State focusing: Lazy abstraction for the mu-calculus*

Harald Fecher[1] and Sharon Shoham[2]

[1] Albert-Ludwigs-Universität Freiburg, Germany, fecher@informatik.uni-freiburg.de
[2] The Technicon, Haifa, Israel, sharonsh@cs.technion.ac.il

Abstract. A key technique for the verification of programs is counterexample-guided abstraction refinement (CEGAR). In a previous approach, we developed a CEGAR-based algorithm for the modal μ-calculus, where refinement applies only locally, i.e. *lazy abstraction* techniques are used. Unfortunately, our previous algorithm was not completely lazy and had some further drawbacks, like a possible local state explosion. In this paper, we present an improved algorithm that maintains all advantages of our previous algorithm but eliminates all its drawbacks. The improvements were only possible by changing the philosophy of refinement from *state splitting* into the new philosophy of *state focusing*, where the states that are about to be split are not removed.

1 Introduction

The modal μ-calculus [19] is an expressive modal logic that allows to express safety, reachability, and mixtures of these properties, by using fixpoint constructions. The μ-calculus is a sensible choice for branching time properties, which are relevant whenever nondeterminism occurs from external factors (e.g. user input), from the modeling of faulty systems/channels [9], or from the abstraction of time or arguments [7]. In particular, the μ-calculus can express most of the standard logics, like LTL and CTL. Hence, the μ-calculus is an ideal basis for foundation-examinations.

For automatic verification of properties, the state explosion has to be tackled. One of the most successful techniques to checking correctness of large or even infinite programs is predicate abstraction [12] with *counterexample-guided abstraction refinement* (CEGAR) [5]. This approach consists of three phases: abstraction, model checking, and refinement. Refinement is performed by adding a new predicate that splits the abstract states. A prominent safety-checking tool based on CEGAR is BLAST [16], where refinement is applied locally (called *lazy abstraction*), i.e., only the abstract states of a trace which comprises a spurious counterexample are refined. This avoids the state space doubling obtained when the whole state space is split via a new predicate.

Adapting the idea of lazy abstraction to the μ-calculus is not straightforward. One reason is that in order to preserve branching time properties, an abstract model needs two kinds of transitions (called *may*, respectively *must, transitions*). Examples of such models are (Kripke) modal transition systems [20, 17]. They also allow to preserve both *validity* and *invalidity* from the abstract model to the concrete model, at the cost of introducing a third truth value *unknown*, which means that the truth value in the concrete

* This work is financially supported by the DFG projects (FE 942/2-1) and (SFB/TR 14 AVACS).

model is unknown. This leads to a *3-valued semantics*. In this setting, refinement is no longer needed when the result is *invalid*, as in traditional CEGAR approaches. Instead, refinement is needed when the result is *unknown*. As such, the role of a counterexample as guiding the refinement is taken by some cause of the indefinite result.

In [10], we have developed a preliminary algorithm for μ-calculus verification, having the following advantages: (i) Refinement is made lazily. More precisely, some, but not all, *configurations* (abstract states combined with subproperties) having the same abstract state are split during a refinement phase: The state space remains smaller and verification is sped up. (ii) The more expressive *generalized Kripke modal transition systems* [24] are used as underlying abstract models (they have must hypertransitions, i.e. transitions pointing to a set of states rather than to a singleton): A smoother refinement determination is obtained [24] and more properties (in principle, every least fixpoint free μ-calculus formula) can be shown. (iii) Refinement determination is separated from the model checking: Refinement-heuristics can be defined independently. (iv) Configurations and transitions that become irrelevant by newly obtained information of (in)validity are removed: Complexity is reduced.

The algorithm of [10] still has the following disadvantages: (I) A set of configurations rather than the single configuration determined by a refinement-heuristic is split: Verification is unnecessarily slowed down, since often expensive splits that do not contribute to the verification are made. (II) All may-/must-transitions that can arise as a result of a split are calculated even if they are not needed: Avoidable, expensive satisfiability checks are made. (III) An exponential blowup can occur during a refinement phase, since all hypertransitions obtained as the powerset of the may-transitions are calculated. (IV) Some interesting information for defining refinement-heuristics is lost: The split of accompanying configurations along with the one that is determined by the refinement-heuristic obscures the intermediate results obtained after each split, and might divert the refinement into undesired directions. These disadvantages cannot be eliminated with existing techniques, with the exception of (III) that was addressed by [24, 25], yet their approaches rely on particular model checking algorithms.

Contribution. We develop the new technique of *state focusing* for refinement: the states that are about to be split are not removed. Instead, the 'old' states are connected to the 'new' states that result from their split via *focus-transitions*. This allows to encode hypertransitions, but more importantly, it allows to perform a local refinement, in which propagation of a split is deferred until it is called for by a refinement-heuristic. We use this new technique to construct a new lazy, CEGAR-based algorithm for the μ-calculus. Our algorithm uses a configuration structure (where the abstract states are combined with subproperties) to encode the verification problem. In each iteration, (in)valid configurations are determined, the structure is simplified accordingly, a refinement-heuristic is used to determine a refinement step, and refinement is performed *locally* by either splitting (focusing) one configuration, or propagating a previous split to other configurations or components of the structure. Our new algorithm still has all the advantages (i) – (iv) and additionally does not have the disadvantages (I) – (IV). In particular, our algorithm combines the following properties:

- At most two configurations are added during a refinement step.
- No (in)validity is lost during a refinement or simplification calculation.

– Every satisfiability check is made on demand, i.e. unnecessary satisfiability checks are avoided except if called for by the refinement-heuristic.
– The capability of verification with generalized Kripke modal transition systems as the underlying abstract models is preserved while avoiding state explosion, since only the hypertransitions obtained constructively via old configurations are present, as in the non-lazy abstraction approach of [24].
– Improved simplifications of the underlying configuration structure are made by using a 9-valued logic, which is an extension of the 6-valued one used in [3].
– A separation of the refinement determination from the model checking is made, allowing to define refinement-heuristics separately.
– Improved refinement-heuristics (compare with (IV)) are possible, since only elementary updates of the configurations structure are made during a refinement step.
– Other refinements (e.g., that of [10]) can be imitated at no further cost by gathering together several local refinement steps.

Further related work. The state space doubling occurring after splitting the whole state space via a predicate can also be tackled by the usage of BDDs, as in the tool SLAM [4]. There the abstract transition relation is encoded as a BDD, avoiding the explicit calculation of the exponentially large state space. Such a BDD approach is generalized from the safety properties checked by SLAM to μ-calculus properties in the tool YASM [15]. There, the underlying abstract model is equivalent to a Kripke modal transition system, which is less expressive than generalized Kripke modal transition systems.

A CEGAR-approach to branching time properties is given in [23], where, contrary to our approach, only the transition relation is under, resp., over approximated (the state space remains unchanged). In [14, 13], CEGAR-based algorithms for the μ-calculus are presented having only Kripke modal transition systems as underlying abstract model. Furthermore, there every configuration for which (in)validity is not yet shown is split, i.e. only a weak form of lazy abstraction is made.

In [22] models are abstracted by *alternating transition systems* with *focus predicates*. These resemble game-graphs with must-hypertransitions. Refinement is not discussed in this paper. Must-hypertransitions were first introduced in disjunctive modal transition systems [21]. A CEGAR-approach for the more general alternating μ-calculus is given in [1], where must- as well as may-hypertransitions are used in the underlying abstract model. Refinement is made globally (not locally) and the refinement determination depends on the model checking algorithm, i.e. no separation is used. [2] increases expressiveness without using hypertransitions: Backward must-transitions and entry/exit points are used to conclude the existence of transitive must-transitions.

In [11], cartesian abstraction, where 'previous' abstract states also remain existent, is used for improving under approximations. However, there, the 'old' states are not used to encode hypertransitions, and thus less expressive abstractions are obtained. Moreover, our technique also improves the over approximation by forbidding may-transitions subsumed by may-transitions whose targets are more precise (less abstract).

2 Underlying structures

Notations. Throughout, functional composition is denoted by $\circ$. Given a relation $\rho \subseteq B \times D$ with subsets $X \subseteq B$ and $Y \subseteq D$ we write $X.\rho$ for $\{d \in D \mid \exists b \in X : (b, d) \in \rho\}$

and $\rho.Y$ for $\{b \in B \mid \exists d \in Y\colon (b,d) \in \rho\}$. Let $\mathrm{map}(f, \Phi)$ be the sequence obtained from the sequence Φ by applying function f to all elements of Φ pointwise. $f[b \mapsto d]$ denotes the function that behaves as f except on b, which is mapped to d. Suppose D is ordered, then $\sqsubseteq\ \subseteq (B \to D) \times (B \to D)$ denotes the derived pointwise order between functions in $B \to D$. Furthermore, for $f, f' : B \to D$ expression $f \sqcup f'$ denotes the least function (if existent) that is above f and f' w.r.t. $\sqsubseteq$.

System. Without loss of generality, we will not consider action labels on models in this paper. A *rooted transition system* $T = (S, s^{\mathrm{i}}, \to, \mathcal{L})$ consists of a (possibly infinite) set S of states, an initial state $s^{\mathrm{i}} \in S$, a transition relation $\to\ \subseteq S \times S$, and a *predicate language* $\mathcal{L}$, which is a set of predicates that are interpreted over the states in S (i.e. each predicate $p \in \mathcal{L}$ denotes a set $[\![p]\!] \subseteq S$), such that there exists $p^{\mathrm{i}} \in \mathcal{L}$ with $[\![p^{\mathrm{i}}]\!] = \{s^{\mathrm{i}}\}$. The boolean and exact predecessor closure of $\mathcal{L}$ is denoted by $\overline{\mathcal{L}}$, where $[\![\_]\!]$ over boolean operators is straightforwardly extended and $[\![\mathrm{pre}(\psi)]\!] = \to .[\![\psi]\!]$ for $\psi \in \overline{\mathcal{L}}$.

Intermediate games. They are a generalization of the three-valued parity games of [10, 13] by using a third-kind of states (called intermediate game states) that are not controlled by a unique player, but change the player depending on the type of the play (validity vs. invalidity). The intermediate game states are used to model state-focusing: The states to be split become intermediate game states. Intermediate games also use a more complex validity image to improve refinement and simplification determinations:

Definition 1. *An* intermediate game $G = (C_0, C_1, C_{\frac{1}{2}}, C^{\mathrm{i}}, R, R^-, R^+, \theta, \omega)$ *has*
- *pairwise disjoint sets of game states for Player 0 (C_0), for Player 1 (C_1), and intermediate game states $C_{\frac{1}{2}}$; the union of all game states is denoted $C = C_0 \cup C_{\frac{1}{2}} \cup C_1$,*
- *a set of initial game states $C^{\mathrm{i}} \subseteq C$,*
- *a set of normal transitions $R \subseteq C \times C$,*
- *a set of must- and a set of may-transitions $R^-, R^+ \subseteq (C_0 \cup C_1) \times C$,*
- *a parity function $\theta : C \to \mathbb{N}$ with finite image, and*
- *a validity function $\omega : C \to \{\mathrm{tt},\mathrm{ff},\bot\} \times \{\mathrm{tt},\mathrm{ff},\bot\}$, where we write ω_1, respectively ω_2, for applying the projection to the first, respectively second, component of ω.*

The values of ω are explained in detail in Sec. 3. In general, ω_1 is used to determine the winner in the validity game, whereas ω_2 is used in the invalidity game:

Definition 2.
- *Finite validity (resp. invalidity) plays for intermediate game G have the rules and winning conditions as stated in Table 1. An infinite play Φ is a* win *for Player 0 iff* $\sup(\mathrm{map}(\theta, \Phi))$ *is even; otherwise it is won by Player 1.*
- G is valid *(*invalid*) in $c \in C$ iff Player 0 (resp. Player 1) has a strategy for the corresponding validity (resp. invalidity) game such that Player 0 (resp. Player 1) wins all validity (resp. invalidity) plays started at c with her strategy.* G is valid *(*invalid*) iff G is valid (resp. invalid) in all games states of C^{i}.*

In the validity game, Player 0 has the role of the checker, and Player 1 has the role of the refuter. In the invalidity game their roles switch. In both cases, must-transitions are only used by the checker, whereas may-transitions are only used by the refuter. In addition, the $C_{\frac{1}{2}}$ game states are always controlled by the refuter.

Remark 1. The validity, as well as the invalidity, game obviously correspond to a parity game. Therefore, decidability of validity, resp. invalidity, is in UP $\cap$ coUP [18].

Moves of the validity game:
$\omega_1(c) \neq \bot$: Player 0 wins if $\omega_1(c) = \mathrm{tt}$; otherwise Player 1 wins;
$\omega_1(c) = \bot$ and $c \in C_0$: Player 0 picks as next game state $c' \in \{c\}.(R \cup R^-)$;
$\omega_1(c) = \bot$ and $c \in C_{\frac{1}{2}} \cup C_1$: Player 1 picks as next game state $c' \in \{c\}.(R \cup R^+)$;

Moves of the invalidity game:
$\omega_2(c) \neq \bot$: Player 1 wins if $\omega_2(c) = \mathrm{ff}$; otherwise Player 0 wins;
$\omega_2(c) = \bot$ and $c \in C_1$: Player 1 picks as next game state $c' \in \{c\}.(R \cup R^-)$;
$\omega_2(c) = \bot$ and $c \in C_0 \cup C_{\frac{1}{2}}$: Player 0 picks as next game state $c' \in \{c\}.(R \cup R^+)$;

Table 1. Moves of (in)validity game at game state c, specified through a case analysis. A player also wins if its opponent has to move but cannot.

Property language. We use an automata representation of the μ-calculus [19]:

Definition 3 (Tree automata). *An* (alternating tree) automaton $A = (Q, q^{\mathrm{i}}, \delta, \Theta)$ *has*

- *a finite, nonempty set of states* $(q \in)Q$ *with the initial element* $q^{\mathrm{i}} \in Q$
- *a transition relation* δ *mapping automaton-states to one of the following forms, where* q, q_1, q_2 *are automaton states and* $p \in \mathcal{L}$*:* $p \mid \neg p \mid q_1 \widetilde{\wedge} q_2 \mid q_1 \widetilde{\vee} q_2 \mid \Diamond q \mid \Box q$
- *an acceptance condition* $\Theta \colon Q \to \mathbb{N}$ *with finite image.*

An example of an automaton is depicted in Fig. 1 (α) on page 9, where all automaton-states have acceptance value 0. In the following, $Q_{\mathcal{L}} = \{q \in Q \mid \delta(q) \in \mathcal{L}\}$, $Q_{\mathrm{qua}} = \{q \in Q \mid \delta(q) \in \bigcup_{q' \in Q} \{\Diamond q', \Box q'\}\}$ and $Q_0 = \{q \in Q \mid \delta(q) \in \bigcup_{q_1, q_2 \in Q} \{q_1 \widetilde{\vee} q_2, \Diamond q_1\}\}$ resp. $Q_1 = \{q \in Q \mid \delta(q) \in \bigcup_{q_1, q_2 \in Q} \{q_1 \widetilde{\wedge} q_2, \Box q_1\}\}$ denotes those under control of Player 0 resp. Player 1. Satisfaction of a rooted transition system w.r.t. an automaton is obtained via transformation into an intermediate game:

Definition 4. *The* property-game *for* T *and* A*, denoted* $P_{T,A}$*, is an intermediate game* $(S \times Q_0, S \times Q_1, S \times Q_{\mathcal{L}}, \{(s^{\mathrm{i}}, q^{\mathrm{i}})\}, R, R^-, R^+, \Theta \circ \pi_Q, \omega)$*, where* π_Q *denotes the projection to the second component and*

$$R = \{((s,q),(s,q')) \mid \exists q'' : \delta(q) \in \{q' \widetilde{\wedge} q'', q'' \widetilde{\wedge} q', q' \widetilde{\vee} q'', q'' \widetilde{\vee} q'\}\}$$
$$R^- = R^+ = \{((s,q),(s',q')) \mid \delta(q) \in \{\Diamond q', \Box q'\} \wedge (s,s') \in \rightarrow\}$$
$$\omega(s,q) = \begin{cases} (\mathrm{tt},\mathrm{tt}) & \textit{if } q \in Q_{\mathcal{L}}, s \in [\![\delta(q)]\!] \\ (\mathrm{ff},\mathrm{ff}) & \textit{if } q \in Q_{\mathcal{L}}, s \notin [\![\delta(q)]\!] \\ (\bot,\bot) & \textit{otherwise} \end{cases}$$

Furthermore, we write $T \models A$*, whenever* $P_{T,A}$ *is valid, and otherwise, we write* $T \not\models A$ *(which is equivalent to* $P_{T,A}$ *being invalid).*

Note that our definition of $T \models A$ coincides with the standard definition of satisfaction, and $T \not\models A$ coincides with the satisfaction of the dual formula, i.e. corresponds to negation. Next, special intermediate games derived for automata satisfaction on abstracted systems are introduced. In the following, Z is used to describe subsets of the system's state space. More precisely, $Z = \{z : \overline{\mathcal{L}} \to \{+, ?, -\} \mid \infty > |z|\}$ with $|z| = |\{\psi \in \overline{\mathcal{L}} \mid z(\psi) \neq ?\}|$. The set $\{+, ?, -\}$ is ordered by $^{+}\!\diagdown_{?}\!\diagup^{-}$. The elements of Z can be thought of as abstract states obtained through predicate abstraction, which

is made explicit as follows: The derived formula for $z \in Z$, which characterizes the underlying system states, is $\psi_z = ((\bigwedge_{\psi:z(\psi)=+} \psi) \wedge (\bigwedge_{\psi:z(\psi)=-} \neg\psi))$. We say that z is *finer* than z' if $z \sqsupseteq z'$, i.e. $\forall \psi \in \overline{\mathcal{L}} : z(\psi) \geq z'(\psi)$, which ensures that $[\![\psi_z]\!] \subseteq [\![\psi_{z'}]\!]$.

Definition 5. *An* abstract property-game *P w.r.t. Q is $(G, R^{-?}, R^{+?}, Z_{\mathrm{sat}}, Z_{\mathrm{unsat}})$, where*

- *G is an intermediate game such that $C \subseteq Z \times Q$. We write π_Z, respectively π_Q, for the projection from C to its first, respectively second, component.*
- *$R^{-?}, R^{+?} \subseteq C \times C$ are a set of possible must- and a set of possible not-may-transitions such that $R^{-?} \cap R^{-} = \emptyset$ and $R^{+?} \subseteq R^{+}$.*
- *$Z_{\mathrm{sat}}, Z_{\mathrm{unsat}} \subseteq Z$ are disjoint sets of abstract states indicating for which abstract state satisfiability (resp. unsatisfiability) is ensured.*

The set of all its transitions is $R_{\mathrm{all}} = R \cup R^{-} \cup R^{-?} \cup R^{+} \cup R^{+?}$.

The underlying states of an abstract property-game are also called configurations in the sequel. In our algorithm, we are only interested in abstract property-games where Q is the underlying state space of an automaton and where G obeys the following additional notations and constraints, described only informally: Configurations whose Q-component is in Q_0 (Q_1) belong in general to C_0 (resp. C_1), except that they can also be intermediate game states (in $C_{\frac{1}{2}}$). The parity function θ of G is given by $\Theta \circ \pi_Q$, where Θ is the automaton's acceptance condition.

The normal transitions of G consist of two types; those leaving configurations from $C_{\frac{1}{2}}$ are called *focus-transitions* and those leaving configurations from $C_0 \cup C_1$ and having a property of type $\widetilde{\vee}$ or $\widetilde{\wedge}$ are called *junction-transitions*. The corresponding configurations are called *junction configurations*. Junction, as well as $C_{\frac{1}{2}}$, configurations, have at most two outgoing transitions. The must- and may-transitions leave $C_0 \cup C_1$ configurations having type $\Diamond$ or $\Box$, also called *quantifier configurations*.

Focus-transitions are used to partition an abstract state into several configurations: Each of the targets of the outgoing focus-transitions of an abstract game state $c \in C_{\frac{1}{2}}$ describes a part of the set of concrete states described by c. The automaton component does not change along focus-transitions. Junction-transitions imitate the automaton transitions, but the abstract state of the source of a junction transition might be finer than the abstract state of its target (as a result of refinement). Must and may transitions change both the Q-component of the game state according to the automaton transition relation, and the abstract states according to the system. The must-transitions are used to underapproximate the concrete transitions of T, whereas the may-transitions are used as an overapproximation. Namely, a must-transition exists only if *all* the concrete system states represented by the source game state have a corresponding transition to *some* concrete state represented by the target game state. This is called the $\forall\exists$ rule. Every must-transition satisfies it, but possibly not all the transitions that satisfy it are included. On the other hand, a may-transition exists (at least) if *some* concrete state represented by the source state has a corresponding transition to *some* concrete state represented by the target state. This is called the $\exists\exists$ rule. Every transition that satisfies it has to be included as a may-transition, but possibly more are contained.

Since the approximations given by the must and may transitions are not always precise, we use $R^{-?}$ to denote the transitions that are candidates to be *added* as must-transitions, as they might satisfy the $\forall\exists$ rule. Dually, $R^{+?}$ denotes transitions that are

candidates to be *removed* from the set of may-transitions, as they might not satisfy the $\exists\exists$ rule. The initial abstract property-game is the following:

Definition 6. *The* initial abstract property-game $P^I_{T,A}$ *for* T *and* A *is* $((C_0, C_1, \{z_?\} \times Q_{\mathcal{L}}, \{(z_?, q^{\mathrm{i}})\}, R, \emptyset, R^+, \Theta \circ \pi_Q, \omega), R^+, R^+, \{z_?\}, \emptyset)$*, where*

$z_?(\psi) =?$ *for any* $\psi \in \overline{\mathcal{L}}$

$C_0 = (\{z_?\} \times Q_0) \cup \{(z_?[p \mapsto x], q) \mid (x = -, \delta(q) = p)$ *or* $(x = +, \delta(q) = \neg p)\}$

$C_1 = (\{z_?\} \times Q_1) \cup \{(z_?[p \mapsto x], q) \mid (x = +, \delta(q) = p)$ *or* $(x = -, \delta(q) = \neg p)\}$

$R = \{((z_?, q), (z_?, q')) \mid \exists q'' : \delta(q) \in \{q' \widetilde{\wedge} q'', q'' \widetilde{\wedge} q', q' \widetilde{\vee} q'', q'' \widetilde{\vee} q'\}\} \cup$
$\{((z_?, q), (z_?[p \mapsto x], q)) \mid \delta(q) \in \{p, \neg p\}, x \in \{+, -\}\}$

$R^+ = \{((z_?, q), (z_?, q')) \mid \delta(q) \in \{\Diamond q', \Box q'\}\}$

$\omega(z, q) = (\bot, \bot)$ *for* $(z, q) \in C_0 \cup C_1 \cup \{z_?\} \times Q_{\mathcal{L}}$.

Note that the initial abstract property-game does not depend on T. It consists of a single abstract state $z_?$, which abstracts any concrete system-state. The may-transitions overapproximate the concrete transitions by including a transition from $z_?$ to itself. The underapproximation is empty. All the may-transitions are candidates to be added as must-transitions, or alternatively be removed from the set of may-transitions. Here, all the configurations whose property is in Q_0 are C_0-states, and all the configurations whose property is in Q_1 are C_1-states, as in the concrete property-game. The $C_{\frac{1}{2}}$ configurations consist of the combination of $z_?$ with the predicate subproperties. Such a game state is divided according to the predicate, via focus-transitions, into two configurations having the same subproperty (predicate) but whose states are less abstract: one where the predicate is added to $z_?$, and another where its negation is added to $z_?$. The game state where the abstract state agrees with the predicate is a C_1-state, meaning Player 0 wins in it (since it has no outgoing transition for Player 1 to use). The game state that represents disagreement between the abstract state and the predicate is a C_0-state, meaning Player 1 wins in it. This makes the $C_{\frac{1}{2}}$-state, which is controlled by Player 1 in the validity game and by Player 0 in the invalidity game, neither valid nor invalid, indicating that the value of the predicate in $z_?$ is unknown. The set of configurations for which satisfiability is ensured (Z_{sat}) is initialized to $\{z_?\}$, since system states exist, and the set where unsatisfiability is ensured (Z_{unsat}) is initialized as empty. The initial abstract property-game for the property and system given in Fig. 1 is illustrated in Fig. 2 (a). There, the abstract state $z_?$ is denoted by $\emptyset$ since no predicate is set in it.

3 CEGAR via lazy abstraction

Validity values. We start by explaining the 9 different validity-values. A configuration with abstract state z and property q of an abstract property-game can only be valid (resp. invalid) if all the underlying concrete states of z satisfy (resp. falsify) q. Thus, validation in abstract property-games is no longer 2-valued, since it is possible that some underlying concrete states satisfy the formula and some do not. This typically leads to a 3-valued setting, where (z, q) can be neither valid nor invalid. In case unsatisfiable abstract states are allowed, as in our case, it even leads to a 4-valued setting, where (z, q) can be both valid and invalid (if z is unsatisfiable). We use further validity values that help us define improved simplifications of the game structure. Namely, we distinguish between (in)validity and existential-(in)validity: (in)validity ensures that *all* the

underlying concrete states are (in)valid, possibly vacuously. Existential-(in)validity ensures that there *exists* an underlying (in)valid concrete state. Existential-invalidity thus ensures that the configuration is *not* valid, and dually for existential-validity. These possibilities are recorded by ω_1 w.r.t. validity and by ω_2 w.r.t. invalidity. Namely, we use ω_1 to determine if the configuration is valid (tt), existential-invalid (ff) – meaning it is *not* valid, or its validity is unknown ($\bot$). Dually for ω_2.

More precisely, $(\mathrm{tt},\bot)$ stands for valid, $(\bot,\mathrm{tt})$ stands for existential-valid, and $(\mathrm{tt},\mathrm{tt})$ stands for valid-and-satisfiable. Similarly, $(\bot,\mathrm{ff})$ stands for invalid, $(\mathrm{ff},\mathrm{ff})$ stands for invalid-and-satisfiable, and $(\mathrm{ff},\bot)$ stands for existential-invalid. Value $(\mathrm{tt},\mathrm{ff})$ stands for unsatisfiable, $(\mathrm{ff},\mathrm{tt})$ stands for existential-mixed, and $(\bot,\bot)$ stands for unknown.

Example. We illustrate how our algorithm works on a toy example. We describe the underlying abstract property-game, the underlying model checking algorithm, the simplifications of the underlying game structure, and the possible refinement steps (which depend on a heuristic). The algorithm is illustrated by checking the μ-calculus formula presented via an automaton in Fig. 1 (α) at the system depicted in Fig. 1 (β).

Fig. 2 (a) presents the initial abstract property-game, defined in Def. 6 and explained thereafter. The unknown existence of may and must transitions is indicated by the symbol ? on the transitions. The focusing of predicates configurations can be viewed as a degenerate split that takes place in the initial abstract property-game. (In)validity is determined via a parity game algorithm. The configurations where Player 0 has a winning strategy in the validity game are labeled as valid $(\mathrm{tt},\bot)$, whereas the ones where Player 1 has a winning strategy in the invalidity game are labeled as invalid $(\bot,\mathrm{ff})$, as shown in Fig. 2 (b). No further validity-label improvements and redundant transitions/configurations removals are possible in Fig. 2 (b).

Since the initial configuration in Fig. 2 (b) is undetermined, refinement is needed. Thus, a heuristic, determining how to refine the abstraction, is applied. The different possibilities are illustrated by the remainder of the example. Assume that the heuristic determines that the initial configuration, denoted c, needs to be split according to the least precondition of $true$ (the characterizing formula of the abstract state $\emptyset$), denoted $\tilde{p} \equiv \mathrm{pre}(true)$. Then the structure Fig. 2 (c) is obtained, where two initial configurations that correspond to the division of c by $\tilde{p}$ are added, c becomes an intermediate $(C_{\frac{1}{2}})$ configuration, corresponding focus-transitions are added from c to the new configurations, and the outgoing transitions of c are redirected to the new configurations (by doubling them). Note that only c is split. Here we already determine the existence of the must-transition from $(\tilde{p},\Diamond)$ to $(\emptyset,\wedge)$ since $\tilde{p}$ represents the least precondition of $\emptyset$, thus the $\forall\exists$ condition necessarily holds. By analogous arguments, we determine the absence of the may-transition from $(\neg\tilde{p},\Diamond)$ to $(\emptyset,\wedge)$ (this detection mechanism is not yet included in the algorithm in order to increase readability). The validity check determines the upper left configuration to be invalid, yielding Fig. 2 (d).

Next, assume that the heuristic determines that the lower left configuration needs to be checked to determine if it contains the initial concrete state (since two initial configurations currently exist as a result of the previous split). This is indeed the case here. Therefore, the set of initial configurations becomes this singleton set. Moreover, the satisfiability of the corresponding abstract state $\tilde{p}$ is implied, thus Z_{sat} is extended. Fig. 2 (e) is obtained.

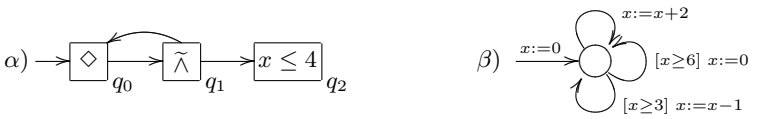

Fig. 1. A μ-calculus formula (α) in terms of automata (see Def. 3), and a system (β). The property of (α) holds if there is an infinite path possible such that always $x \leq 4$ holds. It corresponds to the CTL formula $EG(x \leq 4)$. In (β), the range of x is $\mathbb{N}$, initialized with 0. The actions of the transitions can be executed whenever the guard, depicted in rectangular brackets, is valid.

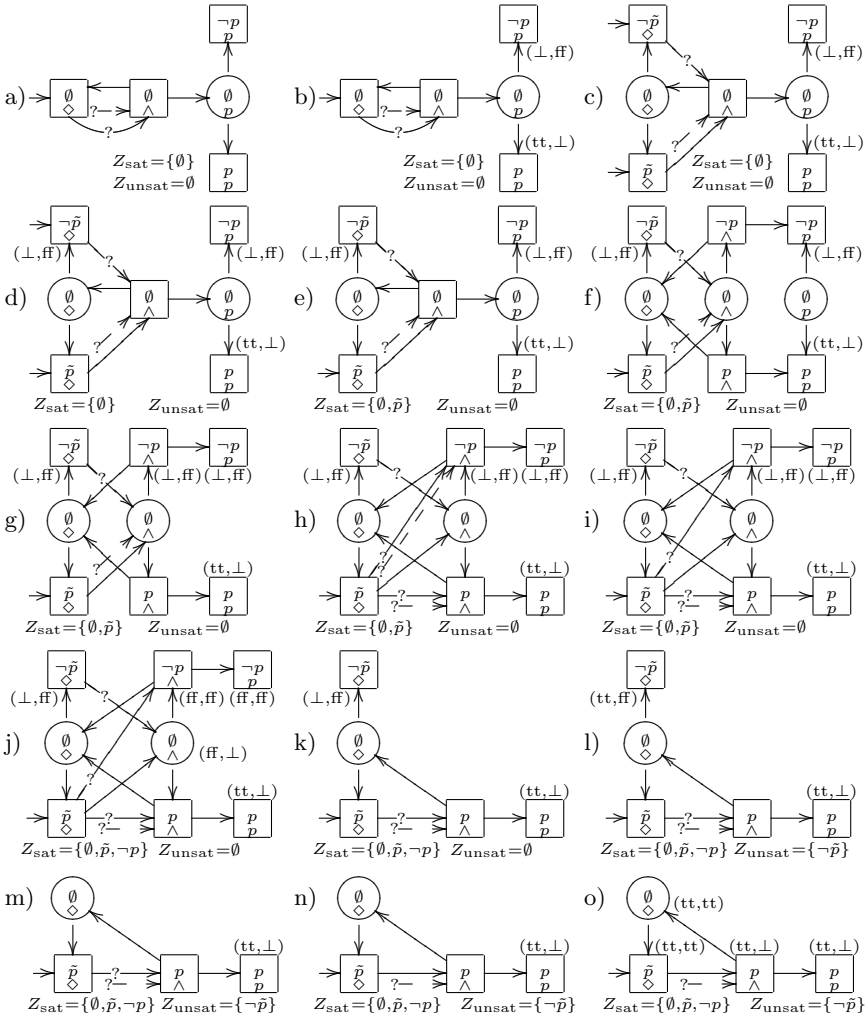

Fig. 2. Example of a property check via lazy abstraction. Here, $p \equiv x \leq 4$ and $\tilde{p} \equiv \mathrm{pre}(true)$. May-transitions are depicted as dashed arrows and must-, as well as focus- and junction-, transitions as solid arrows. May-/must-transition whose existence is unknown contain symbol ?. Intermediate configurations are depicted as circles and the others are depicted as rectangles.

Now, assume the heuristic determines that the "degenerate" split of the intermediate configuration $(\emptyset, p)$ needs to be propagated backwards along the junction-transition t that points to it. Then the source $(\emptyset, \wedge)$ of t, denoted $\mathrm{sor}(t)$, is divided into two new configurations via the predicate p that determined the split of $(\emptyset, p)$. This is done by dividing $\mathrm{sor}(t)$ according to the targets of the focus-transitions that leave $(\emptyset, p)$ (these are the configurations to which $(\emptyset, p)$ was split). As a result, $\mathrm{sor}(t)$ becomes an intermediate configuration, corresponding focus-transitions are added from $\mathrm{sor}(t)$ to the new configurations, t is redirected (by doubling it) such that each of its copies connects one of the new configurations directly to the corresponding target of the focus-transitions leaving $(\emptyset, p)$ that agrees with it on the splitting predicate p (instead of connecting it to $(\emptyset, p)$), and other transitions leaving $\mathrm{sor}(t)$ are redirected (by doubling them) such that their sources become the new configurations. As a result, the original target $(\emptyset, p)$ of t becomes unreachable. Thus Fig. 2 (f) is obtained and after (in)validity determination and removal of the unreachable configuration, Fig. 2 (g) is obtained.

Assume the heuristic yields the (unique) may-transition t that points to the intermediate configuration $(\emptyset, \wedge)$ in order to redirect it to the configurations that resulted from the previous split of $(\emptyset, \wedge)$. Then t is replaced by may-transitions (for which existence is not ensured) pointing to the targets of the focus-transitions leaving $(\emptyset, \wedge)$. Also, possible must-transitions are added to those configurations, but previous must-transitions (to the intermediate configuration $(\emptyset, \wedge)$) are not removed. These previous must-transitions can be considered as hypertransitions. Thus Fig. 2 (h) is obtained and after the removal of irrelevant transitions Fig. 2 (i) is obtained. There, the (not ensured) may transition from $(\tilde{p}, \diamond)$ to $(\neg p, \wedge)$ is removed. This is because the source configuration is controlled by Player 0 and may transitions are used by Player 0 in the invalidity game. However, since the target of the transition is labeled by $(\bot,\mathrm{ff})$, Player 0 will not use this transition in a winning strategy in the invalidity game, since it will make him lose (by reaching a configuration whose ω_2-value is ff). Thus, removing it does not change the outcome.

Assume the heuristic determines that satisfiability of the abstract state encoded by $\neg p$ needs to be checked. The state is satisfiable and therefore Z_{sat} is extended. Furthermore, the validity-value of the two configurations having this abstract state and that are invalid $(\bot,\mathrm{ff})$ is modified to $(\mathrm{ff},\mathrm{ff})$, indicating that, beside the fact that *all* underlying concrete states are invalid, there also *exists* an invalid underlying concrete state. Using this information, the intermediate configuration $(\emptyset, \wedge)$ pointing to one of those configurations via a focus-transition is labeled with a value $(\mathrm{ff},\bot)$ (existential-invalid) indicating that there is an underlying concrete state which is invalid. This is justified by the fact that the intermediate configuration has the same subproperty as the targets of its outgoing focus-transitions and its abstract state represents a superset of their abstract states. Thus Fig. 2 (j) is obtained and after the removal of irrelevant transitions and unreachable configurations Fig. 2 (k) is obtained. Namely, first both of the (possible-)must transitions pointing to the intermediate configuration $(\emptyset, \wedge)$ are removed. This is because the sources of these transitions are controlled by Player 0 and must transitions are used by Player 0 in the validity game, but the ω_1-value of the target $(\emptyset, \wedge)$ of the transitions is ff, which makes Player 0 lose. Therefore, Player 0 will not use these transitions in a winning strategy in the validity game, and removing them does not change the outcome. Similar arguments are responsible for the removal of the possible-must transition

pointing to $(\neg p, \wedge)$. The removal of these transitions makes $(\emptyset, \wedge)$, $(\neg p, \wedge)$ and $(\neg p, p)$ unreachable and they are removed with their transitions.

Assume the heuristic yields the abstract state encoded by $\tilde{p}$ for which satisfiability needs to be checked. The state is unsatisfiable and therefore Z_{unsat} is extended. Furthermore, the upper left configuration having this abstract state is labeled as unsatisfiable $(\mathrm{tt},\mathrm{ff})$. Thus Fig. 2 (l) is obtained and after the removal of irrelevant transitions and unreachable configurations Fig. 2 (m) is obtained. Namely, the focus-transition pointing to the unsatisfiable configuration is removed (along with the unsatisfiable configuration), since it will never be used as a part of a winning strategy: in the validity game its source is controlled by Player 1, yet, the ω_1-value of its target is tt, making Player 1 lose. Analogously, in the invalidity game it makes Player 0 who controls it lose, since the ω_2-value of its target is ff.

Finally, assume the heuristic yields the (unique) possible-must-transition from $(\tilde{p}, \Diamond)$ to $(p, \wedge)$, for which existence needs to be checked. After checking the $\forall\exists$ condition by an unsatisfiability check of $\tilde{p} \wedge \neg\mathrm{pre}(p)$, the must transition is added to the structure. Thus Fig. 2 (n) is obtained. The parity-game algorithm determines all the configurations as valid, after which the validity function is adapted to $(\mathrm{tt},\mathrm{tt})$ in the configurations where the states are known to be satisfiable. This yields Fig. 2 (o), where the calculation terminates, since the property is verified: the single initial configuration is valid (the first component of its validity-value is tt).

Base algorithm. Table 2 presents the verification algorithm PropertyCheck and its used procedure Validity, which determines the (in)valid configurations of a given abstract property-game and adapts the validity function as best as possible. In Line 1, Z_{unsat} is used to determine unsatisfiable states. In Line 2 the validity algorithm is applied and the determined validity is stored in the first component of ω. In Line 3, valid configurations become valid-and-satisfiable if the underlying abstract state is known to be satisfiable, i.e. is in Z_{sat}. In Line 4, a configuration c that points via a chain of focus-transitions to a configuration for which the existence of a concrete state satisfying the corresponding property is known, i.e. where ω_2 is tt, is also updated to have value tt for ω_2. This is because the concrete states described by c are a superset of those described by the targets of its outgoing focus-transitions. Lines 5-7 make the analogous adaptations concerning invalidity determination. As a result of Lines 4 and 7, $C_{\frac{1}{2}}$-configurations may get the "pure existential" values $(\bot,\mathrm{tt}),(\mathrm{ff},\bot),(\mathrm{ff},\mathrm{tt})$, which are later used to simplify the game. No other configurations can get these values.

PropertyCheck starts by constructing the initial abstract property-game obtained from a given automaton. Then it repeatedly applies Validity, makes some simplifications (explained below), and calculates a refinement step until the property is verified or falsified, which is the case if the initial configurations are either all valid or all invalid. Redundant transitions (and configurations) are removed as follows. In Line 3, the outgoing transitions of configurations whose validity value is from $\{(\mathrm{tt},\mathrm{tt}),(\mathrm{ff},\mathrm{ff}),(\mathrm{tt},\mathrm{ff})\}$ are removed. This is reasonable, since any play in one of the games will end at such configurations. The same argument holds for value $(\mathrm{ff},\mathrm{tt})$ that might be given to a $C_{\frac{1}{2}}$ configuration, but here (in)validity is not completely resolved, and therefore the outgoing transitions are not removed, since they might be necessary during refinement. Transitions that will never be chosen in a winning strategy are removed as follows. In

Algorithm PropertyCheck (A : automaton, T : rooted transition system)
Local variables P : an abstract property-game, initialized with $P^I_{T,A}$

1: Validity (P)
2: `while` $(\exists c^{\mathrm{i}} \in C^{\mathrm{i}} : \omega_1(c^{\mathrm{i}}) \neq \mathrm{tt}) \wedge (\exists c^{\mathrm{i}} \in C^{\mathrm{i}} : \omega_2(c^{\mathrm{i}}) \neq \mathrm{ff})$ `do`
3: Remove all transitions in R_{all} leaving a configuration c with $\omega(c) \in \{_{(\mathrm{tt},\mathrm{tt}),(\mathrm{ff},\mathrm{ff}),(\mathrm{tt},\mathrm{ff})}\}$.
4: `If` $c \in C_0 \wedge (\omega_1(c) = \bot \Rightarrow \omega_1(c') = \mathrm{ff}) \wedge (\omega_2(c) = \bot \Rightarrow \omega_2(c') = \mathrm{ff})$ `or` $c \in C_1 \wedge (\omega_1(c) = \bot \Rightarrow \omega_1(c') = \mathrm{tt}) \wedge (\omega_2(c) = \bot \Rightarrow \omega_2(c') = \mathrm{tt})$ `or` $c \in C_{\frac{1}{2}} \wedge \omega(c') = (\mathrm{tt}, \mathrm{ff})$ `then` Remove (c, c') from R.
5: `If` $c \in C_0 \wedge (\omega_2(c) = \bot \Rightarrow \omega_2(c') = \mathrm{ff})$ `or` $c \in C_1 \wedge (\omega_1(c) = \bot \Rightarrow \omega_1(c') = \mathrm{tt})$ `then` Remove (c, c') from $R^+ \cup R^{+?}$.
6: `If` $c \in C_0 \wedge (\omega_1(c) = \bot \Rightarrow \omega_1(c') = \mathrm{ff})$ `or` $c \in C_1 \wedge (\omega_2(c) = \bot \Rightarrow \omega_2(c') = \mathrm{tt})$ `then` Remove (c, c') from $R^- \cup R^{-?}$.
7: Remove from P all the configurations which are unreachable from the initial configurations via R_{all}.
8: Refine (P)
9: Validity (P)
10: `if` $\forall c^{\mathrm{i}} \in C^{\mathrm{i}} : \omega_1(c^{\mathrm{i}}) = \mathrm{tt}$ `then` return(tt) `else` return(ff)

Algorithm Validity (P : an abstract property-game)

1: Set ω to $(\mathrm{tt}, \mathrm{ff})$ on every configuration $c \in C$ where $\pi_Z(c) \in Z_{\mathrm{unsat}}$.
2: Determine the valid configurations via a parity-game algorithm and set ω_1 to tt on those.
3: Set $\omega_2(c)$ to tt on every configuration $c \in C$ where $\omega_1(c) = \mathrm{tt}$ and $\pi_Z(c) \in Z_{\mathrm{sat}}$.
4: Set ω_2 to tt on every configuration from $C_{\frac{1}{2}}$ that points to a configuration where ω_2 is tt.
5: Determine the invalid configurations via a parity-game algorithm and set ω_2 to ff on those.
6: Set $\omega_1(c)$ to ff on every configuration $c \in C$ where $\omega_2(c) = \mathrm{ff}$ and $\pi_Z(c) \in Z_{\mathrm{sat}}$.
7: Set ω_1 to ff on every configuration from $C_{\frac{1}{2}}$ that points to a configuration where ω_1 is ff.

Table 2. A model checking algorithm PropertyCheck for the μ-calculus. Its used procedure for validity determination Validity is presented here and the procedure for the refinement calculation is given in Table 3. Here, $P = ((C_0, C_1, C_{\frac{1}{2}}, C^{\mathrm{i}}, R, R^-, R^+, \theta, \omega), R^{-?}, R^{+?}, Z_{\mathrm{sat}}, Z_{\mathrm{unsat}})$.

Line 4 junction-transitions leaving $\widetilde{\vee}$-configurations c, i.e. $c \in C_0$, are removed whenever each not yet determined component of $\omega(c)$ is ff in the target of the transition. This is justified since Player 0 would lose (in both games) by using such transitions. Analogously for $c \in C_1$. Focus-transitions to unsatisfiable configurations are also removed since they are losing for their controlling player (in both games). In Line 5 (resp. Line 6) may-(resp. must-)transitions are removed in similar conditions as for junction-transitions except it is sufficient to consider only one component of $\omega(c)$, since may-(resp. must-)transitions can only be used by Player 0 in the invalidity game or Player 1 in the validity game (resp. Player 0 in the validity game or Player 1 in the invalidity game). Finally, all unreachable configurations are removed in Line 7.

Refinement algorithm. The pseudo code of the refinement algorithm is presented in Table 3. There, a heuristic is used to determine the refinement. The heuristic may de-

Algorithm Refine (P: a property-game)

A heuristic determines one of the following cases including the determination of the corresponding configuration, transition, etc.:

1: [determine $((z,q),\psi) \in (C_0 \cup C_1) \times \overline{\mathcal{L}}$ with $q \in Q_{\mathrm{qua}} \wedge z(\psi) =? \wedge \omega(z,q) = (\bot,\bot)$]:

% Here, $c = (z,q)$ and $C' = \{(\{z[\psi \mapsto +]\}, q), (\{z[\psi \mapsto -]\}, q)\}$ and $j \in \{1,2\}$ with $q \in Q_j$.

2: $C_j := (C_j \setminus \{c\}) \cup C'$ and $C_{\frac{1}{2}} := C_{\frac{1}{2}} \cup \{c\}$ and $R := R \cup (\{c\} \times C')$

3: $R^{+?} := \{(c', c'') \mid c' \in C' \wedge (c, c'') \in R^+\} \cup R^{+?} \setminus (\{c\} \times C)$

4: $R^{-?} := \{(c', c'') \mid c' \in C' \wedge (c, c'') \in R^+ \setminus R^-\} \cup R^{-?} \setminus (\{c\} \times C)$

5: $R^u := \{(c', c'') \mid c' \in C' \wedge (c, c'') \in R^u\} \cup R^u \setminus (\{c\} \times C)$ with $u \in \{+,-\}$

6: if $c \in C^{\mathrm{i}}$ then $C^{\mathrm{i}} := C' \cup C^{\mathrm{i}} \setminus \{c\}$

7: [determine $(c,c') \in R \cap ((C_0 \cup C_1) \times C_{\frac{1}{2}})$ with $\omega(c) = (\bot,\bot) \wedge \omega(c') \notin \{_{(\mathrm{tt},\mathrm{tt}),(\mathrm{ff},\mathrm{ff}),(\mathrm{tt},\mathrm{ff})}\}$]:

% Here, $c = (z,q)$ and $c' = (z',q')$ and $C'' = \{(\tilde{z}, q) \mid \exists(\tilde{z}', q') \in \{c'\}.R \wedge \tilde{z} = (\tilde{z}' \sqcup z)\}$.

8: $R := R \setminus \{(c,c')\} \cup \{((\tilde{z},q),(\tilde{z}',q')) \mid (\tilde{z}',q') \in \{c'\}.R \wedge \tilde{z} = (\tilde{z}' \sqcup z)\}$

9: if $c \notin C''$ then % otherwise $C'' = \{c\}$

10: $C_j := (C_j \setminus \{c\}) \cup C''$ and $C_{\frac{1}{2}} := C_{\frac{1}{2}} \cup \{c\}$ % Here, $j \in \{0,1\}$ with $q \in Q_j$.

11: $R := (\{c\} \times C'') \cup \{(c'', \tilde{c}) \mid c'' \in C'' \wedge (c, \tilde{c}) \in R\} \cup (R \setminus (\{c\} \times C))$

12: if $c \in C^{\mathrm{i}}$ then $C^{\mathrm{i}} := C'' \cup C^{\mathrm{i}} \setminus \{c\}$

13: [determine $(c,c') \in R^+$ with $c' \in C_{\frac{1}{2}} \wedge \omega(c') \notin \{_{(\mathrm{tt},\mathrm{tt}),(\mathrm{ff},\mathrm{ff}),(\mathrm{tt},\mathrm{ff})}\}$]:

% Here $R' = \{c\} \times (\{c'\}.R)$

14: $R^{-?} := R^{-?} \cup R'$ and $R^u := (R^u \setminus \{(c,c')\}) \cup R'$ with $u \in \{+,+?\}$

15: [determine $(c,c') = ((z,q),(z',q')) \in R^{+?}$]: if Unsatisfiable$(\psi_z \wedge \mathrm{pre}(\psi_{z'}))$

16: then $R^u := R^u \setminus \{(c,c')\}$ with $u \in \{+,+?\}$

17: else $R^{+?} := R^{+?} \setminus \{(c,c')\}$

18: [determine $(c,c') = ((z,q),(z',q')) \in R^{-?}$]: if Unsatisfiable$(\psi_z \wedge \neg\mathrm{pre}(\psi_{z'}))$

19: then $R^- := (R^- \setminus R_{\mathrm{ta}}^{\uparrow(c,c')}) \cup \{(c,c')\}$ and $R^{-?} := R^{-?} \setminus R_{\mathrm{ta}}^{\uparrow(c,c')}$

20: else $R^{-?} := R^{-?} \setminus R_{\mathrm{ta}}^{\downarrow(c,c')}$

21: [determine $(z,q) \in C^{\mathrm{i}}$]: if Satisfiable$(p^{\mathrm{i}} \wedge \psi_z)$

22: then $C^{\mathrm{i}} := \{(z,q)\}$ and $Z_{\mathrm{sat}} := Z_{\mathrm{sat}} \cup \{z\}$

23: [determine $z \in Z$]: if Satisfiable(ψ_z)

24: then $Z_{\mathrm{sat}} := Z_{\mathrm{sat}} \cup \{z\}$

25: else (if Unsatisfiable(ψ_z) then $Z_{\mathrm{unsat}} := Z_{\mathrm{unsat}} \cup \{z\}$)

Table 3. Here, $P = ((C_0, C_1, C_{\frac{1}{2}}, C^{\mathrm{i}}, R, R^-, R^+, \theta, \omega), R^{-?}, R^{+?}, Z_{\mathrm{sat}}, Z_{\mathrm{unsat}})$. Newly introduced configurations map via ω to $(\bot,\bot)$ and via θ as $\Theta \circ \pi_Q$. Satisfiable(ψ) denotes a satisfiability check of ψ (checks if $[\![\psi]\!] \neq \emptyset$) made by a theorem prover, and similarly for Unsatisfiable(ψ). For $(c_s, c_t) \in R_{\mathrm{all}}$ let $R_{\mathrm{ta}}^{\uparrow(c_s,c_t)} = \{(c_s, c_t') \in R_{\mathrm{all}} \mid c_t \in \{c_t'\}.R_{\frac{1}{2}}^*\}$ and $R_{\mathrm{ta}}^{\downarrow(c_s,c_t)} = \{(c_s, c_t') \in R_{\mathrm{all}} \mid c_t' \in \{c_t\}.R_{\frac{1}{2}}^*\}$, where $R_{\frac{1}{2}}^*$ is the transitive closure of the focus transitions $R \cap (C_{\frac{1}{2}} \times C)$.

pend on further arguments other than the abstract property-game, like the system to be checked or the history of refinement. It can also be probabilistic. We describe each possible refinement-scenario along with the way it is handled algorithmically.

The first two scenarios of the refinement correspond to the local split, better focusing, of a quantifier-, resp. junction-, configuration (Line 1, resp. Line 7). A split always originates either in a configuration whose subproperty is a predicate (such configurations are split in the initial abstract property-game already) or in a quantifier-configuration. It later might be propagated to junction-configurations. A local split is performed by turning the configuration into a $C_{\frac{1}{2}}$ configuration, which serves as an auxiliary configuration, and introducing new subconfigurations. Newly introduced configurations map via ω to $(\bot,\bot)$ and via θ as $\Theta \circ \pi_Q$. Thus, during refinement, C_0 or C_1 configurations can become $C_{\frac{1}{2}}$ configurations. However, a C_0 configuration will never become a C_1 configuration and vice versa. Similarly, $C_{\frac{1}{2}}$-configurations never become C_0 or C_1 configurations. In addition, the initial configurations always remain $C_0 \cup C_1$ configurations. Furthermore, as a result of the local splitting, the configurations used in the abstract property-game might overlap. However, we have the property that when considering only $C_0 \cup C_1$ configurations having the same property (i.e. the same second component), then their abstract states have disjoint underlying sets of concrete states. The different calculations for the two splitting scenarios are described in more detail below, followed by the other scenarios of the refinement, which are aimed at updating the components of the abstract property-game and making them more precise (e.g. after a split took place). Note that in previous papers the propagation of a split to junction-configurations, as well as the updates of the other components, were all performed together in each refinement step.

Splitting quantifier-configurations (Line 1). A quantifier-configuration $c = (z, q) \in C_0 \cup C_1$ whose validity is unknown, i.e. $\omega(c) = (\bot,\bot)$, is determined together with a predicate $\psi \in \overline{\mathcal{L}}$ performing the split. The predicate is unused in z, i.e. $z(\psi) = ?$ (otherwise no improvement takes place). Two new configurations where z is set to positive, resp. negative, at ψ are added to the configurations of the corresponding player, and c becomes a $C_{\frac{1}{2}}$ configuration that points via focus-transitions to the newly added configurations (Line 2). The transitions outgoing c are redirected (and doubled) such that they leave the two new configurations (Line 5). The redirected must-transitions remain ensured. The redirected may-transitions, on the other hand, might be overly approximated due to the specialization of the source. Thus, they are added as possible not-may-transitions (Line 3). Furthermore, the may-transitions which are not subsumed by ensured must-transitions are also added as possible must-transitions (Line 4), since the specialization of the source can cause them to fulfill the $\forall\exists$ rule. Finally, if c is an initial configuration, it is replaced by the new ones (Line 6). In Fig. 2, 'splitting quantifier-configuration' is performed from (b) into (c).

Splitting junction-configurations (Line 7). Here, the purpose is to propagate a previous split to the anteceding junction-configurations of the split configuration. Therefore, a transition (c, c') from a junction-configuration whose validity is unknown, i.e. $\omega(c) = (\bot,\bot)$, to a $C_{\frac{1}{2}}$-configuration is determined, i.e. $(c, c') \in R \cap ((C_0 \cup C_1) \times C_{\frac{1}{2}})$. The idea is to split c via the two configurations reachable from c' by focus-transitions, which are the configurations to which c' was split earlier. Thus, c' is determined such that

its validity is not from $\{_{(\mathrm{tt},\mathrm{tt}),(\mathrm{ff},\mathrm{ff}),(\mathrm{tt},\mathrm{ff})}\}$, since otherwise either all the underlying concrete states of c' are in agreement or none exist, and in both cases no improvement will result from splitting c according to the split of c'.

Then c is (possibly) split as follows. For each of the two configurations $\tilde{c}'$, reachable via a focus-transition from c', we consider in C'' a configuration that corresponds to the least upper bound $z \sqcup \tilde{z}'$ of the abstract state of c and the abstract state of $\tilde{c}'$, if it exists. The concrete states encoded by $z \sqcup \tilde{z}'$ correspond to the intersection of the underlying concrete states of z and $\tilde{z}'$. Note that while $\tilde{z}'$ is finer than the abstract state z' of c', it is not guaranteed that $\tilde{z}'$ is finer than z, since z could become finer than z' by previous splits based on other outgoing junction-transitions of c. In particular, z and $\tilde{z}'$ might give contradictory values ($+$ vs. $-$) to some predicate (meaning they represent disjoint sets of concrete states), in which case $z \sqcup \tilde{z}'$ does not exist. Still, at least for one focus-transition target such an upper bound exists, since by an invariant z is finer than z'. Moreover, exactly one additional predicate $\psi \in \overline{\mathcal{L}}$ is set (either to $+$ or to $-$) in $\tilde{z}'$ compared to z' (along the focus-transition). ψ is the predicate that c' was split by. This means that z is finer than $\tilde{z}'$ w.r.t. all predicates, except for possibly ψ. Now, if ψ is not set in z, then ψ does not introduce contradictions as well, thus (1) upper bounds exist for both of the focus-transitions targets. Otherwise (ψ is already set in z as a result of a previous split), then (2) the least upper bound exists (only) for the one focus-transition target in which ψ is set the same as in z.

In case (2), the only existing least upper bound is equal to z, since in this case z is already finer than $\tilde{z}'$ ($\tilde{z}' \sqsubseteq z$), i.e., it was already split by ψ (and therefore the abstract state of the other focus-transition target is disjoint from z). Thus, $C'' = \{c\}$. In this case, c is not split but the transition (c, c') is redirected to point directly to the configuration $\tilde{c}'$ for which $\tilde{z}' \sqsubseteq z$ (Line 8).

In case (1), the least upper bounds are $z[\psi \mapsto +]$ and $z[\psi \mapsto -]$, each of which represents the intersection of the underlying states of z with the states satisfying ψ or $\neg\psi$, resp., meaning z is split by ψ. In this case, identified by the condition $c \notin C''$ in Line 9, c is split into the two new configurations collected in C''. These are added to the configurations of the corresponding player (Line 10) and c becomes a $C_{\frac{1}{2}}$ configuration (Line 10) pointing via focus-transitions to the new configurations (Line 11). Furthermore, the outgoing transitions of c are redirected to the new configurations as follows. First, instead of the transition (c, c'), each of the new configurations points directly to the focus-transition target $\tilde{c}'$ that "created" it, i.e., whose least upper bound (intersection) w.r.t. z it represents (Line 8). Additionally, the outgoing transitions of c pointing to a target different than c' are redirected (by doubling them) to leave the new configurations (Line 11 combined with Line 8). Finally, if c is an initial configuration, it is replaced by the new ones (Line 12). In Fig. 2, 'splitting junction-configuration' (based on case (1)) takes place from (e) into (f).

Focusing may-transitions (Line 13). Here, the purpose is to propagate a previous split to the incoming may-transitions of the split configuration. Therefore, a may-transition $(c, c') \in R^+$ with $c' \in C_{\frac{1}{2}}$ is determined. Such a transition models a hypertransition. It is redirected (by doubling it) such that it points directly to the focus-transition targets of c', i.e. to the configurations to which c' was 'split' earlier. The determined transition is such that $\omega(c') \notin \{_{(\mathrm{tt},\mathrm{tt}),(\mathrm{ff},\mathrm{ff}),(\mathrm{tt},\mathrm{ff})}\}$, ensuring that these focus-transition targets

were not removed during simplification. The new may-transitions also become possible not-may-transitions, since they might be overly approximated. Furthermore, they are also added as possible must-transitions. Note that unlike the removal of may-transitions pointing to c', (possible) must-transitions that point to c' (if exist) remain intact, since hypertransitions are needed in the case of must transitions to increase expressiveness. In Fig. 2, 'focusing may-transition' is performed from (g) into (h).

Ascertaining may-transitions (Line 15). A possible not-may-transition $((z,q),(z',q')) \in R^{+?}$ is determined and it is checked if it is overly approximated. This is done by a theorem prover call of $\mathsf{Unsatisfiable}(\psi_z \wedge \mathrm{pre}(\psi_{z'}))$. If this call is successful, meaning the $\exists\exists$ condition does not hold w.r.t. ψ_z and $\psi_{z'}$, the transition is removed as a may-transition. Otherwise, it is only removed from $R^{+?}$. Note that here and in the next scenario an unsatisfiability call is made instead of a satisfiability call in order to remain sound if incomplete satisfiability checks are applied.

Ascertaining must-transitions (Line 18). A possible must-transition $((z,q),(z',q')) \in R^{-?}$ is determined and it is checked if it is a real must transition. This is done by a theorem prover call of $\mathsf{Unsatisfiable}(\psi_z \wedge \neg \mathrm{pre}(\psi_{z'}))$. If this call is successful, meaning the $\forall\exists$ condition holds w.r.t. ψ_z and $\psi_{z'}$, the transition is added as a real must-transition and all (possible) must-transitions that have the same source but a less precise target are removed, since their existence does not increase precision. Otherwise, the transition and all possible must-transitions that have the same source but a more precise target are removed (they cannot become real must-transitions). The less (resp. more) precise targets are given by the configurations that are backwards (resp. forwards) reachable from (z',q') via focus-transitions. This is justified by the invariant that the abstract state of the target of a focus-transition is always finer, i.e. more precise, than the abstract state of its source. In Fig. 2, 'ascertaining must-transition' is performed from (m) into (n).

Ascertaining initial configuration (Line 21). Splitting of configurations might result in multiple initial configurations. However, recall that initial configurations are always in $C_0 \cup C_1$ and thus their abstract states are disjoint. This ensures that only one of them abstracts the concrete initial configuration, and the rest are merely overly approximated. Thus an initial configuration $(z,q) \in C^{\mathrm{i}}$ is determined and it is checked if it contains the concrete initial state. This is done by a theorem prover call $\mathsf{Satisfiable}(p^{\mathrm{i}} \wedge \psi_z)$. If successful, C^{i} becomes $\{(z,q)\}$ and z is added to Z_{sat} (since its satisfiability is ensured). In Fig. 2, 'ascertaining initial configuration' is performed from (d) into (e).

Checking satisfiability of abstract states (Line 23). An abstract state $z \in Z$ is determined and its (un)satisfiability is checked. This is done by a theorem prover call $\mathsf{Satisfiable}(\psi_z)$, resp. $\mathsf{Unsatisfiable}(\psi_z)$. If the call is successful, z is added to Z_{sat}, resp. to Z_{unsat}. Both theorem prover calls are necessary for soundness if incomplete satisfiability checks are applied. In Fig. 2, 'checking satisfiability of abstract states' is performed from (i) into (j) and from (k) into (l).

Properties of the algorithm. Satisfiability checks are said to be *sound* if $\mathsf{Satisfiable}(\psi)$ implies that ψ is satisfiable, i.e. $[\![\psi]\!] \neq \emptyset$, and if $\mathsf{Unsatisfiable}(\psi)$ implies that ψ is not satisfiable, i.e. $[\![\psi]\!] = \emptyset$. Satisfiability checks are *complete* if the reverse implications of the above constraints hold. Let $\mathcal{O} = \{P.3, ..., P.7\} \cup \{V.1, ..., V.7\}$ denote the execution lines of PropertyCheck, resp. of Validity, in which simplifications of the game structure as well as (in)validity determinations are made. In the following, PropertyCheck also

denotes a more liberal version of it where the lines from $\mathcal{O}$ are not always applied after every refinement step as long as (in)validity determinations are applied infinitely often (more precisely, the bundle of the three Lines V.1,V.2,V,5 are infinitely often calculated after a refinement step). This means that more than one refinement step is calculated at once. We have that our algorithm is correct and no validity information is lost during an execution (even if the more liberal version is used):

Theorem 1 (Soundness). *Let satisfiability checks be sound. If* $\mathsf{PropertyCheck}(A, T)$ *based on any heuristic returns* tt *(resp.* ff*), then* $T \models A$ *(resp.* $T \not\models A$*) holds.*

Theorem 2 (Incremental). *Suppose satisfiability checks are sound,* P *is a property-game obtained during the execution of* $\mathsf{PropertyCheck}(A, T)$*, and* P *is valid (resp. invalid) in* $c \in C$*. Then the execution of any line of* PropertyCheck *or* Validity *yields a property-game that is valid (resp. invalid) in* c *or that does not contain* c *anymore.*

The algorithm is relatively complete for least fixpoint free formulas (and is often also successful for formulas containing least fixpoints). Note that this statement is not implied by the relative completeness of generalized Kripke modal transition systems, since not all hypertransitions are calculated.

Theorem 3 (Relative completeness). *Suppose satisfiability checks are sound and complete and* $\mathcal{L}$ *can describe every subset of* S*. If the acceptance function of* A *maps always to zero (i.e. a least fixpoint free* μ*-calculus formula is encoded) and* $T \models A$*, then any heuristic applied for the first, say* n*, refinement determination steps, can be extended to a (not necessarily computable) one such that* $\mathsf{PropertyCheck}(A, T)$ *returns* tt*.*

Theorem 3 does not hold if we restrict to computable refinement heuristics, since otherwise the halting problem would be decidable. Furthermore, Theorem 3 does not hold for automata with arbitrary acceptance function, since the underlying class of abstract models is not expressive enough. To handle arbitrary functions, fairness constraints, as in [6, 8], are needed.

Remark 2. Previous CEGAR-algorithms, including ours [10], usually need less refinement steps than our new algorithm, since it has a very fundamental laziness. Nevertheless, our algorithm can mimic the refinement steps of the other algorithms without increasing its computation time by calculating all refinements made by the other algorithms in a single step before calling Validity. We expect our algorithm to be in general faster than the others, since we can use improved heuristics that avoid the expensive cost of refinement-calculations by restricting to the relevant calculations.

4 Conclusion

We presented a new CEGAR-based algorithm for μ-calculus verification, which is based on the lazy abstraction technique. We obtained the high level of laziness by developing a new philosophy of a refinement step, namely *state focusing*: The to be split configuration is not removed and is, e.g., used to model hypertransitions. Our algorithm avoids state explosion and, at the same time, remains complete for least fixpoint free formulas. The heuristics presented in [10] can be straightforwardly adapted to our setting. Determination of heuristics that better support the finer lazy abstraction approach of our new algorithm, and a prototype implementation, are topics for future work.

References

1. T. Ball and O. Kupferman. An abstraction-refinement framework for multi-agent systems. In *LICS*, pages 379–388, 2006.
2. T. Ball, O. Kupferman, and M. Sagiv. Leaping loops in the presence of abstraction. In *CAV*, volume 4590 of *LNCS*, pages 491–503, 2007.
3. T. Ball, O. Kupferman, and G. Yorsh. Abstraction for falsification. In *CAV*, volume 3576 of *LNCS*, pages 67–81, 2005.
4. T. Ball, A. Podelski, and S. K. Rajamani. Boolean and cartesian abstraction for model checking C programs. In *TACAS*, volume 2031 of *LNCS*, pages 268–283, 2001.
5. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.
6. D. Dams and K. S. Namjoshi. The existence of finite abstractions for branching time model checking. In *LICS*, pages 335–344, 2004.
7. H. Fecher and M. Huth. Model checking for action abstraction. In *VMCAI*, volume 4905 of *LNCS*, pages 112-126, 2008.
8. H. Fecher and M. Huth. Ranked predicate abstraction for branching time: Complete, incremental, and precise. In *ATVA*, volume 4218 of *LNCS*, pages 322–336, 2006.
9. H. Fecher, M. Huth, H. Schmidt, and J. Schönborn. Refinement sensitive formal semantics of state machines with persistent choice. In *AVoCS*, 2007. will appear in ENiTCS.
10. H. Fecher and S. Shoham. Local abstraction-refinement for the mu-calculus. In *SPIN*, volume 4595 of *LNCS*, pages 4–23, 2007.
11. P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In *CONCUR*, volume 2154 of *LNCS*, pages 426–440, 2001.
12. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *CAV*, volume 1254 of *LNCS*, pages 72–83, 1997.
13. O. Grumberg, M. Lange, M. Leucker, and S. Shoham. When not losing is better than winning: Abstraction and refinement for the full μ-calculus. *Information and Compuatation*, 205(8):1130-1148, 2007.
14. O. Grumberg, M. Lange, M. Leucker, and S. Shoham. *Don't know* in the μ-calculus. In *VMCAI*, volume 3385 of *LNCS*, pages 233–249, 2005.
15. A. Gurfinkel and M. Chechik. Why waste a perfectly good abstraction?. In *TACAS*, volume 3920 of *LNCS*, pages 212–226, 2006.
16. T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *POPL*, pages 58–70, 2002.
17. M. Huth, R. Jagadeesan, and D. A. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *ESOP*, volume 2028 of *LNCS*, pages 155–169, 2001.
18. M. Jurdzinski. Deciding the winner in parity games is in UP $\cap$ co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998.
19. D. Kozen. Results on the propositional μ-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.
20. K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, pages 203–210, 1988.
21. K. G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *LICS*, pages 108–117, 1990.
22. K. S. Namjoshi. Abstraction for branching time properties. In *CAV*, volume 2725 of *LNCS*, pages 288–300, 2003.
23. A. Pardo and G. D. Hachtel. Incremental CTL model checking using BDD subsetting. In *DAC*, pages 457–462, 1998.
24. S. Shoham and O. Grumberg. Monotonic abstraction-refinement for CTL. In *TACAS*, volume 2988 of *LNCS*, pages 546–560, 2004.
25. S. Shoham and O. Grumberg. 3-Valued Abstraction: More Precision at Less Cost. In *LICS*, pages 399–410, 2006.