APERO Tools User's Guide

Version 2.0

Charles Pecheur, Université de Liège (B), Systèmes et Automatique[1].
<pecheur@montefiore.ulg.ac.be>

INTRODUCTION

This document describes the usage and command line interface of a set of programs that offer facilities for specifying data types in LOTOS.

The APERO[2] data type pre-processors seek to offer a technical solution helping the specification of common data structures in LOTOS while keeping compatibility with the standard language, and therefore with existing tools. The proposed solution is based on two complementary mechanisms:

- APERO-SYN is a pre-processor that catches and expands some language extensions into standard LOTOS,
- APERO-LIB is an extension of the standard library mechanism, giving access to non-finite collections of generic data types (records, enumerated types etc.).

This architecture is illustrated in figure 1.

Both functions rely on a generic text transformation algorithm and externally specified transformation rules. These rules are defined in the APERO language, which is a pattern-based formalism derived from "Macro By Example", a macro processing formalism used in the Scheme language. On one hand, this facilitates the modification and/or extension of the provided extensions and library types. On the other hand, this allows several alternative transformation rules for the same set of facilities, where the translated specification is tuned for several environments (human reading, compiler, simulator…). See the document "APERO Language Reference" for a detailed description of this language.

[1]This work is supported by the CEC under EC-CA Project 001:76099 "EUCALYPTUS".
[2]APERO is a loose acronym for Act one PrE-pROcessor.

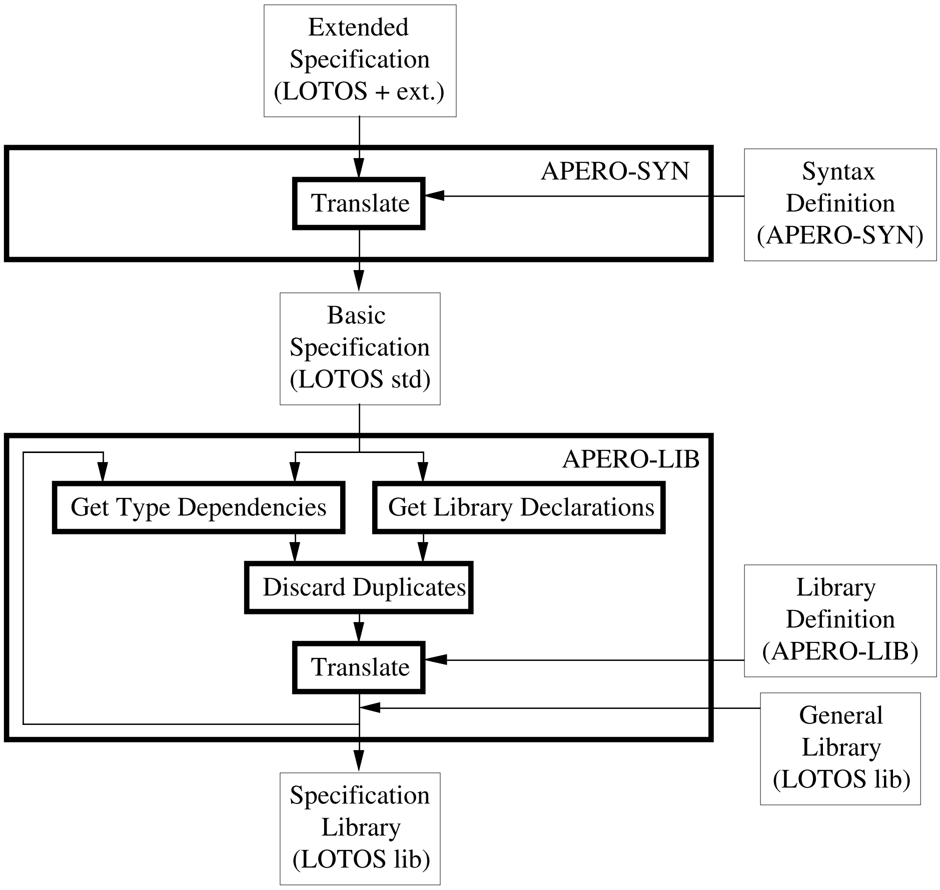

figure 1 — architecture of APERO tools

PROVIDED TOOLS

Four main facilities are provided:

apero-syn is a program performing the translation of syntax extensions, according to an APERO full transformation definition file. This is a ML exported executable file.

apero-lib is a program generating a LOTOS library for a standard LOTOS specification, according to an APERO single transformation definition file. This is a ML exported executable file.

apero is a front-end to apero-lib and apero-syn offering auxiliary facilities such as file location, checking of file availability, integration of both tools, default values etc. This is a UNIX shell script.

apero.mk is a makefile for automating access to APERO tools.

APERO-SYN

Synopsis

apero-syn *specfile* -x *execfile*

apero-syn *specfile* [-i *infile*] [-o *outfile*] [-r] [-v]

Parameters

specfile	APERO full transformation definition file for the provided syntax extensions.
-x *execfile*	generate a standalone executable in file *execfile*.
-i *infile*	read source text from *infile* (default is stdin).
-o *outfile*	write target text to *outfile* (default is stdout).
-r	apply transformation recursively.
-v	verbose - generate information messages.

Usage

apero-syn reads *infile*, applies an APERO full transformation defined in *specfile*, then performs a complementary transformation step dealing with library declarations (described below) and writes the final text to *outfile*. The transformation is applied recursively if option -r is present.

The associated lexical processing is based on LOTOS syntactical conventions:

- Scanning performs according to the following conventions:
    a. Any letter or digit, followed by any sequence of letters, digits or underscores, and not terminated by an underscore, is a word.
    b. Any sequence of characters from the set {#, %, &, +, -, *, ., /, <, =, >, @, \ , ^, {, }, ~} is a word.
    c. Any single non-white printable ASCII character not covered by rules (a) and (b) is a word.
    d. Any sequence of space or tabulation characters is white space.
    e. Any single other character is white space.
- Pretty printing inserts a space after all words that do not end with a white character.
- Word equivalence does not differenciate lower and upper case letters.

Note: this does not implement the full LOTOS lexical syntax. For example, the interleaving operator "|||" is scanned as three distinct words, according to rule (c). Also note that adjacent braces "{{" make up one word while adjacent brackets "[[" or parentheses "((" are separated. This is consistent with LOTOS syntax that allows braces in special identifiers. The current implementation also considers as white space any non-ASCII character such as ISO accented letters — this should change in the next version.

Library post-processing

After the full transformation has been applied, the text is re-arranged so that all library declarations within any type definition sequence are deleted from their original position, and replaced by a unique declaration at the end of the type definition sequence (either the global one or local ones within "where" clauses). During that process, duplicate declarations are detected and reduced to one.

This step is necessary to allow library declarations to be inserted during translation without violating LOTOS static rules that do not allow duplicate declarations.

This step uses a very crude syntax analysis, based on the detection of a few LOTOS keywords. It will work correctly with correct LOTOS specifications but will not detect most syntax errors.

Standalone executables

The -x option allows the generation of a standalone executable file that encapsulates the definition file as an internal data structure. All static checks are performed before the executable is generated. The standalone program accepts the same parameters as the second line of the above synopsis, except the initial specfile, i.e.

execfile [-i *infile*] [-o *outfile*] [-r] [-v]

Definition Guidelines

Though any APERO full transformation will be accepted, **apero-syn** is deemed to transform LOTOS data type definitions. As a guideline, all offered facilities should extend LOTOS syntax to:

data-type-definition ::= ...
| "`{facility}type`" *type-identifier* "`is`"
{facility}expression "`endtype`"

where {facility} is to be replaced by the name of the different offered facilities.

In APERO, this is handled by catching the pair ("`{facility}type`", "`endtype`") according to the following general template:

```
'{facility}type'..'endtype' :
        '{facility}type' T 'is'
        … (* extended syntax pattern *)
        'endtype'
->
        'type' T 'is'
        … (* standard LOTOS translation *)
        'endtype'
.
```

Since LOTOS *data-type-definition* always occur within sequences, the expansion may be a single or a sequence of *data-type-definition*.

Transparent extension definitions are quite difficult to achieve; there are usually potential conflicts between identifiers used within the translation and other identifiers in the specification. Such conflicts conditions should be reduced as much as possible and documented with the definitions of the extensions. In particular, variable identifiers should be well chosen since within their scope they will shadow any constant with the same name. As a guide line we suggest to names x, y and z for variables, suffixed with as many numeric indices as needed, separated by underscores (e.g. x, y_1, z_2_2_1).

Translations of syntax extensions may themselves use other syntax extensions using the recursive option. Termination of this loop is of course under the responsibility of the programmer.

All defined types should also offer the two equality predicates `_eq_` and `_ne_`, and may require their herited sorts to offer them.

APERO-LIB

Synopsis

apero-lib *specfile* -x *execfile*

apero-lib *specfile* [-i *infile*] [-o *outfile*] [-a *libfile*] [-v] [-t | -l]

Parameters

specfile	APERO single transformation definition file for the provided library types.
-x *execfile*	generate a standalone executable in file *execfile*.
-i *infile*	read LOTOS text from *infile* (default is stdin).
-o *outfile*	write library text to *outfile* (default is stdout).
-a *libfile*	include LOTOS library *libfile*.
-v	verbose - generate information messages.
-l	generate library according to library declarations (default).
-t	generate library according to undefined herited types.

Usage

By default, **apero-lib** reads *infile*, scans it for library declarations and adds all type identifiers that such declarations contain to a *todo* list of types to be generated. An initial empty *done* list of already generated types is also created.

The APERO single transformation defined by *specfile* is then applied in turn to each type identifier in *todo*. The resulting phrase is pretty printed and written to *outfile*. Moreover, this text is also scanned for type definitions and the defined and herited type identifiers are extracted of those definitions. Defined types are added to *done* and herited types are added to *todo* if they are not in *done*. The program terminates when *todo* becomes empty.

The translation of a given type identifier is expected to generate a definition for that type (i.e. an instance of the syntax class *data-type-definition*), though it may generate other definitions as well. If the expected definition is not found (e.g. because no matching transformation was provided), then an error is signalled and the type is added to done.

If -a *libfile* is given, then *libfile* is copied to *outfile* and scanned in the same way as generated definitions, filling the *done* and *todo* lists. This allows for inclusion of a normal LOTOS library in the generated library.

If the -t option is given, then the types of *infile* are taken into account instead of the library declarations, filling *todo* and *done* in the same way. This allows a "quick and dirty" library generation that attempts to generate library definitions for all type identifiers that are referenced but not defined in *infile*.

NB: library declaration and type definition scanning both use a very crude syntax analysis, based on the detection of a few LOTOS keywords. It will work correctly with correct LOTOS specifications but will not detect most syntax errors.

The lexical conventions are identical to those of **apero-syn** — see section on **apero-syn** for a description. Fused source patterns may be used to decompose the source type identifier into arbitrary components.

Standalone executables

The -x option allows the generation of a standalone executable file that encapsulates the definition file as an internal data structure. All static checks are performed before the executable is generated. The standalone program accepts the same parameters as the second line of the above synopsis, except the initial *specfile*, i.e.

execfile [-i *infile*] [-o *outfile*] [-a *libfile*] [-v] [-t | -l]

Definition Guidelines

apero-lib uses *specfile* as the definition of a single transformation that translates a (implicitly structured) type identifier into a LOTOS type definition (or a sequence of definitions). As a guideline, each family of similar types should be generated by one rule whose source pattern matches a type identifier with some fixed radix {family} :

```
"`f´`a´`m´`i´`l´`y´ p ..."   (* character-based pattern *)
-> ! param = "p ..." :       (* re-assemble the parameter *)
`type´ ("`f´`a´`m´`i´`l´`y´ p ...") `is´
…                            (* type definition body *)
`endtype´
```

Note that since the target phrase is scanned for type dependencies, library types may herit from other library types. Termination of this loop is of course under the responsibility of the programmer.

All defined types should also offer the two equality predicates `_eq_` and `_ne_`, and may require their herited sorts to offer them.

APERO

Synopsis

apero [-in *srcfile*] [-out *dstfile*] [-lib *libfile*] [-*debug*] [*srcfile*]
[-nosyn] [-syndef *syndeffile*] [-rec]
[-nolib] [-libdef *libdeffile*] [-include *libincfile*] [-usetypes]
[-defs *prefix*]

Parameters

-nosyn	no syntax translation.
-nolib	no library generation.
-usetypes	generate library from type dependencies instead of library declarations.
-rec	recursive syntax translation.
-in *srcfile*	read source specification from *srcfile* (mandatory).
srcfile	same as -in *srcfile*.
-out *dstfile*	write translated specification to *dstfile* (default extension is .lotos).
-lib *libfile*	write generated library to *libfile* (default extension is .lib).
-syndef *syndeffile*	read syntax extensions definition from *syndeffile* (default is pure/full.apsyn).
-libdef *libdeffile*	read library definition from *libdeffile* (default pure/full.aplib).
-include *libincfile*	read included library from *libincfile* (default is pure/full.apinc).
-defs *prefix*	shorthand for: -syndef *prefix*.apsyn -libdef *prefix*.aplib -include *prefix*.apinc.
-debug	generate debugging messages.

Options are evaluated sequentially from left to right: an option may be overridden by another that follows. The only mandatory parameter is *srcfile*.

Usage

In the general case, apero first processes the syntax extensions defined by *syndeffile* from an extended specification that is read from *srcfile*, writing the translated specification to *dstfile*. Then a library is generated in *libfile*, for the specification in *dstfile*, according to the definition in *libdeffile*, and including the standard library in *libincfile*.

If -nosyn is given, or if an usable syntax definition file cannot be found, no syntax extension is performed, and the library generation is applied to *srcfile* instead of *dstfile*. If -nolib is given, or if an usable library definition file cannot be found, no library generation is performed.

The APERO package is expected to be installed in some directory APERODIR, which defaults to /usr/local/apero. This value may be overridden by defining an environment variable $APERODIR. The default value may be easily modified in the **apero** shell script as well, though this requires write access to the APERO package. The script is located in ./$arch/bin, where $arch is the local processor architecture.

Default names are provided for the different APERO definition files (*syndeffile*, *libdeffile* and *libincfile*). These files may be located either in the current directory or in a directory APERODIR/lib. When looking for a syntax or library definition file *deffile*, if an executable file *deffile*.exe is found, it is assumed to be a standalone executable for that file and is used instead of the generic tool.

APERO.MK

Synopsis

make [*parm*=*val* …] *spec*.lotos

make [*parm*=*val* …] *spec*.lib

make [*parm*=*val* …] *deffile*.exe

Parameters

parm=*val*	parameter value for **make**. The parameters used by **apero.mk** are:
	APERODIR = root of APERO directory.
	DEFDIR = directory containing the definition files.
	DEFNAME = base name of definition files.
	SYNDEF = **apero-syn** definition file.
	LIBDEF = **apero-lib** definition file.
	SYNFLAGS = options for **apero-syn**.
	LIBFLAGS = options for **apero-lib**
spec.lotos	translated LOTOS specification.
spec.lib	generated LOTOS library.
deffile.exe	standalone executable APERO file.

Usage

apero.mk is a makefile, i.e. a "program" for the UNIX utility **make**. It is used by issuing the command **make** -f **apero.mk** (assuming a copy of **apero.mk** exists in the working directory), or renaming **apero.mk** to Makefile and using **make** without the -f option. It may also be used as a basis for writing application-specific makefiles.

apero.mk contains four rules:

- generate *spec*.lotos from *spec*.LOTOS via **apero-syn**,
- generate *spec*.lib from *spec*.lotos via **apero-lib**,
- generate a standalone **apero-syn** executable *def*.apsyn.exe from *def*.apsyn,
- generate a standalone **apero-lib** executable *def*.aplib.exe from *def*.aplib.

The first two take their definitions from parameters SYNDEF and LIBDEF, respectively. By default, these values are built from other parameters as follows:

APERODIR = /usr/local/apero

DEFDIR = $(APERODIR)/lib

DEFNAME = pure/full

SYNDEF = $(DEFDIR)/$(DEFNAME).apsyn

LIBDEF = $(DEFDIR)/$(DEFNAME).aplib

These values can be overridden by command line declarations, as described in the synopsis. Environment variable values (in particular $APERODIR) can also be used via the **make** option -e. Other options for **apero-syn** and **apero-lib** can de given in variables SYNFLAGS and LIBFLAGS, respectively. The default values may be easily modified in the **apero.mk** file as well, though this requires write access to the APERO package. The file is located in ./$arch/bin, where $arch is the local processor architecture.