NAME

uno – static analysis tool for ANSI-C programs

SYNOPSIS

uno [-D...] [-U...] [-I...] [-CPP=...] [-a] [-g x] [-h] [-l] [-m x] [-n] [-p x] [-s] [-t] [-u] [-V] [-v] [-w] [-x f] *.c

DESCRIPTION

UNO is a tool for analyzing programs written in ANSI-C. By default, the tool scans the sources for the three most commonly occuring defects of C programs: use of *u*ninitialized variables, *n*il-pointer dereferencing problems, and *o*ut-of-bound array indexing problems. It can optionally also report on a series of other, more cosmetic, flaws of the code, such as redundant variable and function declarations, unused fields in structures, variables set but not used, the use of conditions with side-effects, etc.

The UNO analysis proceeds in two phases: a local analysis of each function in the code, and a global analysis for the entire program. The local analysis can be done on also incomplete code, but the global analysis expects a complete program that can be searched starting from the *main()* routine.

UNO allows the user to define new properties to check for, by writing simple C-functions that encode the required check. The properties can specify either a local check, applied to each function separately, or a global check, applied to the program as a whole. The local check applies to the use of local variables of any type, the global check applies to the use of global pointers only.

The first group of options allows for the definition of compiler directives on the command line, to guide the preprocessing of the sources.

-Dname=def
Define **name** with value **def** as if by a *#define*.

-Dname
Define **name** with value 1.

-Idir Add directory **dir** to the list of directories that is searched by the preprocessor for include files.

-Uname
Remove any definitions of name, where name is a reserved symbol that may be predefined by the preprocessor. If present, this action supersedes the possible use of **-D** for the same symbol, irrespective of the order in which these options are given.

-CPP=...
Set the preprocessor to the name specified. For instance, **CPP="cl -EP -nologo"**.

The next set of options controls how the analysis is performed.

-a Report all error paths in the local analyses, rather than only paths that end in distinct statements in the source.

-g x Check the *global* property definition stored in file *x*, instead of the default property for the use or dereferencing of uninitialized global pointers (by default initialized to zero). By convention, the property function must be declared as *void uno_check(void) {}*.

-h or **-help** Prints a usage summary with the main tool options.

-l Perform only the local analysis, do not write intermediate files.

-m f Use a master definitions file, with UNO type definitions, for the local analyses. This can be useful in cases where the source being analyzed is incomplete, e.g., header files are missing. The user can add terse declarations of symbol names that should be understood to be typenames by the UNO parser. By convention this is done in a file named **_uno_.dfn**, which is placed in the same directory where UNO is invoked. The file may contains entries of the form: UnoType bool; UnoType complex;

which suffice to identify them as typenames to the tool, without requiring further detail. Definitions are given one per line, and terminated by a semi-colon. The file may also contain any standard preprocessing command understood by ANSI-compliant C preprocessors. This can be used to avoid the expansion of macro names, for instance, so that they can be tracked in UNO properties, e.g.: #define assert(x) Assert(x) /* avoid macro-expansion */

If the filename for the definitions file is **_uno_.dfn**, and the file is placed in the directory where UNO is invoked, the definitions file will automatically be included. If the file name is different, or located elsewhere, the **-m** option can be used.

-n Ignore all preprocessing directives in the source files being analyzed. This can be useful for analyzing output from a preprocessor, where the directives can be non-ANSI compliant. Cross-referencing information to the original source files is lost in this case.

-p x Check the *local* property definition stored in file *x*. As with global checks, the property function must be declared as *void uno_check(void) {}*.

-s Print only the symbol table information for each source file, and exit.

-t Provide detailed function call traces for any error scenario found during the global analysis.

-u Complain about redundancies of all sorts.

-V Print the current UNO version number and exit.

-v Verbose mode, currently mostly for debugging purposes.

-w Picky, or lint-like, mode. Complains about a larger variety of things, including more cosmetic flaws in the code. Includes **-u** and **-l .**

-x f Declare f to be a function that does not return. This affects the control-flow of the program and can therefore be important for the results of the analysis. By default, only the functions named *exit* , *fatal* , and *panic* are presumed not to return control to the caller.

NOTES

Unless the **-l** flag is used, UNO writes a small intermediate file at the end of the local analysis for each source file. The intermediate files for all source files enables the global analysis. Each intermediate file has the same base-name as the *.c* source file from which it was generated, but with the extension *.uno* instead of *.c*. UNO cleans up the intermediate files at the end of the global analysis. For very large source trees it can be beneficial to preserve the *.uno* files in between subsequent analysis, so that they are only recreated when necessary.

SEE ALSO

More background information on the design of the tool, examples of properties and applications, can be found in:

G.J. Holzmann, ‘UNO: Static Source Code Checking for User-Defined Properties,’
Proc. IDPT 2002, 6th World Conf. on Integrated Design & Process Technology, June 2002, Pasadena, CA.