Some Definitions of Vacuity

Dana Fisman

DRAFT - This is a working document for sv-ac subcommittee.

Abstract

This is a working document for sv-ac subcommittee. It presents some definitions of vacuity for temporal logic that are applicable to linear-time logics (taken from literature on the subject). The aim is to (a) asses if we are really ready to provide a formal definition of vacuity in the LRM, in view of the fact that there is no consensus in the literature on what the right definition of vacuity is; and (b) if we decide to include a definition on vacuity, we should be aware of the attempts of doing so this far.

Notations

In this document we use D for a *design*, i.e. a finite state machine that could be written in some hardware description language, e.g. SystemVerilog. We use `f`, `g` and `h` for temporal logic *formulas*, e.g. expressions written in some temporal logic, e.g. SVA. We assume a set P of *atomic propositions*, usually P is the set of input/output/internal/auxiliary signals of a design. We use Σ to abbreviate 2^P. Thus Σ holds all possible valuations of the design signals. Σ is referred to as the *alphabet*. We use w, u, v to denote finite/infinite *words* over Σ. Such words usually represents computations/trances of a design or prefixes of their of. For a temporal logic formula `f` and a word w we use $w \models$ `f` to denote that `f` *holds* on w (equivalently, w *satisfies* `f`). We use the term *property* to refer to a set of finite/infinite words. Thus a formula `f` defines a property — the set $\{w \mid w \models$ `f`$\}$ of words satisfying it. Since our attention is focused on linear time logics, we can regard D as a set of words over Σ, and use $D \models$ `f` to denote that `f` holds on every word $w \in D$. In which case we say that D *satisfies* `f` (equivalently, that `f` *holds* on D).

1 Formula Vacuity [2],[6]

The first definition for vacuity was proposed in [2]. According to their definition, formula `f` is said to be *vacuous* in a design D if $D \models$ `f`, and there exists a sub-

formula g of f that does not affect the value of f in D. A sub-formula g of f *does not affect* f in D iff for every formula h, $D \models$ f iff $D \models$ f[g $\leftarrow$ h] where f[g $\leftarrow$ h] is obtained by replacing the sub-formula g in f with h. In [6] it was shown that it is enough to replace the subformula g with `true` and `false` — if $D \models$ f[g $\leftarrow$ `false`] iff $D \models$ f[g $\leftarrow$ `true`] then g does not affect f in M.[1]

Examples

1. `always (req implies s_eventually grant)`
   - This formula holds vacuously if `req` always holds (in this case `s_eventually grant` does not affect)
   - This formula holds vacuously if `s_eventually grant` always holds (in this case `req` does not affect)
2. `req or s_eventually grant`
   - This formula holds vacuously if `req` holds now (in this case `s_eventually grant` does not affect)
   - This formula holds vacuously if `s_eventually grant` holds now (in this case `req` does not affect)
3. `busy s_until ack`
   - This formula holds vacuously if `ack` holds now (in this case `busy` does not affect)
4. `busy until ack`
   - This formula holds vacuously if `ack` holds now (in this case `busy` does not affect)
   - This formula holds vacuously if `busy` always holds (in this case `ack` does not affect)

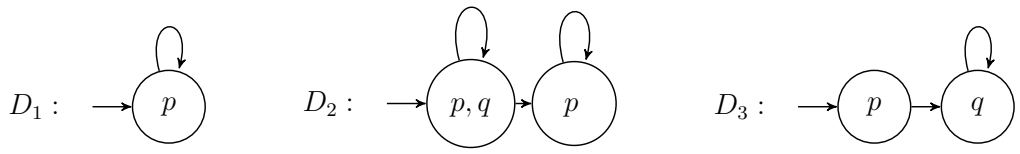
2 Trace Vacuity [1]

In [1] it was argued that the definition of [2,6] is not suited for formulas with multiple occurrences of subformulas. They showed that for such formulas the definition of [2,6] is sensitive to changes in the design that do not relate to the formula as well as to changes in the specification language.

To see the sensitivity with respect to the design, consider the formula `f = p implies s_nexttime p` and the designs D_1 and D_2 in the figure below. The formula f holds on both. Note that D_2 is an extension of D_1 with a new proposition q that is independent on the the behavior of p. Thus we expect a formula that relates only to p to have the same vacuous result on both D_1 and D_2. However, according to [2] p

[1] As will be discussed below, this is true only for sub-formula that occur only once in f, or to replacing a single occurrence of the sub-formula.

does not affect f in D_1, yet it does affect f in D_2 since $D_2 \not\models \mathtt{f}[\mathtt{p} \leftarrow \mathtt{q}]$. Thus f is vacuous on the design D_1 but not on D_2.



To see the sensitivity with respect to the specification language, consider the formula $\mathtt{h} = \mathtt{s\_nexttime\ q} \rightarrow \mathtt{s\_nexttime\ s\_nexttime\ q}$ and the design D_3 of the figure above. The formula f holds on D_3. Moreover, for any formula g of LTL the formula $\mathtt{h}[\mathtt{q} \leftarrow \mathtt{g}]$ holds on D_3 as well. However, if we consider an extension of LTL with the previous operator $\mathtt{\$past}()$ this is no longer true. This is since $D_3 \not\models \mathtt{h}[\mathtt{q} \leftarrow \mathtt{\$past(p)}]$. Thus h holds vacuously when considering LTL as the specification language but holds non-vacuously when the specification is LTL extended with $\mathtt{\$past}()$.

To overcome these unwanted sensitivities they propose to define vacuity using a trace assignment to a fresh proposition. Formally, they define that a subformula g *does not affect* formula f in M iff $\forall x.\ \mathtt{f}[\mathtt{g} \leftarrow x]$ where $\forall x$ checks x with respect to all possible trace assignments $\tau_x : \mathbb{N} \mapsto \{\mathtt{true}, \mathtt{false}\}$.

Under this definition f holds non-vacuously on both D_1 and D_2 (since e.g. under the trace assignment $\tau_x(n) = \mathtt{true}$ iff $n \leq 4$ we have $D_1 \not\models \mathtt{f}[\mathtt{p} \leftarrow x]$ thus $D_1 \not\models \forall x.\mathtt{f}[\mathtt{p} \leftarrow x]$ and similarly for D_2). And h holds non-vacuously on D_3 whether the specification is LTL or LTL extended with $\mathtt{\$past}$.

3 Parameterized Vacuity [7,8]

In [7] it was argued that intuitively a formula is vacuously satisfied if there is a subformula of it that can be strengthened without affecting the truth value. This work was motivated by a question Pnueli asked in CAV97 where [2] was presented. Pnueli observed that the formula `always s_eventually p` will hold non-vacuously on a design D where `always p` holds (since `s_eventually p` affects the truth value of `always s_eventually p` in D). Another example is the formula `s_nexttime s_eventually p` which is trivially satisfied in a design where the stronger formula `s_nexttime p` holds. In [7] it is proposed to check vacuity with respect to a set of vacuity grounds the user has defined. In [8] it is proposed to check for vacuity with respect to grounds up to a given size, and to try to find interesting grounds using several heuristics. A first step in providing a systematic way to find a strongest formula that satisfies the design is taken in [5] which uses lattice automata for the task.

4 Temporal Antecedent Failure [3]

In [4] it is argued that in many cases vacuities detected according to existing definitions are not considered a problem by the verifier of the system. For example, consider the formula `f` = `always (req implies (s_eventually ack))`, and a design is in which `ack` is given whenever the system is ready. Thus, if the system behaves correctly, it infinitely often gets into a "ready" state, and the formula `s_eventually ack` always holds. Accordingly, by [2,6,1], `f` holds vacuously in the design (since `req` does not affect the truth value of `f` in the design), although no real problem exists.

To overcome this it was proposed in [3] to focus on a certain type of vacuity, that according to experience is always considered a problem by the verifier. The proposed definition is termed *Temporal Antecedent Failure (TAF)* and it carries with it an efficient algorithm for both detecting TAF, and finding its reason.

A property may suffer from TAF if it can be characterized by a formula of *temporal implication form*, that is a formula of the form `always (f implies g)` where `f` is a past formula and `g` is a future formula. Loosely speaking, a formula is a past formula if it does not refer to the strict future. A formula is a future formula if it does not refer to the strict past. In SVA we can characterize such properties by `always (S |-> f)` where `S` is sequence and `f` is a future formula.

1. `always (a ##1 b[*1:$] ##1 c) |=> s_eventually grant)`
   - This formula suffers from TAF if the sequence `(a ##1 b[*1:$] ##1 c)` is never matched.
   - The reasons could be either `a` (if it never occurs) or `b` (if it never follows an `a`) or `c` (if it never follows a sequence matching `(a ##1 b[*1:$])`.
2. `always (req implies s_eventually grant)`
   - This formula suffers from TAF if `req` never holds (in this case `req` is the reason
   - This formula does not suffer from TAF if `s_eventually grant` always holds.
3. `always (!req or s_eventually grant)`
   - This formula suffers from TAF if `req` never holds (in this case `req` is the reason
   - This formula does not suffer from TAF if `s_eventually grant` always holds.
4. ```
   always
       ((req and s_nexttime req and s_nexttime s_nexttime req)
           implies (s_nexttime grant))
   ```
   - This formula does not suffer from TAF since the requirement is expected before the evaluation of the antecedent is completed.
5. `always`
   ```

```
((req and s_nexttime req and s_nexttime s_nexttime req)
    implies (s_nexttime s_nexttime grant))
```

– This formula suffers from TAF if `(req[*3])` is never matched, since it can be expressed as `always (req[*3] |-> grant)`.

References

[1] R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M. Vardi. Enhanced vacuity detection in linear temporal logic. In *Proc. 15th International Conference on Computer Aided Verification (CAV 2003)*, volume 2725 of *LNCS*, pages 368–380. Springer, 2003.

[2] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Detection of vacuity in ACTL formulas. In *Proc. 9th International Conference on Computer Aided Verification (CAV 1997)*, volume 1254 of *LNCS*, pages 279–290. Springer, 1997.

[3] S. Ben-David, D. Fisman, and S. Ruah. Temporal antecedent failure: Refining vacuity. In *CONCUR*, volume 4703 of *Lecture Notes in Computer Science*, pages 492–506, 2007.

[4] M. Chechik, M. Gheorghiu, and A. Gurfinkel. Finding environment guarantees. In M. B. Dwyer and A. Lopes, editors, *FASE*, volume 4422 of *Lecture Notes in Computer Science*, pages 352–367. Springer, 2007.

[5] H. Chockler, A. Gurfinkel, and O. Strichman. Beyond vacuity: Towards the strongest passing formula. In *FMCAD*, pages 1–8, 2008.

[6] O. Kupferman and M. Vardi. Vacuity detection in temporal model checking. In *Proc. 10th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME 1999)*, volume 1703 of *LNCS*, pages 82–96. Springer, 1999.

[7] M. Samer and H. Veith. Parameterized vacuity. In *Proc. 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2004)*, volume 3312 of *LNCS*, pages 322–336. Springer, 2004.

[8] M. Samer and H. Veith. On the notion of vacuous truth. In *LPAR*, pages 2–14, 2007.