CAVERN

Constraints and Abstractions for program VERificatioN

ANR SESUR 2007
(Fév. 2008 – Déc. 2011)

Changing the rules of business

Arnaud Gotlieb

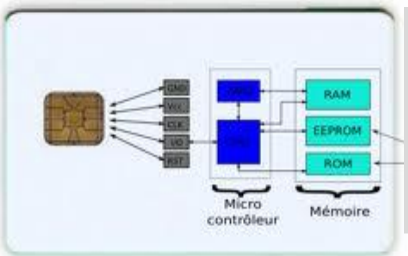
INRIA Rennes, France
SIMULA Oslo, Norway

Congrès ANR STIC, Lyon, 6 Janvier 2012

Why Testing is so important?

for checking unspecified behaviour...

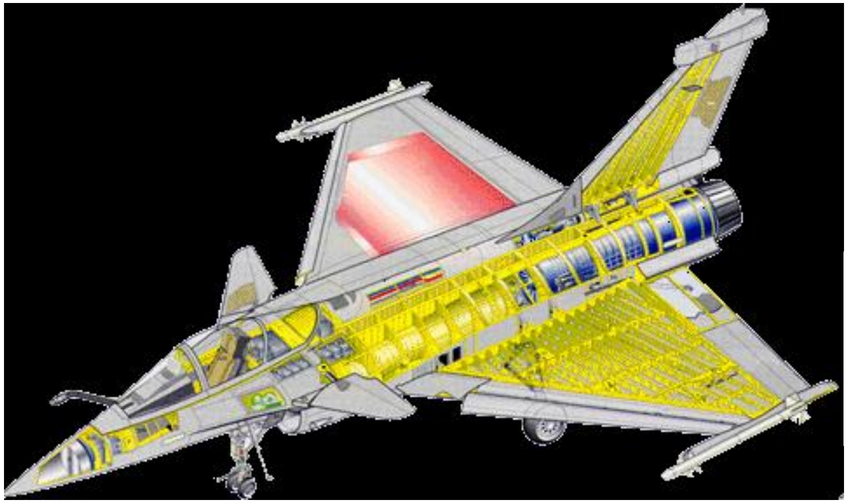
Embedded Software Testing



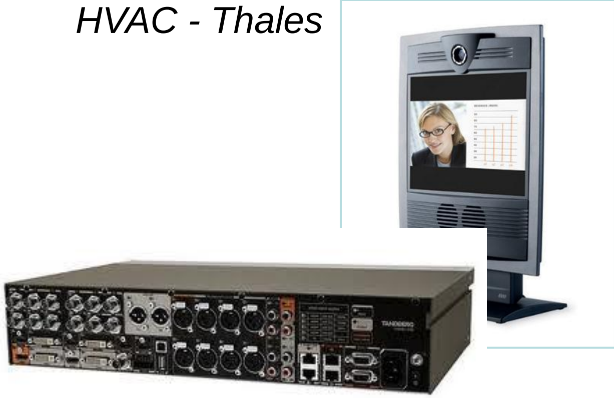
Java Card - Oberthur

HVAC - Thales

TCAS



BCE Rafale – Dassault Electronics



Saturn C90 – Cisco Norway

Critical software development involves strong V&V requirements:

- At system testing level, safety-related properties have to be checked
- At integration testing level, HW/SW integration failures must be detected
- At unit testing level, programming faults must be detected and removed

At these levels, conformance to software certification standards is enforced

Several (complementary) techniques at the unit level (code level):

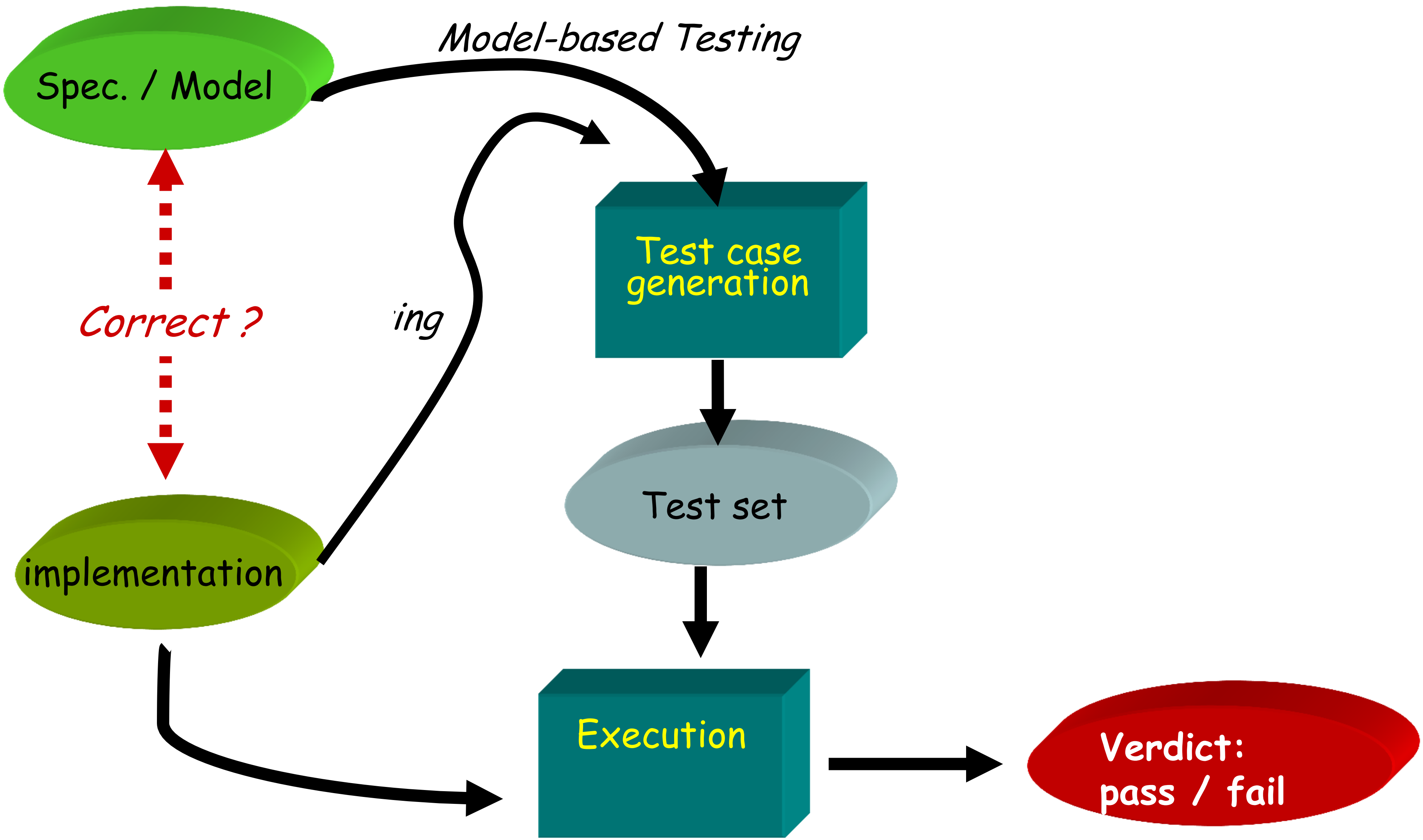
software model-checking

static-analysis based verification

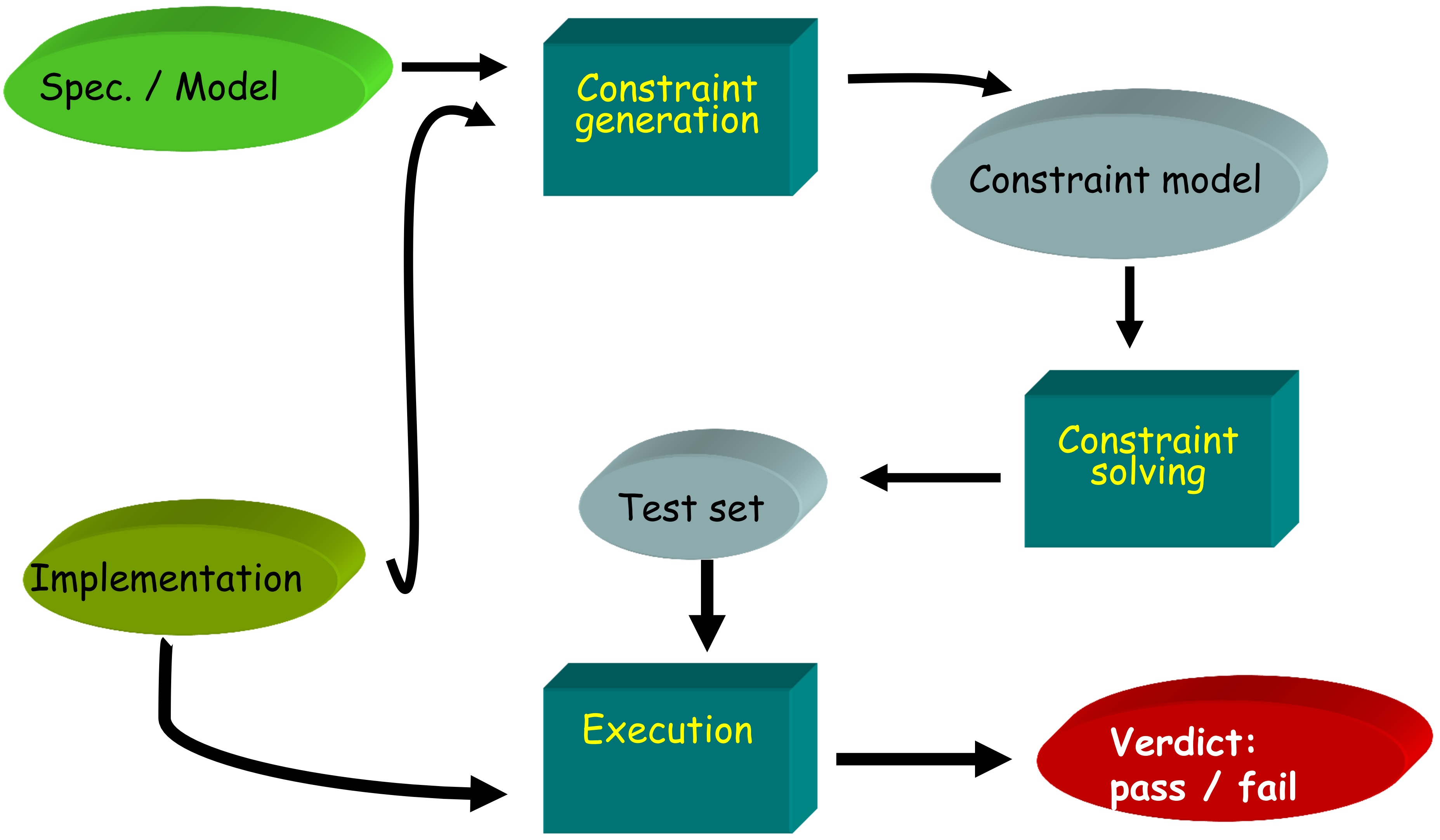
software testing

and Constraint-Based Testing…

Software Testing



Constraint-Based Testing



Constraint-Based Testing (CBT)

Constraint-Based Testing (CBT) is the process of **generating test cases** against a **testing objective** by using **constraint solving techniques**

Developed in the context of both **code-based testing** and **model-based testing**

In France:

CEA - List (**Osmose** S. Bardin…)
(**GATEL** B. Marre…)
(**PathCrawler** N. Williams…)

Univ. of Nice Sophia-Antipolis (**CPBPV** M. Rueher, H. Collavizza, …)

INRIA - Celtique (**Euclide, JAUT** A. Gotlieb, …)

Abroad:

Microsoft Research (**DART, PEX, SAGE** P. Godefroid…)

Univ. of Madrid (**PET** E. Albert, G. Puebla, …)

Univ. of Stanford (**EXE** C. Cadar, …)

In the Industry:

Smartesting (**Test Designer** B. Legeard, …)

IBM Ilog Lab. (**Jsolver for ILOG Rules**, M. Leconte,..)

The automatic test data generation problem (1)

- Select either a path, branch, source code element, or testing criterion
- Generate a test input or a test set that covers the element
- Predict the expected outputs

f (int x_1*, int* x_2*, int* x_3*) {*

if($x_1 == x_2$ && $x_2 == x_3$*)*

if($x_3 == x_1 * x_2$*) ... }*

(i.e., reachability problem in infinite-state systems)

Solving this problem would have <u>broad industrial impact</u>:

- increase software quality and reliability through better code coverage and more systematic test inputs generation;

- decrease testing costs through augmented automation;

- automate conformance to Software Certification Standards as they require covering testing criteria (e.g., DO-178C, ISO 2626-1,...) ;

The automatic test data generation problem (2)

Given a location k in a program under test, generate a test input that reaches k

Reachability problem in infinite-state systems is undecidable in general!

Even when adding bounds, hard combinatorial problem

f (int x_1, int x_2, int x_3) {

if(x_1 == x_2 && x_2 ==x_3)

*if(x_3==x_1*x_2) ... }*

Using Random Testing,

Prob{ reack k} = 2 over $2^{32} \times 2^{32} \times 2^{32}$ = $\mathbf{2^{-95}}$ **= 0.00000…1.**

Constraint solving techniques are required!

- ✓ Loops (i.e., infinite-state systems) and infeasible paths
- ✓ Pointers, dynamic structures, higher-order computations (virtual calls)
- ✓ Floating-point computations, modular computations

The goal of the CAVERN project:

To improve ***Constraint-Based Testing*** with
Constraint Programming techniques
to effectively and efficiently address these problems

Illustrated with **a selected contribution**, in my today's talk:

Constraint-based program exploration for automatic test data generation

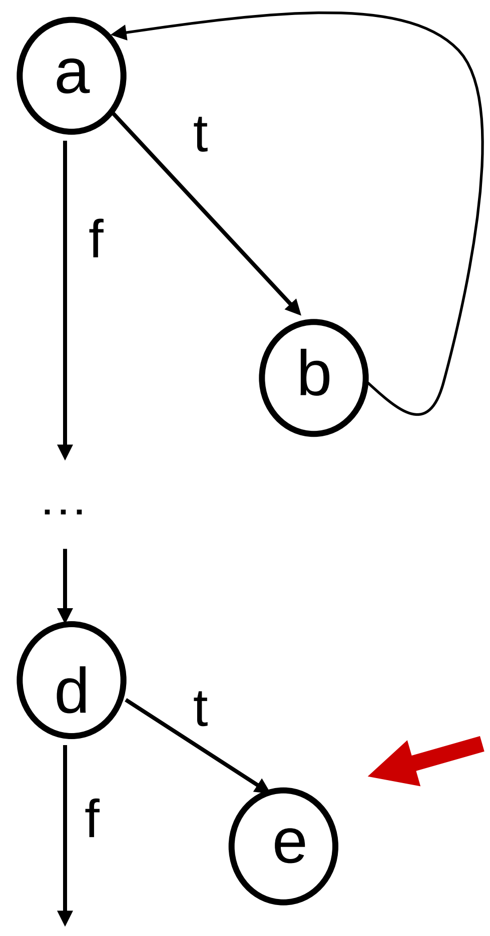
Outline

- Motivations of the CAVERN project
- Constraint-based program exploration for automatic test data generation
- Scientific results of the CAVERN project
- Achievment & Conclusions

A reacheability problem

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

       …
d.     if( j > 500)
e.         …
```

value of i to reach e ?



Path-oriented exploration

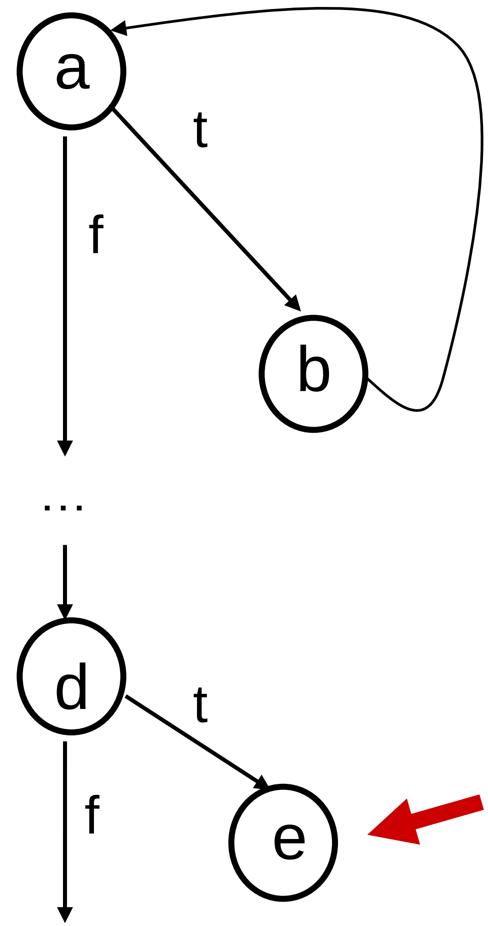
```
  f(  int i, …  )
   {
a.     j = 100;
       while( i > 1)
b.         { j++ ; i-- ;}

       …
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

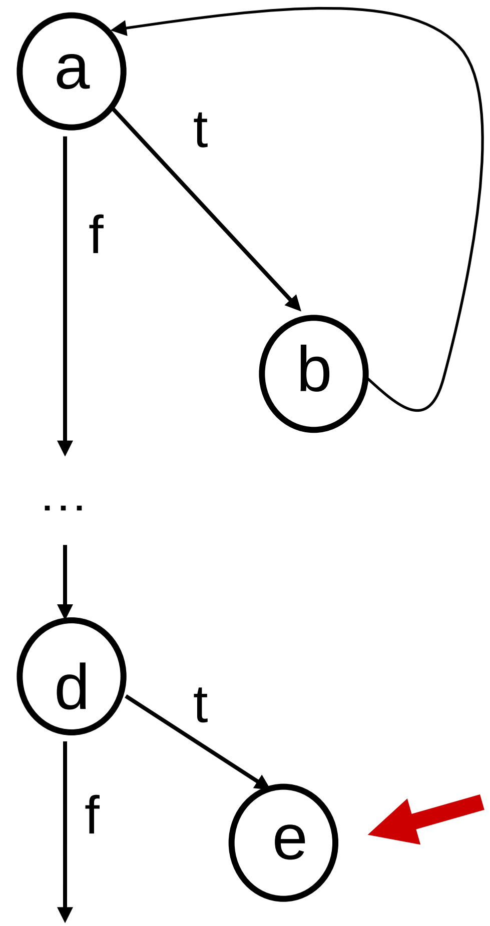
Backtrack !



Even without loops, #paths is exponential with #decisions

Constraint-based program exploration

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

       …
d.     if( j > 500)
e.         …
```



1. Constraint model generation

2. Control dependencies generation;

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

3. Constraint model solving

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

No backtrack !

Constraint-based program exploration
(Contribution of the CAVERN project)

- Based on a constraint model of the whole program (i.e., each statement is seen as a relation)

- Constraint reasoning over control structures

- Requires to build **dedicated constraint solvers**:

  * propagation queue management with priorities

  * specific propagators for meta-constraints

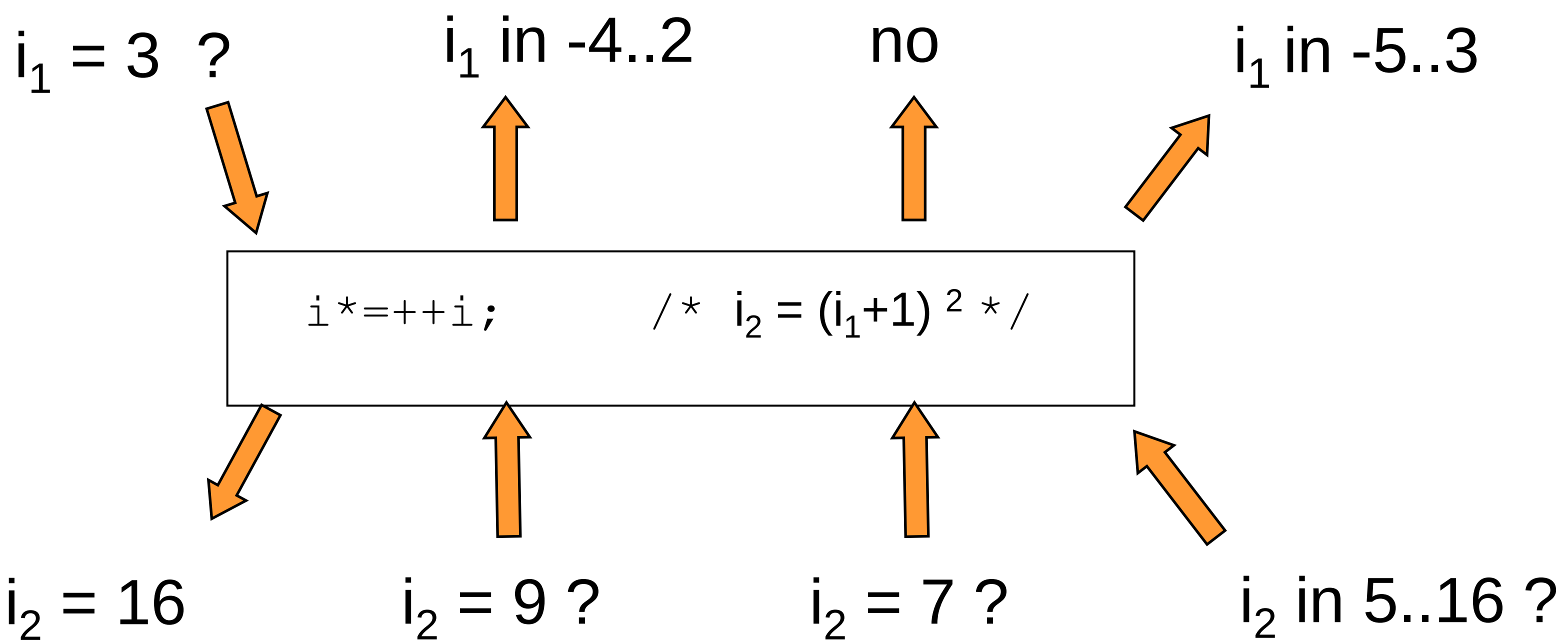
  * structure-aware labelling heuristics

Assignment as Constraint

Viewing an assignment as a relation requires to normalize expressions and rename variables (through single assignment languages, e.g. SSA)

`i*=++i ;` ⟶ $i_2 = (i_1+1)^2$

Using finite-domains **bound-consistency** filtering:



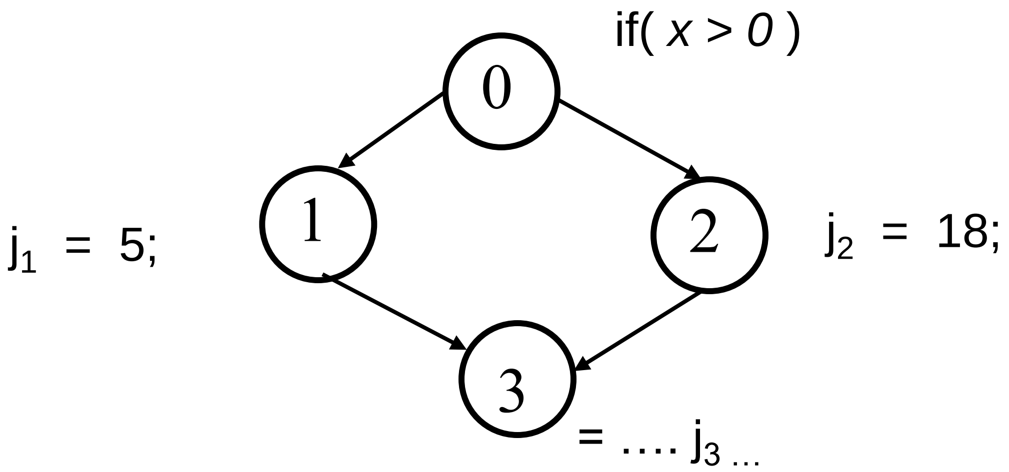
Statements as constraints

- ✓ 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:
  `v=A[i]` (or `p=Mem[&p]`) → variations of element/3
- ✓ Control structures: dedicated meta-constraints
  (interface, awakening conditions and filtering algorithms)

  Conditionnals (SSA) `if D then` C_1; `else` C_2 → ite

  Loops (SSA) `while D do C` → w

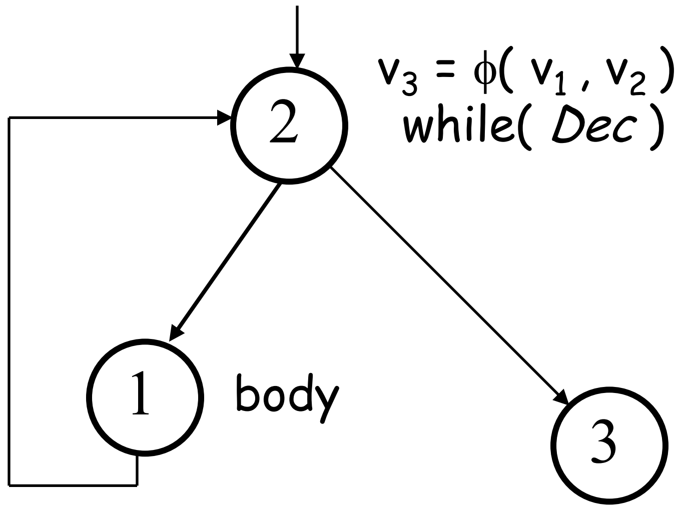
Conditional as meta-constraint: ite/6



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

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

Loop as meta-constraint: w/5



$w(Dec, V_1, V_2, V_3, body)$ iff

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

```
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 $401..2^{31}-1$

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 constraint-based exploration

- ✓ Special meta-constraints implementation for ite and w, memory accesses, function calls, and so on

By construction, w is unfolded only when necessary
but w may NOT terminate !
→ only a **semi-correct** test data generation procedure

- ✓ Join is implemented using ***Abstract Interpretation*** operators (e.g., interval-based union, weak-join operator, widening in **Euclide**)

- ✓ Special propagators based on linear-based relaxations
Using **Linear Programming over rationals** (i.e., Q_polyhedra)

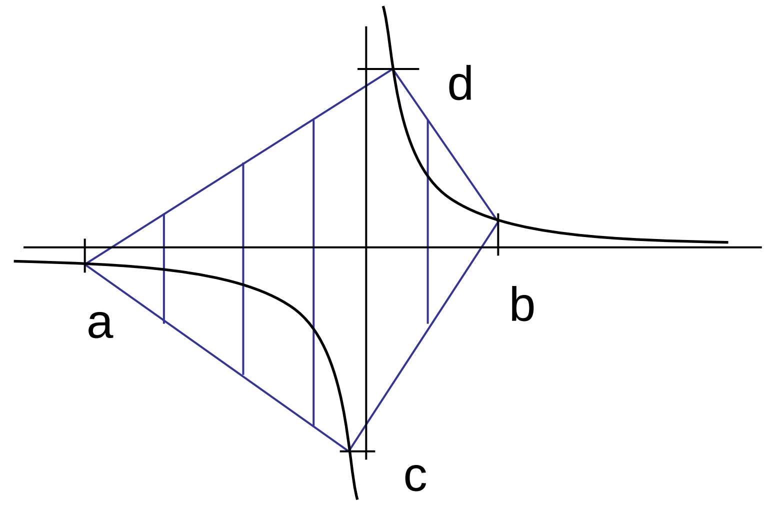
Abstraction-based relaxations

Abstraction-based relaxations

→ During constraint propagation, constraints can be relaxed in Abstract Domains (e.g., Q-Polyhedra, Octagons, …)

$Z = X * Y, \quad X \text{ in } a..b, Y \text{ in } c..d$

$$\Leftrightarrow \{\; Z - Ya - Xc + ac \geq 0,\; Xd - Z - ad + aY \geq 0,\; bY - bc - Z + Xc \geq 0,\; bd - bY - Xd + Z \geq 0,\; a \leq X \leq b,\; c \leq Y \leq d \}$$



→ To benefit from specialized algorithm (e.g., simplex for linear constraints) and capture global states of the constraint system

→ Require safe/correct over-approximation (to preserve property such as: *if the Q-Polyhedra is void then the constraint system is unsatisfiable*)

→ **Dynamic Linear Relaxation,** propagation queue with priorities

Constraint-based program exploration
(contribution of the CAVERN project – WP3)

Euclide: A Constraint-based testing platform for C **(Gotlieb ICST'09)**

Constraints for memory access/updates (i.e., load/store/new/delete)
(Charreteur Botella Gotlieb JSS'09)

Application on the TCAS case study
(Gotlieb KER Journal 2011)

TCAS

****

Prototype tool implementation:

Euclide (INRIA A. Gotlieb in 2009)

Scientific results of the CAVERN project

- Constraints over Memory Models (WP2)

  For object-oriented programs (Bytecode Java): Inheritance and virtual calls
  (Charreteur Gotlieb ISSRE'10)

  PhD Thesis of Florence Charreteur (Defense 9 Mar. 2010)
  Prototype tool **JAUT**

- Constraints over floating-point variables (WP4)

  - Filtering by ULP Max for addition/substraction **(Marre Michel CP'10),** for multiplication/division **(Carlier Gotlieb ICTAI'11)**

    Postdoc Matthieu Carlier
    Prototype tool for C floating-point computations = **FPSE**

  - Solving linear constraints over fp variables
    (Belaid Michel SCAN'11)

    PhD Thesis of Mohammed Said Belaid

Scientific results of the CAVERN project

- Constraints over modular integer variables (WP3)

  (Gotlieb Leconte Marre ModRef'10)
  Implantation in **GaTel** and **JSolver**

- Explanation-based generalization of infeasible paths in *Dynamic Symbolic Execution* (WP3)

  (Delahaye Botella Gotlieb ICST'10, TSE in revision)
  PhD Thesis of Mickael Delahaye (Defense 25 Oct. 2011)
  Prototype tool for C programs = **IPEG**

- Inferring loop invariants for Java programs (WP3)

  (Ponsini Collavizza Rueher ICSM'10)
  Postdoc of Olivier Ponsini

Achievments

- 7 publications involving more than 2 partners:
  3 Int. Journals, 3 Int. Conf., 1 Nat. Conf.

  INRIA-ILOG-CEA
  CEA-I3S
  CEA-INRIA

  More than 20 publications in total!

- 4 PhD among which 2 have already been completed

  (B. Berstel ILOG, M. Said Belaid I3S, F. Charreteur INRIA, M. Delahaye CEA)

  1 HDR

  12 months of post-docs

- Development of several prototype tools (Euclide, JAUT, Jsolver,….)

Conclusions
Constraint-Based Testing

- Emerging concept in code- and model-based automatic test data gener.
- Constraint Programming techniques offers:

  - Global constraints modelling to handle control and data structures (while pure SAT-solving does not work well in that context)

  - Versatility and flexibility of CP (while pure LP or SMT approaches are very rigid). Handles non-linear constraints over finite domains.

  - Generic techniques to implement new solvers, with abstraction-based relaxation, even if unsatisfiability detection has to be improved by combining techniques (e.g., SMT/CP)

- Mature tools (academic and industrial) already exist, but application on real-sized industrial cases still have to be demonstrated

- PhD students

  M. Said Belaid
  Bruno Berstel
  Florence Charreteur,
  Mickael Delahaye,

- Post-doc

  Matthieu Carlier, Olivier Ponsini

- Partners

  I3S: Michel Rueher, Claude Michel
  ILOG: Michel Leconte,
  CEA: Bernard Botella, Bruno Marre, Nicky Williams

Thank you!