPS op_UniformSample,op_P
```

8.2.1.2 FLEET_GRAMMAR_TYPES

```
#define FLEET_GRAMMAR_TYPES S,bool,double,StrSet
```

8.2.2 Typedef Documentation

8.2.2.1 S

```
using S = std::string
```

8.2.2.2 StrSet

```
using StrSet = std::set<S>
```

8.2.3 Function Documentation

8.2.3.1 main()

```
int main (
             int argc,
             char ** argv )
```

8.2.4 Variable Documentation

8.2.4.1 alphabet

```
S alphabet ="nvadt"
```

8.2.4.2 current_data

```
S current_data = ""
```

8.2.4.3 data_amounts

```
std::vector<S> data_amounts ={"50000"}
```

8.2.4.4 max_length

```
size_t max_length = 256
```

8.2.4.5 MAX_LINES

```
const size_t MAX_LINES = 1000000
```

8.2.4.6 MAX_OUTPUTS_PER_FACTOR

```
unsigned long MAX_OUTPUTS_PER_FACTOR = 512
```

8.2.4.7 MAX_PR_LINES

```
const size_t MAX_PR_LINES = 1000000
```

8.2.4.8 max_setsize

```
size_t max_setsize = 64
```

8.2.4.9 MAX_STEPS_PER_FACTOR

```
unsigned long MAX_STEPS_PER_FACTOR = 2048
```

NOTE: IF YOU CHANGE, CHANGE BELOW TOO ∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼∼.

8.2.4.10 MAXTEMP

```
const size_t MAXTEMP = 1000.0
```

8.2.4.11 MIN_LP

```
const double MIN_LP = -25.0
```

8.2.4.12 my_default_input

```
const std::string my_default_input = "data/SimpleEnglish"
```

8.2.4.13 nfactors

```
size_t nfactors = 2
```

8.2.4.14 NTEMPS

```
const size_t NTEMPS = 10
```

8.2.4.15 prdata

```
MyHypothesis::t_data prdata
```

8.2.4.16 prdata_path

```
std::string prdata_path = ""
```

8.2.4.17 PREC_REC_N

```
const size_t PREC_REC_N = 25
```

8.2.4.18 PRIMITIVES

```
std::tuple PRIMITIVES
```

```
    These define all of the types that are used in the grammar.
    This macro must be defined before we import Fleet.
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
#define FLEET_GRAMMAR_TYPES
#define CUSTOM_OPS
/*
```

This is a global variable that provides a convenient way to wrap our primitives where we can pair up a function with a name, and pass that as a constructor to the grammar. We need a tuple here because Primitive has a bunch of template types to handle thee function it has, so each is actually a different type. ~~~~~~

8.2.4.19 PRINT_STRINGS

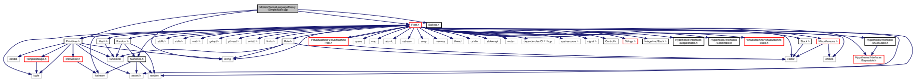
```
const unsigned long PRINT_STRINGS = 128
```

NOTE: IF YOU CHANGE, CHANGE BELOW TOO ~~~~~~.

8.3 Models/FormalLanguageTheory-Simple/Main.cpp File Reference

```
#include <string>
#include "Primitives.h"
#include "Builtins.h"
#include "Fleet.h"
```

Include dependency graph for Main.cpp:



Macros

- #define FLEET_GRAMMAR_TYPES S,bool

Typedefs

- using S = std::string

8.3.1 Macro Definition Documentation

8.3.1.1 FLEET_GRAMMAR_TYPES

```
#define FLEET_GRAMMAR_TYPES S,bool
```

```
    Set up some basic variables (which may get overwritten)
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

S alphabet = "01"; // the alphabet we use (possibly specified on command line)
//S datastr = "";
//S datastr  = "011,011011,011011011"; // the data, comma separated
//S datastr = "01,01001";
S datastr  = "01,01001,010010001,01001000100001"; // the data, comma separated
const double strgamma = 0.95; //75; // penalty on string length
const size_t MAX_LENGTH = 64; // longest strings cons will handle

/*
```

These define all of the types that are used in the grammar. This macro must be defined before we import Fleet. ~~~~~~

8.3.2 Typedef Documentation

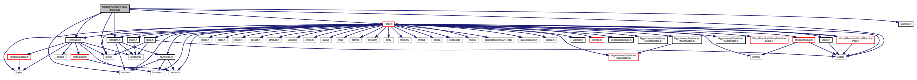
8.3.2.1 S

```
using S = std::string
```

8.4 Models/Number-Fancy/Main.cpp File Reference

```
#include <vector>
#include <string>
#include <random>
#include "Random.h"
#include "Primitives.h"
#include "Builtins.h"
#include "Fleet.h"
```

Include dependency graph for Main.cpp:



Macros

- #define FLEET_GRAMMAR_TYPES bool,word,set,objectkind,utterance,wmset,magnitude,double

Variables

- double recursion_penalty = -75.0

8.4.1 Macro Definition Documentation

8.4.1.1 FLEET_GRAMMAR_TYPES

```
#define FLEET_GRAMMAR_TYPES bool,word,set,objectkind,utterance,wmset,magnitude,double
```

```
    Set up some basic variables for the model
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

typedef int          word;
typedef std::string set;     /* This models sets as strings */
typedef char         objectkind; // store this as that
typedef struct { set s; objectkind o; } utterance;
typedef short        wmset; // just need an integerish type
typedef float        magnitude;

std::vector<objectkind>   OBJECTS = {'a', 'b', 'c', 'd', 'e'}; //, 'f', 'g', 'h', 'i', 'j'};
std::vector<std::string>    WORDS = {"one", "two", "three", "four", "five", "six", "seven", "eight", "nine"
      , "ten"};
std::vector<magnitude> MAGNITUDES = {1,2,3,4,5,6,7,8,9,10};

const word U = -999;

const size_t MAX_SET_SIZE = 25;
const double alpha = 0.9;
const double W = 0.3; // weber ratio for ans

// TODO: UPDATE WITH data from Gunderson & Levine?
std::discrete_distribution<> number_distribution({0, 7187, 1484, 593, 334, 297, 165, 151, 86, 105, 112});
      // 0-indexed

std::vector<int> data_amounts = {1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100, 125, 150,
      200, 250, 300, 350, 400, 500, 600};//, 500, 600, 700, 800, 900, 1000};
//std::vector<int> data_amounts = {600};

/*
```

These define all of the types that are used in the grammar. This macro must be defined before we import Fleet.
~~~~~~

8.4.2 Variable Documentation

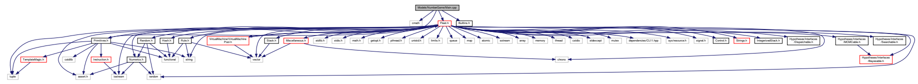
8.4.2.1 recursion_penalty

```
double recursion_penalty = -75.0
```

8.5 Models/NumberGame/Main.cpp File Reference

```
#include <cmath>
#include "Primitives.h"
#include "Builtins.h"
#include "Fleet.h"
```

Include dependency graph for Main.cpp:


~~~~~~

Classes

- class MyHypothesis

Macros

- #define FLEET_GRAMMAR_TYPES float
- #define CUSTOM_OPS op_Constant

Functions

- int main (int argc, char **argv)

Variables

- std::tuple PRIMITIVES

8.5.1 Macro Definition Documentation

8.5.1.1 CUSTOM_OPS

```
#define CUSTOM_OPS op_Constant
```

8.5.1.2 FLEET_GRAMMAR_TYPES

```
#define FLEET_GRAMMAR_TYPES float
```

8.5.2 Function Documentation

8.5.2.1 main()

```
int main (
             int argc,
             char ** argv )
```

8.5.3 Variable Documentation

8.5.3.1 PRIMITIVES

```
std::tuple PRIMITIVES
```

Initial value:

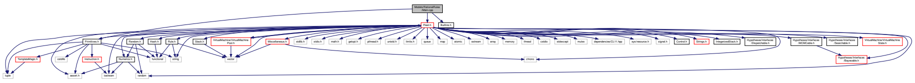
```
= {
    Primitive("(%s+%s)",     +[](float a, float b) -> float { return a+b; }),
    Primitive("(%s-%s)",     +[](float a, float b) -> float { return a-b; }),
    Primitive("(%s*%s)",     +[](float a, float b) -> float { return a*b; }),
    Primitive("(%s/%s)",     +[](float a, float b) -> float { return (b==0 ? 0 : a/b); }),
    Primitive("(%s^%s)",     +[](float a, float b) -> float { return pow(a,b); }),
    Builtin::X<D>("x")
}


    These define all of the types that are used in the grammar.
    This macro must be defined before we import Fleet.
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

const double reliability = 0.99;
const int N = 100; // what number do we go up to?

#define FLEET_GRAMMAR_TYPES

// We're going to need to define our own constant, which will be an instruction
// corresponding to a single float
#define CUSTOM_OPS

/*
```

Define magic primitives ~~~~~~

8.6 Models/RationalRules/Main.cpp File Reference

```
#include "Primitives.h"
#include "Builtins.h"
#include "Fleet.h"
```

Include dependency graph for Main.cpp:



Classes

- struct Object
- class MyHypothesis

Macros

- #define FLEET_GRAMMAR_TYPES bool,Object

Enumerations

- enum Shape { Shape::Square, Shape::Triangle, Shape::Circle }

- enum Color { Color::Red, Color::Green, Color::Blue }

8.6.1 Macro Definition Documentation

8.6.1.1 FLEET_GRAMMAR_TYPES

```
#define FLEET_GRAMMAR_TYPES bool,Object
```

8.6.2 Enumeration Type Documentation

8.6.2.1 Color

```
enum Color  [strong]
```

Enumerator

Red	
Green	
Blue	

8.6.2.2 Shape

```
enum Shape  [strong]
```

```
    These define all of the types that are used in the grammar.
    This macro must be defined before we import Fleet.
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

#define FLEET_GRAMMAR_TYPES

/*
```

We need to define some structs to hold the object features ∼∼

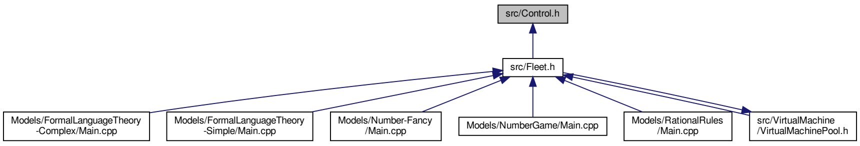
Enumerator

Square	
Triangle	
Circle	

8.7 src/Control.h File Reference

This bundles together information for running MCMC or MCTS, including number of steps, amount of time, etc. NOTE: In general this should NOT be passed by reference because we want start_time to be the time we started the function it is passed to (start time is the time of construction, here)

This graph shows which files directly or indirectly include this file:


src/Control.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Classes

- class Control

8.7.1 Detailed Description

This bundles together information for running MCMC or MCTS, including number of steps, amount of time, etc. NOTE: In general this should NOT be passed by reference because we want start_time to be the time we started the function it is passed to (start time is the time of construction, here)

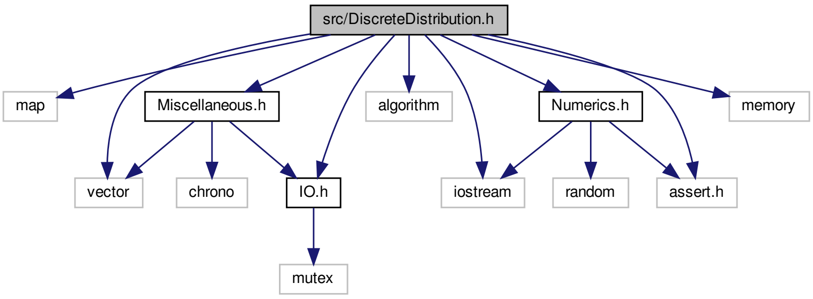
8.8 src/DiscreteDistribution.h File Reference

This stores a distribution from values of T to log probabilities. It is used as the return value from calls with randomness.

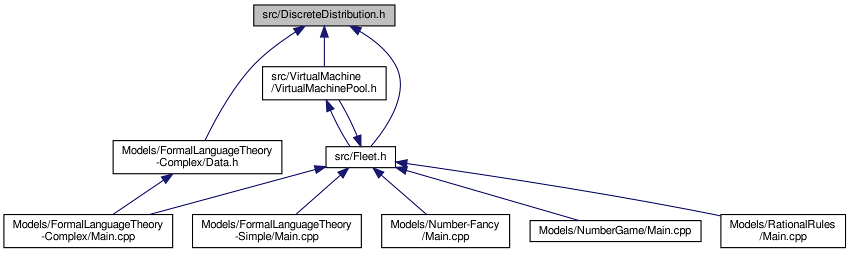
```
#include <map>
#include <vector>
#include <algorithm>
#include <iostream>
#include <assert.h>
#include <memory>
#include "Miscellaneous.h"
#include "Numerics.h"
```

```
#include "IO.h"
```

Include dependency graph for DiscreteDistribution.h:



This graph shows which files directly or indirectly include this file:



Classes

- class DiscreteDistribution< T >

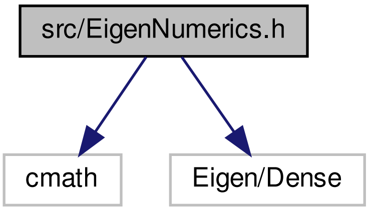
8.8.1 Detailed Description

This stores a distribution from values of T to log probabilities. It is used as the return value from calls with randomness.

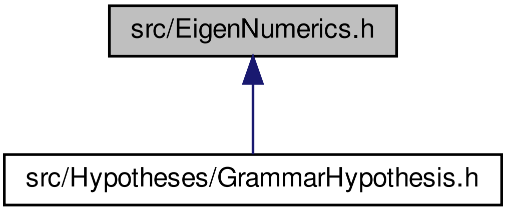
8.9 src/EigenNumerics.h File Reference

```
#include <cmath>
#include <Eigen/Dense>
```

Include dependency graph for EigenNumerics.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- using Vector = Eigen::VectorXf
- using Matrix = Eigen::MatrixXf

Functions

- double logsumexp (const Vector &v)
- Vector lognormalize (const Vector &v)
- Vector eigenslice (const Vector &v, const size_t offset, const size_t len)

8.9.1 Typedef Documentation

8.9.1.1 Matrix

```
using Matrix = Eigen::MatrixXf
```

8.9.1.2 Vector

```
using Vector = Eigen::VectorXf
```

8.9.2 Function Documentation

8.9.2.1 eigenslice()

```
Vector eigenslice (
             const Vector & v,
             const size_t offset,
             const size_t len )
```

8.9.2.2 lognormalize()

```
Vector lognormalize (
             const Vector & v )
```

8.9.2.3 logsumexp()

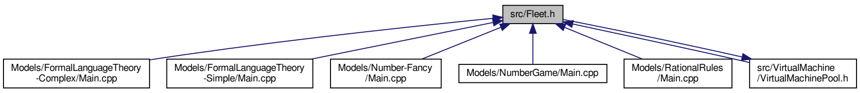
```
double logsumexp (
             const Vector & v )
```

8.10 src/Fleet.h File Reference

```
#include "stdlib.h"
#include "stdio.h"
#include "math.h"
#include <iostream>
#include <getopt.h>
#include <pthread.h>
#include <unistd.h>
#include <limits.h>
#include <vector>
#include <queue>
#include <tuple>
#include <map>
#include <atomic>
#include <assert.h>
#include <string>
#include <sstream>
#include <array>
```

```
#include <memory>
#include <thread>
#include <cstdio>
#include <stdexcept>
#include <random>
#include <mutex>
#include "dependencies/CL11.hpp"
#include <sys/resource.h>
#include <signal.h>
#include <chrono>
#include "Stack.h"
#include "Control.h"
#include "Numerics.h"
#include "Random.h"
#include "Strings.h"
#include "Hash.h"
#include "Miscellaneous.h"
#include "IntegerizedStack.h"
#include "Hypotheses/Interfaces/Dispatchable.h"
#include "Hypotheses/Interfaces/Bayesable.h"
#include "Hypotheses/Interfaces/MCMCable.h"
#include "Hypotheses/Interfaces/Searchable.h"
#include "Rule.h"
#include "VirtualMachine/VirtualMachinePool.h"
#include "VirtualMachine/VirtualMachineState.h"
#include "Node.h"
#include "IO.h"
#include "Grammar.h"
#include "CaseMacros.h"
#include "DiscreteDistribution.h"
#include "Hypotheses/LOTHypothesis.h"
#include "Hypotheses/Lexicon.h"
#include "Inference/MCMCChain.h"
#include "Inference/MCTS.h"
#include "Inference/ParallelTempering.h"
#include "Inference/ChainPool.h"
#include "Top.h"
#include "Primitives.h"
```

This graph shows which files directly or indirectly include this file:


src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Typedefs

- typedef unsigned long time_ms
- typedef std::chrono::steady_clock::time_point timept

Functions

- void fleet_interrupt_handler (int signum)

- timept now ()
- time_ms time_since (timept x)
- void tic ()
- double elapsed_seconds ()
- std::string datestring ()

Variables

- const std::string FLEET_VERSION = "0.0.9"
- volatile sig_atomic_t CTRL_C = false
- timept tic_start
- time_ms tic_elapsed

8.10.1 Typedef Documentation

8.10.1.1 time_ms

```
typedef unsigned long time_ms
```

```
    These are standard variables that occur nearly universally in these searches
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

const std::string ChildStr = "%s"; // how do strings get substituted?

unsigned long random_seed  = 0;
unsigned long mcts_steps   = 0;
unsigned long mcmc_steps   = 0;
unsigned long thin         = 0;
unsigned long ntop         = 100;
unsigned long mcmc_restart = 0;
unsigned long checkpoint   = 0;
double        explore      = 1.0; // we want to exploit the string prefixes we find
size_t        nthreads     = 1;
unsigned long runtime      = 0;
unsigned long nchains      = 1;
bool          quiet      = false; // this is used to indicate that we want to not print much out (typically
       only posteriors and counts)
std::string   input_path   = "input.txt";
std::string   tree_path    = "tree.txt";
std::string   output_path  = "output.txt";
std::string   timestring   = "0s";


namespace Fleet {
    CLI::App DefaultArguments(const char* brief) {
        CLI::App app{brief};

        app.add_option("-R,--seed",     random_seed, "Seed the rng (0 is no seed)");
        app.add_option("-s,--mcts",     mcts_steps, "Number of MCTS search steps to run");
        app.add_option("-m,--mcmc",      mcmc_steps, "Number of mcmc steps to run");
        app.add_option("-t,--thin",      thin, "Thinning on the number printed");
        app.add_option("-o,--output",    output_path, "Where we write output");
        app.add_option("-O,--top",       ntop, "The number to store");
        app.add_option("-n,--threads",   nthreads, "Number of threads for parallel search");
        app.add_option("-e,--explore",   explore, "Exploration parameter for MCTS");
        app.add_option("-r,--restart",   mcmc_restart, "If we don't improve after this many, restart");
        app.add_option("-i,--input",     input_path, "Read standard input from here");
        app.add_option("-T,--time",      timestring, "Stop (via CTRL-C) after this much time (takes smhd as
       seconds/minutes/hour/day units)");
        app.add_option("-E,--tree",      tree_path, "Write the tree here");
        app.add_option(  "-c,--chains",   nchains, "How many chains to run");

        app.add_flag(  "-q,--quiet",  quiet, "Don't print very much and do so on one line");
//      app.add_flag(  "-C,--checkpoint",   checkpoint, "Checkpoint every this many steps");

        return app;
    }
}


/*
```

Time is a goddamn nightmare, we are going to define our own time type In Fleet, EVERYTHING is going to be stored as an unsigned long for ms ∼∼∼

8.10.1.2 timept

```
typedef std::chrono::steady_clock::time_point timept
```

8.10.2 Function Documentation

8.10.2.1 datestring()

```
std::string datestring ( )
```

8.10.2.2 elapsed_seconds()

```
double elapsed_seconds ( )
```

8.10.2.3 fleet_interrupt_handler()

```
void fleet_interrupt_handler (
             int signum )
```

8.10.2.4 now()

```
timept now ( )
```

8.10.2.5 tic()

```
void tic ( )
```

8.10.2.6 time_since()

```
time_ms time_since (
              timept x )
```

8.10.3 Variable Documentation

8.10.3.1 CTRL_C

```
volatile sig_atomic_t CTRL_C = false
```

```
    Tracking Fleet statistics
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

namespace FleetStatistics {
    // Running MCMC/MCTS updates these standard statistics

    std::atomic<uintmax_t> posterior_calls(0);
    std::atomic<uintmax_t> hypothesis_births(0);  // how many total hypotheses have been created? -- useful
       for tracking when we found a solution
    std::atomic<uintmax_t> vm_ops(0);
    std::atomic<uintmax_t> mcmc_proposal_calls(0);
    std::atomic<uintmax_t> mcmc_acceptance_count(0);
    std::atomic<uintmax_t> global_sample_count(0);

    void reset() {
        posterior_calls = 0;
        hypothesis_births = 0;
        vm_ops = 0;
        mcmc_proposal_calls = 0;
        mcmc_acceptance_count = 0;
        global_sample_count = 0;
    }
}

namespace Fleet {
    size_t GRAMMAR_MAX_DEPTH = 64;
    const size_t MAX_CHILD_SIZE = 8; // rules can have at most this many children  -- for now (we can
       change if needed)

    int Pdenom = 24; // the denominator for probabilities in op_P --  we're going to enumeraet fractions in
       24ths -- just so we can get thirds, quarters, fourths



}
/*
```

A Handler for CTRL_C ~~~~~~

8.10.3.2 FLEET_VERSION

```
const std::string FLEET_VERSION = "0.0.9"
```

8.10.3.3 tic_elapsed

```
time_ms tic_elapsed
```

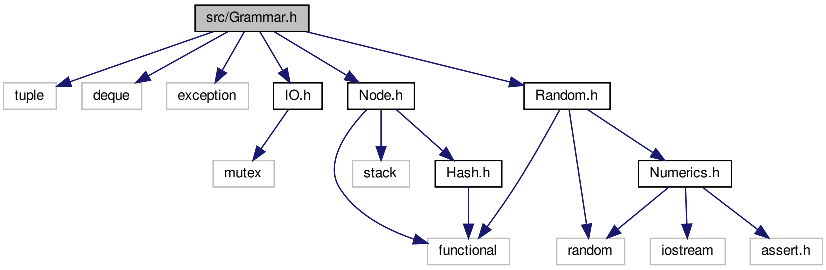
8.10.3.4 tic_start

```
timept tic_start
```

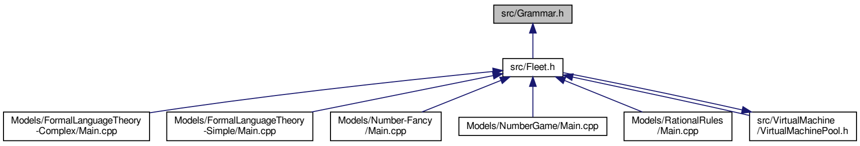
8.11 src/Grammar.h File Reference

```
#include <tuple>
#include <deque>
#include <exception>
#include "IO.h"
#include "Node.h"
#include "Random.h"
```

Include dependency graph for Grammar.h:



This graph shows which files directly or indirectly include this file:



Classes

- class DepthException
- struct Primitive< T, args >
- class Grammar

Variables

- DepthException depth_exception

8.11.1 Variable Documentation

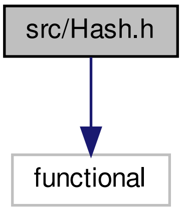
8.11.1.1 depth_exception

```
DepthException depth_exception
```

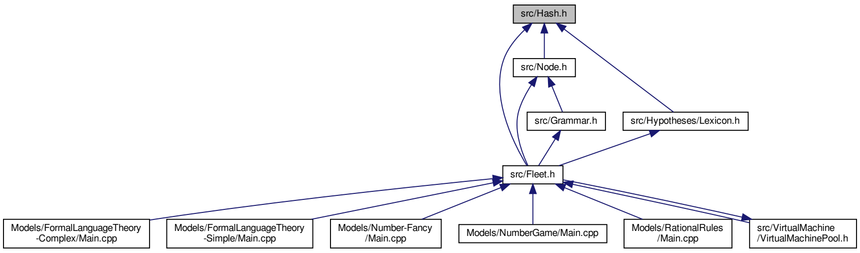
8.12 src/Hash.h File Reference

```
#include <functional>
```

Include dependency graph for Hash.h:



This graph shows which files directly or indirectly include this file:


src/Hash.h
src/Node.h
src/Grammar.h
src/Hypotheses/Lexicon.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Functions

- void hash_combine (std::size_t &seed)
- template<typename T , typename... Rest>
  void hash_combine (std::size_t &seed, const T &v, Rest... rest)

8.12.1 Function Documentation

8.12.1.1 hash_combine() [1/2]

```
void hash_combine (
             std::size_t & seed )  [inline]
```

8.12.1.2 hash_combine() [2/2]

```
template<typename T , typename...  Rest>
void hash_combine (
             std::size_t & seed,
             const T & v,
             Rest...  rest )  [inline]
```

Simple way to combine hash functions

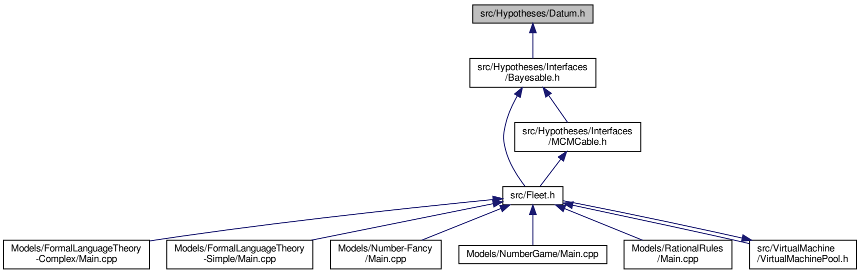
Parameters

seed	
v	

8.13 src/Hypotheses/Datum.h File Reference

A datum is the default data point for likelihoods, consisting of an input and output type. The reliability is measures the reliability of the data (sometimes number of effective data points, sometimes its the noise in the likelihood.

This graph shows which files directly or indirectly include this file:


src/Hypotheses/Datum.h
src/Hypotheses/Interfaces /Bayesable.h
src/Hypotheses/Interfaces /MCMCable.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Classes

- class default_datum< t_input, t_output >

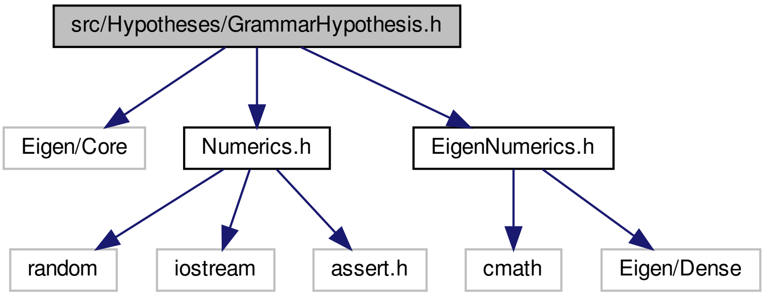
8.13.1 Detailed Description

A datum is the default data point for likelihoods, consisting of an input and output type. The reliability is measures the reliability of the data (sometimes number of effective data points, sometimes its the noise in the likelihood.

8.14 src/Hypotheses/GrammarHypothesis.h File Reference

```
#include <Eigen/Core>
#include "Numerics.h"
#include "EigenNumerics.h"
```

Include dependency graph for GrammarHypothesis.h:



Classes

- struct HumanDatum< t_learnerdatum, t_learnerdata >
- class GrammarHypothesis< HYP, t_datum, t_data >

Functions

- template<typename HYP >
  Vector counts (HYP &h)
- template<typename HYP >
  Matrix counts (std::vector< HYP > &hypotheses)
- template<typename HYP , typename t_data >
  Matrix incremental_likelihoods (std::vector< HYP > &hypotheses, t_data &human_data)
- template<typename HYP , typename t_data >
  Matrix model_predictions (std::vector< HYP > &hypotheses, t_data &human_data)

8.14.1 Function Documentation

8.14.1.1 counts() [1/2]

```
template<typename HYP >
Vector counts (
            HYP & h )
```

8.14.1.2 counts() [2/2]

```
template<typename HYP >
Matrix counts (
             std::vector< HYP > & hypotheses )
```

8.14.1.3 incremental_likelihoods()

```
template<typename HYP , typename t_data >
Matrix incremental_likelihoods (
             std::vector< HYP > & hypotheses,
             t_data & human_data )
```

8.14.1.4 model_predictions()

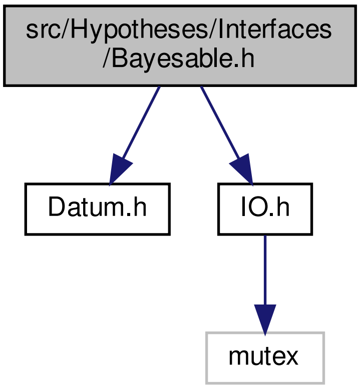
```
template<typename HYP , typename t_data >
Matrix model_predictions (
             std::vector< HYP > & hypotheses,
             t_data & human_data )
```

8.15 src/Hypotheses/Interfaces/Bayesable.h File Reference

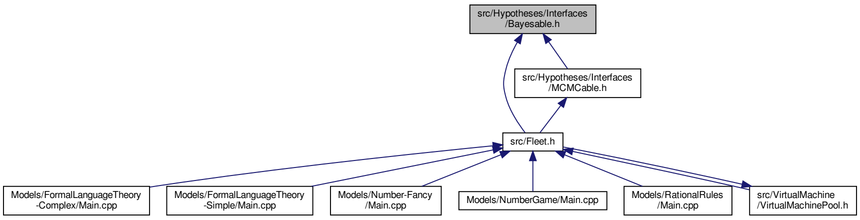
The Bayesable class provides an interface for hypotheses that support Bayesian inference (e.g. computing priors, likelihoods, and posteriors) Note that this class stores prior, likelihood, posterior always at temperature 1.0, and you can get the values of the posterior at other temperatures via Bayesable.at_temperature(double t)

```
#include "Datum.h"
#include "IO.h"
```

Include dependency graph for Bayesable.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Bayesable< _t_datum, _t_data >

Functions

- template<typename _t_datum , typename _t_data >
  std::ostream & operator<< (std::ostream &o, Bayesable< _t_datum, _t_data > &x)

8.15.1 Detailed Description

The Bayesable class provides an interface for hypotheses that support Bayesian inference (e.g. computing priors, likelihoods, and posteriors) Note that this class stores prior, likelihood, posterior always at temperature 1.0, and you can get the values of the posterior at other temperatures via Bayesable.at_temperature(double t)

8.15.2 Function Documentation

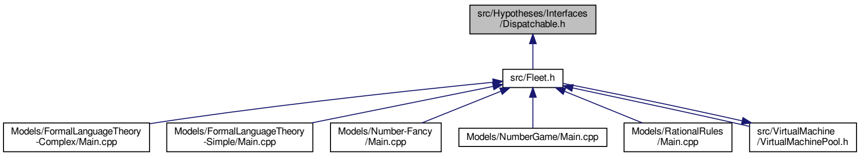
8.15.2.1 operator<<()

```
template<typename _t_datum , typename _t_data >
std::ostream& operator<< (
            std::ostream & o,
            Bayesable< _t_datum, _t_data > & x )
```

8.16 src/Hypotheses/Interfaces/Dispatchable.h File Reference

A class is dispatchable if it is able to implement custom operations and put its program onto a Program.

This graph shows which files directly or indirectly include this file:



Classes

- class VirtualMachineState< t_x, t_return >
- class VirtualMachinePool< t_x, t_return >
- class Dispatchable< t_input, t_output >

8.16.1 Detailed Description

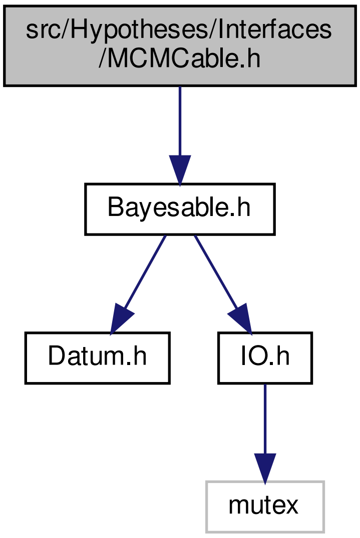
A class is dispatchable if it is able to implement custom operations and put its program onto a Program.

8.17 src/Hypotheses/Interfaces/MCMCable.h File Reference

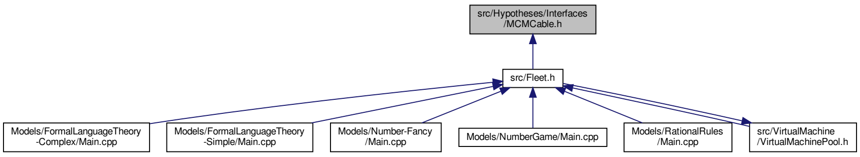
A class is MCMCable if it is Bayesable and lets us propose, restart, and check equality (which MCMC does for speed).

```
#include "Bayesable.h"
```

Include dependency graph for MCMCable.h:



This graph shows which files directly or indirectly include this file:



Classes

- class MCMCable< HYP, Args >

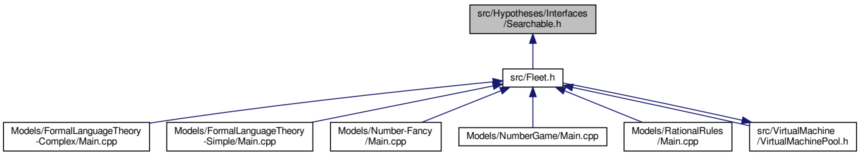
8.17.1 Detailed Description

A class is MCMCable if it is Bayesable and lets us propose, restart, and check equality (which MCMC does for speed).

8.18 src/Hypotheses/Interfaces/Searchable.h File Reference

A class is searchable if permits us to enumerate and make its neighbors. This class is used by MCTS and allows us to incrementally search a hypothesis.

This graph shows which files directly or indirectly include this file:



Classes

- class Searchable< HYP, t_input, t_output >

8.18.1 Detailed Description

A class is searchable if permits us to enumerate and make its neighbors. This class is used by MCTS and allows us to incrementally search a hypothesis.

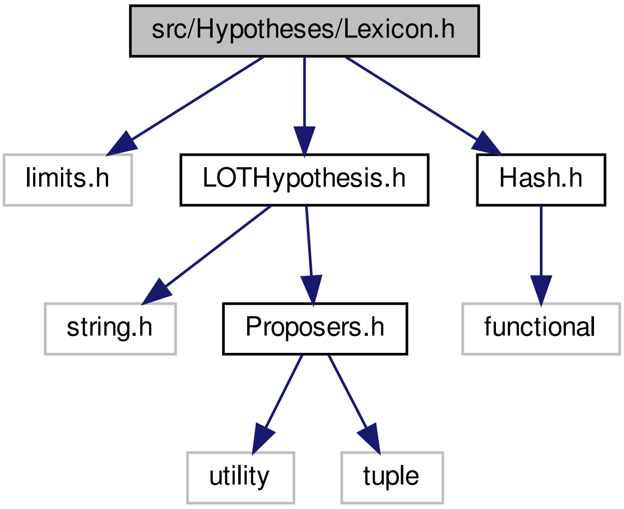
8.19 src/Hypotheses/Lexicon.h File Reference

A lexicon stores an association of numbers (in a vector) to some other kind of hypotheses (typically a LOT↩ Hypothesis). Each of these components is called a "factor.".

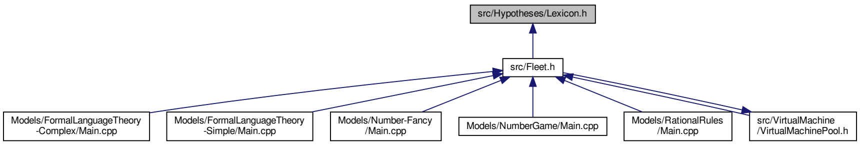
```
#include <limits.h>
#include "LOTHypothesis.h"
```

```
#include "Hash.h"
```

Include dependency graph for Lexicon.h:



This graph shows which files directly or indirectly include this file:


src/Hypotheses/Lexicon.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Classes

- class Lexicon< HYP, T, t_input, t_output, t_datum >

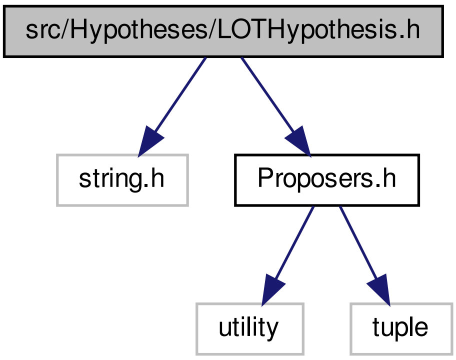
8.19.1 Detailed Description

A lexicon stores an association of numbers (in a vector) to some other kind of hypotheses (typically a LOT← Hypothesis). Each of these components is called a "factor.".

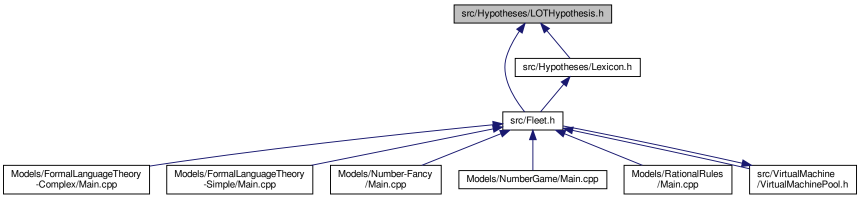
8.20 src/Hypotheses/LOTHypothesis.h File Reference

```
#include <string.h>
#include "Proposers.h"
```

Include dependency graph for LOTHypothesis.h:



This graph shows which files directly or indirectly include this file:


src/Hypotheses/LOTHypothesis.h
src/Hypotheses/Lexicon.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

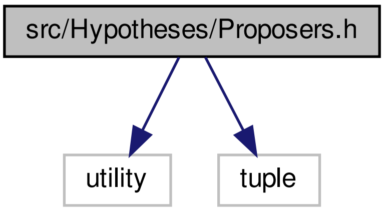
Classes

- class LOTHypothesis< HYP, T, t_input, t_output, _t_datum, _t_data >

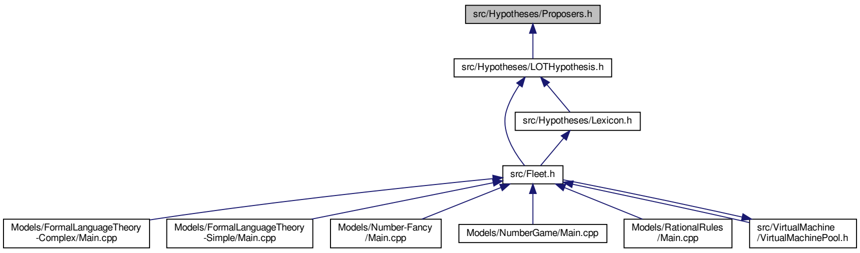
8.21 src/Hypotheses/Proposers.h File Reference

```
#include <utility>
#include <tuple>
```

Include dependency graph for Proposers.h:



This graph shows which files directly or indirectly include this file:


src/Hypotheses/Proposers.h
src/Hypotheses/LOTHypothesis.h
src/Hypotheses/Lexicon.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Namespaces

- Proposals

Functions

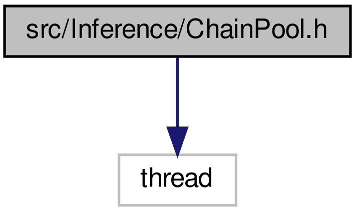
- double Proposals::can_resample (const Node &n)
- std::pair< Node, double > Proposals::prior_proposal (Grammar ∗grammar, const Node &from)
- std::pair< Node, double > Proposals::regenerate (Grammar ∗grammar, const Node &from)
- std::pair< Node, double > Proposals::insert_tree (Grammar ∗grammar, const Node &from)
- std::pair< Node, double > Proposals::delete_tree (Grammar ∗grammar, const Node &from)

8.22 src/Inference/ChainPool.h File Reference

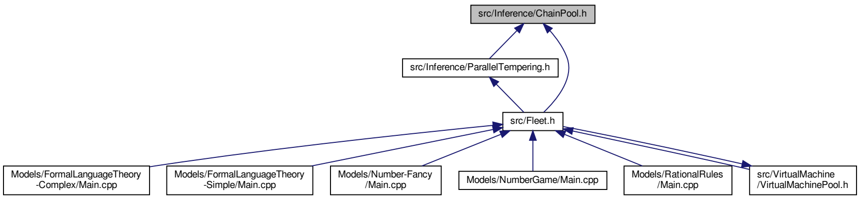
A ChainPool stores a bunch of MCMCChains and allows you to run them serially or in parallel.

```
#include <thread>
```

Include dependency graph for ChainPool.h:



This graph shows which files directly or indirectly include this file:


src/Inference/ChainPool.h
src/Inference/ParallelTempering.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Classes

- class ChainPool< HYP, callback_t >

8.22.1 Detailed Description

A ChainPool stores a bunch of MCMCChains and allows you to run them serially or in parallel.

8.23 src/Inference/Enumeration.h File Reference

Functions

- Node increment_tree (Node ∗t, const Grammar &g)

8.23.1 Function Documentation

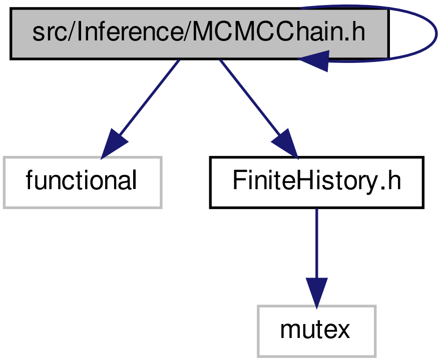
8.23.1.1 increment_tree()

```
Node increment_tree (
            Node * t,
            const Grammar & g )
```

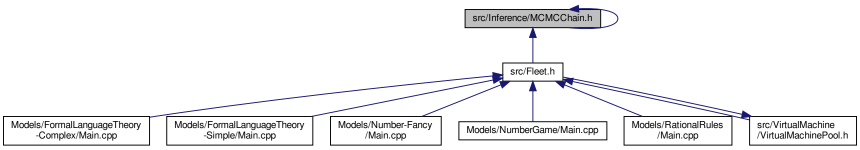
8.24 src/Inference/MCMCChain.h File Reference

```
#include <functional>
#include "MCMCChain.h"
#include "FiniteHistory.h"
```

Include dependency graph for MCMCChain.h:



This graph shows which files directly or indirectly include this file:



Classes

- class MCMCChain< HYP, callback_t >

Variables

- template<typename HYP >
  std::function< void(HYP &)> null_callback = [ ](HYP&){}

8.24.1 Variable Documentation

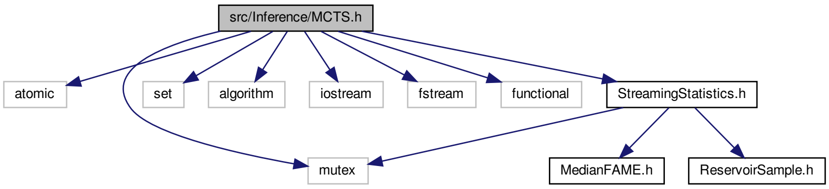
8.24.1.1 null_callback

```
template<typename HYP >
std::function<void(HYP&)> null_callback = [](HYP&){}
```

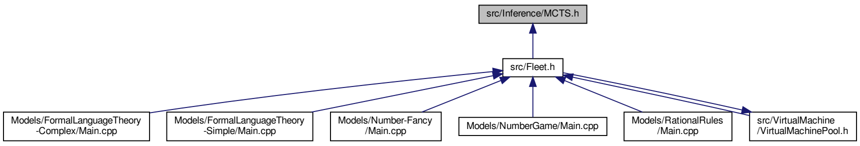
8.25 src/Inference/MCTS.h File Reference

```
#include <atomic>
#include <mutex>
#include <set>
#include <algorithm>
#include <iostream>
#include <fstream>
#include <functional>
#include "StreamingStatistics.h"
```

Include dependency graph for MCTS.h:



This graph shows which files directly or indirectly include this file:


src/Inference/MCTS.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

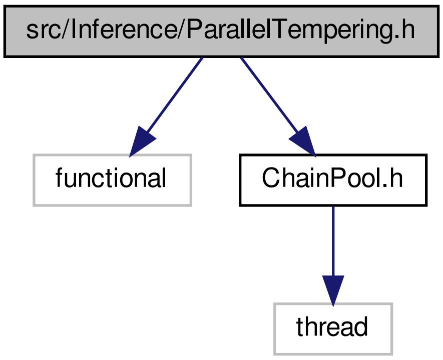
Classes

- class MCTSNode< HYP, callback_t >

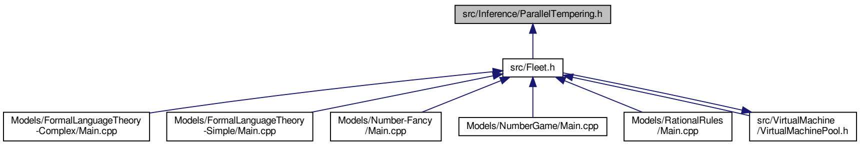
8.26 src/Inference/ParallelTempering.h File Reference

```
#include <functional>
#include "ChainPool.h"
```

Include dependency graph for ParallelTempering.h:



This graph shows which files directly or indirectly include this file:

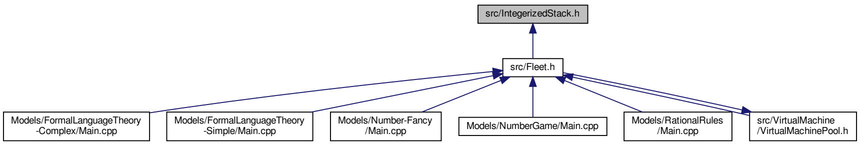

src/Inference/ParallelTempering.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Classes

- class ParallelTempering< HYP, callback_t >

8.27 src/IntegerizedStack.h File Reference

This graph shows which files directly or indirectly include this file:


src/IntegerizedStack.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

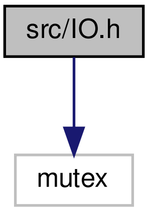
Classes

- class IntegerizedStack

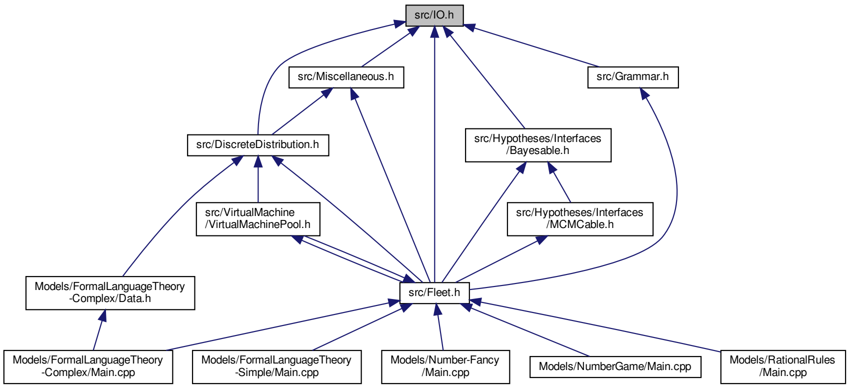
8.28 src/IO.h File Reference

```
#include <mutex>
```

Include dependency graph for IO.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- Fleet

Macros

- #define TAB <<"\t"<<
- #define ENDL <<std::endl;
- #define CERR std::cerr<<
- #define COUT std::cout<<

Variables

- std::mutex Fleet::output_lock

8.28.1 Macro Definition Documentation

8.28.1.1 CERR

```
#define CERR std::cerr<<
```

8.28.1.2 COUT

```
#define COUT std::cout<<
```

8.28.1.3 ENDL

```
#define ENDL <<std::endl;
```

8.28.1.4 TAB

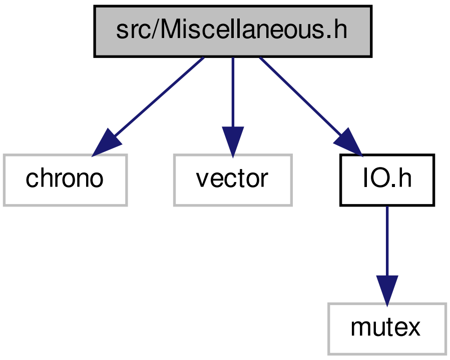
```
#define TAB <<"\t"<<
```

8.29 src/Miscellaneous.h File Reference

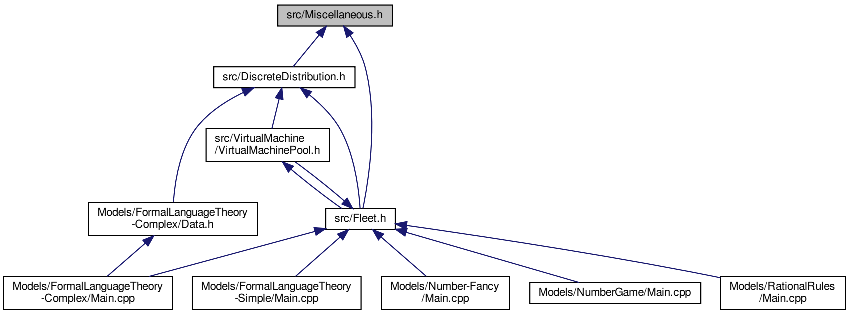
```
#include <chrono>
#include <vector>
```

```
#include "IO.h"
```

Include dependency graph for Miscellaneous.h:



This graph shows which files directly or indirectly include this file:


src/Miscellaneous.h
src/DiscreteDistribution.h
src/VirtualMachine /VirtualMachinePool.h
Models/FormalLanguageTheory -Complex/Data.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp

Classes

- struct t_null
- struct is_iterable< T, typename >

  Converts our own time format to ms, which is what Fleet's time utilities use The time format we accept is #+(.#+)[smhd] where shmd specifies seconds, minutes, hours days.
- struct is_iterable< T, std::void_t< decltype(std::begin(std::declval< T >())), decltype(std::end(std::declval< T >())) > >

Macros

- #define pass ((void)0)

Typedefs

- typedef struct t_null t_null

Functions

- template<typename T >
  std::vector< T > slice (const std::vector< T > &v, size_t start, int len)
- template<typename T >
  std::vector< T > slice (const std::vector< T > &v, size_t start)
- template<typename T >
  bool is_prefix (const T &prefix, const T &x)
- std::string system_exec (const char ∗cmd)

Variables

- template<typename T >
  constexpr bool is_iterable_v = is_iterable<T>::value

8.29.1 Macro Definition Documentation

8.29.1.1 pass

```
#define pass ((void)0)
```

8.29.2 Typedef Documentation

8.29.2.1 t_null

```
typedef struct t_null t_null
```

8.29.3 Function Documentation

8.29.3.1 is_prefix()

```
template<typename T >
bool is_prefix (
             const T & prefix,
             const T & x )
```

For any number of iterable types, is prefix a prefix of x

Parameters

prefix	
x	

Returns

8.29.3.2 slice() [1/2]

```
template<typename T >
std::vector<T> slice (
            const std::vector< T > & v,
            size_t start,
            int len )
```

Take a slice of a vector v starting at start of length len

Parameters

v	
start	
len	

Returns

8.29.3.3 slice() [2/2]

```
template<typename T >
std::vector<T> slice (
            const std::vector< T > & v,
            size_t start )
```

Take a slice of a vector until its end

Parameters

v	
start	

Returns

8.29.3.4 system_exec()

```
std::string system_exec (
             const char * cmd )
```

Call cmd on the system

Parameters

cmd	

Returns

8.29.4 Variable Documentation

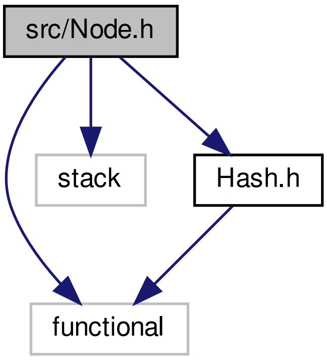
8.29.4.1 is_iterable_v

```
template<typename T >
constexpr bool is_iterable_v = is_iterable<T>::value
```

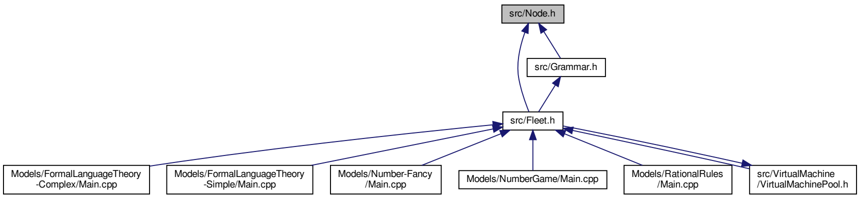
8.30 src/Node.h File Reference

```
#include <functional>
#include <stack>
#include "Hash.h"
```

Include dependency graph for Node.h:



This graph shows which files directly or indirectly include this file:


src/Node.h
src/Grammar.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Classes

- class Node
- class Node::NodeIterator

Functions

- std::ostream & operator<< (std::ostream &o, Node &n)

8.30.1 Function Documentation

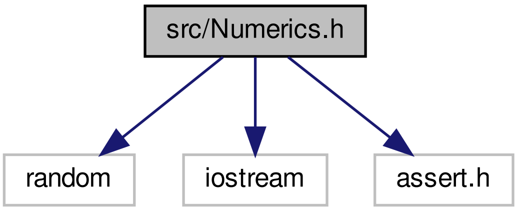
8.30.1.1 operator<<()

```
std::ostream& operator<< (
            std::ostream & o,
            Node & n )
```

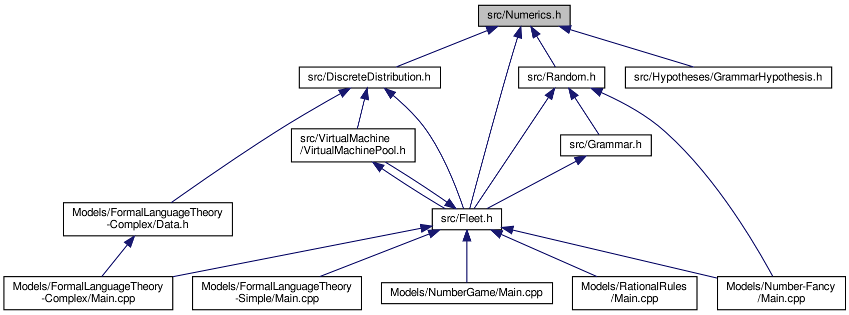
8.31 src/Numerics.h File Reference

```
#include <random>
#include <iostream>
#include <assert.h>
```

Include dependency graph for Numerics.h:



This graph shows which files directly or indirectly include this file:


src/Numerics.h
src/DiscreteDistribution.h
src/Random.h
src/Hypotheses/GrammarHypothesis.h
src/VirtualMachine/VirtualMachinePool.h
src/Grammar.h
Models/FormalLanguageTheory-Complex/Data.h
src/Fleet.h
Models/FormalLanguageTheory-Complex/Main.cpp
Models/FormalLanguageTheory-Simple/Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules/Main.cpp
Models/Number-Fancy/Main.cpp

Typedefs

- typedef size_t enumerationidx_t

Functions

- template<typename t >
  t logplusexp (const t a, const t b)
- std::pair< enumerationidx_t, enumerationidx_t > cantor_decode (const enumerationidx_t z)
- std::pair< enumerationidx_t, enumerationidx_t > rosenberg_strong_decode (const enumerationidx_t z)
- enumerationidx_t rosenberg_strong_encode (const enumerationidx_t x, const enumerationidx_t y)
- std::pair< enumerationidx_t, enumerationidx_t > mod_decode (const enumerationidx_t z, const enumerationidx_t k)
- enumerationidx_t mod_encode (const enumerationidx_t x, const enumerationidx_t y, const enumerationidx_t k)
- enumerationidx_t rosenberg_strong_pop (enumerationidx_t &z)
- enumerationidx_t mod_pop (enumerationidx_t &z, const enumerationidx_t k)

Variables

- const double LOG05 = -log(2.0)
- const double infinity = std::numeric_limits<double>::infinity()
- const double NaN = std::numeric_limits<double>::quiet_NaN()
- const double pi = M_PI

8.31.1 Typedef Documentation

8.31.1.1 enumerationidx_t

```
typedef size_t enumerationidx_t
```

8.31.2 Function Documentation

8.31.2.1 cantor_decode()

```
std::pair<enumerationidx_t, enumerationidx_t> cantor_decode (
            const enumerationidx_t z )
```

8.31.2.2 logplusexp()

```
template<typename t >
t logplusexp (
            const t a,
            const t b )
```

8.31.2.3 mod_decode()

```
std::pair<enumerationidx_t,enumerationidx_t> mod_decode (
            const enumerationidx_t z,
            const enumerationidx_t k )
```

8.31.2.4 mod_encode()

```
enumerationidx_t mod_encode (
             const enumerationidx_t x,
             const enumerationidx_t y,
             const enumerationidx_t k )
```

8.31.2.5 mod_pop()

```
enumerationidx_t mod_pop (
             enumerationidx_t & z,
             const enumerationidx_t k )
```

8.31.2.6 rosenberg_strong_decode()

```
std::pair<enumerationidx_t, enumerationidx_t> rosenberg_strong_decode (
             const enumerationidx_t z )
```

8.31.2.7 rosenberg_strong_encode()

```
enumerationidx_t rosenberg_strong_encode (
             const enumerationidx_t x,
             const enumerationidx_t y )
```

8.31.2.8 rosenberg_strong_pop()

```
enumerationidx_t rosenberg_strong_pop (
             enumerationidx_t & z )
```

8.31.3 Variable Documentation

8.31.3.1 infinity

```
const double infinity = std::numeric_limits<double>::infinity()
```

8.31.3.2 LOG05

```
const double LOG05 = -log(2.0)
```

8.31.3.3 NaN

```
const double NaN = std::numeric_limits<double>::quiet_NaN()
```

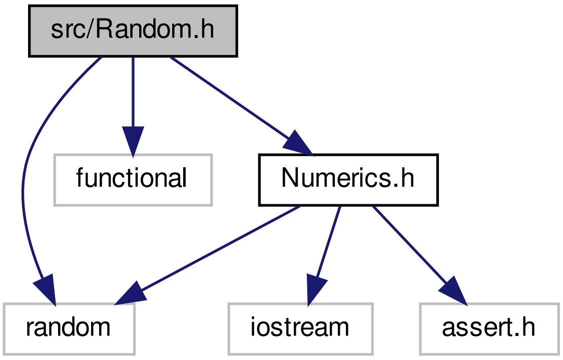
8.31.3.4 pi

```
const double pi = M_PI
```

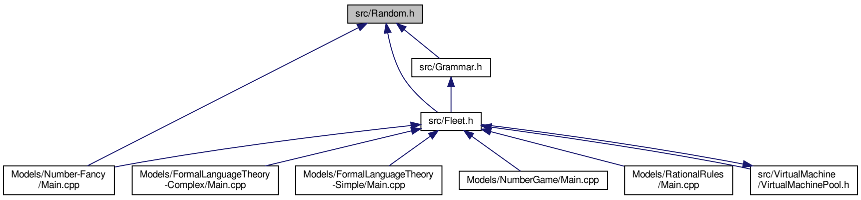
8.32 src/Random.h File Reference

```
#include <random>
#include <functional>
#include "Numerics.h"
```

Include dependency graph for Random.h:



This graph shows which files directly or indirectly include this file:


src/Random.h
src/Grammar.h
src/Fleet.h
Models/Number-Fancy /Main.cpp
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Functions

- thread_local std::mt19937 rng (rd())
- thread_local std::uniform_real_distribution< double > uniform_dist (0, 1.0)
- thread_local std::normal_distribution< float > normal (0.0, 1.0)
- double uniform ()
- double cauchy_lpdf (double x, double loc=0.0, double gamma=1.0)
- double normal_lpdf (double x, double mu=0.0, double sd=1.0)
- double random_cauchy ()
- template< typename t >
  std::vector< t > random_multinomial (t alpha, size_t len)
- template< typename T >
  T myrandom (T max)
- template< typename T >
  T myrandom (T min, T max)
- bool flip ()
- template< typename t , typename T >
  double sample_z (const T &s, std::function< double(const t &)> &f)
- template< typename t , typename T >
  std::pair< t ∗, double > sample (const T &s, std::function< double(const t &)> &f=[ ](const t &v){return 1.0;})
- template< typename t , typename T >
  std::pair< t ∗, double > sample (const T &s, double(∗f)(const t &))
- template< typename t , typename T >
  double lp_sample_eq (const t &x, const T &s, std::function< double(const t &)> &f=[ ](const t &v){return 1.0;})
- template< typename t , typename T >
  double lp_sample_eq (const t &x, const T &s, double(∗f)(const t &))
- template< typename t , typename T >
  double lp_sample_one (const t &x, const T &s, std::function< double(const t &)> &f=[ ](const t &v){return 1.0;})
- template< typename t , typename T >
  double lp_sample_one (const t &x, const T &s, double(∗f)(const t &))

Variables

- thread_local std::random_device rd

8.32.1 Function Documentation

8.32.1.1 cauchy_lpdf()

```
double cauchy_lpdf (
             double x,
             double loc = 0.0,
             double gamma = 1.0 )
```

Compute the log PDF of a cauchy distribution

Parameters

x	- value
loc	- location
gamma	- scale

Returns

8.32.1.2 flip()

```
bool flip ( )
```

Random bool

Returns

8.32.1.3 lp_sample_eq() [1/2]

```
template<typename t , typename T >
double lp_sample_eq (
             const t & x,
             const T & s,
             std::function< double(const t &)> & f = [](const t& v){return 1.0;} )
```

8.32.1.4 lp_sample_eq() [2/2]

```
template<typename t , typename T >
double lp_sample_eq (
             const t & x,
             const T & s,
             double(*)(const t &) f )
```

8.32.1.5 lp_sample_one() [1/2]

```
template<typename t , typename T >
double lp_sample_one (
             const t & x,
             const T & s,
             std::function< double(const t &)> & f = [](const t& v){return 1.0;} )
```

8.32.1.6 lp_sample_one() [2/2]

```
template<typename t , typename T >
double lp_sample_one (
             const t & x,
             const T & s,
             double(*)(const t &) f )
```

8.32.1.7 myrandom() [1/2]

```
template<typename T >
T myrandom (
             T max )
```

My own random integer distribution, going [0,max-1]

Parameters

max	

Returns

8.32.1.8 myrandom() [2/2]

```
template<typename T >
T myrandom (
             T min,
             T max )
```

Random integer distribution [min,max-1]

Parameters

min	
max	

Returns

8.32.1.9 normal()

```
thread_local std::normal_distribution<float> normal (
            0.  0,
            1.  0 )
```

8.32.1.10 normal_lpdf()

```
double normal_lpdf (
            double x,
            double mu = 0.0,
            double sd = 1.0 )
```

Compute the log PDF of a normal distribution

Parameters

x	
mu	
sd	

Returns

8.32.1.11 random_cauchy()

```
double random_cauchy ( )
```

Generate a random sample from a standard cauchy

Returns

8.32.1.12 random_multinomial()

```
template<typename t >
std::vector<t> random_multinomial (
            t alpha,
            size_t len )
```

Generate a random vector from a multinomial, with constant alpha

Parameters

alpha	
len	

Returns

8.32.1.13 rng()

```
thread_local std::mt19937 rng (
             rd()  )
```

8.32.1.14 sample() [1/2]

```
template<typename t , typename T >
std::pair<t*,double> sample (
             const T & s,
             std::function< double(const t &)> & f = [](const t& v){return 1.0;} )
```

8.32.1.15 sample() [2/2]

```
template<typename t , typename T >
std::pair<t*,double> sample (
             const T & s,
             double(*)(const t &) f )
```

8.32.1.16 sample_z()

```
template<typename t , typename T >
double sample_z (
             const T & s,
             std::function< double(const t &)> & f )
```

If f specifies the probability (NOT log probability) of each element of s, compute the normalizing constant.

Parameters

s	- a collection of objects
f	- a function to map over s to get each probability

Returns

8.32.1.17 uniform()

```
double uniform ( )
```

Sample from a uniform distribution

Returns

8.32.1.18 uniform_dist()

```
thread_local std::uniform_real_distribution<double> uniform_dist (
             0 ,
             1.  0 )
```

8.32.2 Variable Documentation

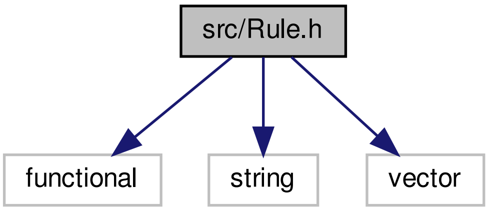
8.32.2.1 rd

```
thread_local std::random_device rd
```

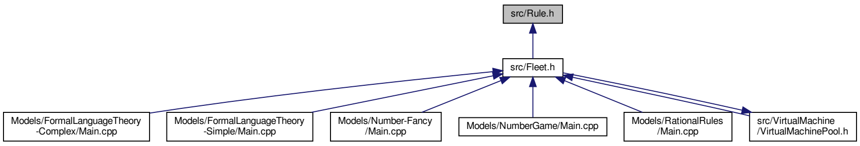
8.33 src/Rule.h File Reference

```
#include <functional>
#include <string>
#include <vector>
```

Include dependency graph for Rule.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Rule

Variables

- const Rule ∗ NullRule = new Rule((nonterminal_t)0, BuiltinOp::op_NOP, "\u2b1c", {}, 0.0)

8.33.1 Variable Documentation

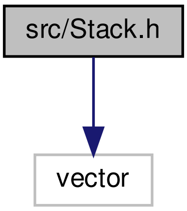
8.33.1.1 NullRule

```
const Rule* NullRule = new Rule((nonterminal_t)0, BuiltinOp::op_NOP, "\u2b1c", {}, 0.0)
```

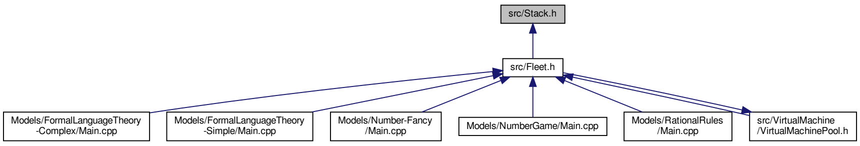
8.34 src/Stack.h File Reference

```
#include <vector>
```

Include dependency graph for Stack.h:



This graph shows which files directly or indirectly include this file:


src/Stack.h
src/Fleet.h
Models/FormalLanguageTheory-Complex/Main.cpp
Models/FormalLanguageTheory-Simple/Main.cpp
Models/Number-Fancy/Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules/Main.cpp
src/VirtualMachine/VirtualMachinePool.h

Classes

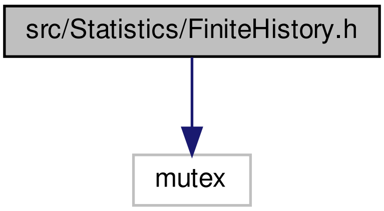
- class Stack< T >

8.35 src/Statistics/FiniteHistory.h File Reference

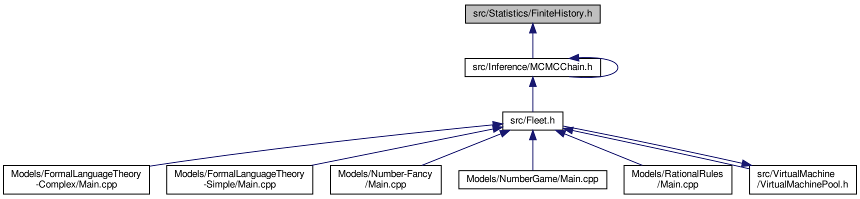
A FiniteHistory stores the previous N examples of something of type T. This is used e.g. in MCMC in order to count the acceptance ratio on the previous N samples.

```
#include <mutex>
```

Include dependency graph for FiniteHistory.h:



This graph shows which files directly or indirectly include this file:


src/Statistics/FiniteHistory.h
src/Inference/MCMCChain.h
src/Fleet.h
Models/FormalLanguageTheory-Complex/Main.cpp
Models/FormalLanguageTheory-Simple/Main.cpp
Models/Number-Fancy/Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules/Main.cpp
src/VirtualMachine/VirtualMachinePool.h

Classes

- class Fleet::Statistics::FiniteHistory< T >

Namespaces

- Fleet
- Fleet::Statistics

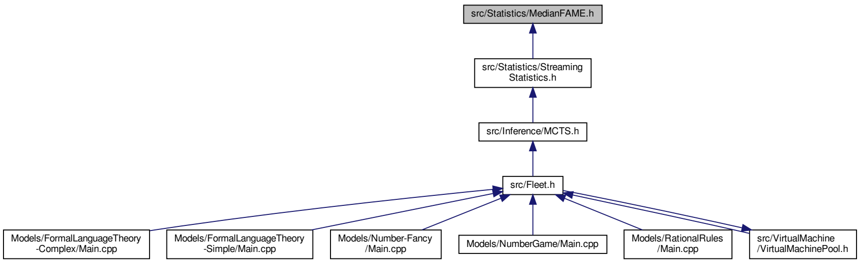
8.35.1 Detailed Description

A FiniteHistory stores the previous N examples of something of type T. This is used e.g. in MCMC in order to count the acceptance ratio on the previous N samples.

8.36 src/Statistics/MedianFAME.h File Reference

A streaming median class implementing the FAME algorithm Here, we initialize both the step size and M with the current sample http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.↩ 7376&rep=rep1&type=pdf.

This graph shows which files directly or indirectly include this file:


src/Statistics/MedianFAME.h
src/Statistics/Streaming Statistics.h
src/Inference/MCTS.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Classes

- class Fleet::Statistics::MedianFAME< T >

Namespaces

- Fleet
- Fleet::Statistics

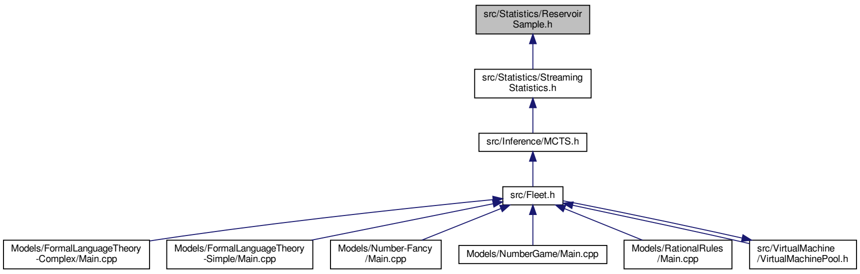
8.36.1 Detailed Description

A streaming median class implementing the FAME algorithm Here, we initialize both the step size and M with the current sample http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.↩ 7376&rep=rep1&type=pdf.

8.37 src/Statistics/ReservoirSample.h File Reference

A special weighted reservoir sampling class that allows for logarithmic weights to do this, we use a transformation following `https://en.wikipedia.org/wiki/Reservoir_sampling#Weighted_random_sampling_using_Reservoir` basically, we want to give a weight that is r^1/w, or log(r)/w, or log(log(r))-log(w). But the problem is that log(r) is negative so log(log(r)) is not defined. Instead, we'll use the function f(x)=-log(-log(x)), which is monotonic. So then, -log(-log(r^1/w)) = -log(-log(r)/w) = -log(-log(r)∗1/w) = -[log(-log(r)) - log(w)] = -log(-log(r)) + log(w)

This graph shows which files directly or indirectly include this file:


src/Statistics/Reservoir Sample.h
src/Statistics/Streaming Statistics.h
src/Inference/MCTS.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Classes

- class Fleet::Statistics::ReservoirSample< T >
- class Fleet::Statistics::ReservoirSample< T >::Item

Namespaces

- Fleet
- Fleet::Statistics

8.37.1 Detailed Description

A special weighted reservoir sampling class that allows for logarithmic weights to do this, we use a transformation following `https://en.wikipedia.org/wiki/Reservoir_sampling#Weighted_random_sampling_using_Reservoir` basically, we want to give a weight that is r^1/w, or log(r)/w, or log(log(r))-log(w). But the problem is that log(r) is negative so log(log(r)) is not defined. Instead, we'll use the function f(x)=-log(-log(x)), which is monotonic. So then, -log(-log(r^1/w)) = -log(-log(r)/w) = -log(-log(r)∗1/w) = -[log(-log(r)) - log(w)] = -log(-log(r)) + log(w)

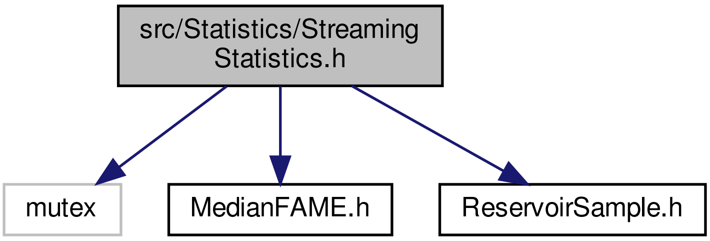
An item in a reservoir sample, grouping together an element x and its log weights, value, etc.

8.38 src/Statistics/StreamingStatistics.h File Reference

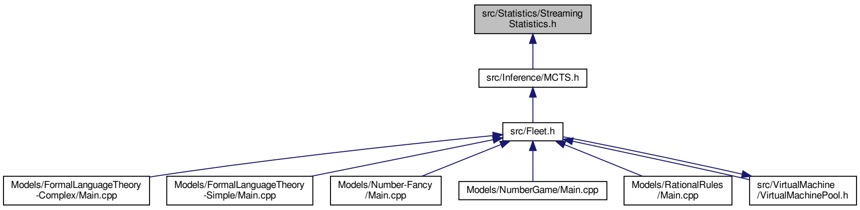
A class to store a bunch of statistics about incoming data points, including min, max, mean, etc. This also stores a reservoir sample and allow us to compute how often one distribution exceeds another.

```
#include <mutex>
#include "MedianFAME.h"
#include "ReservoirSample.h"
```

Include dependency graph for StreamingStatistics.h:



This graph shows which files directly or indirectly include this file:


src/Statistics/Streaming Statistics.h
src/Inference/MCTS.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Classes

- class Fleet::Statistics::StreamingStatistics

Namespaces

- Fleet
- Fleet::Statistics

8.38.1 Detailed Description

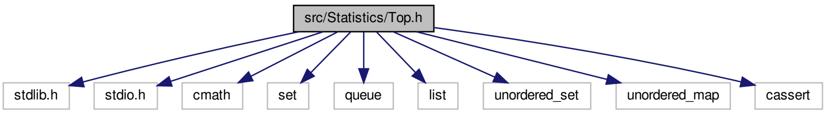
A class to store a bunch of statistics about incoming data points, including min, max, mean, etc. This also stores a reservoir sample and allow us to compute how often one distribution exceeds another.

8.39 src/Statistics/Top.h File Reference

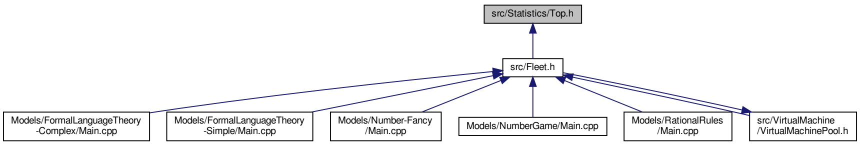
A TopN is a n object you can "add" hypotheses to (via add or <<) and it stores the best N of them. This is widely used in Fleet in order to find good approximations to the top hypotheses found in MCTS or MCMC.

```
#include "stdlib.h"
#include "stdio.h"
#include <cmath>
#include <set>
#include <queue>
#include <list>
#include <unordered_set>
#include <unordered_map>
#include <cassert>
```

Include dependency graph for Top.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Fleet::Statistics::TopN< T >

Namespaces

- Fleet
- Fleet::Statistics

Functions

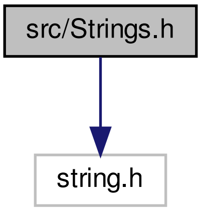
- template<typename HYP >
  void Fleet::Statistics::operator<< (std::set< HYP > &s, TopN< HYP > &t)

8.39.1 Detailed Description

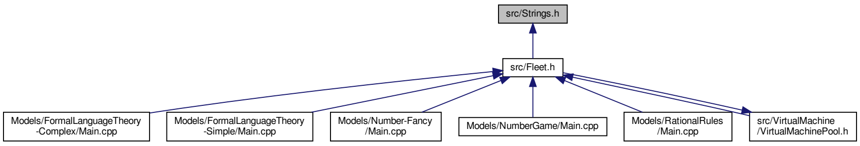
A TopN is a n object you can "add" hypotheses to (via add or <<) and it stores the best N of them. This is widely used in Fleet in order to find good approximations to the top hypotheses found in MCTS or MCMC.

8.40 src/Strings.h File Reference

```
#include <string.h>
```
Include dependency graph for Strings.h:



This graph shows which files directly or indirectly include this file:


src/Strings.h
src/Fleet.h
Models/FormalLanguageTheory -Complex/Main.cpp
Models/FormalLanguageTheory -Simple/Main.cpp
Models/Number-Fancy /Main.cpp
Models/NumberGame/Main.cpp
Models/RationalRules /Main.cpp
src/VirtualMachine /VirtualMachinePool.h

Functions

- template<typename T >
  std::string str (T x)
- std::deque< std::string > split (const std::string &s, const char delimiter)
- unsigned int levenshtein_distance (const std::string &s1, const std::string &s2)
- size_t count (const std::string &str, const std::string &sub)
- std::string QQ (std::string x)
- std::string Q (std::string x)

8.40.1 Function Documentation

8.40.1.1 count()

```
size_t count (
            const std::string & str,
            const std::string & sub )
```

How many times does sub occur in str?

Parameters

str	
sub	

Returns

8.40.1.2 levenshtein_distance()

```
unsigned int levenshtein_distance (
             const std::string & s1,
             const std::string & s2 )
```

Compute levenshtein distiance between two strings (NOTE: Or O(N^2))

Parameters

s1	
s2	

Returns

8.40.1.3 Q()

```
std::string Q (
             std::string x )
```

Handy adding single quotes to a string

Parameters

x	- input string

Returns

8.40.1.4 QQ()

```
std::string QQ (
            std::string x )
```

Handy adding double quotes to a string

Parameters

x	- input string

Returns

8.40.1.5 split()

```
std::deque<std::string> split (
            const std::string & s,
            const char delimiter )
```

Split is returns a deque of s split up at the character delimiter. It handles these special cases: split("a:", ':') -> ["a", ""] split(":", ':') -> [""] split(":a", ':') -> ["", "a"]

Parameters

s	
delimiter	

Returns

8.40.1.6 str()

```
template<typename T >
std::string str (
            T x )
```

A pythonesque string function

Parameters

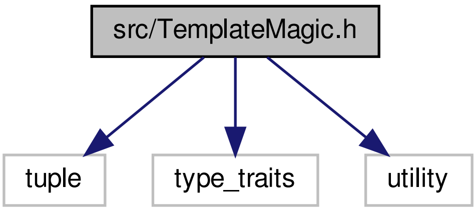
x	

Returns

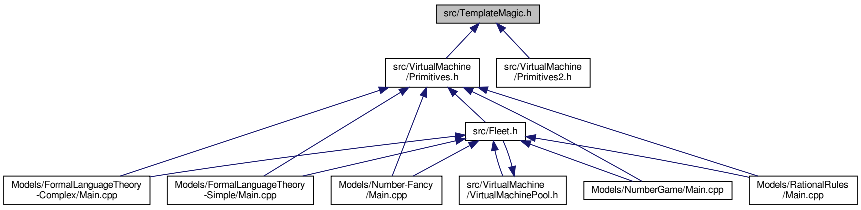
8.41 src/TemplateMagic.h File Reference

```
#include <tuple>
#include <type_traits>
#include <utility>
```

Include dependency graph for TemplateMagic.h:



This graph shows which files directly or indirectly include this file:


src/TemplateMagic.h
src/VirtualMachine/Primitives.h
src/VirtualMachine/Primitives2.h
src/Fleet.h
Models/FormalLanguageTheory-Complex/Main.cpp
Models/FormalLanguageTheory-Simple/Main.cpp
Models/Number-Fancy/Main.cpp
src/VirtualMachine/VirtualMachinePool.h
Models/NumberGame/Main.cpp
Models/RationalRules/Main.cpp

Classes

- struct has_operator_lessthan_impl< T, EqualTo >
- struct has_operator_lessthan< T, EqualTo >
- struct TypeHead< Args >
- struct HeadIfReferenceElseT< T, args >
- struct HeadIfReferenceElseT< T >

Functions

- template<typename X , typename... Ts>
  constexpr bool contains_type ()

  ∼∼

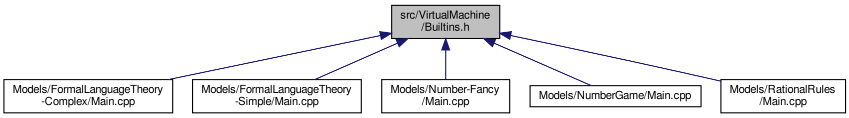
8.41.1 Function Documentation

8.41.1.1 contains_type()

```
template<typename X , typename...  Ts>
constexpr bool contains_type ( )
```

8.42 src/VirtualMachine/Builtins.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

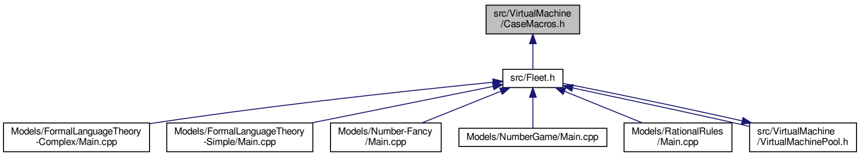
- struct BuiltinPrimitive< t, args >
- struct Builtin::If< t >
- struct Builtin::X< t >
- struct Builtin::Flip
- struct Builtin::FlipP
- struct Builtin::Recurse< t_out, t_in >
- struct Builtin::SafeRecurse< t_out, t_in >
- struct Builtin::SafeMemRecurse< t_out, t_in >

Namespaces

- Builtin

8.43 src/VirtualMachine/CaseMacros.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define CASE_FUNC0(opcode, returntype, f)
- #define CASE_FUNC1(opcode, returntype, a1type, f)
- #define CASE_FUNC2(opcode, returntype, a1type, a2type, f)
- #define CASE_FUNC3(opcode, returntype, a1type, a2type, a3type, f)
- #define CASE_FUNC4(opcode, returntype, a1type, a2type, a3type, a4type, f)
- #define CASE_FUNC0e(opcode, returntype, f, errcheck)
- #define CASE_FUNC1e(opcode, returntype, a1type, f, errcheck)
- #define CASE_FUNC2e(opcode, returntype, a1type, a2type, f, errcheck)
- #define CASE_FUNC3e(opcode, returntype, a1type, a2type, a3type, f, errcheck)
- #define CASE_FUNC4e(opcode, returntype, a1type, a2type, a3type, a4type, f, errcheck)

8.43.1 Macro Definition Documentation

8.43.1.1 CASE_FUNC0

```
#define CASE_FUNC0(
                opcode,
                returntype,
                f )
```

Value:

```
case opcode: {                                                               \
        vms.template push<returntype>(std::move(f()));                                      \
        break;                                                                   \
    }                                                                            \
```

8.43.1.2 CASE_FUNC0e

```
#define CASE_FUNC0e(
                opcode,
                returntype,
                f,
                errcheck )
```

Value:

```
case opcode: {                                                               \
        abort_t e = errcheck();                                                  \
        if(e != abort_t::GOOD) return e;                                                 \
        vms.template push<returntype>(std::move(f()));                                      \
        break;                                                                   \
    }                                                                            \
```

8.43.1.3 CASE_FUNC1

```
#define CASE_FUNC1(
                opcode,
                returntype,
                a1type,
                f )
```

Value:

```
case opcode: {                                                               \
        a1type a1 = vms.template getpop<a1type>();                               \
        vms.template push<returntype>(std::move(f(a1)));                                    \
        break;                                                                   \
    }                                                                            \
```

8.43.1.4 CASE_FUNC1e

```
#define CASE_FUNC1e(
                opcode,
                returntype,
                a1type,
                f,
                errcheck )
```

Value:

```
case opcode: {                                                               \
        a1type a1 = vms.template getpop<a1type>();                               \
        abort_t e = errcheck(a1);                                                \
        if(e != abort_t::GOOD) return e;                                                 \
        vms.template push<returntype>(std::move(f(a1)));                                    \
        break;                                                                   \
    }                                                                            \
```

8.43.1.5 CASE_FUNC2

```
#define CASE_FUNC2(
                opcode,
                returntype,
                a1type,
                a2type,
                f )
```

Value:

```
case opcode: {                                                               \
        a1type a1 = vms.template getpop<a1type>();                               \
        a2type a2 = vms.template getpop<a2type>();                               \
        vms.template push<returntype>(std::move(f(a1,a2)));                                 \
        break;                                                                   \
    }                                                                            \
```

8.43.1.6 CASE_FUNC2e

```
#define CASE_FUNC2e(
                opcode,
                returntype,
                a1type,
                a2type,
                f,
                errcheck )
```

Value:

```
case opcode: {                                                                \
        a1type a1 = vms.template getpop<a1type>();                               \
        a2type a2 = vms.template getpop<a2type>();                               \
        abort_t e = errcheck(a1,a2);                                             \
        if(e != abort_t::GOOD) return e;                                                 \
        vms.template push<returntype>(std::move(f(a1,a2)));                                  \
        break;                                                                   \
    }                                                                            \
```

8.43.1.7 CASE_FUNC3

```
#define CASE_FUNC3(
                opcode,
                returntype,
                a1type,
                a2type,
                a3type,
                f )
```

Value:

```
case opcode: {                                                                \
        a1type a1 = vms.template getpop<a1type>();                               \
        a2type a2 = vms.template getpop<a2type>();                               \
        a3type a3 = vms.template getpop<a3type>();                               \
        vms.template push<returntype>(std::move(f(a1,a2,a3)));                               \
        break;                                                                   \
    }                                                                            \
```

8.43.1.8 CASE_FUNC3e

```
#define CASE_FUNC3e(
                opcode,
                returntype,
                a1type,
                a2type,
                a3type,
                f,
                errcheck )
```

Value:

```
case opcode: {                                                                \
        a1type a1 = vms.template getpop<a1type>();                               \
        a2type a2 = vms.template getpop<a2type>();                               \
        a3type a3 = vms.template getpop<a3type>();                               \
        abort_t e = errcheck(a1,a2,a3);                                          \
        if(e != abort_t::GOOD) return e;                                                 \
        vms.template push<returntype>(std::move(f(a1,a2,a3)));                               \
        break;                                                                   \
    }                                                                            \
```

8.43.1.9 CASE_FUNC4

```
#define CASE_FUNC4(
              opcode,
              returntype,
              a1type,
              a2type,
              a3type,
              a4type,
              f )
```

Value:

```
case opcode: {                                                               \
        a1type a1 = vms.template getpop<a1type>();                               \
        a2type a2 = vms.template getpop<a2type>();                               \
        a3type a3 = vms.template getpop<a3type>();                               \
        a4type a4 = vms.template getpop<a4type>();                               \
        vms.template push<returntype>(std::move(f(a1,a2,a3,a4)));                              \
        break;                                                                   \
    }
```

8.43.1.10 CASE_FUNC4e

```
#define CASE_FUNC4e(
              opcode,
              returntype,
              a1type,
              a2type,
              a3type,
              a4type,
              f,
              errcheck )
```

Value:

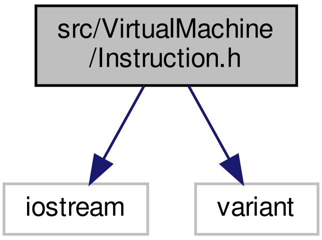
```
case opcode: {                                                               \
        a1type a1 = vms.template getpop<a1type>();                               \
        a2type a2 = vms.template getpop<a2type>();                               \
        a3type a3 = vms.template getpop<a3type>();                               \
        a4type a4 = vms.template getpop<a4type>();                               \
        abort_t e = errcheck(a1,a2,a3,a4);                                           \
        if(e != abort_t::GOOD) return e;                                                 \
        vms.template push<returntype>(std::move(f(a1,a2,a3,a4)));                              \
        break;                                                                   \
    }                                                                            \
```

8.44 src/VirtualMachine/Instruction.h File Reference

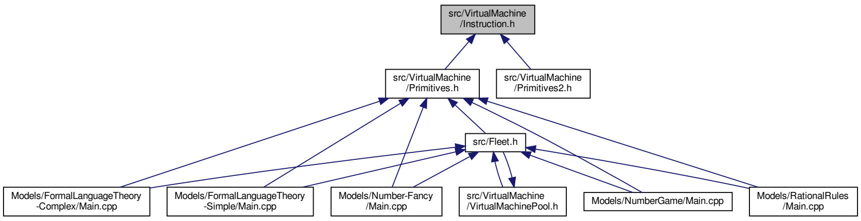
This is an error type that is returned if we get a runtime error (e.g. string length, etc.)

```
#include <iostream>
#include <variant>
```

Include dependency graph for Instruction.h:



This graph shows which files directly or indirectly include this file:



Classes

- class VMSRuntimeError_t
- class Instruction

Macros

- #define CUSTOM_OPS

Typedefs

- typedef short PrimitiveOp

Enumerations

- enum vmstatus_t {
  vmstatus_t::GOOD =0, vmstatus_t::ERROR, vmstatus_t::RECURSION_DEPTH, vmstatus_t::RANDOM_CHOICE,
  vmstatus_t::RANDOM_CHOICE_NO_DELETE, vmstatus_t::SIZE_EXCEPTION, vmstatus_t::OP_ERR_ABORT, vmstatus_t::RANDOM_BREAKOUT }
- enum CustomOp { CustomOp::CUSTOM_OPS }
- enum BuiltinOp {
  BuiltinOp::op_NOP =0, BuiltinOp::op_X, BuiltinOp::op_POPX, BuiltinOp::op_MEM,
  BuiltinOp::op_RECURSE, BuiltinOp::op_MEM_RECURSE, BuiltinOp::op_SAFE_RECURSE, BuiltinOp::op_SAFE_MEM_RECURSE,
  BuiltinOp::op_FLIP, BuiltinOp::op_FLIPP, BuiltinOp::op_IF, BuiltinOp::op_JMP,
  BuiltinOp::op_TRUE, BuiltinOp::op_FALSE, BuiltinOp::op_ALPHABET, BuiltinOp::op_ALPHABETchar,
  BuiltinOp::op_INT, BuiltinOp::op_P }

Functions

- std::ostream & operator<< (std::ostream &stream, Instruction &i)

Variables

- VMSRuntimeError_t VMSRuntimeError

8.44.1 Detailed Description

This is an error type that is returned if we get a runtime error (e.g. string length, etc.)

This is how we store an instruction in the program. It can take one of three types: BuiltinOp – these are operations that are defined as part of Fleet's core library, and are implemented automatically in VirtualMachineState::run PrimitiveOp – these are defined *automatically* through the global variable PRIMITIVES (see Models/RationalRules). When you construct a grammar with these, it automatically figures out all the types and automatically gives each a sequential numbering, which it takes some template magic to access at runtime CustomOp – these are for when you need more access to VMS' stack, and they require you to implement custom_dispatch (where the instruction is handled)

For any op type, an Instruction always takes an "arg" type that essentially allow us to define classes of instructions for instance, jump takes an arg, factorized recursion uses arg for index, in FormalLanguageTheory we use arg to store which alphabet terminal, etc.

8.44.2 Macro Definition Documentation

8.44.2.1 CUSTOM_OPS

```
#define CUSTOM_OPS
```

8.44.3 Typedef Documentation

8.44.3.1 PrimitiveOp

```
typedef short PrimitiveOp
```

8.44.4 Enumeration Type Documentation

8.44.4.1 BuiltinOp

```
enum BuiltinOp  [strong]
```

Enumerator

op_NOP	
op_X	
op_POPX	
op_MEM	
op_RECURSE	
op_MEM_RECURSE	
op_SAFE_RECURSE	
op_SAFE_MEM_RECURSE	
op_FLIP	
op_FLIPP	
op_IF	
op_JMP	
op_TRUE	
op_FALSE	
op_ALPHABET	
op_ALPHABETchar	
op_INT	
op_P	

8.44.4.2 CustomOp

```
enum CustomOp  [strong]
```

Enumerator

CUSTOM_OPS	

8.44.4.3 vmstatus_t

```
enum vmstatus_t  [strong]
```

Enumerator

GOOD	
ERROR	
RECURSION_DEPTH	
RANDOM_CHOICE	
RANDOM_CHOICE_NO_DELETE	
SIZE_EXCEPTION	
OP_ERR_ABORT	
RANDOM_BREAKOUT	

8.44.5 Function Documentation

8.44.5.1 operator<<()

```
std::ostream& operator<< (
            std::ostream & stream,
            Instruction & i )
```

Output for instructions.

Parameters

stream	
i	

Returns

8.44.6 Variable Documentation

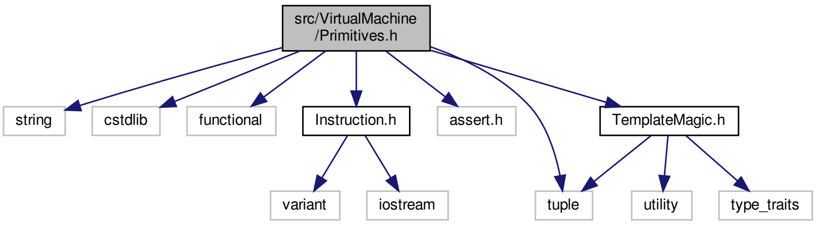
8.44.6.1 VMSRuntimeError

```
VMSRuntimeError_t VMSRuntimeError
```

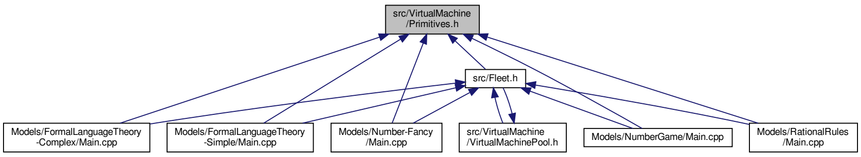
8.45 src/VirtualMachine/Primitives.h File Reference

```
#include <string>
#include <cstdlib>
#include <functional>
#include <tuple>
#include <assert.h>
#include "Instruction.h"
#include "TemplateMagic.h"
```

Include dependency graph for Primitives.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct PrePrimitive
- struct Primitive< T, args >

Namespaces

- Fleet::applyVMS

Functions

- template<int n, class T , typename V , typename P , typename L >
  vmstatus_t Fleet::applyVMS::applyToVMS_one (T &p, V ∗vms, P ∗pool, L ∗loader)
- template<class T , typename V , typename P , typename L , size_t... Is>
  vmstatus_t Fleet::applyVMS::applyToVMS (T &p, int index, V ∗vms, P ∗pool, L ∗loader, std::index_↩
  sequence< Is... >)
- template<class T , typename V , typename P , typename L >
  vmstatus_t applyToVMS (T &p, int index, V ∗vms, P ∗pool, L ∗loader)

8.45.1 Function Documentation

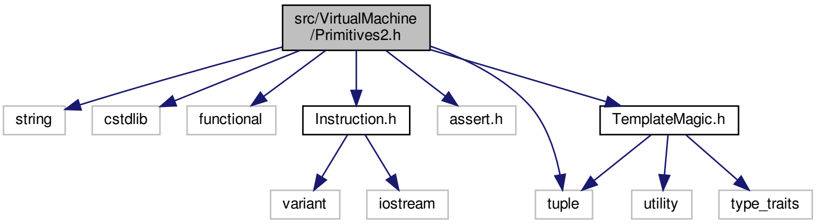
8.45.1.1 applyToVMS()

```
template<class T , typename V , typename P , typename L >
vmstatus_t applyToVMS (
            T & p,
            int index,
            V * vms,
            P * pool,
            L * loader )  [inline]
```

8.46 src/VirtualMachine/Primitives2.h File Reference

```
#include <string>
#include <cstdlib>
#include <functional>
#include <tuple>
#include <assert.h>
#include "Instruction.h"
#include "TemplateMagic.h"
```

Include dependency graph for Primitives2.h:



Classes

- struct PrePrimitive
- struct Primitive< T, args >

Functions

- template<typename V >
  vmstatus_t applyToVMS (size_t index, V *vms)

8.46.1 Function Documentation

8.46.1.1 applyToVMS()

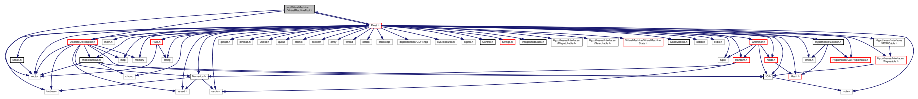
```
template<typename V >
vmstatus_t applyToVMS (
            size_t index,
            V * vms )
```

8.47 src/VirtualMachine/VirtualMachinePool.h File Reference

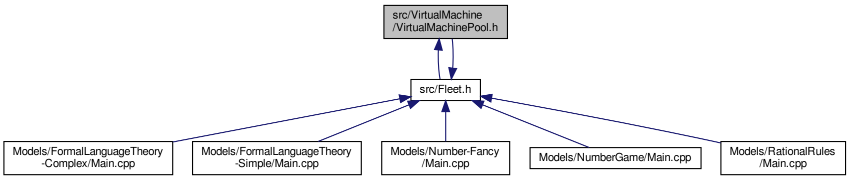
This manages a collection of VirtualMachines – this is what handles the enumeration of flip by probability. Basically each machine state stores the state of some evaluator and is able to push things back on to the Q if it encounters a random flip This stores pointers because it is impossible to copy out of std collections, so we are constantly having to call VirtualMachineState constructors. Using pointers speeds us up by about 20%.

```
#include "Fleet.h"
#include "DiscreteDistribution.h"
#include <vector>
```

Include dependency graph for VirtualMachinePool.h:



This graph shows which files directly or indirectly include this file:



Classes

- class VirtualMachinePool< t_x, t_return >

8.47.1 Detailed Description

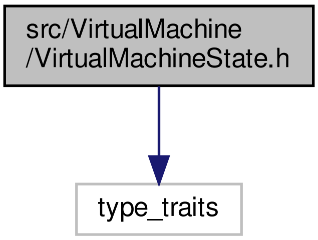
This manages a collection of VirtualMachines – this is what handles the enumeration of flip by probability. Basically each machine state stores the state of some evaluator and is able to push things back on to the Q if it encounters a random flip This stores pointers because it is impossible to copy out of std collections, so we are constantly having to call VirtualMachineState constructors. Using pointers speeds us up by about 20%.

8.48 src/VirtualMachine/VirtualMachineState.h File Reference

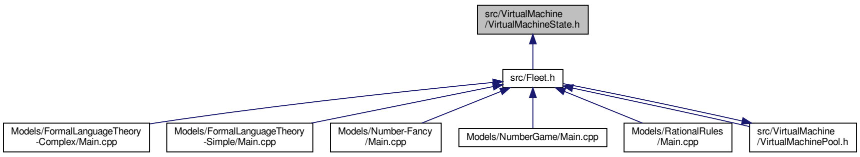
This represents the state of a partial evaluation of a program, corresponding to the value of all of the stacks of various types (which are stored as templates from FLEET_GRAMMAR_TYPES). The idea here is that we want to be able to encapsulate everything about the evaluation of a tree so that we can stop it in the middle and resume later, as is required for stochastics. This must be templated because it depends on the types in the grammar. These will typically be stored in a VirtualMachinePool and not called directly, unless you know that there are no stochastics.

```
#include <type_traits>
```

Include dependency graph for VirtualMachineState.h:



This graph shows which files directly or indirectly include this file:



Classes

- class VirtualMachineState< t_x, t_return >
- struct VirtualMachineState< t_x, t_return >::t_stack< args >

Functions

- void popn (Program &s, size_t n)

8.48.1 Detailed Description

This represents the state of a partial evaluation of a program, corresponding to the value of all of the stacks of various types (which are stored as templates from FLEET_GRAMMAR_TYPES). The idea here is that we want to be able to encapsulate everything about the evaluation of a tree so that we can stop it in the middle and resume later, as is required for stochastics. This must be templated because it depends on the types in the grammar. These will typically be stored in a VirtualMachinePool and not called directly, unless you know that there are no stochastics.

8.48.2 Function Documentation

8.48.2.1 popn()

```
void popn (
             Program & s,
             size_t n )
```