Formal Analysis of the Linux Kernel Configuration with SAT Solving

Martin Walch
Rouven Walter
Wolfgang Küchlin
Symbolic Computation Group
WSI for Informatics
University of Tübingen, Germany

2015-09-11

17^{th} International Configuration Workshop
Vienna, Austria 2015

Motivation

Variability of the Linux Kernel:

Motivation

Variability of the Linux Kernel:

- $> 12,300$ features (Linux 4.1)

Motivation

Variability of the Linux Kernel:

- $> 12,300$ features (Linux 4.1)
- 30 architectures

Motivation

Variability of the Linux Kernel:

- $> 12,300$ features (Linux 4.1)
- 30 architectures
- Statically configurable at compile time

Motivation

Variability of the Linux Kernel:

- $> 12,300$ features (Linux 4.1)
- 30 architectures
- Statically configurable at compile time
- $> 23,000$ explicit constraints

Motivation

Variability of the Linux Kernel:

- $> 12,300$ features (Linux 4.1)
- 30 architectures
- Statically configurable at compile time
- $> 23,000$ explicit constraints
- POF: $\sim 1,000,000$ variables in $\sim 2,000,000$ clauses

Motivation

Variability of the Linux Kernel:

- $> 12,300$ features (Linux 4.1)
- 30 architectures
- Statically configurable at compile time
- $> 23,000$ explicit constraints
- POF: $\sim 1,000,000$ variables in $\sim 2,000,000$ clauses

Hard questions:

Motivation

Variability of the Linux Kernel:

- $> 12,300$ features (Linux 4.1)
- 30 architectures
- Statically configurable at compile time
- $> 23,000$ explicit constraints
- POF: $\sim 1,000,000$ variables in $\sim 2,000,000$ clauses

Hard questions:

- Are there inadmissible features?

Motivation

Variability of the Linux Kernel:

- $> 12,300$ features (Linux 4.1)
- 30 architectures
- Statically configurable at compile time
- $> 23,000$ explicit constraints
- POF: $\sim 1,000,000$ variables in $\sim 2,000,000$ clauses

Hard questions:

- Are there inadmissible features?
- Are there necessary features? (Kaiser et Küchlin 2001 [1])

Motivation

Variability of the Linux Kernel:

- $> 12,300$ features (Linux 4.1)
- 30 architectures
- Statically configurable at compile time
- $> 23,000$ explicit constraints
- POF: $\sim 1,000,000$ variables in $\sim 2,000,000$ clauses

Hard questions:

- Are there inadmissible features?
- Are there necessary features? (Kaiser et Küchlin 2001 [1])
- Constraints violations possible?
- ...

Approach

- Encoding in Propositional Logic: “POF”
  (Küchlin et Sinz 2000 [2], Zengler et Küchlin 2010 [4])

Approach

- Encoding in Propositional Logic: “POF”
  (Küchlin et Sinz 2000 [2], Zengler et Küchlin 2010 [4])
- SAT-solving

Approach

- Encoding in Propositional Logic: “POF”
  (Küchlin et Sinz 2000 [2], Zengler et Küchlin 2010 [4])
- SAT-solving
- Similar project: VAMOS (Tartler et al. 2011 [3])

Approach

- Encoding in Propositional Logic: “POF”
  (Küchlin et Sinz 2000 [2], Zengler et Küchlin 2010 [4])
- SAT-solving
- Similar project: VAMOS (Tartler et al. 2011 [3])
- Main task: translation *Kconfig* $\rightarrow$ POF

Approach

- Encoding in Propositional Logic: “POF”
  (Küchlin et Sinz 2000 [2], Zengler et Küchlin 2010 [4])
- SAT-solving
- Similar project: VAMOS (Tartler et al. 2011 [3])
- Main task: translation *Kconfig* $\rightarrow$ POF
- Three intermediate representations

Kconfig Language

Kconfig

- describes features as *symbols*

Kconfig Language

Kconfig

- describes features as *symbols*
- symbols carry attributes, attributes have dependencies

Kconfig Language

Kconfig

- describes features as *symbols*
- symbols carry attributes, attributes have dependencies
- most constraints are dependencies of *attributes*

Kconfig Language

Kconfig

- describes features as *symbols*
- symbols carry attributes, attributes have dependencies
- most constraints are dependencies of *attributes*

Obstacles

- 1,195 files, hierarchically organized

Kconfig Language

Kconfig

- describes features as *symbols*
- symbols carry attributes, attributes have dependencies
- most constraints are dependencies of *attributes*

Obstacles

- 1,195 files, hierarchically organized
- Constraints interweaved with interface data

Kconfig Language

Kconfig

- describes features as *symbols*
- symbols carry attributes, attributes have dependencies
- most constraints are dependencies of *attributes*

Obstacles

- 1,195 files, hierarchically organized
- Constraints interweaved with interface data
- 1 symbol $\leftrightarrow$ 1 or more configuration blocks

Kconfig Language

Kconfig

- describes features as *symbols*
- symbols carry attributes, attributes have dependencies
- most constraints are dependencies of *attributes*

Obstacles

- 1,195 files, hierarchically organized
- Constraints interweaved with interface data
- 1 symbol $\leftrightarrow$ 1 or more configuration blocks
- Different types of dependencies

Kconfig Language

Kconfig

- describes features as *symbols*
- symbols carry attributes, attributes have dependencies
- most constraints are dependencies of *attributes*

Obstacles

- 1,195 files, hierarchically organized
- Constraints interweaved with interface data
- 1 symbol $\leftrightarrow$ 1 or more configuration blocks
- Different types of dependencies
- Attributes: different types, interaction, non-local constraints

Kconfig Language

Kconfig

- describes features as *symbols*
- symbols carry attributes, attributes have dependencies
- most constraints are dependencies of *attributes*

Obstacles

- 1,195 files, hierarchically organized
- Constraints interweaved with interface data
- 1 symbol $\leftrightarrow$ 1 or more configuration blocks
- Different types of dependencies
- Attributes: different types, interaction, non-local constraints
- Three-valued *"Tristate Logic"*

Listing 1: net/netfilter/Kconfig

```
83  ...
84  config NF_NAT_IPV4
85          tristate "IPv4 NAT"
86          depends on NF_CONNTRACK_IPV4
87          default m if NETFILTER_ADVANCED=n
88          select NF_NAT
89          help
90            The IPv4 NAT option allows masquerading,...
91            forms of full Network Address Port Trans...
92            controlled by iptables or nft.
93
94  if NF_NAT_IPV4
95  ...
```

Translation into Zengler Model

Translation into Zengler Model

Actions in this step:

Translation into Zengler Model

Actions in this step:

- Abstract from underlying input files

Translation into Zengler Model

Actions in this step:

- Abstract from underlying input files
- Strip configuration invariant data

Translation into Zengler Model

Actions in this step:

- Abstract from underlying input files
- Strip configuration invariant data
- Group configuration blocks by symbols

The Zengler Model

Database:

The Zengler Model

Database: Symbols as lists of configuration blocks

The Zengler Model

Database: Symbols as lists of configuration blocks

- dependencies as lists of Tristate expressions

The Zengler Model

Database: Symbols as lists of configuration blocks

- dependencies as lists of Tristate expressions
- lists of attributes

The Zengler Model

Database: Symbols as lists of configuration blocks

- dependencies as lists of Tristate expressions
- lists of attributes
  - visibility
  - default
  - select
  - range

Attributes Model

Attributes Model

Actions in this step:

Attributes Model

Actions in this step:

- Abstract from individual configuration blocks

Attributes Model

Actions in this step:

- Abstract from individual configuration blocks
- Collect dependencies for each attribute

Attributes Model

Actions in this step:

- Abstract from individual configuration blocks
- Collect dependencies for each attribute
- Store *Kconfig* "`select`" attributes with referenced symbols

Attributes Model

Actions in this step:

- Abstract from individual configuration blocks
- Collect dependencies for each attribute
- Store *Kconfig* "`select`" attributes with referenced symbols

Database:

- Symbols as sets of attributes
- Dependencies as lists of expressions in *Tristate Logic*

What is Tristate Logic?

What is Tristate Logic?

Tristate Logic is a three-valued logic

What is Tristate Logic?

Tristate Logic is a three-valued logic – Why?

What is Tristate Logic?

Tristate Logic is a three-valued logic – Why?
$\Rightarrow$ Features have three states of activation:

What is Tristate Logic?

Tristate Logic is a three-valued logic – Why?
$\Rightarrow$ Features have three states of activation:

- Inactive
- Runtime loadable module
- Integration at compile time

Tristate Logic

Domain: $\{0, 1, 2\}$

Tristate Logic

Domain: $\{0, 1, 2\}$

Three operators:

	!
0	2
1	1
2	0

Tristate Logic

Domain: $\{0, 1, 2\}$

Three operators:

	!
0	2
1	1
2	0

&&	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

Tristate Logic

Domain: $\{0, 1, 2\}$

Three operators:

	!
0	2
1	1
2	0

&&	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

$\|\|$	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

Tristate Logic

Domain: $\{0, 1, 2\}$

Three operators:

	!
0	2
1	1
2	0

&&	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

$\|\|$	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

- Idea: First create POF in Tristate Logic

Tristate Logic

Domain: $\{0, 1, 2\}$

Three operators:

	!
0	2
1	1
2	0

&&	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

$\|\|$	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

- Idea: First create POF in Tristate Logic
- Interpretation: Valid configuration 2, invalid configuration 0

Tristate Logic

Domain: $\{0, 1, 2\}$

Three operators:

	!
0	2
1	1
2	0

&&	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

$\|\|$	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

- Idea: First create POF in Tristate Logic
- Interpretation: Valid configuration 2, invalid configuration 0

$\Rightarrow$ Not possible: Functionally not complete.

Tristate* Extension

Solution: Define new operators

Tristate* Extension

Solution: Define new operators

$\Leftrightarrow$	0	1	2
0	2	0	0
1	0	2	0
2	0	0	2

$\Rightarrow$	0	1	2
0	2	2	2
1	0	2	2
2	0	0	2

Tristate* Extension

Solution: Define new operators

$\Leftrightarrow$	0	1	2
0	2	0	0
1	0	2	0
2	0	0	2

$\Rightarrow$	0	1	2
0	2	2	2
1	0	2	2
2	0	0	2

POF Creation

POF Creation

- Symbols remain as variables

POF Creation

- Symbols remain as variables
- Auxiliary variables for attributes

POF Creation

- Symbols remain as variables
- Auxiliary variables for attributes
- Auxiliary variables for upper and lower bound

POF Creation

- Symbols remain as variables
- Auxiliary variables for attributes
- Auxiliary variables for upper and lower bound
- Fixed correlation between a symbol and its auxiliary variables

POF Creation

- Symbols remain as variables
- Auxiliary variables for attributes
- Auxiliary variables for upper and lower bound
- Fixed correlation between a symbol and its auxiliary variables
- Dependencies are expressions consisting of symbols and affect auxiliary variables

POF Creation

- Symbols remain as variables
- Auxiliary variables for attributes
- Auxiliary variables for upper and lower bound
- Fixed correlation between a symbol and its auxiliary variables
- Dependencies are expressions consisting of symbols and affect auxiliary variables
- $\Rightarrow$ POF creation straightforward as each symbol can be considered separately

Translation into Propositional POF

Translation from Tristate* Logic into Propositional Logic:

Translation into Propositional POF

Translation from Tristate* Logic into Propositional Logic:

- Tristate Variable $A \rightsquigarrow$ 2 Propositional Variables $p_0(A)$, $p_1(A)$

Translation into Propositional POF

Translation from Tristate* Logic into Propositional Logic:

- Tristate Variable $A \rightsquigarrow$ 2 Propositional Variables $p_0(A)$, $p_1(A)$
- Tristate Formula $e \rightsquigarrow$ 2 projections $\pi_0(e), \pi_1(e)$ to Propositional Logic

Translation into Propositional POF

Translation from Tristate* Logic into Propositional Logic:

- Tristate Variable $A \rightsquigarrow 2$ Propositional Variables $p_0(A)$, $p_1(A)$
- Tristate Formula $e \rightsquigarrow 2$ projections $\pi_0(e), \pi_1(e)$ to Propositional Logic

Translation rules:

Translation into Propositional POF

Translation from Tristate* Logic into Propositional Logic:

- Tristate Variable $A \rightsquigarrow$ 2 Propositional Variables $p_0(A)$, $p_1(A)$
- Tristate Formula $e \rightsquigarrow$ 2 projections $\pi_0(e), \pi_1(e)$ to Propositional Logic

Translation rules:

e'	$\pi_0(e')$	$\pi_1(e')$
A	$p_0(A)$	$p_1(A)$
$!\,e$	$\neg\pi_0(e) \wedge \neg\pi_1(e)$	$\pi_1(e)$
$e_0 \,\&\&\, \cdots \,\&\&\, e_n$	$\pi_0(e_0) \wedge \cdots \wedge \pi_0(e_n)$	$\bigwedge_{i\in\{0,\ldots,n\}}(\pi_0(e_i) \vee \pi_1(e_i)) \wedge \bigvee_{i\in\{0,\ldots,n\}} \pi_1(e_i)$
$e_0 \,\|\, \cdots \,\|\, e_n$	$\pi_0(e_0) \vee \cdots \vee \pi_0(e_n)$	$\bigwedge_{i\in\{0,\ldots,n\}}(\neg\pi_0(e_i)) \wedge \bigvee_{i\in\{0,\ldots,n\}} \pi_1(e_i)$
$e_1 \Leftrightarrow e_2$	$(\pi_0(e_1) \leftrightarrow \pi_0(e_2)) \wedge (\pi_1(e_1) \leftrightarrow \pi_1(e_2))$	$\bot$
$e_1 \Rightarrow e_2$	$\pi_0(e_2) \vee \neg\pi_0(e_1) \wedge (\neg\pi_1(e_1) \vee \pi_1(e_2))$	$\bot$

Translation into Propositional POF

Translation from Tristate* Logic into Propositional Logic:

- Tristate Variable $A \rightsquigarrow$ 2 Propositional Variables $p_0(A)$, $p_1(A)$
- Tristate Formula $e \rightsquigarrow$ 2 projections $\pi_0(e), \pi_1(e)$ to Propositional Logic

Translation rules:

e'	$\pi_0(e')$	$\pi_1(e')$
A	$p_0(A)$	$p_1(A)$
$!\,e$	$\neg\pi_0(e) \wedge \neg\pi_1(e)$	$\pi_1(e)$
$e_0 \,\&\&\, \cdots \,\&\&\, e_n$	$\pi_0(e_0) \wedge \cdots \wedge \pi_0(e_n)$	$\bigwedge_{i\in\{0,\ldots,n\}}(\pi_0(e_i) \vee \pi_1(e_i)) \wedge \bigvee_{i\in\{0,\ldots,n\}} \pi_1(e_i)$
$e_0 \,\|\, \cdots \,\|\, e_n$	$\pi_0(e_0) \vee \cdots \vee \pi_0(e_n)$	$\bigwedge_{i\in\{0,\ldots,n\}}(\neg\pi_0(e_i)) \wedge \bigvee_{i\in\{0,\ldots,n\}} \pi_1(e_i)$
$e_1 \Leftrightarrow e_2$	$(\pi_0(e_1) \leftrightarrow \pi_0(e_2)) \wedge (\pi_1(e_1) \leftrightarrow \pi_1(e_2))$	$\bot$
$e_1 \Rightarrow e_2$	$\pi_0(e_2) \vee \neg\pi_0(e_1) \wedge (\neg\pi_1(e_1) \vee \pi_1(e_2))$	$\bot$

$\Rightarrow$ Tristate* POF $\Phi^T \rightsquigarrow \pi_0\left(\Phi^T\right)$

Translation into Propositional POF

Translation from Tristate* Logic into Propositional Logic:

- Tristate Variable $A \rightsquigarrow$ 2 Propositional Variables $p_0(A)$, $p_1(A)$
- Tristate Formula $e \rightsquigarrow$ 2 projections $\pi_0(e), \pi_1(e)$ to Propositional Logic

Translation rules:

e'	$\pi_0(e')$	$\pi_1(e')$
A	$p_0(A)$	$p_1(A)$
$!\,e$	$\neg\pi_0(e) \wedge \neg\pi_1(e)$	$\pi_1(e)$
$e_0 \,\&\&\, \cdots \,\&\&\, e_n$	$\pi_0(e_0) \wedge \cdots \wedge \pi_0(e_n)$	$\bigwedge_{i\in\{0,\ldots,n\}}(\pi_0(e_i) \vee \pi_1(e_i)) \wedge \bigvee_{i\in\{0,\ldots,n\}} \pi_1(e_i)$
$e_0 \,\|\, \cdots \,\|\, e_n$	$\pi_0(e_0) \vee \cdots \vee \pi_0(e_n)$	$\bigwedge_{i\in\{0,\ldots,n\}}(\neg\pi_0(e_i)) \wedge \bigvee_{i\in\{0,\ldots,n\}} \pi_1(e_i)$
$e_1 \Leftrightarrow e_2$	$(\pi_0(e_1) \leftrightarrow \pi_0(e_2)) \wedge (\pi_1(e_1) \leftrightarrow \pi_1(e_2))$	$\bot$
$e_1 \Rightarrow e_2$	$\pi_0(e_2) \vee \neg\pi_0(e_1) \wedge (\neg\pi_1(e_1) \vee \pi_1(e_2))$	$\bot$

$\Rightarrow$ Tristate* POF $\Phi^T \rightsquigarrow \pi_0\left(\Phi^T\right)$, Plaisted-Greenbaum for CNF

$\Rightarrow$ Same methods as in automotive industry

Sizes of Formulae

Table: Sizes of POFs for Linux 4.0

arch	variables TPOF	aux TPOF	total TPOF	variables L-POF	variables CNF	clauses CNF
arm	11976	55760	67736	134270	1299812	2849653
c6x	10548	48174	58722	115799	949031	1708805
ia64	10866	49850	60716	119837	1010856	1834072
m68k	10717	49136	59853	118115	1008800	1836987
mips	11249	52034	63283	125090	1048937	1909971
powerpc	11247	51964	63211	124935	1055822	1917736
s390	10699	49084	59783	117997	998901	1813210
score	10539	48168	58707	115783	949788	1710461
sh	10955	50336	61291	121037	1020515	1854779
sparc	10774	49327	60101	118582	1004762	1823946
x86	11135	51280	62415	123314	1051478	1913811

Analysis Results

Table: Redundant or necessary symbols in Linux 4.0

arch	inadmissible	necessary
arm	1691	75
c6x	4644	42
ia64	3454	74
m68k	3741	32
mips	2773	64
powerpc	2652	94
s390	4149	107
score	7068	36
sh	3297	67
sparc	3201	51
x86	2301	138

Using unmodified picosat: < 0.3s in > 99 % of the cases

Outlook

- Re-configuration
- List possible constraints violations
- Incorporate Linux 4.2 Updates (<=, >=, <, >)
- Design new configuration language?
- SATCOUNT?
- Configuration Lifting
- ...

Questions?


Andreas Kaiser and Wolfgang Küchlin, 'Detecting inadmissible and necessary variables in large propositional formulae', Technical report, University of Siena, (2001).


Wolfgang Küchlin and Carsten Sinz, 'Proving consistency assertions for automotive product data management', *Journal of Automated Reasoning*, **24**(1–2), 145–163, (2000).


Reinhard Tartler, Daniel Lohmann, Julio Sincero, and Wolfgang Schröder-Preikschat, 'Feature Consistency in Compile-Time Configurable System Software', in *Proceedings of the EuroSys 2011 Conference (EuroSys '11)*, eds., Gernoth Heiser and Christoph Kirsch, pp. 47–60, New York, NY, USA, (2011).


Christoph Zengler and Wolfgang Küchlin, 'Encoding the Linux Kernel Configuration in Propositional Logic', in *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010) Workshop on Configuration*, eds., Lothar Hotz and Alois Haselböck, pp. 51–56, (2010).