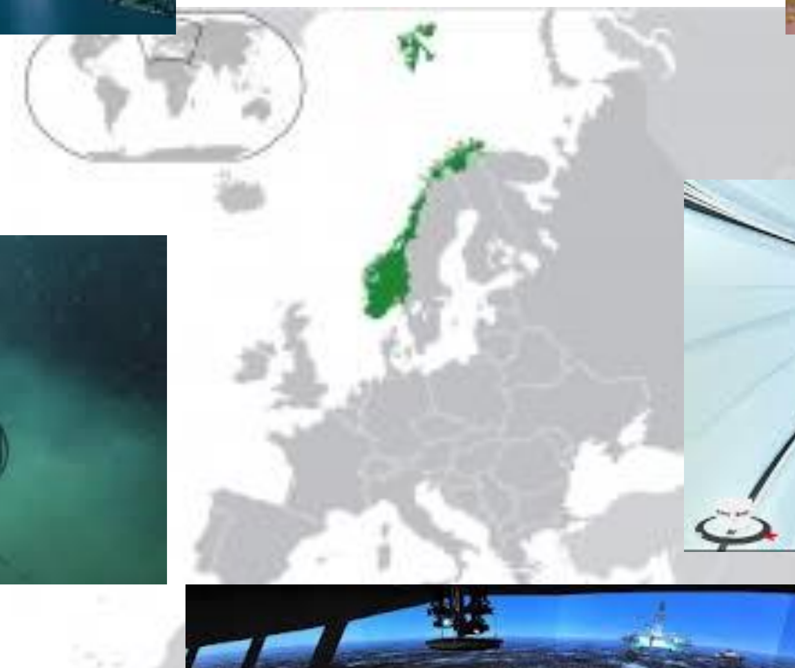
Global Constraints in Software Testing Applications

Arnaud Gotlieb

Simula Research Laboratory
Norway




The Certus Centre

Software Validation and Verification

Hosted by SIMULA

Established and awarded SFI in Oct. 2011

duration: 8 years

www.certus-sfi.no

Cisco Systems Norway

Norwegian Custom and excise

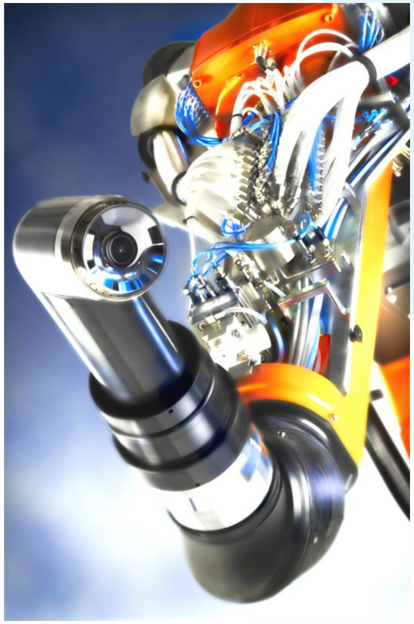


ABB Robotics Stavanger

Kongsberg Maritime

esito

[simula . research laboratory]
- by thinking constantly about it

Certus

sfi Centre for Research-based Innovation

Agenda

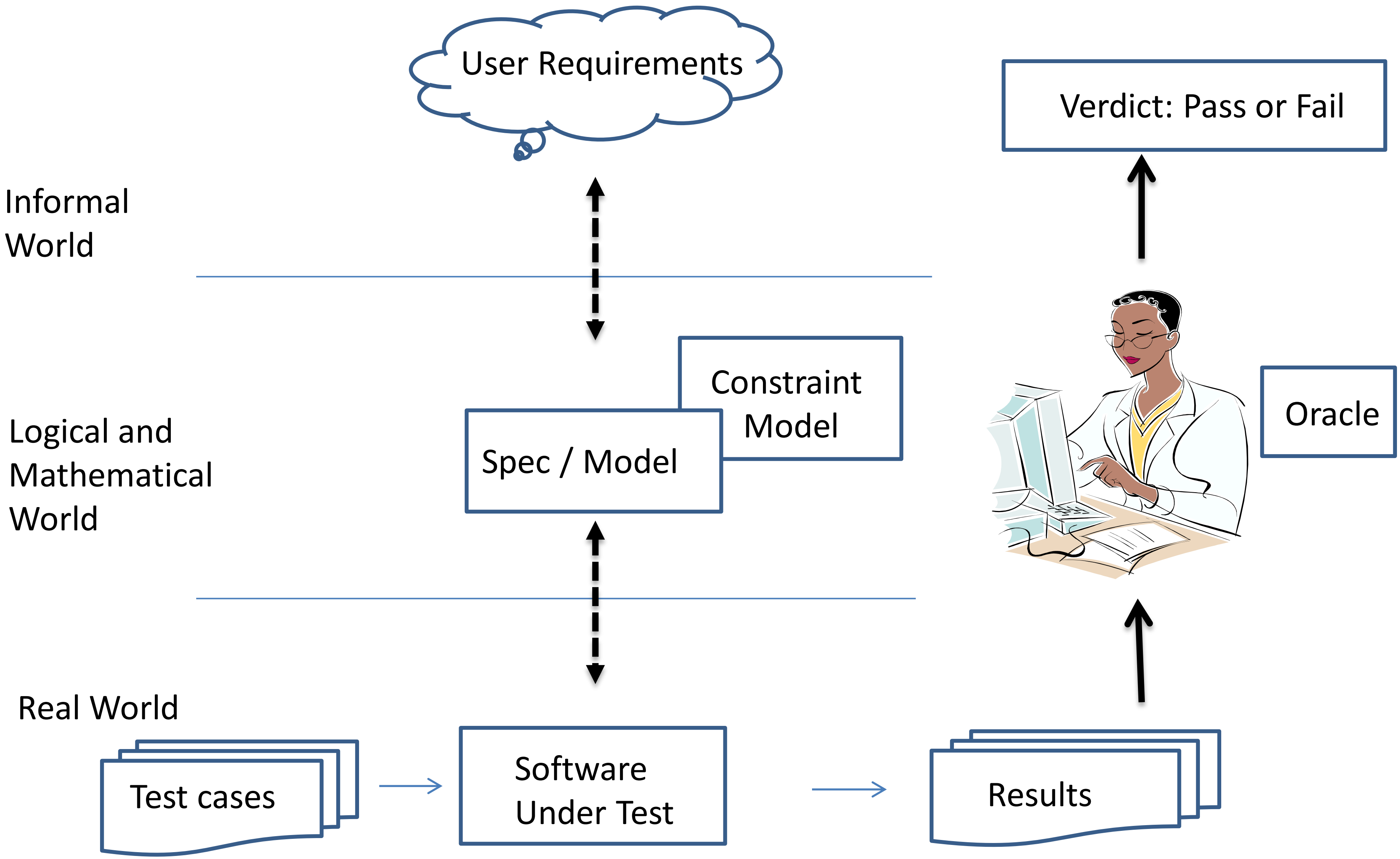
I. Software Testing

II. Optimal Test Suite Reduction

III. Automatic Test Case Generation

IV. Conclusions and Perspectives

Software Testing



Software Testing

Software test preparation is a **cognitively complex task**:

- Requires to understand both model and code to create interesting test cases ;
- Program's input space is usually very large (sometimes unbounded) ;
- Complex software (e.g., implementing ODEs or PDEs) yields to complex bugs ;
- Test oracles are hard to define (non-testable programs) ;

Not easily amenable to automation:

- Automatic test data generation is undecideable in the general case!
- Exploring the input space yields to combinatorial explosion ;
- Fully automated oracles are usually not available ;

How software testing differs from other program verification techniques?

☞ **Static analysis** finds simple faults (division-by-zero, overflows, …) at compile-time, while **software testing** finds functional faults at run-time (P returns 3 while 2 was expected)

☞ **Program proving** aims at formally proving mathematical invariants, while **software testing** evaluates the program in its execution environment

☞ **Model-checking** explores paths of a model of the software under test for checking temporal properties or finding counter-examples, while **software testing** is based on program executions

Some Hot Research Topics in Software Testing

- ❑ Automatic test case generation

  Find test cases to exercise specific behaviors, to execute specific code locations, to cover some test objectives (e.g., all-statements, all-k-paths)

- ❑ Test suite reduction, test suite prioritization, test execution scheduling

- ❑ Robustness and performence testing

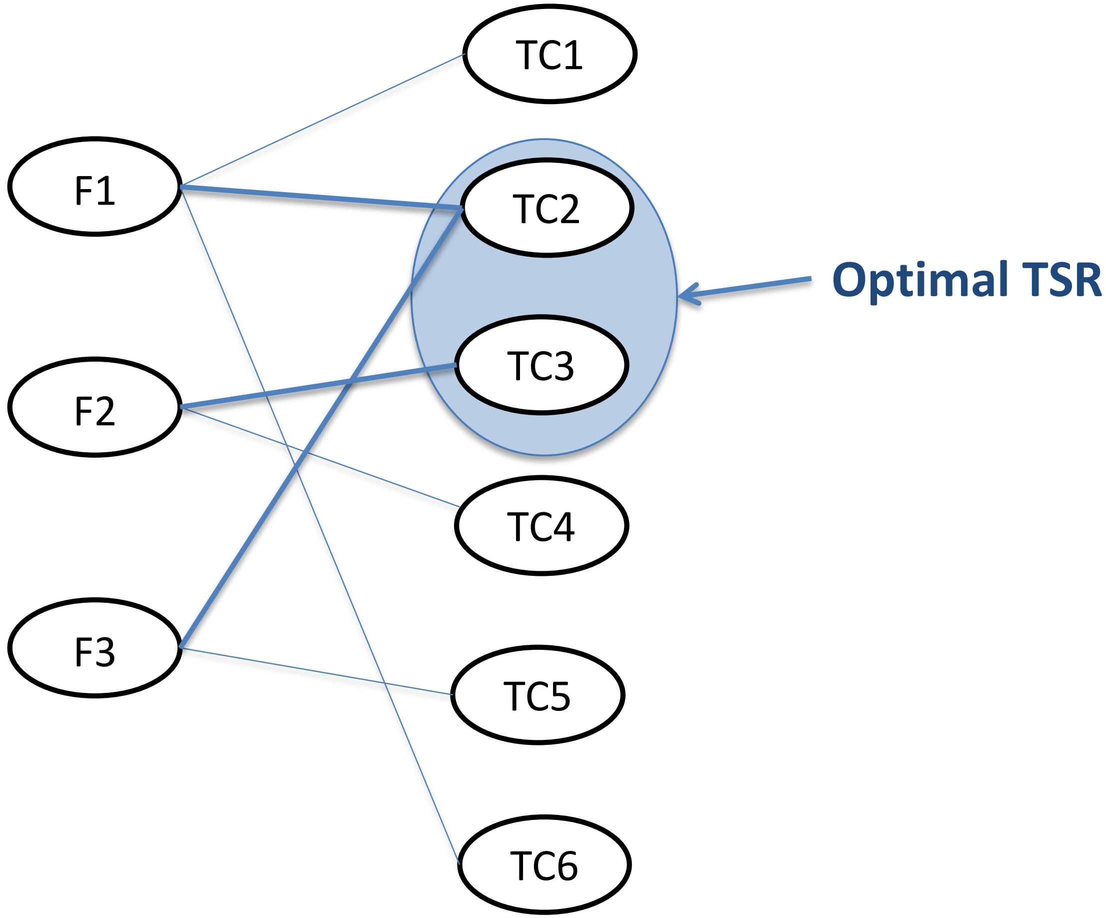
- ❑ Testing complex code (e.g., floating-point and iterative computations)

Our thesis: Global constraints can efficiently tackle these problems!

(*High-level primitives with specialised filtering algorithms*)

Optimal Test Suite Reduction

Optimal TSR: the core problem



Optimal TSR: find a minimal subset of TC such that each F is covered at least once
(Practical importance but NP-hard problem!) – An instance of *Minimum Set Cover*

The nvalue global constraint

$$nvalue(\, n, v)$$

Where:

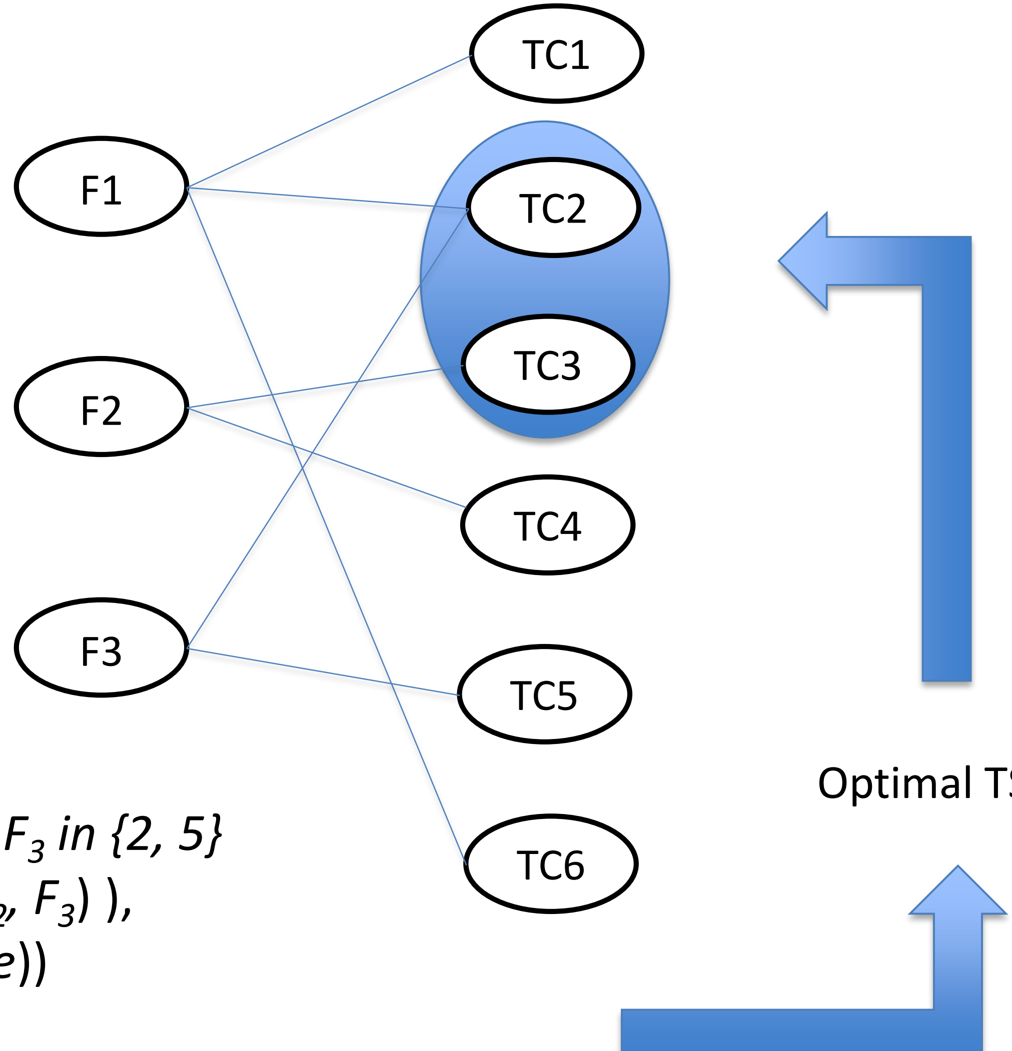
n is an FD_variable

$v = (v_1, \ldots, \ v_k)$ is a vector of FD_variables

$nvalue(n, v)$ holds iff $n = card(\ \{v_i\}_{\, i \ in \ 1\,..\,k})$

Introduced in [Pachet and Roy'99], first filtering algorithm in [Beldiceanu'01]
Solution existence for nvalue is NP-hard [Bessiere et al. '04]

Optimal TSR: CP model with nvalue (1)



F_1 in {1, 2, 6}, F_2 in {3, 4}, F_3 in {2, 5}
nvalue(*MaxNvalue*, (F_1, F_2, F_3)),
label(minimize(*MaxNvalue*))

/* branch-and-bound search among feasible solutions */

The global_cardinality constraint

$$gcc(\ t, d, v)$$

Where

$t = (t_1, \ldots, \ t_N)$ is a vector of N variables, each t_j in $Min_j .. Max_j$

$d = (d_1, \ldots., d_k)$ is a vector of k values

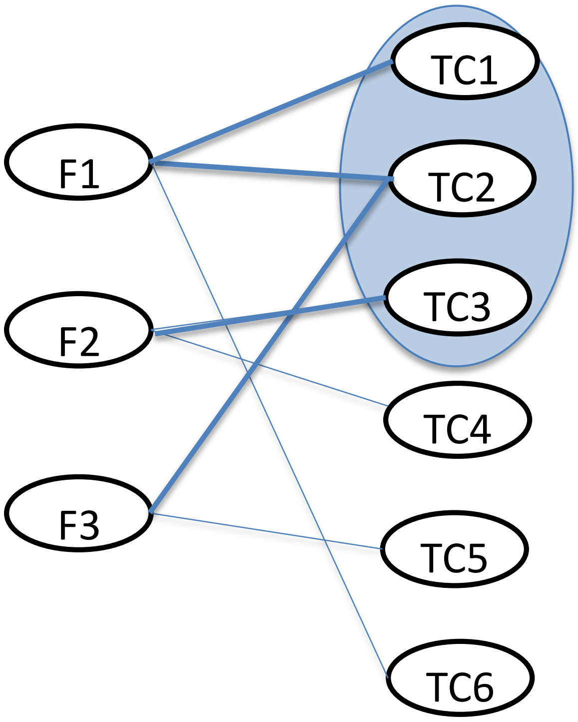
$v = (v_1, \ldots, \ v_k)$ is a vector of k variables, each v_i in $Min_i..Max_i$

$gcc(t, d, v)$ holds iff

$$\forall i \ in \ 1..k,$$
$$v_i = card(\ \{t_j = di\}_{j \ in \ 1..N})$$

Filtering algorithms for *gcc* are based on max flow computations in a network flow [Regin AAAI'96]

Example



gcc((F_1, F_2, F_3), (1,2,3,4,5,6), (V_1,V_2,V_3,V_4,V_5,V_6))
means that:

In a solution of TSR
TC_1 covers exactly V_1 requirements in (F_1, F_2, F_3)
TC_2 " V_2 "
TC_3 " V_3 "
...

Where F_1, F_2, F_3, V_1, V_2, V_3, ... denote finite-domain variables

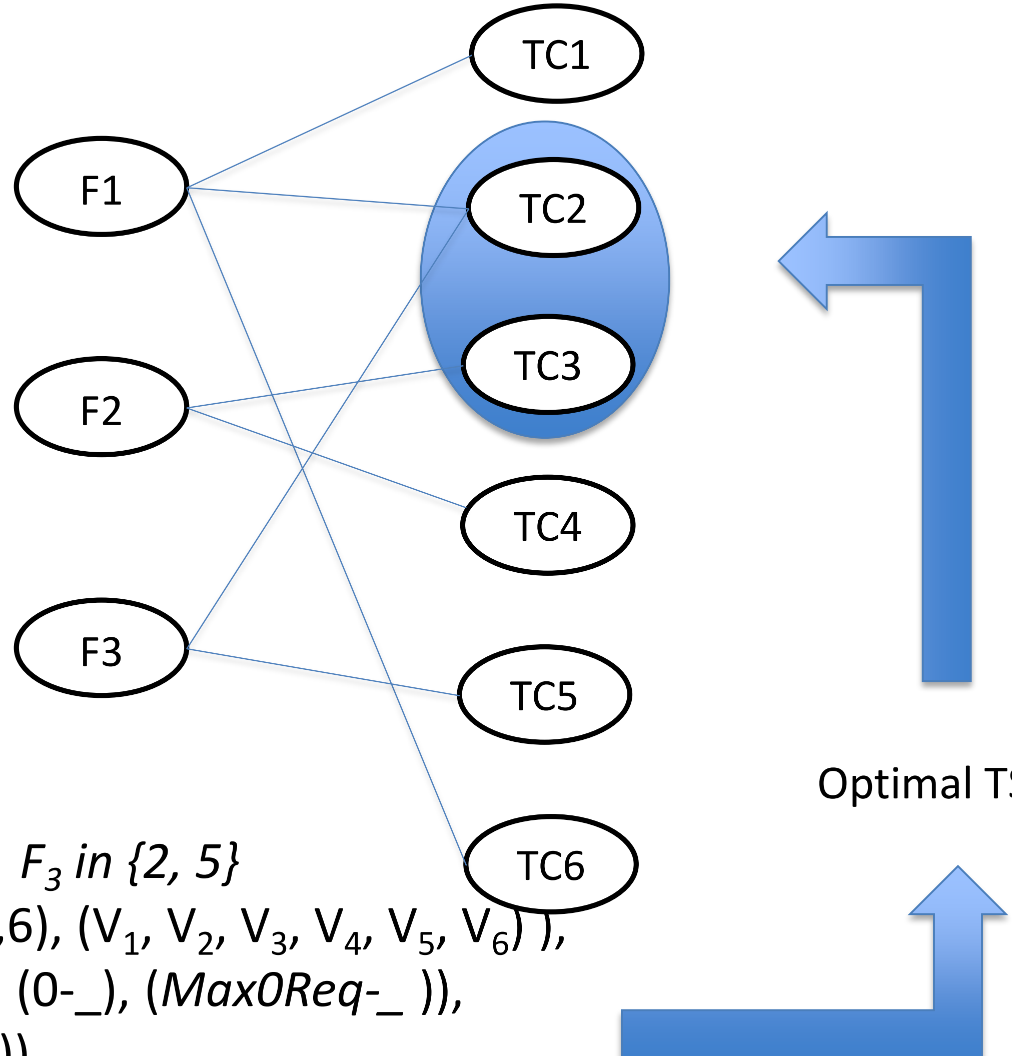
F_1 in {1, 2, 6}, F_2 in {3, 4}, F_3 in {2, 5}
V_1 in {0, 1}, V_2 in {0, 2}, V_3 in {0, 1}, V_4 in {0, 1}, V_5 in {0, 1}, V_6 in {0, 1}

Here, for example, **V_1 = 1, V_2 = 2, V_3 = 1, V_4 = 0, V_5 = 0, V_6 = 0** is a feasible solution

But, not an optimal one!

Optimal TSR: CP model with two gcc (2)

[Gotlieb Marijan ISSTA'2014]



F_1 *in {1, 2, 6},* F_2 *in {3, 4},* F_3 *in {2, 5}*
gcc((F_1, F_2, F_3), (1,2,3,4,5,6), (V_1, V_2, V_3, V_4, V_5, V_6)),
gcc((V_1, V_2, V_3, V_4, V_5, V_6), (0-_), (*MaxOReq*-_)),
label(maximize(*MaxOReq*))

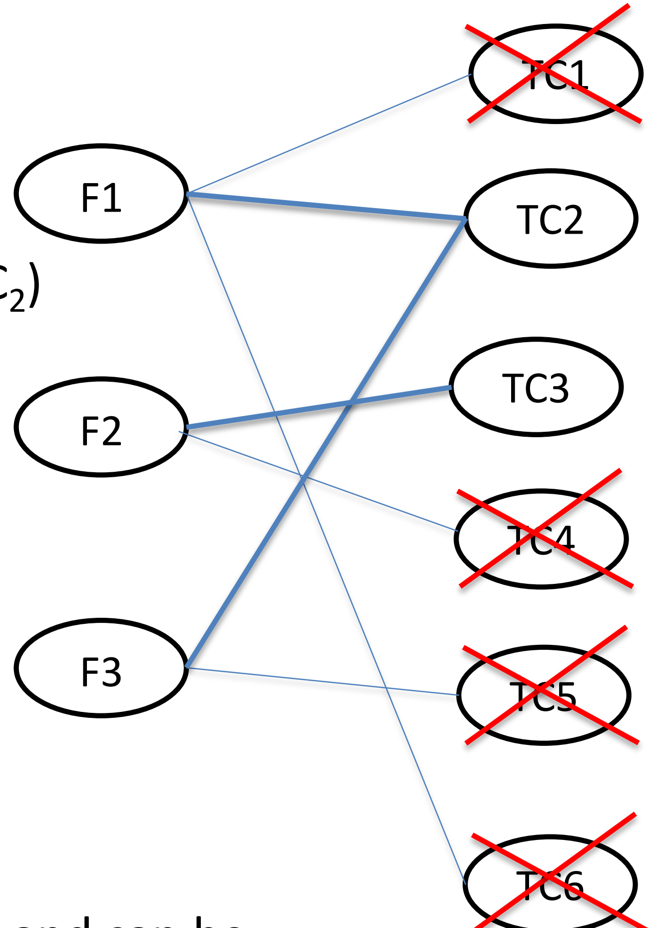
/* search heuristics by enumerating the Vi first */

Introducing model pre-treatment

F_1 in {1, 2, 6} → $F_1 = 2$ as $cov(TC_1) = cov(TC_6) \subset cov(TC_2)$
withdraw TC_1 and TC_6

F_3 is covered → withdraw TC_5

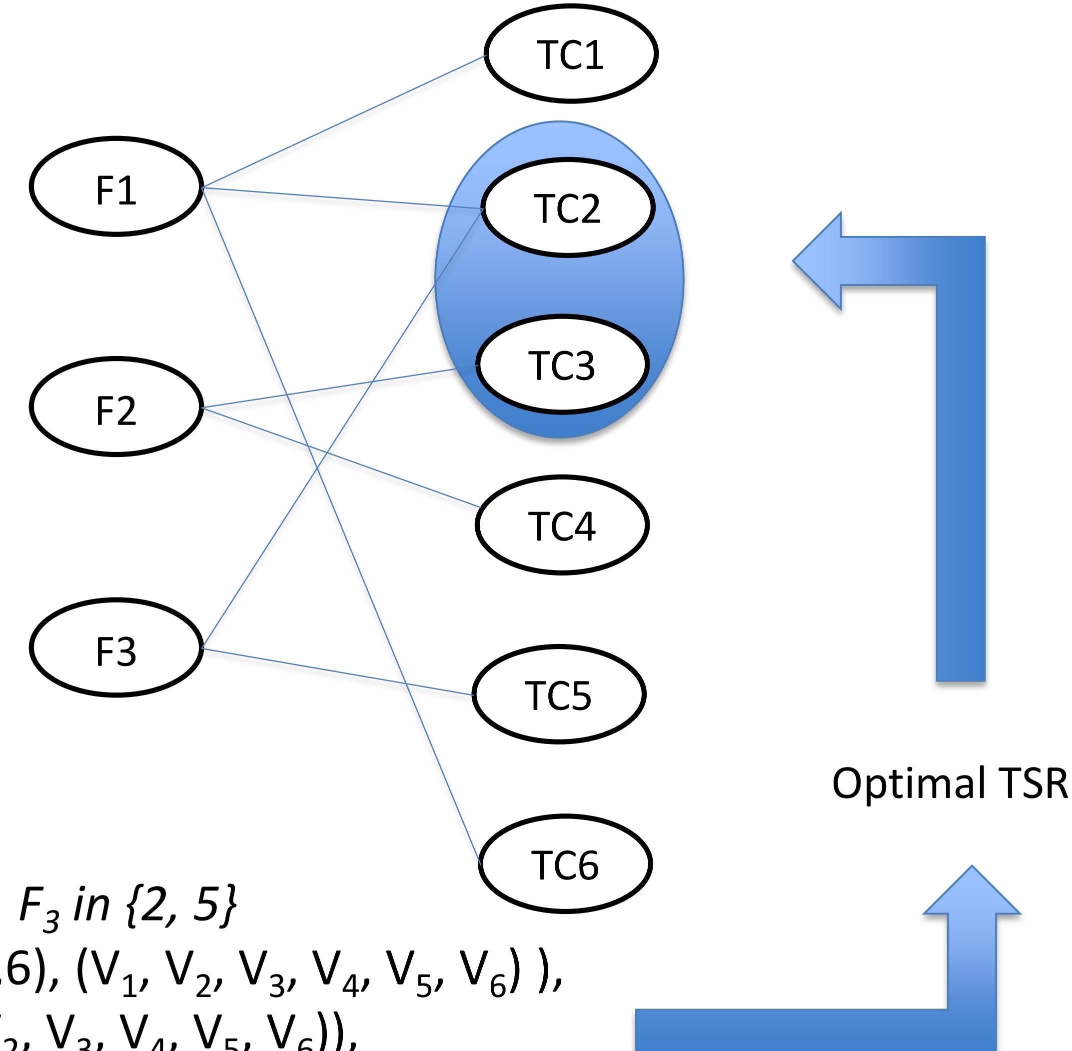
F_2 in {3,4} → e.g., $F_2 = 3$, withdraw TC_4



Three such pre-treatment rules have been identified and can be included to simplify the problem

But, they are currently statically applied!

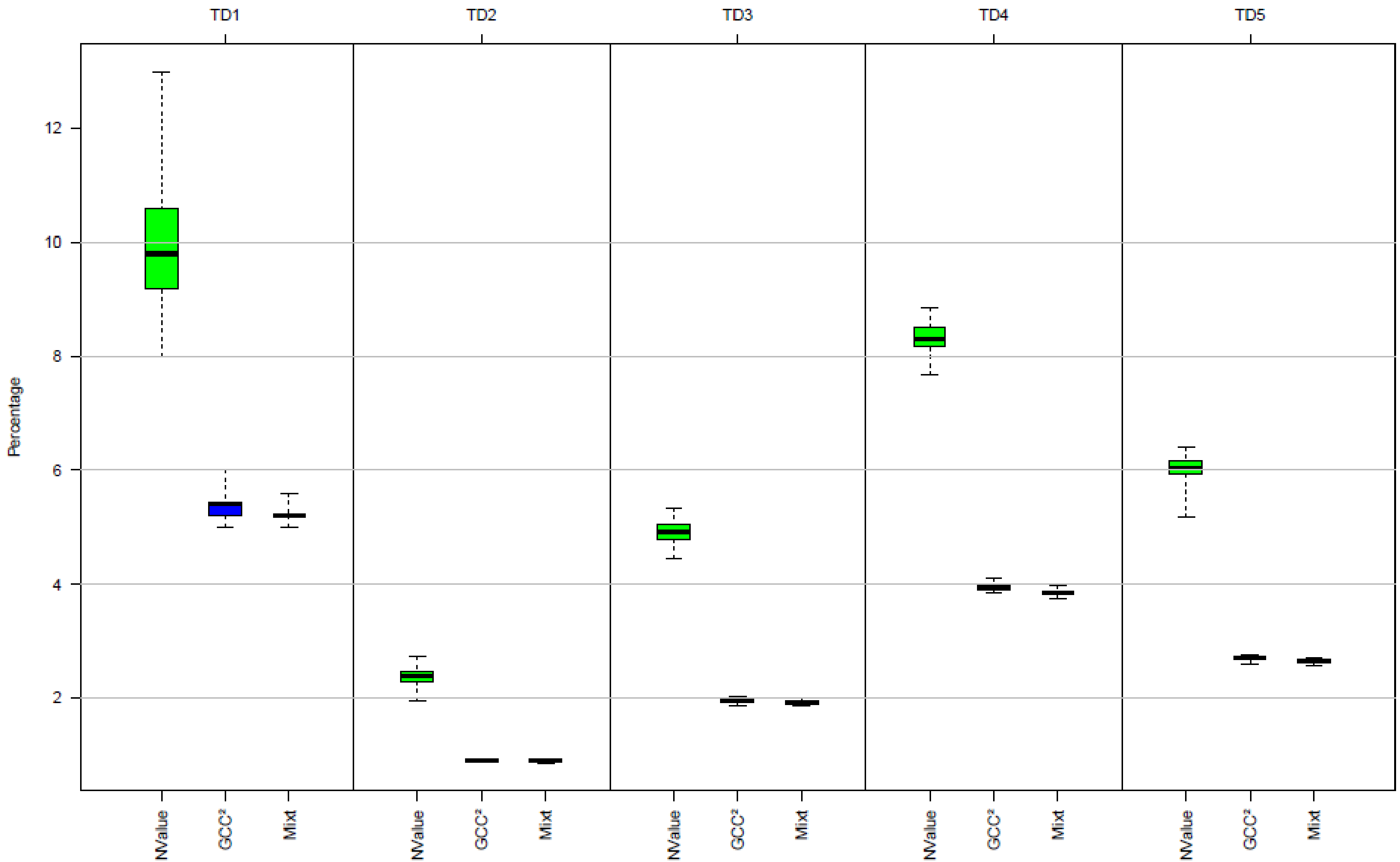
3. Optimal TSR: CP model Mixt (3)



F_1 *in {1, 2, 6}*, F_2 *in {3, 4}*, F_3 *in {2, 5}*
gcc((F_1, F_2, F_3), (1,2,3,4,5,6), (V_1, V_2, V_3, V_4, V_5, V_6)),
nvalue(MaxNvalue, (V_1, V_2, V_3, V_4, V_5, V_6)),
label(minimize(*MaxNvalue*))

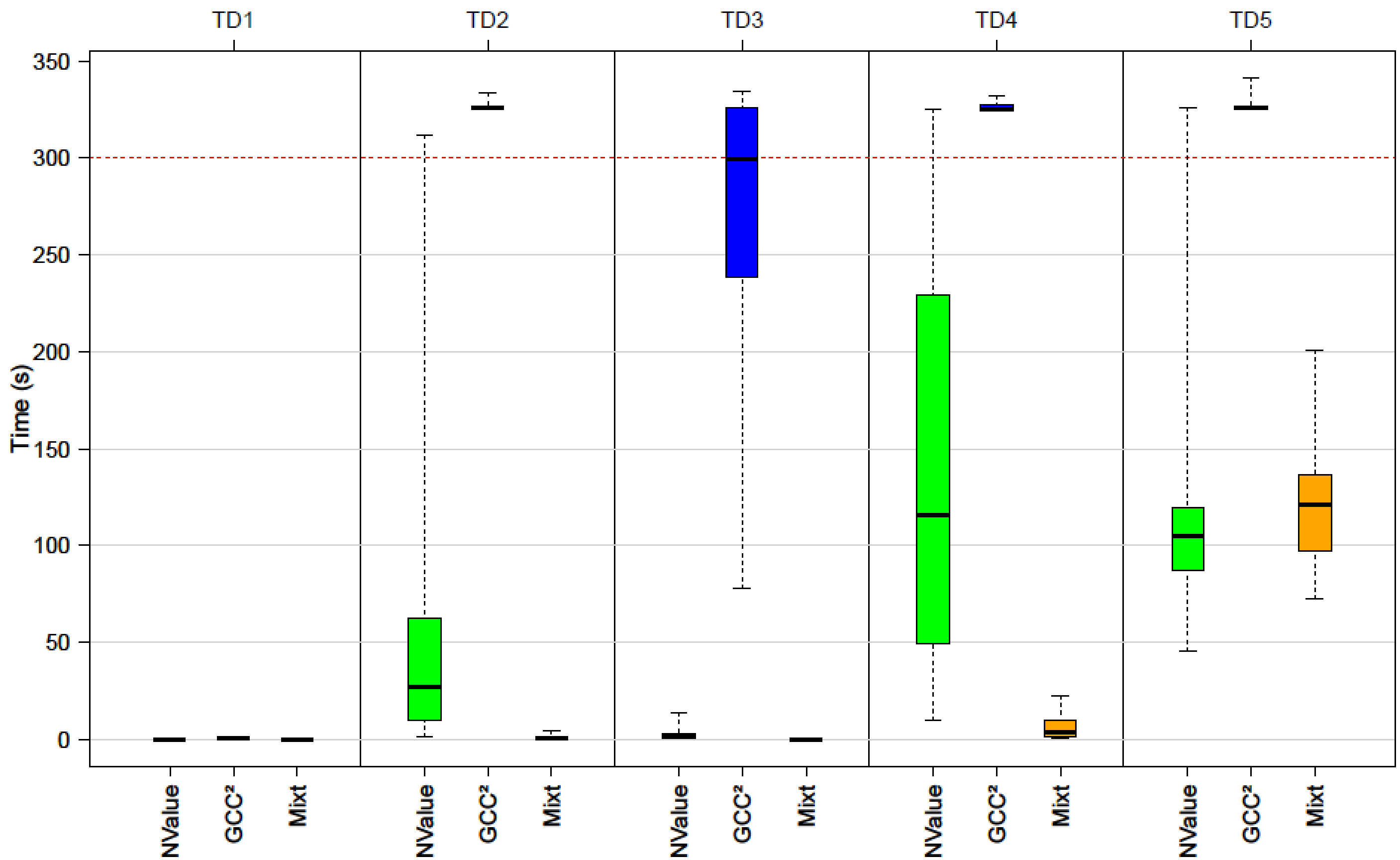
/* + pre-treatment + labelling heuristics based on max */

Model comparison on random instances (Reduced Test Suite percentage in 30sec of search)



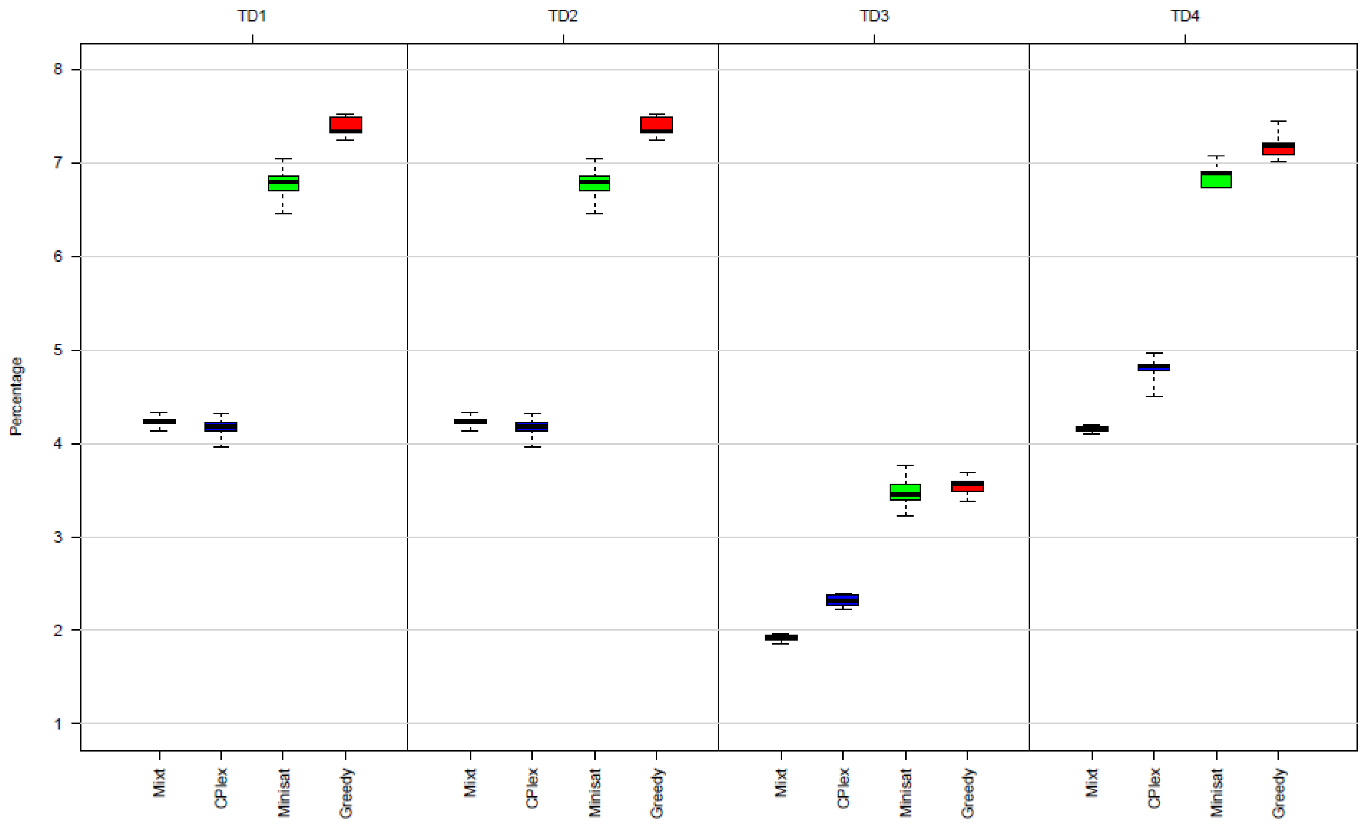
	TD1	TD2	TD3	TD4	TD5
Requirements	250	500	1000	1000	1000
Test cases	500	5000	5000	5000	7000
Density	20	20	20	8	8

Model comparison on random instances (CPU time to find a global optimum)



	TD1	TD2	TD3	TD4	TD5
Requirements	20	90	60	60	30
Test cases	70	100	100	200	500
Density	8	20	20	20	8

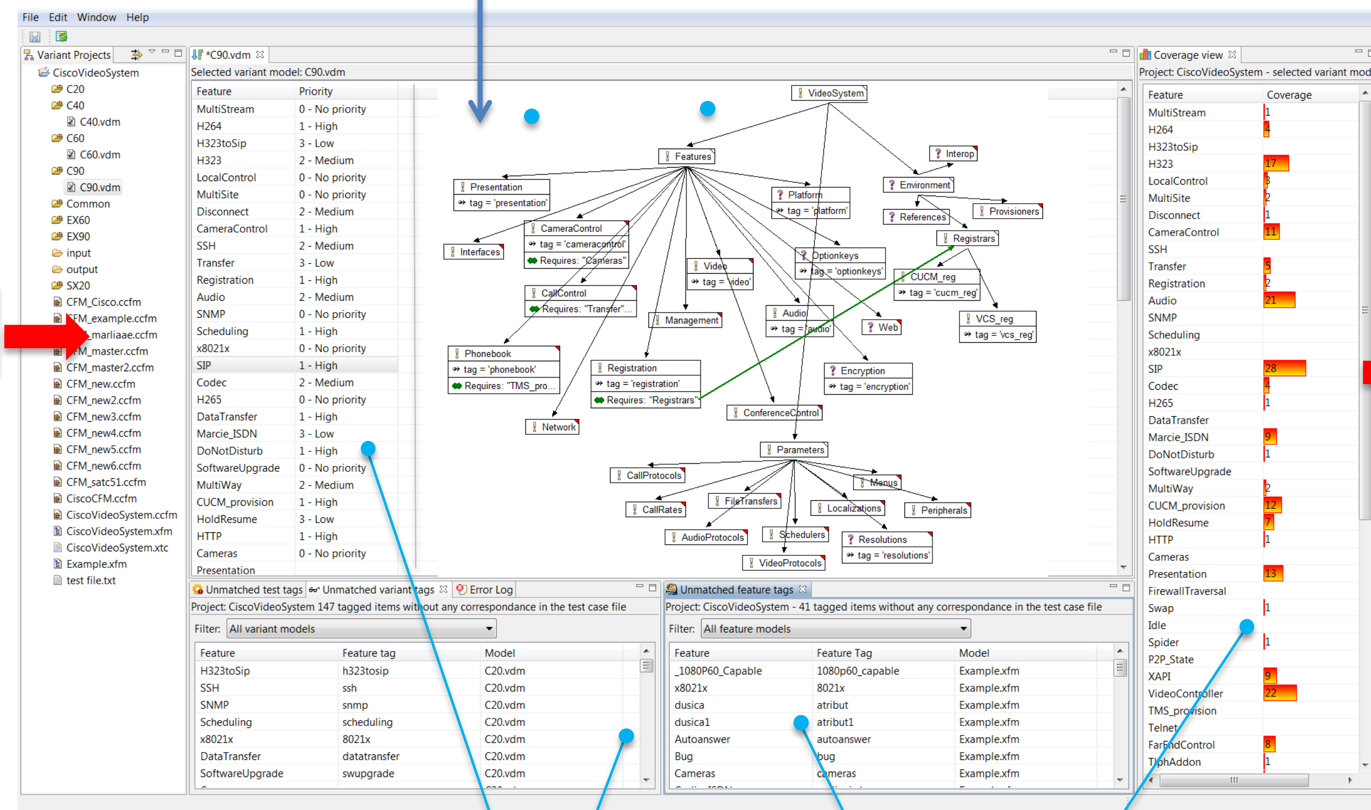
Comparison with other approaches (Reduced Test Suite percentage in 60 sec)



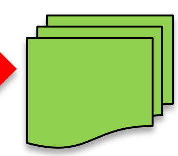
	TD1	TD2	TD3	TD4
Requirements	1000	1000	1000	2000
Test cases	5000	5000	5000	5000
Density	7	7	20	20

TITAN [Marijan et al. SPLC’13, SPLC’14]

Variability model to describe a software product line



Unoptimized test suite



Optimized (reduced/ prioritized) test suite

Diagnostic views, feature coverage

Industrial case studies: ABB, Cisco

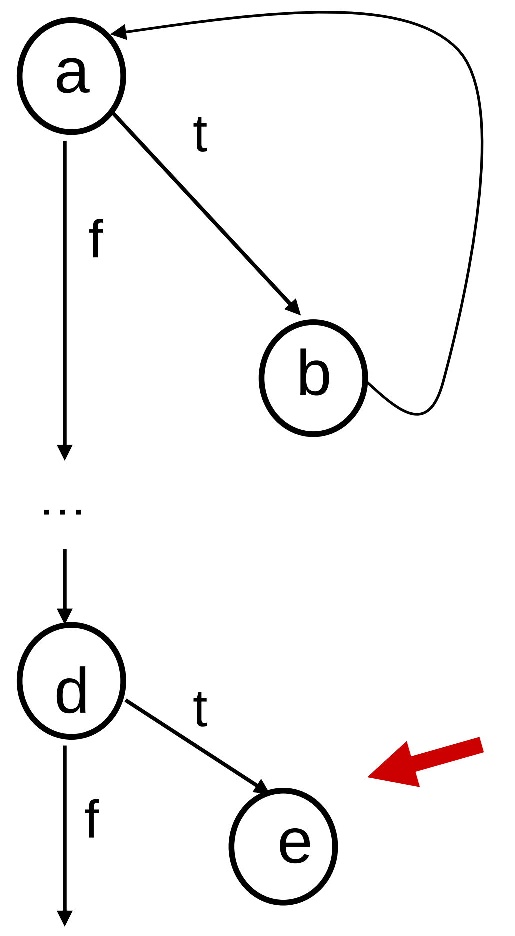
Automatic Test Case Generation

An example problem

```
  f(  int i, …  )
   {
a.      j = 100;
        while( i > 1)
b.            { j++ ; i-- ;}


    …
d. if( j > 500)
e.      …
```

value of i to reach e ?

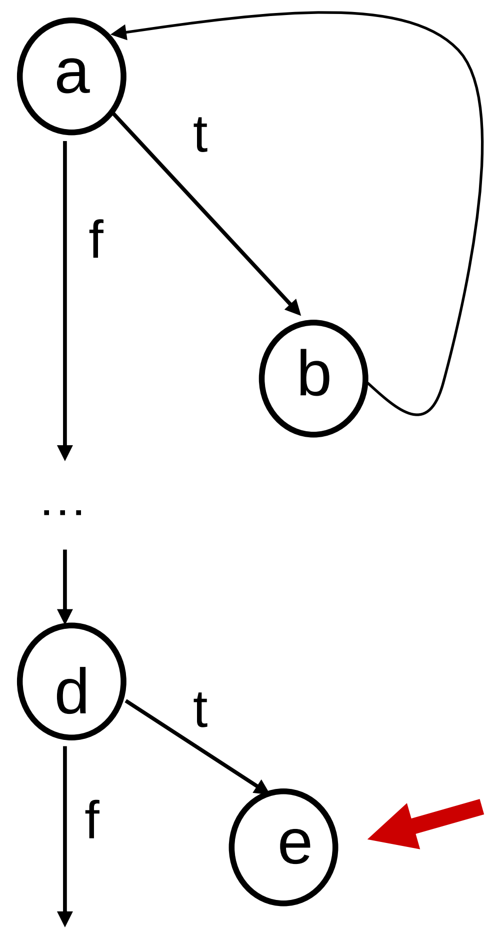


Undecideable problem!

Path-oriented exploration

```
  f(  int i, …  )
   {
a.      j = 100;
        while( i > 1)
b.          { j++ ; i-- ;}

    …
d. if( j > 500)
e.       …
```



1. Path selection

   e.g., $(a\text{-}b)^{14}\text{-}\ldots\text{-}d\text{-}e$

2. Path condition generation (via symbolic exec.)

   $j_1=100,\ i_1>1,\ j_2=j_1+1,\ i_2=i_1-1,\ i_2>1,\ldots,\ j_{15}>500$

3. Path condition solving

   unsatisfiable → FAIL

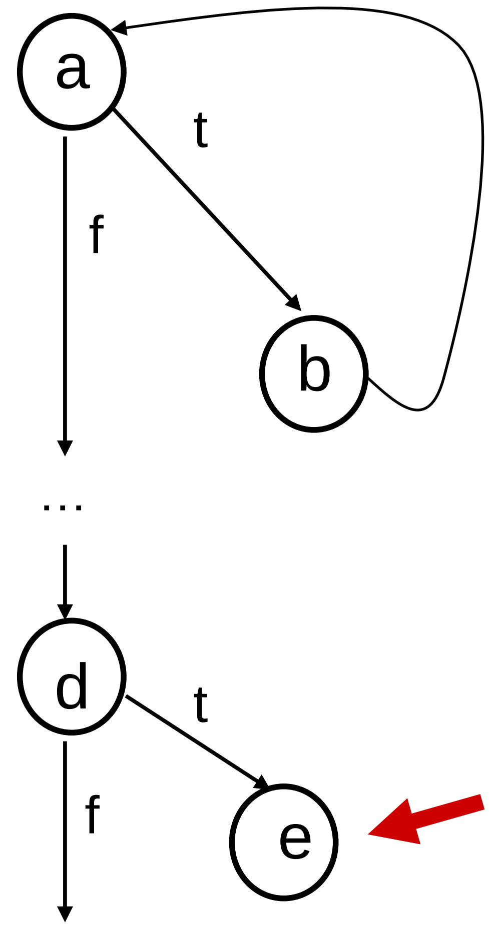
Backtrack !

Constraint-based program exploration

(Gotlieb et al. ISSTA'98)

```
 f(  int i, …   )
  {
a.     j = 100;
       while( i > 1)
b.         { j++ ; i-- ;}

    …
d. if( j > 500)
e.       …
```



1. Program (under Static Single Assignment form) as constraints

2. Control dependencies generation;

j_1=100, $i_3 \leq 1$, $j_3 > 500$

3. Global constraint reasoning

$j_1 \neq j_3$ entailed ➔ unroll the loop 400 times ➔ i_1 in 401 .. 2^{31}-1

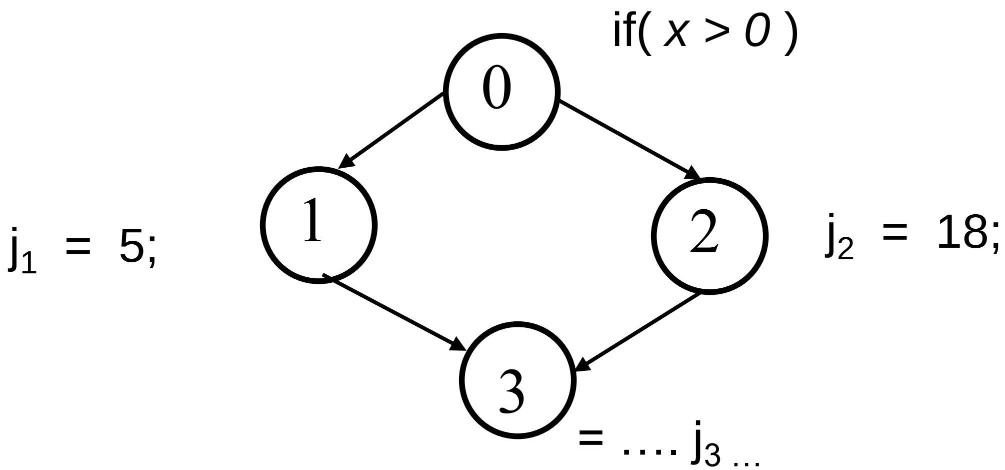
No backtrack !

Program as constraints

- ✓ Type declaration: signed long x; → x in $-2^{31}..2^{31}-1$
- ✓ Assignments: i*=++i ; → $i_2 = (i_1+1)^2$
- ✓ Memory and array accesses and updates *(Charreteur et al. JSS'09, Bardin et al. CPAIOR'12):*
  v=A[i] (or p=Mem[&p]) → variations of element/3
- ✓ Control structures: dedicated global constraints

Conditionnals (SSA)	if D then $C_{1;}$ else C_2	→	ite/6
Loops (SSA)	while D do C	→	w/5

Conditional as a global constraint: ite/6

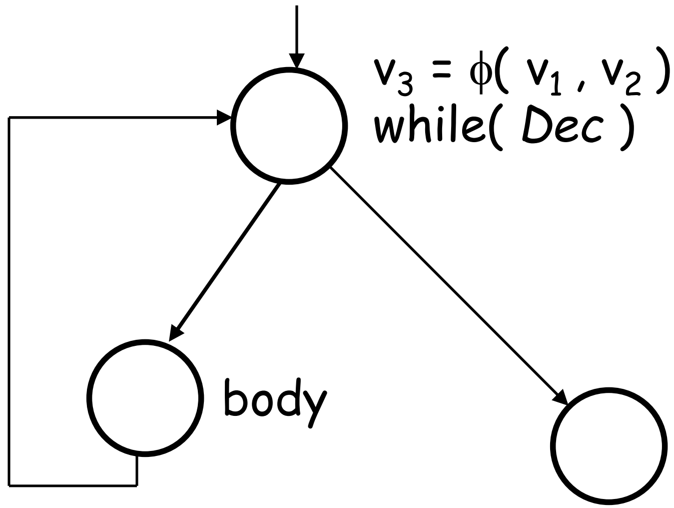


ite($x > 0$, j_1, j_2, j_3, $j_1 = 5$, $j_2 = 18$)   iff

- $x > 0 \quad \rightarrow \quad j_1 = 5 \;\wedge\; j_3 = j_1$
- $\neg(x > 0) \quad \rightarrow \quad j_2 = 18 \;\wedge\; j_3 = j_2$
- $\neg(x > 0 \;\wedge\; j_1 = 5 \;\wedge\; j_3 = j_1) \;\rightarrow\; \neg(x > 0) \wedge\; j_2 = 18 \;\wedge j_3 = j_2$
- $\neg(\neg(x > 0) \wedge\; j_3 = j_2) \;\rightarrow\; x > 0 \wedge\; j_1 = 5 \;\wedge j_3 = j_1$
- Join($x > 0 \wedge j_1 = 5 \wedge\; j_3 = j_1$, $\neg(x > 0) \wedge\; j_1 = 18 \wedge\; j_3 = j_2$)

Implemented as a global constraint: interface, awakening conditions, filtering algo.

While loop as a global constraint: w/5



$w(\mathrm{Dec}, V_1, V_2, V_3, \mathrm{body})$ iff

- $\mathrm{Dec}_{V3\leftarrow V1} \rightarrow \mathrm{body}_{V3\leftarrow V1} \wedge \mathbf{w}(\mathrm{Dec}, v_2, v_{new}, v_3, \mathrm{body}_{V2\leftarrow Vnew})$
- $\neg\mathrm{Dec}_{V3\leftarrow V1} \rightarrow v_3 = v_1$

- $\neg(\mathrm{Dec}_{V3\leftarrow V1} \wedge \mathrm{body}_{V3\leftarrow V1}) \rightarrow \neg\mathrm{Dec}_{V3\leftarrow V1} \wedge v_3 = v_1$
- $\neg(\neg\mathrm{Dec}_{V3\leftarrow V1} \wedge v_3 = v_1) \rightarrow \mathrm{Dec}_{V3\leftarrow V1} \wedge \mathrm{body}_{V3\leftarrow V1} \wedge \mathbf{w}(\mathrm{Dec}, v_2, v_{new}, v_3, \mathrm{body}_{V2\leftarrow Vnew})$

- $\mathrm{join}(\mathrm{Dec}_{V3\leftarrow V1} \wedge \mathrm{body}_{V3\leftarrow V1} \wedge \mathbf{w}(\mathrm{Dec}, v_2, v_{new}, v_3, \mathrm{body}_{V2\leftarrow Vnew}),\ \neg\mathrm{Dec}_{V3\leftarrow V1} \wedge v_3 = v_1)$

```
f(  int i  ) {
  j = 100;
  while( i > 1)
   { j++ ; i-- ;}
  ...
  if( j > 500)
    ...
```

w(Dec, V_1, V_2, V_3, body) :-

- $Dec_{V3\leftarrow V1} \rightarrow body_{V3\leftarrow V1} \wedge$ **w**($Dec, v_2, v_{new}, v_3, body_{V2\leftarrow Vnew}$)
- $\neg Dec_{V3\leftarrow V1} \rightarrow v_3 = v_1$
- $\neg(Dec_{V3\leftarrow V1} \wedge body_{V3\leftarrow V1}) \rightarrow \neg Dec_{V3\leftarrow V1} \wedge v_3 = v_1$
- $\neg(\neg Dec_{V3\leftarrow V1} \wedge v_3 = v_1) \rightarrow$
  $Dec_{V3\leftarrow V1} \wedge body_{V3\leftarrow V1} \wedge$ **w**($Dec, v_2, v_{new}, v_3, body_{V2\leftarrow Vnew}$)
- join($Dec_{V3\leftarrow V1} \wedge body_{V3\leftarrow V1} \wedge$ **w**($Dec, v_2, v_{new}, v_3, body_{V2\leftarrow Vnew}$,
  $\neg Dec_{V3\leftarrow V1} \wedge v_3 = v_1$)

$i = 23, j_1 = 100$?

no

i in $401..2^{31}-1$

$$\mathbf{w}(i_3 > 1, (i, j_1), (i_2, j_2), (i_3, j_3), \; j_2 = j_3 + 1 \wedge i_2 = i_3 - 1)$$

$i_3 = 1, j_3 = 122$

$i_3 = 10$?

$j_1 = 100$,

$j_3 > 500$?

Features of the w/5 relation

- ✓ It can be imbricated with other relations (e.g., nested loops w($cond_1$, v_1,v_2,v_3, w($cond_2$, ...)) – It handles unbounded loops

- ✓ Managed by the solver as any other constraint (its consistency is iteratively checked, awakening conditions, success/failure/suspension)

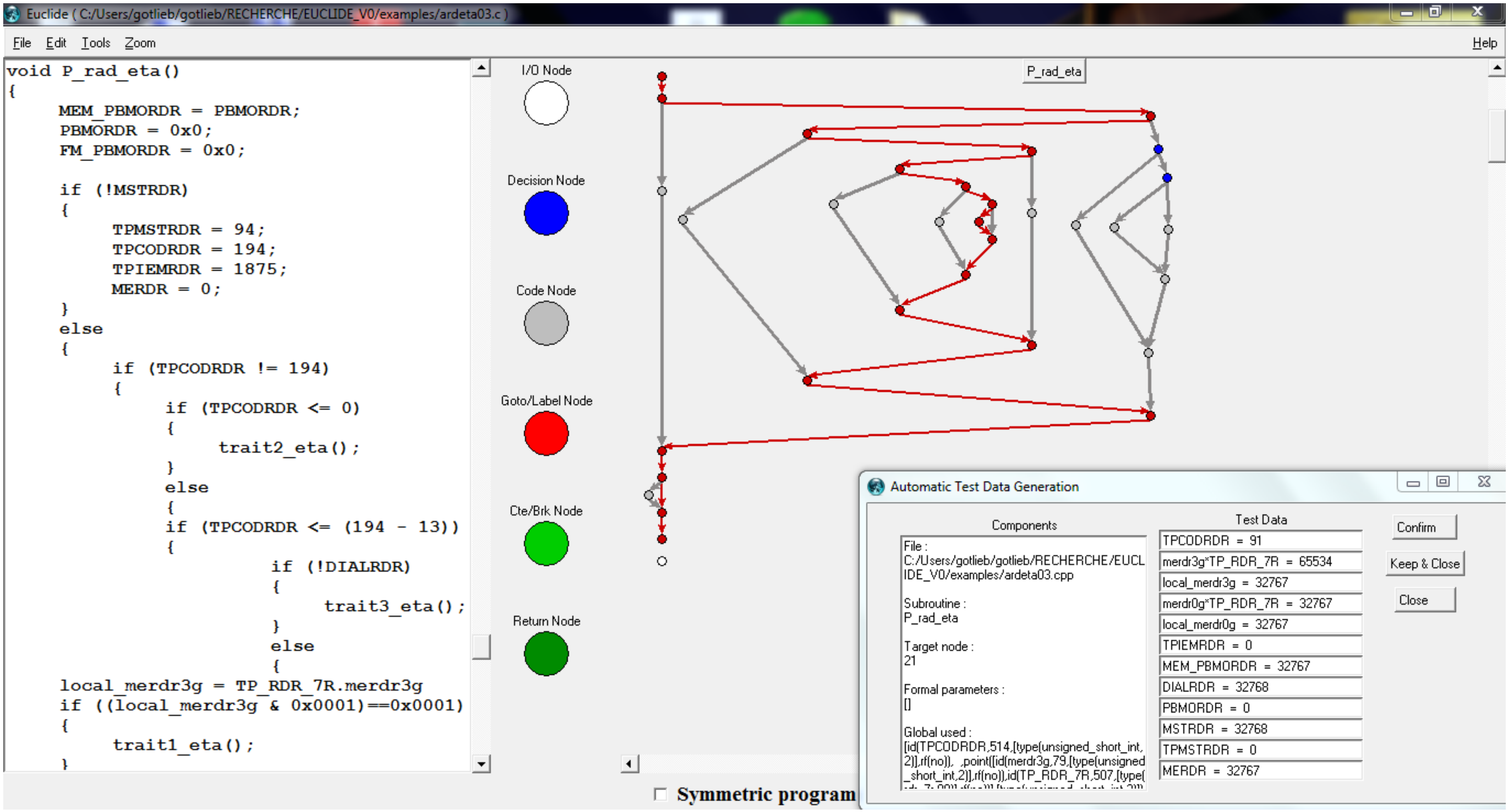
- ✓ By construction, w is unfolded only when necessary but w may NOT terminate ! (only a semi-correct procedure)

- ✓ Join is implemented using Abstract Interpretation operators (interval union, Q-polyhedra weak-join operator, simple widening operators)

(Gotlieb et al. CL'2000, Denmat et al. CP'2006)

EUCLIDE: Automatic Test Case Generation for C Programs

[Gotlieb ICST'09, KER'12]



Conclusions

- Global constraints (existing ones or user-defined) can efficiently and effectively tackle difficult software testing problems – experimental results and industrial case studies

- So far, only a few subset of existing global constraints have been explored for that purpose (e.g., nvalue, gcc, element, all_different,…)

- Some software testing problems require the creation of dedicated global constraints to facilitate disjunctive reasoning, case-based reasoning or probabilistic reasoning

→ there is room for research in that area!

Perspectives

- More industrial case studies for demonstrating_the potential of global constraints for software testing applications
- Using **GCC WITH COSTS** to deal with bi-objective optimisation in test suite reduction (e.g., to also select test cases based on execution time in addition to reauirement coverage)
- Test Case Execution Scheduling with **CUMULATIVE**

Thanks to my co-authors:

Bernard Botella (CEA)
Mats Carlsson (SICS)
Tristan Denmat (Inria)
MariusLiaeen (CISCO)
Dusica Marijan (SIMULA)
Alexandre Pétillon (SIMULA)