Introduction
CP framework
DPVS
FM Application
Experiments
Discussion

A Dynamic Constraint-Based BMC Strategy For Generating Counterexamples

Michel Rueher[1]

Joined work with:

Hélène Collavizza[1], **Nguyen Le Vinh**[1], **Samuel Devulder**[2], **Thierry Gueguen**[2]

[1]University of Nice Sophia-Antipolis / I3S – CNRS, France
[2]Geensys / Dassault Systèmes, France

SAC-SVT, March 2011

This work was partially supported by the ANR-07-SESUR-003 project CAVERN and the ANR-07 TLOG 022 project TESTEC

Outline

Introduction

CP framework

DPVS, informal presentation

The Flasher Manager Application

Experiments

Discussion

Motivations

Introduction
Motivations
key points
CP framework
DPVS
FM Application
Experiments
Discussion

Formal proof methods that ensure the ***absence of all bugs*** are **too expensive**, or require manual efforts

→ **Automatic generation of counterexamples** violating a property on a limited model of the program is very useful

→ **Challenge**: finding bugs for **realistic time periods** for **real time applications**

Key points of DPVS

Introduction
Motivations
key points
CP framework
DPVS
FM Application
Experiments
Discussion

- **Context: constraint-programming framework** for bounded program verification
- A **new search strategy** for verifying a restricted class of Java or C programs:

  → **Non sequential dynamic exploration of the CFG**

Constraint-Programming Framework

Constraint-Programming Framework For Bounded Program Verification

- Falsification of a given property is checked for a given **bound**
- Mainly involves three steps:
  1. The **program is unwound k** times,
  2. An annotated and simplified **CFG** (Control Flow Graph) is built
  3. **Different exploration strategies** of the CFG
     - → **CPBPV: Depth first dynamic exploration** of the CFG
     - → **DPVS: Non-sequential exploration** of the CFG
- Generation of **counterexamples** (**= input values for a test cases)** violating a property on a limited model

Introduction
CP framework
Overview
CP vs BMC
Constraint store
Solvers
DPVS
FM Application
Experiments
Discussion

CP versus BMC

Bounded Model Checking

- Falsification of a given property is checked for a given **bound**
- Mainly involves three steps:
  1. The **program is unwound k** times,
  2. The unwound program and the property are translated into a propositional formula ϕ
     ϕ **is satisfiable iff there exists a counterexample of depth less than** k
  3. **A SAT-solver or SMT-solver** is used for checking the satisfiability of ϕ
- Generation of **execution traces** ($\neq$ input values for a test cases) violating a property on a limited model

Introduction
CP framework
Overview
CP vs BMC
Constraint store
Solvers
DPVS
FM Application
Experiments
Discussion

Building the constraint store: principle

Introduction
CP framework
Overview
CP vs BMC
Constraint store
Solvers
DPVS
FM Application
Experiments
Discussion

- Each **expression** is mapped to a **constraint**:
  ρ transforms program expressions into constraints
- SSA-like **variable renaming**: σ[**v**] is the current renaming of variable **v**
- JML :
  - **forall i** $\longrightarrow$ conjunction of conditions
  - **exist i** $\longrightarrow$ disjunction of conditions

  (**i** has bounded values)

Building the constraint store ...

- **scalar assignment**
  Program
  ```
  x=x+1; y=x*y; x=x+y;
  ```
  Constraints
  $\{x_1 = x_0 + 1, y_1 = x_1 * y_0, x_2 = x_1 * y_1\}$
- **array assignment**
  Program (a.length=8)
  ```
  a[i] = x;
  ```
  Constraints
  $\{a_1[i_0] = x_0, i_0 \neq 0 \rightarrow a_1[0] = a_0[0],$
  $i_0 \neq 1 \rightarrow a_1[1] = a_0[1], ..., i_0 \neq 7 \rightarrow a_1[7] = a_0[7]\}$

  guard $\rightarrow$ *body* is a **guarded constraint**
  $a[i] = x$ is the **element constraint**: i and x are constrained variables whose values may be unknown

Introduction
CP framework
Overview
CP vs BMC
Constraint store
Solvers
DPVS
FM Application
Experiments
Discussion

Solving the constraints

- **Dedicated solvers**
  - **ad-hoc simplifier** : trivial simplifications and calculus on constants
  - **linear solver** (LP algorithm) + **MIP solver**
  - **Boolean solver** (SAT solver)
    (Boolean relaxation of the **non linear** constraints)
  - **CSP solver** : used if none of the other solver did find an inconsistency

Introduction
CP framework
Overview
CP vs BMC
Constraint store
Solvers
DPVS
FM Application
Experiments
Discussion

Different exploration strategies

- **CPBPV: Depth first dynamic exploration** of the CFG

  → *Postcondition is used very late* because of the variables renaming

- **DPVS: Non-sequential exploration** of the CFG

  → *Starts from the postcondition and jumps to the locations where the variables are assigned*

Introduction
CP framework
DPVS
Strategy
Example
Pre-processing
Algorithm
FM Application
Experiments
Discussion

Non sequential dynamic constraint based exploration strategy

DPVS

Introduction

CP framework

DPVS

Strategy

Example

Pre-processing

Algorithm

FM Application

Experiments

Discussion

Why can we do it ?

Essential observation

When the program is in an SSA-like form, **a path can be built in a non-sequential dynamic way**

CFG does not have to be explored in a top down (or bottom up) way: compatible blocks can just be collected in a non-deterministic way

Non sequential dynamic constraint based exploration strategy

Introduction
CP framework
DPVS
Strategy
Example
Pre-processing
Algorithm
FM Application
Experiments
Discussion

Why does it pay off

- **DPVS starts from the post-condition** and dynamically collects program blocks which involve **variables of the post-condition**
- Collecting as much information as possible on a given variable

  → **enforces the constraints on its domain and reduces the search space**

- **Constraint solving is integrated with state exploration** to prune the state space as early as possible

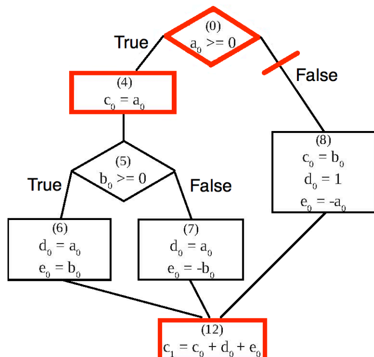
A small exemple


```
void foo(int a, int b)
int c, d, e, f;
if(a >= 0) {
      if(a < 10) {f = b - 1;}
      else {f = b - a; }
      c = a;
      if(b >= 0) {d = a; e = b;}
      else {d = a; e = -b;}  }
  else {
      c = b; d = 1; e = -a;
      if(a > b) {f = b + e + a;}
      else {f = e * a - b;}  }
  c = c + d + e;
  assert(c >= d + e); // property p1
  assert(f >= -b * e); // property p2
```


Introduction
CP framework
DPVS
Strategy
Example
Pre-processing
Algorithm
FM Application
Experiments
Discussion

A small exemple(continued)

Introduction
CP framework
DPVS
Strategy
Example
Pre-processing
Algorithm
FM Application
Experiments
Discussion




```
void foo(int a, int b)
int c, d, e, f;
if(a >= 0) {
      if(a < 10) {f = b - 1;}
      else {f = b - a; }
      c = a;
      if(b >= 0) {d = a; e = b;}
      else {d = a; e = -b;}  }
  else {
      c = b; d = 1; e = -a;
      if(a > b) {f = b + e + a;}
      else {f = e * a - b;}  }
  c = c + d + e;
  assert(c >= d + e); // property p1
  assert(f >= -b * e); // property p2
```

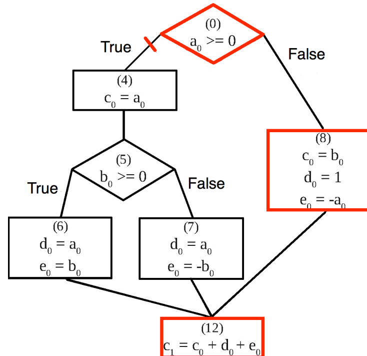
To prove property p_1, select node (12), then select node (4)

$\rightarrow$ the condition in node (0) must be true

$S = \{c_1 < d_0 + e_0 \wedge c_1 = c_0 + d_0 + e_0 \wedge c_0 = a_0 \wedge a_0 \geq 0\}$

$= \{a_0 < 0 \wedge a_0 \geq 0\}$ **... inconsistent**

A small exemple(continued)



Select node (8) $\rightarrow$ condition in node (0) must be false

$$S = \{c_1 < d_0 + e_0 \wedge c_1 = c_0 + d_0 + e_0 \wedge c_0 = b_0 \wedge a_0 < 0 \wedge d_0 = 1 \wedge e_0 = -a_0\}$$

$$= \{a_0 < 0 \wedge b_0 < 0\}$$

Solution $\{a_0 = -1, b_0 = -1\}$

Introduction
CP framework
DPVS
Strategy
Example
Pre-processing
Algorithm
FM Application
Experiments
Discussion

DPVS, pre-processing

Introduction
CP framework
DPVS
Strategy
Example
Pre-processing
Algorithm
FM Application
Experiments
Discussion

Pre-processing

1. P is **unwound k times** $\rightarrow \; P_{uw}$
2. $P_{uw} \rightarrow \; DSA_{P_{uw}}$, **Dynamic Single Assignment form** (each variable is assigned exactly once on each program path)
3. $DSA_{P_{uw}}$ is **simplified according to the specific property** *prop* by applying slicing techniques
4. Domains of all variables are filtered by **propagating constant values** along G, the simplified CFG

DPVS, Algorithm (scheme)

$S \leftarrow$ negation of *prop % constraint store*
$Q \leftarrow$ variables in *prop % queue of variables*

- IF $Q \neq \emptyset, v \leftarrow$ POP(Q)
  - **Search for a program block $PB(v)$ where v is defined**
    PUSH(Q, new_var), $new_var =$ new variables ($\neq$ input variables) of $PB(v)$
    $S \leftarrow S \cup$ {definition of v and conditions required to reach definition of v }
  - IF S **is inconsistent, backtrack & search another definition** (otherwise the dual condition is cut off)
- IF $Q = \emptyset$ search for an **instantiation of the input variables (= counterexample)**

  If no solution exists, DPVS backtracks.

FM Application: Description of the module

Introduction
CP framework
DPVS
FM Application
Description
Simulink model
Program
Experiments
Discussion

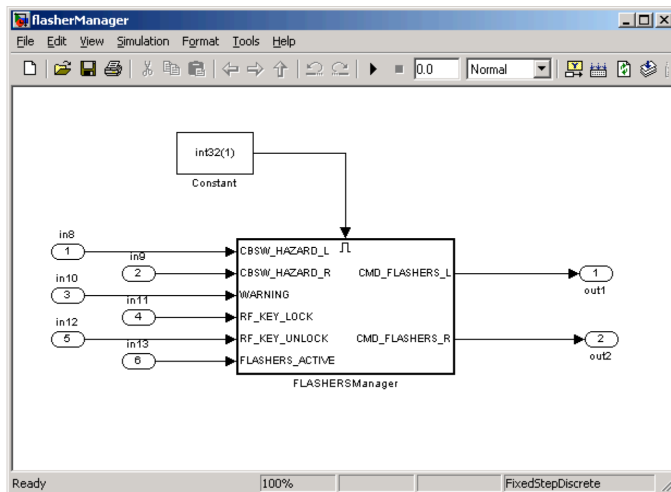
- **A real time industrial application** from a car manufacturer (provided by Geensoft)
- **Flasher Manager (FM)**: controller that drives several functions related to the flashing lights

  Purpose:

  - to indicate a direction change
  - to lock and unlock the car from the distance
  - to activate the warning lights
- **Simulink model** of FM $\rightarrow$ C function f_1

FM Application: Simulink model

Introduction
CP framework
DPVS
FM Application
Description
Simulink model
Program
Experiments
Discussion



FM Application: Function f_1

Introduction
CP framework
DPVS
FM Application
Description
Simulink model
Program
Experiments
Discussion

Simulink model of FM $\longrightarrow$ C function f_1

- 81 Boolean variables (6 inputs, 2 outputs) and 28 integer variables
- 300 **lines of code: nested conditionals including linear operations** and constant assignments

  Piece of code:

```
and1_a=((Switch5==TRUE)&&(TRUE!=Unit_Delay3_a_DSTATE));
if ((TRUE==((and1_a-Unit_Delay_c_DSTATE)!= 0))) {
   rtb_Switch_b=0;
}
else {
  add_a = (1+Unit_Delay1_b_DSTATE);
  rtb_Switch_b = add_a;
}
superior_a = (rtb_Switch_b>=3);
```

FM Application: property p_1

Introduction
CP framework
DPVS
FM Application
Description
Simulink model
Program
Experiments
Discussion

- Property p_1: ***The lights should never remain lit***

  Property p_1 concerns the behaviour of FM for an **infinite time period**

  $\rightarrow$ p_1 is violated when the lights remain on for ***N*** **consecutive time period**

  $\rightarrow$ a loop (bounded by N) that counts the number of times where the output of FM has consecutively been true

Challenge: bound N **as great as possible**

Experiments: tools

Introduction
CP framework
DPVS
FM Application
Experiments
Tools
Exp. on FM
Discussion

- **DPVS, implemented in Comet**, a hybrid optimization platform for solving combinatorial problems
- **CPBPV***, an optimized version of CPBPV based on a dynamic **top down strategy**
- **CBMC**, one of the best bounded model checkers

Experiments were performed on a Quad-core Intel Xeon X5460 3.16GHz clocked with 16Gb memory
All times are given in seconds.

Experiments (results)

Introduction
CP framework
DPVS
FM Application
Experiments
Tools
Exp. on FM
Discussion

Solving time:

N	CBMC	DPVS	CPBPV*
5	0.03	**0.02**	0.84
100	57.27	**1.95**	TO
200	232.19	**3.45**	TO
400	TO	**4.66**	TO

Pre-processing time:

N	CBMC	DPVS	CPBPV*
5	0.366	**0.480**	0.480
100	65.190	**9.750**	9.750
200	395.46	**21.65**	21.65
400	TO	**50.90**	50.90

Discussion

Introduction
CP framework
DPVS
FM Application
Experiments
Discussion
Exp. on BN
Future work

Experiments on the binary search

Length	CBMC	DPVS	CPBPV*
4	5.732	0.529	**0.107**
8	110.081	35.074	**0.298**
16	TO	TO	**1.149**
64	TO	TO	**27.714**
128	TO	TO	**153.646**

- **DPVS and CBMC waste a lot of time in exploring the different paths**
- **CPBPV* incrementally adds the decisions taken along a path**
  $\rightarrow$ well adapted for the *Binary Search* program

Discussion (continued)

Introduction
CP framework
DPVS
FM Application
Experiments
Discussion
Exp. on BN
Future work

Future work

- **Experiments on other applications**
- **Extension of our prototype**
  - → **handling pointers**
  - → **interfacing with a floating point number solver**
- **Combining strategies**
- **Using counter examples for errors localization**