Generation Of A Complete Set Of Properties

Eugene Goldberg
eu.goldberg@gmail.com

***Abstract*—One of the problems of formal verification is that it is not functionally complete. The fact that a set of properties of a specification holds for a design implementation, in general, does not mean that this implementation is bug-free. In testing, this issue is addressed by replacing functional completeness with *structural* one. The latter is achieved by generating a set of tests probing every piece of a design implementation. We show that a similar approach can be used in formal verification. The idea here is to generate a property of the implementation at hand that is not implied by *specification properties*. Finding such a property means that the specification is not complete. If this is an *unwanted* property, then the implementation is buggy. Otherwise, a new specification property needs to be added. Generation of implementation properties related to different parts of the design followed by adding new specification properties produces a structurally-complete specification. Implementation properties are built by *partial quantifier elimination*, a technique where only a part of the formula is taken out of the scope of quantifiers. An implementation property is generated by applying partial quantifier elimination to a formula defining the "truth table" of the implementation. We show how our approach works on specifications of combinational and sequential circuits.**

I. Introduction

One of the problems of formal verification is that it is functionally incomplete. Let us consider this problem by the example of combinational circuits. Suppose a set $\mathcal{P} = \{P_1(X,Z),\ldots,P_k(X,Z)\}$ of formulas[1] specify properties of a combinational circuit to be designed. Here X and Z are sets of input and output variables of this circuit respectively[2]. (A correct implementation has to exclude the input/output behaviors *falsifying* P_i, $i=1,\ldots,k$.) Let $N(X,Y,Z)$ be a combinational circuit implementing the *specification* $\mathcal{P}$ where Y is the set of internal variables. Let $F(X,Y,Z)$ be a formula describing the functionality of N. The circuit N satisfies property $P_i, i=1,\ldots,k$ iff $F \Rightarrow P_i$.

In general, $P_1 \wedge \cdots \wedge P_k$ does not necessarily imply $\exists Y[F]$ (where the latter specifies the input/output behavior of N). If so, then some input/output behaviors of N are not defined by $\mathcal{P}$ i.e. the latter is incomplete. Note that even *checking* the completeness of $\mathcal{P}$ is inherently hard. For instance, verifying the implication above requires performing quantifier elimination (**QE**) for $\exists Y[F]$.

In testing, the incompleteness of functional verification is addressed by using a set of tests that is complete *structurally* rather than functionally. Structural completeness is achieved by probing every piece of the design under test. In this paper, we use a similar approach for formal verification. This approach is based on two ideas. The first idea is to check the completeness of the specification $\mathcal{P}$ by generating *implementation* properties i.e. those satisfied by N. Let $Q(X,Z)$ be a property N (and so $F \Rightarrow Q$). If $P_1 \wedge \cdots \wedge P_k \not\Rightarrow Q$, then the specification $\mathcal{P}$ is *incomplete*. If Q is an unwanted property, then N is buggy (and it should be modified so that it does not satisfy Q). Otherwise, a new property should be added to the specification $\mathcal{P}$ to make the latter imply Q. A trivial way to achieve this goal is just to add to $\mathcal{P}$ the property Q itself.

The second idea is to generate implementation properties by a technique called **partial** QE (**PQE**) [4]. In terms of formula $\exists Y[F]$, PQE takes a subset of clauses of F out of the scope of quantifiers. (So QE is special case of PQE where the entire formula is taken out of the scope of quantifiers.) This results in generation of a formula $Q(X,Z)$ implied by F i.e. a property of N. Importantly, by taking different subsets of clauses of F out of the scope of quantifiers, one builds a *structurally complete set of properties*. By updating specification properties every time an implementation property proves $\mathcal{P}$ incomplete, one gets a structurally complete specification. By combining PQE with simple formula transformations (based on splitting clauses on variables), one can generate *any property* of N. Importantly, by varying the size of the subformula taken out of the scope of quantifiers one can control the complexity of PQE and hence that of property generation. The latter ranges from essentially *linear* (for trivial properties excluding wrong outputs for one particular input of N) to exponential.

Incompleteness of the specification $\mathcal{P}$ may lead to two kinds of bugs. A bug of the first kind that we mentioned above occurs when N has an unwanted property. In this case, N *excludes* some *correct* input/output behaviors. (An example of an unwanted property is given in Appendix A.) A bug of the second kind occurs when N *allows* some *incorrect* input/output behaviors. This type of bugs can be exposed by generating properties that are *inconsistent* with N. (As opposed to the implementation properties that are *consistent* with N by definition.) Such inconsistent properties are meant to imitate the missing properties of $\mathcal{P}$ that are not satisfied by N (if any). Tests falsifying inconsistent properties may expose incorrect input/output behaviors allowed by N. These properties can also be generated by PQE. Besides, one can follow the same idea of structural completeness by building a set of inconsistent properties relating to different parts of N. However, this topic is beyond the scope of this paper. (It is covered in [3].) So here, we consider generation of a

[1] In this paper, we consider only propositional formulas. We assume that every formula is in conjunctive-normal form (CNF). A *clause* is a disjunction of literals (where a literal of a Boolean variable w is either w itself or its negation $\overline{w}$). So a CNF formula H is a conjunction of clauses: $C_1 \wedge \cdots \wedge C_k$. We also consider H as the *set of clauses* $\{C_1,\ldots,C_k\}$.

[2] For the sake of simplicity, in the introduction we assume that properties $P_i(X,Z)$ depend on *all* input/output variables. In Section III, we consider a more general case where a property depends on a *subset* of $X \cup Z$.

specification that is structurally complete only with respect to *consistent* properties of the implementation at hand.

The contribution of this paper is as follows. First, we show that one can use PQE for generation of implementation properties. Second, we sketch an algorithm for generation of a structurally complete specification. Third, we show that by combining clause splitting with PQE one can generate an arbitrary implementation property of a combinational circuit. In particular, we prove that clause splitting allows one to reduce the complexity of PQE (and hence property checking) to virtually linear. The latter result also shows that QE can be exponentially more complex than PQE.

This paper is structured as follows. Basic definitions are given in Section II. In Section III, we describe generation of implementation properties of combinational circuits by PQE. A procedure for making a specification structurally complete is presented in Section IV. In Sections V and VI we extend our approach to sequential circuits. Some concluding remarks are made in Section VII.

II. Basic Definitions

Definition 1: Let V be a set of variables. An **assignment** $\vec{q}$ to V is a mapping $V' \rightarrow \{0,1\}$ where $V' \subseteq V$. We will refer to $\vec{q}$ as a **full assignment** to V if $V' = V$.

From now on, by saying "an assignment to a set of variables" we mean a *full* assignment, unless otherwise stated.

Definition 2: Let F be a formula. $\boldsymbol{Vars(F)}$ denotes the set of variables of F.

Definition 3: Let $H(V,W)$ be a formula where V, W are sets of Boolean variables. The **Quantifier Elimination (QE)** problem specified by $\exists V[H(V,W)]$ is to find formula $H^*(W)$ such that $\boldsymbol{H^* \equiv \exists V[H]}$.

Definition 4: Let $H_1(V,W)$, $H_2(V,W)$ be Boolean formulas where V, W are sets of Boolean variables. The **Partial QE** (**PQE**) problem of taking H_1 out of the scope of quantifiers in $\exists X[H_1(V,W) \wedge H_2(V,W)]$ is to find formula $H_1^*(W)$ such that $\boldsymbol{\exists X[H_1 \wedge H_2] \equiv H_1^* \wedge \exists X[H_2]}$. Formula H_1^* is called a **solution** to PQE.

Remark 1: Note that if H_1^* is a solution to the PQE problem above and a clause $C \in H_1^*$ is implied by H_2 *alone*, then $H_1^* \setminus \{C\}$ is a solution too. So if all clauses of H_1^* are implied by H_2, then an empty set of clauses is a solution too (in this case, $H_1^* \equiv 1$).

Let $N(X,Y,Z)$ be a combinational circuit where X, Y, Z are sets of input, internal and output variables respectively. Let N consist of gates $g_1, \ldots, g_k$. A formula $F(X,Y,Z)$ specifying the functionality of N can be built as $G_1 \wedge \cdots \wedge G_k$ where $G_i, 1 \leq i \leq k$ is a formula specifying gate g_i. Formula G_i is constructed as a conjunction of clauses falsified by the incorrect combinations of values assigned to G_i. Then every assignment satisfying G_i corresponds to a consistent assignment of values to g_i and vice versa.

Example 1: Let g be a 2-input AND gate specified by $v_3 = v_1 \wedge v_2$. Then formula G is constructed as $C_1 \wedge C_2 \wedge C_3$ where $C_1 = v_1 \vee \overline{v}_3$, $C_2 = v_2 \vee \overline{v}_3$, $C_3 = \overline{v}_1 \vee \overline{v}_2 \vee v_3$. Here, the clause C_1, for instance, is falsified by assignment $v_1 = 0, v_3 = 1$ that is inconsistent with the truth table of g.

III. Generation Of Implementation Properties

In this section, we describe generation of implementation properties by PQE and discuss how one can control the complexity of those properties. Let $N(X,Y,Z)$ be a combinational circuit where X, Y, Z are sets of input, internal and output variables. Let $F(X,Y,Z)$ be a formula specifying N.

Let H be a non-empty subset of clauses of F. Let F' denote $F \setminus H$ (and so $F = H \wedge F'$). Consider the PQE problem of taking H out of the scope of quantifiers in $\exists W[H \wedge F']$ where $Y \subseteq W \subset Vars(F)$. Let formula $Q(V)$ where $V = Vars(F) \setminus W$ be a solution to this problem i.e. $\exists W[H \wedge F'] \equiv Q \wedge \exists W[F']$. Since Q is implied by F, it is a **property** of the circuit N. Note that by taking different subsets of F out of the scope of quantifiers in $\exists W[F]$ one gets different properties. From now on, we will assume that all the clauses implied by F' are removed from Q (see Remark 1).

Intuitively, the smaller the size of H is, the simpler PQE becomes. So, the simplest case of the PQE problem above is when a single clause of F is taken out of the scope of quantifiers. However, the complexity of PQE can be reduced much more by using clause splitting to transform F.

Definition 5: Let $V = \{v_1, \ldots, v_m\}$ be a subset of $Vars(F)$. Let $l(v_1), \ldots, l(v_m)$ be a set of literals where $l(v_i), i = 1, \ldots, m$ is either v_i itself of $\overline{v}_i$. Let C be a clause of F such that $V \cap Vars(C) = \emptyset$. The **splitting** of C on variables of V is to replace C with clauses $C \vee l(v_1)$,..., $C \vee l(v_m)$, $C \vee \overline{l}(v_1) \vee \cdots \vee \overline{l}(v_m)$.

The idea here is to take the clause $C \vee \overline{l}(v_1) \vee \cdots \vee \overline{l}(v_m)$ out of the scope of quantifiers *instead* of C. In Appendix B, we show that such replacement can reduce the complexity of PQE to essentially **linear**. This also proves that PQE can be **exponentially simpler** than QE. In addition to making property generation simpler, clause splitting also helps to derive a richer set of properties. In Appendix C, we show that by applying PQE and clause splitting to $\exists W[F]$ one can derive **any given property** $Q(V)$ where $V = Vars(F) \setminus W$.

IV. Producing Complete Set Of Properties

In this section, we give an example of a procedure called *CmplSet* that generates a structurally complete specification. The pseudocode of *CmplSet* is shown in Figure 1. *CmplSet* accepts

- a specification $\mathcal{P}$ (i.e. a set of properties P_1,...,P_k)
- an "informal" specification $\mathcal{P}^{inf}$ used to decide if a property of N is unwanted
- an implementation $F(X,Y,Z)$ defining a circuit N
- the set of variables $V \subseteq (X \cup Z)$ on which implementation properties will depend on.

CmplSet returns an unwanted property of N exposing a bug (if any) or a structurally complete specification $\mathcal{P}$.

CmplSet starts with initializing a copy *Cls* of formula F (lines 1). Then *CmplSet* runs a 'while' loop until *Cls* is empty. *CmplSet* starts an iteration of the loop by extracting a clause

```
CmplSet(P, P^inf, F, V){
1  Cls := F
2  while (Cls ≠ ∅) {
3     C := PickCls(Cls)
4     Cls := Cls \ {C}
5     Q := PQE(F, C, V)
6     Clean(Q, F, C)
7     if (Impl(P, Q)) continue
8     if (Unwanted(P^inf, Q)) return(Q,nil)
9     P := SpecProp(P, P^inf, F, Q)
10    P := P ∪ {P}}
11 return(nil, P)}
```

Fig. 1. The *CmplSet* procedure

C from *Cls* (lines 3-4). Then it builds an implementation property $Q(V)$ as a solution to the PQE problem $\exists W[C \wedge F']$ where $F' = F \setminus \{C\}$ and $W = Vars(F) \setminus V$ (line 5). That is $\exists W[C \wedge F'] \equiv Q \wedge \exists W[F']$. One can view Q as a property 'probing' the part of N represented by C. Then *CmplSet* calls the procedure called *Clean* (line 6) to remove the clauses implied by F' from Q (see Remark 1). At this point Q consists only of clauses whose derivation depends on the clause C.

Then *CmplSet* checks if $P_1 \wedge \cdots \wedge P_k \Rightarrow Q$ (line 7). If so, then a new iteration starts. Otherwise, the current specification $\mathcal{P}$ is incomplete, which requires either modification of implementation N or specification $\mathcal{P}$. If Q is an unwanted property *CmplSet* returns it as a proof that N is buggy (line 8). (In this case, N excludes some correct input/output behaviors. To decide whether Q is unwanted, one needs some kind of an *informal* specification $\mathcal{P}^{inf}$ that is *complete*.) If Q is a desired property, *CmplSet* generates a new *specification* property P such that $P_1 \wedge \cdots \wedge P_k \wedge P \Rightarrow Q$ and adds it to $\mathcal{P}$ (lines 9-10). A trivial way to update $\mathcal{P}$ is just to use Q as a new specification property P. If *CmplSet* terminates the loop without finding a bug, it returns $\mathcal{P}$ as a structurally complete specification.

V. Extending Idea To Sequential Circuits

In this section and Section VI, we extend our approach to sequential circuits. Subsections V-A and V-B provide some definitions. Subsection V-C gives a high-level view of building a structurally complete specification for a sequential circuit (in terms of safety properties).

A. Some definitions

Let $M(S, X, Y, S')$ be a sequential circuit. Here X, Y denote input and internal combinational variables respectively and S, S' denote the present and next state variables respectively. Let $F(S, X, Y, S')$ be a formula describing the circuit M. (F is built for M in the same manner as for a combinational circuit N, see Section II.) Let $I(S)$ be a formula specifying the *initial states* of M. Let $T(S, S')$ denote $\exists X \exists Y[F]$ i.e. the *transition relation* of M.

A *state* $\vec{s}$ is an assignment to S. Any formula $P(S)$ is called a *safety property* for M. A state $\vec{s}$ is called a *P-state* if $P(\vec{s}) = 1$. A state $\vec{s}$ is called *reachable in n transitions* (or in n-th *time frame*) if there is a sequence of states $\vec{s}_1, \ldots, \vec{s}_{n+1}$ such that $\vec{s}_1$ is an I-state, $T(\vec{s}_i, \vec{s}_{i+1}) = 1$ for $i = 1, \ldots, n$ and $\vec{s}_{n+1}$=$\vec{s}$.

We will denote the *reachability diameter* of M with initial states I as $\boldsymbol{Diam(M, I)}$. That is if $n = Diam(M, I)$, every state of M is reachable from I-states in at most n transitions. We will denote as $\boldsymbol{Rch(M, I, n)}$ a formula specifying the set of states of M reachable from I-states in n transitions. We will denote as $\boldsymbol{Rch(M, I)}$ a formula specifying all states of M reachable from I-states. A property P *holds* for M with initial states I, if no $\overline{P}$-state is reachable from an I-state.

B. Stuttering

In the following explanation, we assume that the circuit M above has the **stuttering** feature. This means that $T(\vec{s}, \vec{s})$=1 for every state $\vec{s}$ and so M can stay in any given state arbitrarily long. If M does not have this feature, one can introduce stuttering by adding a combinational input variable v. The modified circuit works as before if $v = 1$ and remains in its current state if $v = 0$.

On one hand, introduction of stuttering does not affect the reachability of states of M. On the other hand, stuttering guarantees that the transition relation of M has two nice properties. First, $\exists S[T(S, S')] \equiv 1$, since for every next state $\vec{s}'$, there is a "stuttering transition" from $\vec{s}$ to $\vec{s}'$ where $\vec{s}$ = $\vec{s}'$. Second, if a state is unreachable in M in n transitions it is also unreachable in i transitions if $i < n$. Conversely, if a state is reachable in M in n transitions, it is also reachable in i transitions where $i > n$.

Remark 2: Note that for a circuit M with the stuttering feature, formula $Rch(M, I, n)$ specifies not only the states reachable in n transitions but also those reachable in *at most* n transitions.

C. High-level view

In this paper, we consider a specification of the sequential circuit M above in terms of safety properties. So, when we say a specification property $P(S)$ of M we *mean a safety property*. Let $F_{1,i}$ denote $F_1 \wedge \cdots \wedge F_i$ where F_j, $1 \leq j \leq i$ is the formula F in j-th time frame i.e. expressed in terms of sets of variables S_j, X_j, Y_j, S_{j+1}. Formula $Rch(M, I, n)$ can be computed by QE on formula $\exists W_{1,n}[I_1 \wedge F_{1,n}]$. Here $I_1 = I(S_1)$ and $W_{1,n} = Vars(F_{1,n}) \setminus S_{n+1}$. If $n \geq Diam(M, I)$, then $Rch(M, I, n)$ is also $Rch(M, I)$ specifying all states of M reachable from I-states.

Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be a set of properties forming a *specification* of a sequential circuit with initial states defined by I. Let a sequential circuit M be an implementation of the specification $\mathcal{P}$. So every property P_i,$i = 1, \ldots, k$ holds for M and I. Verifying the completeness of $\mathcal{P}$ reduces to checking if $P_1 \wedge \cdots \wedge P_k \Rightarrow Rch(M, I)$. Assume that computing $Rch(M, I)$ is hard. So one does not know if the specification $\mathcal{P}$ is complete. Then one can use the approach described in the previous sections to form a specification that is complete *structurally* rather than functionally.

We exploit here the same idea of using PQE to compute properties of M i.e. *implementation* properties. Let Q be such a property. If $P_1 \wedge \cdots \wedge P_k \not\Rightarrow Q$, then the specification $\mathcal{P}$ is *incomplete*. If some states of M falsifying Q (and hence

unreachable from I-states) should be reachable, then M is buggy and must be modified. Otherwise, one needs to update $\mathcal{P}$ by adding a specification property P to guarantee that $P_1 \wedge \cdots \wedge P_k \wedge P \Rightarrow Q$. The simplest way to achieve this goal is just to add Q to $\mathcal{P}$. Using a procedure similar to that shown in Fig. 1 one can construct a structurally complete specification.

VI. Generation Of Safety Properties

In this section, we continue using the notation of the previous section. Here, we discuss generation of properties for a sequential circuit $M(S, X, Y, S')$, i.e. *implementation* properties. Subsection VI-A considers the case where the reachability diameter $Diam(M, I)$ is known. (In [2] we showed that one can use PQE to find $Diam(M, I)$ without generation of all reachable states.) Subsection VI-B describes an approach to generation of properties when $Diam(M, I)$ is not known.

A. The case of known reachability diameter

As we mentioned in Subsection V-C, formula $Rch(M, I)$ can be obtained by QE on $\exists W_{1,n}[I_1 \wedge F_{1,n}]$ where $n \geq Diam(M, I)$. Here $I_1 = I(S_1)$, $F_{1,n} = F_1 \wedge \cdots \wedge F_n$, and $W_{1,n} = Vars(F_{1,n}) \setminus S_{n+1}$.

Below we show how one can build a property of M by PQE. Let C be a clause of $F_{1,n}$. Let $Q(S_{n+1})$ be a solution to the PQE problem of taking C out of the scope of quantifiers in $\exists W_{1,n}[I_1 \wedge C \wedge F'_{1,n}]$ where $F'_{1,n} = F_{1,n} \setminus \{C\}$. That is $\exists W_{1,n}[I_1 \wedge C \wedge F'_{1,n}] \equiv Q \wedge \exists W_{1,n}[I_1 \wedge F'_{1,n}]$. Let us show that Q is a *property* of M. Let $\vec{s}$ be a state falsifying Q i.e. $\vec{s}$ in unreachable from an I-state in n transitions. On one hand, since M has the stuttering feature, $\vec{s}$ cannot be reached in i transitions where $i \leq n$. On the other hand, since $n \geq Diam(M, I)$, $\vec{s}$ cannot be reached in i transitions where $i > n$. So all states falsifying Q are unreachable and thus Q is a property of M. By taking different clauses of $F_{1,n}$ out of the scope of quantifiers one can generate different implementation properties. Following a procedure similar to that of Fig. 1, one can generate a specification of M that is structurally complete.

B. The case of unknown reachability diameter

Suppose that the reachability diameter of M is unknown. Then one needs to modify the procedure of the previous subsection as follows. Let $Q(S_{n+1})$ be a solution to the PQE problem of taking C out of the scope of quantifiers in $\exists W_{1,n}[I_1 \wedge C \wedge F'_{1,n}]$ where $F'_{1,n} = F_{1,n} \setminus \{C\}$. Assume that $n < Diam(M, I)$. Then Q is not a property of M. One can only guarantee that the states falsifying Q cannot be reached in at most n transitions.

```
MakeInv(F, I, Q){
1 while (true) }
2    (Cex, Q) := MC(F, I, Q)
3    if (Cex = nil) return(Q)
4    Q := Relax(Q, Cex)
5    if (Q ≡ 1) return(Q)}}
```

Fig. 2. The *MakeInv* procedure

One can turn Q into a property by using procedure *MakeInv* shown in Figure 2. *MakeInv* runs a while loop (lines 1-5). First, *MakeInv* calls a model checker *MC* (e.g. IC3 [1]) to prove property Q. If *MC* succeeds, *MakeInv* returns Q as a property of M. Otherwise, *MC* finds a counterexample *Cex*. This means that a state $\vec{s}$ falsifying Q (and thus unreachable in at most n transitions) is reachable in i transitions where $i > n$. Then one needs to relax Q by replacing it with a property implied by Q but not falsified by $\vec{s}$.

One way to relax Q is to replace it with a solution $R(S')$ to the PQE problem of taking $\exists W[Q(S) \wedge F(S, X, Y, S')]$ out of the scope of quantifiers where $W = X \cup Y \cup S$. That is $\exists W[Q \wedge F] \equiv R \wedge \exists W[F]$. Since the circuit M has the stuttering feature, $\exists W[F] \equiv 1$. So R just specifies the set of states reachable from Q-states in one transition. If $\vec{s}$ still falsifies R one can use PQE to find the set of states reachable from R-states and so on. If $\vec{s}$ does not falsify R, the latter is used as a new formula Q (line 4). If relaxation ends up with a trivial property, *MakeInv* terminates (line 5). Otherwise a new iteration starts.

By taking different clauses of $F_{1,n}$ out of the scope of quantifiers in $\exists W_{1,n}[I_1 \wedge F_{1,n}]$ one can generate different properties of the circuit M.

VII. Conclusions

Incompleteness of a specification *Spec* creates two problems. First, an implementation *Impl* of *Spec* may have some *unwanted* properties that *Spec* does not ban. Second, *Impl* may break some *desired* properties that are not in *Spec*. In either case, *Spec* fails to expose bugs of *Impl*. In testing, the problem of functional incompleteness is addressed by running a test set that is complete *structurally* rather than functionally. This structural completeness is achieved by generating tests probing every piece of *Impl*. We apply this idea to formal verification. Namely, we show that by using a technique called partial quantifier elimination (PQE) one can generate properties probing different parts of *Impl*. By checking that no property of *Impl* generated by PQE is unwanted one addresses the first problem above. By updating *Spec* to make it imply the desired properties of *Impl* generated by PQE one builds a specification that is structurally complete. One can use a similar approach to address the second problem above [3].

References

[1] A. R. Bradley. Sat-based model checking without unrolling. In *VMCAI*, pages 70–87, 2011.

[2] E. Goldberg. Property checking without invariant generation. Technical Report arXiv:1602.05829 [cs.LO], 2016.

[3] E. Goldberg. On verifying designs with incomplete specification. Technical Report arXiv:2004.09503 [cs.LO], 2020.

[4] E. Goldberg and P. Manolios. Partial quantifier elimination. In *Proc. of HVC-14*, pages 148–164. Springer-Verlag, 2014.

Appendix A
Unwanted Property Derived By PQE

In this appendix, we give an example of an unwanted property derived by PQE. Consider the design of a combinational

circuit called a *sorter*. It accepts m r-bit numbers ranging from 0 to 2^r-1, sorts them, and outputs the result. Let X and Z be sets of input and output variables of the sorter respectively. Let $\mathbb{x}_1,\ldots,\mathbb{x}_m$ and $\mathbb{z}_1,\ldots,\mathbb{z}_m$ be numbers specified by input $\vec{x}$ and output $\vec{z}$ respectively. The properties $P'(X)$ and $P''(X,Z)$ below form a *complete* specification of the sorter.

- $P'(\vec{z})=1$ iff $\mathbb{z}_1\le\cdots\le\mathbb{z}_m$,
- $P''(\vec{x},\vec{z})=1$ iff $\mathbb{z}_1,\ldots,\mathbb{z}_m$ is a permutation of $\mathbb{x}_1,\ldots,\mathbb{x}_m$.

Let the designer use an *incomplete* specification $\mathcal{P}$ consisting only of the property P'. Let $N(X,Y,Z)$ be an *implementation* of the sorter and $F(X,Y,Z)$ be a formula describing the functionality of N. Assume $F\Rightarrow P'$ i.e. N satisfies the specification $\mathcal{P}$. Suppose N is buggy. Namely, let $\mathbb{z}_1=0$ for every output of N. (This does not contradict $F\Rightarrow P'$, since $\mathbb{z}_i\ge 0$, $1<i\le m$.) Then N has a property Q falsified by the outputs $\vec{z}$ where $\mathbb{z}_1=b$ and b is a constant $1\le b\le 2^r-1$.

Suppose Q is obtained by taking $C\in F$ out of the scope of quantifiers in $\exists Y[F]$ i.e. by PQE. On one hand, $P'\not\Rightarrow Q$. Indeed, P' is satisfied by an assignment $\vec{z}$ where $\mathbb{z}_1,\ldots,\mathbb{z}_m$ are sorted and $\mathbb{z}_1=b$. So derivation of Q proves $\mathcal{P}$ incomplete. On the other hand, Q is an *unwanted* property of N. In a correct sorter, $\mathbb{z}_1$ can take any value from 0 to 2^r-1. So derivation of Q exposes a hole in $\mathcal{P}$ and proves N buggy.

Appendix B
Generation Of Simple properties

In this appendix, we show that clause splitting can reduce the complexity of PQE and, hence, property generation to essentially **linear**. For the sake of simplicity, in our exposition, we use a particular clause of an AND gate of N (also explaining how this exposition can be extended to an arbitrary clause of an arbitrary gate).

Let g be an AND gate of a combinational circuit N whose functionality is described by $v_3=v_1\wedge v_2$ (see Example 1). Let clause $C\in F$ be equal to $\overline{v}_1\vee\overline{v_2}\vee v_3$. This clause forces assigning the output variable v_3 of g to 1 when the input variables v_1 and v_2 of g are assigned 1. Let $Vars(C)\cap X=\emptyset$. (In the general case, $C\in F$ is one of the clauses specifying a gate g of N. The clause C has one variable specifying the output of g. The remaining variables of C correspond to the input variables of g.) Consider splitting C on the variables of $X=\{x_1,\ldots,x_m\}$. That is C is replaced in F with $m+1$ clauses $C\vee l(x_1),\ldots,C\vee l(x_m),C\vee\overline{l}(x_1)\vee\cdots\vee\overline{l}(x_m)$.

Let C' denote the clause $C\vee\overline{l}(x_1)\vee\cdots\vee\overline{l}(x_m)$ above. Let F' denote $F\setminus\{C'\}$. Consider the PQE problem of taking C' out of the scope of quantifiers in $\exists Y[C'\wedge F']$. Now we describe a procedure called *QuickPQE* that solves the PQE problem above. (The proposition below proves *QuickPQE* correct.) Let $\vec{x}\,'$ denote the assignment to X falsifying the literals $\overline{l}(x_1),\ldots,\overline{l}(x_m)$ of C'.

QuickPQE starts with applying $\vec{x}\,'$ to N. Let $\vec{z}\,'$ be the output assignment produced by N for $\vec{x}\,'$. Suppose that v_1 and/or v_2 are assigned 0 when computing $\vec{z}\,'$. (In the general case, this means that the clause C and hence the clause C' is satisfied by an assignment to an *input* variable of the gate g.) Then *QuickPQE* declares C' redundant claiming that $\exists Y[C'\wedge F']\equiv\exists Y[F']$.

If both v_1 and v_2 are assigned 1, then *QuickPQE* performs one more run. (In the general case, this means that the literals of C corresponding to the input variables of the gate g are falsified.) In this run, *QuickPQE* also applies input $\vec{x}\,'$ but modifies the operation of the gate g. Namely, g produces the output value 0 (instead of the value 1 implied by assignment $v_1=1,v_2=1$). Note that in the second run, the clause C' is falsified. One can view the second run as applied to the circuit N whose functionality is modified by removing the clause C'. If the second run produces the same output assignment $\vec{z}\,'$, then *QuickPQE* again declares C' redundant. Now, suppose that N outputs an assignment $\vec{z^*}$ different from $\vec{z}\,'$. Then, *QuickPQE* produces a solution consisting of clauses $B^{\vec{x}'}\vee l(z_1),\ldots,B^{\vec{x}'}\vee l(z_p)$ where

- $B^{\vec{x}'}=\overline{l}(x_1)\vee\cdots\vee\overline{l}(x_m)$ (i.e. $B^{\vec{x}'}$ is the longest clause falsified by $\vec{x}\,'$);
- $z_1,\ldots,z_p$ are the output variables of N assigned *differently* in $\vec{z}\,'$ and $\vec{z^*}$;
- $l(z_1),\ldots,l(z_p)$ are literals satisfied by $\vec{z}\,'$ (and falsified by $\vec{z^*}$).

Proposition 1: Let $C\in F$ be one the clauses specifying a gate g of N. Let $Vars(C)\cap X=\emptyset$. Let C be replaced with with the clauses $C\vee l(x_1),\ldots,C\vee l(x_m),C\vee\overline{l}(x_1)\vee\cdots\vee\overline{l}(x_m)$ obtained by splitting C on X. Denote the last clause as C' and the formula $F\setminus\{C'\}$ as F'. Consider the PQE problem of taking C' out of the scope of quantifiers in $\exists Y[C'\wedge F']$. Let $Nlits(F)$ denote the number of literals of F. The *QuickPQE* procedure above has complexity $\mathcal{O}(Nlits(F)+|X|*|Z|)$ and produces a correct result.

Proof: The complexity of *QuickPQE* is linear in $Nlits(F)$ because the former performs two test runs, each run having linear complexity in the number of literals of F. The term $|X|*|Z|$ is due to the fact that the solution produced by *QuickPQE* may consist of $|Z|$ clauses of $|X|+1$ literals (see above).

Now, let us show that *QuickPQE* produces a correct solution. Let $w\in Y\cup Z$ denote the output variable of the gate g. Assume for the sake of clarity that C contains the positive literal of w. So, in the second run of *QuickPQE* (where C' and C are falsified) the value of w is set to 0.

Denote the solution produced by *QuickPQE* as $Q(X,Z)$ and so $\exists Y[C'\wedge F']\equiv Q\wedge\exists Y[F']$. We prove this equivalence by showing that $C'\wedge F'$ and $Q\wedge F'$ are equisatisfiable for every assignment $(\vec{x},\vec{z})$ to $X\cup Z$. (Recall that X and Z specify the input and output variables of the circuit N.) Below, we consider the three possible cases.

Case 1. *In the first run of QuickPQE, an* ***input*** *variable of the gate g is assigned the value satisfying the clause C (and hence the clause C').* In this case $Q\equiv 1$. So one needs to show that for every assignment $(\vec{x},\vec{z})$ formulas $C'\wedge F'$ and F' are equisatisfiable. Consider the following two subcases.

(a) $\vec{x}\neq\vec{x}\,'$. Then C' is satisfied by $\vec{x}$ and so $C'\wedge F'$ and F' are logically equivalent in subspace $(\vec{x},\vec{z})$.

(b) $\vec{x} = \vec{x}\,'$. In this case, C' is satisfied by an assignment to an input variable of the gate g. The latter is true because the execution trace of N under input $\vec{x}$ can be obtained by Boolean Constraint Operation (BCP) in subspace $\vec{x}$ over formula $C' \wedge F'$. The fact that BCP leads to satisfying C' means that a clause implying C' in subspace $\vec{x}$ can be derived by resolving[3] clauses of F'. This means that C' is implied by formula F' in subspace $\vec{x}$. So $C' \wedge F'$ and F' are logically equivalent in subspace ($\vec{x}$,$\vec{z}$).

Case 2. *In the first run of QuickPQE, all **input** variables of the gate g are assigned values **falsifying** C. In the second run of QuickPQE, N outputs **the same** assignment $\vec{z}\,'$ as in the first run*. In this case, like in the first case, $Q \equiv 1$. So one needs to show that for every assignment ($\vec{x}$,$\vec{z}$) formulas $C' \wedge F'$ and F' are equisatisfiable. Consider the following three subcases.

(a) $\vec{x} \neq \vec{x}\,'$. Then C' is satisfied by $\vec{x}$ and so $C' \wedge F'$ and F' are logically equivalent in subspace ($\vec{x}$,$\vec{z}$).

(b) $\vec{x} = \vec{x}\,'$ and $\vec{z} \neq \vec{z}\,'$. Let us show that in this case both $C' \wedge F'$ and F' are unsatisfiable in subspace ($\vec{x}$,$\vec{z}$). Let $z_i \in Z$ be a variable assigned differently in $\vec{z}$ and $\vec{z}\,'$. Let $l(z_i)$ be the literal of z_i falsified by $\vec{z}$ (and satisfied by $\vec{z}\,'$). The fact that N outputs $\vec{z}\,'$ under input $\vec{x}\,'$ means that F' implies the clause $B^{\vec{x}'} \vee l(z_i)$. So $C' \wedge F'$ is falsified in subspace ($\vec{x}$,$\vec{z}$). The fact that N outputs $\vec{z}\,'$ in both runs means that F' implies clauses $B^{\vec{x}'} \vee l(z_i) \vee \overline{w}$ and $B^{\vec{x}'} \vee l(z_i) \vee w$. (Recall that the variable w specifies the output of the gate g. The variable w is assigned 1 in the first run to satisfy C' because the literals of all other variables of C' are falsified. The variable w is assigned 0 in the second run.) So F' implies the resolvent of these two clauses equal to $B^{\vec{x}'} \vee l(z_i)$. Hence F' is falsified in subspace ($\vec{x}$,$\vec{z}$) too.

(c) $\vec{x} = \vec{x}\,'$ and $\vec{z} = \vec{z}\,'$. Let us show that in this case $C' \wedge F'$ and F' are both satisfiable in subspace ($\vec{x}$,$\vec{z}$). Let $\vec{p}$ be the assignment to the variables of N produced in the first run of *QuickPQE*. By definition, the assignment to $X \cup Z$ in $\vec{p}$ is the same as in ($\vec{x}\,'$,$\vec{z}\,'$) and hence in ($\vec{x}$,$\vec{z}$). Besides, $\vec{p}$ satisfies $C' \wedge F'$ and hence F'.

Case 3. *In the first run of QuickPQE, all **input** variables of the gate g are assigned values **falsifying** C. In the second run of QuickPQE, N outputs an assignment $\vec{z^*}$ that is **different** from the assignment $\vec{z}\,'$ output in the second run of QuickPQE*. In this case, the solution $Q(X,Z)$ consists of the clauses $B^{\vec{x}'} \vee l(z_1)$,...,$B^{\vec{x}'} \vee l(z_p)$ where $\{z_1,\ldots,z_p\}$ is the set of variables assigned differently in $\vec{z}\,'$ and $\vec{z^*}$. So one needs to show that for every assignment ($\vec{x}$,$\vec{z}$) formulas $C' \wedge F'$ and $Q \wedge F'$ are equisatisfiable. Consider the following four subcases. (We denote the set of variables where $\vec{z}\,'$ and $\vec{z^*}$ have the same value as Z^*.)

(a) $\vec{x} \neq \vec{x}\,'$. Then C' and Q are satisfied by $\vec{x}$. So $C' \wedge F'$ and $Q \wedge F'$ are logically equivalent in subspace ($\vec{x}$,$\vec{z}$).

(b) $\vec{x} = \vec{x}\,'$ and there is a variable $z_i \in Z^*$ that is assigned in $\vec{z}$ differently than in $\vec{z}\,'$. Then both $C' \wedge F'$ and $Q \wedge F'$ are unsatisfiable in subspace ($\vec{x}$,$\vec{z}$). This can be shown as in case 2b above.

(c) $\vec{x} = \vec{x}\,'$ and all variables of Z^* are assigned the same value in $\vec{z}$ and $\vec{z}\,'$ and there is a variable $z_i \in (Z \setminus Z^*)$ that is assigned in $\vec{z}$ as in $\vec{z^*}$ (i.e. differently from $\vec{z}\,'$). Let us show that in this case both $C' \wedge F'$ and $Q \wedge F'$ are unsatisfiable in subspace ($\vec{x}$,$\vec{z}$). The formula $C' \wedge F'$ is falsified because it implies the clause $B^{\vec{x}'} \vee l(z_i)$ that is falsified by ($\vec{x}$,$\vec{z}$). The formula $Q \wedge F'$ is falsified by ($\vec{x}$,$\vec{z}$) because it contains the clause $B^{\vec{x}'} \vee l(z_i)$.

(d) $\vec{x} = \vec{x}\,'$ and $\vec{z} = \vec{z}\,'$. Let us show that in this case $C' \wedge F'$ and $Q \wedge F'$ are both satisfiable in subspace ($\vec{x}$,$\vec{z}$). Let $\vec{p}$ be the assignment to the variables of N produced in the first run of *QuickPQE*. By definition, $\vec{p}$ agrees with assignment ($\vec{x}\,'$,$\vec{z}\,'$) and hence with ($\vec{x}$,$\vec{z}$). Besides, $\vec{p}$ satisfies $C' \wedge F'$ and hence F'. Since $\vec{p}$ also satisfies Q it satisfies $Q \wedge F'$ as well.

[3]Let clauses C',C'' have opposite literals of exactly one variable $w \in Vars(C') \cap Vars(C'')$. Then clauses C',C'' are called *resolvable* on w. The clause C having all literals of C', C'' but those of w is called the *resolvent* of C',C'' on w. The clause C is said to be obtained by *resolution* on w.

APPENDIX C
ANY PROPERTY CAN BE GENERATED BY PQE

Let $F(X,Y,Z)$ be a formula specifying a combinational circuit $N(X,Y,Z)$. In this appendix, we prove that by combining clause splitting with PQE one can generate any property of N. The procedure used in this proof is not very realistic because it requires an exponential blow-up of the formula. Nevertheless, our result has some practical meaning, which is twofold. First, clause splitting significantly *extends* the set of properties that can be generated by PQE. Second, there is a set of properties *natural* to N that can be *efficiently* produced by clause splitting and PQE. Those are the properties that can be generated without an exorbitant amount of clause splitting.

Let $Q(V)$ be a property of N where $V \subseteq (X \cup Z)$. So $F \Rightarrow Q$. Let us show that Q can be obtained by applying PQE to formula $\exists W[G]$ where $W = Vars(G) \setminus V$ and G is obtained from F by clause splitting. (So $Vars(G) = Vars(F) = X \cup Y \cup Z$.) Formula G is obtained by splitting the clauses of F on variables of W that is different from that of Definition 5. Namely, a clause $C \in F$ is replaced with $2^{|V|}$ clauses $C \vee B^{\vec{v}}$ where $\vec{v}$ is an assignment to V and $B^{\vec{v}}$ is the longest clause falsified by $\vec{v}$. Note that if C and $B^{\vec{v}}$ have opposite literals of a variable $w \in (Vars(C) \cap V)$ then the clause $C \vee B^{\vec{v}}$ is tautologous and can be removed from the formula G. Denote by G_1 (respectively G_2) the clauses $C \vee B^{\vec{v}}$ where $\vec{v}$ falsifies (respectively satisfies) the property Q.

Proposition 2: The property Q above is a solution to the PQE problem of taking G_1 out of the scope of quantifiers in $\exists W[G_1 \wedge G_2]$. That is $\exists W[G_1 \wedge G_2] \equiv Q \wedge \exists W[G_2]$.

Proof: One needs to show that for every assignment $\vec{v}$ to V formulas, $G_1 \wedge G_2$ and $Q \wedge G_2$ are equisatisfiable in subspace $\vec{v}$. Consider the following two cases.

- $Q(\vec{v}) = 0$. By construction, $G_1 \wedge G_2$ is logically equivalent to F. The fact that $F \Rightarrow Q$ entails that $G_1 \wedge G_2$ is unsatisfiable in subspace $\vec{v}$. So both $G_1 \wedge G_2$ and $Q \wedge \exists W[G_2]$ are unsatisfiable in subspace $\vec{v}$.

- $Q(\vec{v}) = 1$. By construction, every clause $C \wedge B^{\vec{v}'}$ of G_1 is satisfied by $\vec{v}$ (because $\vec{v} \neq \vec{v}'$). So $G_1 \wedge G_2$ and $Q \wedge \exists W[G_2]$ are logically equivalent in subspace $\vec{v}$.