Ample Set Partial Order Reduction for Actions

Yihao Yan and Stephen F. Siegel

Verified Software Laboratory
Department of Computer and Information Sciences
University of Delaware, Newark, Delaware, USA 19716
{yihaoyan,siegel}@udel.edu
https://vsl.cis.udel.edu

Abstract. In a standard model checking approach, a system is modeled as a Kripke structure, which specifies a set of state predicates, and a property is expressed as a linear temporal logic (LTL) formula over those predicates. If that formula is stutter-invariant, ample set partial order reduction (POR) can be used to reduce the number of states explored—often dramatically. A different, but equally important, approach uses a labeled transition system (LTS) as the model, and properties are expressed as LTL formulas over the "actions" labeling the edges. There is a standard way to translate an action-based model to a state-based one, but we show that this translation often renders POR ineffective. Instead of translating to a state-based model and using state-based POR, we propose a native action-based POR, which preserves a class of properties we call *blank-invariant*. We have implemented a simple LTS model checker that uses this action-based POR, and applied it to the RERS 2017 benchmark suite, comparing its performace with that of Spin on the translated models. We conclude that using action-based POR yields significant performance gains.

Keywords: Labeled Transition System, model checking, partial order reduction, linear temporal logic

1 Introduction

To apply model checking to a concurrent system, one must formulate properties that the system is expected to satisfy. A property may be expressed by specifying acceptable sequences of states, or it may be expressed by specifying acceptable sequences of actions (events). Each approach has advantages and disadvantages, and in any particular context one may be more appropriate than the other. In either case, one needs to deal with the state explosion problem [20].

Partial Order Reduction (POR) is an effective technique to alleviate the state explosion problem. There are several general approaches to POR, including ample sets [2,16], stubborn sets [20,18], and persistent sets [11]. Their differences are subtle and discussed in [22]. All mature state-based model checkers use POR, including Spin [14] (which uses the ample set approach), DiVinE [1], and CIVL [24].

There is a well-known, natural way to translate a model and property expressed in an action-based formalism to a state-based formalism (see Section 4.5). In this way, all of the tools mentioned above can be applied to action-based problems. However, this approach has a serious drawback: the translation often yields a model for which standard state-based POR methods are very ineffective. In a nutshell, almost all transitions become "visible" (in the language of ample-set POR) in the state-based model, leading to many missed reduction opportunities.

Despite some effort, we have not found a translation approach which avoids this problem. Instead, we have adapted the state-based ample-set POR method to an action-based formalism. This action-based POR can be implemented directly in any action-based model checker—specifically, any model checker in which properties are expressed as linear temporal logic (LTL) formulas over the set of actions.

We use *labeled transition systems* (LTS) as the semantic model for concurrent systems and ALTL (LTL formulas in which the atomic propositions are actions [7]) for specifications; these formalisms are described in Section 2. In the state-based context, ample set reduction preserves a class of properties known as *stutter-invariant*. The analog in the action-based context is a class of formulas we call *blank-invariant*. We define this class and explore some of its properties in Section 3; further details and proofs of the theorems stated in Section 3 can be found in our technical report [23]. The POR technique itself is described in detail in Section 4.

To evaluate, we developed a simple model checker for LTSs that are expressed as a parallel composition of components, a standard way to express a concurrent system using actions. We call this tool LTSChecker, and we compare it with Spin on the 2017 RERS benchmarks suite [6,15]. The evaluation is discussed in Section 5. Our experiments confirm that our native action-based approach achieves significantly better reduction than applying a state-based approach to a translated model. This results in dramatically reduced verification times when verifying properties that hold.

1.1 Related Work

In addition to the tools and methods described above, several approaches for verifying action-based systems have been explored. mCRL2 [12] is a language that can be used to describe behaviors of action-based systems; it comes with a toolset [4] that can verify mCRL2 models against specifications in modal μ-calculus. FDR [9] and PAT [17] are two CSP [13] based tools which can be used to verify properties of action-based compositional system models. PAT uses a stubborn-set based POR for refinement checking.

Valmari has explored ways to adapt stubborn sets to process algebras [19,21]. That work has been extended in several directions: e.g., in [10], weak stubborn sets are strengthened to preserve not only deadlock, but also traces, failures and divergences.

2 Notation and Definitions

2.1 Sequences

Let S be a set. S^* denotes the set of finite sequences of elements of S; S^ω the infinite sequences. Given a non-empty (finite or infinite) sequence ζ, $\mathsf{first}(\zeta)$ denotes the first element of ζ.

For $\zeta \in S^*$, $\mathsf{last}(\zeta)$ denotes the last element of ζ. If η is a (finite or infinite) sequence, then $\zeta \circ \eta$ denotes the concatenation of ζ and η.

For $\eta = a_0 a_1 \cdots \in S^\omega$ and $i \geq 0$, η^i denotes the suffix $a_i a_{i+1} \cdots \in S^\omega$.

For $S \subseteq T$ and η a sequence of elements of T, $\eta|_S$ denotes the sequence obtained by deleting from η all elements not in S.

2.2 Büchi Automata

Let Act be a universal set of actions. We assume Act is infinite.

Definition 1. A *Büchi Automaton* over Act is a 5-tuple $(S, \Sigma, \rightarrow, S^0, F)$, where

1. S is a finite set of *states*,
2. $\Sigma \subseteq \mathsf{Act}$ is the *alphabet*,
3. $\rightarrow \subseteq S \times \Sigma \times S$ is the *transition relation*,
4. $S^0 \subseteq S$ is the set of *initial states*, and
5. $F \subseteq S$ is the set of *accepting states*.

We write $s \xrightarrow{a} s'$ as shorthand for $(s, a, s') \in \rightarrow$. We write $s \xrightarrow{a_0 a_1 \ldots a_n} s'$ as shorthand for $\exists s_1, s_2, \ldots s_n \in S \,.\, s \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \ldots s_n \xrightarrow{a_n} s'$.

For $s \in S$ and $a \in \Sigma$, define

$$\mathsf{nxt}(s, a) \equiv \{s' \in S \mid s \xrightarrow{a} s'\}$$
$$\mathsf{enabled}(s) \equiv \{a \in \Sigma \mid \exists s' \in S \,.\, s \xrightarrow{a} s'\}.$$

Definition 2. Let $B = (S, \Sigma, \rightarrow, S^0, F)$ be a Büchi automaton. For $s \in S$, *a trace in B starting from s* is a (finite or infinite) sequence of actions $a_0 a_1 \ldots$ such that $s \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \ldots$ for some $s_1, s_2 \ldots \in S$. *A trace of B* is a trace in B starting from some $s^0 \in S^0$. A trace of B is *accepting* if it visits some $s \in F$ infinitely often. The *language of B* is the set of all accepting traces of B.

Definition 3. Let $B = (S, \Sigma, \rightarrow, S^0, F)$ be a Büchi automaton. Action $a \in \Sigma$ is *deterministic* if

$$(s \xrightarrow{a} s' \wedge s \xrightarrow{a} s'') \implies s' = s''.$$

for all $s, s', s'' \in S$. B is *deterministic* if all $a \in \Sigma$ are deterministic.

Definition 4. Let $B_i = (S_i, \Sigma_i, \rightarrow_i, S_i^0, F_i)$ $(i = 1, 2)$ denote two Büchi automata over Act. The *parallel composition of B_1 and B_2* is the Büchi automaton

$$B_1 || B_2 \equiv (S_1 \times S_2, \Sigma_1 \cup \Sigma_2, \rightarrow, S_1^0 \times S_2^0, F_1 \times F_2),$$

where $\to$ is defined by

$$\frac{s_1 \xrightarrow{a}_1 s_1' \quad a \notin \Sigma_2}{\langle s_1, s_2\rangle \xrightarrow{a} \langle s_1', s_2\rangle} \qquad \frac{s_2 \xrightarrow{a}_2 s_2' \quad a \notin \Sigma_1}{\langle s_1, s_2\rangle \xrightarrow{a} \langle s_1, s_2'\rangle} \qquad \frac{s_1 \xrightarrow{a}_1 s_1' \quad s_2 \xrightarrow{a}_2 s_2'}{\langle s_1, s_2\rangle \xrightarrow{a} \langle s_1', s_2'\rangle}.$$

If we flatten all tuples (e.g., identify $(S_1 \times S_2) \times S_3$ with $S_1 \times S_2 \times S_3$) then $||$ is an associative operator.

Definition 5. A *labeled transition system* (LTS) over Act is a 4-tuple $(Q, A, \to, q^0)$ for which $(Q, A, \to, \{q^0\}, Q)$ is a Büchi automaton over Act. In other words, it is a Büchi automaton in which all states are accepting and there is only one initial state. We call the alphabet A the set of *actions*.

2.3 Multiprocess systems

Definition 6. A *multiprocess action system* (MAS) over Act is a tuple $\mathcal{M} = (M_1, \ldots, M_n)$ of LTSs over Act ($n \geq 1$). A component of this tuple is called a *process*. Define $\mathsf{lts}(\mathcal{M}) \equiv M_1 || \cdots || M_n$.

Let $\mathcal{M} = (M_1, \ldots, M_n)$ be an MAS. Write $M_i = (Q_i, A_i, \to_i, q_i^0)$ $(1 \leq i \leq n)$, and $\mathsf{lts}(\mathcal{M}) = (Q, A, \to, q^0)$. Let $q = \langle q_1, \ldots, q_n\rangle \in Q$ and $1 \leq k \leq n$. We define:

1. for $a \in \mathsf{Act}$, $\mathsf{proc}(a) \equiv \{i \in \mathbb{N} \mid a \in A_i\}$,
2. $\mathsf{out}_k(q) \equiv \{a \in A \mid \exists p \in Q_k \,.\, q_k \xrightarrow{a}_k p\}$,
3. $\mathsf{enabled}_k(q) \equiv \{a \in \mathsf{out}_k(q) \mid \forall i \in \mathsf{proc}(a) \,.\, a \in \mathsf{out}_i(q)\}$,
4. $\mathsf{enabled}(q) \equiv \bigcup_{1 \leq i \leq n} \mathsf{enabled}_i(q)$.

We say an action $a \in \mathsf{Act}$ is *locally enabled* at state q if $a \in \mathsf{out}_k(q)$ for some k. We say a is *enabled* at state q if $a \in \mathsf{enabled}(q)$.

2.4 LTL of Actions (ALTL)

Syntactically, an ALTL formula is an LTL formula in which the atomic propositions are elements of Act. Specifically,

- true is an ALTL formula,
- if $a \in \mathsf{Act}$, a is an ALTL formula,
- if f and g are ALTL formulas, so are $\neg f$, $f \wedge g$, $f \vee g$, $f \to g$, $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$, and $f\mathbf{U}g$. The symbol false is shorthand for the ALTL formula $\neg\mathsf{true}$.

Definition 7. The *alphabet* of an ALTL formula f, denoted αf, is the set of actions that occur syntactically within f.

For example, if $f = a \to \neg b$ then $\alpha f = \{a, b\}$.

The semantics of ALTL is defined very much like in the state-based theory. One defines a relation $\zeta \models f$, where $\zeta \in \mathsf{Act}^\omega$ and f is a formula:

- $\zeta \models \mathsf{true}$,

- $\zeta \models a$ iff $\mathsf{first}(\zeta) = a$,
- $\zeta \models \neg f$ iff $\zeta \not\models f$,
- $\zeta \models f \wedge g$ iff $\zeta \models f$ and $\zeta \models g$,
- $\zeta \models \mathbf{X}f$ iff $\zeta^1 \models f$,
- $\zeta \models f\mathbf{U}g$ iff $\exists i \geq 0 \,.\, (\zeta^i \models g \ \wedge \ \forall j \in 0..i-1 \,.\, \zeta^j \models f)$,

and so on.

The main difference between the state- and action-based formalisms is that in the state-based formalism, any number of atomic propositions can hold at each step. In the action-based formalism, precisely one action occurs in each step.

Definition 8. We say two ALTL formulas f and g are *action-equivalent*, written $f \equiv_A g$, if

$$\zeta \models f \iff \zeta \models g$$

for all $\zeta \in \mathsf{Act}^\omega$.

Action equivalence should not be confused with the usual (state-based) notion of equivalence of LTL formulas. Recall that two LTL formulas f and g are equivalent, written $f \equiv g$, if, for any sequence σ of *sets of atomic propositions*,

$$\sigma \models f \iff \sigma \models g. \tag{1}$$

Now suppose a and b are two distinct elements of Act. We have $a \wedge b \equiv_A \mathsf{false}$, since the first action of an action sequence could be a or b, but not both. In contrast, $a \wedge b \not\equiv \mathsf{false}$, because it is possible for the first set in a sequence of sets of atomic propositions to contain both a and b, i.e., two propositions may hold in the same state.

On the other hand, for any formulas f and g, if $f \equiv g$ then $f \equiv_A g$. This is because (1) must hold in particular for all σ in which each set contains exactly one element. This is equivalent to the statement that f and g are action-equivalent.

Definition 9. The *language* of f is the set of all $\zeta \in \mathsf{Act}^\omega$ for which $\zeta \models f$.

Definition 10. Given an LTS M, we write $M \models f$ if the language of M is contained in the language of f.

The goal of this paper is to explore efficient methods for determining whether the LTS of a multiprocess action system satisfies an ALTL formula.

3 Blank Invariant ALTL Properties

In this section we describe a class of ALTL formulas that has several "nice" properties. These *blank-invariant* formulas are amenable to partial order reduction techniques. This class is roughly analogous to the class of stutter-invariant LTL formulas in the state-based formalism. The stutter-invariant formulas have a simple characterization: f is stutter-invariant if, and only if, f is equivalent to a formula that does not use the $\mathbf{X}$ operator. We have not found such a simple characterization for blank-invariance, but we do provide a description of a large class of blank-invariant formulas.

Definition 11. Let f be an ALTL formula over Act and $\zeta, \eta \in \mathsf{Act}^\omega$. We say ζ and η are *blank-equivalent for* f if $\zeta|_{\alpha f} = \eta|_{\alpha f}$.

Blank-equivalence for f is an equivalence relation on Act^ω. If ζ and η are blank-equivalent for f, then η can be obtained from ζ by a (possibly infinite) sequence of operations, each of which either inserts or removes an action that does not occur in f.

Definition 12. An ALTL formula f is *blank-invariant* if

$$\zeta \models f \iff \eta \models f$$

whenever $\zeta, \eta \in \mathsf{Act}^\omega$ are blank-equivalent for f. The set of all blank-invariant formulas over Act is denoted $\mathsf{BInv}(\mathsf{Act})$, or just BInv if the universal set of actions is clear from the context.

Similar to the way in which stutter-invariance is a natural restriction for LTL properties, blank-invariance is a natural restriction for ALTL properties. When creating an event-based model of a system in the world, one identifies certain events and formulates properties using them. Later, one might introduce new events, which may have nothing to do with the properties already formulated. A "reasonable" property should not "care" about these new events. Examples of such reasonable properties include "the event a never occurs", "a occurs at least twice", and "between every occurrence of a and b, c must occur."

We now give some examples of ALTL formulas that are, and are not, blank-invariant. Suppose $a, b \in \mathsf{Act}$ and $a \neq b$. The formula $\mathbf{G}a$ is not blank-invariant, because $\zeta = a^\omega$ satisfies $\mathbf{G}a$ but $\eta = ba^\omega$ does not, even though ζ and η are blank-equivalent for $\mathbf{G}a$. On the other hand, $\mathbf{G}\neg a \in \mathsf{BInv}$, since if two action sequences are blank-equivalent for $\mathbf{G}\neg a$, one will contain an a if, and only if, the other contains an a. Similarly, $\mathbf{F}a \in \mathsf{BInv}$ and $\mathbf{F}\neg a \notin \mathsf{BInv}$. The formula $\mathbf{F}(a \wedge \mathbf{XF}a)$ is also blank-invariant and means "a occurs at least twice."

Proposition 1. *Suppose f and g are action-equivalent ALTL formulas over Act. Then $f \in \mathsf{BInv} \iff g \in \mathsf{BInv}$.*

The proof of Proposition 1 is obvious if $\alpha f = \alpha g$. The case where the two formulas have different alphabets is more subtle. The proof in that case requires that there exists at least one action not in $\alpha f \cup \alpha g$, which is guaranteed by our assumption that Act is infinite. To see why that assumption is necessary, consider the following example: $\mathsf{Act} = \{a, b\}$, $f = \mathbf{G}a$, $g = f \wedge (b \vee \neg b)$. Clearly f and g are action-equivalent, but $\alpha f = \{a\}$ and $\alpha g = \{a, b\}$. Since αg contains every action, if ζ and η are blank-equivalent for g then $\zeta = \eta$. Therefore g is blank-invariant. As we have seen, f is not blank-invariant.

The set of blank-invariant formulas is distinct from the set of stutter-invariant formulas. For example, $f = \mathbf{F}(a \wedge \mathbf{XF}a)$ is blank-invariant, but not stutter-invariant. In fact f is not action-equivalent to any stutter-invariant formula g, since if there were such a g, the sequence aab^ω would satisfy g, but the stutter-equivalent sequence ab^ω cannot satisfy g.

Conversely, the formulas a and $\mathbf{G}a$ are both stutter-invariant, but neither is blank-invariant. The formula $\mathbf{F}a$ is both stutter- and blank-invariant. Finally, the formula $\mathbf{X}a$ is neither stutter- nor blank-invariant.

Like the stutter-invariant formulas, BInv is closed under all ALTL operations other than $\mathbf{X}$:

Proposition 2. *Suppose* $f, g \in \mathsf{BInv}$. *Then* true, $\neg f$, $f \wedge g$, $f \vee g$, $f \rightarrow g$, $\mathbf{F}f$, $\mathbf{G}f$, *and* $f\mathbf{U}g$ *are all in* BInv.

3.1 A class of blank-invariant formulas

A well-known result states that an LTL formula f is stutter-invariant if, and only if, it is equivalent to an LTL formula that does not use the $\mathbf{X}$ operator ([5], [2, Thm. 10]). We do not know of a similarly simple characterization of the blank-invariant formulas, but we can describe a large class of formulas which are blank-invariant. In our examination of the RERS benchmarks, we have found that this class captures most of the blank-invariant formulas that arise in practice.

Proposition 3. *There exist sets* Pos *and* Neg *of ALTL formulas satisfying (i) for all ALTL formulas* f *and* f',

$$(f \in \mathsf{Pos} \wedge f' \equiv_A f) \implies f' \in \mathsf{Pos}$$
$$(f \in \mathsf{Neg} \wedge f' \equiv_A f) \implies f' \in \mathsf{Neg},$$

and (ii) for all $a \in \mathsf{Act}$, $f_1, f_2 \in \mathsf{BInv}$, $g_1, g_2 \in \mathsf{Pos}$, *and* $h_1, h_2 \in \mathsf{Neg}$,

$$\mathsf{false},\ a,\ \neg h_1,\ g_1 \wedge g_2,\ g_1 \vee g_2,\ a \wedge f_1,\ a \wedge \mathbf{X}f_1 \ \in\ \mathsf{Pos}$$
$$\mathsf{true},\ \neg a,\ \neg g_1,\ h_1 \wedge h_2,\ h_1 \vee h_2,\ \neg a \vee f_1,\ \neg a \vee \mathbf{X}f_1 \ \in\ \mathsf{Neg}$$
$$\mathsf{true},\ \mathsf{false},\ f_1 \wedge f_2,\ f_1 \vee f_2,\ \neg f_1,\ \mathbf{F}g_1,\ \mathbf{G}h_1,\ f_1\mathbf{U}f_2,\ h_1\mathbf{U}g_1,\ h_1\mathbf{U}f_1 \ \in\ \mathsf{BInv}.$$

Proposition 3 can be used to show that many formulas are blank-invariant. For example, $\mathbf{G}(a \rightarrow \mathbf{F}b)$, which occurs multiple times in the RERS suite, is seen to be blank-invariant as follows. First $b \in \mathsf{Pos}$, so by the proposition, $\mathbf{F}b \in \mathsf{BInv}$. Again by the proposition, $\neg a \vee \mathbf{F}b \in \mathsf{Neg}$. Since this last formula is equivalent to $a \rightarrow \mathbf{F}b$, we have $a \rightarrow \mathbf{F}b \in \mathsf{Neg}$. Therefore $\mathbf{G}(a \rightarrow \mathbf{F}b) \in \mathsf{BInv}$.

The proofs of these propositions can be found in [23].

4 Partial Order Reduction

4.1 Ample Sets

The basic idea is that there are often many blank-equivalent traces which differ only in how actions from different processes are interleaved. When checking blank-invariant formulas (Definition 12), one only needs to check a single trace from each equivalence class.

Given an MAS $\mathcal{M}$, the goal of POR is to avoid exploring all of $M = \mathsf{lts}(\mathcal{M})$, and instead to explore a "reduced" LTS M', with the property that $M \models f \iff M' \models f$. M' is typically discovered by selecting a subset of the set of enabled transitions at each state during the search. For a state q, this subset is denoted $\mathsf{ample}(q, f) \subseteq \mathsf{enabled}(q)$. In the standard state-based ample set method, these "ample" sets are required to satisfy four conditions named **C0**-**C3**. These conditions are framed using the concepts of *independence* and *invisibility*. Our action-based approach is very similar.

Definition 13. Given an ALTL formula f, action a is *invisible* for f if $a \notin \alpha f$, otherwise, a is *visible* for f.

Definition 14. Let M be an LTS. Two actions a_1, a_2 are *independent* if they satisfy the following two conditions:

- *enabledness*: for all states q of M, if $a_1, a_2 \in \mathsf{enabled}(q)$, then

$$(\forall q' \in \mathsf{nxt}(q, a_1) \,.\, a_2 \in \mathsf{enabled}(q')) \wedge \forall q' \in \mathsf{nxt}(q, a_2) \,.\, a_1 \in \mathsf{enabled}(q'),$$

- *commutativity*: for all states q, q' of M:

$$q \xrightarrow{a_1 a_2} q' \iff q \xrightarrow{a_2 a_1} q'.$$

Note that, in contrast with [2], we do not assume actions are deterministic.

We now restate the conditions **C0**–**C3**. For any state q,

C0: $\mathsf{ample}(q, f) = \emptyset \iff \mathsf{enabled}(q) = \emptyset$.
C1: For every trace in M (the full LTS) starting from q, an action that is dependent on an action in $\mathsf{ample}(q, f)$ can not occur without an action in $\mathsf{ample}(q, f)$ occurring first.
C2: If $\mathsf{ample}(q, f) \neq \mathsf{enabled}(q)$, then every $a \in \mathsf{ample}(q, f)$ is invisible for f.
C3: A cycle in M' is not allowed if there exists an action a that is enabled at some state on the cycle but is never put into the ample set of any state on the cycle.

4.2 Soundness

Theorem 1. *Let M be an LTS and f a blank-invariant ALTL formula. Suppose M' is the reduced LTS specified by ample sets satisfying **C0**–**C3**. For every trace π of M, there exists a trace π' of M' such that π and π' are blank-equivalent for f.*

The proof of Theorem 1 is very similar to that given in [2, §10.6], so we will only give a brief sketch.

Let σ be an infinite trace of M. An infinite sequence $\pi_0, \pi_1, \ldots$ of traces will be constructed, where $\pi_0 = \sigma$. For each $i \geq 0$, π_i will be decomposed as $\eta_i \circ \theta_i$ where η_i is a finite trace of length i, θ_i is an infinite trace, and η_i is a prefix of η_{i+1}. For $i = 0$, η_0 is empty and $\theta_0 = \pi_0$. Assume $i > 0$ and we have defined η_j and θ_j for $j \leq i$. Then η_{i+1} and θ_{i+1} are defined as follows.

Since π_i is a trace, there is a path through M in which the edges are labeled by the actions of π_i. Choose one such path, and let q be the state which is the final state reached by η_i and the initial state of θ_i. Let a=$\mathsf{first}(\theta_i)$.

- If $a \in \mathsf{ample}(q, f)$, let $\eta_{i+1} = \eta_i \circ a$ and $\theta_{i+1} = (\theta_i)^1$.
- If $a \notin \mathsf{ample}(q, f)$, there are two cases:
  1. Some action in $\mathsf{ample}(q, f)$ occurs in θ_i. Let b be the first such action and q' be the state in the path just after b. By **C1**, the actions $a_1, \ldots, a_n$ in θ_i preceding b are independent of b. By repeated application of the independence property, the sequence $b, a_1, \ldots, a_n$ is a trace starting from q and ending in q'. Let $\eta_{i+1} = \eta_i \circ b$, and θ_{i+1} is obtained by removing b from θ_i.
  2. No element of $\mathsf{ample}(q, f)$ occurs in θ_i. By **C1**, every element of θ_i is independent of every action in $\mathsf{ample}(q, f)$. By **C0**, $\mathsf{ample}(q, f) \neq \emptyset$. Let $b \in \mathsf{ample}(q, f)$, $\eta_{i+1} = \eta_i \circ b$ and $\theta_{i+1} = \theta_i$.

Lemma 1. *Let $i \geq 0$ and $a = \mathsf{first}(\theta_i)$. There exists $j > i$ such that $a = \mathsf{last}(\eta_j)$.*

Proof. Let q be the state at the end of η_i, as above. If $a \neq \mathsf{last}(\eta_{i+1})$ then $a \in \mathsf{enabled}(q) \setminus \mathsf{ample}(q, f)$. Suppose there does not exist $j > i$ such that $a = \mathsf{last}(\eta_j)$. Then there is an infinite path starting from q such that for every state q' on that path, $a \in \mathsf{enabled}(q) \setminus \mathsf{ample}(q', f)$. Since the number of states is finite, there must be a cycle on this path. Then a is enabled but never put into an ample set along the cycle, contradicting **C3**. □

Lemma 2. *Let $i \geq 0$ and a the first visible action for f on θ_i. There exists $j > i$ such that $a = \mathsf{last}(\eta_j)$ and for all $i < k < j$, $\mathsf{last}(\eta_k)$ is invisible for f.*

Proof. According to Lemma 1, there will exists $j > i$ such that $a = \mathsf{last}(\eta_j)$. And for all $i < k < j$, there will be two possible cases for $\mathsf{last}(\eta_k)$:

- $\mathsf{last}(\eta_k)$ is an action from the ample set of the source state of $\mathsf{last}(\eta_k)$, or
- $\mathsf{last}(\eta_k)$ is an action that precedes a in θ_i.

In both cases, we know that $\mathsf{last}(\eta_k)$ is invisible for f. □

Lemma 3. *Let ρ be a prefix of $\sigma|_{\alpha f}$, then there exists η_i such that ρ=$\eta_i|_{\alpha f}$.*

Proof. The proof is by induction on the length of ρ. The base case when ρ has length 0 is trivial. Assume, $\rho \circ a$ is a prefix of $\sigma|_{\alpha f}$, and $\eta_k|_{\alpha f} = \rho|_{\alpha f}$. We need to prove that there exists $j > k$ such that $\mathsf{last}(\eta_j)$=a and for all $k < m < j$, $\mathsf{last}(\eta_m)$ is invisible for f. But this follows from Lemma 2. □

Lemma 4. *Let η be the limit of the η_i. Then η is an infinite trace of M', and η and σ are blank-equivalent for f.*

Proof. According to the definition of η, η is a sequence of actions that can be executed from some initial state of M, and each action is in the ample set. Therefore, η is a trace of M'.

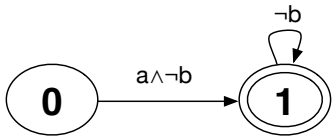

Fig. 1. An example of translation from ALTL formula $\neg f = \neg(a \rightarrow \mathbf{F}b)$ to a Büchi automaton treating $\neg f$ as a regular LTL formula.

We must show $\eta|_{\alpha f} = \sigma|_{\alpha f}$. Lemma 1 shows that every action—including visible actions—in σ will be the last action of some η_i. Since each η_i is a prefix of η, η contains all the visible actions in σ. Also, Lemma 3 shows that the order of visible actions in η must be the same as in σ. Hence $\eta|_{\alpha f} = \sigma|_{\alpha f}$. □

Lemma 4 shows that for every trace σ of M, there will be a trace σ' of M' such that σ and σ' are blank-equivalent for f, completing the proof of Theorem 1.

4.3 Algorithm

Our ample set method can be easily applied with automata theoretic, "on-the-fly" model checking. We can construct a Büchi automaton B of an ALTL formula $\neg f$ by treating $\neg f$ as a regular LTL formula [2]. And note that B should have the same alphabet as the system to be checked.

In the state-based context, the result of formula translation is a Büchi automaton "template" in which each transition is labeled by a boolean expression ϕ. This transition template may represent multiple actual transitions, each of which corresponds to a subset of the alphabet. When using this automaton in the ALTL context, we interpret ϕ as representing a set of transitions, each of which is labeled by precisely one action: there is a transition labeled by action a iff the assignment of *true* to a and *false* to every action other than a causes ϕ to evaluate to *true*.

Figure 1 gives an example of translating the ALTL formula $\neg(a \rightarrow \mathbf{F}b)$. If the alphabet is $\{a, b, c\}$, then $a \wedge \neg b$ represents a single transition labeled by a; $\neg b$ represents two transitions, one labeled by a and the other by c. The formula $a \wedge \neg b$ could be replaced by the action-equivalent a, but this is not necessary.

The verification problem is equivalent to checking the emptiness of the automaton $M||B$ where M is the LTS of the system and B is the automaton of the negation of the property, and this procedure can be done on-the-fly. Algorithm 1 shows the on-the-fly emptiness checking algorithm with POR. Procedure *dfs1* (line 7–18) explores the composition automaton of the reduced LTS of the system and the property automaton. Procedure *prod_ample* at line 8 returns the ample set of a composition automaton state. Procedure *dfs2* (line 19–29) tries to detect cycles when backtracking from an accepting state. Note that in *dfs2*, procedure *succ* (line 21) should return the same set of successors used in *dfs1*.

Algorithm 1: On-the-fly emptiness checking algorithm with POR

```
   Input: A MAS M
   Input: An automaton (S, Σ, →, S^0, F)
 1 Procedure emptiness:
 2   foreach s^0 ∈ S^0 do
 3     dfs1(⟨q^0, s^0⟩);
 4   end
 5   terminate(false);
 6 end
 7 Procedure dfs1(⟨q, s⟩):
 8   foreach a ∈ prod_ample(⟨q, s⟩) do
 9     foreach ⟨q', s'⟩ ∈ nxt(⟨q, s⟩, a) do
10       if ⟨q', s'⟩ is new then
11         dfs1(⟨q', s'⟩);
12       end
13     end
14   end
15   if ⟨q, s⟩ is accepting then
16     dfs2(⟨q, s⟩);
17   end
18 end

19 Procedure dfs2(⟨q, s⟩):
20   checked2(⟨q, s⟩);
21   foreach ⟨q', s'⟩ ∈ succ(⟨q, s⟩) do
22     if ⟨q', s'⟩ on dfs1 stack then
23       terminate(true);
24     end
25     if !checked2(⟨q', s'⟩) then
26       dfs2(⟨q', s'⟩);
27     end
28   end
29 end
```

Algorithm 2 produces the ample set at a given state. This procedure requires a partitioning of the set of processes (cf. [3]). Given a global MAS state, define a binary relation R on the set of processes as follows: $(p_1, p_2) \in R$ iff there is an action a such that a is locally enabled in p_1 (i.e., there is an outgoing edge labeled a from the current state of p_1) and a is in the alphabet of p_2. Let R' be the smallest equivalence relation containing R. The *partition* procedure returns the set of equivalence classes defined by R'. It is clear that if a is an enabled action in one class, then a is independent of *all* actions from a process in another class.

A candidate ample set consists of all actions in one class that are enabled in the product automaton. This candidate is guaranteed to satisfy **C1**. The other 3 conditions are then checked: if there is a visible action (line 4), or the candidate is empty, or a transition yields a back edge (creating a cycle, at line 8), the candidate is rejected. If no candidate satisfies all four conditions, the set of all actions enabled in the product automaton is returned (line 17).

4.4 Example

Figure 2 gives a simple example. For each $n \geq 1$, there is an MAS $\mathcal{M}(n)$ consisting of $n + 1$ processes. Process 0 will repeatedly execute action a and process i $(1 \leq i \leq n)$ will execute action a_i and terminate. The property to be checked is $f = \mathbf{F}a$, which clearly holds. The automaton for $\neg f$ is shown in Figure 2 (right).

When there is no reduction, all interleavings of the a_i will be explored, and 2^n different states will be visited during the search. If the ample set reduction

Algorithm 2: Composition state ample set computation algorithm

```
1  Procedure prod_ample(⟨q, s⟩):
2    foreach proc_group ∈ partition() do
3      result ← {};
4      if contain_visible(proc_group) then goto_next_proc_group ;
5      foreach procId ∈ proc_group do
6        foreach a ∈ enabled_procId(q) ⋂ enabled(s) do
7          foreach {⟨q′, s′⟩|q -a-> q′ ∧ s -a-> s′} do
8            if on(stack1, ⟨q′, s′⟩) then
9              go_to_next_proc_group;
10           end
11           result ← result ⋃ {a};
12         end
13       end
14     end
15     if result is not empty then return result ;
16   end
17   return enabled(⟨q, s⟩)
18 end
```

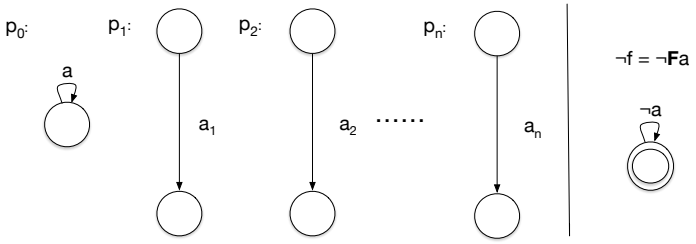

Fig. 2. A simple example showing that ample set method would bring significant reduction.

is applied, we only need to pick one process to form the ample set, only one interleaving of the a_i will be explored, and only $n + 1$ different states will be visited.

4.5 State-based Translation

There is a standard way to translate an LTS M_a into an equivalent state-based model M_s by introducing a global variable *lastAction* which records the last executed action. Each action a in M_a will be mapped to an atomic proposition $(lastAction = a)$ in M_s, and among all those atomic propositions, only one of them can be true at any time.

The following is a snippet of Promela code which models the example in Figure 2 with 3 processes:

```
active proctype proc0() { do :: lastAction = a od }
active proctype proc1() { lastAction = a1 }
active proctype proc2() { lastAction = a2 }
ltl p {<>(lastAction == a)}
```

In this code, every action is modeled as a statement that updates the global variable *lastAction*. Each of these statements might possibly alter the truth value of any atomic proposition. Hence a local action in M_a becomes a *visible* transition in the Spin model, and therefore cannot be included in any proper ample set. Indeed, a simple experiment applying Spin to this example shows the number of states grows exponentially with n, as expected. In contrast, the growth is linear with our tool.

This simple example does not involve any communication. However, a two-process synchronization action in an MAS can always be modeled using a Promela channel with capacity 0. All synchronizations in RERS challenges are two-way and RERS provides a Spin translation for each problem using this approach. Using these translations, Spin was the winner of the 2017 RERS challenge in the"Parallel LTL" category. We improved somewhat on the RERS translations by eliminating some unnecessary auxiliary processes and channels. In all of our experiments, Spin performs better (in terms of time and number of states) using our translation than it does using the official RERS translation, so we have used our translation in the evaluation of this work.

5 Experiments

The RERS Challenges provide an abundant set of verification problems ranging from sequential to concurrent with increasing complexity [15]. Thirty-five parallel problems are provided: 20 for training and 15 for participants to solve. Each problem consists of an MAS composed of some number of LTSs together with 20 properties expressed as ALTL formulas. In total, there are $35*20 = 700$ formulas. We wrote a simple Java tool, using a formal grammar based on Proposition 3, to detect blank-invariant formulas. This tool is conservative, in that if it says a formula is blank-invariant, the formula is blank-invariant; otherwise, nothing can be concluded. The tool determined that at least 665 of the formulas are blank-invariant, and by manual inspection we found 8 more blank-invariant formulas, for a total of 673. Of the remaining formulas, 20 are not blank-invariant, and 7 remain unknown to us. Note that these formulas were generated through a random "property mining" process, and were not selected with blank-invariance in mind.

An example of a RERS property is $\mathbf{G}(c1\_t6 \rightarrow \mathbf{F}(c0\_t13))$, where $c1\_t6$ and $c0\_t13$ are actions that label edges. As we have seen, a formula of this form is blank-invariant.

We developed a simple model checker named LTSChecker which implements the action-based ample set method. LTSChecker uses Java PathFinder's LTL to Büchi converter [8] to construct a Büchi automaton corresponding to the negation of the formula. We also developed a tool to translate an LTS model

into a Spin model using the approach described in Section 4.5. In the following, we will refer to the use of LTSChecker with POR enabled as LTSChecker$_y$ and LTSChecker with POR disabled as LTSChecker$_n$.

We ran LTSChecker and Spin on those 673 properties on a machine with 32GB memory and a 3.5GHz Intel i7 CPU. We used a time limit of 30 minutes for each verification task (a single formula and model). The results show that LTSChecker$_n$ solved 8 fewer properties than Spin within these time and memory bounds, while LTSChecker$_y$ solved 44 more properties than Spin. The total number of properties solved by LTSChecker$_n$, LTSChecker$_y$ and Spin are 322, 374 and 330, respectively. In the following, we focus on satisfied (true) properties, as these are the ones that require exhaustive exploration (since a search can terminate immediately if a violation is found).

Figure 3 presents the comparison between LTSChecker$_y$ and LTSChecker$_n$ in terms of the total number of states and the total time on satisfied (true) properties solved by both configurations. The results are grouped by problem. The statistics shows that our ample set method brings significant reduction in the size of the state space and speedup for problems with more than 3 processes except for problems 110–115. Problems 110–115 are from a category of challenges called "state space" and in those problems, most actions are synchronized and our ample set method can barely help. Note that there are 4 properties which are valid for trivial reasons (the formulas are equivalent to true); these are verified by LTSChecker just by translating the negated formula to a Büchi automaton. We exclude these four cases from the tables in this paper. There are 19/35 problems presented in Figure 3 and for the rest 16/35 problems, there are no satisfied properties solved by both configurations.

Figure 4 presents the comparison between LTSChecker$_y$ and Spin on the total number of states and the total time consumed for true properties solved by both tools grouped by problems. Note that there exist several properties with next operator and we ran Spin on them with POR disabled; there are 17/35 problems presented in Figure 4 and for the rest problems, there are no satisfied properties solved by both tools. The statistics shows that the state spaces of the Spin models are significantly larger than those of LTSChecker$_y$. The reason is that each labeled transition in LTS model may be translated into multiple transitions in Spin model, and there is at least one global transition (the one that updates *lastAction*) among them. And LTSChecker$_y$ is faster than Spin for solving satisfied properties for problems with more than 4 processes.

In terms of false properties, LTSChecker$_y$ solves 33 more than LTSChecker$_n$ and 6 more than Spin, and LTSChecker$_y$ solves the majority of the both solved properties faster when compared with LTSChecker$_n$ and Spin. However, there is no guarantee that LTSChecker$_y$ will have a better performance on false properties. Comparing to LTSChecker$_n$, LTSChecker$_y$ will search through a different path and it is likely that LTSChecker$_n$ will find the error first. When comparing with Spin, there are several differences that may introduce unpredictability: first, difference in ample set method implementation and property automaton construction would lead to a different search order; second, Spin inserts asser-

Problem	Procs	Solved	States	Time(s)	State/POR	Time(s)/POR
t101	1	12/12	4.80e1	<0.1	4.80e1	<0.1
t102	2	14/14	7.70e2	<0.1	2.94e2	<0.1
t103	3	8/8	2.84e2	<0.1	2.00e2	<0.1
t104	4	11/11	2.50e4	0.3	1.63e3	<0.1
t105	5	9/9	4.40e6	26.0	8.13e4	4.2
t106	6	9/9	5.80e6	43.0	1.40e5	1.6
t107	7	8/8	1.31e8	1566.2	3.45e7	262.3
t108	8	10/13	6.72e8	10062.1	1.65e8	1415.1
t109	9	5/10	2.31e8	3861.9	4.75e7	589.5
t111	11	4/11	5.50e7	917.6	3.32e6	22.4
101	4	7/7	1.30e4	0.6	1.32e3	<0.1
102	6	11/11	5.11e6	33.0	1.02e6	5.9
103	8	6/6	2.04e7	207.9	2.85e6	15.9
110	8	1/4	2.00e0	<0.1	2.00e0	<0.1
111	10	3/9	9.39e3	0.1	9.39e3	0.5
112	16	1/9	2.00e0	<0.1	2.00e0	<0.1
113	13	2/9	9.38e3	0.2	9.38e3	2.0
114	18	1/4	2.00e0	<0.1	2.00e0	<0.1
115	19	1/7	9.38e3	0.5	9.38e3	3.4
total		123/171	1.48e8	2249.9	3.10e7	306.4

Fig. 3. Comparison between LTSChecker$_y$ and LTSChecker$_n$. Statistics are for those satisfied(true) properties that are verified by both configurations (except 4 valid properties). The first column is the problem number. t101 stands for training problem 101. The second column is the parallelism and the third column is the number of solved properties. The fourth and fifth columns are the total number of states and the total time for LTSChecker$_n$, the sixth and seventh columns are for LTSChecker$_y$.

tions for some properties and may be able to find violations before a cycle is detected.

6 Conclusion

In this paper, we have described an ample set POR method for action-based formalisms. This method preserves a class of ALTL formulas we call the *blank-invariant* formulas. We have explored some properties of this class, and shown how to recognize a large set of blank-invariant formulas. Blank-invariance appears to be a natural restriction for ALTL properties, similar to, but different from, stutter-invariance. Based on our examination of RERS formulas, blank-invariant properties seem to be abundant in the wild.

The proof of soundness of the action-based ample set method is similar to that for the state-based method, and the action-based method can be easily used in the automata theoretic, on-the-fly model checking context. We developed a simple model checker, LTSChecker, realizing these ideas, and evaluated it on the 2017 RERS benchmark suite. The results indicate that action-based ample

Problem	Procs	Solved	LTS States	LTS Time(s)	Spin States	Spin Time(s)
t101	1	12/12	4.80e1	<0.1	4.80e1	<0.1
t102	2	14/14	2.94e2	<0.1	2.33e3	<0.1
t103	3	8/8	2.00e2	<0.1	2.39e3	<0.1
t104	4	11/11	1.63e3	<0.1	1.28e5	0.2
t105	5	9/9	8.13e4	4.2	1.75e7	15.3
t106	6	9/9	1.40e5	1.6	5.19e7	60.7
t107	7	8/8	3.45e7	262.3	1.38e9	3560.5
t111	11	2/11	2.69e5	2.0	3.07e8	944.0
101	4	7/7	1.32e3	<0.1	6.70e4	<0.1
102	6	11/11	1.02e6	5.9	3.17e7	34.1
103	8	6/6	2.85e6	15.6	5.54e8	1266.0
110	8	1/4	2.00e0	<0.1	1.50e1	<0.1
111	10	3/9	9.39e3	0.5	3.40e5	3.5
112	16	1/9	2.00e0	<0.1	2.00e1	<0.1
113	13	2/9	9.38e3	2.0	9.28e5	22.2
114	18	1/4	2.00e0	<0.1	2.50e1	<0.1
115	19	1/7	9.38e3	3.4	8.00e5	46.0
total		106/148	5.05e6	42.3	4.31e8	1197.9

Fig. 4. Comparison between LTSChecker$_y$ and Spin. Statistics are for those satisfied(`true`) properties that are verified by both tools (except 4 valid properties). The first column is the problem number and t101 stands for training problem 101. The second column is the parallelism and the third column is the number of solved properties. The fourth and fifth are the total number of states and the total time for the LTSChecker$_y$, and the sixth and seventh columns are for Spin.

set POR can result in significant reduction, and LTSChecker can perform better than Spin when verifying *true* properties. (There is no clear winner when it comes to *false* properties.)

Further study is needed to confirm these preliminary findings. We have tried various methods to translate an LTS model into the language of Spin (and used the best method we could find), but it is possible there is another method we have not tried that would extract better performance from Spin. Studies in which the same LTL-to-Büchi translator is used for both tools would also improve on the experiments reported here. On the theoretical side, a simple characterization of the blank-invariant properties—similar to the characterization of stutter-invariant properties as those expressible without using the next-time operator—remains to be found. The action-based ample set POR technique seems to improve on the current state-of-the-art, and raises several new and interesting questions.

References

1. Barnat, J., Brim, L., Havel, V., Havlek, J., Kriho, J., Leno, M., Rokai, P., till, V., Weiser, J.: Divine 3.0 an explicit-state model checker for multithreaded c & c programs p. 863868 (2013), https://doi.org/10.1007/978-3-642-39799-8_60
2. Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
3. Clarke, E., Grumberg, O., Minea, M., Peled, D.: State space reduction using partial order techniques. International Journal on Software Tools for Technology Transfer 2(3), 279–287 (Nov 1999), https://doi.org/10.1007/s100090050035
4. Cranen, S., Groote, J.F., Keiren, J.J.A., Stappers, F.P.M., de Vink, E.P., Wesselink, W., Willemse, T.A.C.: An overview of the mCRL2 toolset and its recent advances. In: Piterman, N., Smolka, S.A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 199–213. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), https://doi.org/10.1007/978-3-642-36742-7_15
5. Doron, P., Wilke, T.: Stutter-invariant temporal properties are expressible without the next-time operator. Information Processing Letters 63(5), 243 – 246 (1997), https://doi.org/10.1016/S0020-0190(97)00133-6
6. Geske, M., Jasper, M., Steffen, B., Howar, F., Schordan, M., van de Pol, J.: RERS 2016: Parallel and sequential benchmarks with focus on LTL verification. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications. pp. 787–803. Springer International Publishing, Cham (2016), https://doi.org/10.1007/978-3-319-47169-3_59
7. Giannakopoulou, D.: Model checking for concurrent software architectures. Ph.D. thesis, Imperial College of Science, Technology and Medicine, University of London (1999), https://pdfs.semanticscholar.org/0215/b74b21112520569f6e6b930312e228c90e0b.pdf
8. Giannakopoulou, D., Lerda, F.: From states to transitions: Improving translation of LTL formulae to Büchi automata. In: Peled, D., Vardi, M. (eds.) Formal Techniques for Networked and Distributed Sytems – FORTE 2002, pp. 308–326. LNCS, Springer, http://dx.doi.org/10.1007/3-540-36135-9_20
9. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.W.: FDR3: a parallel refinement checker for CSP. International Journal on Software Tools for Technology Transfer 18(2), 149–167 (Apr 2016), https://doi.org/10.1007/s10009-015-0377-y
10. Gibson-Robinson, T., Hansen, H., Roscoe, A.W., Wang, X.: Practical partial order reduction for CSP. In: Havelund, K., Holzmann, G., Joshi, R. (eds.) NASA Formal Methods. pp. 188–203. Springer International Publishing, Cham (2015), https://doi.org/10.1007/978-3-319-17524-9_14
11. Godefroid, P.: Using partial orders to improve automatic verification methods. In: Clarke, E.M., Kurshan, R.P. (eds.) Computer-Aided Verification. pp. 176–185. Springer Berlin Heidelberg, Berlin, Heidelberg (1991), https://doi.org/10.1007/BFb0023731
12. Groote, J.F., Mathijssen, A., Reniers, M., Usenko, Y., van Weerdenburg, M.: The formal specification language mCRL2. In: Methods for Modelling Software Systems (MMOSS). Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, Dagstuhl, Germany (2007), http://drops.dagstuhl.de/opus/volltexte/2007/862

13. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM 21(8), 666–677 (Aug 1978), http://doi.acm.org/10.1145/359576.359585
14. Holzmann, G.J.: The model checker SPIN. IEEE Trans. Softw. Eng. 23(5), 279–295 (May 1997), http://dx.doi.org/10.1109/32.588521
15. Jasper, M., Fecke, M., Steffen, B., Schordan, M., Meijer, J., Pol, J.v.d., Howar, F., Siegel, S.F.: The RERS 2017 challenge and workshop (invited paper). In: Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software. pp. 11–20. SPIN 2017, ACM, New York, NY, USA (2017), http://doi.acm.org/10.1145/3092282.3098206
16. Peled, D.: All from one, one for all: on model checking using representatives. In: Courcoubetis, C. (ed.) Computer Aided Verification. pp. 409–423. Springer Berlin Heidelberg, Berlin, Heidelberg (1993), https://doi.org/10.1007/3-540-56922-7_34
17. Sun, J., Liu, Y., Dong, J.S.: Model checking CSP revisited: Introducing a process analysis toolkit. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. pp. 307–322. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), https://doi.org/10.1007/978-3-540-88479-8_22
18. Valmari, A.: Stubborn sets for reduced state space generation. In: Rozenberg, G. (ed.) Advances in Petri Nets 1990. pp. 491–515. Springer Berlin Heidelberg, Berlin, Heidelberg (1991), https://doi.org/10.1007/3-540-53863-1_36
19. Valmari, A.: Stubborn set methods for process algebras. In: Proceedings of the DIMACS Workshop on Partial Order Methods in Verification. pp. 213–231. POMIV '96, AMS Press, Inc., New York, NY, USA (1997), http://dl.acm.org/citation.cfm?id=266557.266608
20. Valmari, A.: The state explosion problem. Lectures on Petri Nets I: Basic Models Lecture Notes in Computer Science pp. 429–528 (1998), https://doi.org/10.1007/3-540-65306-6_21
21. Valmari, A.: More Stubborn Set Methods for Process Algebras, pp. 246–271. Springer International Publishing, Cham (2017), https://doi.org/10.1007/978-3-319-51046-0_13
22. Valmari, A., Hansen, H.: Stubborn Set Intuition Explained, pp. 140–165. Springer Berlin Heidelberg, Berlin, Heidelberg (2017), https://doi.org/10.1007/978-3-662-55862-1_7
23. Yan, Y., Siegel, S.F.: Ample set partial order reduction for actions technical report (2018), http://vsl.cis.udel.edu/pubs/ample_set_action_2018.html
24. Zheng, M., Rogers, M.S., Luo, Z., Dwyer, M.B., Siegel, S.F.: Civl: Formal verification of parallel programs. In: Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 830–835. ASE '15, IEEE Computer Society, Washington, DC, USA (2015), https://doi.org/10.1109/ASE.2015.99

A Proof of Proposition 1

A.1 Notation.

Given $\zeta \in \mathsf{Act}^\omega$, and $x, y \in \mathsf{Act}$, we write $\zeta[x/y]$ for the sequence obtained by replacing every occurrence of y in ζ by x.

Definition 15. Let f be an ALTL formula over Act and $a \in \mathsf{Act}$. We say a is *irrelevant* to f if the following holds for every $x \in \mathsf{Act} \setminus \alpha f$:

$$\forall \zeta \in \mathsf{Act}^\omega \,.\, (\zeta \models f \iff \zeta[x/a] \models f). \tag{2}$$

Definition 15 requires that (2) hold *for all* x not in the alphabet of f. Lemma 5 shows that it is enough to check this for one such x:

Lemma 5. *Let f be an ALTL formula over Act and $a \in \mathsf{Act}$. Suppose $x \in \mathsf{Act} \setminus \alpha f$ and x satisfies (2). Then a is irrelevant to f.*

Proof. It suffices to show

$$\forall \zeta \in \mathsf{Act}^\omega,\ x, y \in \mathsf{Act} \setminus \alpha f \,.\, (\zeta[x/a] \models f \iff \zeta[y/a] \models f). \tag{3}$$

We prove (3) by induction on the structure of f.

It clearly holds if f is true. Suppose $f = b \in \mathsf{Act}$. If $a = b$ then neither $\zeta[x/b]$ nor $\zeta[y/b]$ satisfy b, so (3) holds. If $a \neq b$ then

$$\zeta[x/a] \models b \iff \mathsf{first}(\zeta) = b \iff \zeta[y/a] \models b,$$

so (3) holds again.

Suppose (3) holds for f, and x is not in

$$\begin{aligned} \zeta[x/a] \models \neg f &\iff \zeta[x/a] \not\models f \\ &\iff \zeta[y/a] \not\models f \\ &\iff \zeta[y/a] \models \neg f \end{aligned}$$

showing (3) holds for $\neg f$.

Suppose (3) holds for f and g, and x, y are not in $\alpha(f \wedge g) = \alpha(f) \cup \alpha(g)$. Then

$$\begin{aligned} \zeta[x/a] \models f \wedge g &\iff \zeta[x/a] \models f \text{ and } \zeta[x/a] \models g \\ &\iff \zeta[y/a] \models f \text{ and } \zeta[y/a] \models g \\ &\iff \zeta[y/a] \models f \wedge g, \end{aligned}$$

proving (3) for $f \wedge g$. Moreover,

$$\begin{aligned} \zeta[x/a] \models f\mathbf{U}g &\iff \exists i.(i \geq 0 \wedge \zeta[x/a]^i \models g \wedge \forall j.(0 \leq j < i \to \zeta[x/a]^j \models f)) \\ &\iff \exists i.(i \geq 0 \wedge \zeta^i[x/a] \models g \wedge \forall j.(0 \leq j < i \to \zeta^j[x/a] \models f)) \\ &\iff \exists i.(i \geq 0 \wedge \zeta^i[y/a] \models g \wedge \forall j.(0 \leq j < i \to \zeta^j[y/a] \models f)) \\ &\iff \exists i.(i \geq 0 \wedge \zeta[y/a]^i \models g \wedge \forall j.(0 \leq j < i \to \zeta[y/a]^j \models f)) \\ &\iff \zeta[y/a] \models f\mathbf{U}g, \end{aligned}$$

proving (3) for $f\mathbf{U}g$.

Now suppose (3) holds for f, $f \equiv_A g$, $\alpha f = \alpha g$, and $x, y \notin \alpha f$. Then

$$\zeta[x/a] \models g \iff \zeta[x/a] \models f \iff \zeta[y/a] \models f \iff \zeta[y/a] \models g,$$

so (3) holds for g. Since

$$\begin{aligned} f \to g &\equiv (\neg f) \vee g \\ f \vee g &\equiv \neg((\neg f) \wedge \neg g) \\ \mathbf{F}f &\equiv \mathsf{true}\mathbf{U}f \\ \mathbf{G}f &\equiv \neg\mathbf{F}\neg f \end{aligned}$$

the remaining cases follow. □

Lemma 6 states that any action not in the alphabet of f is irrelevant to f:

Lemma 6. *Let f be an ALTL formula over Act and $a \in \mathsf{Act} \setminus \alpha f$. Then a is irrelevant to f.*

Proof. Let $x = a$. Then $\zeta[x/a] = \zeta$, so (2) clearly holds. By Lemma 5, a is irrelevant to f. □

Actions that do occur in f may also be irrelevant to f. For example, $a \wedge \neg a$ is equivalent to false, so clearly a is irrelevant to $a \wedge \neg a$. Similarly, if a and b are distinct actions, then $a \wedge b$ is action-equivalent to false, so both a and b are irrelevant to $a \wedge b$. More generally:

Lemma 7. *Suppose f and g are action-equivalent ALTL formulas, and $a \in \mathsf{Act}$. Then a is irrelevant to f if, and only if, a is irrelevant to g.*

Proof. Suppose a is irrelevant to f. Since Act is infinite, there is some $x \in \mathsf{Act} \setminus (\alpha f \cup \alpha g)$. Then for any $\zeta \in \mathsf{Act}^\omega$,

$$\zeta \models g \iff \zeta \models f \iff \zeta[x/a] \models f \iff \zeta[x/a] \models g.$$

By Lemma 5, a is irrelevant to g. □

Now we prove

Proposition 1. *Suppose f and g are action-equivalent ALTL formulas over Act. Then $f \in \mathsf{BInv} \iff g \in \mathsf{BInv}$.*

Proof. Let $Y = \alpha f \setminus \alpha g$. Say $Y = \{y_1, \ldots, y_n\}$. Let $x \in \mathsf{Act} \setminus (\alpha f \cup \alpha g)$. For $\zeta \in \mathsf{Act}^\omega$, we write $\zeta[x/Y]$ as shorthand for $\zeta[x/y_1] \cdots [x/y_n]$. Note

$$\zeta[x/Y]|_{\alpha f} = \zeta|_{\alpha f \cap \alpha g} = \zeta|_{\alpha g}|_{\alpha f \cap \alpha g}. \tag{4}$$

Suppose f is blank-invariant. We must show g is blank-invariant. So assume $\zeta, \eta \in \mathsf{Act}^\omega$ and $\zeta|_{\alpha g} = \eta|_{\alpha g}$. By (4),

$$\zeta[x/Y]|_{\alpha f} = \zeta|_{\alpha g}|_{\alpha f \cap \alpha g} = \eta|_{\alpha g}|_{\alpha f \cap \alpha g} = \eta[x/Y]|_{\alpha f},$$

whence $\zeta[x/Y]$ and $\eta[x/Y]$ are blank-equivalent for f.

By Lemma 6, every action in Y is irrelevant to g. Therefore

$$\begin{aligned}
\zeta \models g &\iff \zeta[x/Y] \models g && \text{by Definition 15} \\
&\iff \zeta[x/Y] \models f && \text{since } f \equiv_A g \\
&\iff \eta[x/Y] \models f && \text{as } \zeta[x/Y], \eta[x/Y] \text{ are blank-equivalent for } f \\
&\iff \eta[x/Y] \models g && \text{since } f \equiv_A g \\
&\iff \eta \models g && \text{by Definition 15.}
\end{aligned}$$

Hence g is blank-invariant. □

B Proof of Proposition 2

In this section, we prove

Proposition 2. *Suppose* $f, g \in \mathsf{BInv}$. *Then* true, $\neg f$, $f \wedge g$, $f \vee g$, $f \to g$, $\mathbf{F}f$, $\mathbf{G}f$, *and* $f\mathbf{U}g$ *are all in* BInv.

Proof. The proof is by induction over the formula structure.

Suppose ζ and η are blank-equivalent for f. Then

$$\zeta \models \neg f \iff \zeta \not\models f \iff \eta \not\models f \iff \eta \models \neg f,$$

so $\neg f$ is blank-invariant.

Suppose ζ and η are blank-equivalent for $f \wedge g$. Note $\alpha(f \wedge g) = \alpha f \cup \alpha g$, so

$$\zeta|_{\alpha f} = \zeta|_{\alpha(f \wedge g)}|_{\alpha f} = \eta|_{\alpha(f \wedge g)}|_{\alpha f} = \eta|_{\alpha f},$$

so ζ and η are blank-equivalent for f. Similarly, ζ and η are blank-equivalent for g. Hence

$$\begin{aligned}
\zeta \models f \wedge g &\iff \zeta \models f \text{ and } \zeta \models g \\
&\iff \eta \models f \text{ and } \eta \models g \\
&\iff \eta \models f \wedge g,
\end{aligned}$$

which shows $f \wedge g$ is blank-invariant.

Let us show the case for $f\mathbf{U}g$. The other cases follow since they are action-equivalent to formulas expressed using $\mathbf{U}$, true, $\neg$, and $\wedge$.

Suppose $\zeta \models f\mathbf{U}g$. Then there is some $i \geq 0$ such that $\zeta^i \models g$ and $\zeta^j \models f$ whenever $0 \leq j < i$.

Since ζ and η are blank-equivalent, there exists $i' \geq 0$ such that the prefix of length i of ζ is blank-equivalent to the prefix of length i' of η, and the suffix ζ^i is blank-equivalent to $\eta^{i'}$.

For $j' < i'$, $\eta^{j'} \models f$ since $\eta^{j'}$ is blank equivalent to ζ^j for some j satisfying $0 \leq j < i$. Similarly, $\eta^{i'} \models g$ since $\eta^{i'}$ is blank-equivalent to ζ^i. Hence $\eta \models f\mathbf{U}g$, as required. □

C Proof of Proposition 3

In this section, we prove

Proposition 3. *There exist sets* Pos *and* Neg *of ALTL formulas satisfying (i) for all ALTL formulas* f *and* f'*,*

$$(f \in \mathsf{Pos} \wedge f' \equiv_A f) \implies f' \in \mathsf{Pos}$$
$$(f \in \mathsf{Neg} \wedge f' \equiv_A f) \implies f' \in \mathsf{Neg},$$

and (ii) for all $a \in \mathsf{Act}$*,* $f_1, f_2 \in \mathsf{BInv}$*,* $g_1, g_2 \in \mathsf{Pos}$*, and* $h_1, h_2 \in \mathsf{Neg}$*,*

$$\mathsf{false},\ a,\ \neg h_1,\ g_1 \wedge g_2,\ g_1 \vee g_2,\ a \wedge f_1,\ a \wedge \mathbf{X} f_1 \ \in\ \mathsf{Pos}$$
$$\mathsf{true},\ \neg a,\ \neg g_1,\ h_1 \wedge h_2,\ h_1 \vee h_2,\ \neg a \vee f_1,\ \neg a \vee \mathbf{X} f_1 \ \in\ \mathsf{Neg}$$
$$\mathsf{true},\ \mathsf{false},\ f_1 \wedge f_2,\ f_1 \vee f_2,\ \neg f_1,\ \mathbf{F} g_1,\ \mathbf{G} h_1,\ f_1 \mathbf{U} f_2,\ h_1 \mathbf{U} g_1,\ h_1 \mathbf{U} f_1 \ \in\ \mathsf{BInv}.$$

C.1 Definition of positive and negative formulas

Let Pos be the set of all ALTL formulas f satisfying both of the following:

$$\forall a \in \mathsf{Act}, \zeta, \eta \in \mathsf{Act}^\omega \,.\, (\zeta|_{\alpha f} = \eta|_{\alpha f} \implies (a \circ \zeta \models f \iff a \circ \eta \models f)) \tag{5}$$

$$\forall \zeta \in \mathsf{Act}^\omega \,.\, (\mathsf{first}(\zeta) \notin \alpha f \implies \zeta \not\models f). \tag{6}$$

We say a formula in Pos is *positive*.

Let Neg be the set of all ALTL formulas f satisfying both (5) and

$$\forall \zeta \in \mathsf{Act}^\omega \,.\, (\mathsf{first}(\zeta) \notin \alpha f \implies \zeta \models f). \tag{7}$$

We say a formula in Neg is *negative*.

Lemma 8. *Suppose* f *and* g *are ALTL formulas over* Act *and* $f \equiv_A g$*. All of the following hold:*

1. $f \in \mathsf{Pos} \iff g \in \mathsf{Pos}$*,*
2. $f \in \mathsf{Neg} \iff g \in \mathsf{Neg}$*,*
3. $f \in \mathsf{Pos} \iff \neg f \in \mathsf{Neg}$*, and*
4. $f \in \mathsf{Neg} \iff \neg f \in \mathsf{Pos}$*.*

Proof. Clearly, (5) holds for f if and only if (5) holds for $\neg f$. Equation (6) holds for f if and only if (7) holds for $\neg f$. This proves Lemma 8(3). Moreover, (7) holds for f if and only if (6) holds for $\neg f$, proving Lemma 8(4).

Now if Lemma 8(1) holds, we can prove Lemma 8(2): if f is negative then $\neg f$ is positive, so since $\neg f \equiv_A \neg g$, $\neg g$ is positive, therefore g is negative.

We now turn to the proof of Lemma 8(1). Let $Y = \alpha f \setminus \alpha g$ and $x \in \mathsf{Act} \setminus (\alpha f \cup \alpha g)$. Assume f is positive. We wish to show g is positive.

We first show g satisfies (5). So assume $\zeta|_{\alpha g} = \eta|_{\alpha g}$ and $a \circ \zeta \models g$. We must show $a \circ \eta \models g$. First, since $f \equiv_A g$, we have

$$a \circ \zeta \models f. \tag{8}$$

Moreover,

$$\begin{aligned} \zeta[x/Y]|_{\alpha f} &= \zeta|_{\alpha f \cap \alpha g} = \zeta|_{\alpha g}|_{\alpha f \cap \alpha g} = \eta|_{\alpha g}|_{\alpha f \cap \alpha g} = \eta|_{\alpha f \cap \alpha g} \\ &= \eta[x/Y]|_{\alpha f}. \end{aligned} \tag{9}$$

By Lemma 6, every element of Y is irrelevant to g, and by Lemma 7, every element of Y is irrelevant to f. From (8), we can therefore conclude

$$(a \circ \zeta)[x/Y] \models f. \tag{10}$$

We claim $a \notin Y$. For if $a \in Y$, then $(a \circ \zeta)[x/Y] = x \circ \zeta[x/Y] \models f$ but $\mathsf{first}(x \circ \zeta[x/Y]) = x \notin \alpha f$, so (6) implies $x \circ \zeta[x/Y] \not\models f$, contradicting (10).

Therefore

$$a \circ \zeta[x/Y] = (a \circ \zeta)[x/Y] \models f. \tag{11}$$

But f is positive, so by (5), (9), and (11),

$$a \circ \eta[x/Y] \models f.$$

Since $f \equiv_A g$,

$$(a \circ \eta)[x/Y] = a \circ \eta[x/Y] \models g,$$

and since Y is irrelevant to g, $a \circ \eta \models g$, completing the proof that g satisfies (5).

We next show g satisfies (6). Let $\zeta \in \mathsf{Act}^\omega$, $a = \mathsf{first}(\zeta)$, and suppose $a \notin \alpha g$. Say $\zeta = a \circ \xi$. We have

$$\zeta[x/a] = (a \circ \xi)[x/a] = x \circ \xi[x/a].$$

Since $x \notin \alpha f$ and f satisfies (6), $x \circ \xi[x/a] \not\models f$. Since $f \equiv_A g$, $\zeta[x/a] \not\models g$. By Lemma 6, a is irrelevant to g, so $\zeta \not\models g$. This shows g satisfies (6), completing the proof of Lemma 8(1). □

C.2 Positive formulas

The proof of Proposition 3 is by induction on formula structure. We begin with the formulas in Pos.

Case false Since formula false trivially satisfies (5) and (6), false is clearly positive.

Case a Let $a \in \mathsf{Act}$. For any $x \in \mathsf{Act}$,

$$x \circ \zeta \models a \iff x = a \iff x \circ \eta \models a,$$

so (5) holds for the formula a. Clearly, if ζ does not start with a then ζ cannot satisfy a, so (6) holds as well. Hence the formula a is positive.

Case $\neg h$ Suppose $h \in \mathsf{Neg}$. By the inductive hypothesis, h is negative. By Lemma 8(4), $\neg h$ is positive.

Case $g_1 \wedge g_2$ Suppose $g_1, g_2 \in \mathsf{Pos}$. By the inductive hypothesis, g_1 and g_2 are positive. We show $g_1 \wedge g_2$ is positive. Let $S = \alpha(g_1 \wedge g_2) = \alpha g_1 \cup \alpha g_2$. Suppose $a \in \mathsf{Act}$, $\zeta, \eta \in \mathsf{Act}^\omega$, $\zeta|_S = \eta|_S$, and $a \circ \zeta \models g_1 \wedge g_2$. Then $a \circ \zeta \models g_1$ and

$$\zeta|_{\alpha g_1} = \zeta|_S|_{\alpha g_1} = \eta|_S|_{\alpha g_1} = \eta|_{\alpha g_1}.$$

Since g_1 is positive, $a \circ \eta \models g_1$. A similar statement holds with g_2 in place of g_1. Hence $a \circ \eta \models g_1 \wedge g_2$, and $g_1 \wedge g_2$ satisfies (5). Furthermore, if $\mathsf{first}(\zeta) \notin S$, then in particular $\mathsf{first}(\zeta) \notin \alpha g_1$, so $\zeta \not\models g_1$, hence $\zeta \not\models g_1 \wedge g_2$. Therefore $g_1 \wedge g_2$ also satisfies (6), and $g_1 \wedge g_2$ is positive.

Case $g_1 \vee g_2$ One can argue exactly as above to see that $g_1 \vee g_2$ satisfies (5). If $\mathsf{first}(\zeta) \notin S$, then $\mathsf{first}(\zeta) \notin \alpha g_1$, whence $\zeta \not\models g_1$, and $\mathsf{first}(\zeta) \notin \alpha g_2$, whence $\zeta \not\models g_2$. Therefore $\zeta \not\models g_1 \vee g_2$, so $g_1 \vee g_2$ is positive.

Cases $a \wedge f$ and $a \wedge \mathbf{X} f$ Suppose $a \in \mathsf{Act}$ and f is blank-invariant. Let $S = \{a\} \cup \alpha f$, and assume $b \in \mathsf{Act}$ and $\zeta|_S = \eta|_S$.

Suppose $b \circ \zeta \models a \wedge f$. Then $a = b$ and $a \circ \zeta \models f$. Moreover,

$$(a \circ \zeta)|_{\alpha f} = (a \circ \zeta)|_S|_{\alpha f} = (a \circ \eta)|_S|_{\alpha f} = (a \circ \eta)|_{\alpha f}. \tag{12}$$

Since f is blank-invariant, $a \circ \eta \models f$, hence $a \circ \eta \models a \wedge f$, and $a \wedge f$ satisfies (5). Suppose $\mathsf{first}(\zeta) \notin S$. Then, in particular, $\mathsf{first}(\zeta) \neq a$, so $\zeta \not\models a \wedge f$, and $a \wedge f$ satisfies (6). So $a \wedge f$ is positive.

Suppose instead $b \circ \zeta \models a \wedge \mathbf{X} f$. Then $a = b$ and $a \circ \zeta \models \mathbf{X} f$. Hence $\zeta \models f$. Moreover, $\zeta|_{\alpha f} = \eta|_{\alpha f}$. Since f is blank-invariant, $\eta \models f$. Hence $a \circ \eta \models a \wedge \mathbf{X} f$, i.e., $a \wedge \mathbf{X} f$ satisfies (5). The argument that $a \wedge \mathbf{X} f$ satisfies (6) is identical to the one above. So $a \wedge \mathbf{X} f$ is positive.

C.3 Negative formulas

Each formula in Neg is easily seen to be equivalent to the negation of a formula in Pos. By Lemma 8, this completes the inductive step for Neg.

C.4 Blank-invariant formulas

Proposition 2 has already established that the blank-invariant formulas are closed under all LTL operators other than $\mathbf{X}$. Morover, $\mathbf{F}g \equiv \mathsf{true}\mathbf{U}g$, and $\mathsf{true} \in \mathsf{Neg}$; and $\mathbf{G}h \equiv \neg\mathbf{F}\neg h$. This leaves two cases: $h\mathbf{U}g$ and $h\mathbf{U}f$.

Case $h\mathbf{U}g$ Suppose h is negative and g is positive. We will show $h\mathbf{U}g$ is blank-invariant. Let $Y = \alpha h \cup \alpha g$. Suppose ζ and η are blank-equivalent for $h\mathbf{U}g$ and $\zeta \models h\mathbf{U}g$. Say $\zeta^i \models g$ and $\zeta^j \models h$ for $j < i$.

Let $a = \mathsf{first}(\zeta^i)$. Since g is positive, $a \in \alpha g$. Now since $\zeta|_Y = \eta|_Y$, there is a unique integer i' such that

$$\begin{aligned} \zeta &= \zeta_1 \circ a \circ \zeta^{i+1} \\ \eta &= \eta_1 \circ a \circ \eta^{i'+1} \\ \zeta_1|_Y &= \eta_1|_Y \\ \zeta^{i+1}|_Y &= \eta^{i'+1}|_Y, \end{aligned}$$

where ζ_1 is the prefix of length i of ζ and η_1 is the prefix of length i' of η. Since $a \circ \zeta^{i+1} = \zeta^i \models g$, we have $a \circ \eta^{i'+1} = \eta^{i'} \models g$.

Suppose $0 \le j' < i'$. Let $b = \mathsf{first}(\eta^{j'})$. If $b \notin Y$, then since h is negative, $\eta^{j'} \models h$. If $b \in Y$, then there exists a unique integer j such that

$$\begin{aligned} \zeta &= \zeta_2 \circ b \circ \zeta^{j+1} \\ \eta &= \eta_2 \circ b \circ \eta^{j'+1} \\ \zeta_2|_Y &= \eta_2|_Y \\ \zeta^{j+1}|_Y &= \eta^{j'+1}|_Y, \end{aligned}$$

where ζ_2 is the prefix of length j of ζ and η_2 is the prefix of length j' of η. Furthermore, $j < i$, so $b \circ \zeta^{j+1} \models h$. Since h is negative, $\eta^{j'} = b \circ \eta^{j'+1} \models h$. This shows $\eta \models h\mathbf{U}g$, so $h\mathbf{U}g$ is blank-invariant.

Case $h\mathbf{U}f$ Suppose h is negative and f is blank-invariant. We will show $h\mathbf{U}f$ is blank-invariant. Let $Y = \alpha h \cup \alpha g$. Suppose ζ and η are blank-equivalent for $h\mathbf{U}f$ and $\zeta \models h\mathbf{U}f$. Let i be the least nonnegative integer such that $\zeta^i \models f$ and $\zeta^j \models h$ for $j < i$.

If $i = 0$ then $\zeta \models f$, and therefore $\eta \models f$, since f is blank-invariant. Hence $\eta \models h\mathbf{U}f$, as required.

So assume $i > 0$. Let $a = \mathsf{first}(\zeta^{i-1})$. We must have $a \in \alpha f$, since otherwise ζ^{i-1} and ζ^i would be blank-equivalent for f, and therefore $\zeta^{i-1} \models f$, contradicting the minimality of i.

There exists a unique nonnegative integer i' such that

$$\begin{aligned}\zeta &= \zeta_1 \circ a \circ \zeta^i \\ \eta &= \eta_1 \circ a \circ \eta^{i'} \\ \zeta_1|_Y &= \eta_1|_Y \\ \zeta^i|_Y &= \eta^{i'}|_Y,\end{aligned}$$

where ζ_1 is the prefix of length i of ζ and η_1 is the prefix of length i' of η.

Since f is blank-invariant, $\eta^{i'} \models f$. Suppose $0 \leq j' < i'$. We will show $\eta^{j'} \models h$. If $j' = i' - 1$, then $\eta^{j'}|_Y = a \circ \eta^{i'} \models h$ since h is negative, $a \circ \zeta^i \models h$, and $\zeta^i|_Y = \eta^i|_Y$.

Suppose $j' < i' - 1$. Then we argue just as in the $h\mathbf{U}g$ case that $\eta^{j'} \models h$. This completes the proof of Proposition 3.

D An alternative characterization of irrelevance

We now turn to an equivalent formulation of the notion of *irrelevant* actions. If f and g are ALTL formulas, and $a \in \mathsf{Act}$, we write $f[g/a]$ for the ALTL formula obtained by replacing every occurrence of a in f with g.

Lemma 9. *Let f be an ALTL formula over Act, $a \in \mathsf{Act}$, $x \in \mathsf{Act} \setminus \alpha f$, and $\zeta \in \mathsf{Act}^\omega$. Then*

$$\zeta[x/a] \models f \iff \zeta \models f[\mathsf{false}/a] \tag{13}$$

Proof. Proof is by induction on the structure of f. The lemma certainly holds for the formula true. Suppose $b \in \mathsf{Act}$ and we wish to show it holds for the formula b. If $a \neq b$, then

$$\zeta[x/a] \models b \iff \mathsf{first}(\zeta) = b \iff \zeta \models b = b[\mathsf{false}/a],$$

so (13) holds. If $a = b$, then

$$\zeta[x/a] \models a \iff \mathsf{false} \iff \zeta \models \mathsf{false} = a[\mathsf{false}/a],$$

so (13) holds in this case as well.

Suppose the Lemma holds for f and g. Then

$$\begin{aligned}\zeta[x/a] \models f \wedge g &\iff \zeta[x/a] \models f \text{ and } \zeta[x/a] \models g \\ &\iff \zeta \models f[\mathsf{false}/a] \text{ and } \zeta \models g[\mathsf{false}/a] \\ &\iff \zeta \models f[\mathsf{false}/a] \wedge g[\mathsf{false}/a] \\ &\iff \zeta \models (f \wedge g)[\mathsf{false}/a],\end{aligned}$$

so the Lemma holds for $f \wedge g$. Furthermore,

$$\begin{aligned}\zeta[x/a] \models \neg f &\iff \zeta[x/a] \not\models f \\ &\iff \zeta \not\models f[\mathsf{false}/a] \\ &\iff \zeta \models \neg(f[\mathsf{false}/a]) \\ &\iff \zeta \models (\neg f)[\mathsf{false}/a],\end{aligned}$$

so it holds for $\neg f$. Finally,

$$\begin{aligned}
\zeta[x/a] \models f\mathbf{U}g &\iff \exists i.(i \geq 0 \wedge (\zeta[x/a]^i \models g) \wedge \forall j.(0 \leq j < i \rightarrow \zeta[x/a]^j \models f)) \\
&\iff \exists i.(i \geq 0 \wedge (\zeta^i[x/a] \models g) \wedge \forall j.(0 \leq j < i \rightarrow \zeta^j[x/a] \models f)) \\
&\iff \exists i.(i \geq 0 \wedge (\zeta^i \models g[\mathsf{false}/a]) \wedge \forall j.(0 \leq j < i \rightarrow \zeta^j \models f[\mathsf{false}/a])) \\
&\iff \zeta \models f[\mathsf{false}/a]\mathbf{U}g[\mathsf{false}/a] \\
&\iff \zeta \models (f\mathbf{U}g)[\mathsf{false}/a],
\end{aligned}$$

so the Lemma holds for $f\mathbf{U}g$.

The remaining cases follow from the standard equivalences. For example, for any formula h (including $h = \mathsf{false}$),

$$\begin{aligned}
(f \vee g)[h/a] &= f[h/a] \vee g[h/a] \\
&\equiv \neg((\neg f[h/a]) \wedge \neg g[h/a]) \\
&= (\neg((\neg f) \wedge \neg g))[h/a]
\end{aligned}$$

from which we can prove the Lemma holds for $f \vee g$. □

Proposition 4. *Let f be an ALTL formula over* Act*, and $a \in \mathsf{Act}$. Then a is irrelevant to f if, and only if, $f \equiv_A f[\mathsf{false}/a]$.*

Proof. Let x be any element of $\mathsf{Act} \setminus \alpha f$. (Such an element exists since Act is infinite and αf is finite.)

Suppose a is irrelevant to f. Let $\zeta \in \mathsf{Act}^\omega$. Then

$$\begin{aligned}
\zeta \models f &\iff \zeta[x/a] \models f && \text{by Definition 15} \\
&\iff \zeta \models f[\mathsf{false}/a] && \text{by Lemma 9}
\end{aligned}$$

whence $f \equiv_A f[\mathsf{false}/a]$.

Suppose $f \equiv_A f[\mathsf{false}/a]$. Let $\zeta \in \mathsf{Act}^\omega$. Then

$$\begin{aligned}
\zeta \models f &\iff \zeta \models f[\mathsf{false}/a] && \text{by Definition 8} \\
&\iff \zeta[x/a] \models f && \text{by Lemma 9.}
\end{aligned}$$

By Lemma 5, a is irrelevant to f. □