Vacuity Checking in the Modal Mu-Calculus

Yifei Dong, Beata Sarna-Starosta, C.R. Ramakrishnan, and Scott A. Smolka

Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY, 11794-4400, USA
E-mail:{ydong,bss,cram,sas}@cs.sunysb.edu

Abstract. Vacuity arises when a logical formula is trivially true in a given model due, for example, to antecedent failure. Beer et al. have recently introduced a logic-independent notion of vacuity and shown that certain logics, i.e., those with polarity, admit an efficient decision procedure for vacuity detection. We show that the modal mu-calculus, a very expressive temporal logic, is a logic with polarity and hence the results of Beer et al. are applicable. We also extend the definition of vacuity to achieve a new notion of *redundancy* in logical formulas. Redundancy captures several forms of antecedent failure that escape traditional vacuity analysis, including vacuous actions in temporal modalities and unnecessarily strong temporal operators. Furthermore, we have implemented an efficient redundancy checker for the modal mu-calculus in the context of the XMC model checker. Our checker generates *diagnostic information* in the form of all *maximal* subformulas that are redundant and exploits the fact that XMC can cache intermediate results in *memo tables* between model-checking runs. We have applied our redundancy checker to a number of previously published case studies, and found instances of redundancy that have gone unnoticed till now. These findings provide compelling evidence of the importance of redundancy detection in the design process.

1 Introduction

Model checking [7, 19, 8] is a verification technique aimed at determining whether a system specification possesses a property expressed as a temporal-logic formula. Model checking has enjoyed wide success in verifying, or finding design errors in, real-life systems. An interesting account of a number of these success stories can be found in [9, 15].

Most model checkers produce a *counter example* when the system under investigation does not satisfy a given temporal-logic formula. Such a counter example typically takes the form of an execution trace of the system leading to the violation of the formula. Until recently, however, standard practice was to move on to another formula when the model checker returned a result of true.

Starting with [3] and continuing with [4, 17, 5], researchers have been working on the problem of "suspecting a positive answer." In particular, this pioneering work shows how to detect situations of *vacuity*, where a model checker returns

true but the formula is vacuously true in the model. Vacuity can indicate a serious underlying flaw in the formula itself or in the model. The most basic form of vacuity is *antecedent failure*, where a formula containing an implication proves true because the antecedent, or pre-condition, of the implication is never satisfied in the model. The work of [4, 17, 5] has shown that this notion extends in a natural way to various temporal logics. This work also considers the problem of generating "interesting witnesses" to valid formulas that are not vacuously true.

The point that we would like to reinforce in the present paper is that a vacuously true formula may represent a design problem as serious in nature as a model-checking result of false. In fact, we present compelling anecdotal evidence that *redundancy checking*—a more encompassing form of vacuity detection that we discuss below—is just as important as model checking, and a model-checking process that does not include redundancy checking is inherently incomplete and suspect.

Our investigation of redundancy checking has been conducted in the context of XMC [20, 21], a model checker for systems described in XL, a highly expressive extension of value-passing CCS [18], and properties given as formulas of the modal mu-calculus. We have both implemented a redundancy checker for XMC that extends the approaches put forth [17, 5] in several significant ways, and assessed its performance and utility on several substantive case studies. Our main contributions can be summarized as follows.

- We extend the vacuity-checking results of [17, 5] to the modal mu-calculus [16], a very expressive temporal logic whose expressive power supersedes that of CTL* and related logics such as CTL and LTL (Section 2). In particular, [5] presents an effective procedure for vacuity checking for any logic with *polarity*. We prove that the modal mu-calculus is also a logic with polarity and therefore the results of [5] apply.
- We introduce a notion called *redundancy* that extends vacuity and identifies forms of antecedent failure that escape traditional vacuity analysis (Section 3). For example, consider the modal mu-calculus formula $[a, b]\phi$, meaning that in the given model, ϕ holds necessarily after every a- and b-transition. This formula is trivially true if the formula $[-]\phi$ also holds, meaning that regardless of the type of transition taken, necessarily ϕ.
- The algorithm of [5] pursues a bottom-up approach to vacuity checking in logics with polarity, replacing minimal subformulas by true or false. In contrast, our algorithm returns all *maximal* redundant subformulas. Such information can assist in isolating and correcting sources of redundancy in a model and its logical requirements. Our algorithm for redundancy checking is described in Section 4.
- XMC uses the XSB logic-programming system [23] as its underlying resolution engine, which, in turn implements tabled resolution. It is therefore possible when using XMC to cache intermediate results in *memo tables* between model-checking runs. As pointed out in [5], this is likely to significantly improve the performance of a vacuity checker, as the process of vacuity de-

2

tection for a logic with polarity requires a series of model-checking runs, all involving the same model and highly similar formulas. Our performance results bear out this conjecture (Section 5).

– We have applied our redundancy checker to a number of previously published case studies, and found instances of redundancy that have gone unnoticed till now (Section 5). These case studies include the Rether real-time ethernet protocol [13]; the Java meta-locking algorithm for ensuring mutually exclusive access by threads to object monitor queues [2]; and the safety-critical part of a railway signaling system [10]. These findings provide compelling evidence of the importance of redundancy detection in the design process.

2 Vacuity in the Modal Mu-Calculus

In [5], Beer et al. describe an efficient procedure for vacuity checking in logics with *polarity*. In this section, we first show that the modal mu-calculus is a logic with polarity, and hence the results of [5] are applicable. We then expand the notion of vacuity to one of *redundancy* to handle subtle forms of antecedent failure that are not considered vacuous. We begin by recalling the pertinent definitions and results from [5].

2.1 Vacuity and Logics with Polarity

We use φ, possibly with subscripts and primes, to denote formulas in a given logic, and ψ to denote subformulas of a formula. In the following, identical subformulas that occur at different positions in a formula are considered to be distinct. We use $\varphi[\psi \leftarrow \psi']$ to denote the formula obtained from φ by replacing subformula ψ with ψ'.

Consider a formula φ of the form $\psi_1 \Rightarrow \psi_2$ and a model M such that ψ_1 *does not hold* in M. Note that φ holds in M (due to antecedent failure) regardless of the whether ψ_2 holds or not. This form of "redundancy" is captured by the following definitions.

Definition 1 (Affect [5]) *A subformula ψ of formula φ affects φ in model M if there is a formula ψ', such that the truth values of φ and $\varphi[\psi \leftarrow \psi']$ are different in M.*

Definition 2 (Vacuity [5]) *Formula φ is said to be ψ-vacuous in model M if there is a subformula ψ of φ such that ψ does not affect φ in M.*

Although Definitions 1 and 2 capture a notion of vacuity independently of any particular logic, they cannot be directly used to *check* vacuity. Beer et al. describe a procedure to check vacuity of formulas in logics with polarity, where the candidate ψ' chosen as a replacement in Definition 1 is drawn from $\{true, false\}$.

These ideas are formalized in [5] as follows. Let S be a set of subformulas of φ and $\min(S)$ be the subset of S of formulas that are minimal with respect to

the (syntactic) subformula preorder. Also, let $\llbracket \varphi \rrbracket$ denote the set of all models in which formula φ is valid ($\llbracket \varphi \rrbracket = \{M | M \vDash \varphi\}$).

Definition 3 (Polarity of an operand [5]) *If σ is an n-ary operator in a logic,* the *i*-th operand of σ has positive (negative) polarity *if for every formula $\varphi_1, \ldots, \varphi_{i-1}, \varphi_{i+1}, \ldots, \varphi_n$, and two formulas ψ_1 and ψ_2 such that $\llbracket \psi_1 \rrbracket \subseteq \llbracket \psi_2 \rrbracket$ ($\llbracket \psi_2 \rrbracket \subseteq \llbracket \psi_1 \rrbracket$) we have that*

$$\llbracket \sigma(\varphi_1, \ldots, \varphi_{i-1}, \psi_1, \varphi_{i+1}, \ldots, \varphi_n) \rrbracket \subseteq \llbracket \sigma(\varphi_1, \ldots, \varphi_{i-1}, \psi_2, \varphi_{i+1}, \ldots, \varphi_n) \rrbracket$$

An operator has polarity if every one of its operands has some polarity (positive or negative). The notion of polarity is extended from operands to formulas as follows. The top-level formula φ is assumed to have positive polarity. Let ψ be some subformula of φ with $\circ$ as its top-level operator. Let ψ_i be the *i*-th operand of $\circ$ in ψ. Then ψ_i has positive polarity if the polarities of ψ and the *i*-th operand of $\circ$ are identical (i.e. both positive or both negative); ψ_i has negative polarity otherwise. For instance, let $\varphi = \psi_1 \wedge \neg(\psi_2 \vee \neg\psi_3)$. Note that both operands of '$\wedge$' and '$\vee$' have positive polarity and the operand of '$\neg$' has negative polarity. The subformulas of φ have the following polarities: φ, ψ_1 and ψ_3 have positive polarities; ψ_2 has negative polarity.

Definition 4 (Logic with polarity [5]) *A* logic with polarity *is a logic in which every operator has polarity.*

Theorem 5 ([5]) *In a logic with polarity, for a formula φ and a set S of subformulas of φ, for every model M, the following are equivalent:*

1. *φ is S-vacuous in M*
2. *There is a ψ such that:*

$$M \vDash \varphi \iff M \vDash \varphi[\psi \leftarrow \perp_\psi]$$

   where
   - *$\perp_\psi = false$ if $M \vDash \varphi$ and ψ is of positive polarity, or*
               $M \nvDash \varphi$ and ψ is of negative polarity;
   - *$\perp_\psi = true$ otherwise*

Theorem 5 directly yields a procedure to check vacuity of a formula with complexity $O(|\varphi| \cdot C_M(|\varphi|))$, where $C_M(n)$ denotes the complexity of model checking a formula of size n with respect to model M.

2.2 The Modal Mu-Calculus is a Logic with Polarity

In order to be able to apply the vacuity-detection procedure of [5] to mu-calculus formulas, we need to show that every operator in the mu-calculus has a polarity. Following [6], the syntax of modal mu-calculus formulas over a set of variable

names Var, a set of atomic propositions $Prop$, and a set of labels $\mathcal{L}$ is given by the following grammar:

$$\varphi \rightarrow \quad Prop \mid Var \mid \varphi \wedge \varphi \mid \neg\varphi \mid [a]\varphi \text{ where } a \in \mathcal{L}$$
$$\mid \nu Z.\varphi \text{ if every free occurrence of } Z \text{ in } \varphi \text{ is positive}$$

Additional derived operators, such as $\langle\cdot\rangle$, $\vee$ and μ (the duals of $[\cdot]$, $\wedge$ and ν, respectively), are introduced for convenience. Moreover, the $\langle\cdot\rangle$ and $[\cdot]$ modalities may be specified with *sets* of action labels: $[S]\varphi$ stands for $\bigwedge_{a \in S}[a]\varphi$, and $[-S]\varphi$ stands for $\bigwedge_{a \notin S}[a]\varphi$.

A mu-calculus *structure* $\mathcal{T}$ (over $Prop$, $\mathcal{L}$) is a labeled transition system with set $\mathcal{S}$ of states and transition relation $\Rightarrow\, \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ (also written as $s \xrightarrow{a} t$), together with an interpretation $\mathcal{V}_{Prop} : Prop \Rightarrow \wp\mathcal{S}$ for the atomic propositions.

Proposition 6 *The modal mu-calculus is a logic with polarity.*

Proof. The polarity of $\wedge$ and $\neg$ has been shown in [5]. It remains to show that operators $[a]$ and ν also have polarity.

Polarity of $[a]$. Given a mu-calculus structure $\mathcal{T}$ and an interpretation $\mathcal{V}$: $Var \Rightarrow \wp\mathcal{S}$ of the variables, the set of states satisfying $[a]\varphi$ is defined as:

$$[\![[a]\varphi]\!]^{\mathcal{T}}_{\mathcal{V}} = \{s \mid \forall t. s \xrightarrow{a} t \Rightarrow t \in [\![\varphi]\!]^{\mathcal{T}}_{\mathcal{V}}\}$$

For φ_1, φ_2 such that $[\![\varphi_1]\!] \subseteq [\![\varphi_2]\!]$, $[\![[a]\varphi_1]\!] = \{s \mid \forall t. s \xrightarrow{a} t \Rightarrow t \in [\![\varphi_1]\!]\}$. But this means that $t \in [\![\varphi_2]\!]$ and, thus, $s \in [\![[a]\varphi_2]\!]$. Therefore, $[\![[a]\varphi_1]\!] \subseteq [\![[a]\varphi_2]\!]$, and so, $[a]$ has positive polarity.

Polarity of ν. Similarly, the meaning of $\nu Z.\varphi$ is given as:

$$[\![\nu Z.\varphi]\!]^{\mathcal{T}}_{\mathcal{V}} = \bigcup\{S \subseteq \mathcal{S} \mid S \subseteq [\![\varphi]\!]^{\mathcal{T}}_{\mathcal{V}[Z:=S]}\}$$

where $\mathcal{V}[Z := S]$ maps Z to S and otherwise agrees with $\mathcal{V}$.

For φ_1, φ_2 such that $[\![\varphi_1]\!] \subseteq [\![\varphi_2]\!]$, $[\![\nu Z.\varphi_1]\!] = \bigcup\{S \subseteq \mathcal{S} \mid S \subseteq [\![\varphi_1]\!]_{\mathcal{V}[Z:=S]}\}$. This implies that $S \subseteq [\![\varphi_2]\!]_{\mathcal{V}[Z:=S]}$ and therefore $[\![\nu Z.\varphi_1]\!] \subseteq [\![\nu Z.\varphi_2]\!]$, which demonstrates positive polarity of ν. $\qquad\square$

3 Redundancy

Proposition 6 permits us to directly apply the vacuity-detection procedure of [5] to mu-calculus formulas. However, in the case of mu-calculus, the notion of vacuity does not capture subtle forms of antecedent failure. For example, the action labels in modal operators can be a source of antecedent failure. In this section, we extend the notion of vacuity to take into account sources of antecedent failure that escape traditional vacuity analysis.

Consider the mu-calculus formula $\varphi_1 = \mu X.(p \vee \langle\{a,b\}\rangle X)$ which states that proposition p holds eventually along some path containing action labels a or b. Now consider the formula $\varphi_2 = \mu X.(p \vee \langle a \rangle X)$ which states that p holds eventually along some path containing only label a. The formula φ_2 is more specific that φ_1: if φ_2 holds in a model M, so does φ_1. Hence there are syntactic elements in φ_1 that are not *needed* for its validity in M.

The above example is actually a form of antecedent failure. To see this, notice that $\langle\{a,b\}\rangle X \equiv \langle a \rangle X \vee \langle b \rangle X$ which may be written as $\neg\langle a \rangle X \Rightarrow \langle b \rangle X$. Clearly, the antecedent in this implication fails whenever $\langle a \rangle X$ holds. Since syntactic elements such as action labels in the mu-calculus are not subformulas, they are not covered by vacuity defined in [5]. Section 5 describes several previously published case studies that suffer from antecedent failure due to unnecessary actions in modal operators. Thus, this is a problem that occurs in practice and tends to obscure bugs in system models.

To capture these other forms of antecedent failure, we extend the notion of vacuity to one of *redundancy*. Vacuity detection can be seen as determining whether a formula can be strengthened or weakened without affecting its validity. For example, let $\varphi = \psi_1 \vee \psi_2$, and let M be a model such that $M \models \varphi$. If $M \models \psi_1$, then φ can be replaced by ψ_1. Since $[\![\psi_1]\!] \subseteq [\![\varphi]\!]$, this replacement has in effect strengthened the formula. Vacuity checking, as defined in [5], does precisely this by checking for the validity of $\varphi[\psi_2 \leftarrow false]$, which is same as ψ_1. Similarly, the vacuity of a formula invalid in M is detected by testing the validity of a weaker formula. However replacement of subformulas by *true* or *false* alone is insufficient in the case of mu-calculus. Let M be a structure such that $M \models \langle\{a,b\}\rangle\psi$, but $M \not\models \langle\{a,b\}\rangle false$. Following [5], $\langle\{a,b\}\rangle\psi$ is not ψ-vacuous in M. However, if $M \models \langle a \rangle\psi$, there is antecedent failure in the proof of $M \models \langle\{a,b\}\rangle\psi$ as explained in the previous paragraph. We can show that b in the above formula is superfluous if we can strengthen $\langle\{a,b\}\rangle\psi$ to $\langle a \rangle\psi$.

Checking for vacuity by strengthening (for valid formulas) or weakening (for invalid formulas) needs to be done judiciously. For instance, if φ holds in M, even if a stronger formula φ' holds in M, there may be no redundancy in φ. Consider a model M that satisfies the formula $\mu X.\ p \vee \langle - \rangle X$ in two steps, i.e., as $\langle - \rangle\langle - \rangle p$. Note that X can be written as $\neg p \Rightarrow \langle - \rangle(\neg p \Rightarrow \langle - \rangle(\neg p \Rightarrow \langle - \rangle X))$, and hence the antecedent of the innermost '$\Rightarrow$' fails in M. However, X is not trivially true in M, and hence should not be considered redundant. Hence we define a syntactic preorder over formulas, and restrict the choices of replacements during vacuity detection to only formulas related by this preorder.

The preorder $\prec$ on which we base redundancy checking in the mu-calculus captures a notion of *syntactic simplification* and is defined as follows:

- $false \prec \varphi$, and $true \prec \varphi$ for any $\varphi \notin \{true, false\}$
- $[S]\varphi \prec [T]\varphi$ and $[-S]\varphi \prec [-T]\varphi$ for any φ if $S \subset T$.
- $\langle S \rangle\varphi \prec \langle T \rangle\varphi$ and $\langle -S \rangle\varphi \prec \langle -T \rangle\varphi$ for any φ if $S \subset T$.

Intuitively, $\varphi_1 \prec \varphi_2$ means that φ_1 is syntactically simpler than φ_2. A formula φ is redundant with respect to a model M if it can be strengthened or weakened by replacing a subformula ψ with a simpler one without changing φ's validity.

Definition 7 (Redundancy) *A formula φ is said to be* redundant *with respect to model M if there is a subformula ψ such that*

$$M \models \varphi \iff \exists \psi' \prec \psi \text{ such that } M \models \varphi[\psi \leftarrow \psi']$$

where

- $[\![\psi']\!] \subseteq [\![\psi]\!]$ *if $M \models \varphi$ and ψ has positive polarity or $M \not\models \varphi$ and ψ has negative polarity in φ,*
- $[\![\psi]\!] \subseteq [\![\psi']\!]$ *otherwise*

Definition 7 specifies an effective procedure for finding redundancy. The test for $[\![\psi']\!] \subseteq [\![\psi]\!]$ (or its dual) for formulas ψ, ψ' such that $\psi' \prec \psi$ is straightforward based on the following identities that hold for all φ:

- $[\![false]\!] \subseteq [\![\varphi]\!]$ and $[\![\varphi]\!] \subseteq [\![true]\!]$
- $[\![[-S]\varphi]\!] \subseteq [\![[-T]\varphi]\!]$ if $S \subset T$
- $[\![[T]\varphi]\!] \subseteq [\![[S]\varphi]\!]$ if $S \subset T$
- $[\![\langle S\rangle\varphi]\!] \subseteq [\![\langle T\rangle\varphi]\!]$ if $S \subset T$
- $[\![\langle -T\rangle\varphi]\!] \subseteq [\![\langle -S\rangle\varphi]\!]$ if $S \subset T$

Observe that the notion of redundancy extends vacuity, since the constants $true$ and $false$ are defined to be simpler than any formula, and hence are always candidates for replacement. Hence we have:

Proposition 8 *For all formulas φ in modal mu-calculus, if φ is vacuous then φ is redundant.*

The simplification preorder defined above can be further refined by introducing additional cases. For example, we can stipulate that:

- $[-]\varphi \prec [T]\varphi$ for any φ.
- $\langle -\rangle\varphi \prec \langle T\rangle\varphi$ for any φ.

For an illustration of the rationale behind these additions, consider the formulas $\varphi_1 = [a]\psi$ and $\varphi_2 = [-]\psi$. If $M \models \varphi_2$ then clearly also $M \models \varphi_1$, but the label a itself plays no role, and hence is redundant.

Redundancy of Temporal Operators For the mu-calculus, we do not need additional rules for simplifying fixed-point operators. For example, consider formula $\varphi = \mu X.p \vee \langle -\rangle X$ (EFp in CTL) and a model M where φ holds. Replacing $\langle -\rangle X$ by $false$ we get $\mu X.p$, which is equivalent to p, as the strengthened formula. In terms of CTL, this is as if we check the vacuity of EF operator. The explicit nature of the fixed-point operators lets us check for their redundancy with no additional rules.

To check for redundancy in temporal logics without explicit fixed-point operators, such as CTL and CTL*, a simplification preorder over temporal operators must be specified. For example, in the case of CTL, the natural simplification preorder would stipulate that $false \prec AG\varphi \prec EG\varphi \prec \varphi \prec AF\varphi \prec EF\varphi$. Such a simplification preorder would therefore enable one to detect redundancies that arise from subformulas as well as temporal operators.

7

4 Algorithm for Redundancy Detection

Our algorithm for redundancy checking is based on Definition 7: a subformula ψ in φ is redundant if there is a "simpler" ψ' (i.e. ψ' is smaller than ψ in the simplification preorder $\prec$) such that $M \models \varphi \equiv M \models \varphi[\psi \leftarrow \psi']$. When ψ does not contain modal operators, only $true$ and $false$ are simpler than ψ and hence the definition of redundancy collapses to the definition of vacuity in [5]. For a formula ψ with modal operators, the number of simpler formulas ψ' is exponential in the number of action labels in the modal operator of ψ. However, not every simplification of ψ needs to be considered for checking the redundancy of ψ'. Below we define a set of possible replacements for each *modal* formula that is sufficient for the redundancy check.

Definition 9 (Replacement Set) *Given a model M, formula φ, and a subformula ψ of φ, the replacement set of ψ with respect to φ and M, denoted by $\Sigma(\psi)$, is the set:*

- *if $M \models \varphi$ and ψ has positive polarity or $M \not\models \varphi$ and ψ has negative polarity*

$$\Sigma(\langle T \rangle \phi) = \{\langle T - \{a\} \rangle \phi | a \in T\}$$
$$\Sigma(\langle -T \rangle \phi) = \emptyset$$
$$\Sigma([T]\phi) = \{[-]\phi\}$$
$$\Sigma([-T]\phi) = \{[-(T - \{a\})]\phi | a \in T\}$$

- *if $M \models \varphi$ and ψ has negative polarity or $M \not\models \varphi$ and ψ has positive polarity*

$$\Sigma(\langle T \rangle \phi) = \{\langle - \rangle \phi\}$$
$$\Sigma(\langle -T \rangle \phi) = \{\langle -(T - \{a\}) \rangle \phi | a \in T\}$$
$$\Sigma([T]\phi) = \{[T - \{a\}]\phi | a \in T\}$$
$$\Sigma([-T]\phi) = \emptyset$$

It should be noted that the replacement set for a formula ψ with modal operators is linear in the number of action labels in the operators,

The rationale for the above definition of Σ is two-fold. First, it incorporates the semantic condition $[\![\psi']\!] \subseteq [\![\psi]\!]$ (or its dual) into the syntactic check. Second, it picks only maximal candidates for replacement, ignoring those that are "covered" by others. For instance, let $\psi = \langle T \rangle \psi_1$ be a subformula of φ. Furthermore let ψ have positive polarity and $M \models \varphi$. Formulas of the form $\langle S \rangle \psi_1$ for some subset $S \subset T$ are simplifications of ψ. From the polarity of ψ, we have $M \not\models \varphi[\psi \leftarrow \langle S \rangle \psi_1] \Rightarrow \forall S' \subset S \quad M \not\models \varphi[\psi \leftarrow \langle S' \rangle \psi_1]$. Hence if some S is insufficient to show the redundancy of T then no $S' \subset S$ can show the redundancy of T. Therefore it is sufficient to consider the maximal sets S such that $S \subset T$ in the replacement formula $\langle S \rangle \psi_1$. The following proposition can be readily established considering the different cases in the definition of Σ.

Proposition 10 *A modal subformula ψ is redundant in φ with respect to model M if there is some $\psi' \in \Sigma(\psi)$ such that $M \models \varphi \equiv M \models \varphi[\psi \leftarrow \psi']$.*

Algorithm 1 Redundancy detection by subformula simplification

algorithm $find\_redundancies(M, \varphi)$
$NonVac$ keeps track of non-redundant subformulas;
$subs$ returns a formula's immediate subformulas.

```
 1:  mck(M, φ)
 2:  NonVac = {φ}
 3:  Red = ∅
 4:  while NonVac ≠ ∅ do
 5:     remove a ψ from NonVac
 6:     for all χ ∈ subs(ψ) do
 7:        if ψ[χ ← ⊥_χ] ⟺ ⊥_ψ then
 8:           add χ to NonVac
 9:        else
10:           mck(M, φ[χ ← ⊥_χ])
11:           if φ[χ ← ⊥_χ] has the same value as that of φ then
12:              add χ to Red
13:           else
14:              add χ to NonVac
15:              if χ is a modal formula then
16:                 for all χ' ∈ Σ(χ) do
17:                    mck(M, φ[χ ← χ'])
18:                    if φ[χ ← χ'] has the same value as that of φ then
19:                       add χ to Red
20:  if Red = ∅ then
21:     report "φ is not redundant in M"
22:  return Red
```

We can further reduce the number of model-checking runs needed to determine redundancy by exploiting the relationship between vacuity of a formula and its immediate subformulas. Since $\varphi[\chi \leftarrow \bot_\chi] \equiv \varphi[\psi \leftarrow \psi[\chi \leftarrow \bot_\chi]]$ when χ is a subformula of ψ, applying Theorem 5 we have the following proposition.

Proposition 11 *If χ is a subformula of ψ and $\psi[\chi \leftarrow \bot_\chi] \iff \bot_\psi$, then φ is not χ-vacuous in M iff φ is not ψ-vacuous in M.*

Since redundancy extends vacuity, if φ is ψ-vacuous in M, then all subformulas of ψ are redundant in φ.

Note that the proof of the if-direction of the above proposition follows from Lemma 3 of [5]. Whether $\psi[\chi \leftarrow \bot_\chi]$ is equal to $\bot_\psi$ can be checked by syntactic transformation techniques such as partial evaluation, using rules such as $false \wedge X \iff false$ and $\mu X.p \wedge \langle - \rangle X \iff false$.

The pseudo-code for the redundancy-checking algorithm is presented in Algorithm 1. For a given model M and formula φ, the algorithm returns the set of maximal subformulas that are redundant in φ. Finding the *maximal* subformula of φ for which φ is redundant is accomplished by traversing φ's syntax tree in a top-down manner. For each subformula ψ, we first check if φ is ψ-vacuous by model checking $\varphi[\psi \leftarrow \bot_\psi]$, as required in [5]. If φ is ψ-vacuous, we do

9

not traverse the children of ψ since we know from Proposition 11 that they are all vacuous. If φ is not ψ-vacuous and φ contains a modal operator, we check whether it is redundant by model checking φ after replacing ψ by some formula from $\Sigma(\psi)$. Note that in the worst case we will visit all subformulas and hence use $O(|\varphi|)$ model-checking runs.

5 Case Studies and Performance Measurements

Below we describe how we applied the redundancy checker to case studies provided with the XMC and CWB-NC [11] distributions. In summary, we found nontrivial redundancies in modal mu-calculus formulas taken from XMC's Meta-lock and Rether examples and from CWB-NC's railway example. We also used the Meta-lock example as a performance benchmark for our redundancy checker.

5.1 Case Study 1: Java Meta-lock (XMC)

Meta-locking is a highly optimized technique to ensure that threads have mutually exclusive access to object monitor queues, and forms an important component of the Java virtual machine [1]. An abstract specification of the Meta-lock algorithm was verified in XMC [2] and has since been distributed as a part of the XMC system. The specification, whose top-level process is denoted by `metaj(M, N)` is parameterized with respect to the number of threads (`M`) and number of objects (`N`) that are accessed by these threads.

One of the properties verified was `liveness(I,J)`, which states that a thread `I` requesting a meta-lock to object `J`, will eventually obtain this meta-lock:

```
liveness(I,J) -= [requesting_metalock(I,J)] formula(I,J)
             /\ [-] liveness(I,J).

formula(I,J) += <got_metalock(I,J)> tt
             \/ form(I,J)
             \/ [-] formula(I,J).

form(I,J) += <got_metalock(I,J)> tt
          \/ [-{requesting_metalock(_,_)}] form(I,J).
```

Our checker detected redundancy in the subformula `[-] formula(1,1)` with respect to the transition system generated by `metaj(2,1)`. In particular, our checker found that subformulas `<got_metalock(1,1)> tt` of `formula(1,1)`, and `<got_metalock(1,1)> tt` of `form(1,1)` were redundant. In retrospect, the problem with the Meta-lock liveness formula is that it attempted to use the alternation-free fragment of the mu-calculus supported by XMC to incorporate a notion of strong fairness into the property. Unfortunately, alternating fixed points must be used to express strong fairness.

Performance of Redundancy Checking. Performance results for the redundancy checker on the liveness formula are given in Table 1 for various values of M and N. The formula's syntax tree is depicted in Figure 1, and associates the DFS number of each node with the corresponding subformula. Each row of the table corresponds to a subformula ψ (indexed by its DFS number in the syntax tree) and contains the time (in seconds) and space (in MB) needed to check the redundancy of ψ; i.e., model-check $\varphi[\psi \leftarrow \psi']$. Model checking is not needed for two kinds of subformulas: those that can be determined to be non-redundant by Proposition 11 (marked by an 'x'), and those that can be determined to be redundant because they are subformulas of formulas that have already been determined redundant (marked by '-').

From Table 1 we notice that model checking is is quicker when a subformula is not redundant (corresponding to a model-checking result of false), than when a formula is redundant (a model-checking result of true). This is due to the combination of the formula structure and the fact that XMC's model checker is *local*; i.e., it operates in a goal-directed manner, examining only those portions of the state space needed to prove or disprove the formula. In particular, the liveness formula has universal modalities, causing the entire state space to be searched to establish its truth. In contrast, if it is false, this can be determined after visiting only a small fraction of the reachable state space.

Effect of Caching. Redundancy checking involves multiple runs of the model checker, each involving the same model and formulas similar up to subformula replacement. Recognizing this situation, we exploited the memo tables built by the underlying XSB resolution engine to share computations across model-checking runs. Although retaining memo tables after each model-checking run resulted in the memory usage growing monotonically, the additional memory overhead is low compared to the savings in time: up to 25% savings.

5.2 Case Study 2: Rether (XMC)

Rether is a software-based, real-time ethernet protocol [1]. Its purpose is to provide guaranteed bandwidth for real-time (RT) data over commodity ethernet hardware, while also accommodating non-real-time (NRT) traffic. This protocol was first formally verified by Du et al. [12] using the Concurrency Factory.

Among the properties listed in [12], we found redundancies in a group of formulas that specify a "no starvation for NRT traffic" property. For each node i, a formula NS_i is defined as

$$NS_i = \nu X.([-]X \wedge [start]\nu Y.([-\{nrt_i, cycle\}]Y \wedge [cycle]\mu Z.([-nrt_i]Z \wedge \langle-\rangle\mathtt{tt})))$$

which means that if during a particular cycle (between actions $\mathtt{start}$ and $\mathtt{cycle}$) node i does not transmit any NRT data (signaled by action nrt_i), then eventually node i will transmit some NRT data after the cycle. Our checker shows the actions in the box modalities $[start]$, $[-\{nrt_i, cycle\}]$, and $[cycle]$ are redundant. Indeed, the simpler formula

$$NS_i' = \nu X.([-]X \wedge \mu Z.([-nrt_i]Z \wedge \langle-\rangle\mathtt{tt}))$$
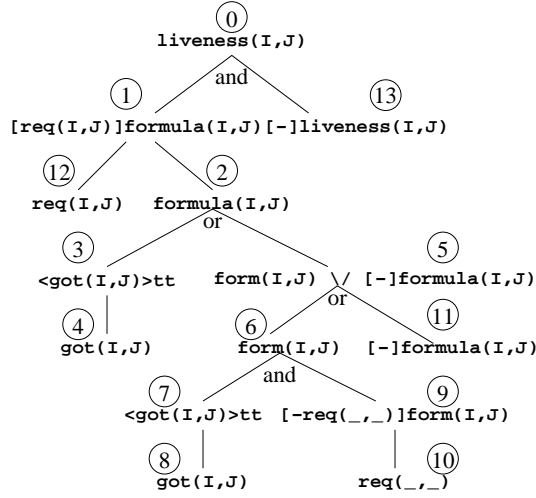
11

Fig. 1. Syntax tree of liveness property. Action labels `requesting_metalock` and `got_metalock` are abbreviated to `req` and `got` respectively.

which means always eventually nrt_i will occur, holds for the protocol. Hence the cycle-based argument is unnecessarily complicated.

Redundancy checking also helped us find a bug in the XMC model of Rether. At first we found another formula was also redundant:

$$RT_0 = \nu X.([-]X \wedge [reserve_0]\mu Y.([-cycle]Y \wedge [rt_0]\texttt{ff} \wedge$$
$$[cycle]\mu Z.(\langle-\rangle\texttt{tt} \wedge [-rt_0]Z \wedge [cycle]\texttt{ff})))$$

Redundancy was detected in the μY subformula by showing that $\nu X.([-]X \wedge [reserve_0]\texttt{ff})$ is true. That implies action $reserve_0$ is never enabled, i.e. node 0 never reserves a real-time slot. Examining the XMC model, we found a bug that causes node 0, which initially holds a real-time slot, to never release it, and hence has no need to reserve one! RT_0 became non-redundant after we fixed the bug.

5.3 Case Study 3: Railway Interlocking System (CWB-NC)

The CWB-NC's railway example models a slow-scan system over a low-grade link in British Rail's Solid State Interlocking system [10]. We found several instances of redundancy in this case study. The properties in question use the following four actions: $\overline{\texttt{fail\_wire}}$ and $\overline{\texttt{fail\_overfull}}$ report link failure due to a broken wire or due to a buffer overflow in a channel; $\overline{\texttt{det}}$ signals the detection of a failure; $\overline{\texttt{recovered}}$ marks the system's reinitialization to the correct state.

The first property is "a failure is possible":

$$failures\_possible = \mu X.\langle\overline{\texttt{fail\_overfull}}, \overline{\texttt{fail\_wire}}\rangle\texttt{tt}$$
$$\vee \langle-\overline{\texttt{recovered}}\rangle X$$

ψ	(2,2)			(2,3)			(3,1)			(3,2)		
	Truth	Time	Mem	Truth	Time	Mem	Truth	Time	Mem	Truth	Time	Mem
0	true	2.15	11.9	true	28.24	142.00	true	1.44	10.29	true	55.27	285.47
1	x	x	x	x	x	x	x	x	x	x	x	x
2	false	0.79	12.11	false	10.27	144.4	false	0.50	10.45	false	18.90	289.82
3	true	1.36	12.51	true	17.79	148.15	true	0.88	10.69	true	34.82	296.88
4	-	-	-	-	-	-	-	-	-	-	-	-
5	false	0.81	12.76	false	10.41	150.60	false	0.51	10.85	false	19.23	301.24
6	true	0.93	13.08	true	11.47	152.95	true	0.70	11.18	false	21.16	307.34
7	-	-	-	-	-	-	-	-	-	true	51.81	325.95
8	-	-	-	-	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-	-	false	21.07	330.91
10	-	-	-	-	-	-	-	-	-	false	21.07	335.88
11	true	0.82	13.64	true	10.67	158.79	true	0.55	11.50	true	20.75	340.29
12	true	1.71	14.31	true	22.18	165.34	true	1.02	11.84	true	40.22	349.39
13	x	x	x	x	x	x	x	x	x	x	x	x
total		8.58	14.31		111.11	165.34		5.61	11.84		304.63	349.39

Table 1. Redundancy checker performance measurements.

Since $\langle \overline{\texttt{fail\_overfull}}, \overline{\texttt{fail\_wire}} \rangle \texttt{tt} = \langle \overline{\texttt{fail\_overfull}} \rangle \texttt{tt} \vee \langle \overline{\texttt{fail\_wire}} \rangle \texttt{tt}$, $failures\_possible$ can be converted to the disjunction of two stronger formulas: $fail\_overfull\_possible = \mu X.\langle \overline{\texttt{fail\_overfull}} \rangle \texttt{tt} \vee \langle -\overline{\texttt{recovered}} \rangle X$ and $fail\_wire\_possible = \mu X.\langle \overline{\texttt{fail\_wire}} \rangle \texttt{tt} \vee \langle -\overline{\texttt{recovered}} \rangle X$, which both turn out to be true. Therefore either $\overline{\texttt{fail\_overfull}}$ or $\overline{\texttt{fail\_wire}}$ (but not their simultaneous occurrence) does not contribute to the truth of $failure\_possible$. This also holds for the derived formula

$$failure\_possible\_again = \nu Y.failure\_possible \wedge [-]Y$$

The second property is "a failure can be detected only after an occurrence of a failure":

$$no\_false\_alarms = \nu X.([\overline{\texttt{det}}]\texttt{ff} \vee \langle \overline{\texttt{fail\_overfull}} \rangle \texttt{tt})$$
$$\wedge \; [-\overline{\texttt{fail\_overfull}}, \overline{\texttt{fail\_wire}}, \overline{\texttt{recovered}}]X$$

which means that along a path without failure and recovery, either it is impossible to detect a failure or there is a failure. Removing $\overline{\texttt{recovered}}$ from the formula means that even if recovery occurs, the property still holds. Furthermore, the formula is redundant in $\langle \overline{\texttt{fail\_overfull}} \rangle \texttt{tt}$: $\nu X.[\overline{\texttt{det}}]\texttt{ff} \wedge [-\overline{\texttt{fail\_overfull}}, \overline{\texttt{fail\_wire}}]X$ is also true. Two types of redundancies also exist in the derived formula:

$$no\_false\_alarms\_again = \nu Y.no\_false\_alarms \wedge [-]Y$$

These case studies illustrate that subtle redundancies can creep into complicated temporal properties. In some cases, the redundancies hide problems in the underlying system model. Redundancy detection provides significant guidance in deriving simpler and stronger temporal formulas for the intended properties.

6   Conclusions

In this paper, we have extended the notion of vacuity to the more encompassing notion of *redundancy*. Redundancy detection, unlike vacuity detection, can handle sources of vacuity that are not adequately treated by subformula replacement alone. The main idea behind redundancy checking is that a formula φ is redundant with respect to a model M if it has a subformula ψ that can be "simplified" without changing its validity. For this purpose, we have defined a simplification preorder for the modal mu-calculus that, among other things, addresses vacuity in the action structure of modal operators.

A simplification preorder for CTL that enables detection of unnecessarily strong temporal operators can also be defined. A simplification-based notion of redundancy addresses an open problem posed in [5], which was prompted by a question from the audience by Amir Pnueli at CAV '97.

We have also implemented an efficient redundancy checker for the XMC verification tool set. The checker exploits the fact that XMC can be directed to cache intermediate model-checking results in memo tables, addressing another open problem in [5]. We have revisited several existing case studies with our redundancy checker and uncovered a number of instances of redundancy that have gone undetected till now. Like XMC, our redundancy checker is written declaratively in 300 lines of XSB tabled Prolog code.

For future work we are interested in pursuing the problem of generating interesting witness to valid formulas that are not vacuously true, á la [4, 17, 5]. The approach we plan to take to this problem will be based on traversing the proof tree generated by XMC's justifier [22, 14].

References

1. O. Agesen, D. Detlefs, A. Garthwaite, R. Knippel, Y. S. Ramakrishna, and D. White. An efficient meta-lock for implementing ubiquitous synchronization. In *Proceedings of OOPSLA '99*, 1999.
2. S. Basu, S. A. Smolka, and O. R. Ward. Model checking the Java Meta-Locking algorithm. In *Proceedings of 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2000)*, April 2000.
3. D. Beatty and R. Bryant. Formally verifying a multiprocessor using a simulation methodology. In *Design Automation Conference '94*, pages 596–602, 1994.
4. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. In *CAV '97*, pages 279–290. LNCS 1254, Springer-Verlag, 1997.
5. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in temporal model checking. *Formal Methods in System Design*, 18(2):141–163, March 2001.
6. J. Bradfield and C. Stirling. Modal logics and mu-calculi: An introduction. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*. Elsevier, 2001.
7. E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen, editor, *Proceedings of the Workshop on Logic of Programs*, Yorktown Heights, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

8. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS*, 8(2), 1986.

9. E. M. Clarke and J. M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4), December 1996.

10. R. Cleaveland, G. Luettgen, V. Natarajan, and S. Sims. Modeling and verifying distributed systems using priorities: A case study. *Software Concepts and Tools*, 17:50–62, 1996.

11. R. Cleaveland and S. Sims. The NCSU Concurrency Workbench. In R. Alur and T. A. Henzinger, editors, *Computer Aided Verification (CAV '96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 394–397, New Brunswick, New Jersey, July 1996. Springer-Verlag.

12. X. Du, K. T. McDonnel, E. Nanos, Y. S. Ramakrishna, and S. A. Smolka. Software design, specification, and verification: Lessons learned from the Rether case study. In *Proceedings of the Sixth International Conference on Algebraic Methodology and Software Technology (AMAST'97)*, Sydney, Australia, December 1997.

13. X. Du, S. A. Smolka, and R. Cleaveland. Local model checking and protocol analysis. *Software Tools for Technology Transfer*, 2(3):219–241, November 1999.

14. H.-F. Guo, C.R. Ramakrishnan, and I.V. Ramakrishnan. Speculative beats conservative justification. In *Proc. of 17th International Conference on Logic Programming (ICLP'01)*, November 2001.

15. G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

16. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

17. O. Kupferman and M. Y. Vardi. Vacuity detection in temporal model checking. In *CHARME 99*. LNCS 1703, Springer-Verlag, 1999.

18. R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

19. J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proceedings of the International Symposium in Programming*, volume 137 of *Lecture Notes in Computer Science*, Berlin, 1982. Springer-Verlag.

20. Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, T. W. Swift, and D. S. Warren. Efficient model checking using tabled resolution. In *CAV '97*, LNCS 1254, Springer-Verlag, 1997.

21. C.R. Ramakrishnan, I.V. Ramakrishnan, S.A. Smolka, et al. XMC: A logic-programming-based verification toolset. In *CAV 2000*. Springer-Verlag, June 2000.

22. A. Roychoudhury, C.R. Ramakrishnan, and I.V. Ramakrishnan. Justifying proofs using memo tables. In *Proc. of Second International Conference on Principles and Practice of Declarative Programming (PPDP 2000)*, September 2000.

23. XSB. The XSB logic programming system v2.4, 2001. Available by anonymous ftp from `ftp.cs.sunysb.edu`.