CBEL (Constraint-Based Error Localization)

Michel Rueher[1]

University of Nice Sophia Antipolis – I3S / CNRS

[1] This work was started in collaboration with Prof. Hosobe during a two-months visit at NII in 2011

Outline

- **Problem**:
  - Informal presentation
  - Bounded Model Checking (recall)
  - Constraint-Based Error Localization: Formalization

- **Existing approaches**
  - Syntactic comparison of the paths
  - Derivation of "close" correct traces
  - MAX-SAT based approach

- **Capabilities of CP, LP, MIP**
  - IIS (irreducible Infeasibility set) for the linear constraint systems
  - Minimum conflict sets, nogoods (CSP,MIP)

Problem: informal presentation

- **Model checking**, testing

  Generation of counterexamples

  - Input data & wrong output (testing)
  - Input data & violated post condition / property

    → Execution trace

- **Problems**:
  - Execution trace: often *lengthy* and *difficult to understand*
  - Location of the portions of code that contain errors
    → *Very expensive*

Bounded Model Checking framework (recall)

- **Models** → finite automates, labelled transition systems
- **Properties**:
  - Safety → something bad should not happen
  - Liveness → something good should happen
- **Bound *k*** → look only for counter examples made of *k* states

Bounded Model Checking framework (recall cont.)

% set of states: S, initial states: I, transition relation: T

*% **bad states B reachable from I via T?***

$\text{bounded\_model\_checker}_{\text{forward}}(I,T,B,k)$

$SC = \varnothing$; SN =I; n=1

while $S_C \neq S_N$ **and** n<k **do**

If $B \cap S_N \neq \varnothing$ **then return** "*found error trace to bad states*";

else $S_C = S_N$;

$S_N = S_C \cup T(SC)$;

$n = n + 1$;

done

return "*no bad state reachable*";

SAT/SMT- based BMC: Bounded Model Checking

1. The **program is unwound *k*** times

2. The unwound (and simplified) program and the property are translated into

   a big 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

CP-based Bounded Program Verification

1. The **program is unwound *k*** times,

2. An **annotated** and **simplified CFG** is built

3. Program is translated in constraints **on the fly**

   → A **list of solvers** tried in sequence (LP, MILP, Boolean, CP)

Constraint-Based Error Localization: Formalization

- **P**: program
- **Post_P**: post condition of P
- **Pre_P**: precondition of P
- **CST_P**: constraints of faulty path of P (Input data provided by Model checker)

→ **Pred_P** ∧ **CST_P** ∧ ¬ **Post_P** holds

→ **Pred_P** ∧ **CST_P** ∧ **Post_P** fails

Problem: to finding "smallest" subsets of **Pred_P** ∧ **CST_P** ∧ **Post_P** that are inconsistent

Example

Program:

```
% Input : int input1, int input2
int x = 1, y = 1, z = 1;
if (input1 > 0) {x += 5; y += 6; z += 1; }
if (input2 > 0) {x += 6; y += 5; z += 4; }
% Post-condition: x < 10 ∧ y < 10
```

Counterexample: *input1=1, input2=1*

CSP P:

input1=1, input2=1, $x_{10} = 1$, $y_{10} = 1$,

$z_{10} = 1$, $x_{11} = 6$, $y_{11} = 7$, $z_{11} = 2$, $x_{12} = x_{11}$, $y_{12} = y_{11}$, $z_{12} = z_{11}$,

$x_{13} = x_{12}+6$, $y_{13} = y_{12}+5$, $z_{13} = z_{12}+4$, $x_{14} = x_{13}$, $y_{14} = y_{13}$, $z_{14} = z_{13}$,

$x_{14} < 10$, $y_{14} < 10$

Example (cont.)

CS_P can be divided into 3 sub-CSPs (computations for x, y, and z are independent)

sub CSP_x is: $x_{10} = 1$, $x_{11} = 6$, $x_{12} = x_{11}$, $x_{13} = x_{12}+6$, $x_{14} = x_{13}$, $x_{14} < 10$

sub CSP_y is: $y_{10} = 1$, $y_{11} = 7$, $y_{12} = y_{11}$, $y_{13} = y_{12}+5$, $y_{14} = y_{13}$, $y_{14} < 10$

Smallest inconsistent CSP for x: ~~$x_{10} = 1$~~, $x_{11} = 6$, $x_{12} = x_{11}$, $x_{13} = x_{12}+6$, $x_{14} = x_{13}$, $x_{14} < 10$

Smallest inconsistent CSP, for y: ~~$y_{10} = 1$~~, $y_{11} = 7$, $y_{12} = y_{11}$, $y_{13} = y_{12}+5$, $y_{14} = y_{13}$, $y_{14} < 10$

Existing approaches

- **Syntactic comparison of the paths** [BNR03]:
  - Multiple calls to a model checker and comparison of the counter examples to a ***successful*** execution trace
  - Transitions that do not appear in a correct trace are reported as a possible cause of fault

Existing approaches (cont.)

- **Derivation of "close" correct traces**
  (distance metric on executions of P)
  Explain [GKL04,GCK06]):

  1) Calls CBMC for finding a faulty execution P

  2) Uses a pseudo-Boolean solver to identify the closest correct run

  3) Computes the difference between the two traces.

  → S, a Boolean formula associated to P but with assignments that do not violate the specification

  → Extends S with constraints representing an optimization problem: find a satisfying assignment as similar as possible to the counterexample (use a distance metric on executions of P)

  [ReN03]: based on testing rather on model checking (use correct and faulty test runs)

Existing approaches (cont.)

- **Derivation of "close" correct traces** (cont.)

  (Predicate switching)

  - [GBC06, GSB07] start from the specification to derive a correct program with the same input data.
    → identifies a superset of erroneous instructions
    → the process is restarted for different counterexamples

  - [ZGG06, LiL10] → for faults in the control predicate and right-hand sides of assignments

Existing approaches (cont.)

MAX-SAT based appoach [MaM11]
(implemented in Bug-Assist with CBMC)

1. Encoding a trace of a program as a Boolean formula F that is satisfiable iff the trace is satisfiable

2. Building a failing formula F' by asserting that the post condition must hold

3. Computing with MAX-SAT the maximum number of clauses that can be satisfied in F'
→ *complement as a potential cause of the errors*

Capabilities of CP, LP, MIP

- No Boolean abstraction (or bit vector encoding) required to capture the semantics of the constraints

  → ***Generalisation of MAX-SAT approach***

- A lot of work on **Minimum conflict sets** [PBR01, PBR03, Jun04, Ach07], **IIS** (irreducible Infeasibility set) [Chin97,01], Nogoods,…

IIS - Definitions

An ***irreducible inconsistent system (IIS)*** is an infeasible set of constraints that does not contain any constraints which do not contribute to the infeasibility

A ***cluster of IISs*** is a maximal set of IISs constructed from a single IIS by iteratively adding all other IISs that overlap at least one other IIS in the set

IIS – Problems and challenges

Useful IISs:

→ ***Minimizing the numbers of rows*** (functional constraints) because column constraints (bounds on variables) are easier to understand

Problems:

- The number of IISs in an infeasible LP can be ***exponential*** in the worst case
- Quickest algorithms for finding IISs often return IISs having many rows

IIS – Algorithms

The Deletion Filter:

```
% Input : an infeasible set of linear constraints

FOR each constraint in the set:

  1.   Temporarily drop the constraint from the set
  2.   Test the feasibility of the reduced set

       IF feasible THEN return dropped constraint to the set

       ELSE % infeasible

             Drop the constraint permanently

END FOR

OUTPUT : constraints constituting a single IIS
```

IIS – Algorithms (cont.)

The Deletion Filter:

- **not so slow** in practice as might be expected
- **Order** in which the constraints are tested affects the IISs which is found

Example:

Considers constraints set {A,B,C,D,E,F} with two IISs: {A,B} and {C,E,F}

Order from A to F $\rightarrow$ {C,E,F}

Order from F to A $\rightarrow$ {A,B}

IIS – Algorithms (cont.)

Sensitivity filter: any non-basic variable having a reduced cost or shadow price in the final basis of zero can be deleted

The Deletion / Sensitivity Filter:

```
% Input : an infeasible set of linear constraints

FOR each constraint in the set:

  1.    Temporarily drop the constraint from the set
  2.    Test the feasibility of the reduced set

        IF feasible THEN return dropped constraint to the set

        ELSE (% infeasible)   Drop the constraint permanently

                              Apply the sensitivity filter

END FOR

OUTPUT : constraints constituting a single IIS
```

IIS – Algorithms (cont.)

Use the concept of "elastic programming": ***non-negative "elastic variables"*** are added to the constraints to provide elasticity

Non-elastic constraint	Elastic constraint
$\Sigma_j\ a_{ij}x_i \geq b_t$	$\Sigma_j\ a_{ij}x_i + \mathbf{e_t} \geq b_t$
$\Sigma_j\ a_{ij}x_i \leq b_t$	$\Sigma_j\ a_{ij}x_i - \mathbf{e_t} \leq b_t$
$\Sigma_j\ a_{ij}x_i = b_t$	$\Sigma_j\ a_{ij}x_i + \mathbf{e'_t} - \mathbf{e''_t} = b_t$

IIS – Algorithms (cont.)

The Elastic Filter:

```
% Input : an infeasible set of linear constraints

    1.  Make all constraints elastic by adding non-negative elastic
        variables
    2.  Solve LP using elastic objective function

        IF feasible THEN   enforce the constraints with any e_t > 0 by

                           permanently removing their elastic variable(s)

                           GO TO step 2

        ELSE (% infeasible)   EXIt

END FOR

OUTPUT : the set of enforced constraints contains at least one IIS
```

Minimum Conflict Sets in CSP

Problems:

- The number conflict sets in an infeasible CSP can be ***exponential*** in the worst case

- ***Testing the consistency of CSP can be very costly***

→ **AC- Minimum Conflict Sets**

Consider CSP P ={x≠y, y≠z, z≠x} with x, y, z ∈ {a,b}

P is a *Minimum Conflict Set but not an AC- Minimum Conflict Set*

Minimum Conflict Sets in CSP (cont.)

Algorithm of [PBR01, PBR03]

Example:

- Consider the CSP = {C_1,C_2,C_3,C_4,C_5,C_6,C_7,C_8}
- Assume that the constraints are posted in the order $C_1 \rightarrow C_8$ and that the CP solver detects an inconsistency when C_8 is added
- Restart the process by posting the constraints in the order C_8,C_1,C_2,C_3,C_4,C_5,C_6,C_7
- Assume that the CP solver detects an inconsistency when C_5 is added, when restart the process by posting the constraints in the order C_5,C_8,C_1,C_2,C_3,C_4
- If a fail occurs when C_3 is added, we have to restart with C_3, C_5, C_8, C_1, C_2 Otherwise, we are done

Optimized in [Jun04] (gain of a *log* factor)

TO DO

1. **Evaluate** IISs algorithms and conflicts set algorithms on **real benchmarks**

2. Evaluate various potential helps in the constraint framework

   a. Computing some *additional counter examples* and identify constraints that occur in numerous conflict set (or clusters of IISs) to narrow the set of suspect locations
   b. Searching a *"slightly" different paths* from the faulty path but which provides a correct answer
   c. Computing the *upper and lower bound for each input variable* of a faulty path
   d. Try to prove that the program is partially correct with some *additional preconditions* provided by the user
   Goal: to demonstrate that the result always violates the post-condition with some restrictions on the input data

References

- [Ach07] Tobias Achterberg: Conflict analysis in mixed integer programming. Discrete Optimization 4(1): 4-20 (2007).
- [BNR03] Thomas Ball, Mayur Naik, and Sriram K. Rajamani. From Symptom to Cause: Localizing Errors in Counterexample Traces. Proc. POPL 2003. ACM Press, pp. 97-105.
- [Chin96] John W. Chinneck: An Effective Polynomial-Time Heuristic for the Minimum-Cardinality IIS Set-Covering Problem. Ann. Math. Artif. Intell. 17(1-2): 127-144 (1996).
- [Chin01] John W. Chinneck: Fast Heuristics for the Maximum Feasible Subsystem Problem. INFORMS Journal on Computing 13(3): 210-223 (2001)
- [CRH10] Hélène Collavizza, Michel Rueher, Pascal Van Hentenryck: CPBPV: a constraint-programming framework for bounded program verification. Constraints 15(2): 238-264 (2010)
- [CVR11] Hélène Collavizza, Nguyen Le Vinh, Michel Rueher, Samuel Devulder, Thierry Gueguen. Efficient Constraint-Based Dynamic Strategies For Generating Counterexamples. 26th ACM Symposium On Applied Computing (SAC 2011), Software Verification and Testing Track.
- [LiL10] Yongmei Liu, Bing Li: Automated Program Debugging Via Multiple Predicate Switching. Proc. AAAI 2010, AAAI Press.
- [GBC06] Andreas Griesmayer, Roderick Bloem, and Byron Cook. Repair of Boolean Programs with an Application to C. Proc of CAV'06 LNCS 4144, pp. 358-371.
- [GCK06] Alex Groce, Sagar Chaki, Daniel Kroening , and Ofer Strichman. Error explanation with distance metrics. International Journal on Software Tools for Technology (2006) 8(3): 229–247

- [GKL04] Alex Groce, Daniel Kroening, and Flavio Lerda. Understanding Counterexamples with explain. Proc. of CAV 2004, LNCS 3114, pp. 453–456, 2004.
- [Jun04] Ulrich Junker: QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. Proc. of AAAI 2004. Pp. 167-172
- [GSB07] Andreas Griesmayer, Stefan Staber, and Roderick Bloem. Automated Fault Localization for C Programs. Electronic Notes in Theoretical Computer Science 174 (2007) 95–111.
- [MaM11] Manu Jose, Rupak Majumdar. Cause Clue Clauses: Error Localization using Maximum Satisfiability. Proc. of PDLI 11 (32nd ACM SIGPLAN conference on Programming Language Design and Implementation).
- [Nak10] Shin Nakajima . Semi-automated diagnosis of FODA feature diagram. Proceedings of the 2010 ACM Symposium on Applied Computing (SAC '10), pp. 2191-2197.
- [PBR03] T. Petit, C. Bessière, and J-C Régin . A General Conflict-Set Based Framework for Partial Constraint Satisfaction. CP'03, proceedings workshop on Soft Constraints, Kinsale, Ireland, 2003
- [PBR01] T. Petit, J-C. Régin, and C. Bessière . Specific Filtering Algorithms for Over-Constrained Problems. CP'01, LNCS, Chyprus, pp 451--463, 2001
- [ReR03] Manos Renieris, Steven P. Reiss: Fault Localization With Nearest Neighbor Queries. Proc of 18th IEEE International Conference on Automated Software Engineering (ASE 2003), pp. 30-39.
- [SQL05] ShengYu Shen, Ying Qin, Sikun Li: Minimizing Counterexample with Unit Core Extraction and Incremental SAT. Proc of VMCAI 2005, LNCS 3385, pp. 298-312
- [ZGG06]Xiangyu Zhang, Neelam Gupta, Rajiv Gupta: Locating faults through automated predicate switching. Proc. ICSE 2006, ACM press, pp. 272-281