A Framework for Debugging

Olivier, P.A.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
The operations described in Section 6.2 are not sufficient for all applications. Some applications need more control over the underlying implementation, or need operations that can be implemented using level one constructs but can be expressed more concisely and implemented more efficiently using more specialized constructs.

We have therefore designed a level 2 interface that is a strict superset of the level 1 interface described in Section 6.2. Some new datatypes are introduced, as well as some new operations on ATerms.

The level 2 interface introduces 7 new datatypes. Except for the auxiliary datatype AFun for representing function symbols, they are subtypes of the ATerm datatype, and implement the different term types. These subtypes allow us to introduce operations that are only valid for one specific term type, instead of the general ATerm operations described earlier.

**ATermInt:** This datatype represents integer terms. The operations on ATermInt are:

- ATermInt ATmakeInt(Integer v): Construct a new integer term corresponding to the integer value v.
- Integer ATgetInt(ATermInt i): Retrieve the value of an integer term.

**ATermReal:** This datatype represents real-number terms. The operations on ATermReal are:

- ATermReal ATmakeReal(Real v): Construct a new real term.
- Real ATgetReal(ATermReal r): Retrieve the value of a real term.

**AFun:** An AFun consists of a string defining the function name, an arity, and an indication whether the symbol name is quoted or not. The operations on symbols are:

- AFun ATmakeAFun(String nm, Integer ar, Boolean q): Construct a new symbol. If a symbol with the given name nm, arity ar, and quotation q already exists, the existing symbol is returned. Otherwise a new symbol is created and returned. AFuns are also subject to garbage collection in order to avoid long running (interactive) programs from slowly running out of symbols.
E / LEVEL 2 INTERFACE FOR ATERMS

- String ATgetNanme(AFun s): Retrieve the name of symbol s.
- Integer ATgetArity(AFun s): Retrieve the arity of a symbol.
- Boolean ATisQuoted(AFun s): Check if a symbol is quoted.

ATermAppl: This datatype represents function applications consisting of a function symbol and a number of arguments. The operations on this datatype are:

- ATermAppl ATmakeAppln(AFun f, ATerm a0, ..., ATerm an): This is a family of operations, one for each n between 0 and 6 (inclusive). These operations are used to construct a new function application with the given function symbol f and arguments.
- ATermAppl ATmakeAppl(AFun f, ATermList as): Construct a new function application with the given function symbol f and a list of arguments args.
- AFun ATgetFun(ATermAppl ap): Retrieve the function symbol of a function application.
- ATerm ATgetArgument(ATermAppl ap, Integer n): Retrieve a specific argument.

ATermList: This datatype represents the binary list constructor. Element indices start at 0. Thus a list of length n has elements 0, ..., n − 1. The operations on ATermList are:

- ATermList ATmakeListn(ATerm e0, ..., ATerm en): This is a family of operations, one for each n between 0 and 6 (inclusive). These operations are used to quickly construct small lists of terms.
- Integer ATgetLength(ATermList l): Retrieve the length of l.
- ATerm ATgetFirst(ATermList l): Retrieve the first element of list l.
- ATermList ATgetNext(ATermList l): Retrieve all but the first element of list l.
- ATermList ATgetPrefix(ATermList l): Retrieve all but the last element of list l.
- ATerm ATgetLast(ATermList l): Retrieve the last element from list l.
- ATermList ATgetSlice(ATermList l, Integer from, Integer to): Retrieve the portion of list l from position from through to − 1.
- Boolean ATisEmpty(ATermList l): Check if list l contains zero elements.
- `ATermList ATinsert(ATermList l, ATerm e)`: Insert a single element `e` at the start of list `l`.

- `ATermList ATinsertAt(ATermList l, ATerm e, Integer i)`: Insert a single element `e` at position `i` in list `l`.

- `ATermList ATappend(ATermList l, ATerm e)`: Append a single element `e` to the end of list `l`.

- `ATermList ATconcat(ATermList l1, ATermList l2)`: Concatenate lists `l1` and `l2`.

- `Integer ATindexOf(ATermList l, ATerm e, Integer i)`: Search for an element `e` in list `l` and return the index of the first location where `e` is present. Start searching at index `i`. If the element is not present, return `-1`.

- `Integer ATlastIndexOf(ATermList l, ATerm e, Integer i)`: Search backwards for element `e` in list `l`, and return the index of the last location where the element is present. Start searching at index `i`. If the element is not present, return `-1`.

- `ATerm ATelementAt(ATermList l, Integer i)`: Retrieve element at position `i` from list `l`.

- `ATermList ATremoveElement(ATermList l, ATerm e)`: Remove once occurrence of element `e` from list `l`.

- `ATermList ATremoveElementAt(ATermList l, Integer i)`: Remove the element at position `i` from list `l`.

**ATermPlaceholder:** This datatype represents placeholder terms. The operations on this datatype are:

- `ATermPlaceholder ATmakePlaceholder(ATerm tp)`: Construct a new placeholder term.

- `ATerm ATgetPlaceholder(ATermPlaceholder ph)`: Retrieve the type of this placeholder.

**ATermBlob:** This datatype represents Binary Large OBject terms. The operations on ATermBlob are:

- `ATermBlob ATmakeBlob(Integer n, Data d)`: Construct a new blob term of size `n` and containing data `d`.

- `Integer ATgetBlobSize(ATermBlob b)`: Retrieve the size of blob `b`.

165
E/level 2 interface for aterms

- Data $\text{ATgetBlobData(ATErmBlob \ blob)}$: Retrieve the data pointer stored in blob $\blob$.

The memory management of blobs must be done explicitly by the application programmer.

Auxiliary: The level two interface provides functionality to create and manipulate user-defined hash tables.