Partial Quantifier Elimination And Property Generation

Eugene Goldberg

CAV-2023, Paris, France

Outline

- Partial Quantifier Elimination (PQE)
- Property generation by PQE
- Two PQE algorithms
- Experiments

Some Basic Information

- We examine propositional formulas
- We consider only existential quantifiers
- All formulas are in Conjunctive Normal Form (CNF)
- CNF: conjunction $C_1 \bigwedge \ldots \bigwedge C_k \Leftrightarrow$ set $\{C_1, \ldots, C_k\}$
- A clause: $\neg x_1 \bigvee x_5 \bigvee y_{10}$

Partial Quantifier Elimination

Quantifier Elimination (**QE**):

Given $\exists X\,[F(X,Y)]$, find $H(Y)$ such that

$\exists X\,[F] \equiv H$

Goldberg, Manolios, HVC-14

Partial Quantifier Elimination (**PQE**):

Given $\exists X\,[F(X,Y)]$ and $G \subseteq F$, find $H(Y)$ such that

$\exists X\,[F] \equiv H \bigwedge \exists X\,[F \setminus G]$

H is called a **solution** to the PQE problem

PQE reduces to QE if $G = F$

Motivation

QE is ubiquitous but **inherently hard**

One approach: Use SAT, *a special case* of QE: $\exists X\,[F\,(X\,)]$

Benefit: SAT is efficient
Downside: loss of semantic power of quantifiers

An alternative: Use PQE, *a generalization* of QE
Benefits:

- PQE can be drastically more **efficient** than QE
- Many problems (e.g. SAT, equivalence and model checking) can be solved **in terms of PQE**, see the technical report
- One **gains** semantic power

Outline

- Partial Quantifier Elimination (PQE)
- Property generation by PQE
- Two PQE algorithms
- Experiments

Property Generation (motivation)

- Imp – design implementation, $P_1,..,P_k$ – specification properties
- Even if Imp satisfies $P_1,..,P_k$ it can be buggy (incomplete specification)
- Let $P_1,..,P_k$, $Q_{k+1},..,Q_m$ be **an imaginary complete specification**
- If Imp buggy, some Q_i fails i.e. **the unwanted property** $\neg Q_i$ holds

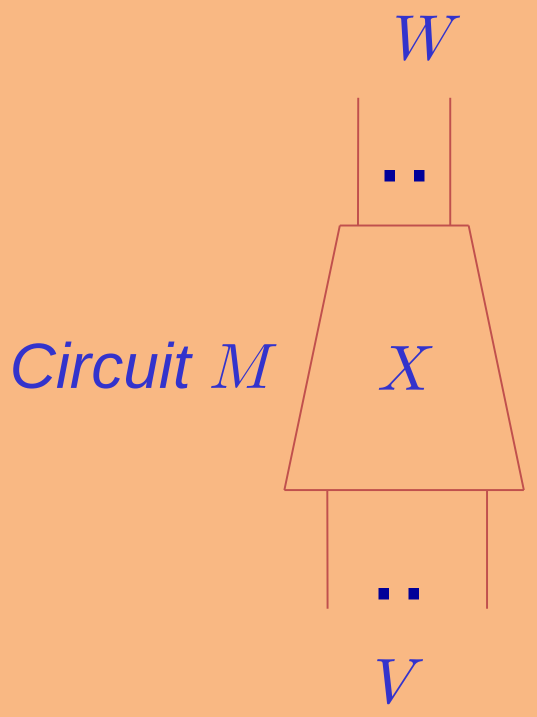
Two ways to detect bugs:

- A **desired predefined** property P_i **fails** (formal verification)
- An **unexpected and unwanted** property **holds** (testing)

Problems:

- testing examines very simple properties
- an unwanted property can be overlooked (e.g. $s_i \Rightarrow s_j$)

Property Generation By PQE



Consider **QE**: $\exists X\,[F] \equiv H$ where $F\,(X,V,W)$ specifies M.
$H(V,W)$ is the “truth table” (the ***strongest* property** of M)

Generating **weaker properties** $\boldsymbol{H\,(V,W)}$ by **PQE**

Take a clause C out of $\exists X\,[F]$:
$\exists X\,[F] \equiv H \;\bigwedge\; \exists X\,[F \setminus \{C\}]$.
$F \Rightarrow H$. So, H is a **property** of M.
Every clause of H is a property of M too.

PQE + clause splitting $\Rightarrow$ [single-test properties,.., truth table]

The Appeal of Property Generation By PQE

- More complex properties $\Rightarrow$ an overlooked **unwanted** property
- PQE is simpler than QE $\Rightarrow$ **efficient** property generation (for single-test properties the complexity of PQE is **linear**)
- Properties "**cover**" the entire implementation $\Rightarrow$ taking out a clause of the buggy part is likely to generate an unwanted property

Invariant Generation By PQE
(for sequential circuits)

Bug: a state $\vec{s}$ of sequential circuit N must be reachable but it is not.

Exposed by an **unwanted invariant** H where $H(\vec{s}) = 0$.

Invariant generation in 3 steps (steps 2,3 repeated in a loop).

Step1: Build $\exists X_k\,[F_k]$ where

F_k – describes unfolding of N for k time frames

X_k - all variables of F_k but S_k (the state variables of time frame k)

Step 2: Produce $H(S_k)$ by taking C out of $\exists X_k\,[F_k]$

where $\exists X_k\,[F_k] \equiv H \wedge \exists X_k\,[F_k \setminus \{C\}]$

Step 3: Check **every clause** of H if it is a **global invariant**

Outline

- Partial Quantifier Elimination (PQE)
- Property generation by PQE
- Two PQE algorithms
- Experiments

PQE Solver Named EG-PQE

(EG stands for 'Enumerate and Generalize')

EG-PQE is a very simple SAT-based algorithm

Given $\exists X\,[F(X,Y)]$, EG-PQE takes out a **single clause** $C \in F$.
C is called the **target** clause.

Redundancy based reasoning:

Build $H(Y)$ implied by F that makes C **redundant** in $H \wedge \exists X\,[F]$.
So, $\boldsymbol{\exists X\,[F] \equiv H \wedge \exists X\,[F \setminus \{C\}]}$ and H is a solution to PQE.

PQE Solver Named EG-PQE (cont.)

- EG-PQE enumerates assignments $\vec{y}$ to Y.
- It recognizes three sets of $\vec{y}$: A_{impl} , A_{unsat} , A_{sat}
- $\vec{y} \in A_{\mathrm{impl}}$ if $F \setminus \{C\} \Rightarrow C$ in subspace $\vec{y}$ (C is redundant)
- $\vec{y} \in A_{\mathrm{sat}}$ if F is satisfiable in subspace $\vec{y}$ (C is redundant)
- $\vec{y} \in A_{\mathrm{unsat}}$ if F is unsatisfiable in subspace $\vec{y}$ (C is not redundant)

  A Y-clause is added to H **to make** C redundant in $H \wedge \exists X[F]$

EG-PQE is similar to QE algorithm introduced by K.McMillan (CAV-02).
Difference: a) using redundancy based reasoning; b) employing A_{impl}.

From EG-PQE To EG-PQE+

Enumerating $\vec{y} \in (A_{\mathrm{unsat}} \cup A_{\mathrm{impl}})$ is relatively **easy**

Enumerating $\vec{y} \in (A_{\mathrm{sat}} \setminus A_{\mathrm{impl}})$ is **hard**

The reason: EG-PQE uses the *satisfiability of* F in subspace $\vec{y}$ to prove C redundant there. This proof is too strong. It proves *every clause* redundant in $\exists X[F]$ in subspace $\vec{y}$.

EG-PQE+ uses a weaker proof *meant only for* C .
EG-PQE+ is much more complex than EG-PQE.

Outline

- Partial Quantifier Elimination (PQE)
- Property generation by PQE
- Two PQE algorithms
- Experiments

Finding Bug In A FIFO Buffer

- **Correct FIFO buffer:** every state of the data buffer is reachable
- **Bug:** an element Val is not pushed into the buffer
- **An unwanted invariant:** states with Val in the buffer are unreachable
- **Invariant generation:** the 3-step procedure described earlier

Time limit is 10 sec. per PQE problem.
DS-PQE is the algorithm presented at HVC-14

number of elements	latches	time frames	unwanted invar.			run time (s.)		
			ds-pqe	eg-pqe	eg-pqe+	ds-pqe	eg-pqe	eg-pqe+
8	300	5	no	**yes**	**yes**	12,141	2,138	**52**
8	300	10	**yes**	**yes**	**yes**	5,551	7,681	**380**
16	560	5	no	no	**yes**	22,612	9,506	**50**
16	560	10	**yes**	no	**yes**	6,541	16,554	**153**

Other Experiments With PQE

- We experimented with 98 multi-property benchmarks of HWMCC-13. The number of latches ranged from 111 to 8,000
- We used PQE to generate invariants covering different parts of the design
- We showed that PQE can be much faster than QE and conducted many other experiments

A Few Takeaways

- PQE provides a way to achieve efficiency **without losing** the power of quantifiers

- Property generation by PQE helps to **plug the hole** caused by incompleteness of specification