Analysis of Software Variants

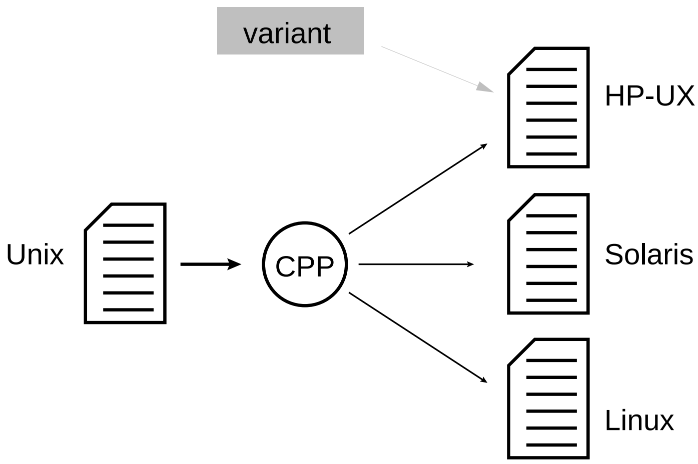
Christian Lindig

Software Technology Group
Technical University of Braunschweig
Germany

Outline

Computer platform diversity causes software diversity – software exists in variants.

Most common strategy: separation at the module level and source code preprocessing for individual platforms.



Aim: A better understanding of all the variants a specific source can generate.

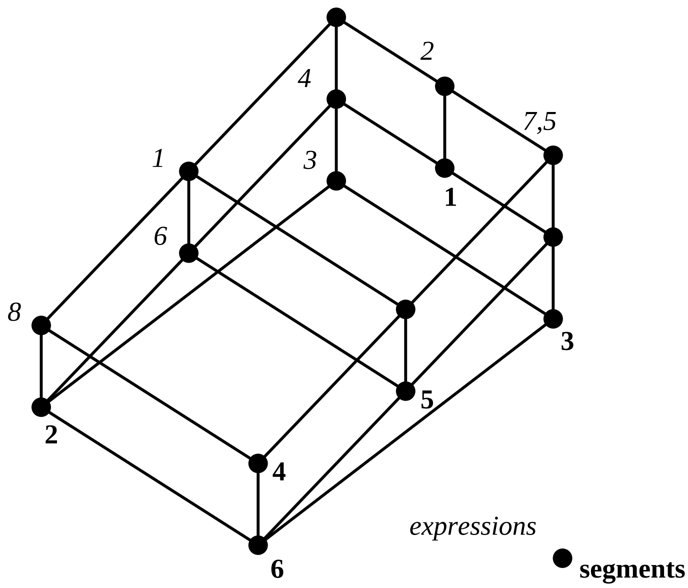
- How many variants exist?
- How are variants related?
- How to generate a specific variant?
- Is there an easier way to create the *same* set of variants?

Example – getopt.c

```
/* This tells Alpha OSF/1 not to define a getopt prototype
   in <stdio.h>. Ditto for AIX 3.2 and <stdlib.h>.  */
#ifndef _NO_PROTO
#define _NO_PROTO
#endif

#ifdef HAVE_CONFIG_H
#if defined (emacs) || defined (CONFIG_BROKETS)

/* We use <config.h> instead of "config.h" so that a
   compilation using -I. -I\$srcdir will use ./config.h
   rather than \$srcdir/config.h (which it would do because
   it found this file in \$srcdir). */

#include <config.h>
#else
#include "config.h"
#endif
#endif

#ifndef __STDC__
/* This is a separate conditional since some stdc systems
   reject ‘defined (const)’.  */
#ifndef const
#define const
#endif
#endif
```

Result

Result of an analysis: the *variant lattice* for a source file.



It shows:

- all variants and their relations,
- how to create them,
- and redundant expressions.

Analysis technique: formal concept analysis.

Keywords

Unix-System-Calls	access	change	check	create	directory	file	get	group	input	mode	new	open	output	owner	permission	process	read	remove	status	write
access	●		●			●														
chdir		●			●															
chmod		●				●				●					●					
chown		●				●		●						●						
creat				●		●					●									
fork				●							●					●				
fstat						●	●												●	
mkdir				●	●						●									
open				●		●						●					●			●
read						●			●								●			
rmdir					●	●												●		
write						●							●							●

A (formal) *context* is a triple $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ where $\mathcal{O}$ and $\mathcal{A}$ are sets of objects and attributes respectively and $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$ is a relation among them.

A set O (A) of objects (attributes) shares a set of common attributes (objects):

$$O' \stackrel{\text{def}}{=} \{a \in \mathcal{A} \mid \forall o \in O : (o, a) \in \mathcal{R}\}$$

$$A' \stackrel{\text{def}}{=} \{o \in \mathcal{O} \mid \forall a \in A : (o, a) \in \mathcal{R}\}$$

[Krone, Snelting ICSE'94] defined O' as $\omega(O)$ and A' as $\alpha(A)$

Context Table

Keywords

Unix-System-Calls	access	change	check	create	directory	file	get	group	input	mode	new	open	output	owner	permission	process	read	remove	status	write
access	●		●			●														
chdir		●			●															
chmod		●				●				●					●					
chown		●				●		●						●						
creat				●		●					●									
fork				●							●					●				
fstat						●	●												●	
mkdir				●	●						●									
open				●		●						●					●			●
read						●			●								●			
rmdir					●	●												●		
write						●							●							●

Keywords

Unix-System-Calls	access	change	check	create	directory	file	get	group	input	mode	new	open	output	owner	permission	process	read	remove	status	write
access	●		●			●														
chdir		●			●															
chmod		●				●				●					●					
chown		●				●		●						●						
creat				●		●					●									
fork				●							●					●				
fstat						●	●												●	
mkdir				●	●						●									
open				●		●						●					●			●
read						●			●								●			
rmdir					●	●												●		
write						●							●							●

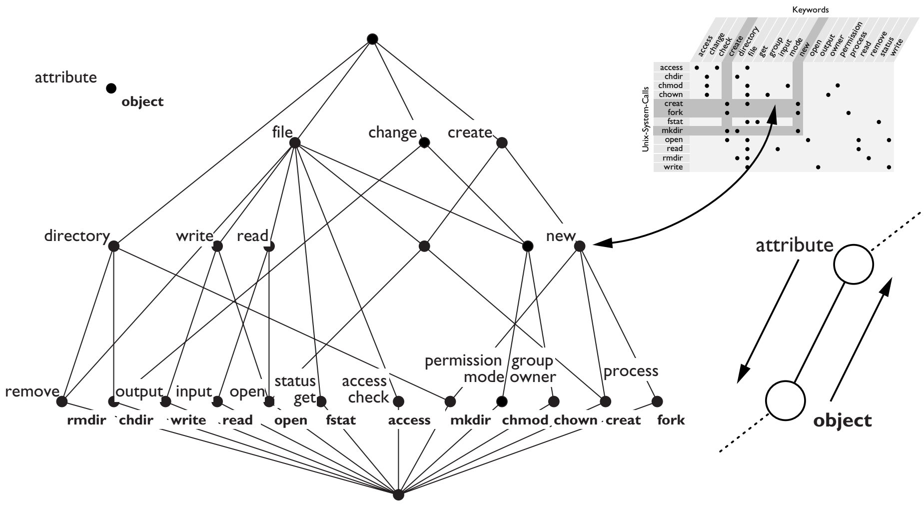
A *concept* $c = (O, A)$ of a context $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ is a pair where $O' = A$ and $A' = O$ and also $O \subseteq \mathcal{O}, A \subseteq \mathcal{A}$.

Concepts are (partially) ordered:

$$(O_1, A_1) \leq (O_2, A_2) \Leftrightarrow O_1 \subseteq O_2 \quad (\Leftrightarrow A_1 \supseteq A_2)$$

The set of all concepts of $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ is denoted by $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$.

Concept Lattice



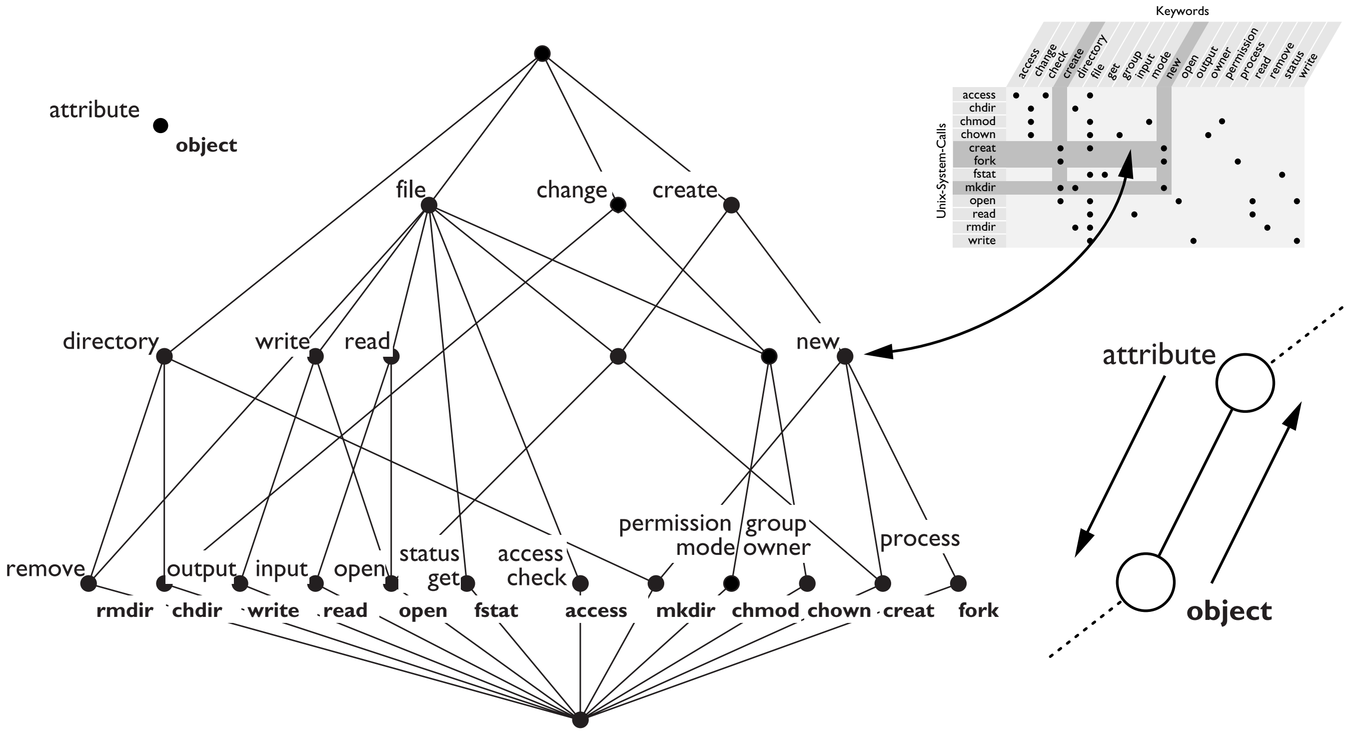
Basic theorem of context analysis by Wille: Let $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ be a context. Then $\mathcal{B}(\mathcal{O}, \mathcal{A}, \mathcal{R})$ is a complete lattice, the *concept lattice* of $(\mathcal{O}, \mathcal{A}, \mathcal{R})$.

$$
\begin{aligned}
(O_1, A_1) \wedge (O_2, A_2) &= (O_1 \cap O_2, (O_1 \cap O_2)') \\
(O_1, A_1) \vee (O_2, A_2) &= ((A_1 \cap A_2)', A_1 \cap A_2)
\end{aligned}
$$

The maximal number of concepts is 2^n where $n = \max(|O|, |A|)$.

Ganter's algorithm computes all concepts of $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ with time complexity $O(|B(\mathcal{O}, \mathcal{A}, \mathcal{R})|)$.

Concept Lattice



The Idea

```
#if defined(E1) && defined(E6) && defined(E8)
  S3
#if defined (E3) && defined(E5) && defined(E7)
  S1
#endif
#endif
  S6
#if defined(E2) && defined(E5) && defined(E7)
  S2
#endif
#ifdef E3
#if defined(E4) && defined(E6)
  S4
#ifdef E8
  S5
#endif
#endif
```

	Expression							
Segment	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
S_1	•		•		•	•	•	•
S_2		•			•		•	
S_3	•					•		•
S_4			•	•		•		
S_5			•					•
S_6								

Encode the meaning of CPP-expressions into a binary relation (i.e. formal context) and analyze it using concept analysis [cf. Krone, Snelting, ICSE ’94].

A Model of CPP

- A *configuration function* f evaluates expressions $e \in \mathcal{E}$: $f : \mathcal{E} \to \{0, 1\}$.
- A *variant* is a set of segments, selected by a configuration function f from a *configuration table* $(\mathcal{S}, \mathcal{E}, \mathcal{T})$: $f(\mathcal{S}, \mathcal{E}, \mathcal{T}) = \{s \in \mathcal{S} \mid f(e) = 1 \text{ for all } (s, e) \in \mathcal{T}\}$.
- Configuration functions f_1 and f_2 are called *equivalent* (with respect to $(\mathcal{S}, \mathcal{E}, \mathcal{T})$), if they have the same effect:

$$f_1 \doteq f_2 \iff f_1(\mathcal{S}, \mathcal{E}, \mathcal{T}) = f_2(\mathcal{S}, \mathcal{E}, \mathcal{T})$$

  $[f]$ denotes configuration functions equivalent to f.

	Expression							
f_1			1	1		1	1	1
f_2			1	1	1	1	1	1
Segment	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
S_1	•		•		•	•	•	•
S_2		•			•		•	
S_3	•					•		•
S_4			•	•		•		
S_5			•					•
S_6								

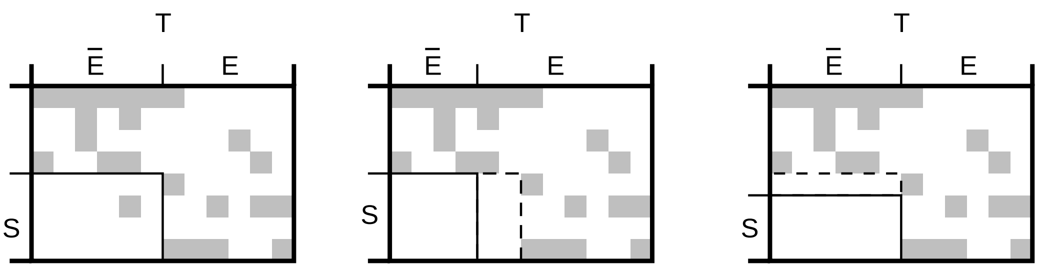
	T							
	$\overline{E}$			E				
f	0	0	0	1	1	1	1	1

Segment	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
S_1	•		•		•	•	•	•
S_2		•			•		•	
S_3	•					•		•
S_4				•		•		
S_5		$(S,\overline{E})$			•			•
S_6							•	

We find: $[f]$ is an equivalence class where $f(\mathcal{S},\mathcal{E},\mathcal{T}) = S$ iff $(S,\overline{E}) \in B(\mathcal{S},\mathcal{E},\overline{\mathcal{T}})$.

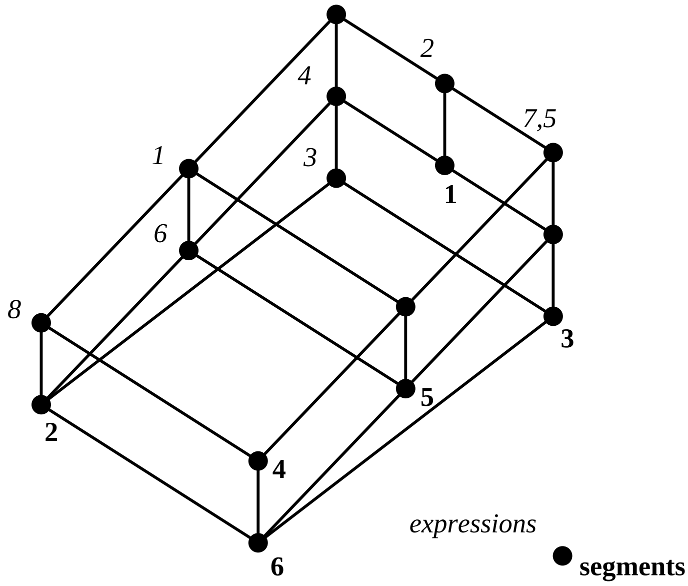
This means: every concept of the inverted configuration table describes an equivalence class.

Proof sketch: there is no variant not described by a concept in $(\mathcal{S},\mathcal{E},\overline{\mathcal{T}})$.



Variant Lattice

The example's variant lattice (i.e. the concept lattice of $(\mathcal{S}, \mathcal{E}, \overline{\mathcal{T}})$).

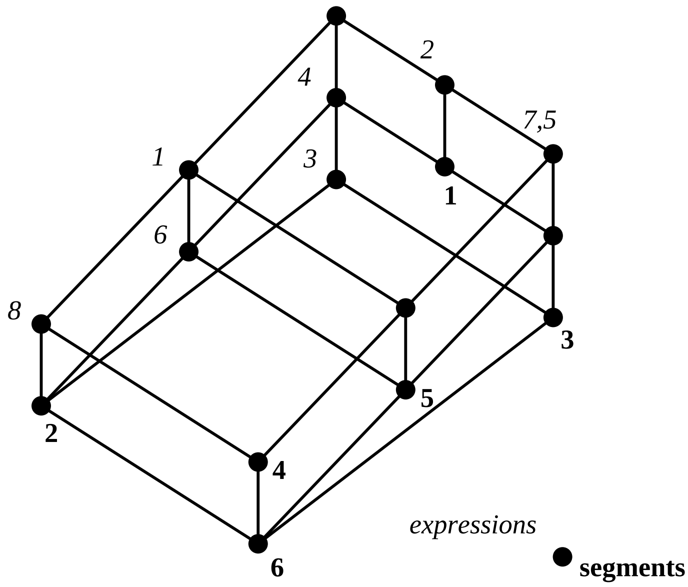


Each concept describes a variant and the canonical way to create it.

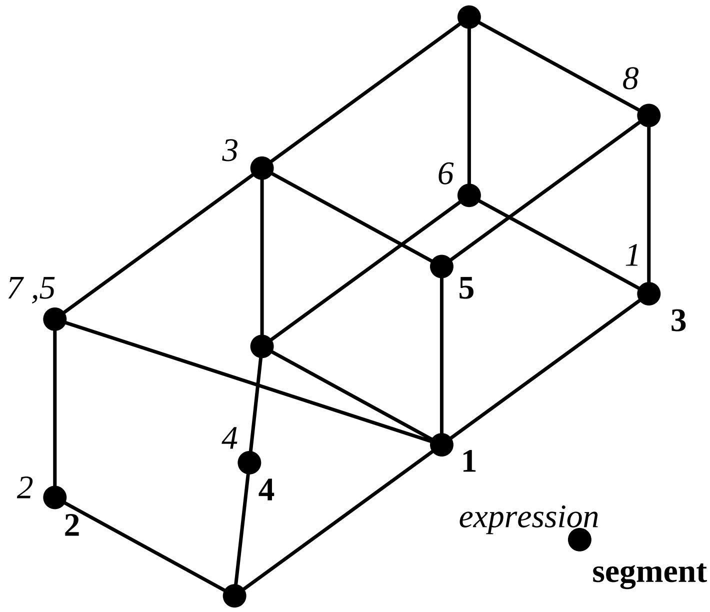
	T							
f	0	0	1	1	0	1	0	1

Segment	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
S_1	•		•		•	•	•	•
S_2		•			•		•	
S_3	•					•		•
$\mathbf{S_4}$			•	•		•		
$\mathbf{S_5}$			•					•
$\mathbf{S_6}$								

The variant lattice for the example:



And its configuration lattice [cf. Krone, Snelting, ICSE 94].



Redundancies

Are there expressions in my source file that can be safely eliminated?

T

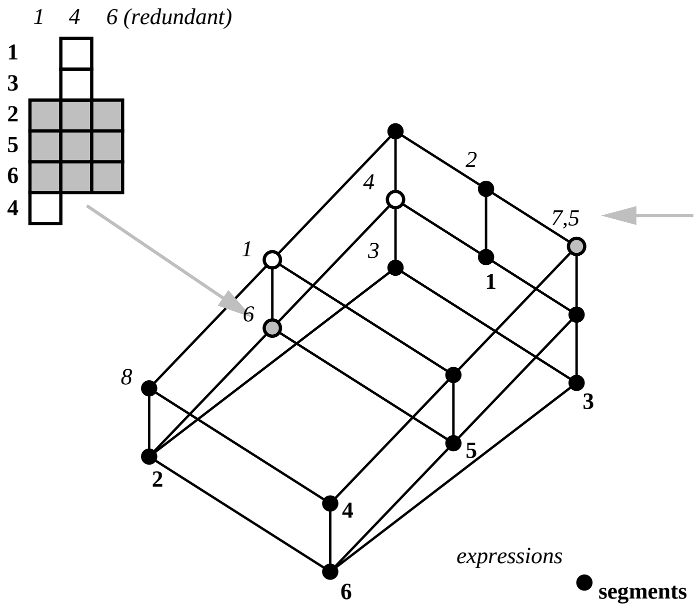
Segment	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
S_1	•		•		•	•	•	•
S_2		•			•		•	
S_3	•					•		•
S_4			•	•		•		
S_5			•					•
S_6								

An expression is called *redundant* if it can be removed from a configuration table without effect on *any* configuration function.

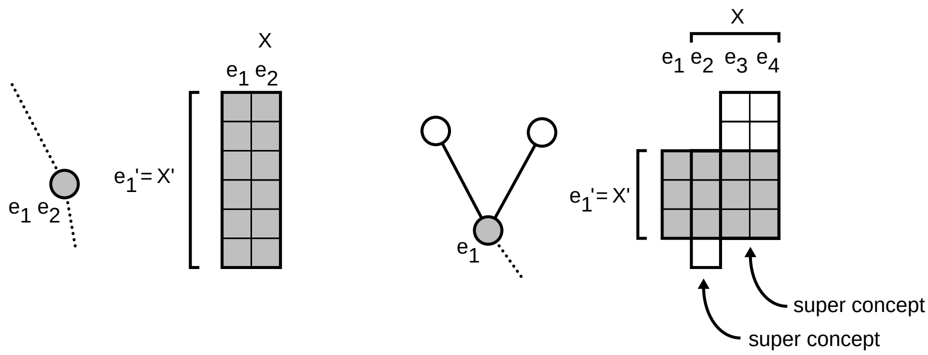
Formally, e is redundant iff the following holds for any f:

$f(\mathcal{S}, \mathcal{E}, \mathcal{T}) = f(\mathcal{S}, \mathcal{E}^-, \mathcal{T}^-)$ where $\mathcal{E}^- = \mathcal{E} \setminus \{e\}$ and $\mathcal{T}^- = \mathcal{T} \cap (\mathcal{S} \times \mathcal{E}^-)$

Redundant expressions show up in the concept lattice of the inverted configuration table as ∧-reducible concepts:



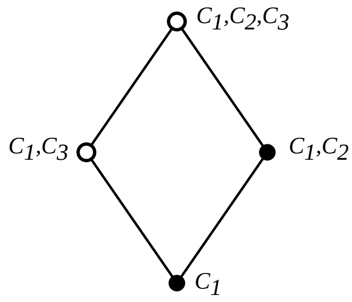
Formally: $e \in \mathcal{E}$ is redundant in $(\mathcal{S}, \mathcal{E}, \mathcal{T})$ if $X \subseteq \mathcal{E}, e \notin X$ exists in $(\mathcal{S}, \mathcal{E}, \overline{\mathcal{T}})$ such that $\{e\}' = X'$.



Problems

```
#if defined(a) || defined (b)
C1
#endif
#if (defined(a) || defined(b)) && defined (c)
C2
#endif
#if !defined(a) && !defined(b)
C3
#endif
```

	Expression			
Segment	$a \vee b$	c	$\neg a$	$\neg b$
C_1	•	•		
C_2	•			
C_3			•	•



Consider f with $f\{a \vee b, \neg a, \neg b\} \rightarrow 1$ then theory says: $S = \{C_2, C_2\}$ is a variant.

Problem: Theory does not take interdependencies into account (yet).

Future work: use background implications to express dependencies.

Future Work

- Extend theory with background implications to overcome limitations.
- Implementation
  - Concept analysis tool written in C available (send email!)
  - Rewrite of a concept analysis library in progress
  - Implementation will be based on new library
- Evaluation

Conclusions

- Concept analysis is a promising theory for software re-engineering
- Variants generated from a source have a rich structure which can be effectively analyzed using concept analysis
- Variant description may contain redundant expressions which can be detected and removed
- This is work in progress – still much room for improvement and research.