**Motivations**

- **The CAVERN project** (ANR, 2008-2010, part: ILOG, INRIA, CEA, U. of Nice):
  To explore the capabilities of Constraint Programming for Automated Program Testing and Verification
- We build a unified framework (called Euclide) to perform:
  - test case generation for structural coverage
  - counter-example generation to safety properties
  - (partial) program proving for safety-critical C programs
- TCAS (Traffic Anti-Collision Avoidance System) software is a real-life example of safety-critical embedded system.
  Strong requirements in terms of verification.

**Agenda**

- Motivations
- TCAS software verification
- A Constraint Programming approach
- Experimental results
- Further work

**Traffic alert and Collision Avoidance System**

Embedded system on commercial aircrafts
Publicly available implementation for Test and Evaluation (from the Siemens suite)
[http://sir.unl.edu/portal/](http://sir.unl.edu/portal/)
DO-178 Level B
Statement and decision coverage is mandatory
Tcas.c: component that issues Traffic Advisory and Resolution Advisory (170 LOC)

The alt_sep_test function

-14 global variables
Own_Track_Alt
Other_Track_Alt
Up_Separation
Down_Separation
Positive_RA_Alt_Tresh...

- Calls 9 distinct functions
- Nested conditionals, logical operators, refinements...
But, no loops, no fp, no pointers

Safety properties in ACSL

P1a /*@ behavior P1a : assumes Up_Separation ≥ Positive_RA_Alt_Tresh && Down_Separation ≤ Positive_RA_Alt_Tresh; ensures result != need_Downward_RA; */
P1b /*@ behavior P1b : assumes Up_Separation < Positive_RA_Alt_Tresh && Down_Separation < Positive_RA_Alt_Tresh; ensures result != need_Downward_RA; P1b */
P3a /*@ behavior P3a : assumes Up_Separation < Positive_RA_Alt_Tresh && Down_Separation < Positive_RA_Alt_Tresh && Own_Tracked_Alt > Other_Tracked_Alt; ensures result != need_Downward_RA; */
P3b /*@ behavior P3b : assumes Up_Separation < Positive_RA_Alt_Tresh && Down_Separation < Positive_RA_Alt_Tresh && Own_Tracked_Alt > Other_Tracked_Alt; ensures result != need_Downward_RA; */
P3c /*@ behavior P3c : assumes Up_Separation < Positive_RA_Alt_Tresh && Down_Separation < Positive_RA_Alt_Tresh && Own_Tracked_Alt > Other_Tracked_Alt; ensures result != need_Downward_RA; */

A bit of anti-collision Theory
Safety properties:
P1: Safe advisory selection
P2: Best advisory selection
P3: Avoid unnecessary crossing
P4: No crossing advisory selection
P5: Optimal advisory selection (subsumes P1 and P2)

The alt_sep_test function

int alt_sep_test()
{
    bool enabled, tcas_equipped, intent_not_known;
    bool need_upward_RA, need_downward_RA;
    int alt_sep;
    enabled = High_Confidence && (Own_Tracked_Alt <= 600) && (Cur_Vertical_Sep > 600);
    tcas_equipped = Other_Capability == 1;
    intent_not_known = Two_of_Three_Reports_Valid && Other_RAC == 0;
    alt_sep = 0;
    if (enabled && ((tcas_equipped && intent_not_known) || !tcas_equipped))
    {
        need_upward_RA = Non_Crossing_Biased_Climb() && Own_Below_Threat();
        need_downward_RA = Non_Crossing_Biased_Descend() && Own_Above_Threat();
        if (need_upward_RA && need_downward_RA)
            alt_sep = 0;
        else if (need_upward_RA)
            alt_sep = 1;
        else if (need_downward_RA)
            alt_sep = 2;
        else
            alt_sep = 0;
    }
    return alt_sep;
}
How prove these properties?

• In practice, manual code review and testing
• In Theory:
  - Hoare Logic (weakest precondition computations)
  - Model checking
  - Abstract Interpretation-based static analysis
  - Constraint-based verification and testing

Agenda

• Motivations
• TCAS software verification
• A Constraint Programming approach
• Experimental results
• Further work

Our CP approach

Constraint generation

Translate the each C function into a constraint program \( P \)
Translate the property into a constraint \( S \)

Constraint solving

Try to prove that \( \text{sol}( P \land \lnot S) = \emptyset \)

A constraint model of C functions

Viewing an assignment statement as a relation requires to rename the variables

\[
i++; \quad i_2 = i_1 + 1
\]

⇒ Static Single Assignment (SSA) form (Cytron et al. TOPLAS 1991)
or single assignment language
SSA form

Each use of a variable refers to a single definition

\begin{align*}
x &= x + y; \\
y &= x - y; \\
x &= x - y;
\end{align*}

\begin{align*}
\text{if}(\ldots) & \quad \text{if}(\ldots) \\
& \quad i \ldots; \\
& \quad \text{else} \\
& \quad i \ldots; \\
& \quad i = \ldots; \\
& \quad i = \ldots;
\end{align*}

Constraint Program

Variable declaration

unsigned short i;

Assignment, decision

Arithmetical constraints ($x, y$)

\begin{align*}
j_x &= j_x \cdot i \\
i &< j
\end{align*}

Conditionnal (SSA)

\begin{align*}
\text{if} \; d \; \text{then} \; c_1, \; \text{else} \; c_2, \; \phi(v_1, v_2)
\end{align*}

Function call (SSA)

\begin{align*}
r &= f(a); \\
\text{SP\_CALL}(f, A, G_1, \ldots, G_n, R)
\end{align*}

Translating a property into constraints

```c
/*@ behavior P1b : 
assumes
Up\_Separation < Positive_RA\_Alt\_Tresh \&
Down\_Separation \geq Positive_RA\_Alt\_Tresh; 
ensures \result \neq need\_Upward\_RA;
---------------------------------------------------------------
S = \text{Preconditions} \implies \text{Post-relations}
Then, \; \neg S = \text{Preconditions} \wedge \neg \text{Post-relations}

Up\_Separation < Positive_RA\_Alt\_Tresh \wedge
Down\_Separation \geq Positive_RA\_Alt\_Tresh \wedge
R = need\_Upward\_RA
```

Constraint solving

\begin{align*}
sol( P \wedge \neg S) ?
\end{align*}

- $P \wedge \neg S$ is a non-linear FD constraint problem with global constraints
- We develop our own constraint solver based on:
  - Constraint propagation + bound-consistency filtering
  - Linear Programming techniques over $Q$
- Why $LP$?: capturing linear global behaviour
- Why $Q$?: preserving correctness is essential for program verification!
- Property: if the LP relaxed problem does not contain integer points then the original problem is unsatisfiable (but, the converse is false!)
- Synchronous cooperation of constraint propagation and simplex over $Q$ through the usage of Dynamic Linear Relaxations
Non-linear expressions in tcas.c

- Multiplication
- Logical operations \((z > x+y \text{ || } z < x+y-3)\)
- Reification \((z = x > y)\)
- Conditionals \((\text{if then else})\)

→ Dynamic Linear Relaxations (DLRs)

DLR of multiplication [McCormick 76]

\[
\begin{align*}
Z = X \cdot Y, \quad & X \text{ in } a..b, \ Y \text{ in } c..d, \ Z \text{ in } e..f \\
\{ & Z - Ya - Xc + ac \geq 0, \\
& Xd - Z - ad - aY \geq 0, \\
& bY - bc - Z + Xc \geq 0, \\
& bd - bY - Xd + Z \geq 0, \\
& a \leq X \leq b, \\
& c \leq Y \leq d, \\
& e \leq Z \leq f \\
\}
\end{align*}
\]

A consequence of \((X - a)(Y - c) \geq 0\)

\((X - a)(d - Y) \geq 0\)

...

DLR of reification

- Reification associates a boolean var. to an expression
- \(Z = (X \leq Y) \quad \text{where } X \text{ in } a..b, \ Y \text{ in } c..d \quad \text{and } Z \text{ in } 0..1\)

\[
\begin{align*}
(1 - (X - Y) - (1 - a - d))Z \leq 0, \quad & (X - Y) - (b - c)(1 - Z) \leq 0 \\
\text{Min}(F(X,Y)) \leq & F(X,Y) \leq \text{Max}(F(X,Y)) \\
Z = (F(X,Y) \leq 0)
\end{align*}
\]

DLR of ITE( Dec, C1, C2)

- Global constraint \(\rightarrow\) is considered iteratively in the constraint store
- Variables of the relation = input-output variables of the conditional
- Awakened when a bound of at least one variable has been pruned
- Filtering algorithm (performed when awakened):

\[
\begin{align*}
\text{if } & \text{post(Dec } \land C1) \text{ fails} \\
& \text{then DLR}(-\text{Dec } \land C1) \text{ and remove ITE} \\
\text{else if } & \text{post(Dec } \land C1) \text{ fails} \\
& \text{then DLR}(-\text{Dec } \land C2) \text{ and remove ITE} \\
& \text{else join_dom(Dom1, Dom2) and join_poly(Qpoly1, Qpoly2)}
\end{align*}
\]
How to implement \( \text{join\_poly}(QPoly_1, QPoly_2) \) with a linear solver?

- Convex hull computation [Benoy, King, Mesnard TPLP 2004]
- Big-M relaxation + projection
- Simplex-based weak_join operator (from the Abstract Interpretation community) [Sankaranarayanan et al. VMCAI’06]

**NB1:** All these computations are exponential in the number of dimensions in the worst case

**NB2:** switching to the so-called polyhedra « generator representation » is prohibitive in our context

**Weak_join operator**

The disjunction:

\[
\left\{ g_1^i(x) \geq c_1^i \right\}_{k=1} \lor \left\{ g_2^i(x) \geq c_2^i \right\}_{k=1} \\
\]

\[ x = (x_1, \ldots, x_n), \text{ where } x \in \mathbb{Z} \]

\[
\text{Weak}_i : \begin{array}{l}
\alpha_1 = \text{Minimize } g_1^i(x) \text{ subject to } \left\{ g_1^i(x) \right\}_{k=1} \\
\alpha_d = \text{Minimize } g_1^{\text{int}(d)}(x) \text{ subject to } \left\{ g_1^i(x) \right\}_{k=1} \\
\alpha_{d+1} = \text{Minimize } g_1^{\text{int}(d+1)}(x) \text{ subject to } \left\{ g_1^i(x) \right\}_{k=1} \\
\ldots \\
\alpha_{2^d} = \text{Minimize } g_1^{\text{int}(2^d)}(x) \text{ subject to } \left\{ g_1^i(x) \right\}_{k=1} \\
g_1^i(x) \geq \text{Min}(\alpha_i, c_1^i), \\
\ldots \\
g_2^{\text{int}(2^d)}(x) \geq \text{Min}(\alpha_{2^d}, c_2^{\text{int}(2^d)})
\end{array}
\]
Weak_join operator

- Doesn’t require any Fourier’s Elimination step ! → Very good running time on tcas.c, acceptable loss of precision
- But, doesn’t commute with Join_dom
- Doesn’t « discover » new linear relations among the two disjuncts

Agenda

• Motivations
• TCAS software verification
• A Constraint Programming approach
• Experimental results
• Further work
Euclide’s architecture

C program
Preprocessed file
Normalized code
Points-to analysis
SSA form

Implemented in SICStus Prolog, SSA form generated by a single-pass algorithm [Brandis & Mössenbock 94], clpfd and clpq libraries.

- Use of the clpfd library for Constraint Propagation over Finite Domains
- Use of the clpq library for Linear Programming over Q

Euclide Program

Negated property
-test data
- fail
- ?

First experimental results

<table>
<thead>
<tr>
<th>Test Results</th>
<th>Time</th>
<th>Mem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1a</td>
<td>0.7s</td>
<td>4.85Mo</td>
</tr>
<tr>
<td>P1b</td>
<td>0.7s</td>
<td>4.85Mo</td>
</tr>
<tr>
<td>P2a</td>
<td>0.6s</td>
<td>4.85Mo</td>
</tr>
<tr>
<td>P2b</td>
<td>0.6s</td>
<td>4.85Mo</td>
</tr>
<tr>
<td>P3a</td>
<td>5.4s</td>
<td>6.35Mo</td>
</tr>
<tr>
<td>P3b</td>
<td>5.4s</td>
<td>6.35Mo</td>
</tr>
<tr>
<td>P4a</td>
<td>6.8s</td>
<td>6.35Mo</td>
</tr>
<tr>
<td>P4b</td>
<td>6.8s</td>
<td>6.35Mo</td>
</tr>
<tr>
<td>P5a</td>
<td>2.7s</td>
<td>5.95Mo</td>
</tr>
<tr>
<td>P5b</td>
<td>2.7s</td>
<td>5.95Mo</td>
</tr>
<tr>
<td>P6a</td>
<td>0.6s</td>
<td>4.85Mo</td>
</tr>
<tr>
<td>P6b</td>
<td>0.6s</td>
<td>4.85Mo</td>
</tr>
</tbody>
</table>

Intel Core Duo 2.4GHz clocked PC with 2Go of RAM

A simple collaboration principle

Euclide

Post: \( \min \leq X, X \leq \max \)
Post: \( \text{relax}(C) \)
Post: \( C \)

Maintains coherence
throughs DLRs

Simplex calls
cutting planes
Propagation/Search
+ alarms

Solved form of the polyhedron

Solution, fail or timeout

And in the Literature !!!

Table 2: Results on the verification of safety properties

Author said:
"I think that your analysis of P3A is right.
Recently we have redone the TCAS experiment for a workshop paper (attached for your reference) with a different symbolic executor and we found an error in that property too. I did not check your output in detail, but I guess that you bumped in the same error. "

Fig. Extracted from « Using Symbolic Execution for Verifying Safety-Critical Systems » ESEC-FSE 2001
The bugs were previously unreported while the other three had been detected by the developers and fixed.

Finally, we also used MAGIC on the source code of the Air Traffic Collision Avoidance System (TCAS). The source code was obtained from the heavens suite, a standard set of benchmark used by the testing and formal verification community. The benchmarks provide both correct and buggy implementations of TCAS. Using assertions, several standard safety properties [98, 77] are isolated in the implementation. These safety properties refer to upward/downward resolution abnormalities, i.e., instructions for an airplane to increase or decrease to altitude. MAGIC successfully detected assertion violations in the buggy implementation and also proved that the correct implementation is safe.

### Agenda

- **Motivations**
- **TCAS software verification**
- **A Constraint Programming approach**
- **Experimental results**
- **Further work**

### Further work

- Improving our weak_join implementation
  - Removing spurious equalities
    - `tmp = ...` → `... = tmp + ...`
    - Adds a dimension to the polyhedron!
  - Replacing SICStus clpQ library by a verified LP solver (Osqp_ex for example [Applegate et al. OR Letters 2007])

- An efficient global constraint for function calls:
  - Abstracting the relations due to function calls (replace the constraints of the callee by a polyhedral abstraction)

- Deal with modular integer computations

---

**Fig. Extracted from « Modular Verification of Software Components in C »**

---

**Thanks you!**