ANC: ANDF Notation Compiler

François de Ferrière

Open Software Foundation
Research Institute

June 1993

This paper presents a tool which allows the production of any ANDF capsule according to the ANDF specification and encoding.

1. Introduction

The ANDF Notation Compiler (ANC) is a tool which translates an ASCII source file of ANDF language into an ANDF capsule[1]. ANC was written in support of the the ANDF Validation Suite (AVS) [OSF-AVS-93] being developed at the OSF Research Institute in Grenoble. AVS is a collection of handcrafted test-cases, written in ANDF. Their purpose is to check installer conformance to ANDF specifications.

DRA already provides and maintains a tool similar to ANC, named TNC. This tool is delivered with DRA's code drop. TNC could be used to write the AVS test cases, but there are some drawbacks:

1. ANDF is an architecture- and language- neutral distribution format being developed by OSF and other collaborators around the world. It is based on the TDF technology provided by the Defence Research Agency (DRA) of the UK Ministry of Defense.

- We do not have any control on the ANDF capsules produced. If TNC produces capsules according to unspecified rules or particular profiles, this will not be detected.
- TNC performs many checks to ensure both *sort* and *shape* correctness. This prevents generation of most of the error test cases.
- TNC provides all the constructs which define *data* and *code*, but does not give access to the building of a capsule. The AVS test cases need to check the whole ANDF specification.

For these reasons, an additional tool is needed to produce the ANDF capsules.

2. Objectives

ANC needs to meet several objectives to allow the production of the AVS tests cases.

Accept the whole ANDF specification

It must be possible to write any ANDF code which is allowed by the ANDF specification [ANDF-SPEC-92]. The generation of ANDF constructs and encoding must be unrestricted. The ANDF specification allows some parts of the ANDF to be extended, for future extensions or for private standards. This must be allowed.

Allow incorrect ANDF code

The AVS will contain error test cases for many constructs. They will be used to check that incorrect ANDF is detected and handled as described in the ANDF specification. So it must be possible to write incorrect ANDF code as well as malformed ANDF capsules. However, *sort* correctness cannot be checked since there is no information in the encoding to verify it. So it is of no use to generate ANDF with incorrect *sort*, and this is not allowed by ANC.

Tokens are defined with the *sort* of their parameters and result. As errors due to misused of tokens can be detected, this is allowed.

Provide facilities to write large amounts of ANDF code.

When there is no need to control the building of the ANDF capsule, this must be done automatically as this is a very repetitive and lengthy task. Another facility is to allow some ANDF constructs to be left out. There are many places in ANDF where the same construct for a *sort* is almost always used. For these cases, a construct can be defined as a default one for a *sort* and will be used implicitly each time it is expected but not present.

3. Using ANC.

Syntax

The syntax to call anc is:

```
anc [options ...] [input_file [output_file]]
```

Options are the following:

`-f` *`file`*
: Names the file which contains the ANDF syntax and encoding. If this option is not present, a default file is used.

`-I`*`directory`*
: Searches *`directory`* for include files. This option can be repeated to add more directories.

`-tld2`
: Produces a TLD2 unit. This unit is used by the ANDF Linker to resolve undefined *tags* and *tokens*.

`-no_tokdec`
: Suppresses the production of the TOKDEC unit when *tokens* are

declared. This unit is not required by any tool, but TOKDECs are useful for ANDF source code writing facilities.

`-no_check`
Disables writing facilities.

Input and output files:

`input_file`
This is the ANDF source file. STDIN will be used if `input_file` is omitted or if ‘`-`’ is used,

`output_file`
This is the ANDF capsule file. If this file is omitted, STDOUT will be used.

ANDF language syntax

The ANDF language is specified as a set of *sorts*, for which a list of constructs is defined. Constructs may take *sorts* as their parameters. Thus, an ANDF source file consists of constructs, named as in the ANDF specification, followed by their arguments, exactly as the constructs are defined.

Brackets must be used to enclose the elements of a list.

A ‘`-`’ character is used to indicate the absence of an optional argument.

Tags, *tokens*, *al_tags* and *labels* are positive integers in the ANDF encoding. ANC allows the use of either integers or identifiers to identify them. When identifiers are used, the internal value is computed automatically, unless the `-no_check` option is used. In this case, the value must be given explicitly.

A number consists of any sequence of digits, which can be preceded by a ‘`-`’ sign. An octal number will be prefixed by a ‘`0`’, while a hexadecimal one will be prefixed by ‘`0x`’ or ‘`0X`’.

An identifier consists of any sequence of letters, digits, ‘_’ and ‘~’, not beginning with a digit and containing at least one letter.
The following syntax is used to explicitly assign an internal value to an

identifier:

```
        identifier:value
```

ANDF strings consist of a sequence of characters enclosed in ”, and the following escape sequences can be used:

- `\n` represents a newline
- `\t` represents a tabulation
- `\xxx` represents the character with the ASCII code xxx given in octal
- `\”` represents the `”` character.

Comments may be inserted anywhere in the source file. A comment begins with a ‘`#`’ character and runs to the end of the line.

Include files can be used. Each include command must be written on a single line with the following syntax:

```
        include ”file”
```

This can be enclosed in parentheses.

ANC facilities for writing ANDF

In many case, we need not write the complete ANDF code to express the right encoding. ANC provides many facilities to reduce the amount of writing.

- Many constructs can be omitted. This is allowed for constructs which are unique for a *sort*, or for constructs which are usually used for a *sort*.
  `make_link`, `make_linkextern` and `make_unique` are the only constructs of their *sort*, so they can be omitted.
  `make_al_tag`, `make_label`, `make_tag`, `make_tok`, `make_nat`, `make_signed_nat` are usually used for their *sort*, so they are marked as default construct for their *sort*.
  When a *token* has been declared or defined, and is referenced with an identifier, the `..._apply_token` constructs can be omitted as there is enough information to decide which construct must be used.
- The building of a capsule can be automatically done if needed. In that case, `make_capsule` is not used, but replaced by constructs for the *sorts* AL_TAGDEF, TAGDEC, TAGDEF, TOKDEC and TOKDEF. The syntax is exactly the same as that described in the ANDF specification, except

that a construct for the EXTERNAL *sort* may be added. The syntax is the following:

```
( make_tokdec my_tok ... )
( make_tokdec ( string_extern ″ext_name″ ) my_tok ...)
```

If an EXTERNAL construct is used, an external link will be created so that the *tag*, *token* or *al_tag* will be visible outside of the capsule.

- Building a capsule requires the specification of the number of linkable entities of each kind for the capsule and the units, and the number of *labels* used in PROPS. ANC will automatically compute this number if a ‘*’ is used instead of a numeric value.

Syntax and encoding file description

This file contains the ANDF syntax and encoding, plus some information to provide some facilities for writing ANDF. It can be semi-automatically generated from a database provided with DRA’s code drop.

In this file ANDF *sorts* are described, with their associated constructs.

A *sort* is defined according to the following syntax:

```
      :<Sort_name>:<encoding_bits>:<Flags>
```

`<Sort_name>` is the name of a *sort*.
`<encoding_bits>` is an integer value stating the number of encoding bits of the *sort*. If `<encoding_bits>` is zero, this means that the *sort* will not appear in the ANDF encoding.
`<Flags>` is used for additional information. Currently, only one flag is defined which indicates if the *sort* can be extended or not (See TDF encoding [DRA-ENC-92]). A ’`E`’ character means that the *sort* can be extended, another character means that the *sort* cannot be extended.

Constructs for a *sort* are listed following its definition. A construct is defined according to the following syntax:

```
<Construct_name>:<flags><encoding> [<args>...]
```

`<Construct_name>` is the name of a construct.
`<flags>` is used for additional information. Currently, only one flag is defined which indicates if the construct is a default one or not. A ’`*`’ character

means that this construct will be used when a construct for this *sort* is expected but is not present in the source file. At most one default construct per *sort* can be defined. A '`@`' means that this construct will be used when a *token* is read instead of a construct for this *sort*. At most one default *token* construct per *sort* can be defined. The flag is omitted if the construct is not a default one. When defining default constructs we must be careful to avoid infinite loops when they are applied. For example, the first argument of a default construct must not be of the same *sort* as the construct itself. Constructs for the SORTNAME *sort* can take a string '`(<Sort_name>)`' before the `<flags>`. This is used to define which *sort* the construct refers to. If it is absent, the uppercase string of the construct is used. '()' is used to indicate that the construct doesn't refer to any *sort*.

`<args>` describes the syntax for the construct arguments. They are defined using keywords, a set of special *sorts* and usual ANDF *sorts*. In the following, the notation `<args>` is used to express a possibly empty set of arguments.

Keywords are listed below:

- `OPTION { <args> }`
  An ANDF OPTION is used to encode `<args>`.
- `LIST { <args> }`
  The encoding will be a list of `<args>`. The syntax `( <args> )` can also be used.
- `ARRAY { <args> }`
  A set of `<arg>` is expected.
- `BYTESTREAM { <args> }`
  `<args>` will be encoded in a BYTESTREAM.
- `BITSTREAM { <args> }`
  `<args>` will be encoded in a BITSTREAM.
- `BYTE_ALIGN { <args> }`
  `<args>` will be byte aligned.
- `RESOLVE { <args> }`
  The semantics may be used to modify `<args>` according to the context.

Special *sorts* are listed below:

- `AUTO_PROP`
  This *sort* is used to accept the right constructs when `make_capsule` is not used.
- `TDFALL`
  Any *sort* is expected here. Although this is never true in ANDF, it can be used when semantics does not provide enough information to check the *sort*.
- `TDFSINT`
  A signed integer value is expected and will be encoded as a TDFBOOL for the sign and a TDFINT for the unsigned value.
- `TDFINT`
  An integer value is expected and will be encoded as a TDFINT.
- `TDFBOOL`
  An integer value is expected and will be encoded as a TDFBOOL.
- `TDFIDENT`
  An identifier is expected and will be encoded as a TDFIDENT.
- `TDFSTRING`
  A string is expected and will be encoded as a TDFSTRING.
- `TOKEN`
  This *sort* needs to be defined because of some special cases for *tokens*.
- `SORTNAME`
  This *sort* needs to be defined for the right definition of *tokens*.
- `EXTERNAL`
  This *sort* needs to be defined for the definition of external links.

Other *sorts* can be used to indicate the *sort* of an argument. Non special *sorts* need to be defined in the file, but not necessarily before being referenced.

A *sort* must define the `__main__` construct which is the entry point in the grammar. The `__make_auto__` construct is used instead of `make_capsule` when we want the capsule to be automatically built. The grammar cannot be modified without modifying the semantics part of ANC, unless the `-no_check` option is used.

4. Conclusion

ANC is still a prototype and needs to be improved to support the whole ANDF specification and to satisfy all our requirements. But it is already sufficient to write most of the AVS test cases and it will be used instead of TNC.

5. Bibliography

[DRA-SPEC-92]	The TDF specification - Issue 2.0 Revision 1 - December 1992. The Defence Research Agency, St. Andrews Road, Malvern, Worcestershire, WR14 3PS, United Kingdom.
[DRA-ENC-92]	TDF Bit Encoding - Issue 2.0 Revision 1 - December 1992. The Defence Research Agency, St. Andrews Road, Malvern, Worcestershire, WR14 3PS, United Kingdom.
[OSF-AVS-93]	ANDF Validation Suites Specifications - V1.0 - March 25, 1993.