

# Formal Concept Analysis And Delayed Greedy algorithm for Min-Test-Suite

- Reference paper
- G. Snelting,  
``Concept Analysis -- A New Framework for Program Understanding,"  
*PASTE*, 1998.
- S. Tallam and N. Gupta,  
``A Concept Analysis Inspired Greedy Algorithm for Test Suite Minimization,"  
*PASTE*, 2005

# Introduction of Concept Analysis

- **Concept**
  - Intent :Attributes
  - Extent: Objects
- **Concept analysis**
  - transforms any relation between ‘objects’ and ‘attributes’ into a complete lattice.

# Introduction of Concept Analysis

- Formal context  $\mathcal{C} = (\mathcal{O}, \mathcal{A}, T)$
- Common attributes of objects  $\sigma(O) = \{a \in \mathcal{A} \mid \forall o \in O : (o, a) \in T\}$
- Common objects of attributes  $\tau(A) = \{o \in \mathcal{O} \mid \forall a \in A : (o, a) \in T\}$
- Concept is pair(O,A)  
 $A = \sigma(O)$  and  $O = \tau(A)$
- Partial order  $(O_1, A_1) \leq (O_2, A_2) \iff O_1 \subseteq O_2 \iff A_1 \supseteq A_2$

# Construction of concept lattice:example

```

SUBROUTINE R1(...)
COMMON /C1/ V1,V2
...
END

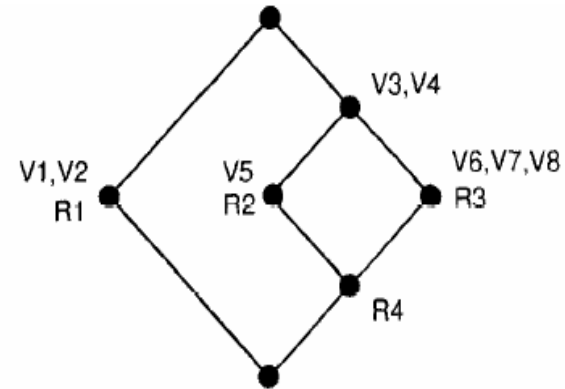
SUBROUTINE R2(...)
COMMON /C2/ V3,V4
COMMON /C3/ V5
...
END

SUBROUTINE R3(...)
COMMON /C2/ V3,V4
COMMON /C4/ V6,V7,V8
...
END

SUBROUTINE R4(...)
COMMON /C2/ V3,V4
COMMON /C3/ V5
COMMON /C4/ V6,V7,V8
...
END

```

	V1	V2	V3	V4	V5	V6	V7	V8
R1	x	x						
R2			x	x	x			
R3			x	x		x	x	x
R4			x	x	x	x	x	x



```

V8 → V7 V6 V4 V3
V7 → V8 V6 V4 V3
V6 → V8 V7 V4 V3
V5 → V4 V3
V4 V3 V2 V1 → V8 V7 V6 V5
V4 → V3
V3 → V4
V2 → V1
V1 → V2

```

Figure 1: A formal context, its concept lattice, and its minimal implication base; extracted from a source text.

# Construction of concept lattice:example

	V1	V2	V3	V4	V5	V6	V7	V8
R1	x	x						
R2			x	x	x			
R3			x	x		x	x	x
R4			x	x	x	x	x	x

Top: ( $\{R1,R2,R3,R4\},\{\}$ )

C1: ( $\{R1\},\{V1,V2\}$ )

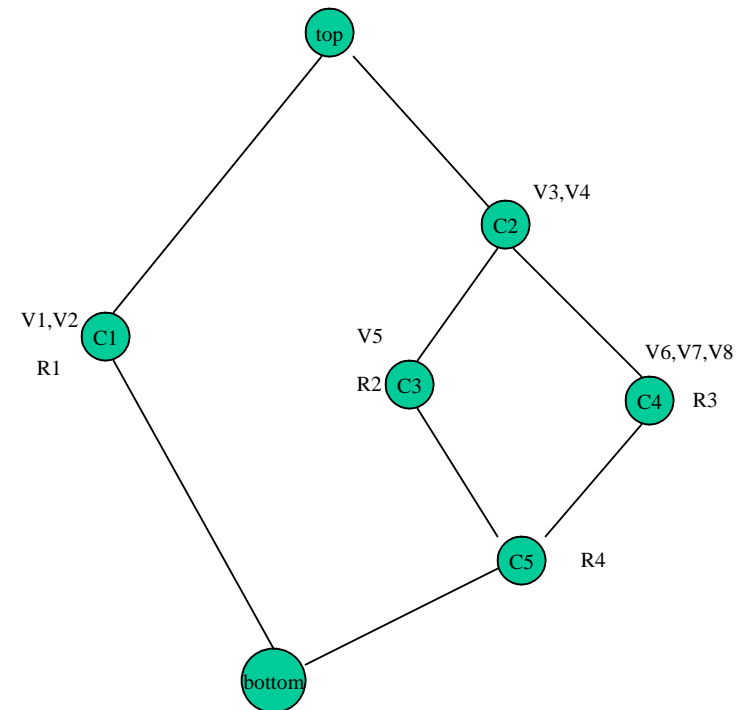
C2: ( $\{R2,R3,R4\},\{V3,V4\}$ )

C3: ( $\{R2,R4\},\{V3,V4,V5\}$ )

C4: ( $\{R3,R4\},\{V3,V4,V6,V7,V8\}$ )

C5: ( $\{R4\},\{V3,V4,V5,V6,V7,V8\}$ )

Bottom: ( $\{\},\{V1,V2,V3,V4,V5,V6,V7,V8\}$ )



- A lattice element is labeled with attribute a,if it is the largest concept having attribute a

- A lattice element is labeled with object o, if it is the smallest concept having object o

## Construction of concept lattice:example

- Interpretation

- The lattice show that all the subroutines(R2,R3,R4) use V3 are below u(V3)
- All variable above r(R4) –V3,V4,V5,V6,V7 are used by R4
- So the concept labeled by R4 ,and V5/R2 is

$$c_1 = \gamma(R4) = (\{R4\}, \{V3,V4,V5,V6,V7,V8\})$$

$$c_2 = \mu(V5) = \gamma(R2) = (\{R2,R4\}, \{V3,V4,V5\})$$

Hence  $c_1 < c_2$ .

## Construction of concept lattice

- Connections between relation table and concept lattice
  - They can be constructed from each other

$$(o, a) \in T \iff \gamma(o) \leq \mu(a)$$

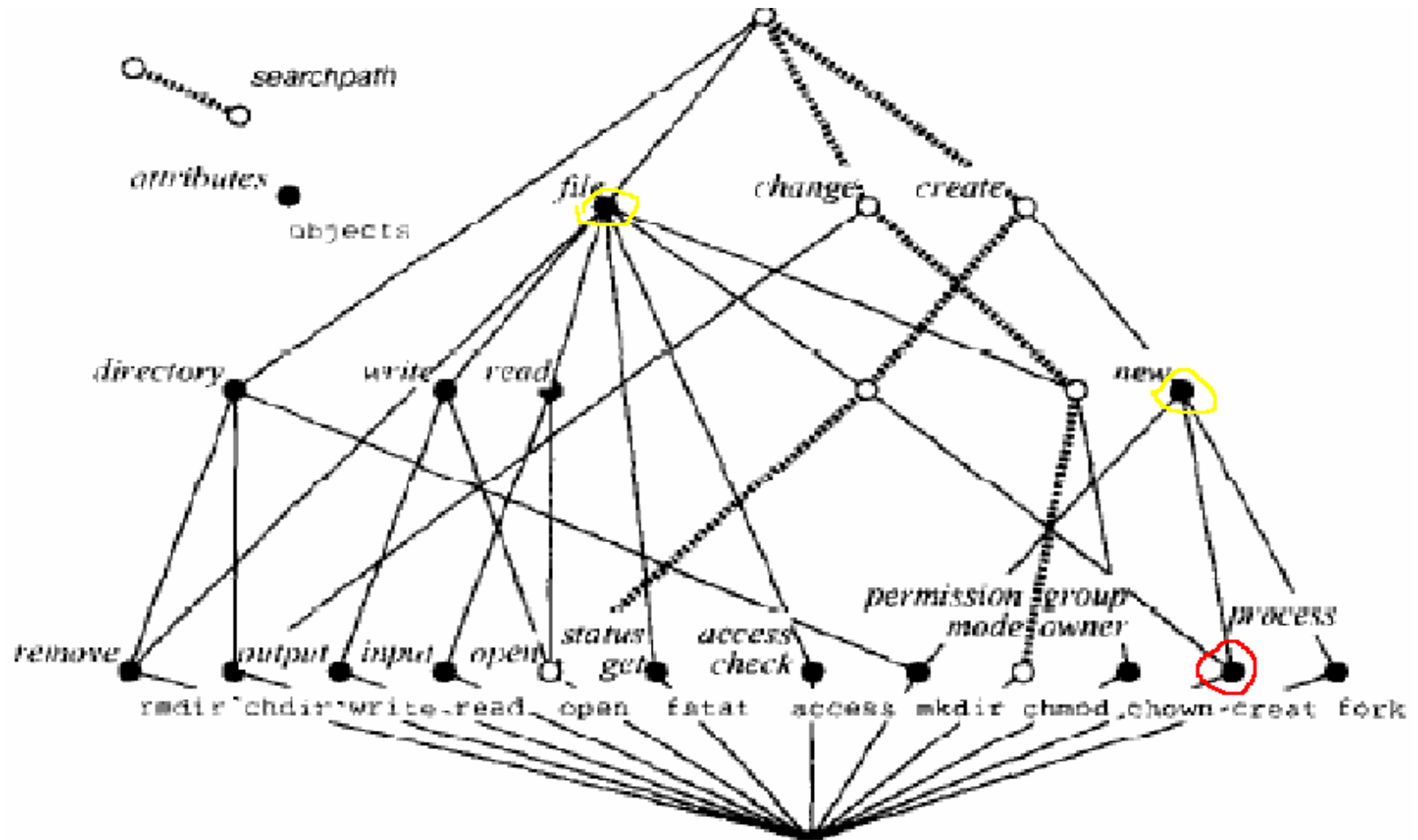
Hence the attributes of object  $o$  are just those which show up above  $o$  in the lattice, and the objects for attribute  $a$  are those which show up below  $a$ .

## Improve software components retrieval

- Assume the components are represented by keywords

	<i>access</i>	<i>change</i>	<i>check</i>	<i>create</i>	<i>directory</i>	<i>file</i>	<i>get</i>	<i>group</i>	<i>input</i>	<i>mode</i>	<i>new</i>	<i>open</i>	<i>output</i>	<i>owner</i>	<i>permission</i>	<i>process</i>	<i>read</i>	<i>remove</i>	<i>status</i>	<i>write</i>	
<i>access</i>	X	X				X															
<i>chdir</i>		X			X																
<i>chmod</i>		X				X				X					X						
<i>chown</i>		X				X		X						X							
<i>creat</i>				X		X					X										
<i>fork</i>				X							X					X					
<i>fstat</i>						X	X													X	
<i>mkdir</i>				X	X						X										
<i>open</i>				X		X						X					X				X
<i>read</i>						X			X								X				
<i>rmdir</i>					X	X												X			
<i>write</i>						X							X								X

# Improve software components retrieval



The search key  $Q = \{ \text{file, new} \}$  identify component create.

## Improve software components retrieval

- Lattice can support searching and narrow search space

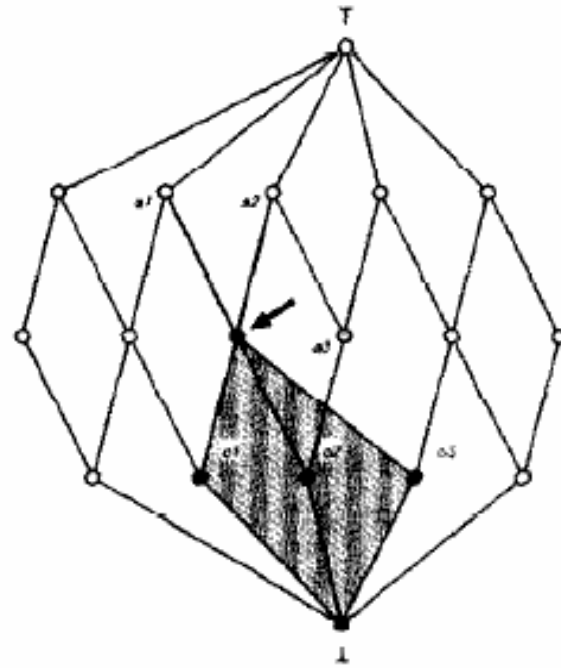


Figure 3: Narrowing the search space

# Exploring configuration tables

...I...

```
#ifdef DOS
```

...II...

```
#endif
```

```
#ifdef OS2
```

...III...

```
#endif
```

```
#if defined(DOS)
```

```
    && defined(X_win)
```

...IV...

```
#endif
```

```
#ifdef X_win
```

...V...

```
#endif
```

...VI...

	DOS	OS2	X_win
I			
II	x		
III		x	
IV	x		x
V			x
VI			

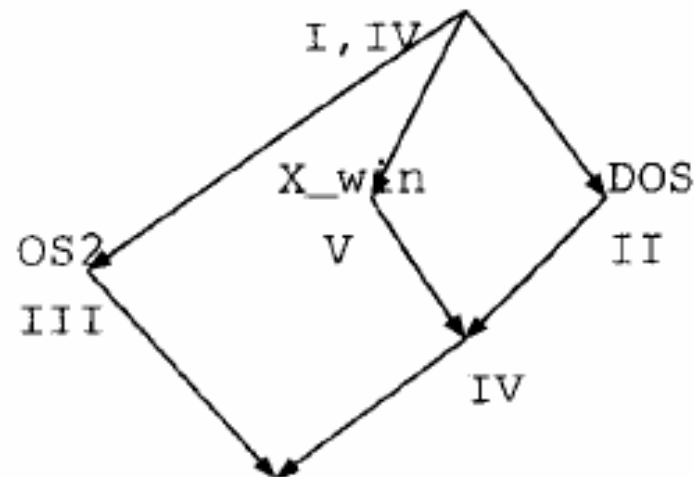


Figure 4: A simple CPP file and its configuration lattice

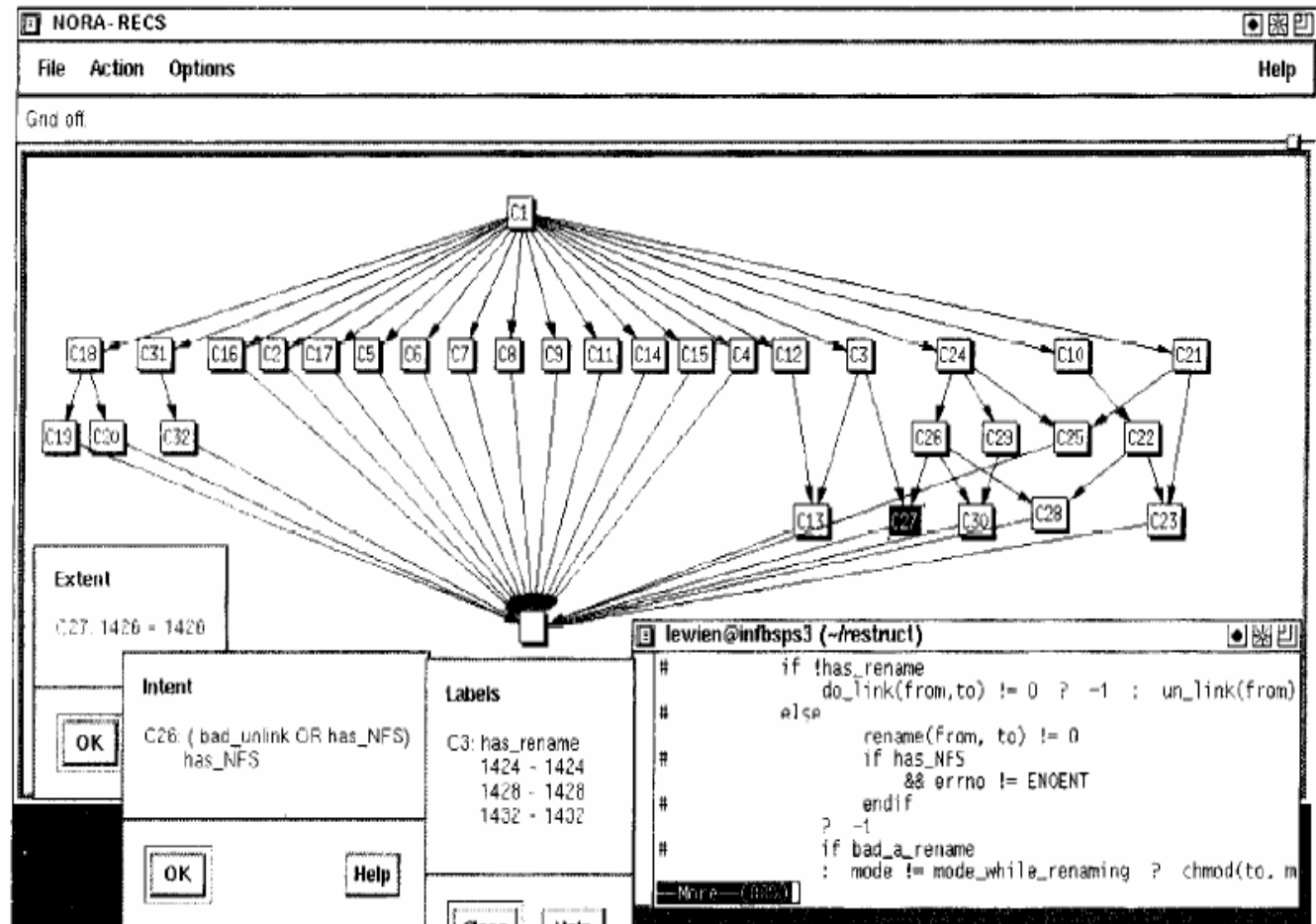
# Interference analysis

Source file of rcsedit, uses 1656 lines and 21 variables for configuration management

C1 is represent code piece not governed by anything.

The left side of lattice is quit flat,

However, the right side has some interference.



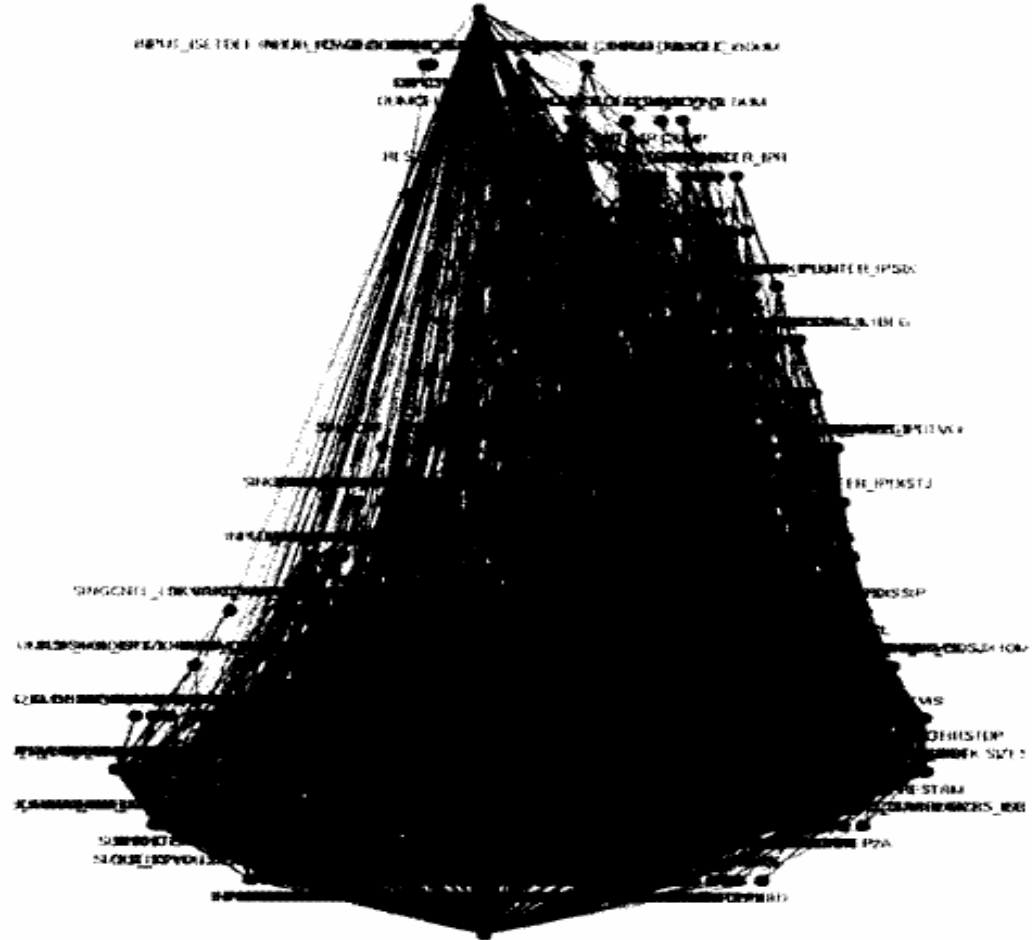
# Assessing modular structure

A module consists of a set of procedures in  $P$  use only variables in  $V$  and all variables in  $V$  are only used by  $P$ .

Good modular structure has low interference.

The right software system has 492 global variables and 42 common block.

It has very high interferences



# A Concept Analysis Inspired Greedy Algorithm for Test Suite Minimization

## Motivation and Goal:

- Software test is expensive, so we need to minimize the test suite satisfying the requirement set.
- Selecting a minimal subset of  $T$  that covers all the requirements is a NP complete problem. Min-test-suite is equivalent to *min-set-cover* problem, which is a NPC problem.
- So a heuristic algorithm which could lead to a sub optimal solution is needed

## A simple greedy heuristic algorithm

- Strategy: Always choose the test case covers max requirements.

**Table 1: An Example showing the requirements exercised by test cases in a test suite.**

	$r_1$	$r_2$	$r_3$	$r_4$	$r_5$	$r_6$
$t_1$	X	X	X			
$t_2$	X			X		
$t_3$		X			X	
$t_4$			X			X
$t_5$					X	

The result is (t1,t2,t3,t4)

The optimal is(t2,t3,t4)

Reason: make decision to choose t1 too early

# HGS

- Consider the subset of Test suites  $T$ , such that any one of the test case  $t_j$  in  $T_i$  can be used to test  $r_i$ .
- First includes all the tests cases that occur in  $T_i$ 's of cardinality one and mark all  $T_i$ 's containing any of these test cases.
- Then consider  $T_i$ 's cardinality two, test case that occurs in the maximum number of  $T_i$ 's cardinality two is choose.
- repeat this process until reach cardinality max and all unmarked  $T_i$ 's containing these test cases are marked.
- If a tie, choose the test case that occurs the max number of unmarked  $T_i$  of cardinality  $m+1$  is chosen.

	$r_1$	$r_2$	$r_3$	$r_4$	$r_5$
$t_1$	X	X			
$t_2$	X		X		X
$t_3$		X	X	X	
$t_4$			X	X	
$t_5$				X	
$t_6$					X
$t_7$					X

$$T_1 = \{t_1, t_2\}$$

$$T_2 = \{t_1, t_3\}$$

$$T_3 = \{t_2, t_3, t_4\}$$

$$T_4 = \{t_3, t_4, t_5\}$$

$$T_5 = \{t_2, t_6, t_7\}$$

Firstly, consider  $T_1$  and  $T_2$  (each with cardinality two) and select test case  $t_1$ .

Next consider  $T_3, T_4, T_5$  (each with cardinality three), choose  $t_2$ .

Only  $r_4$  is not covered, and  $T_4$  is not marked, so we randomly choose one of  $t_3, t_4, t_5$ , suppose choose  $t_3$ , the final result is  $\{t_1, t_2, t_3\}$

However, the optimal is  $\{t_2, t_3\}$

**Reason: the decision to choose  $t_1$  is too early.**

# Using concept table to do reduction

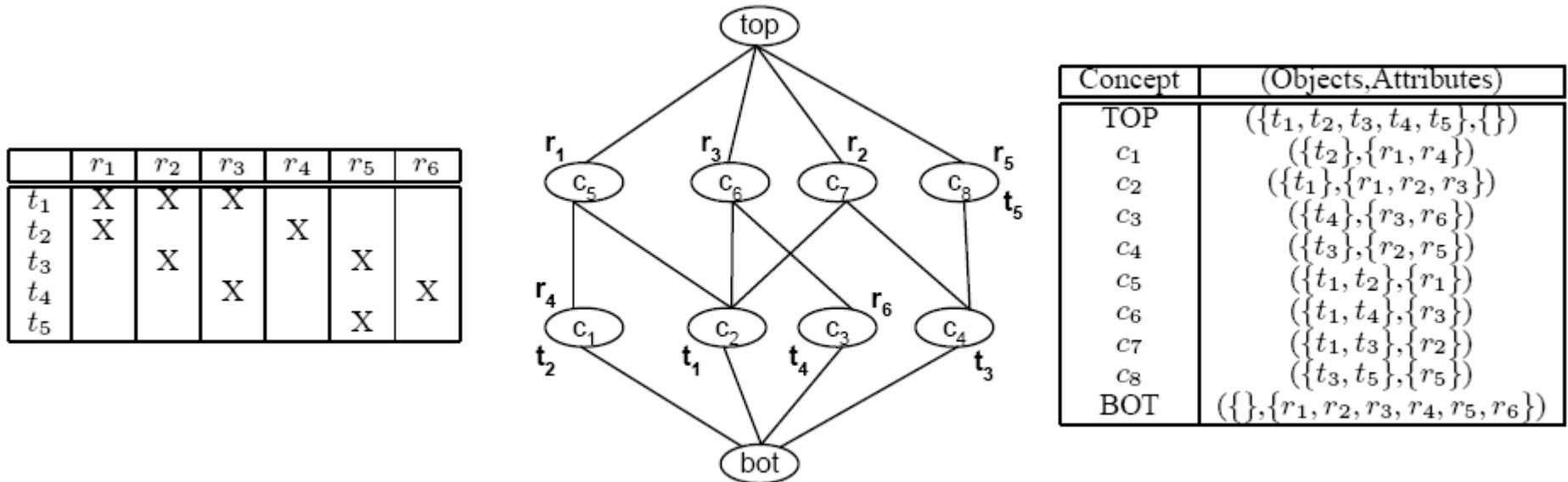


Figure 1: Context table, concept lattice and the table of concepts.

# Using concept table to do reduction

- **Object reduction.**
  - If a concept labeled with object  $o_1$  is lower than concept labeled with object  $o_2$  and the two concept are ordered, all the attributes covered by  $o_2$  will also be covered by  $o_1$ , so we can safely remove  $o_2$ .
- **Attribute Reduction.**
  - If a concept labeled with attribute  $a_1$  is lower than concept labeled with attribute  $a_2$  and the two concept are ordered, in the case, the coverage of  $a_1$  will imply coverage of  $a_2$ , so requirement of  $a_2$  is redundant, so we can safely remove the column corresponding to  $a_2$  in the context table.
- **Ownership reduction.**
  - Define the strongest concepts as the elements in the lattice which is next to bottom. If any strongest concept  $s$  is labeled with attribute  $a$ , then it implies we must choose the objects, since only objects in that concept covers the attribute.

# Reduction

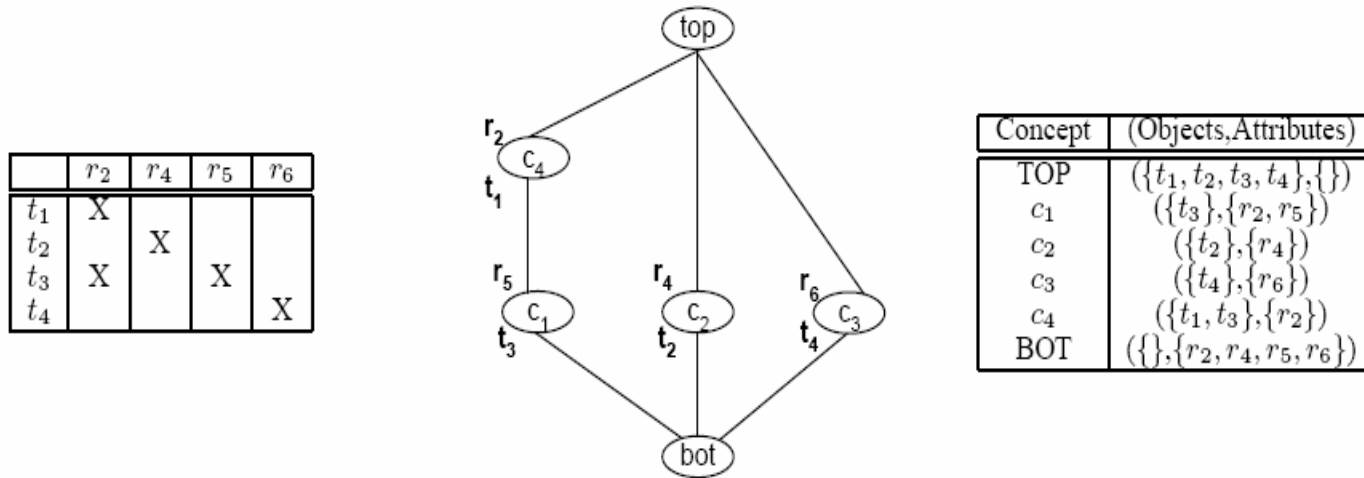


Figure 2: Reduced context table, concept lattice and table of concepts after applying object reduction  $t_3 \Rightarrow t_5$  and attribute reductions  $r_6 \Rightarrow r_3$  and  $r_4 \Rightarrow r_1$  to context table in Figure 1.

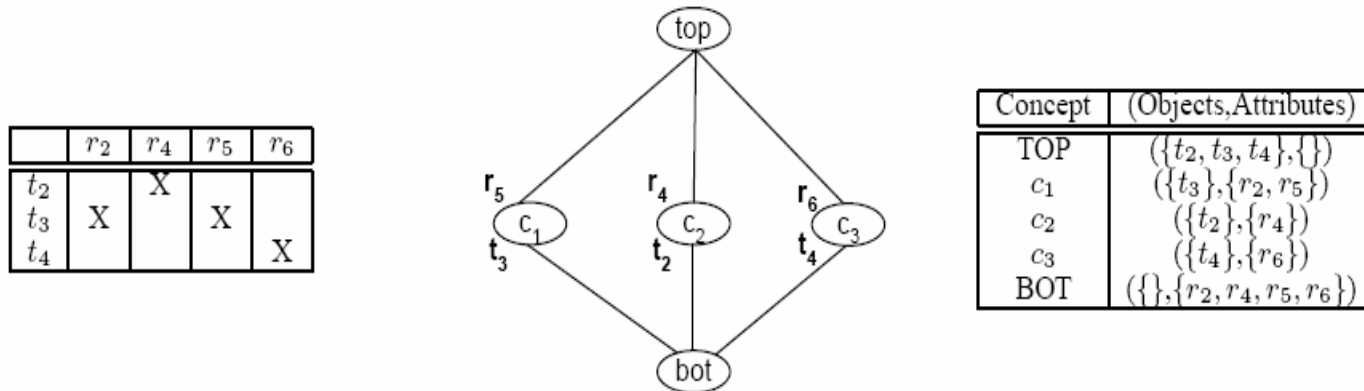


Figure 3: Reduced context table, concept lattice and table of concepts after applying object reduction  $t_3 \Rightarrow t_1$  to context Table in Figure 2.

# DELAYED GREEDY ALGORITHM

**Input:** Context\_table for given test suite T.

**Output:** Set of test cases in minimized suite  $T_{min}$ .

**procedure** Delayed-Greedy(Context\_table)

$T_{min}$ =empty;

**while** (Context\_table  $\neq$  empty) **do**

        fInter=false; detectInter=0;

**while** not(fInter) **and** (Context\_table  $\neq$  empty) **do**

            fInter=true;

**Step 1: For** each *object implication*  $o_i \Rightarrow o_j$  **do**

        Remove row for test case  $o_j$  from Context\_table;

        fInter=false;

**endfor**

**Step 2: For** each *attribute implication*  $r_i \Rightarrow r_j$  **do**

        Remove column for requirement  $r_j$  from Context\_table;

        fInter=false;

**endfor**

**Step 3: For** each attribute  $r_k$  resulting in an *owner reduction* **do**

        Remove row for test case  $t$  that covers requirement  $r_k$ ;

        Remove columns for attributes covered by test case  $t$ ;

$T_{min} = T_{min} \cup \{t\}$ ;

        fInter=false;

**endfor**

**endwhile**

# DELAYED GREEDY ALGORITHM

```

endwhile
Step 4:
if (Context_table  $\neq$  empty) then
    Let  $t$  be test case picked using greedy coverage heuristic.
    Remove row for test case  $t$  from the Context_table;
    Remove columns for attributes covered by test case  $t$ ;
     $T_{min} = T_{min} \cup \{t\}$ ; detectInter=1;
endif
endwhile
if (detectInter=0) then
    report minimized test suite  $T_{min}$  is of optimal size.
else
    report interference encountered.
endif
return( $T_{min}$ )
endprocedure
```

Choose the objects that covers the most number of attributes.

If more objects covers the same number of attributes, break the tie as

Select the object covering an attributes that is least covered by all other objects. To be as good as classic heuristic algorithm

Figure 4: Delayed-Greedy algorithm

## Comparison with other heuristic algorithms

Algorithm	Reduction strategy before heuristic selection
Simple greedy heuristic	No reduction
HGS	No Reduction
SMSP	Only consider the strongest concept, reduce others
Delayed greedy algorithm	Object reduction Attribute reduction Ownership reduction

# Experiment

**Table 3: Experiment Subjects**

Prog.	loc.	<i>Avg. size of un-minimized suite</i>		<i>Total No. of requirements</i>	
		Branch Cov.	Def-use Cov.	Branches	Def-use pairs
space	6218	533	539	1356	5179
tcas	138	20	21	41	51
print tokens	402	64	66	127	275
print tokens2	483	77	79	154	235
schedule	299	46	46	84	148
replace	516	83	108	155	759
totinfo	346	53	53	83	287

# Experiment

Table 4: Frequency of ( size of Tmin by Algo. - size of Tmin by DelGreedy ) for Branch coverage and Def-Use coverage test-suites

Prog.	Algo.	frequency of ( size of Tmin by Algo. - size of Tmin by DelGreedy )											frequency of ( size of Tmin by Algo. - size of Tmin by DelGreedy )										
		Branch Coverage Suites											Def-Use Coverage Suites										
		0	1	2	3	4	5	6	7	8	9	>9	0	1	2	3	4	5	6	7	8	9	>9
space	Greedy	0	4	10	20	22	22	12	8	2	-	-	0	1	3	5	19	17	24	16	8	6	1
	HGS	4	19	22	18	18	10	5	3	1	-	-	7	20	24	16	14	9	4	2	4	-	-
	SMSP	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	100
tcas	Greedy	57	41	2	-	-	-	-	-	-	-	-	68	30	2	-	-	-	-	-	-	-	-
	HGS	58	36	6	-	-	-	-	-	-	-	-	98	2	0	-	-	-	-	-	-	-	-
	SMSP	0	0	0	0	5	6	11	10	13	18	37	0	1	8	26	28	15	14	7	1	-	-
print tokens	Greedy	82	17	1	-	-	-	-	-	-	-	-	78	19	3	-	-	-	-	-	-	-	-
	HGS	43	36	15	4	2	-	-	-	-	-	-	70	22	6	2	-	-	-	-	-	-	-
	SMSP	0	0	0	0	0	0	2	3	2	2	91	0	0	0	0	0	0	1	2	5	7	85
print tokens2	Greedy	86	14	-	-	-	-	-	-	-	-	-	62	34	3	1	-	-	-	-	-	-	-
	HGS	49	44	17	0	-	-	-	-	-	-	-	48	34	15	3	-	-	-	-	-	-	-
	SMSP	0	0	0	2	1	0	0	1	1	1	94	0	0	0	0	0	1	1	2	0	0	96
schedule	Greedy	100	0	-	-	-	-	-	-	-	-	-	100	-	-	-	-	-	-	-	-	-	-
	HGS	37	48	14	1	-	-	-	-	-	-	-	62	32	6	-	-	-	-	-	-	-	-
	SMSP	99	0	1	-	-	-	-	-	-	-	-	66	19	12	2	1	-	-	-	-	-	-
replace	Greedy	49	45	4	2	-	-	-	-	-	-	-	22	36	38	10	3	1	-	-	-	-	-
	HGS	33	37	27	3	-	-	-	-	-	-	-	29	41	24	5	1	-	-	-	-	-	-
	SMSP	0	0	0	0	0	0	0	0	0	1	99	0	0	0	0	0	0	0	0	0	0	100
totinfo	Greedy	84	16	-	-	-	-	-	-	-	-	-	98	2	-	-	-	-	-	-	-	-	-
	HGS	27	47	20	5	1	-	-	-	-	-	-	66	29	4	1	-	-	-	-	-	-	-
	SMSP	1	2	8	12	20	12	17	12	9	4	3	0	0	0	0	1	2	2	7	11	12	65

# Experiment

The average size of test suite are significantly reduced, the size of minimized suite generated by DelGreedy was of the same size or smaller than that generated by o other algorithms

**Table 5: Average size of minimized suite by DelGreedy**

Program	Branch Coverage	Def-Use Coverage
space	123	143
tcas	4	4
print-tokens	6	7
print-tokens2	4	8
schedule	2	2
replace	9	26
totinfo	2	5

**Average branch coverage suites of DelayGreedy is smaller than other algorithms,the smaller percentage**

	Greedy	HGS	SMSP
Smaller%	35%	64%	86%

**Percentage Average def-use coverage suites of DelayGreedy is smaller than other algorithms, the smaller percentage**

	Greedy	HGS	SMSP
Smaller%	39%	46%	91%

# Experiment

In most of the time, the delayGreedy algorithm will output the optimal result

**Table 6: Number of *Optimal size* (#Opt) and *Non-Optimal size* (#Non-Opt) test suites produced by each algorithm and time performance**

Prog.	Algo.	Branch Coverage Suites				Def-Use Coverage Suites			
		#Non-Opt.	#Opt.	#Un-Dec.	Time (sec)	#Non-Opt.	#Opt.	#Un-Dec.	Time (sec)
space	DelGreedy	-	92	8	.737	-	99	1	1.912
	Greedy	100	0	0	.444	100	0	0	1.932
	HGS	96	4	0	.307	93	7	0	.666
	SMSP	100	0	0	-	100	0	0	-
tcas	DelGreedy	-	68	32	.006	-	96	4	.006
	Greedy	43	37	20	.004	32	65	3	.004
	HGS	42	39	19	.002	2	94	4	.001
	SMSP	100	0	0	-	100	0	0	-
print tokens	DelGreedy	-	71	29	.006	-	92	8	.011
	Greedy	18	62	20	.005	22	70	8	.009
	HGS	57	35	8	.006	30	66	4	.008
	SMSP	100	0	0	-	100	0	0	-
print tokens2	DelGreedy	-	84	16	.010	-	80	20	.011
	Greedy	14	70	16	.007	38	50	12	.010
	HGS	51	29	20	.007	52	40	8	.008
	SMSP	100	0	0	-	100	0	0	-
schedule	DelGreedy	-	99	1	.003	-	91	9	.006
	Greedy	0	99	1	.003	0	91	9	.004
	HGS	63	36	1	.006	38	56	6	.008
	SMSP	1	99	0	-	34	66	0	-
replace	DelGreedy	-	53	47	.011	-	94	6	.027
	Greedy	51	25	24	.006	78	11	11	.021
	HGS	67	17	16	.006	71	28	1	.020
	SMSP	100	0	0	-	100	0	0	-
totinfo	DelGreedy	-	46	54	.004	-	88	12	.010
	Greedy	16	32	52	.004	2	87	11	.009
	HGS	73	11	16	.004	34	58	8	.008
	SMSP	99	1	0	-	100	0	0	-

# Summary and Questions

- **Contribution:**
  - Based on the framework of concept analysis, uses object reduction and attributes reduction and ownership reduction to reduce the size of concept relationship table, then do the heuristic search of the min-test-suite.
- **Question:**
  - Is there a more advanced mechanism to deal with the reduced table?