Supporting Materials for Inherent Vacuity for GR(1) Specifications

Shahar Maoz
Tel Aviv University
Tel Aviv, Israel

Rafi Shalom
Tel Aviv University
Tel Aviv, Israel

***Abstract*—This document provides supporting materials for the ESEC/FSE'20 paper by the authors titled "Inherent Vacuity for GR(1) Specifications" [1]. We provide additional observations and examples for how cases we did not define as a vacuity in the paper, indeed do not preserve LTL semantics.**

I. Examples

Below we present a comprehensive set of example specifications with specification elements and domain values that should *not* be considered vacuous. Note that the following exaples are meant to demonstrate a mathematical property. We do not consider them to be similar to specifications written by engineers. Moreover, note that we sometimes use LTL formulas with integer (non-binary) variables without explicitly translating them, e.g., we may write $\mathbf{GF}\ x \geq 2$ where the domain of x is $\{1, 2, 3\}$ without using two binary variables for x.

A. Examples for Specification Elements

First, recall that our GR(1) vacuity cases definitions are asymmetric in their premise-sets. While system elements have environment elements in their premise-sets, environment elements that are not justices do not have system elements in their premise-sets. This asymmetry is a consequence of GR(1)'s asymmetry between the environment and the system. Example 1 below demonstrates that it would be incorrect to include initial system assertions in premise-sets of initial environment elements.

Example 1 (Environment initial assertion implied by system elements: not a vacuity)**.** *Consider a specification $\langle\{a\}, \emptyset, D, M_e, M_s\rangle$ with one Boolean environment variable a, and $B_{M_e} = B_{M_s} = \{a\}$. In this specification, the initial assumption $a \in B_{M_e}$ is implied by the initial guarantee $a \in B_{M_s}$. The specification is realizable, but if we remove the assumption, it becomes unrealizable, clearly not preserving the semantics.*

Second, note that safety element vacuities are defined using a propositional logic implication, without the temporal operator $\mathbf{G}$ (or any other temporal operator). Example 2 below demonstrates that it would be incorrect to consider implication between safeties with the temporal operator $\mathbf{G}$, i.e., to use LTL implication rather than propositional implication.

Example 2 (LTL implication of safeties by other safeties: not a vacuity)**.** *Consider the specification $\langle\{a, a_1\}, \emptyset, D, M_e, M_s\rangle$, where a and a_1 are Boolean environment variables, $B_{M_e} = \{\mathbf{G}\,(a_1 \rightarrow \mathbf{X}a), \mathbf{G}\,\neg a, \mathbf{G}\,\neg a_1\}$, and $B_{M_s} = \{\mathbf{G}\,\neg a_1\}$. In this specification, the assumption $\mathbf{G}\,\neg a_1 \in B_{M_e}$ is LTL-implied by the other two safety assumptions. The specification is realizable, but if we remove this assumption, the specification becomes unrealizable. This is the case because unlike before, the environment can fail the system by setting a_1 to true. Clearly, the semantics has changed.*

Example 2 demonstrates that LTL implication (that of infinite traces) fails for vacuity even if the implication is strictly from safeties, i.e., no justices are involved. However, it is restricted to a safety assumption that is LTL-implied by safety assumptions. Example 3 shows a simple specification for which the LTL equivalence of the strict realizability formula does not hold for safety guarantees that are LTL-implied by other safety guarantees[1].

Example 3 (LTL implication of safety guarantees: not a vacuity (1))**.** *Consider the specification $\langle\{a\}, \{b\}, D, M_e, M_s\rangle$, where a and b are Boolean variables, $B_{M_e} = \{\mathbf{G}\,a\}$, and $B_{M_s} = \{\mathbf{G}\,b, \mathbf{G}\,\mathbf{X}b\}$. Then $\mathbf{G}\,b \rightarrow_{LTL} \mathbf{G}\,\mathbf{X}b$. Since there are no initial assertions and justices, the strict realizability formula of the specification is easily simplified to $\varphi^{sr} = \mathbf{G}((\mathbf{H}\,a) \rightarrow (b \wedge \mathbf{X}\,b))$ and that of the specification without $\mathbf{G}\,\mathbf{X}b$ to $\varphi^{sr}_{\underline{v}} = \mathbf{G}((\mathbf{H}\,a) \rightarrow b)$. Consider also the computation $\sigma = \{a, b\}\emptyset^{\omega}$, then $\sigma, 0 \models \mathbf{H}\,a$ and $\sigma, 0 \models b$ but $\sigma, 0 \not\models b \wedge \mathbf{X}\,b$. Note that this means $\sigma, 0 \not\models (\mathbf{H}\,a) \rightarrow (b \wedge \mathbf{X}\,b)$ thus $\sigma \not\models \mathbf{G}((\mathbf{H}\,a) \rightarrow (b \wedge \mathbf{X}\,b))$ and we have $\sigma \not\models \varphi^{sr}$. On the other hand $\sigma, 0 \models (\mathbf{H}\,a) \rightarrow b$ and since $\sigma, i \not\models \mathbf{H}\,a$ for all $i \geq 1$ we have $\sigma, i \models (\mathbf{H}\,a) \rightarrow b$ for all $i \geq 0$, thus by definition of operator $\mathbf{G}$, $\sigma \models \mathbf{G}((\mathbf{H}\,a) \rightarrow b)$ or equivalently $\sigma \models \varphi^{sr}_{\underline{v}}$, which proves that $\varphi^{sr} \not\equiv_{LTL} \varphi^{sr}_{\underline{v}}$.*

Example 4 has a more complicated specification for the same case. It is a stronger example because it shows that the realizability of the specification is not always preserved.

Example 4 (LTL implication of safety guarantees: not a vacuity (2))**.** *Consider the specification $\langle\{a, a_1\}, \{b\}, D, M_e, M_s\rangle$, where a and a_1 are Boolean, $D(b) = \{1, 2, 3\}$, and $B_{M_e} = \{\mathbf{G}\,\neg a, \mathbf{G}\,(b = 1 \rightarrow \mathbf{X}a)\}$, and $B_{M_s} = \{\mathbf{G}\,(b < 3 \rightarrow (b < \mathbf{X}b \wedge \mathbf{X}b < 3)), \mathbf{G}\,(b = 3), \mathbf{G}\,(b = 3 \rightarrow \mathbf{X}a_1)\}$. Then $\mathbf{G}\,(b = 3)$ is LTL-implied by $\mathbf{G}\,(b < 3 \rightarrow (b < \mathbf{X}b \wedge \mathbf{X}b < 3))$. If we remove $\mathbf{G}\,(b = 3)$ from B_{M_s}, the originally unrealizable specification*

[1]We thank Or Pistiner for providing Example 3.

becomes realizable, because now the system can output $b = 1$ instead of $b = 3$. Clearly, the semantics is not preserved.

By changing $\mathbf{G}\ (b = 3)$ to $b = 3$ in the above example, initial assertions that are LTL-implied by safeties are not vacuous.

One may wonder if safeties could be implied by initial assertions as well as safeties. However, since Examples 2, 3 and 4 prohibit LTL implication for safeties, the propositional implication of safeties from initials would allow declaring $\mathbf{G}\ a$ vacuous because of a, which is evidently undesirable.

Finally, Examples 5,6 and 7 show that initial assertions propositionally implied by safeties should not be defined as vacuous. Example 5 is a cross module example, and Examples 6 and 7 show that such implications within the environment and system modules respectively do not preserve the semantics

Example 5 (Initial assertions propositionally implied by safeties: not a vacuity)**.** *Consider a specification $\langle\{a\}, \{b\}, D, M_e, M_s\rangle$, where a and b are Boolean, $B_{M_e} = \{\mathbf{G}\ b\}$, and $B_{M_s} = \{b, \mathbf{G}\ a\}$. Then $b \in B_{M_s}$ is propositionally implied by b, which is the propositional part of $\mathbf{G}\ b \in B_{M_e}$. The specification is unrealizable, but if we remove the guarantee b, it becomes realizable because the system can win with the initial assignment $\neg b$. Clearly, the semantics is not preserved.*

Example 6 (Initial assertions propositionally implied by safeties within the environment module: not a vacuity)**.** *Consider a specification $\langle\{a\}, \emptyset, D, M_e, M_s\rangle$, where a is Boolean, $B_{M_e} = \{a, \mathbf{G}\ a\}$, and $B_{M_s} = \{a\}$. Then $a \in B_{M_e}$ is propositionally implied by a, which is the propositional part of $\mathbf{G}\ a \in B_{M_e}$. The specification is realizable, but if we remove the assumption $a \in B_{M_e}$, it becomes unrealizable. Clearly, the semantics is not preserved.*

Example 7 (Initial assertions propositionally implied by safeties within the system module: not a vacuity)**.** *Consider a specification $\langle\{a\}, \{b\}, D, M_e, M_s\rangle$, where a and b are Boolean, $B_{M_e} = \{\mathbf{G}\ b\}$, and $B_{M_s} = \{b, \mathbf{G}\ b, b \rightarrow a\}$. Then $b \in B_{M_s}$ is propositionally implied by b, which is the propositional part of $\mathbf{G}\ b \in B_{M_s}$. The specification is unrealizable, but if we remove the guarantee b, it becomes realizable. Clearly, the semantics is not preserved.*

B. Domain Values Examples

Example 8 demonstrates why LTL implication within a module is not useful for unreachable domain values, as the variable a cannot be false in the environment module even without the assumption $\mathbf{G}\ \neg a$.

Example 8 (LTL-implied domain value: not a vacuity)**.** *Consider the specification $\langle\{a\}, \emptyset, D, M_e, M_s\rangle$, where a is Boolean, $B_{M_e} = \{\mathbf{G}\ \neg a, \mathbf{G}(a \rightarrow \mathbf{X}\ a), \mathbf{GF}\ \neg a\}$, and $B_{M_s} = \{\mathbf{G}\ \neg a\}$. Then $\mathbf{G}\ \neg a \in B_{M_e}$ is LTL-implied by the conjunction of $\mathbf{GF}\ \neg a$, which means that a must be false infinitely often, and by $\mathbf{G}(a \rightarrow \mathbf{X}\ a)$, which means that if a is true at some point it must remain true forever. Yet, if we remove the assumption $\mathbf{G}\ \neg a$, the system cannot ensure its identical guarantee, and the original realizable specification becomes unrealizable. Clearly, the semantics is not preserved.*

Finally, domain value vacuities are limited to variables of the same module (i.e., environment variables values are considered only using safety assumptions, and system variables values are considered only using safety guarantees). Even though they are safeties and Theorem 1 in the paper does not limit safeties to module variables, it would be incorrect to declare cross-module variables unreachable, because the limitation should be imposed on the module that has the variable, as Example 9 demonstrates.

Example 9 (Environment module domain value implied by the system module: not a vacuity)**.** *Consider a specification $\langle\{a\}, \emptyset, D, M_e, M_s\rangle$, with a single environment variable a whose domain is $D(a) = \{1, 2, 3\}$, $B_{M_e} = \emptyset$, and $B_{M_s} = \{\mathbf{G}\,(a = 1 \vee a = 2)\}$. According this specification, the value 3 is unreachable according to the system module. When removed (or equivalently we declare $B_{M_e} = \{\mathbf{G}\ a \neq 3\}$) the specification becomes realizable because the environment cannot fail the system by assigning $a = 3$.*

References

[1] S. Maoz and R. Shalom. Inherent Vacuity for GR(1) Specifications. In *ESEC/FSE*. ACM, 2020.