

# Espresso

## Two-level Boolean Minimization

*Michele Lora*

*Tiziano Villa*

University of Verona  
Dep. Computer Science  
Italy



# Agenda

- Introduction
- *espresso* - two-level Boolean minimization
- *espresso* – input file
  - Description format
  - Keywords
- *espresso* – Options
- Exercises

# Introduction

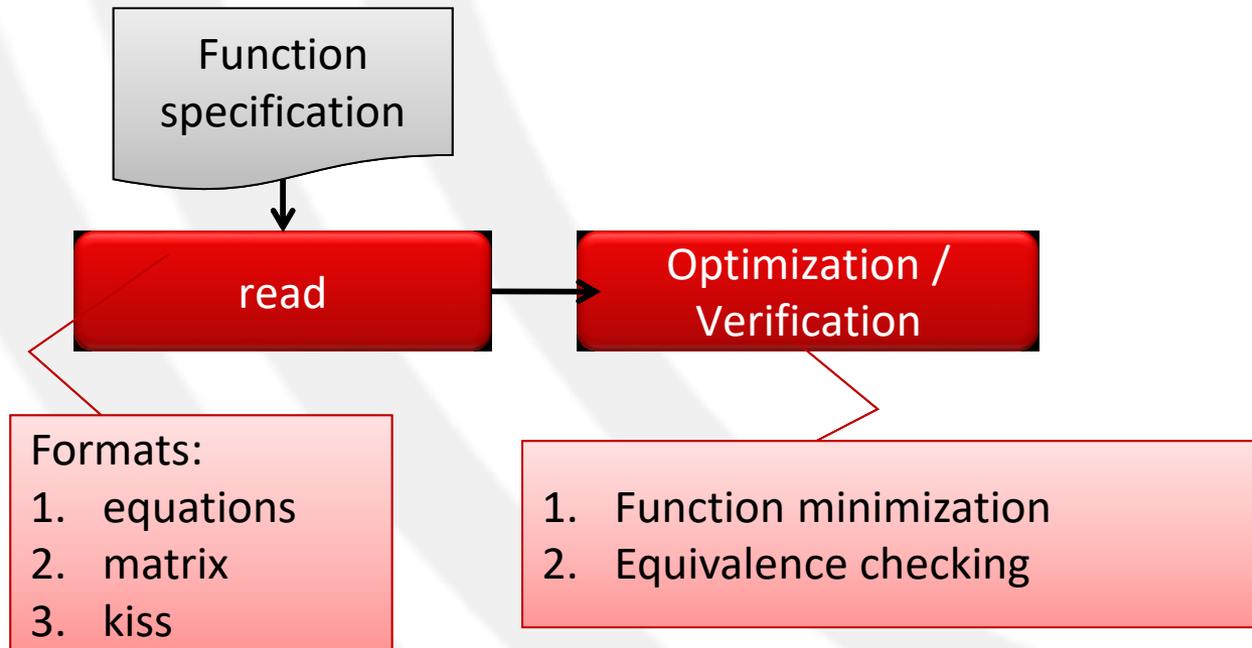
- A Boolean function can be described providing
  - **ON-set**
    - **OFF-set** is the **complement** of the **ON-set**
    - The DC-set is empty
  - **ON-set and DC-set**
    - **OFF-set** is the **complement** of the **union** of **ON-set** and **DC-set**
  - **ON-set and OFF-set**
    - **DC-set** is the **complement** of the **union** of **ON-set** and **OFF-set**
- A Boolean function is completely described by providing its **ON-set**, **OFF-set** and **DC-set**

# espresso – U.C. Berkeley

- *espresso* is a tool developed by the CAD group at U.C. Berkeley  
(software developer: Richard L. Rudell)
- Current release is the #2.3
  - Release date 31<sup>st</sup> January **1988**
- *espresso* is a program for **two-level Boolean minimization**

# espresso – Boolean Minimization

- Input:
  - A sum-of-product (SOP) representation of a two-valued (or multi-valued) Boolean function
- Output:
  - A minimal equivalent SOP representation



# espresso – Basic usage

```
$> espresso [options] [in_file] [>out_file]
```

- Reads the *in\_file* provided
  - Or the **standard input** if no file is specified
- Writes the minimized results in *out\_file*
  - **standard output** if the output is not redirected

# Example - Adder

$$sum = \overline{ain} * \overline{bin} * cin + \overline{ain} * bin * \overline{cin} + ain * \overline{bin} * \overline{cin} + ain * bin * cin$$

$$cout = ain * bin * \overline{cin} + \overline{ain} * bin * cin + ain * \overline{bin} * cin + ain * \overline{bin} * cin$$

ain	bin	cin	sum	cout
0	0	1	1	0
0	1	0	1	0
1	0	0	1	0
1	1	1	1	1
1	1	0	0	1
0	1	1	0	1
1	0	1	0	1

# espresso – Input file format (syntax)

- *espresso* accepts specifications described as a character matrix with keywords embedded
  - Keywords specify:
    - The size of the matrix
    - The format of the function
  - Comments:
    - Allowed using #
  - Whitespaces:
    - Blanks, tabs ... are ignored

# espresso – Input file format (semantics)

- Semantics of **input** part
  - The format of the function
    - each position in the input matrix corresponds to an input variable where:
      - “0” implies the corresponding input literal appears complemented in the product term
      - “1” implies the input literal appears uncomplemented in the product term
      - “-” implies the input literal does not appear in the product term

# espresso – Input file format (semantics)

- Semantics of **output** part
  - Specifying the format of the function
    - type *f*:
      - for each output, a **1** means this product term belongs to the *ON-set*, and **0** or **–** means this product term has *no meaning* for the value of this function
    - type *fd*:
      - for each output, a **1** means this product term belongs to the *ON-set*, **–** implies this product term belongs to the *DC-set* and a **0** means this product term has *no meaning* for the value of this function
      - it is the default type
    - type *fr*:
      - for each output, a **1** means this product term belongs to the *ON-set*, a **0** means this product term belongs to the *OFF-set*, and a **–** means this product term has *no meaning* for the value of this function
    - type *fdr*:
      - for each output, a **1** means this product term belongs to the *ON-set*, a **0** means this product term belongs to the *OFF-set*, a **–** means this product term belongs to the *DC-set*, and a **~** implies this product term has *no meaning* for the value of this function

# espresso – Input file keywords (I)

- The following keywords are recognized by *espresso*:
  - **.i** [d]
    - specifies the number “d” of input variables
  - **.o** [d]
    - specifies the number “d” of output variables
  - **.type** [s]
    - specifies the logical interpretation of the output part of the character matrix
    - this keyword must come before any product term
    - [s] is one of “f” “fd” “fr” “fdr”
  - **.e**
    - optionally marks the end of the description

```
# num of input vars
# e.g., ain, bin, cin
.i 3
# num of output functions
# e.g., sum, cout
.o 2

.type fr
0 0 1    1 0
0 1 0    1 0
1 0 0    1 0
1 1 1    1 1
0 1 1    0 1
1 0 1    0 1
1 1 0    0 1
.e
```

# espresso – Input file keywords (II)

– **.ilb** [s1] [s2] ..  
[sn]

- gives the names of the binary-valued variables
- must come after .i and .o
- as many tokens as input variables

– **.ob** [s1] [s2] .. [sn]

- gives the names of the output function
- must come after .i and .o
- as many tokens as output variables

```
.i 3
.o 2
.ilb ain bin cin
.ob sum cout
.type fr
0 0 1 1 0
0 1 0 1 0
1 0 0 1 0
1 1 1 1 1
0 1 1 0 1
1 0 1 0 1
1 1 0 0 1
.e
```

# espresso – input file keywords (III)

- **.phase** [b1] [b2] .. [bn]
  - specifies the phase of each output
    - positive (1) or negative (0)
  - must come after .i and .o
  - as many tokens as output variables
- **.p** [d];
  - specifies the number [d] of products
  - optional

# espresso – input file keywords (IV)

- Possible to use multi-valued variable
  - **.symbolic** [s0]..[sN] ; [t0] .. [tm] ;
    - the binary variables named [s0] thru [sN] must be considered as a single multiple-valued variable
    - variable with 2<sup>N</sup> parts corresponding to the decodes of the binary-valued variables
    - [s0] is the most significant bit, [sN] is the least significant bit
    - [t0] .. [tm] provide the labels for each decode of [s0] thru [sN]
  - **.mv** [num\_var] [num\_bin\_var] [d1] [dN]
    - specifies the number num\_var of variables, the number num\_bin\_var of binary variables and the size of each of the multiple-valued variables (d1 through dN)

# espresso - input file keywords (V)

```
.i 4
.o 3
.ilb ain<1> ain<0> bin<1> bin<0>
.ob sum<1> sum<0> cout

.symbolic ain<1> ain<0>
.symbolic bin<1> bin<0>
.symbolic sum<1> sum<0>

00 00 00 0
00 01 00 1
00 10 01 0
00 11 01 1
01 00 00 1
...
```

```
...
01 01 01 0
01 10 01 1
01 11 10 0
10 00 01 0
10 01 01 1
10 10 10 0
10 11 10 1
11 00 01 1
11 01 10 0
11 10 10 1
11 11 11 0
.e
```

# espresso – Options (I)

- Interesting options for running espresso are:
  - **-Dcheck**
    - checks that ON-set, OFF-set, DC-set are disjoint
  - **-Dexact**
    - performs exact minimization (potentially expensive)
  - **-Dmany**
    - reads and minimizes all PLA defined into the input file
  - **-Dopo**
    - performs output phase optimization, i.e., reduce the number of terms needed to implement the function or its complement

# espresso – Options (II)

## – -Dverify

- checks for Boolean equivalence of two functions
- requires two filenames from command line

## – -Dequiv

- identifies output variables which are equivalent

## – -Dso

- minimizes each function one at time as a single-output function

## – -epos

- swaps the ON-set and OFF-set of the function after reading the function
- useful for minimizing the OFF-set of a function

# Options (III)

- **-v** "
  - verbose debugging details
  - " activates all details
- **-d**
  - enables debugging
- **-o [type]**
  - selects the output format
  - type can be:
    - f: only On-set
    - fd: ON-set and DC-set
    - fr: ON-set and OFF-set
    - fdr: ON-set, OFF-set and DC-set

# U.C. Berkeley – Official release

- Official *espresso* release is available at <http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm>
  - Source code
  - Examples
  - Man pages for *espresso*



Berkeley  
UNIVERSITY OF CALIFORNIA

# Lab configuration for espresso

- Before starting:
  - Create the installation directory:
    - `$> mkdir sse_tools`
- Laboratory computers or Linux x86\_64
  - Download the binaries from the e-learning in the installation directory
  - Unzip it
    - `$> tar xzfv espresso_binaries.tar.gz`
    - `$> cd espresso`
- Other unix-based O.S. (support not tested)
  - Download the sources from the e-learning in the installation directory
  - Unzip it
    - `$> tar xzfv espresso_srcs.tar.gz`
  - Compile it
    - `$> cd espresso && make`
- Final steps:
  - Export the path to the executable
    - `$> export PATH=$PATH:$(pwd)/bin`
  - If you want, you can add the export of the path to your `.bashrc` file
- Windows:
  - **.exe** available on the e-learning

# Man pages

- Man pages are available
  - <http://user.engineering.uiowa.edu/~switchin/OldSwitching/espresso.5.html>
- PLA format manual (espresso.5)
  - see examples
    - #1, a two bit adder
    - #2, multi-valued function
    - #3, multi-valued function setup for *kiss*-style minimization
- *espresso* usage manual (espresso.1)
  - List options by `espresso -h`

# Exercise 1

- The Indian society of Natchez, who lived in North America, was divided into four groups: *Suns*, *Nobles*, *Honorables*, *Stinkards*. In this society, marriages were allowed according to specific rules, and the corresponding progeny belongs to a particular group as described in the following table:

Mother	Father	Progeny
Sun	Stinkard	Sun
Noble	Stinkard	Noble
Honorable	Stinkard	Honorable
Stinkard	Sun	Noble
Stinkard	Noble	Honorable
Stinkard	Honorable	Stinkard
Stinkard	Stinkard	Stinkard

- Other combinations are not allowed

## Exercise 2 (I)

- Formulate the minimum map coloring problem (coloring a map with the minimum number of colors such that adjacent regions don't have the same color) as a logic minimization problem.
- Apply your formulation to the following map and use *espresso* to find a minimum coloring for the map.

# Exercise 2 (II)

