Quantifier Elimination: Concepts, Algorithms, Tools and Applications

Ratnesh Kumar, Professor, Fellow IEEE
Hao Ren, PhD Student
Iowa State University
Matthew Clark
Air Force Research Lab., Wright Patterson AFB
Quantifier Elimination

- **Quantifier Elimination** is a procedure of removing quantified variables and their quantifiers to obtain an equivalent formula.

E.g., \( \exists x (x^2 \leq y) \equiv \{ y | \exists x \text{ such that } x^2 \leq y \} \equiv (y \geq 0) \).

- A quantified formula in \( n \)-variables with \( m \)-free variables is a subset in \( m \)-dimensional space.

- Quantifier elimination involves **set projection**.

- General than SAT/SMT that only check for satisfiability, and provide a single point in projection.
Compositional Verification: A natural application

Property of \( System_i: \Phi_i(u_i, y_i), i = 1,2 \) (Eg, \( u_i \geq y_i \))

\( \Rightarrow \) property of composed system:
\[ \Phi(u_1, y_2) \equiv \exists x: \Phi_1(u_1, x) \land \Phi_2(x, y_2) \]

Eg, \( \exists x: (u_1 \geq x) \land (x \geq y_2) \equiv (u_1 \geq y_2), \) as expected
Application in Model Simplification

- Given definitions of individual blocks:
  \[
  \begin{align*}
  \text{delay:} & \quad y(k+1)_{\text{delay}} = d(k+1)_{\text{delay}} \\
  \text{delay:} & \quad d(k+1)_{\text{delay}} = y(k)_{\text{limit}} \\
  \text{sum:} & \quad y(k+1)_{\text{sum}} = 1 + y(k+1)_{\text{delay}} \\
  \text{limit:} & \quad y(k+1)_{\text{sum}} < 0 \Rightarrow y(k+1)_{\text{limit}} = 0 \\
  \text{limit:} & \quad y(k+1)_{\text{sum}} > 9 \Rightarrow y(k+1)_{\text{limit}} = 9 \\
  \text{limit:} & \quad 0 \leq y(k+1)_{\text{sum}} \leq 9 \Rightarrow y(k+1)_{\text{limit}} = y(k+1)_{\text{sum}}
  \end{align*}
  \]

- Derive “cycle-invariant”, i.e., difference eqn for block with feedback loop: \( y(k+1)_{\text{limit}} = f(y(k)_{\text{limit}}, u(k) = 1) \)

- Solution: Denoting for simplicity,
  \[
  \begin{align*}
  a & = y(k+1)_{\text{sum}}, \\
  b & = y(k+1)_{\text{limit}}, \\
  c & = y(k+1)_{\text{delay}}, \\
  d & = d(k+1)_{\text{delay}}, \\
  e & = y(k)_{\text{limit}}
  \end{align*}
  \]
  \[
  \begin{align*}
  \exists a, b, c: & \quad (a = c + 1) \land (a > 9 \Rightarrow b = 9) \land (0 \leq a \leq 9 \Rightarrow b = a) \land (c = d = e) \\
  \equiv & \quad (e \leq -1 \land b = 0) \lor (e = b - 1 \land 0 < b < 9) \lor (e \geq 8 \land b = 9) \\
  \equiv & \quad (y(k)_{\text{limit}} \leq -1 \land y(k+1)_{\text{limit}} = 0) \\
  \lor & \quad (y(k)_{\text{limit}} = y(k+1)_{\text{limit}} - 1 \land 0 < y(k+1)_{\text{limit}} < 9) \\
  \lor & \quad (y(k)_{\text{limit}} \geq 8 \land y(k+1)_{\text{limit}} = 9).
  \end{align*}
  \]
QE Tool: Redlog

- Redlog (Reduce Logic) implements multiple quantifier elimination procedures for various supported theories, and is part of interactive computer algebra system Reduce.
Integrating QE with Model-checker JKind

- **Jkind**, a tool by Rockwell, is java-based model-checker for safety verification of systems described in Lustre language, using induction.

\[
\begin{align*}
\text{Base: } & I(x_0) \land \bigwedge_{i=0}^{k-1} T(x_i, x_{i+1}) \implies \bigwedge_{i=0}^k \Phi(x_i) \\
\text{Induction: } & \bigwedge_{i=0}^k T(x_i, x_{i+1}) \land \bigwedge_{i=0}^k \Phi(x_i) \implies \Phi(x_{k+1})
\end{align*}
\]

\[\Rightarrow \forall i: \Phi(x_i)\]

- Each step verified by solving its SMT encoding. JKing uses Z3, Yices, smtinterpol as SMT solvers.
Reduction of SMT solving to QE

- **SMT** resolution is naturally an instance of quantifier elimination:
  - Formula $\Phi(\vec{v})$ is satisfiable iff $\exists \vec{v} \Phi(\vec{v}) \equiv true$
  - E.g. satisfiability of $(x + y = 10) \land (x - y = 2)$ can be checked either way:

SMT-solver Z3:

```plaintext
1 (declare-const x Int)
2 (declare-const y Int)
3 (assert (= (+ x y) 10))
4 (assert (= (- x y) 2))
5 (check-sat)
6 (get-model)
```

QE-solver Redlog:

```plaintext
1: rlset z$

*** turned on switch rlsusi

2: phi:= ex(x,ex(y, x+y=10 and x-y=2));

\phi:= \exists x \exists y(x + y - 10 = 0 \land x - y - 2 = 0)

3: rlqea phi;

{{true, \{y = 4, x = 6\}}}
```

- Z3 is good with solving mixed-integer linear problems; Redlog performs well in nonlinear domain.
Integrating Redlog with JKind

Jkind Architecture (left)

and its

flow with Z3 (below)

By-passing Z3 with Redlog

New flow with Redlog (dotted above)
Redlog-integrated JKind

- Worked out a **nonlinear** example, on which Jkind+Z3 didn’t terminate
  - Depending on the range of 4 inputs, output is computed using one among 54 different 4th-order polynomials of the inputs;
  - Need to verify if output is always below 1 in magnitude

```
warning disabling PDR due to non-linearities
-------------------------------

There are 2 properties to be checked.
PROPERTIES TO BE CHECKED: [a1, FISI-0.spl]

C:\\loorwerks\\jkind\\FLC4in_test.lus.bmc: sat
C:\\loorwerks\\jkind\\FLC4in_test.lus.k-induction: sat

INVALID PROPERTY: a1 || BMC || K - 1 || Time = 10.389

```

* verification results summary
  * Properties to be verified
    - Dist A: Unlimited Distribution, Case No: 88ABW-2016-3178
**Compositional Reasoning**

- System and Component contracts provided in “Assume-Guarantee” format

- Sequentially prove:
  - Each Component assumption is satisfied by System assumption and its upstream Components’ guarantees
  - System guarantee is satisfied by System assumptions and Component guarantees
  - \( n \)-components \( \Rightarrow \ n+1 \) proof-steps

Sequentially prove:

\[
\begin{align*}
A_S & \Rightarrow A_A \\
A_S \land G_A & \Rightarrow A_B \\
A_S \land G_A \land G_B & \Rightarrow A_C \\
A_S \land G_A \land G_B \land G_C & \Rightarrow G_S
\end{align*}
\]
AGREE Tool for Compositional Reasoning

- **AGREE**, another Rockwell tool, developed over **OSATE2**---an extension of Eclipse,
  - Supports AADL architectural models, with behaviors and contracts captured in AGREE annex, performs translation to Luster for JKind based proving.

**AGREE work flow:**

```
AGREE .aadl file .lus file .smt files
AGREE
JKind valid/invalid
SMT solver sat/unsat
```

**OSATE2:** An eclipse-based tool platform to support AADL and various plug-in tools

**Eclipse**
QE-integrated Compositional Reasoning

- Collects Components contracts and derive system’s strongest property by existentially quantifying all internal variables:
  \[ \Phi_{\text{strongest}} = \exists \nu_{\text{internal}} ((A_1 \Rightarrow G_1) \land (A_2 \Rightarrow G_2) \land \cdots \land (A_n \Rightarrow G_n)) \]

- Proves/disproves postulated property by checking if:
  \[ \Phi_{\text{postulated}} \equiv true \iff \forall \nu_{\text{system}} (\Phi_{\text{strongest}} \Rightarrow \Phi_{\text{postulated}}) \equiv true \]

Redlog-integrated AGREE work flow:

1. .aadl file
2. .txt file
3. AGREE
4. Redlog
5. Report results
6. valid/invalid
QE-based Property Composition Example

- **$i_s$** & **$o_s$**: system input & output; **$o_A$** & **$o_B$**: outputs of components A & B

- The component contracts are specified as:
  - $i_s < 20 \Rightarrow o_A < 2i_s$
  - $o_A < 20 \Rightarrow o_B < o_A + 15$
  - $o_s = o_A + o_B$

- Postulated system contract:
  - $i_s < 10 \Rightarrow o_s < 50$?

- Strongest System contract is existential quantification over internal variables ($o_A$ and $o_B$) of the conjunction of all the subsystem contracts:
  - $\exists o_A, o_B: [i_s < 20 \Rightarrow o_A < 2i_s] \land [o_A < 20 \Rightarrow o_B < o_A + 15] \land [o_s = o_A + o_B]$

  Upon quantifier elimination using Redlog tool we get (see next slides):

  - In integer domain: $i_s \leq 10 \Rightarrow o_s \leq 4i_s + 12$ (which implies the postulated system level contract: $[[i_s \leq 10 \Rightarrow o_s \leq 4i_s + 12] \Rightarrow [i_s < 10 \Rightarrow o_s < 50]] \equiv True$).
  - In real domain, a weaker system-level contract holds: $i_s < 10 \Rightarrow o_s < 4i_s + 15$
Redlog-integrated AGREE

- Over Integers, the System contract satisfied by Strongest property:
  \[ [i_s \leq 10 \Rightarrow o_s \leq 4i_s + 12] \Rightarrow [i_s < 10 \Rightarrow o_s < 50] \]
Over Reals, the Strongest property is weaker than System Contract:

$$[i_s < 10 \Rightarrow o_s < 4i_s + 15] \supseteq [i_s < 10 \Rightarrow o_s < 50]$$
Concluding Remarks

- Quantifier elimination can be used for property/model simplification

- Underlying theory for Compositional Verification:
  - Quantifier-Elimination Tools integration with model-checking/compositional-reasoning toolchain, and their demonstration
  - Future direction: Extension to Temporal and Real-Time logics
  - Thanks to Rockwell developers of JKind and AGREE