Assumptio and Stuff: using Z3 in a collaborative parallel formal verification framework.

Adrien Champion
adrien.champion@onera.fr

November 2011

Table of Contents

ONERA THE FRENCH AEROSPACE LAB

Rockwell Collins

Outline

ONERA THE FRENCH AEROSPACE LAB

Rockwell Collins

Verification of Embedded Critical Code

Context

- Main goal: formal reachability analysis of critical embedded systems;
- Synchronous languages, typically Lustre;

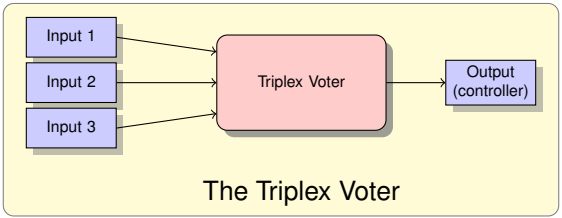
Verification of Embedded Critical Code

Context

- Main goal: formal reachability analysis of critical embedded systems;
- Synchronous languages, typically Lustre;
- Supervisors: Rémi Delmas[†], Michael Dierkes[‡], Pierre-Loïc Garoche[†] and Virginie Wiels[†] .

[†] Onera, The French Aerospace Lab
[‡] Rockwell Collins France

RCF's Triplex Voter [4]

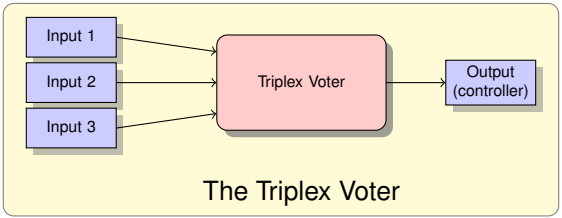


The Triplex Voter

Rockwell Collins' Triplex Voter

- Prevents dysfunctional sensors from corrupting the controller with ill values;
- makes use of saturation, middle value, centering;
- relatively simple code, but difficult to trust;
- was proven correct by Michael Dierkes [4] who found strengthening invariants by hand.

RCF's Triplex Voter [4]



The Triplex Voter

Approach

- Collaboration between K-induction [8] and Abstract Interpretation [3];
- invariant/ potential lemma generation;
- need for a framework to combine methods into.

Outline

ONERA THE FRENCH AEROSPACE LAB

Rockwell Collins

Stuff [2] in a Nutshell

Stuff is a prototype formal verification framework:

- it aims at combining formal methods running in parallel;
- it started with K-induction and Abstract Interpretation,

ONERA THE FRENCH AEROSPACE LAB

Rockwell Collins

Stuff [2] in a Nutshell

Stuff is a prototype formal verification framework:

- it aims at combining formal methods running in parallel;
- it started with K-induction and Abstract Interpretation,
- not limited to these two techniques (BQE, interpolation [5], etc);

ONERA THE FRENCH AEROSPACE LAB

Rockwell Collins

Stuff [2] in a Nutshell

Stuff is a prototype formal verification framework:

- it aims at combining formal methods running in parallel;
- it started with K-induction and Abstract Interpretation,
- not limited to these two techniques (BQE, interpolation [5], etc);
- makes extensive use of SMT-solvers (mainly z3);

Stuff [2] in a Nutshell

Stuff is a prototype formal verification framework:

- it aims at combining formal methods running in parallel;
- it started with K-induction and Abstract Interpretation,
- not limited to these two techniques (BQE, interpolation [5], etc);
- makes extensive use of SMT-solvers (mainly z3);
- written in Scala [7] (except for the Abstract Interpretation, which is written in Ocaml).

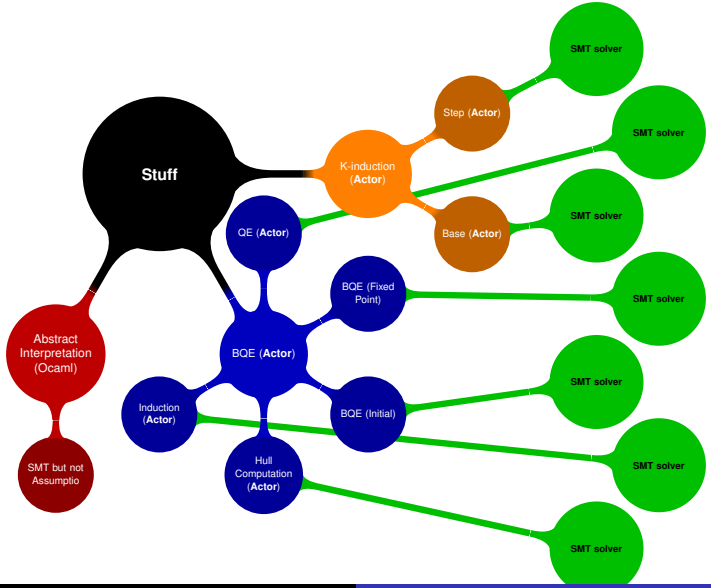
ONERA THE FRENCH AEROSPACE LAB Rockwell Collins

Paralellism

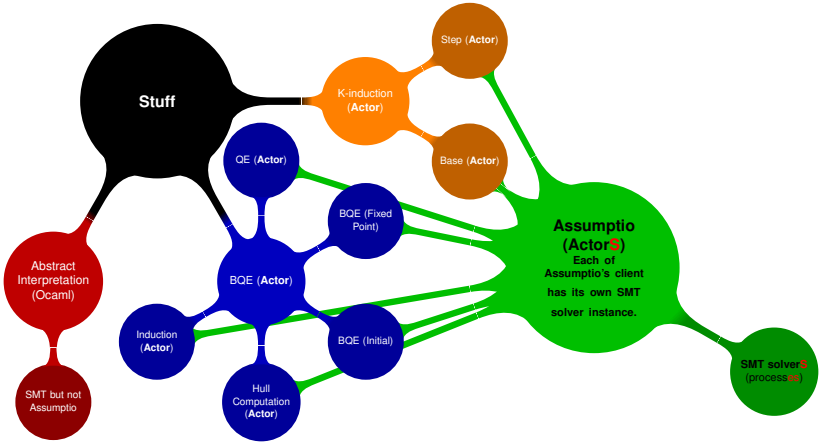
Actors

- Stuff is actor based: parallel communication between *actors* is handled through *messages* which are stored in each actor's *mailbox*;
- a lot easier to handle that many other parallel solutions;
- Scala handles repartition (almost) automatically, over cores, nodes of a computer grid, etc,
- and allows messages to be virtually anything.

A first version of Stuff's architecture

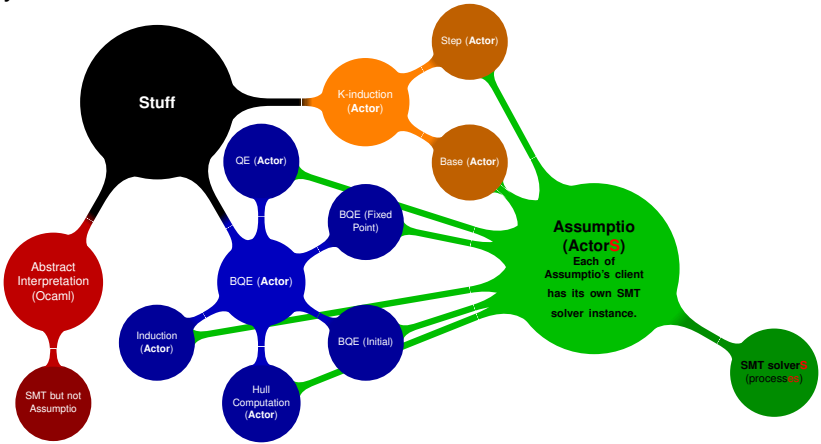


Stuff's actual architecture



Stuff's actual architecture

Additionally, k-induction and BQE do **not** even use the same data structure.



Outline

ONERA THE FRENCH AEROSPACE LAB Rockwell Collins

Assumptio's Philosophy

A Scala interface for SMT-lib 2 compliant solvers which

- allows the user to interact with solvers running in parallel;
- maintains the solver's state in order to send queries dynamically;

Assumptio's Philosophy

A Scala interface for SMT-lib 2 compliant solvers which

- allows the user to interact with solvers running in parallel;
- maintains the solver's state in order to send queries dynamically;
- does not impose a data structure to the user, nor does it constrain the user's data structure;

Assumptio's Philosophy

A Scala interface for SMT-lib 2 compliant solvers which

- allows the user to interact with solvers running in parallel;
- maintains the solver's state in order to send queries dynamically;
- does not impose a data structure to the user, nor does it constrain the user's data structure;
- is extensible: easy to add support for other solvers, and to add new features / modify the existing ones (commands, results format...);

ONERA THE FRENCH AEROSPACE LAB Rockwell Collins

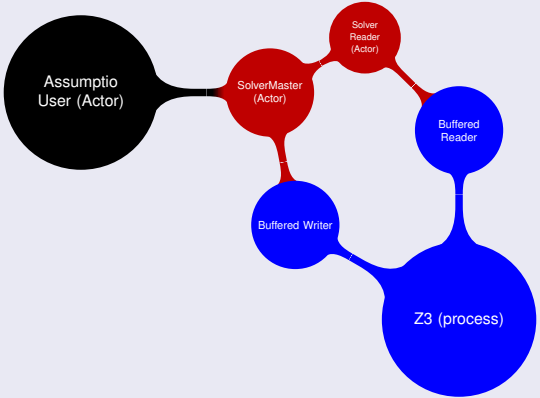
Assumptio's Philosophy

A Scala interface for SMT-lib 2 compliant solvers which

- allows the user to interact with solvers running in parallel;
- maintains the solver's state in order to send queries dynamically;
- does not impose a data structure to the user, nor does it constrain the user's data structure;
- is extensible: easy to add support for other solvers, and to add new features / modify the existing ones (commands, results format...);
- is coherent with the SMT lib 2 standard.

User, Actors and Z3

Interaction Between the User, the Actors and Z3



A Simple Example

Instanciating and Using a Solver

```
val (x,y) = (Ident(x),Ident(y))
val zero = IntLit (0)
val sorts = new ListMap[Expr,SmtSort] + ((x, SmtInt)) + ((y,SmtInt))
val solver = SolverMaster(this, Z3(), QF_LIA, sorts, "DemoSolver")

// Using the solver
solver ! Assert(Eq(Plus(x, y), zero))
solver ! CheckSat(0)
receive {
  case Sat(0) => println ("No surprise.")
  case Unsat(0) => println("Surprise.")
  case Unknown(0) => println("Cannot decide.")
}
// Push
solver ! Push(1)
solver ! Assert(Gt(x, zero))
solver ! Assert(Gt(y, zero))
solver ! CheckSat(1)
receive {
  case Sat(1) => println ("Surprise.")
  case Unsat(1) => println("No surprise.")
  case Unknown(1) => println("Cannot decide.")
}
// Exit
solver ! Exit
```

Outline

ONERA THE FRENCH AEROSPACE LAB

Rockwell Collins

QE

Quantifier Elimination (QE) yields for any formula $\mathcal{F}$ in QFLRA and any vector of variables $\vec{v}$ another formula $\mathcal{G}$ in QFLRA such that

$$\mathcal{G} \equiv \exists \vec{v},\ \mathcal{F}.$$

We will write this operation

$$\mathbb{QE}(\vec{v})(\mathcal{F}).$$

In a nutshell

QE

Quantifier Elimination (QE) yields for any formula $\mathcal{F}$ in QFLRA and any vector of variables $\vec{v}$ another formula $\mathcal{G}$ in QFLRA such that

$$\mathcal{G} \equiv \exists \vec{v},\ \mathcal{F}.$$

We will write this operation

$$\mathtt{QE}(\vec{v})(\mathcal{F}).$$

In a nutshell

- Algorithm introduced by Monniaux in [6];
- SMT based enumeration algorithm.

ONERA THE FRENCH AEROSPACE LAB Rockwell Collins

QE

Quantifier Elimination (QE) yields for any formula $\mathcal{F}$ in QFLRA and any vector of variables $\vec{v}$ another formula $\mathcal{G}$ in QFLRA such that

$$\mathcal{G} \equiv \exists \vec{v},\ \mathcal{F}.$$

We will write this operation

$$\mathtt{QE}(\vec{v})(\mathcal{F}).$$

In a nutshell

- Algorithm introduced by Monniaux in [6];
- SMT based enumeration algorithm.
- Projection: Parma Polyhedra Library [1], which also provides useful hull computation primitives.

Require: F a QFLRA formula
Ensure: $\mathcal{G} \equiv \exists \vec{v}, F$.
  $H \leftarrow F$
  $O \leftarrow$ `false`
  while
  H is satisfiable (call SMT) **do**
    $a \leftarrow$ a model of H
    $M_1 \leftarrow$ GENERALIZE1(F, a)
    $M_2 \leftarrow$
    GENERALIZE2$(\neg F, M_1)$
    $\Pi \leftarrow$ PROJECT$(M_2, \vec{v})$
    $O \leftarrow O \vee \Pi$
    $H \leftarrow H \wedge \neg \Pi$
  end while
  $\mathcal{G} \leftarrow O$
  return $\mathcal{G}$

Backward reachability analysis by Quantifier Elimination

Given a transition system $\vec{s}, D, I, T$ (LRA/LIA), and a safety property P to check:

- $\vec{s'}$ represents next state variables (*i.e.* $T(\vec{s}, \vec{s'})$ is true);
- we will call *grey states* states satisfying the property but which have a way to reach a state violating it in a finite number of transition (assuming the property is true, these should not be reachable).

Pre-image computation

- Characterization of the states satisfying the property but able to violate it in one transition:

Starting from $\neg P$

Backward reachability analysis by Quantifier Elimination

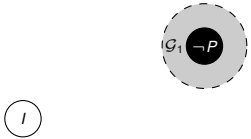
Given a transition system $\vec{s}, D, I, T$ (LRA/LIA), and a safety property P to check:

- $\vec{s'}$ represents next state variables (*i.e.* $T(\vec{s}, \vec{s'})$ is true);
- we will call *grey states* states satisfying the property but which have a way to reach a state violating it in a finite number of transition (assuming the property is true, these should not be reachable).

Pre-image computation

- Characterization of the states satisfying the property but able to violate it in one transition:
  $\mathcal{G}_1 = \mathbb{Q}\mathrm{E}(s')(P(\vec{s}) \wedge T(\vec{s}, \vec{s'}) \wedge \neg P(\vec{s'}))$

$\mathcal{G}_1$ computation



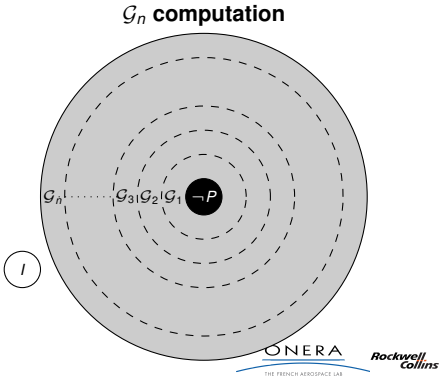
Backward reachability analysis by Quantifier Elimination

Given a transition system $\vec{s}$, D, I, T (LRA/LIA), and a safety property P to check:

- $\vec{s'}$ represents next state variables (*i.e.* $T(\vec{s}, \vec{s'})$ is true);
- we will call *grey states* states satisfying the property but which have a way to reach a state violating it in a finite number of transition (assuming the property is true, these should not be reachable).

Pre-image computation

- Characterization of the states satisfying the property but able to violate it in one transition:
  $\mathcal{G}_1 = \mathbb{Q}\mathbb{E}(s')(P(\vec{s}) \wedge T(\vec{s}, \vec{s'}) \wedge \neg P(\vec{s'}))$
- Iteration to characterize the grey states reaching a violation of the property in n transitions:
  $\forall n \geq 2,\ \mathcal{G}_n = \mathbb{Q}\mathbb{E}(s')(P(\vec{s}) \wedge T(\vec{s}, \vec{s'}) \wedge \mathcal{G}_{n-1}(\vec{s'}))$.



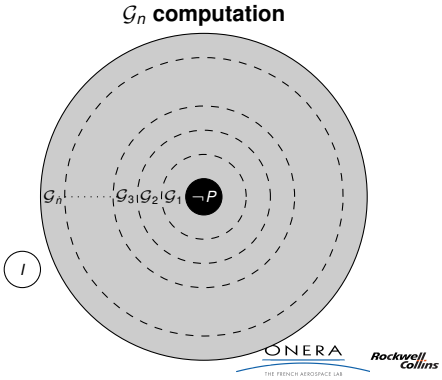
Backward reachability analysis by Quantifier Elimination

Given a transition system $\vec{s}$, D, I, T (LRA/LIA), and a safety property P to check:

- $\vec{s'}$ represents next state variables (*i.e.* $T(\vec{s}, \vec{s'})$ is true);
- we will call *grey states* states satisfying the property but which have a way to reach a state violating it in a finite number of transition (assuming the property is true, these should not be reachable).

Pre-image computation

- Characterization of the states satisfying the property but able to violate it in one transition:
  $\mathcal{G}_1 = \mathbb{QE}(s')(P(\vec{s}) \wedge T(\vec{s}, \vec{s'}) \wedge \neg P(\vec{s'}))$
- Iteration to characterize the grey states reaching a violation of the property in n transitions:
  $\forall n \geq 2,\ \mathcal{G}_n = \mathbb{QE}(s')(P(\vec{s}) \wedge T(\vec{s}, \vec{s'}) \wedge \mathcal{G}_{n-1}(\vec{s'}))$.
- At anytime, $\mathcal{H}_n$ characterizes all the grey states found so far:
  $\forall n \geq 1,\ \mathcal{H}_n \equiv \bigvee_{1 \leq i \leq n} \mathcal{G}_i$.



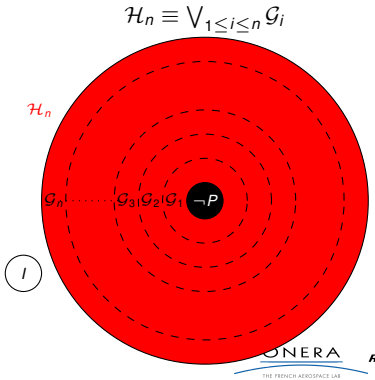
Backward reachability analysis by Quantifier Elimination

Given a transition system $\vec{s}$, D, I, T (LRA/LIA), and a safety property P to check:

- $\vec{s'}$ represents next state variables (*i.e.* $T(\vec{s}, \vec{s'})$ is true);
- we will call *grey states* states satisfying the property but which have a way to reach a state violating it in a finite number of transition (assuming the property is true, these should not be reachable).

Pre-image computation

- Characterization of the states satisfying the property but able to violate it in one transition:
  $\mathcal{G}_1 = \mathbb{Q}\mathbb{E}(s')(P(\vec{s}) \wedge T(\vec{s}, \vec{s'}) \wedge \neg P(\vec{s'}))$
- Iteration to characterize the grey states reaching a violation of the property in n transitions:
  $\forall n \geq 2,\ \mathcal{G}_n = \mathbb{Q}\mathbb{E}(s')(P(\vec{s}) \wedge T(\vec{s}, \vec{s'}) \wedge \mathcal{G}_{n-1}(\vec{s'}))$.
- At anytime, $\mathcal{H}_n$ characterizes all the grey states found so far:
  $\forall n \geq 1,\ \mathcal{H}_n \equiv \bigvee_{1 \leq i \leq n} \mathcal{G}_i$.



Example

The Double Counter

```
 1: node top(a,b,c: bool) returns (o1, o2, ok: bool);
 2: var
 3:   x, y, pre_x, pre_y: int;
 4:   n1, n2: int;
 5: let
 6:   n1    = 10;
 7:   n2    = 6;
 8:   x     = if (b or c) then 0
                          else (if (a and pre_x < n1) then pre_x + 1 else pre_x);
 9:   y     = if   (c)    then 0
                          else (if (a and pre_y < n2) then pre_y + 1 else pre_y);
10:   o1    = x = n1;
11:   o2    = y = n2;
12:   ok    = o1 => o2;
13:   pre_x = 0 -> pre(x);
14:   pre_y = 0 -> pre(y);
15:   prove(ok);                (* main proof obligation *)
18: tel
```

Double Counter

$P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcl}
x & = & 0 \rightarrow \textbf{if } (b \vee c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_x < 10) \textbf{ then } pre\_x + 1 \\
 & & \textbf{else } pre\_x \\
y & = & 0 \rightarrow \textbf{if } (c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_y < 6) \textbf{ then } pre\_y + 1 \\
 & & \textbf{else } pre\_y
\end{array}
$$

Other Methods

Double Counter

$P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcl}
x & = & 0 \rightarrow \textbf{if } (b \vee c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_x < 10) \textbf{ then } pre\_x + 1 \\
 & & \textbf{else } pre\_x \\
y & = & 0 \rightarrow \textbf{if } (c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_y < 6) \textbf{ then } pre\_y + 1 \\
 & & \textbf{else } pre\_y
\end{array}
$$

Other Methods

- K-induction by itself cannot prove the property.

Double Counter

$P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcl}
x & = & 0 \rightarrow \textbf{if } (b \vee c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_x < 10) \textbf{ then } pre\_x + 1 \\
 & & \textbf{else } pre\_x \\
y & = & 0 \rightarrow \textbf{if } (c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_y < 6) \textbf{ then } pre\_y + 1 \\
 & & \textbf{else } pre\_y
\end{array}
$$

Other Methods

- K-induction by itself cannot prove the property.
- Abstract Interpretation only infers bounds on the variables (using only intervals):

$$0 \leq x \leq 10 \ \wedge \ 0 \leq y \leq 6.$$

Double Counter

$P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcl}
x & = & 0 \rightarrow \textbf{if } (b \vee c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_x < 10) \textbf{ then } pre\_x + 1 \\
 & & \textbf{else } pre\_x \\
y & = & 0 \rightarrow \textbf{if } (c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_y < 6) \textbf{ then } pre\_y + 1 \\
 & & \textbf{else } pre\_y
\end{array}
$$

Other Methods

- K-induction by itself cannot prove the property.
- Abstract Interpretation only infers bounds on the variables (using only intervals):

$$0 \leq x \leq 10 \ \wedge \ 0 \leq y \leq 6.$$

- Using the bounds found by AI, K-induction manages to prove the property, but needs to unroll the transition relation a number of times proportionnal to higher bounds, and thus does not scale.

ONERA THE FRENCH AEROSPACE LAB Rockwell Collins

Double Counter

$P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcl}
x & = & 0 \rightarrow \textbf{if } (b \vee c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_x < 10) \textbf{ then } pre\_x + 1 \\
 & & \textbf{else } pre\_x \\
y & = & 0 \rightarrow \textbf{if } (c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_y < 6) \textbf{ then } pre\_y + 1 \\
 & & \textbf{else } pre\_y
\end{array}
$$

BQE: $P \equiv x = 10 \rightarrow y = 6$ (with bounds)

Double Counter

$P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcl}
x & = & 0 \rightarrow \textbf{if } (b \vee c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_x < 10) \textbf{ then } pre\_x + 1 \\
 & & \textbf{else } pre\_x \\
y & = & 0 \rightarrow \textbf{if } (c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_y < 6) \textbf{ then } pre\_y + 1 \\
 & & \textbf{else } pre\_y
\end{array}
$$

BQE: $P \equiv x = 10 \rightarrow y = 6$ (with bounds)

$$\mathcal{G}_1 \equiv x = 9 \wedge 0 \leq y < 5$$

Double Counter

$P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcl}
x & = & 0 \rightarrow \textbf{if } (b \vee c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_x < 10) \textbf{ then } pre\_x + 1 \\
 & & \textbf{else } pre\_x \\
y & = & 0 \rightarrow \textbf{if } (c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_y < 6) \textbf{ then } pre\_y + 1 \\
 & & \textbf{else } pre\_y
\end{array}
$$

BQE: $P \equiv x = 10 \rightarrow y = 6$ (with bounds)

$$
\begin{array}{lcl}
\mathcal{G}_1 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
\mathcal{G}_2 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
 & \bigvee & x = 8 \wedge 0 \leq y < 4
\end{array}
$$

Double Counter

$P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcl}
x & = & 0 \rightarrow \textbf{if } (b \vee c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_x < 10) \textbf{ then } pre\_x + 1 \\
 & & \textbf{else } pre\_x \\
y & = & 0 \rightarrow \textbf{if } (c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_y < 6) \textbf{ then } pre\_y + 1 \\
 & & \textbf{else } pre\_y
\end{array}
$$

BQE: $P \equiv x = 10 \rightarrow y = 6$ (with bounds)

$$
\begin{array}{lcl}
\mathcal{G}_1 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
\mathcal{G}_2 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
 & \bigvee & x = 8 \wedge 0 \leq y < 4 \\
\ldots & & \\
\mathcal{G}_5 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
 & \bigvee & \ldots \\
 & \bigvee & x = 5 \wedge 0 \leq y < 1
\end{array}
$$

Double Counter

$P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcl}
x & = & 0 \rightarrow \textbf{if } (b \vee c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_x < 10) \textbf{ then } pre\_x + 1 \\
 & & \textbf{else } pre\_x \\
y & = & 0 \rightarrow \textbf{if } (c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_y < 6) \textbf{ then } pre\_y + 1 \\
 & & \textbf{else } pre\_y
\end{array}
$$

BQE: $P \equiv x = 10 \rightarrow y = 6$ (with bounds)

$$
\begin{array}{lcl}
\mathcal{G}_1 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
\mathcal{G}_2 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
 & \bigvee & x = 8 \wedge 0 \leq y < 4 \\
\ldots & & \\
\mathcal{G}_5 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
 & \bigvee & \ldots \\
 & \bigvee & x = 5 \wedge 0 \leq y < 1
\end{array}
$$

We will call **hullification** the act of trying to merge together as many of the polyhedra defined by a (DNF) formula's disjuncts as possible, in **exact convex hulls**.

BQE with hullification: $P \equiv x = 10 \rightarrow y = 6$

Double Counter

$P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcl}
x & = & 0 \rightarrow \textbf{if } (b \vee c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_x < 10) \textbf{ then } pre\_x + 1 \\
 & & \textbf{else } pre\_x \\
y & = & 0 \rightarrow \textbf{if } (c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_y < 6) \textbf{ then } pre\_y + 1 \\
 & & \textbf{else } pre\_y
\end{array}
$$

BQE: $P \equiv x = 10 \rightarrow y = 6$ (with bounds)

$$
\begin{array}{lcl}
\mathcal{G}_1 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
\mathcal{G}_2 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
 & \bigvee & x = 8 \wedge 0 \leq y < 4 \\
\ldots & & \\
\mathcal{G}_5 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
 & \bigvee & \ldots \\
 & \bigvee & x = 5 \wedge 0 \leq y < 1
\end{array}
$$

We will call **hullification** the act of trying to merge together as many of the polyhedra defined by a (DNF) formula's disjuncts as possible, in **exact convex hulls**.

BQE with hullification: $P \equiv x = 10 \rightarrow y = 6$

$$\mathcal{G}_1 \equiv x = 9 \wedge 0 \leq y < 5$$

Double Counter

$P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcl}
x & = & 0 \rightarrow \textbf{if } (b \vee c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_x < 10) \textbf{ then } pre\_x + 1 \\
 & & \textbf{else } pre\_x \\
y & = & 0 \rightarrow \textbf{if } (c) \textbf{ then } 0 \\
 & & \textbf{else if } (a \wedge pre\_y < 6) \textbf{ then } pre\_y + 1 \\
 & & \textbf{else } pre\_y
\end{array}
$$

BQE: $P \equiv x = 10 \rightarrow y = 6$ (with bounds)

$$
\begin{array}{lcl}
\mathcal{G}_1 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
\mathcal{G}_2 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
 & \bigvee & x = 8 \wedge 0 \leq y < 4 \\
\ldots & & \\
\mathcal{G}_5 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
 & \bigvee & \ldots \\
 & \bigvee & x = 5 \wedge 0 \leq y < 1
\end{array}
$$

We will call **hullification** the act of trying to merge together as many of the polyhedra defined by a (DNF) formula's disjuncts as possible, in **exact convex hulls**.

BQE with hullification: $P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcl}
\mathcal{G}_1 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
\mathcal{G}_2 & \equiv & 8 \leq x \leq 9 \wedge 0 \leq y < x - 4
\end{array}
$$

Double Counter

$P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{llllllll}
x & = & 0 \rightarrow \textbf{if } (b \vee c) & \textbf{then} & 0 & & & \\
 & & & \textbf{else} & \textbf{if } (a \wedge pre\_x < 10) & \textbf{then} & pre\_x + 1 & \\
 & & & & & \textbf{else} & pre\_x & \\
y & = & 0 \rightarrow \textbf{if } (c) & \textbf{then} & 0 & & & \\
 & & & \textbf{else} & \textbf{if } (a \wedge pre\_y < 6) & \textbf{then} & pre\_y + 1 & \\
 & & & & & \textbf{else} & pre\_y &
\end{array}
$$

BQE: $P \equiv x = 10 \rightarrow y = 6$ (with bounds)

$$
\begin{array}{lllll}
\mathcal{G}_1 & \equiv & x = 9 & \wedge & 0 \leq y < 5 \\
\mathcal{G}_2 & \equiv & x = 9 & \wedge & 0 \leq y < 5 \\
 & \bigvee & x = 8 & \wedge & 0 \leq y < 4 \\
\ldots & & & & \\
\mathcal{G}_5 & \equiv & x = 9 & \wedge & 0 \leq y < 5 \\
 & \bigvee & \ldots & & \\
 & \bigvee & x = 5 & \wedge & 0 \leq y < 1
\end{array}
$$

We will call **hullification** the act of trying to merge together as many of the polyhedra defined by a (DNF) formula's disjuncts as possible, in **exact convex hulls**.

BQE with hullification: $P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lllll}
\mathcal{G}_1 & \equiv & x = 9 & \wedge & 0 \leq y < 5 \\
\mathcal{G}_2 & \equiv & 8 \leq x \leq 9 & \wedge & 0 \leq y < x - 4 \\
\ldots & & & & \\
\mathcal{G}_5 & \equiv & 5 \leq x \leq 9 & \wedge & 0 \leq y < x - 4
\end{array}
$$

ONERA THE FRENCH AEROSPACE LAB Rockwell Collins

Double Counter

$P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcllllll}
x & = & 0 \rightarrow \textbf{if } (b \vee c) & \textbf{then} & 0 & & & \\
 & & & \textbf{else} & \textbf{if } (a \wedge pre\_x < 10) & \textbf{then} & pre\_x + 1 \\
 & & & & & \textbf{else} & pre\_x \\
y & = & 0 \rightarrow \textbf{if } (c) & \textbf{then} & 0 & & \\
 & & & \textbf{else} & \textbf{if } (a \wedge pre\_y < 6) & \textbf{then} & pre\_y + 1 \\
 & & & & & \textbf{else} & pre\_y
\end{array}
$$

BQE: $P \equiv x = 10 \rightarrow y = 6$ (with bounds)

$$
\begin{array}{lcl}
\mathcal{G}_1 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
\mathcal{G}_2 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
 & \bigvee & x = 8 \wedge 0 \leq y < 4 \\
\ldots & & \\
\mathcal{G}_5 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
 & \bigvee & \ldots \\
 & \bigvee & x = 5 \wedge 0 \leq y < 1
\end{array}
$$

We will call **hullification** the act of trying to merge together as many of the polyhedra defined by a (DNF) formula's disjuncts as possible, in **exact convex hulls**.

BQE with hullification: $P \equiv x = 10 \rightarrow y = 6$

$$
\begin{array}{lcl}
\mathcal{G}_1 & \equiv & x = 9 \wedge 0 \leq y < 5 \\
\mathcal{G}_2 & \equiv & 8 \leq x \leq 9 \wedge 0 \leq y < x - 4 \\
\ldots & & \\
\mathcal{G}_5 & \equiv & 5 \leq x \leq 9 \wedge 0 \leq y < x - 4
\end{array}
$$

The property augmented by **the bounds found by AI** and **the lemma** $y \geq x - 4$ **found by BQE** becomes **inductive**.

An Iteration: More Than Just Quantifier Elimination

Given $\mathcal{H}_{n-2}$ and $\mathcal{G}_{n-1}$

```
// Next step's QE computation.
QE ! Eliminate(V_QE, P(s) and T(s,s_p) and Gn-1(s_p))

// Checking if the initial states
// intersect the pre-image computed so far.
solver1 ! Script(Clean::Assert(I(s) and H_n-2(s))::CheckSat(1)::Nil)

// Checking if a fixed point has been reached.
solver2 ! Script(Clean::Assert(H_n-2(s) and Not(G_n-1(s)))::CheckSat(0)::Nil)

// Hullification for Hn-1
hullK ! Hullify (H_n-2 or G_n-1)

// Tries to construct a lemma making the property
// inductive, then tries to construct an inductive
// lemma if the former failed.
abducter ! IsInductive(Hn-2)
```

In Short

BQE

- backward property directed analysis;

In Short

BQE

- backward property directed analysis;
- should **not** be used as a standalone method,

In Short

BQE

- backward property directed analysis;
- should **not** be used as a standalone method,
- but rather as a lemma generator, thanks to hullification;

In Short

BQE

- backward property directed analysis;
- should **not** be used as a standalone method,
- but rather as a lemma generator, thanks to hullification;
- outputs information and refines it at each step;

In Short

BQE

- backward property directed analysis;
- should **not** be used as a standalone method,
- but rather as a lemma generator, thanks to hullification;
- outputs information and refines it at each step;
- can integrate new theorems during analysis;

In Short

BQE

- backward property directed analysis;
- should **not** be used as a standalone method,
- but rather as a lemma generator, thanks to hullification;
- outputs information and refines it at each step;
- can integrate new theorems during analysis;
- is exact...

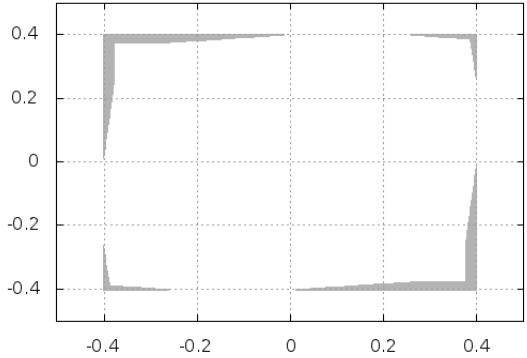
In Short

BQE

- backward property directed analysis;
- should **not** be used as a standalone method,
- but rather as a lemma generator, thanks to hullification;
- outputs information and refines it at each step;
- can integrate new theorems during analysis;
- is exact...
- ... as long as we want it to be.

The Duplex Voter

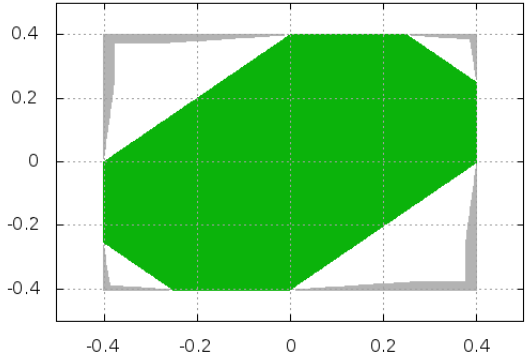
- We work in the $[-0.4; 0.4]x[-0.4; 0.4]$ square (because at each step BQE looks for states satisfying the property),
- grey areas: BQE's result after the 1^{st} iteration,



ONERA THE FRENCH AEROSPACE LAB Rockwell Collins

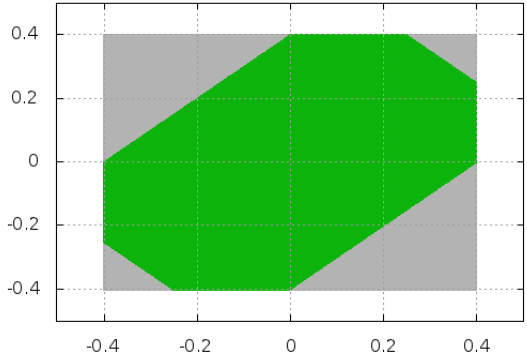
The Duplex Voter

- We work in the $[-0.4; 0.4]x[-0.4; 0.4]$ square (because at each step BQE looks for states satisfying the property),
- grey areas: BQE's result after the 1^{st} iteration,
- central green octagone: invariants found by hand by Michael Dierkes.



The Duplex Voter

- We work in the $[-0.4; 0.4]x[-0.4; 0.4]$ square (because at each step BQE looks for states satisfying the property),
- **grey triangles: directed inexact hullification of BQE's result**,
- central green octagone: invariants found by hand by Michael Dierkes.



Outline

ONERA THE FRENCH AEROSPACE LAB

Rockwell Collins

Our questions

About Z3:

Our questions

About Z3:

- **Push** / **Pop** mechanism;

QE Solver Stack for BQE

$$\neg P(s) \wedge T(s, s')$$

Our questions

About Z3:

- **Push** / **Pop** mechanism;

QE Solver Stack for BQE

$$\frac{G_i(s')}{\neg P(s) \wedge T(s, s')}$$

Our questions

About Z3:

- **Push** / **Pop** mechanism;

QE Solver Stack for BQE

$$\neg P(s) \wedge T(s, s')$$

Our questions

About Z3:

- **Push** / **Pop** mechanism;

QE Solver Stack for BQE

$$\frac{G_{i+1}(s')}{\neg P(s) \wedge T(s, s')}$$

Our questions

About Z3:

- **Push** / **Pop** mechanism;
- Using **let**-s or constraints;

For Example

- $\texttt{assert}(\texttt{let}\ x = F\ \texttt{in}\ P(x))$
- $\texttt{assert}(P(x))$ $\texttt{assert}(x = F)$

Our questions

About Z3:

- **Push** / **Pop** mechanism;
- Using **let**-s or constraints;
- Interpolants ?

References

R. Bagnara, P. M. Hill, and E. Zaffanella.

The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems.
Science of Computer Programming, 72(1–2):3–21, 2008.

Adrien Champion, Rémi Delmas, Pierre-Loïc Garoche, and Pierre Roux.

Towards cooperation of formal methods for the analysis of critical control systems.
In *SAE Aerotech, to be published*, 2011.

Patrick Cousot and Radhia Cousot.

Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints.
In *POPL*, pages 238–252, 1977.

Michael Dierkes.

Formal analysis of a triplex sensor voter in an industrial context.
In G. Salaün and B. Schätz, editors, *Proceedings of the 16th International Workshop on Formal Methods for Industrial Critical Systems, FMICS 2011*, volume 6959 of *LNCS*. Springer, 2011.

Kenneth L. McMillan.

Interpolation and sat-based model checking.
In Warren A. Hunt Jr. and Fabio Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003.

David Monniaux.

A quantifier elimination algorithm for linear real arithmetic.
In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *LPAR*, volume 5330 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 2008.

Martin Odersky and al.

An Overview of the Scala Programming Language.
Technical Report IC/2004/64, EPFL, Lausanne, Switzerland, 2004.

Mary Sheeran, Satnam Singh, and Gunnar Stålmarck.

Checking safety properties using induction and a sat-solver.
In Warren A. Hunt Jr. and Steven D. Johnson, editors, *FMCAD*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2000.

ONERA THE FRENCH AEROSPACE LAB

Rockwell Collins