ATerm++ 0.1

Wieger Wesselink

Table of Contents

Preface 1
User's Guide 1
Introduction 1
Installing ATerm++ 1
Quick Start 2
Creating an aterm 3
Aterm properties 4
Lists and function applications 5
Initialization 7
Reference 7
Header </cygdrive/d/cpp/atermpp/atermpp/aterm.h> 7

Preface

Description

The ATerm++ library is a C++ wrapper around the ATerm Library, which is a C library for manipulating tree-like data structures. Important features of the ATerm Library are maximal subterm sharing, automatic garbage collection, and an efficient binary exchange format. The purpose of the ATerm++ Library is to provide the user with a common C++ interface, and to hide certain implementation details.

User's Guide

Introduction

What is the ATerm++ Library?

The ATerm++ Library is a manipulate tree-like term structures. The library is very space efficient, both in memory and on disk. An application areas are AST's (abstract syntax trees). This chapter describes how to use the ATerm++ Library.

Installing ATerm++

Building with ATerm++

The ATerm++ Library is a header-only template library, which means you don't need to alter your build scripts or link to any separate lib file to use it. All you need to do is `#include "atermpp/aterm.h"`.

Requirements

The ATerm++ Library has two dependencies.

In the first place it depends on the ATerm Library. You can download the latest version of the ATerm Library from CWI. Version 2.3.1 or higher is required.

In versions before 2.3.1 of the ATerm library a patch `aterm2.h` of the ATerm library needs to be patched with the following patch.

The effect of the patch is that anonymous structs like

```
typedef struct
{
        header_type header;
        ATerm       next;
        int         value;
} *ATermInt;
```

are named, in this case `_ATermInt`.

The ATerm++ Library also depends on Boost. You can download the latest version of the Boost libraries from www.boost.org. The ATerm++ Library requires Boost version 1.32 or higher. There is no need to install any library from Boost. Specifying the Boost include path is all that is needed.

Supported Compilers

Currently, the ATerm++ Library has only been tested on the following compilers:

- Visual C++ .NET 2003 (7.1)
- GNU C++ 3.3.3

Please send any questions, comments and bug reports to J.W.Wesselink at tue dot nl.

Quick Start

There are 6 different types of terms supported, as shown in the following table. The class `aterm` is a base class for all other aterms.

type	**description**
aterm	a general aterm
aterm_appl	an aterm containing a function application
aterm_int	an aterm containing an int
aterm_real	an aterm containing a real
aterm_blob	an aterm containing binary data
aterm_list	an aterm containing a list of aterms
aterm_place_holder	a place holder for aterms

Besides that the library contains a few more data types that use aterms internally.

type	description
dictionary	a dictionary based on aterms
table	a table based on aterms
indexed_set	an indexed set based on aterms

For the latter types alternatives are available in the C++ standard library like `set`, `multiset`, `map` and `multimap`.

The ATerm++ Library doesn't support user defined terms.

Creating an aterm

All aterm types have their own appropriate constructors for creating them:

```
aterm_int i(10);
aterm_real r(2.5);
aterm_appl f(function_symbol("f", 2), aterm("x"), aterm("y"));
```

There is also a convenience function `make_term` for easily creating aterms from strings: `make_term(const std::string& format, ...)`. The `format` argument is a string that may contain several patterns as given in the table below. For each occurrence of a pattern, one or more additional arguments need to be supplied to the function `make_term`.

type	pattern	argument
Application	<appl>	`string pattern, arguments`
Blob	<blob>	`int length, void* data`
Integer	<int>	`int value`
List	<list>	`aterm`
Placeholder	<placeholcer>	`string type`
Real	<real>	`double value`
String	<str>	`string pattern, arguments`
Term	<term>	`aterm`

The following program illustrates the usage of `make_term`.

```
#include <iostream>
#include "atermpp/aterm.h"

using namespace genspect;

void foo()
{
  const int i       = 42;
  const char* s     = "example";
  const char* blob  = "12345678";
  const double r    = 3.14;
```

```
  const char *func  = "f";

  aterm term[4];
  aterm list[3];
  aterm appl[3];

  term[0] = make_term("<int>" , i);         // integer value: 42
  term[1] = make_term("<str>" , func);      // quoted application: "f", no args
  term[2] = make_term("<real>", r);         // real value: 3.14
  term[3] = make_term("<blob>", 8, blob);   // blob of size 8, data: 12345678

  list[0] = make_term("[]");
  list[1] = make_term("[1,<int>,<real>]", i, r);
  list[2] = make_term("[<int>,<list>]", i+1, list[1]);

  appl[0] = make_term("<appl>", func);
  appl[1] = make_term("<appl(<int>)>", func, i);
  appl[2] = make_term("<appl(<int>, <term>, <list>)>", func, 42, term[3], list[2])

  std::cout << "appl[2] = " << appl[2] << std::endl;
}

int main(int argc, char *argv[])
{
  foo();
  return 0;
}
```

The function `match` can be used to extract pieces of aterms, as illustrated by the following program fragment:

```
aterm t = make_term("and(a,not(b))");
aterm t1;
aterm t2;
if (match(t, "and(<term>,<term>)", t1, t2))
{
  assert(t1 == aterm("a"));
  assert(t2 == aterm("not(b)"));
}
```

Aterm properties

The aterms in the ATerm++ Library have some properties that need to be understood to use the library effectively. The aterm classes all wrap pointers to ATerm objects of the underlying ATerm Library. Copying an aterm is thus a very cheap operation.

Constant objects

A very important property of aterm objects is that they are constant. All member functions of the aterm classes (except the assignment operator) are constant.

All aterm objects are constant. Whenever you want to modify an attribute of an aterm, a new object has to be created.

Conversions

The class `aterm` is a base class for all other aterm types. To simplify the conversion between `aterm` and its derivatives, there are member functions `to_int`, `to_appl` etc. available, like in the following example:

```
aterm_int x(10);
aterm y = x;
aterm_int z = y.to_int();
assert(z.value() == 10);
aterm_appl f = make_term("f(x,y)").to_appl();
```

One might expect that by the assignment to `y` all information about being an `aterm_int` would be lost. This is not the case! After the assignment both `x` and `y` wrap the same ATerm pointer. And thus it is possible to convert `y` back to an `aterm_int`.

Maximal sharing

The ATerm Library uses maximal sharing of terms, which makes it very memory efficient. Any time a new aterm is created, a lookup is done to see if it already exists. If so, that term is reused and no new term is created. The following code fragment illustrates how this works:

```
aterm_appl f = make_term("f(g(a,b),c)").to_appl();
aterm_appl g = make_term("g(a,b)").to_appl();
assert(f.argument(0) == g);

aterm_list v = make_term("[1,2,3,4]").to_list();
aterm_list w = make_term("[0,1,2,3,4]").to_list();
assert(pop_front(w) == v);
```

If two lists have the same tail, these tails will be shared.

Global aterm objects

The garbage collector of the ATerm Library assumes that all aterms that are in use can be found on the stack. Any aterm that isn't on the stack (for example global variables) needs to be protected explicitly.

Aterm objects that are not on the stack must be protected from being garbage collected, using the `protect` member function.

Annotations

Aterm objects may be annotated by other aterms. In the following example `y` is an annotated version of `x`.

```
aterm x("f(a)");
aterm label("label");
aterm annotation("annotation");
aterm y = set_annotation(x, label, annotation);
assert(x != y);
```

Aterms with different annotations are considered to be different.

Lists and function applications

The ATerm++ Library contains two types that can be used for creating hierarchies of terms: `aterm_list` (a singly linked list) and `aterm_appl` (a function application). Both can contain nested lists and function applications. For example:

```
aterm_list v("[1,[2,3],4]");
aterm_appl f("f(a,(g(b,c))"));
```

The class `aterm_list` has been modeled as a constant singly linked list. It has a C++ standard conforming iterator interface. Thus it operates well with the C++ Standard Library, as illustrated by the following example:

```
#include <algorithm>
#include <iostream>
#include "atermpp/aterm.h"

using namespace std;
using namespace genspect;

struct counter
{
  int& m_sum;

  counter(int& sum)
    : m_sum(sum)
  {}

  void operator()(const aterm& t)
  {
    m_sum += t.to_int().value();
  }
};

int main()
{
  aterm_list q = make_term("[1,2,3,4]").to_list();
  int sum = 0;
  for_each(q.begin(), q.end(), counter(sum));
  assert(sum == 10);

  for (aterm_list::iterator i = q.begin(); i != q.end(); ++i)
  {
    cout << *i;
  }
}
```

An `aterm_appl` is a function application. It consists of a function symbol (of type `function_symbol`) and a list of arguments (of type `aterm_list`).

The conversion to string doesn't always preserve the quoted attribute of a function application. See the example below.

```
function_symbol s("\"F\"", 1, false); // s == "F", not quoted
aterm_appl f(s, aterm("x"));
   // convert to string and back
aterm_appl g = make_term(f.to_string()).to_appl();
   assert(g.is_quoted());                // g.function() == F, quoted(!)
```

Initialization

The ATerm++ Library doesn't require any special initialization. The underlying ATerm initialization function `ATInit` is automatically called by including the file `atermpp/aterm.h`. If this behavior is undesired, it is possible to override this by defining the symbol `ATERM_USER_INITIALIZATION`. Then the user is responsible for adding something similar to

```
aterm bottom_of_stack;
init(argc, argv, bottom_of_stack);
```

Reference

Header </cygdrive/d/cpp/atermpp/atermpp/aterm.h>

This is a C++ wrapper around the ATerm library.

```
namespace genspect {
  class aterm;
  class aterm_appl;
  class aterm_blob;
  class aterm_int;
  class aterm_list;
  class aterm_list_iterator;
  class aterm_place_holder;
  class aterm_real;
  class dictionary;
  class function_symbol;
  class indexed_set;
  class table;
  bool operator==(const function_symbol &, const function_symbol &);
  bool operator!=(const function_symbol &, const function_symbol &);
  bool operator!(const aterm & x);
  bool operator==(const aterm &, const aterm &);
  bool operator!=(const aterm &, const aterm &);
  std::ostream & operator<<(std::ostream & out, const aterm & t);
  aterm_list push_front(aterm_list, aterm);
  aterm_list pop_front(aterm_list);
  aterm_list get_next(aterm_list);
  aterm_list tail(aterm_list, int);
  aterm_list replace_tail(aterm_list, aterm_list, int);
  aterm_list prefix(aterm_list);
  aterm get_last(aterm_list);
  aterm_list slice(aterm_list, int, int);
  aterm_list insert(aterm_list, aterm);
  aterm_list insert_at(aterm_list, aterm, int);
  aterm_list append(aterm_list, aterm);
  aterm_list concat(aterm_list, aterm_list);
  int index_of(aterm_list, aterm, int);
  int last_index_of(aterm_list, aterm, int);
  aterm element_at(aterm_list, int);
  aterm_list remove_element(aterm_list, aterm);
  aterm_list remove_all(aterm_list, aterm);
  aterm_list remove_element_at(aterm_list, int);
  aterm_list replace(aterm_list, aterm, int);
  aterm_list reverse(aterm_list);
  aterm read_from_string(const std::string &);
  aterm read_from_binary_string(const std::string &, unsigned int);
  aterm read_from_shared_string(const std::string &, unsigned int);
  aterm read_from_named_file(const std::string &);
  bool write_to_named_text_file(aterm, const std::string &);
```

```
  bool write_to_named_binary_file(aterm, const std::string &);
  aterm set_annotation(aterm, aterm, aterm);
  aterm get_annotation(aterm, aterm);
  aterm remove_annotation(aterm, aterm);
  void init(int, char *, aterm &);
  aterm_appl set_appl_argument(aterm_appl, unsigned int, aterm);
}
```

Class aterm

Class aterm --

genspect::aterm

```
class aterm {
public:
  // construct/copy/destruct
  aterm();
  aterm(ATerm);
  aterm(ATermList);
  aterm(ATermInt);
  aterm(ATermReal);
  aterm(ATermBlob);
  aterm(ATermAppl);
  aterm(ATermPlaceholder);
  aterm(const std::string &);

  // public member functions
  const ATerm & term() const;
  ATerm & term();
  operator ATerm() const;
  void protect();
  void unprotect();
  int type() const;
  std::string to_string() const;
  aterm annotation(aterm) const;
  aterm_blob to_blob() const;
  aterm_real to_real() const;
  aterm_int to_int() const;
  aterm_list to_list() const;
  aterm_appl to_appl() const;
};
```

Description

`aterm` construct/copy/destruct

1. `aterm();`

2. `aterm(ATerm term);`

3. `aterm(ATermList term);`

4. `aterm(ATermInt term);`

5. `aterm(ATermReal term);`

6. `aterm(ATermBlob term);`

7. `aterm(ATermAppl term);`

8. `aterm(ATermPlaceholder term);`

9. `aterm(const std::string & s);`

`aterm` public member functions

1. `const ATerm & term() const;`

2. `ATerm & term();`

3. `operator ATerm() const;`

4. `void protect();`

   Protect the aterm. Protects the aterm from being freed at garbage collection.

5. `void unprotect();`

   Unprotect the aterm. Releases protection of the aterm which has previously been protected through a call to protect.

6. `int type() const;`

   Return the type of term. Result is one of AT_APPL, AT_INT, AT_REAL, AT_LIST, AT_PLACEHOLDER, or AT_BLOB.

7. `std::string to_string() const;`

   Writes the term to a string.

8. `aterm annotation(aterm label) const;`

   Retrieve the annotation with the given label.

9.
```
aterm_blob to_blob() const;
```

10.
```
aterm_real to_real() const;
```

11.
```
aterm_int to_int() const;
```

12.
```
aterm_list to_list() const;
```

13.
```
aterm_appl to_appl() const;
```

Class aterm_appl

Class aterm_appl --

genspect::aterm_appl

```
class aterm_appl : public genspect::aterm {
public:
  // construct/copy/destruct
  aterm_appl();
  aterm_appl(ATermAppl);

  // public member functions
  function_symbol function() const;
  bool is_quoted() const;
  aterm argument(unsigned int) const;
  aterm_list argument_list() const;
  ATermAppl appl() const;
};
```

Description

Filter entries from a list using a predicate. This function can be used to filter entries that satisfy a given predicate from a list. Each item in list is judged through a call to predicate. If predicate returns true the entry is added to a list, otherwise it is skipped. The function returns the list containing exactly those items that satisfy predicate.

`aterm_appl` construct/copy/destruct

1. ```
   aterm_appl();
   ```

2. ```
   aterm_appl(ATermAppl a);
   ```

`aterm_appl` public member functions

1. ```
   function_symbol function() const;
   ```

   Get the function symbol (function_symbol) of the application.

2. ```
   bool is_quoted() const;
   ```

3. ```
   aterm argument(unsigned int i) const;
   ```

   Get the i-th argument of the application.

4.
   ```

```
aterm_list argument_list() const;
```

Get the list of arguments of the application.

5.
```
ATermAppl appl() const;
```

Returns the ATermAppl that is contained by the aterm_appl.

Class aterm_blob

Class aterm_blob --

genspect::aterm_blob

```
class aterm_blob : public genspect::aterm {
public:
  // construct/copy/destruct
  aterm_blob(ATermBlob);
  aterm_blob(unsigned int, void *);

  // public member functions
  void * data();
  unsigned int size() const;
};
```

Description

aterm_blob construct/copy/destruct

1. ```
   aterm_blob(ATermBlob t);
   ```

2. ```
   aterm_blob(unsigned int size, void * data);
   ```

   Build a Binary Large OBject given size (in bytes) and data. This function can be used to create an aterm of type blob, holding the data pointed to by data. No copy of this data area is made, so the user should allocate this himself. Note: due to the internal representation of a blob, size cannot exceed 224 in the current implementation. This limits the size of the data area to 16 Mb.

aterm_blob public member functions

1. ```
   void * data();
   ```

   Get the data section of the blob.

2. ```
   unsigned int size() const;
   ```

   Get the size (in bytes) of the blob.

Class aterm_int

Class aterm_int --

genspect::aterm_int

```
class aterm_int : public genspect::aterm {
public:
  // construct/copy/destruct
  aterm_int(ATermInt);
  aterm_int(int);

  // public member functions
  int value() const;
};
```

Description

aterm_int construct/copy/destruct

1.
```
aterm_int(ATermInt t);
```

2.
```
aterm_int(int value);
```

aterm_int public member functions

1.
```
int value() const;
```

Get the integer value of the aterm_int.

Class aterm_list

Class aterm_list --

genspect::aterm_list

```
class aterm_list : public genspect::aterm {
public:
  // types
  typedef aterm               value_type;
  typedef aterm *             pointer;
  typedef aterm &             reference;
  typedef const aterm         const_reference;
  typedef size_t              size_type;
  typedef ptrdiff_t           difference_type;
  typedef aterm_list_iterator iterator;
  typedef aterm_list_iterator const_iterator;

  // construct/copy/destruct
  aterm_list();
  aterm_list(ATermList);
  template<typename Iter> aterm_list(Iter, Iter);

  // public member functions
  const_iterator begin() const;
  const_iterator end() const;
  size_type size();
  size_type max_size();
  bool empty() const;
  void swap(aterm_list &);
  aterm front() const;
  const_iterator previous(const_iterator) const;
  void clear();
  operator ATermList() const;
  ATermList list() const;
};
```

Description

`aterm_list` construct/copy/destruct

1. `aterm_list();`

   Creates an empty aterm_list.

2. `aterm_list(ATermList l);`

   The copy constructor.

3. `template<typename Iter> aterm_list(Iter first, Iter last);`

   Creates an aterm_list with a copy of a range.

`aterm_list` public member functions

1. ```
   const_iterator begin() const;
   ```

   Returns an iterator pointing to the end of the aterm_list.

2. ```
   const_iterator end() const;
   ```

   Returns a const_iterator pointing to the beginning of the aterm_list.

3. ```
   size_type size();
   ```

   Returns the size of the aterm_list.

4. ```
   size_type max_size();
   ```

   Returns the largest possible size of the aterm_list.

5. ```
   bool empty() const;
   ```

   true if the list's size is 0.

6. ```
   void swap(aterm_list & l);
   ```

   Swaps the contents of two aterm_lists.

7. ```
   aterm front() const;
   ```

   Returns the first element.

8. ```
   const_iterator previous(const_iterator pos) const;
   ```

   pos must be a valid iterator in *this. The return value is an iterator prev such that ++prev == pos. Complexity: linear in the number of iterators in the range [begin(), pos).

9. ```
   void clear();
   ```

   Erases all of the elements.

10. ```
    operator ATermList() const;
    ```

    Conversion to ATermList.

11.
```
ATermList list() const;
```

Returns the ATermList that is contained by the aterm_list.

Class aterm_list_iterator

Class aterm_list_iterator --

genspect::aterm_list_iterator

```
class aterm_list_iterator {
public:
  // construct/copy/destruct
  aterm_list_iterator();
  aterm_list_iterator(ATermList);

  // public member functions

  // private member functions
  const aterm dereference() const;
  bool equal(aterm_list_iterator const &) const;
  void increment();
};
```

Description

aterm_list_iterator construct/copy/destruct

1. ```
   aterm_list_iterator();
   ```

2. ```
   aterm_list_iterator(ATermList l);
   ```

aterm_list_iterator public member functions

aterm_list_iterator private member functions

1. ```
   const aterm dereference() const;
   ```

2. ```
   bool equal(aterm_list_iterator const & other) const;
   ```

3. ```
   void increment();
   ```
   ```

Class aterm_place_holder

Class aterm_place_holder --

genspect::aterm_place_holder

```
class aterm_place_holder : public genspect::aterm {
public:
  // construct/copy/destruct
  aterm_place_holder(ATermPlaceholder);
  aterm_place_holder(aterm);

  // public member functions
  aterm type();
};
```

Description

aterm_place_holder construct/copy/destruct

1.
```
aterm_place_holder(ATermPlaceholder t);
```

2.
```
aterm_place_holder(aterm type);
```

Build an aterm_place_holder of a specific type. The type is taken from the type parameter.

aterm_place_holder public member functions

1.
```
aterm type();
```

Get the type of the aterm_place_holder.

Class aterm_real

Class aterm_real --

genspect::aterm_real

```
class aterm_real : public genspect::aterm {
public:
  // construct/copy/destruct
  aterm_real(double);
  aterm_real(ATermReal);

  // public member functions
  double value() const;
};
```

Description

`aterm_real` construct/copy/destruct

1.
```
aterm_real(double value);
```

2.
```
aterm_real(ATermReal t);
```

`aterm_real` public member functions

1.
```
double value() const;
```

Get the real value of the aterm_real.

Class dictionary

Class dictionary --

genspect::dictionary

```
class dictionary : public genspect::aterm {
public:
  // construct/copy/destruct
  dictionary();

  // public member functions
  aterm get(aterm);
  void put(aterm, aterm);
  void dict_remove(aterm);
};
```

Description

`dictionary` construct/copy/destruct

1. `dictionary();`

   Create a new dictionary.

`dictionary` public member functions

1. `aterm get(aterm key);`

   Get the value belonging to a given key in the dictionary.

2. `void put(aterm key, aterm value);`

   Add / update a (key, value)-pair in a dictionary. If key does not already exist in the dictionary, this function adds the (key, value)-pair to the dictionary. Otherwise, it updates the value to value.

3. `void dict_remove(aterm key);`

   Remove the (key, value)-pair from the dictionary. This function can be used to remove an entry from the dictionary.

Class function_symbol

Class function_symbol --

genspect::function_symbol

```
class function_symbol {
public:
  // construct/copy/destruct
  function_symbol(const std::string &, int, bool = false);
  function_symbol(AFun);

  // public member functions
  void protect();
  void unprotect();
  std::string name() const;
  unsigned int arity() const;
  bool is_quoted() const;
  operator AFun() const;
};
```

Description

`function_symbol` construct/copy/destruct

1. ```
   function_symbol(const std::string & name, int arity, bool quoted = false);
   ```

2. ```
   function_symbol(AFun function);
   ```

`function_symbol` public member functions

1. ```
   void protect();
   ```

   Protect the function symbol. Just as aterms which are not on the stack or in registers must be protected through a call to protect, so must function_symbols be protected by calling protect.

2. ```
   void unprotect();
   ```

   Release an function_symbol's protection.

3. ```
   std::string name() const;
   ```

   Return the name of the function_symbol.

4. ```
   unsigned int arity() const;
   ```

Return the arity (number of arguments) of the function symbol (function_symbol).

5.
```
bool is_quoted() const;
```

Determine if the function symbol (function_symbol) is quoted or not.

6.
```
operator AFun() const;
```

Conversion to AFun.

Class indexed_set

Class indexed_set --

genspect::indexed_set

```
class indexed_set {
public:
  // construct/copy/destruct
  indexed_set(unsigned int, unsigned int);
  ~indexed_set();

  // public member functions
  void reset();
  std::pair< long, bool > put(aterm);
  long index(aterm);
  aterm get(long);
  void remove(aterm);
  aterm_list elements();
};
```

Description

`indexed_set` construct/copy/destruct

1. ```
   indexed_set(unsigned int initial_size, unsigned int max_load_pct);
   ```

   Create a new indexed_set.

2. ```
   ~indexed_set();
   ```

   This function releases all memory occupied by the indexed_set..

`indexed_set` public member functions

1. ```
   void reset();
   ```

   Clear the hash table in the set. This function clears the hash table in the set, but does not release the memory. Using indexed_set_reset instead of indexed_set_destroy is preferable when indexed sets of approximately the same size are being used.

2. ```
   std::pair< long, bool > put(aterm elem);
   ```

   Enter elem into the set. This functions enters elem into the set. If elem was already in the set the previously assigned index of elem is returned, and new is set to false. If elem did not yet occur in set a new number is assigned, and new is set to true. This number can either be the number of an element that has been removed, or, if such a number is not available, the lowest non used number is

assigned to elem and returned. The lowest number that is used is 0.

3.
```
long index(aterm elem);
```

Find the index of elem in set. The index assigned to elem is returned, except when elem is not in the set, in which case the return value is a negative number.

4.
```
aterm get(long index);
```

Retrieve the element at index in set. This function must be invoked with a valid index and it returns the elem assigned to this index. If it is invoked with an invalid index, effects are not predictable.

5.
```
void remove(aterm elem);
```

Remove elem from set. The elem is removed from the indexed set, and if a number was assigned to elem, it is freed to be reassigned to an element, that may be put into the set at some later instance.

6.
```
aterm_list elements();
```

Retrieve all elements in set. A list with all valid elements stored in the indexed set is returned. The list is ordered from element with index 0 onwards.

Class table

Class table --

genspect::table

```
class table {
public:
  // construct/copy/destruct
  table(unsigned int, unsigned int);
  ~table();

  // public member functions
  void reset();
  void put(aterm, aterm);
  aterm get(aterm);
  void remove(aterm);
  aterm_list table_keys();
};
```

Description

`table` construct/copy/destruct

1. ```
   table(unsigned int initial_size, unsigned int max_load_pct);
   ```

   Create an table. This function creates an table given an initial size and a maximum load percentage. Whenever this percentage is exceeded (which is detected when a new entry is added using table_put), the table is automatically expanded and all existing entries are rehashed into the new table. If you know in advance approximately how many items will be in the table, you may set it up in such a way that no resizing (and thus no rehashing) is necessary. For example, if you expect about 1000 items in the table, you can create it with its initial size set to 1333 and a maximum load percentage of 75%. You are not required to do this, it merely saves a runtime expansion and rehashing of the table which increases efficiency.

2. ```
   ~table();
   ```

   Destroy an table. Contrary to aterm_dictionaries, aterm_tables are themselves not aterms. This means they are not freed by the garbage collector when they are no longer referred to. Therefore, when the table is no longer needed, the user should release the resources allocated by the table by calling table_destroy. All references the table has to aterms will then also be removed, so that those may be freed by the garbage collector (if no other references to them exist of course).

`table` public member functions

1. ```
   void reset();
   ```

   Reset an table. This function resets an ermtable, without freeing the memory it occupies. Its effect is the same as the subsequent execution of a destroy and a create of a table, but as no memory is re-
   ```

leased and obtained from the C memeory management system this function is gen- erally cheaper. but if subsequent tables differ very much in size, the use of table_destroy and table_create may be prefered, because in such a way the sizes of the table adapt automatically to the requirements of the application.

2.
```
void put(aterm key, aterm value);
```

Add / update a (key, value)-pair in a table. If key does not already exist in the table, this function adds the (key, value)-pair to the table. Otherwise, it updates the value to value.

3.
```
aterm get(aterm key);
```

Get the value belonging to a given key in a table.

4.
```
void remove(aterm key);
```

Remove the (key, value)-pair from table.

5.
```
aterm_list table_keys();
```

Get an aterm_list of all the keys in a table. This function can be useful if you need to iterate over all elements in a table. It returns an aterm_list containing all the keys in the table. The corresponding values of each key you are interested in can then be retrieved through respective calls to table_get.

Function operator==

Function operator== --

genspect::operator==

```
bool operator==(const function_symbol & x, const function_symbol & y);
```

Description

Tests equality of function symbols f1 and f2. Function symbols f1 and f2 are considered equal if they have the same name, the same arity and the same value for the quoted attribute.

Function operator!=

Function operator!= --

genspect::operator!=

```
bool operator!=(const function_symbol & x, const function_symbol & y);
```

Description

Returns !(x==y).

Function operator==

Function operator== --

genspect::operator==

```
bool operator==(const aterm & x, const aterm & y);
```

Description

Tests equality of aterms t1 and t2. As aterms are created using maximal sharing (see Section 2.1), testing equality is performed in constant time by comparing the addresses of t1 and t2. Note however that operator== only returns true when t1 and t2 are completely equal, inclusive any annotations they might have!

Function operator!=

Function operator!= --

genspect::operator!=

```
bool operator!=(const aterm & x, const aterm & y);
```

Description

Returns !(x==y).

Function push_front

Function push_front --

genspect::push_front

```
aterm_list push_front(aterm_list l, aterm elem);
```

Description

Inserts a new element at the beginning.

Function pop_front

Function pop_front --

genspect::pop_front

```
aterm_list pop_front(aterm_list l);
```

Description

Removes the first element.

Function get_next

Function get_next --

genspect::get_next

```
aterm_list get_next(aterm_list l);
```

Description

Returns the next part (the tail) of list l.

Function tail

Function tail --

genspect::tail

```
aterm_list tail(aterm_list l, int start);
```

Description

Return the sublist from start to the end of list l.

Function replace_tail

Function replace_tail --

genspect::replace_tail

```
aterm_list replace_tail(aterm_list l, aterm_list new_tail, int start);
```

Description

Replace the tail of list l from position start with new_tail.

Function prefix

Function prefix --

genspect::prefix

```
aterm_list prefix(aterm_list l);
```

Description

Return all but the last element of list l.

Function get_last

Function get_last --

genspect::get_last

```
aterm get_last(aterm_list l);
```

Description

Return the last element of list l.

Function slice

Function slice --

genspect::slice

```
aterm_list slice(aterm_list l, int start, int end);
```

Description

Get a portion (slice) of list l. Return the portion of list that lies between start and end. Thus start is included, end is not.

Function insert

Function insert --

genspect::insert

```
aterm_list insert(aterm_list l, aterm el);
```

Description

Return list l with el inserted. The behaviour of insert is of constant complexity. That is, the behaviour of insert does not degrade as the length of list increases.

Function insert_at

Function insert_at --

genspect::insert_at

```
aterm_list insert_at(aterm_list l, aterm el, int index);
```

Description

Return list l with el inserted at position index.

Function append

Function append --

genspect::append

```
aterm_list append(aterm_list l, aterm el);
```

Description

Return list l with el appended to it. Note that append is implemented in terms of insert by making a new list with el as the first element and then inserting all elements from list. As such, the complexity of append is linear in the number of elements in list. When append is needed inside a loop that traverses a list behaviour of the loop will demonstrate quadratic complexity.

Function concat

Function concat --

genspect::concat

```
aterm_list concat(aterm_list l, aterm_list m);
```

Description

Return the concatenation of the list l and m.

Function index_of

Function index_of --

genspect::index_of

```
int index_of(aterm_list l, aterm el, int start);
```

Description

Return the index of an aterm in a list. Return the index where el can be found in list. Start looking at position start. Returns -1 if el is not in list.

Function last_index_of

Function last_index_of --

genspect::last_index_of

```
int last_index_of(aterm_list l, aterm elem, int start);
```

Description

Return the index of an aterm in a list (reverse). Search backwards for el in list. Start searching at start. Return the index of the first occurrence of l encountered, or -1 when el is not present before start.

Function element_at

Function element_at --

genspect::element_at

```
aterm element_at(aterm_list l, int index);
```

Description

Return a specific element of a list. Return the element at position index in list. Returns `aterm()` when index is not in list.

Function remove_element

Function remove_element --

genspect::remove_element

```
aterm_list remove_element(aterm_list l, aterm elem);
```

Description

Return list with one occurrence of el removed.

Function remove_all

Function remove_all --

genspect::remove_all

```
aterm_list remove_all(aterm_list l, aterm el);
```

Description

Return list l with all occurrences of el removed.

Function remove_element_at

Function remove_element_at --

genspect::remove_element_at

```
aterm_list remove_element_at(aterm_list l, int index);
```

Description

Return list l with the element at index removed.

Function replace

Function replace --

genspect::replace

```
aterm_list replace(aterm_list l, aterm elem, int index);
```

Description

Return list l with the element at index replaced by el.

Function reverse

Function reverse --

genspect::reverse

```
aterm_list reverse(aterm_list l);
```

Description

Return list l with its elements in reversed order.

Function read_from_string

Function read_from_string --

genspect::read_from_string

```
aterm read_from_string(const std::string & s);
```

Description

Read an aterm from string. This function parses a character string into an aterm.

Function read_from_binary_string

Function read_from_binary_string --

genspect::read_from_binary_string

```
aterm read_from_binary_string(const std::string & s, unsigned int size);
```

Description

Read a aterm from a string in baf format. This function decodes a baf character string into an aterm.

Function read_from_shared_string

Function read_from_shared_string --

genspect::read_from_shared_string

```
aterm read_from_shared_string(const std::string & s, unsigned int size);
```

Description

Read a aterm from a string in taf format. This function decodes a taf character string into an aterm.

Function read_from_named_file

Function read_from_named_file --

genspect::read_from_named_file

```
aterm read_from_named_file(const std::string & name);
```

Description

Read an aterm from named binary or text file. This function reads an aterm file filename. A test is performed to see if the file is in baf, taf, or plain text. "-" is standard input's filename.

Function write_to_named_text_file

Function write_to_named_text_file --

genspect::write_to_named_text_file

```
bool write_to_named_text_file(aterm t, const std::string & filename);
```

Description

Writes term t to file named filename in textual format. This function writes aterm t in textual representation to file filename. "-" is standard output's filename.

Function write_to_named_binary_file

Function write_to_named_binary_file --

genspect::write_to_named_binary_file

```
bool write_to_named_binary_file(aterm t, const std::string & filename);
```

Description

Writes term t to file named filename in Binary aterm Format (baf).

Function set_annotation

Function set_annotation --

genspect::set_annotation

```
aterm set_annotation(aterm t, aterm label, aterm annotation);
```

Description

Annotate a term with a labeled annotation. Creates a version of t that is annotated with annotation and labeled by label.

Function get_annotation

Function get_annotation --

genspect::get_annotation

```
aterm get_annotation(aterm t, aterm label);
```

Description

Retrieves annotation of t with label label. This function can be used to retrieve a specific annotation of a term. If t has no annotations, or no annotation labeled with label exists, `aterm()` is returned. Otherwise the annotation is returned.

Function remove_annotation

Function remove_annotation --

genspect::remove_annotation

```
aterm remove_annotation(aterm t, aterm label);
```

Description

Remove a specific annotation from a term. This function returns a version of t which has its annotation with label label removed. If t has no annotations, or no annotation labeled with label exists, t itself is returned.

Function init

Function init --

genspect::init

```
void init(int argc, char * argv, aterm & bottom_of_stack);
```

Description

Initialise the ATerm++ Library. This call has only effect if the symbol ATERM_USER_INITIALIZATION is defined. See the section about initialization.

Function set_appl_argument

Function set_appl_argument --

genspect::set_appl_argument

```
aterm_appl set_appl_argument(aterm_appl appl, unsigned int i, aterm term);
```

Description

Set the i-th argument of an application to term. This function returns a copy of appl with argument i replaced by term.