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

Adrien Champion
adrien.champion@onera.fr

November 2011
Table of Contents

1 Introduction
   • Context

2 Stuff
   • Stuff’s The Ultimate Formal Framework
   • Stuff’s Current State

3 Assumptio
   • A Brief Description
   • A Glimpse at its Architecture
   • A Quick Example

4 BQE
   • Monniaux’s QE algorithm
   • BQE Algorithm
   • Conclusion

5 Questions
Outline

1 Introduction
   • Context

2 Stuff
   • Stuff’s The Ultimate Formal Framework
   • Stuff’s Current State

3 Assumptio
   • A Brief Description
   • A Glimpse at its Architecture
   • A Quick Example

4 BQE
   • Monniaux’s QE algorithm
   • BQE Algorithm
   • Conclusion

5 Questions

A. Champion: Assumptio and Stuff: using Z3 in a collaborative parallel formal verification framework.
Main goal: formal reachability analysis of critical embedded systems;
Synchronous languages, typically Lustre;
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
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.
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.
Introduction

1. Context

2. Stuff
   - Stuff’s The Ultimate Formal Framework
   - Stuff’s Current State

3. Assumptio
   - A Brief Description
   - A Glimpse at its Architecture
   - A Quick Example

4. BQE
   - Monniaux’s QE algorithm
   - BQE Algorithm
   - Conclusion

5. Questions

A. Champion

Assumptio and Stuff: using Z3 in a collaborative parallel formal verification framework.
Stuff \cite{1} 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,
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);
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 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).
A. Champion

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

Parallellism

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**
  - Abstract Interpretation (Ocaml)
  - SMT but not Assumptio

- **BQE (Actor)**
  - Base (Actor)
  - BQE (Fixed Point)
  - BQE (Initial)
  - Hull Computation (Actor)
  - Induction (Actor)

- **K-induction (Actor)**
  - Step (Actor)

- **SMT solver**
  - SMT solver
  - SMT solver
  - SMT solver
  - SMT solver
  - SMT solver

A. Champion

Assumptio and Stuff: using Z3 in a collaborative parallel formal verification framework.
Additionally, k-induction and BQE do not even use the same data structure.

Stuff: The Ultimate Formal Framework

Assumptio and Stuff: using Z3 in a collaborative parallel formal verification framework.
Additionally, k-induction and BQE do **not** even use the same data structure.
Outline

1. Introduction
   - Context

2. Stuff
   - Stuff's The Ultimate Formal Framework
   - Stuff's Current State

3. Assumptio
   - A Brief Description
   - A Glimpse at its Architecture
   - A Quick Example

4. BQE
   - Monniaux's QE algorithm
   - BQE Algorithm
   - Conclusion

5. Questions
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;
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;
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. . .);
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

Assumptio User (Actor)
SolverMaster (Actor)
Solver Reader (Actor)
BufferedReader
Buffered Writer
Z3 (process)
A Simple Example

Instanciating and Using a Solver

```scala
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
```

A. Champion
Outline

1. Introduction
   - Context

2. Stuff
   - Stuff’s The Ultimate Formal Framework
   - Stuff’s Current State

3. Assumptio
   - A Brief Description
   - A Glimpse at its Architecture
   - A Quick Example

4. BQE
   - Monniaux’s QE algorithm
   - BQE Algorithm
   - Conclusion

5. Questions

A. Champion

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

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

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

We will write this operation

$$\text{QE}(\bar{v})(\mathcal{F}).$$ 

In a nutshell

- Algorithm introduced by Monniaux in [6];
- SMT based enumeration algorithm.
Quantifier Elimination (QE) yields for any formula $F$ in QFLRA and any vector of variables $\vec{v}$ another formula $G$ in QFLRA such that

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

We will write this operation

$$\text{QE}(\vec{v})(F).$$

### 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:

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

$H \leftarrow F$

$O \leftarrow \text{false}$

**while**

$H$ is satisfiable (call SMT) **do**

$a \leftarrow$ a model of $H$

$M_1 \leftarrow \text{GENERALIZE1}(F, a)$

$M_2 \leftarrow \text{GENERALIZE2}(\neg F, M_1)$

$\Pi \leftarrow \text{PROJECT}(M_2, \vec{v})$

$O \leftarrow O \lor \Pi$

$H \leftarrow H \land \neg \Pi$

**end while**

$\text{QE(}\vec{v})(F)$

return $G$
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 \( \neg P \)**
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:

$$G_1 = \text{QE}(\vec{s}') (P(\vec{s}) \land T(\vec{s}, \vec{s}') \land \neg P(\vec{s}'))$$

$G_1$ 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:
  $$ \mathcal{G}_1 = \text{QE}(s')(P(\vec{s}) \land T(\vec{s}, \vec{s}') \land \neg P(\vec{s}')) $$

- Iteration to characterize the grey states reaching a violation of the property in $n$ transitions:
  $$ \forall n \geq 2, \quad \mathcal{G}_n = \text{QE}(s')(P(\vec{s}) \land T(\vec{s}, \vec{s}') \land \mathcal{G}_{n-1}(\vec{s}')) $$  

A. Champion
Assumptio and Stuff: using Z3 in a collaborative parallel formal verification framework.
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:
  $$G_1 = \text{QE}(\vec{s}')(P(\vec{s}) \land T(\vec{s}, \vec{s}') \land \neg P(\vec{s}'))$$
- Iteration to characterize the grey states reaching a violation of the property in $n$ transitions:
  $$\forall n \geq 2, \ G_n = \text{QE}(\vec{s}')(P(\vec{s}) \land T(\vec{s}, \vec{s}') \land G_{n-1}(\vec{s}')).$$  
- At anytime, $H_n$ characterizes all the grey states found so far:
  $$\forall n \geq 1, \ H_n \equiv \bigvee_{1 \leq i \leq n} G_i.$$
Given a transition system $\vec{s}$, $D$, $I$, $T$ (LRA/LIA), and a safety property $P$ to check:

- $s'$ 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:
  \[ G_1 = \text{QE}(s') (P(\vec{s}) \land T(\vec{s}, \vec{s}') \land \neg P(\vec{s}')) \]
- Iteration to characterize the grey states reaching a violation of the property in $n$ transitions:
  \[ \forall n \geq 2, \ G_n = \text{QE}(s') (P(\vec{s}) \land T(\vec{s}, \vec{s}') \land G_{n-1}(\vec{s}')) \]
- At anytime, $H_n$ characterizes all the grey states found so far:
  \[ \forall n \geq 1, \ H_n \equiv \bigvee_{1 \leq i \leq n} G_i. \]
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

\[ P \equiv x = 10 \rightarrow y = 6 \]

\[
\begin{align*}
x &= 0 \rightarrow \text{if} \ (b \lor c) \quad \text{then} \quad 0 \\
&\quad \text{else if} \ (a \land \text{pre}_x < 10) \quad \text{then} \quad \text{pre}_x + 1 \\
&\quad \text{else} \quad \text{pre}_x \\
y &= 0 \rightarrow \text{if} \ (c) \quad \text{then} \quad 0 \\
&\quad \text{else if} \ (a \land \text{pre}_y < 6) \quad \text{then} \quad \text{pre}_y + 1 \\
&\quad \text{else} \quad \text{pre}_y
\end{align*}
\]

Other Methods

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

\[ 0 \leq x \leq 10 \land 0 \leq y \leq 6 \]

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

A. Champion

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

\[ P \equiv x = 10 \rightarrow y = 6 \]

\[
\begin{align*}
  x &= 0 \rightarrow \text{if } (b \lor c) \quad \text{then} \quad 0 \\
      &\quad \text{else if } (a \land \text{pre}_x < 10) \quad \text{then} \quad \text{pre}_x + 1 \\
      &\quad \text{else} \quad \text{pre}_x \\
  y &= 0 \rightarrow \text{if } (c) \quad \text{then} \quad 0 \\
      &\quad \text{else if } (a \land \text{pre}_y < 6) \quad \text{then} \quad \text{pre}_y + 1 \\
      &\quad \text{else} \quad \text{pre}_y
\end{align*}
\]

Other Methods

- K-induction by itself cannot prove the property.
Double Counter

\[ P \equiv x = 10 \rightarrow y = 6 \]

\[
\begin{align*}
x &= 0 \rightarrow \text{if } (b \lor c) \quad \text{then} & 0 \\
    & \quad \text{else} \text{ if } (a \land \text{pre}_x < 10) \quad \text{then} & \text{pre}_x + 1 \\
    & \quad \text{else} \quad \text{pre}_x \\
y &= 0 \rightarrow \text{if } (c) \quad \text{then} & 0 \\
    & \quad \text{else} \text{ if } (a \land \text{pre}_y < 6) \quad \text{then} & \text{pre}_y + 1 \\
    & \quad \text{else} \quad \text{pre}_y
\end{align*}
\]

Other Methods

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

\[ 0 \leq x \leq 10 \land 0 \leq y \leq 6. \]
Double Counter

\[ P \equiv x = 10 \rightarrow y = 6 \]

\[
x = 0 \rightarrow \text{if } (b \lor c) \quad \text{then} \quad 0
\]
\[
\text{else if } (a \land \text{pre}_x < 10) \quad \text{then} \quad \text{pre}_x + 1
\]
\[
\text{else} \quad \text{pre}_x
\]

\[
y = 0 \rightarrow \text{if } (c) \quad \text{then} \quad 0
\]
\[
\text{else if } (a \land \text{pre}_y < 6) \quad \text{then} \quad \text{pre}_y + 1
\]
\[
\text{else} \quad \text{pre}_y
\]

Other Methods

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

\[ 0 \leq x \leq 10 \land 0 \leq y \leq 6. \]

- 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.
Double Counter

\[ \text{Double Counter} \]

\[ \begin{align*}
  P & \equiv x = 10 \rightarrow y = 6 \\
  x &= 0 \rightarrow \text{if } (b \lor c) \quad \text{then } 0 \\
      &\quad \text{else if } (a \land \text{pre}_x < 10) \quad \text{then } \text{pre}_x + 1 \\
      &\quad \text{else } \text{pre}_x \\
  y &= 0 \rightarrow \text{if } (c) \quad \text{then } 0 \\
      &\quad \text{else if } (a \land \text{pre}_y < 6) \quad \text{then } \text{pre}_y + 1 \\
      &\quad \text{else } \text{pre}_y
\end{align*} \]

BQE: \[ P \equiv x = 10 \rightarrow y = 6 \text{ (with bounds)} \]
Double Counter

\[ P \equiv x = 10 \rightarrow y = 6 \]

\[
\begin{align*}
  x &= 0 \rightarrow \text{if } (b \lor c) \\
  \quad &\quad \text{then } 0 \\
  \quad &\quad \text{else } \text{if } (a \land \text{pre}_x < 10) \\
  \quad &\quad \text{then } \text{pre}_x + 1 \\
  \quad &\quad \text{else } \text{pre}_x \\
  y &= 0 \rightarrow \text{if } (c) \\
  \quad &\quad \text{then } 0 \\
  \quad &\quad \text{else } \text{if } (a \land \text{pre}_y < 6) \\
  \quad &\quad \text{then } \text{pre}_y + 1 \\
  \quad &\quad \text{else } \text{pre}_y
\end{align*}
\]

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

\[ G_1 \equiv x = 9 \land 0 \leq y < 5 \]
Double Counter

\[ P \equiv x = 10 \rightarrow y = 6 \]

\[
\begin{align*}
x &= 0 \rightarrow \text{if } (b \lor c) \quad \text{then } 0 \\
&\quad \quad \text{else if } (a \land \text{pre}_x < 10) \quad \text{then } \text{pre}_x + 1 \\
&\quad \quad \quad \text{else if } (a \land \text{pre}_y < 6) \quad \text{then } \text{pre}_y + 1 \\
y &= 0 \rightarrow \text{if } (c) \quad \text{then } 0 \\
&\quad \quad \text{else if } (a \land \text{pre}_x < 10) \quad \text{else } \text{pre}_x \\
&\quad \quad \quad \text{else if } (a \land \text{pre}_y < 6) \quad \text{else } \text{pre}_y
\end{align*}
\]

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

\[
\begin{align*}
G_1 &\equiv x = 9 \land 0 \leq y < 5 \\
G_2 &\equiv x = 9 \land 0 \leq y < 5 \\
&\lor x = 8 \land 0 \leq y < 4
\end{align*}
\]
Double Counter

\[ P \equiv x = 10 \rightarrow y = 6 \]

\[
\begin{align*}
  x & = 0 \rightarrow \text{if } (b \vee c) \text{ then } 0 \text{ else } (a \land \text{pre}_x < 10) \text{ then } \text{pre}_x + 1 \text{ else } \text{pre}_x \\
  y & = 0 \rightarrow \text{if } (c) \text{ then } 0 \text{ else } (a \land \text{pre}_y < 6) \text{ then } \text{pre}_y + 1 \text{ else } \text{pre}_y
\end{align*}
\]

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

\[ G_1 \equiv x = 9 \land 0 \leq y < 5 \]
\[ G_2 \equiv \bigvee x = 8 \land 0 \leq y < 4 \]
\[ \ldots \]
\[ G_5 \equiv x = 9 \land 0 \leq y < 5 \]
\[ \bigvee \ldots \bigvee x = 5 \land 0 \leq y < 1 \]
Double Counter

\[ P \equiv x = 10 \rightarrow y = 6 \]

\[
x = 0 \rightarrow \begin{cases} 
    0 & \text{if } (b \lor c) \\
    \text{else if } (a \land \text{pre}_x < 10) \\
    \text{else if } (a \land \text{pre}_y < 6) \\
    \text{else}
  \end{cases}
\]

\[
y = 0 \rightarrow \begin{cases} 
    0 \\
    \text{if } (c) \\
    \text{else if } (a \land \text{pre}_y < 6) \\
    \text{else}
  \end{cases}
\]

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

\[
G_1 \equiv x = 9 \land 0 \leq y < 5
\]

\[
G_2 \equiv \begin{aligned} 
G_1 & \lor \\
G_3 & \lor \\
G_5 & \lor
\end{aligned}
\]

\[
G_3 \equiv x = 9 \land 0 \leq y < 4
\]

\[
G_5 \equiv x = 9 \land 0 \leq y < 5
\]

\[
G_6 \equiv \begin{aligned} 
G_5 & \lor \\
\cdots & \lor \\
G_8 & \lor
\end{aligned}
\]

\[
G_8 \equiv x = 5 \land 0 \leq y < 1
\]
Double Counter

\[ P \equiv x = 10 \rightarrow y = 6 \]

\[
\begin{align*}
  x &= 0 \rightarrow \text{if } (b \lor c) \text{ then } 0 \\
      &\quad \text{else if } (a \land \text{pre}_x < 10) \text{ then } \text{pre}_x + 1 \\
      &\quad \text{else if } (a \land \text{pre}_y < 6) \text{ then } \text{pre}_y + 1 \\
  y &= 0 \rightarrow \text{if } (c) \text{ then } 0 \\
      &\quad \text{else if } (a \land \text{pre}_y < 6) \text{ then } \text{pre}_y + 1 \\
      &\quad \text{else } \text{pre}_y
\end{align*}
\]

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

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

\[
\begin{align*}
  G_1 &\equiv x = 9 \land 0 \leq y < 5 \\
  G_2 &\equiv x = 9 \land 0 \leq y < 5 \\
  \quad \lor x = 8 \land 0 \leq y < 4 \\
  \vdots \\
  G_5 &\equiv x = 9 \land 0 \leq y < 5 \\
  \quad \lor \ldots \\
  \quad \lor x = 5 \land 0 \leq y < 1
\end{align*}
\]
Double Counter

\[ P \equiv x = 10 \rightarrow y = 6 \]

\[
x = 0 \rightarrow \text{if} \ (b \lor c) \quad \text{then} \quad 0 \\
\quad \text{else} \quad \text{if} \ (a \land \text{pre}_x < 10) \quad \text{then} \quad \text{pre}_x + 1 \\
\quad \text{else} \quad \text{pre}_x
\]

\[
y = 0 \rightarrow \text{if} \ (c) \quad \text{then} \quad 0 \\
\quad \text{else} \quad \text{if} \ (a \land \text{pre}_y < 6) \quad \text{then} \quad \text{pre}_y + 1 \\
\quad \text{else} \quad \text{pre}_y
\]

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

- \( G_1 \equiv x = 9 \land 0 \leq y < 5 \)
- \( G_2 \equiv x = 9 \land 0 \leq y < 5 \lor x = 8 \land 0 \leq y < 4 \)
- \( \ldots \)
- \( G_5 \equiv x = 9 \land 0 \leq y < 5 \lor \ldots \lor x = 5 \land 0 \leq y < 1 \)

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

- \( G_1 \equiv x = 9 \land 0 \leq y < 5 \)
- \( G_2 \equiv 8 \leq x \leq 9 \land 0 \leq y < x - 4 \)
Double Counter

\[ P \equiv x = 10 \rightarrow y = 6 \]

\[
\begin{align*}
x &= 0 \rightarrow \text{if } (b \lor c) & \text{then 0 else if } (a \land \text{pre}_x < 10) & \text{then } \text{pre}_x + 1 \text{ else } \text{pre}_x \\
y &= 0 \rightarrow \text{if } (c) & \text{then 0 else if } (a \land \text{pre}_y < 6) & \text{then } \text{pre}_y + 1 \text{ else } \text{pre}_y
\end{align*}
\]

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**.

\[
\begin{align*}
\text{BQE: } P \equiv x = 10 \rightarrow y = 6 \text{ (with bounds)} \\
G_1 &\equiv x = 9 \land 0 \leq y < 5 \\
G_2 &\equiv x = 9 \land 0 \leq y < 5 \lor x = 8 \land 0 \leq y < 4 \\
\text{...} \\
G_5 &\equiv x = 9 \land 0 \leq y < 5 \lor \ldots \lor x = 5 \land 0 \leq y < 1
\end{align*}
\]

\[
\begin{align*}
\text{BQE with hullification: } P \equiv x = 10 \rightarrow y = 6 \\
\text{...} \\
G_5 &\equiv 5 \leq x \leq 9 \land 0 \leq y < x - 4
\end{align*}
\]
Double Counter

\[ P \equiv x = 10 \rightarrow y = 6 \]

\[
x = 0 \rightarrow \text{if } (b \lor c) \text{ then } 0 \text{ else if } (a \land \text{pre}_x < 10) \text{ then } \text{pre}_x + 1 \text{ else } \text{pre}_x
\]

\[
y = 0 \rightarrow \text{if } (c) \text{ then } 0 \text{ else if } (a \land \text{pre}_y < 6) \text{ then } \text{pre}_y + 1 \text{ else } \text{pre}_y
\]

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

\[
\begin{align*}
G_1 & \equiv x = 9 \land 0 \leq y < 5 \\
G_2 & \equiv x = 9 \land 0 \leq y < 5 \\
       & \lor x = 8 \land 0 \leq y < 4 \\
    & \ldots \\
G_5 & \equiv x = 9 \land 0 \leq y < 5 \\
       & \lor \ldots \\
       & \lor x = 5 \land 0 \leq y < 1
\end{align*}
\]

The property augmented by the **bounds** found by AI and the lemma \( y \geq x - 4 \) found by BQE becomes **inductive**.
Given $H_{n-2}$ and $G_{n-1}$

```plaintext
// Next step's QE computation.
QE ! Eliminate(V_QE, P(s) and T(s,s_p) and G_{n-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 H_{n-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(H_{n-2})
```
BQE

- backward property directed analysis;
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 \([-0.4, 0.4] \times [-0.4, 0.4]\) square (because at each step BQE looks for states satisfying the property),
- grey areas: BQE’s result after the 1\(^{st}\) iteration,
The Duplex Voter

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

1 Introduction
   • Context

2 Stuff
   • Stuff’s The Ultimate Formal Framework
   • Stuff’s Current State

3 Assumptio
   • A Brief Description
   • A Glimpse at its Architecture
   • A Quick Example

4 BQE
   • Monniaux’s QE algorithm
   • BQE Algorithm
   • Conclusion

5 Questions

A. Champion
Assumptio and Stuff: using Z3 in a collaborative parallel formal verification framework.
Our questions

About Z3:

A. Champion
Our questions

About Z3:

- **Push / Pop** mechanism;

**QE Solver Stack for BQE**

$$\neg P(s) \land T(s, s')$$
Our questions

About Z3:
- **Push / Pop** mechanism;

QE Solver Stack for BQE

\[ G_i(s') \]

\[ \neg P(s) \land T(s, s') \]
Our questions

About Z3:

- **Push / Pop** mechanism;

QE Solver Stack for BQE

\[ \neg P(s) \land T(s, s^{'}) \]
Our questions

About Z3:
- **Push / Pop** mechanism;

**QE Solver Stack for BQE**

\[
G_{i+1}(s') \\
\neg P(s) \land T(s, s')
\]
Our questions

About Z3:
- **Push** / **Pop** mechanism;
- Using **let**-s or constraints;

For Example
- `assert(let x = F in P(x))`
- `assert(P(x))`  `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.


Patrick Cousot and Radhia Cousot.

Michael Dierkes.

Kenneth L. McMillan.

David Monniaux.

Martin Odersky and al.

Mary Sheeran, Satnam Singh, and Gunnar Stålmarck.

A. Champion

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