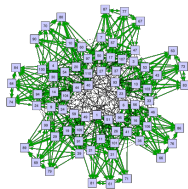


# Encoding the Linux Kernel Configuration in Propositional Logic

Christoph Zengler<sup>1</sup>   Wolfgang Küchlin<sup>1</sup>

<sup>1</sup>Symbolic Computation Group  
Wilhelm-Schickard-Institute of Informatics  
Universität Tübingen, Germany

16-08-2010



# The Linux Kernel

- highly configurable monolithic Kernel
- supports more than 60 different hardware platforms
- kernel sub-systems e.g. for
  - file systems
  - network protocols
  - memory management
  - ...
- thousands of options for compiling the kernel
- options are documented in `KConfig` files

# The Linux Kernel

- highly configurable monolithic Kernel
- supports more than 60 different hardware platforms
- kernel sub-systems e.g. for
  - file systems
  - network protocols
  - memory management
  - ...
- thousands of options for compiling the kernel
- options are documented in `KConfig` files

## Big question

Do these files accurately describe the configuration options?

This work takes the first step and formalizes the Linux Kernel Configuration

# Background

## Our Experience with Configuration

- configuration of car lines for the Daimler AG
- Boolean encodings of configuration problems
- large number of options
- current results: over  $10^{102}$  different configuration options for a single car line

## Our Experience with Linux

- Avinox — Automatic Verification of Linux Device Drivers
- verification of thousands of drivers with BMC
- toolchain for annotating, creating the environment and checking the drivers

# The LKC Configuration Options

- documented in `KConfig` files (configuration database)
- one entry for each configuration option
- most options have three different states:
  - `y` — compiled into the kernel
  - `m` — compiled as module
  - `n` — not available at all

## Dependencies

One option can *depend* on other options.

- If *A* depends on *B*, the evaluation of *B* is an *upper bound* for *A*

## Selections (Reverse Dependencies)

One option can force (*select*) other options.

- If *A* selects *B*, the evaluation of *A* is a *lower bound* for *B*

# Types of Configuration Options

Five different types of configuration options  $\mathcal{O}$

- 1 `tristate`: option can take three different values  $y, m, n$ , ( $\mathcal{O}_T$ )
- 2 `bool`: options can take two different values  $y, n$ , ( $\mathcal{O}_B$ )
- 3 `string`: option is an arbitrary string, ( $\mathcal{O}_S$ )
- 4 `hex`: option has a hexadecimal value, ( $\mathcal{O}_H$ )
- 5 `int`: option has an integer value, ( $\mathcal{O}_I$ )

## Domains

$\text{dom}(\mathcal{O}_T) = \{0, 1, 2\}$ ,  $\text{dom}(\mathcal{O}_B) = \{0, 2\}$ ,  $\text{dom}(\mathcal{O}_S) = \mathbb{S}$ ,  $\text{dom}(\mathcal{O}_I) = \mathbb{S}_I$ ,  
 $\text{dom}(\mathcal{O}_H) = \mathbb{S}_H$

- $\text{dom}(\mathcal{O}_B) \subset \text{dom}(\mathcal{O}_T)$
- $\text{dom}(\mathcal{O}_I) \subset \text{dom}(\mathcal{O}_S)$
- $\text{dom}(\mathcal{O}_H) \subset \text{dom}(\mathcal{O}_S)$

# The Tristate Logic

## Definition (Option Assignment)

An *option assignment* is a partial function  $\alpha_{\mathcal{O}} : \mathcal{O} \rightarrow \{0, 1, 2\} \cup \mathbb{S}$

- maps option of  $\mathcal{O}_T \cup \mathcal{O}_B$  to 0,1,2
- maps options of  $\mathcal{O}_S \cup \mathcal{O}_H \cup \mathcal{O}_I$  to  $\mathbb{S}$

Evaluation  $\beta(e)$  of a tristate expression ( $A \in \mathcal{O}_T \cup \mathcal{O}_B$ )

$$\beta(y) = 2$$

$$\beta(m) = 1$$

$$\beta(n) = 0$$

$$\beta(A) = \alpha_{\mathcal{O}}(A)$$

$$\beta(!e_0) = 2 - \beta(e_0)$$

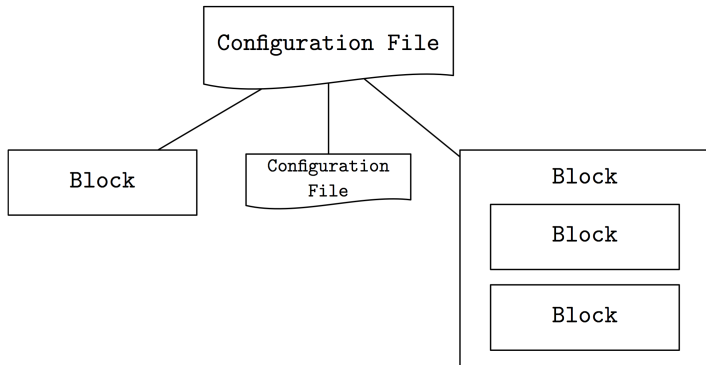
$$\beta(e_0 \&\& \dots \&\& e_n) = \min(\beta(e_0), \dots, \beta(e_n))$$

$$\beta(e_0 || \dots || e_n) = \max(\beta(e_0), \dots, \beta(e_n))$$

+ (dis)equalities for string, hex and int symbols

# The Config Files

Config files consist of blocks



- different kinds of blocks
- blocks can be nested
- other config files can be included

# The Configuration Block

## Configuration Block

A configuration block describes

- type
- dependencies
- selections

of a configuration option

## Configuration Block

```
config A
    tristate
    depends on !B
    depends on F && !(D || E)
    select G
    select H if X = "S"
```

# The Menu Block

## Menu Block

A menu block groups several other blocks

- is used to structure the configuration options
- can have multiple dependencies
- dependencies are propagated to all children blocks

## Menu Block

```
menu "menu title"
  depends on A

  config B
    type tristate

  config C
    type bool

endmenu
```

⇒

```
config B
  type tristate
  depends on A

config C
  type bool
  depends on A
```

# The Conditional Block

## Conditional Block

A conditional block groups several other blocks

- is used for conditional configuration options
- **has only one condition**
- condition is propagated to all children blocks

## Conditional Block

```
if A && (B || !C)
    config B
        type tristate
```

⇒

```
config C
    type bool
endif
```

```
config B
    type tristate
    depends on A && (B || !C)
```

```
config C
    type bool
    depends on A && (B || !C)
```

# The Choice Block

## Choice Block

A choice block groups several configuration options and can have one of two types:

- `bool`: At most one option  $y$ , all others  $n$
- `tristate`: At most one option  $y$ , all others  $m$  or  $n$

can also have dependencies which are propagated to all children

## Choice Block

```

choice
    tristate
    config A
        tristate

    config B
        tristate

endchoice

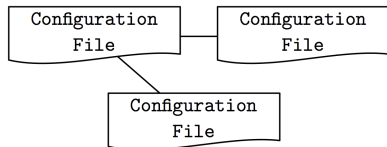
choice
    bool
    config A
        tristate

    config B
        tristate

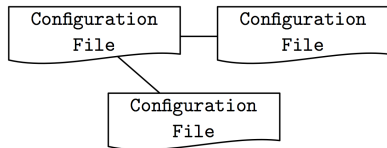
endchoice

```

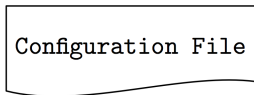
# The Translation Process



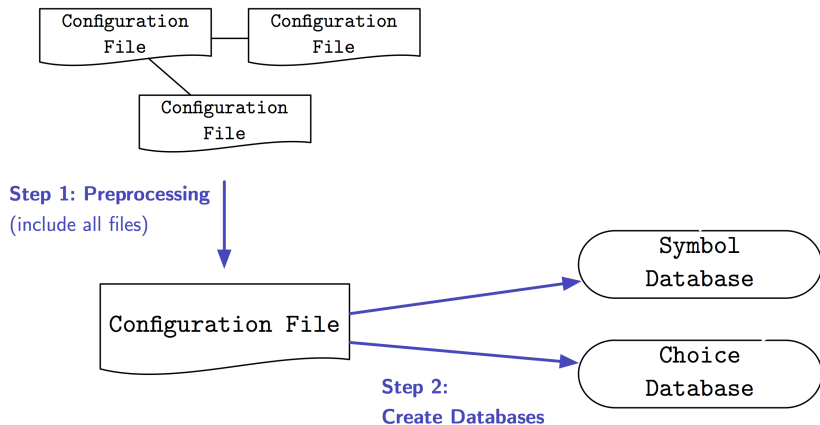
# The Translation Process



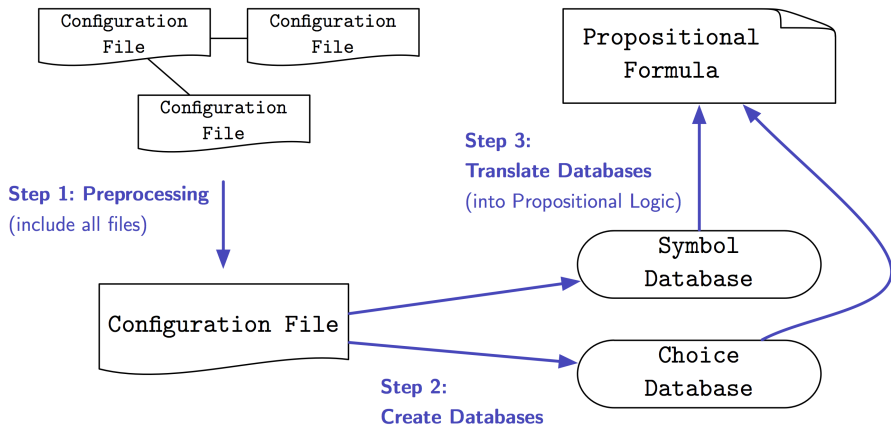
**Step 1: Preprocessing**  
(include all files)



# The Translation Process



# The Translation Process



# Create the Symbol Database

## Definition (Symbol Database)

Symbol database  $\mathcal{D}_S$

- mapping from configuration option names to *symbol descriptors*

Symbol descriptor consists of

- the type of the symbol
- the set of all dependencies
- the set of all selections

**Creation of  $\mathcal{D}_S$ :**

- recursive procedure createDB $_S$
- propagate dependencies of menu, conditional and choice blocks to children
- create database entries of configuration blocks

# The Symbol Database — Example

## Symbol Database

```
config X
  string
```

```
config B
  tristate
```

```
if B
  config C
    tristate
endif
```

```
config A
  tristate
  depends on !B
  select C
```

 $\Rightarrow$ 

$$\mathcal{D}_S = \{(X, (\text{string}, \emptyset, \emptyset)), \\ (B, (\text{tristate}, \emptyset, \emptyset)), \\ (C, (\text{tristate}, \{B\}, \emptyset)), \\ (A, (\text{tristate}, \{\neg B\}, \{C\}))\}$$

# Translate the Option Types

Add a constraint for each type of option

## `tristate` and `bool` Symbols

- encode 3 states with 2 bits

$A$	$a_0$	$a_1$
0	0	0
1	0	1
2	1	0

- $a_0 \leftarrow 1, a_1 \leftarrow 1$  not possible
- for `tristate` options  $A$ : add constraint  $\neg a_0 \vee \neg a_1$  to translation
- for `bool` options  $A$ : add constraint  $\neg a_1$  to translation

# Translate the Option Types

Add a constraint for each type of option

## `tristate` and `bool` Symbols

- encode 3 states with 2 bits

$A$	$a_0$	$a_1$
0	0	0
1	0	1
2	1	0

- $a_0 \leftarrow 1, a_1 \leftarrow 1$  not possible
- for `tristate` options  $A$ : add constraint  $\neg a_0 \vee \neg a_1$  to translation
- for `bool` options  $A$ : add constraint  $\neg a_1$  to translation

## `string`, `hex` and `int` Symbols

For symbol  $X$

- Gather all possible values  $X$  is compared to
- $X$  must take one of these values

# Translate the Tristate Expressions

Two projections for translation of tristate expression  $e$

- $\pi_0(e)$  the first bit of  $e$ 's encoding
- $\pi_1(e)$  the second bit of  $e$ 's encoding

Projections are one-to-one translations of the evaluation rules to bit-logic

## Projection Rule for $\&\&$

Projection for  $e = e_0 \&\& \dots \&\& e_n$

- $\pi_0(e) = \pi_0(e_0) \wedge \dots \wedge \pi_0(e_n)$
- $\pi_1(e) = \bigwedge_{i \in \{0, \dots, n\}} (\pi_0(e_i) \vee \pi_1(e_i)) \wedge \bigvee_{i \in \{0, \dots, n\}} \pi_1(e_i)$

# Example for Tristate Expression Translation

## Example Continued

Tristate expression:  $e = A \& \& ! B$

- $\pi_0(e) = a_0 \wedge \neg b_0 \wedge \neg b_1$
- $\pi_1(e) = (a_0 \vee a_1) \wedge ((\neg b_0 \wedge \neg b_1) \vee b_1) \wedge (a_1 \vee b_1)$

$A$	$B$	$a_0$	$a_1$	$b_0$	$b_1$	$\beta(e)$	$\pi_0(e)$	$\pi_1(e)$
0	0	0	0	0	0	<b>0</b>	<b>0</b>	<b>0</b>
0	1	0	0	0	1	<b>0</b>	<b>0</b>	<b>0</b>
0	2	0	0	1	0	<b>0</b>	<b>0</b>	<b>0</b>
1	0	0	1	0	0	<b>1</b>	<b>0</b>	<b>1</b>
1	1	0	1	0	1	<b>1</b>	<b>0</b>	<b>1</b>
1	2	0	1	1	0	<b>0</b>	<b>0</b>	<b>0</b>
2	0	1	0	0	0	<b>2</b>	<b>1</b>	<b>0</b>
2	1	1	0	0	1	<b>1</b>	<b>0</b>	<b>1</b>
2	2	1	0	1	0	<b>0</b>	<b>0</b>	<b>0</b>

# Translation of Dependencies and Selections

Each dependency and selection is translated

## Translation of Dependencies $\pi_D(s, e)$

Dependency:  $s$  depends on  $e$

- Assure that  $\alpha_O(s) \leq \beta(e)$

$$\pi_D(s, e) = \overbrace{(\neg\pi_0(e) \wedge \neg\pi_1(e) \rightarrow \neg s_0 \wedge \neg s_1)}^{\beta(e)=0 \rightarrow \alpha_O(s)=0} \wedge \overbrace{(\pi_1(e) \rightarrow \neg s_0)}^{\beta(e)=1 \rightarrow \alpha_O(s) \neq 2}$$

## Translation of Selections $\pi_S(s, s')$

Selection:  $s$  selects  $s'$

- Assure that  $\alpha_O(s') \geq \alpha_O(s)$

$$\pi_S(s, s') = \overbrace{(s_0 \rightarrow s'_0)}^{\alpha_O(s)=2 \rightarrow \alpha_O(s')=2} \wedge \overbrace{(s_1 \rightarrow s'_0 \vee s'_1)}^{\alpha_O(s)=1 \rightarrow \alpha_O(s') \neq 0} .$$

# Translate the Whole Symbol Database

$$\mathcal{D}_S = \{(B, (\text{tristate}, \emptyset, \emptyset)), \\ (C, (\text{bool}, \{B\}, \emptyset)), \\ (A, (\text{tristate}, \{!B\}, \{(C)\}))\}$$

- translation of the types:  $\{\neg b_0 \vee \neg b_1, \neg c_1, \neg a_0 \vee \neg a_1\}$

# Translate the Whole Symbol Database

$$\mathcal{D}_S = \{(B, (\text{tristate}, \emptyset, \emptyset)), \\ (C, (\text{bool}, \{B\}, \emptyset)), \\ (A, (\text{tristate}, \{!B\}, \{(C)\}))\}$$

- translation of the types:  $\{\neg b_0 \vee \neg b_1, \neg c_1, \neg a_0 \vee \neg a_1\}$
- translation of the dependencies:  $\{\pi_D(C, B), \pi_D(A, !B)\}$  with

$$\begin{aligned} \pi_D(C, B) &= ((\neg b_0 \wedge \neg b_1) \rightarrow (\neg c_0 \wedge \neg c_1)) \wedge (b_1 \rightarrow \neg c_0) \\ \pi_D(A, !B) &= (b_0 \rightarrow (\neg a_0 \wedge \neg a_1)) \wedge (b_1 \rightarrow \neg a_0) \end{aligned}$$

# Translate the Whole Symbol Database

$$\mathcal{D}_S = \{(B, (\text{tristate}, \emptyset, \emptyset)), \\ (C, (\text{bool}, \{B\}, \emptyset)), \\ (A, (\text{tristate}, \{!B\}, \{(C)\}))\}$$

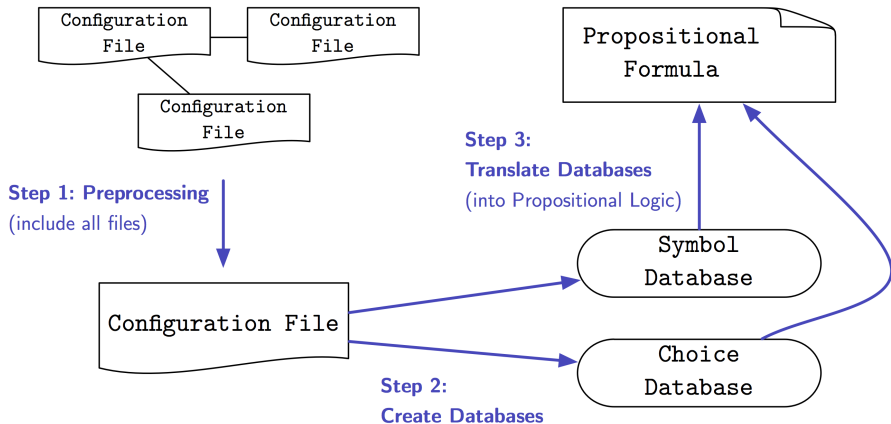
- translation of the types:  $\{\neg b_0 \vee \neg b_1, \neg c_1, \neg a_0 \vee \neg a_1\}$
- translation of the dependencies:  $\{\pi_D(C, B), \pi_D(A, !B)\}$  with

$$\begin{aligned} \pi_D(C, B) &= ((\neg b_0 \wedge \neg b_1) \rightarrow (\neg c_0 \wedge \neg c_1)) \wedge (b_1 \rightarrow \neg c_0) \\ \pi_D(A, !B) &= (b_0 \rightarrow (\neg a_0 \wedge \neg a_1)) \wedge (b_1 \rightarrow \neg a_0) \end{aligned}$$

- translation of the selections:  $\{\pi_S(A, C)\}$  with

$$\pi_S(A, C, ) = (a_0 \rightarrow c_0) \wedge (a_1 \rightarrow c_0 \vee c_1)$$

# Overview



# Experiments

- generated formulas for all hardware platforms
- no simplifications at all
- huge potential for enhancement (boolean variables, redundant clauses)

architecture	$ C_O $	$ C_C $	$ C_D $	$ C_S $	# vars	# clauses
alpha	5931	36	14525	3132	12807	227610
arm	6844	64	15630	4262	14249	246368
cris	3981	62	9097	1520	8772	136834
h8300	3467	33	8034	1473	7692	112149
ia64	6049	43	14775	3201	12999	230185
m68k	5874	35	14468	3125	12793	225735
mips	6321	53	14762	4026	13463	231300
powerpc	6403	46	14997	3634	13558	234287
sparc	5951	40	14571	3169	12811	226660
x86	6297	43	14978	3250	13402	250430

## Detecting Redundant and Necessary Options

**Problem:** find options that cannot be chosen at all (*redundant*) or must be chosen in each configuration (*necessary*)

- **Solution:** check translation formula  $\varphi \wedge \psi$  for additional conditions  $\psi$  with a SAT solver

### Example — revisited

```
config B
    tristate

if B
    config C
        tristate
    endif

config A
    tristate
    depends on !B
    select C
```

Can A be compiled into the kernel?

- Check  $\varphi \wedge a_0$
- Result: UNSAT  $\Rightarrow$  No!

Can A be compiled as module?

- Check  $\varphi \wedge a_1$
- Result: SAT  $\Rightarrow$  Yes!

A is neither redundant nor necessary

## Automatic Listing of Configuration Variants

**Problem:** list all options that are valid under certain conditions

- **Solution:** encode problem as a parametric SAT problem (PSAT)
- quantify options which should be hidden from the result
- **Result:** DNF, each minterm is one possible completion of the config.

### Example — revisited

```
config B
  tristate
```

```
if B
  config C
    tristate
```

```
endif
```

```
config A
  tristate
  depends on !B
  select C
```

Last example:  $A$  can be compiled as module

- **Question:** for which assignments of  $B$  (not  $C$ ) can  $A$  be compiled as module
- Check with PSAT

$$\exists c_0 \exists c_1 \exists a_0 \exists a_1 (\varphi \wedge a_1)$$

- **Result:**  $(\neg b_0 \wedge b_1)$

Only possible choice to compile  $A$  as module is also to compile  $B$  as module

# Conclusion & Outlook

## Achievements in this work

- first complete formalization of the LKC in propositional logic
- implementation of the algorithms
- experimental values
- possible applications

## Outlook

- run application examples
- simplify and compress the resulting formulas
- count valid configurations (at least of subsystems)
- automatically match configuration files with Make files