An Abstract Interpretation-Based Refinement Algorithm for Strong Preservation

Francesco Ranzato and Francesco Tapparo

Dipartimento di Matematica Pura ed Applicata,
Università di Padova, Italy

Abstract. The Paige and Tarjan algorithm (PT) for computing the coarsest refinement of a state partition which is a bisimulation on some Kripke structure is well known. It is also well known in abstract model checking that bisimulation is equivalent to strong preservation of CTL and in particular of Hennessy-Milner logic. Building on these facts, we analyze the basic steps of the PT algorithm from an abstract interpretation perspective, which allows us to reason on strong preservation in the context of generic inductively defined (temporal) languages and of abstract models specified by abstract interpretation. This leads us to design a generalized Paige-Tarjan algorithm, called GPT, for computing the minimal refinement of an abstract interpretation-based model that strongly preserves some given language. It turns out that PT can be obtained by instantiating GPT to the domain of state partitions for the case of strong preservation of Hennessy-Milner logic. We provide a number of examples showing that GPT is of general use. We show how two well-known efficient algorithms for computing simulation and stuttering equivalence can be viewed as simple instances of GPT. Moreover, we instantiate GPT in order to design a $O(|\mathit{Transitions}||\mathit{States}|)$-time algorithm for computing the coarsest refinement of a given partition that strongly preserves the language generated by the reachability operator $\mathbf{EF}$.

1 Introduction

Motivations. The Paige and Tarjan [15] algorithm — in the paper denoted by PT — for efficiently computing the coarsest refinement of a given partition which is *stable* for a given state transition relation is well known. Its importance stems from the fact that PT actually computes *bisimulation equivalences*, because a partition P of a state space Σ is stable for a transition relation $R \subseteq \Sigma \times \Sigma$ if and only if P is a bisimulation equivalence on the transition system $\langle \Sigma, R \rangle$. In particular, PT is widely used in model checking for reducing the state space of a Kripke structure $\mathcal{K}$ because it turns out that the quotient of $\mathcal{K}$ w.r.t. bisimulation equivalence *strongly preserves* branching-time temporal languages like CTL and CTL* [2, 3]. Paige and Tarjan first provide the basic $O(|R||\Sigma|)$-time PT algorithm and then exploit a computational logarithmic improvement in order to design a $O(|R| \log |\Sigma|)$-time algorithm, which is usually referred to as Paige-Tarjan algorithm. It is important to remark that the logarithmic Paige-Tarjan algorithm is derived as a computational refinement of PT that does not affect the correctness of the procedure which is instead proved for the PT algorithm. As shown in [16], it turns out that state partitions can be viewed as *domains in abstract interpretation* and strong preservation can be cast as *completeness in abstract interpretation*. Thus, our first aim was to understand, from an abstract interpretation perspective, why PT is a correct procedure for computing strongly preserving partitions.

The* PT *Algorithm. Let us recall how PT works. Let $\mathrm{pre}_R = \lambda X.\{s \in \Sigma \mid \exists x \in X.\, s \xrightarrow{R} x\}$ denote the usual predecessor transformer on $\wp(\Sigma)$. A partition $P \in \mathrm{Part}(\Sigma)$ is PT stable for R when for any block $B \in P$, if $B' \in P$ then either $B \subseteq \mathrm{pre}_R(B')$ or $B \cap \mathrm{pre}_R(B') = \varnothing$. For a given subset $S \subseteq \Sigma$, we denote by $\mathrm{PTsplit}(S, P)$ the partition obtained from P by replacing each block $B \in P$ with the blocks $B \cap \mathrm{pre}_R(S)$ and $B \smallsetminus \mathrm{pre}_R(S)$, where we also allow no splitting, namely that $\mathrm{PTsplit}(S, P) = P$. When $P \neq \mathrm{PTsplit}(S, P)$ the subset S is called a *splitter* for P. $\mathrm{Splitters}(P)$ denotes the set of splitters of P, while $\mathrm{PTrefiners}(P) \stackrel{\text{def}}{=} \{S \in \mathrm{Splitters}(P) \mid \exists \{B_i\} \subseteq P.\ S = \cup_i B_i\}$. Then, the PT algorithm goes as follows.

```
while ($P$ is not PT stable) do
  choose $S \in \mathrm{PTrefiners}(P)$;
  $P := \mathrm{PTsplit}(S, P)$;
endwhile                                   PT
```

***An Abstract Interpretation Perspective of* PT.** Our work originated from a number of observations on the above PT algorithm. Firstly, we may view the output $\mathrm{PT}(P)$ as the coarsest refinement of a partition P that strongly preserves CTL. For standard abstract models which are partitions, it is known that strong preservation of CTL is equivalent to strong preservation of (finitary) Hennessy-Milner logic HML [12], i.e., the language generated by the grammar: $\varphi ::= p \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathrm{EX}\varphi$, where p ranges over atomic propositions in AP such that $\{[\![p]\!] \subseteq \Sigma \mid p \in AP\} = P$ and the semantic interpretation of EX is $\mathrm{pre}_R : \wp(\Sigma) \to \wp(\Sigma)$. Thus, we observe that $\mathrm{PT}(P)$ indeed computes the coarsest partition P_{HML} that refines P and strongly preserves HML. Moreover, the partition P_{HML} corresponds to the state equivalence $\equiv_{\mathrm{HML}}$ induced by the semantics of HML: $s \equiv_{\mathrm{HML}} s'$ iff $\forall \varphi \in \mathrm{HML}.\, s \in [\![\varphi]\!] \Leftrightarrow s' \in [\![\varphi]\!]$. Hence, we also observe that P_{HML} is an abstraction of the state semantics of HML on the domain $\mathrm{Part}(\Sigma)$ of partitions of Σ. Thus, our starting point was that PT can be viewed as an algorithm for computing the most abstract object on the particular domain $\mathrm{Part}(\Sigma)$ that strongly preserves the particular language HML. We made this view precise within Cousot and Cousot's abstract interpretation framework [4, 5].

We introduced in [16] an abstract interpretation-based framework for reasoning on strong preservation of abstract models w.r.t. generic inductively defined languages. We showed that the lattice $\mathrm{Part}(\Sigma)$ of partitions of the state space Σ can be viewed as an abstraction of the lattice $\mathrm{Abs}(\wp(\Sigma))$ of abstract interpretations of $\wp(\Sigma)$. Thus, a partition $P \in \mathrm{Part}(\Sigma)$ is here viewed as a particular abstract domain $\gamma(P) \in \mathrm{Abs}(\wp(\Sigma))$. This leads to a precise correspondence between *forward complete* abstract interpretations and strongly preserving abstract models. Let us recall that completeness in abstract interpretation [4, 5, 10] encodes an ideal situation where no loss of precision occurs by approximating concrete computations on the abstract domain. The problem of minimally refining an abstract model in order to get strong preservation of some language $\mathcal{L}$ can be cast as the problem of making an abstract interpretation $\mathcal{A}$ forward complete for the semantic operators of $\mathcal{L}$ through a minimal refinement of the abstract domain of $\mathcal{A}$. It turns out that this latter completeness problem always admits a fixpoint solution.

Hence, in our abstract interpretation framework, it turns out that for any $P \in \mathrm{Part}(\Sigma)$, the output $\mathrm{PT}(P)$ is the partition abstraction in $\mathrm{Part}(\Sigma)$ of the minimal refinement of $\gamma(P) \in \mathrm{Abs}(\wp(\Sigma))$ which is complete for the set F of semantic operators of the language HML, where $F_{\mathrm{HML}} = \{\cap, \complement, \mathrm{pre}_R\}$ is the set of operators on $\wp(\Sigma)$ of HML. In particular, it turns out that a partition P is PT stable iff $\gamma(P)$ is complete for the operators in F_{HML}. Also, the following observation is crucial in our approach. The splitting operation $\mathrm{PTsplit}(S, P)$ can be viewed as the *best correct approximation* on $\mathrm{Part}(\Sigma)$ of a refinement operation $\mathrm{refine}_f(S, \cdot) : \mathrm{Abs}(\wp(\Sigma)) \to \mathrm{Abs}(\wp(\Sigma))$ on abstract domains: given an operator $f : \wp(\Sigma) \to \wp(\Sigma)$, $\mathrm{refine}_f(S, A)$ refines an abstract domain A through a "f-refiner" $S \in A$ to the most abstract domain containing both A and $f(S)$. In particular, P results to be PT stable iff the abstract domain $\gamma(P)$ cannot be refined w.r.t. the function pre_R. Thus, if $\mathrm{refine}_f^{\mathrm{Part}}$ denotes the best correct approximation in $\mathrm{Part}(\Sigma)$ of refine_f then the PT algorithm can be formulated as follows.

```
while the set of $\mathrm{pre}_R$-refiners of $P \neq \varnothing$ do
  choose some $\mathrm{pre}_R$-refiner $S \in \gamma(P)$;
  $P := \mathrm{refine}_{\mathrm{pre}_R}^{\mathrm{Part}}(S, P)$;
endwhile
```

Main Results. This abstract interpretation-based view of PT leads us to generalize PT to: (1) a generic domain $\mathcal{A}$ of abstract models generalizing the domain of state partitions $\mathrm{Part}(\Sigma)$ and (2) a generic set F of operators on $\wp(\Sigma)$ providing the semantics of some language $\mathcal{L}_F$ and generalizing the set F_{HML} of operators of HML. We design a generalized Paige-Tarjan refinement algorithm, called GPT, which, for any abstract model $A \in \mathcal{A}$, is able to compute the most abstract refinement of A in $\mathcal{A}$ which is strongly preserving for the language $\mathcal{L}_F$. The correctness of GPT is guaranteed by some completeness conditions on $\mathcal{A}$ and F. We provide a number of applications showing that GPT is an algorithmic scheme of general use. We prove that two well-known algorithms computing *simulation* and *stuttering* equivalence can be obtained as simple instances of GPT. First, we show that the algorithm by Henzinger et al. [13] that computes simulation equivalence in $O(|R||\Sigma|)$-time (as far as time-complexity is concerned, this is the best available algorithm) corresponds to the instance of GPT where the set of operators is $F = \{\cap, \mathrm{pre}_R\}$ and the abstract domain $\mathcal{A}$ is the lattice of disjunctive (i.e. precise for least upper bounds [5]) abstract domains of $\wp(\Sigma)$. We obtain this as a consequence of the fact that simulation equivalence corresponds to strong preservation of the language $\varphi ::= p \mid \varphi_1 \wedge \varphi_2 \mid \mathrm{EX}\varphi$. Second, we show that GPT can be instantiated in order to get the Groote-Vaandrager algorithm [11] that computes divergence blind stuttering equivalence in $O(|R||\Sigma|)$-time (again, this is the best known time bound). Let us recall that the Groote-Vaandrager algorithm can be also used for computing branching bisimulation equivalence, which is the state equivalence induced by CTL*-X [2, 7, 11]. In this case, the set of operators is $F = \{\cap, \complement, \mathbf{EU}\}$, where $\mathbf{EU}$ is the standard semantic interpretation of the existential until, while $\mathcal{A}$ is the domain of partitions $\mathrm{Part}(\Sigma)$. Moreover, we instantiate GPT in order to design a new partition refinement algorithm for the language inductively generated by the reachability operator $\mathbf{EF}$ and propositional logic, namely with $F = \{\cap, \complement, \mathbf{EF}\}$. In this case, we describe a simple implementation for this instance of GPT that leads to a $O(|R||\Sigma|)$-time algorithm.

2 Basic Notions

Notation. Let X be any set. $\mathrm{Fun}(X)$ denotes the set of all the functions $f : X^n \to X$, where $\mathrm{ar}(f) = n > 0$ is the the arity of f. For a set $S \in \wp(\wp(X))$, we write the sets in S in a compact form like $\{1, 12, 123\} \in \wp(\wp(\{1, 2, 3\}))$. We denote by $\complement$ the complement operator w.r.t. some universe set. A function $f : C \to C$ on a complete lattice C is additive when f preserves least upper bounds. We denote by $\mathrm{Part}(X)$ the set of partitions on X. $\mathrm{Part}(X)$ is endowed with the following standard partial order $\preccurlyeq$: given $P_1, P_2 \in \mathrm{Part}(X)$, $P_1 \preccurlyeq P_2$, i.e. P_2 is coarser than P_1 (or P_1 refines P_2) iff $\forall B \in P_1.\exists B' \in P_2.B \subseteq B'$. It turns out that $\langle \mathrm{Part}(X), \preccurlyeq, \curlywedge, \curlyvee, \{X\}, \{\{x\}\}_{x\in X}\rangle$ is a complete lattice. We consider transition systems (Σ, R) where the relation $R \subseteq \Sigma \times \Sigma$ (also denoted by $\xrightarrow{R}$) is total. A Kripke structure (Σ, R, AP, ℓ) consists of a transition system (Σ, R) together with a set AP of atomic propositions and a labelling function $\ell : \Sigma \to \wp(AP)$. For any $s \in \Sigma$, $[s]_\ell \overset{\text{def}}{=} \{s' \in \Sigma \mid \ell(s) = \ell(s')\}$. Also, $P_\ell \overset{\text{def}}{=} \{[s]_\ell \mid s \in \Sigma\} \in \mathrm{Part}(\Sigma)$. A transition relation $R \subseteq \Sigma \times \Sigma$ defines the usual pre/post transformers on $\wp(\Sigma)$: pre_R, post_R, $\widetilde{\mathrm{pre}}_R$, $\widetilde{\mathrm{post}}_R$. When clear from the context, subscripts R are sometimes omitted.

Abstract Interpretation and Completeness. In standard abstract interpretation, abstract domains can be equivalently specified either by Galois connections/insertions (GCs/GIs) or by (upper) closure operators (uco's) [5]. Closure operators have the advantage of being independent from the representation of domain's objects and are therefore appropriate for reasoning on abstract domains independently from their representation. We will denote by (α, C, A, γ) a GC/GI of the abstract domain A into the concrete domain C through the abstraction and concretization maps $\alpha : C \to A$ and $\gamma : A \to C$. Recall that $\mu : C \to C$ is a uco when μ is monotone, idempotent and extensive (i.e., $x \leq \mu(x)$). If μ is reductive (i.e., $\mu(x) \leq x$) instead of extensive then μ is a lower closure operator (lco), namely a uco on the dual lattice $C_{\geq}$. It is known that the set $\mathrm{uco}(C)$ of uco's on C, endowed with the pointwise ordering $\sqsubseteq$, gives rise to the complete lattice $\langle \mathrm{uco}(C), \sqsubseteq, \sqcup, \sqcap, \lambda x.\top_C, id\rangle$. We have that $\mu \sqsubseteq \rho$ iff $\rho(C) \subseteq \mu(C)$; in this case, we say that μ is a refinement of ρ. Also, $\langle \mathrm{lco}(C), \sqsubseteq\rangle$ denotes the complete lattice of lower closure operators on C. It turns out that $\mathrm{uco}(C)$ and $\mathrm{lco}(C)$ are dual isomorphic, namely $\mathrm{uco}(C)_{\sqsupseteq}$ and $\mathrm{lco}(C)_{\sqsubseteq}$ are isomorphic. Hence, notions and results concerning uco's can be stated dually for lco's. Each closure is uniquely determined by the set of its fixpoints, which is also its image. Also, a subset $X \subseteq C$ is the set of fixpoints of some uco on C iff X is meet-closed, i.e. $X = \mathcal{M}(X) \overset{\text{def}}{=} \{\wedge Y \mid Y \subseteq X\}$ (where $\top_C = \wedge_C \varnothing \in \mathcal{M}(X)$). Often, we will identify closures with their sets of fixpoints because this does not give rise to ambiguity. In view of the above equivalence, throughout the paper $\langle \mathrm{uco}(C), \sqsubseteq\rangle$ will play the role of the (complete) lattice of abstract domains of the concrete domain C [4, 5]. The ordering on $\mathrm{uco}(C)$ corresponds to the standard order that compares abstract domains with regard to their precision: A_1 is more precise than A_2 (or A_2 is more abstract than A_1) iff $A_1 \sqsubseteq A_2$ in $\mathrm{uco}(C)$. Let (α, C, A, γ) be a GI, $f : C \to C$ be some concrete semantic function — for simplicity, we consider here 1-ary functions — and $f^\sharp : A \to A$ be a corresponding abstract function. Then, $\langle A, f^\sharp\rangle$ is a sound abstract interpretation when $\alpha \circ f \sqsubseteq f^\sharp \circ \alpha$. The abstract function $f^A \overset{\text{def}}{=} \alpha \circ f \circ \gamma : A \to A$ is called the best correct approximation of f in A. Completeness in abstract interpretation corresponds to require the following strengthening of soundness: $\alpha \circ f = f^\sharp \circ \alpha$. This is called *backward* completeness because a dual *forward* completeness may be considered. The soundness equation $\alpha \circ f \sqsubseteq f^\sharp \circ \alpha$ is equivalent to $f \circ \gamma \sqsubseteq \gamma \circ f^\sharp$, so that forward completeness for $f^\sharp$ corresponds to strengthen soundness by requiring: $f \circ \gamma = \gamma \circ f^\sharp$. Giacobazzi et al. [10] observed that both backward and forward completeness uniquely depend upon the abstraction map, namely they are abstract domain properties. These domain properties can be formulated through uco's as follows: an abstract domain $\mu \in \mathrm{uco}(C)$ is backward complete for f iff $\mu \circ f = \mu \circ f \circ \mu$ holds, while μ is forward complete for f iff $f \circ \mu = \mu \circ f \circ \mu$.

Shells. Refinements of abstract domains have been much studied in abstract interpretation [4, 5] and led to the notion of shell of an abstract domain [10]. Given a generic poset $P_\leq$ of semantic objects — where $x \leq y$ intuitively means that x is a "refinement" of y, i.e. x is more precise than y — and a property $\mathcal{P} \subseteq P$ of these objects, the generic notion of *shell* goes as follows: the $\mathcal{P}$-shell of an object $x \in P$ is defined to be an object $s_x \in P$ such that: (i) s_x satisties the property $\mathcal{P}$, (ii) s_x is a refinement of x, and (iii) s_x is the greatest among the objects satisfying (i) and (ii). Note that if a $\mathcal{P}$-shell exists then it is unique. We will be particularly interested in shells of abstract domains and partitions. Given a state space Σ and a partition property $\mathcal{P} \subseteq \mathrm{Part}(\Sigma)$, the $\mathcal{P}$-shell of $P \in \mathrm{Part}(\Sigma)$ is the coarsest refinement of P that satisfies $\mathcal{P}$, when this exists. Given a concrete domain C and an abstract domain property $\mathcal{P} \subseteq \mathrm{uco}(C)$, the $\mathcal{P}$-shell of $\mu \in \mathrm{uco}(C)$, when this exists, is the most abstract domain that refines μ and satisfies $\mathcal{P}$. Giacobazzi et al. [10] show that backward complete shells always exist when the concrete operations are continuous.

Let us now consider the property of forward completeness. Let $F \subseteq \mathrm{Fun}(C)$ (thus functions in F may have any arity) and $S \in \wp(C)$. We denote by $F(S) \in \wp(C)$ the image of F on S, i.e. $F(S) \stackrel{\text{def}}{=} \{f(\vec{s}) \mid f \in F,\ \vec{s} \in S^{\mathrm{ar}(f)}\}$, and we say that S is F-closed when $F(S) \subseteq S$. An abstract domain $\mu \in \mathrm{uco}(C)$ is forward F-complete when μ is forward complete for any $f \in F$. Thus, the (forward) F-complete shell operator $\mathrm{S}_F : \mathrm{uco}(C) \to \mathrm{uco}(C)$ is defined as follows: $\mathrm{S}_F(\mu) \stackrel{\text{def}}{=} \sqcup \{\eta \in \mathrm{uco}(C) \mid \eta \sqsubseteq \mu,\ \eta \text{ is forward } F\text{-complete}\}$. As already observed by Giacobazzi and Quintarelli [9], it is easy to show that for any domain μ, $\mathrm{S}_F(\mu)$ is forward F-complete, namely forward complete shells always exist. It is worth noting that $\mathrm{S}_F \in \mathrm{lco}(\mathrm{uco}(C)_\sqsubseteq)$ and that $\mathrm{S}_F(\mu)$ is the smallest (w.r.t. set inclusion) set that contains μ and is both F-closed and meet-closed. When C is finite, note that for the meet operator $\wedge : C^2 \to C$ we have that, for any F, $\mathrm{S}_F = \mathrm{S}_{F \cup \{\wedge\}}$, because uco's are meet-closed.

We define $F^{\mathrm{uco}} : \mathrm{uco}(C) \to \mathrm{uco}(C)$ as $F^{\mathrm{uco}} \stackrel{\text{def}}{=} \mathcal{M} \circ F$, namely $F^{\mathrm{uco}}(\rho) = \mathcal{M}(\{f(\vec{x}) \mid f \in F,\ \vec{x} \in \rho^{\mathrm{ar}(f)}\})$. This operator characterizes forward F-completeness because it turns out that ρ is forward F-complete iff $\rho \sqsubseteq F^{\mathrm{uco}}(\rho)$. Moreover, given $\mu \in \mathrm{uco}(C)$, we consider the operator $F_\mu : \mathrm{uco}(C) \to \mathrm{uco}(C)$ defined by $F_\mu(\rho) \stackrel{\text{def}}{=} \mu \sqcap F^{\mathrm{uco}}(\rho)$ and we also note that $F_\mu(\rho) = \mathcal{M}(\mu \cup F(\rho))$. Observe that F_μ is monotone on $\mathrm{uco}(C)$ and therefore it admits (least and) greatest fixpoint. It turns out that we may constructively characterize the shell $\mathrm{S}_F(\mu)$ as the greatest fixpoint (denoted by gfp) in $\mathrm{uco}(C)$ of the operator F_μ.

Lemma 2.1. $\mathrm{S}_F(\mu) = \mathrm{gfp}(F_\mu)$.

Example 2.2. Let $\Sigma = \{1,2,3,4\}$ and $f_\circ : \Sigma \to \Sigma$ be the function $\{1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 4, 4 \mapsto 4\}$. Let $f : \wp(\Sigma) \to \wp(\Sigma)$ be the lifting of $f_\circ$ to the powerset, i.e., $f \overset{\mathrm{def}}{=} \lambda S.\{f_\circ(s) \mid s \in S\}$. Consider the abstract domain $\mu = \{\varnothing, 2, 1234\} \in \mathrm{uco}(\wp(\Sigma)_\subseteq)$. By Lemma 2.1, $\mathrm{S}_f(\mu) = \{\varnothing, 2, 3, 4, 34, 234, 1234\}$ because:

$$\begin{aligned}
\rho_0 &= \{1234\} \quad \text{(top uco)}\\
\rho_1 &= \mathcal{M}(\mu \cup f(\rho_0)) = \mathcal{M}(\mu \cup \{234\}) = \{\varnothing, 2, 234, 1234\}\\
\rho_2 &= \mathcal{M}(\mu \cup f(\rho_1)) = \mathcal{M}(\mu \cup \{\varnothing, 3, 34, 234\}) = \{\varnothing, 2, 3, 34, 234, 1234\}\\
\rho_3 &= \mathcal{M}(\mu \cup f(\rho_2)) = \mathcal{M}(\mu \cup \{\varnothing, 3, 4, 34, 234\}) = \{\varnothing, 2, 3, 4, 34, 234, 1234\}\\
\rho_4 &= \mathcal{M}(\mu \cup f(\rho_3)) = \mathcal{M}(\mu \cup \{\varnothing, 3, 4, 34, 234\}) = \rho_3 \quad \text{(greatest fixpoint).} \quad \square
\end{aligned}$$

3 Generalized Strong Preservation

Partitions as Abstract Domains. Let Σ be any (possibly infinite) set of system states. We recall from [16] how the the lattice of state partitions $\mathrm{Part}(\Sigma)$ can be viewed as an abstraction of the lattice of abstract domains $\mathrm{uco}(\wp(\Sigma))$. Our goal is to perform a *complete abstract computation* of a forward complete shell $\mathrm{S}_F(\mu) = \mathrm{gfp}(F_\mu)$ (cf. Lemma 2.1) on the lattice of partitions. Thus, we need to approximate a greatest fixpoint computation from above so that, as usual in abstract interpretation in these cases, we consider concrete and abstract domains with their dual ordering relations. Hence, we are looking for a GI of $\mathrm{Part}(\Sigma)_\succeq$ into $\mathrm{uco}(\wp(\Sigma))_\sqsupseteq$.

We define a mapping $\mathrm{par} : \mathrm{uco}(\wp(\Sigma)) \to \mathrm{Part}(\Sigma)$ by $\mathrm{par}(\mu) \overset{\mathrm{def}}{=} \{[s]_\mu \mid s \in \Sigma\}$, where $[s]_\mu \overset{\mathrm{def}}{=} \{s' \in \Sigma \mid \mu(\{s'\}) = \mu(\{s\})\}$. On the other hand, $\mathrm{pcl} : \mathrm{Part}(\Sigma) \to \mathrm{uco}(\wp(\Sigma))$ is defined by $\mathrm{pcl}(P) \overset{\mathrm{def}}{=} \lambda X \in \wp(\Sigma). \cup \{B \in P \mid X \cap B \neq \varnothing\}$, i.e. $\mathrm{pcl}(P)(X)$ is the minimal covering of the set $X \subseteq \Sigma$ through blocks in P. Observe that $\mathrm{pcl}(P)$ is indeed a uco whose set of fixpoints is given by all the unions of blocks in P, i.e. $\mathrm{pcl}(P) = \{\cup_i B_i \mid \{B_i\} \subseteq P\}$. It turns out that $(\mathrm{par}, \mathrm{uco}(\wp(\Sigma))_\sqsupseteq, \mathrm{Part}(\Sigma)_\succeq, \mathrm{pcl})$ is a GI. An abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ is *partitioning* — meaning that it represents exactly a partition; also, pcl stands for "partitioning closure" — when $\mathrm{pcl}(\mathrm{par}(\mu)) = \mu$ holds. We denote by $\mathrm{puco}(\wp(\Sigma))$ the set of partitioning abstract domains. As a consequence, the mappings pcl and par give rise to an order isomorphism allowing to view state partitions as partitioning uco's: $\mathrm{Part}(\Sigma)_\preccurlyeq \cong \mathrm{puco}(\wp(\Sigma))_\sqsubseteq$. It turns out that an abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ is partitioning iff $\forall S \in \mu.\ \complement(S) \in \mu$ iff (i) μ is additive and (ii) $\{\mu(\{s\})\}_{s \in \Sigma} \in \mathrm{Part}(\Sigma)$. Therefore, the partition associated to some $\mu \in \mathrm{puco}(\wp(\Sigma))$ is the set of μ-images of singletons, i.e. $\mathrm{par}(\mu) = \{\mu(\{s\}) \mid s \in \Sigma\}$.

Example 3.1. Consider $\Sigma = \{1,2,3,4\}$ and the corresponding lattice $\mathrm{Part}(\Sigma)_\preceq$. The uco's $\mu_1 = \{\varnothing, 12, 3, 4, 1234\}$, $\mu_2 = \{\varnothing, 12, 3, 4, 34, 1234\}$, $\mu_3 = \{\varnothing, 12, 3, 4, 34, 123, 124, 1234\}$, $\mu_4 = \{12, 123, 124, 1234\}$ and $\mu_5 = \{\varnothing, 12, 123, 124, 1234\}$ all induce the same partition $P = \mathrm{par}(\mu_i) = \{12, 3, 4\} \in \mathrm{Part}(\Sigma)$. Observe that μ_3 is the only partitioning closure because $\mathrm{pcl}(P) = \mu_3$. $\square$

Abstract Semantics and Generalized Strong Preservation. Let us now recall from [16] how to cast strong preservation in standard abstract model checking as forward completeness of abstract interpretations. We consider languages $\mathcal{L}$ whose syntactic state formulae φ are inductively defined by a BNF grammar: $\varphi ::= p \mid f(\varphi_1, ..., \varphi_n)$, where $p \in AP$ ranges over a set of atomic propositions that is left unspecified while f ranges over a finite set Op of operators (each $f \in Op$ has an arity $\mathrm{ar}(f) > 0$).

The interpretation of formulae in $\mathcal{L}$ is determined by a *semantic structure* $\mathcal{S} = (\Sigma, I, AP, \ell)$ where: Σ is any set of states, $I : Op \to \mathrm{Fun}(\wp(\Sigma))$ is an interpretation function such that for any $f \in Op$, $I(f) : \wp(\Sigma)^{\mathrm{ar}(f)} \to \wp(\Sigma)$, AP is a set of atomic propositions and $\ell : \Sigma \to \wp(AP)$ is a labelling function. Semantic structures generalize the role of Kripke structures by requiring that the semantic interpretation of a n-ary syntactic state operator is given by any n-ary mapping on $\wp(\Sigma)$. For $p \in AP$ and $f \in Op$ we will also use $\boldsymbol{p}$ and $\boldsymbol{f}$ to denote, respectively, $\{s \in \Sigma \mid p \in \ell(s)\}$ and $I(f)$. Also, $\boldsymbol{Op} \overset{\text{def}}{=} \{\boldsymbol{f} \in \mathrm{Fun}(\wp(\Sigma)) \mid f \in Op\}$. The *concrete state semantic function* $[\![\cdot]\!]_{\mathcal{S}} : \mathcal{L} \to \wp(\Sigma)$ evaluates a formula $\varphi \in \mathcal{L}$ to the set of states making φ true on the semantic structure $\mathcal{S}$, namely it is inductively defined as follows:

$$[\![p]\!]_{\mathcal{S}} = \boldsymbol{p} \text{ and } [\![f(\varphi_1, ..., \varphi_n)]\!]_{\mathcal{S}} = \boldsymbol{f}([\![\varphi_1]\!]_{\mathcal{S}}, ..., [\![\varphi_n]\!]_{\mathcal{S}}).$$

In the following, we will freely use standard logical and temporal operators together with their corresponding usual interpretations: for example, $I(\wedge) = \cap$, $I(\neg) = \complement$, $I(\mathrm{EX}) = \mathrm{pre}_R$, etc. We say that a language $\mathcal{L}$ is closed under a semantic operation $\boldsymbol{g} : \wp(\Sigma)^n \to \wp(\Sigma)$ when for any $\varphi_1, ..., \varphi_n \in \mathcal{L}$, there exists some $\psi \in \mathcal{L}$ such that $\boldsymbol{g}([\![\varphi_1]\!]_{\mathcal{S}}, ..., [\![\varphi_n]\!]_{\mathcal{S}}) = [\![\psi]\!]_{\mathcal{S}}$. It is straightforward to extend this notion to infinitary operators, e.g. infinite logical conjunction.

The state semantics $[\![\cdot]\!]_{\mathcal{S}}$ induces a state logical equivalence $\equiv^{\mathcal{S}}_{\mathcal{L}} \subseteq \Sigma \times \Sigma$ as usual: $s \equiv^{\mathcal{S}}_{\mathcal{L}} s'$ iff $\forall \varphi \in \mathcal{L}. s \in [\![\varphi]\!]_{\mathcal{S}} \Leftrightarrow s' \in [\![\varphi]\!]_{\mathcal{S}}$. The corresponding state partition is denoted by $P_{\mathcal{L}} \in \mathrm{Part}(\Sigma)$ (the index $\mathcal{S}$ for the underlying semantic structure is omitted).

For a number of well known temporal languages like CTL*, ACTL*, CTL*-X, it turns out that if a partition is more refined than $P_{\mathcal{L}}$ then it induces a standard *strongly preserving* (s.p.) abstract model. This means that if we interpret $\mathcal{L}$ on a Kripke structure $\mathcal{K} = (\Sigma, R, AP, \ell)$ and $P \preceq P_{\mathcal{L}}$ then one can define an abstract Kripke structure $\mathcal{A} = (P, R^{\sharp}, AP, \ell^{\sharp})$ that strongly preserves $\mathcal{L}$: for any $\varphi \in \mathcal{L}$ and for any $s \in \Sigma$ and $B \in P$ such that $s \in B$, we have that $B \in [\![\varphi]\!]_{\mathcal{A}} \Leftrightarrow s \in [\![\varphi]\!]_{\mathcal{K}}$. For example, $R^{\sharp} = R^{\exists\exists}$ for CTL* and $R^{\sharp} = R^{\forall\exists}$ for ACTL*, while $\ell^{\sharp}(B) = \cup_{s \in B} \ell(s)$ (see e.g. [3, 6]). Moreover, it turns out that $P_{\mathcal{L}}$ is the smallest s.p. abstract state space, namely if $(A, R^{\sharp}, AP, \ell^{\sharp})$ is any abstract Kripke structure that strongly preserves $\mathcal{L}$ then $|P_{\mathcal{L}}| \leq |A|$. Thus, following Dams [6], the notion of strong preservation can be given for generic state partitions: given a language $\mathcal{L}$ and a semantic structure $\mathcal{S}$, $P \in \mathrm{Part}(\Sigma)$ is strongly preserving for $\mathcal{L}$ (w.r.t. $\mathcal{S}$) when $P \preceq P_{\mathcal{L}}$. Recall that $P_{\ell} \in \mathrm{Part}(\Sigma)$ is the partition induced by the labeling ℓ and observe that $P_{\mathcal{L}} \preceq P_{\ell}$ always holds. Hence, it turns out that $P_{\mathcal{L}}$ is the coarsest refinement of P_{ℓ} which is s.p. for $\mathcal{L}$, namely $P_{\mathcal{L}}$ is the strongly preserving (for $\mathcal{L}$) shell of P_{ℓ}.

Abstract interpretation allows us to define *abstract semantics*. Consider any abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$. The *abstract semantic function* $[\![\cdot]\!]^{\mu}_{\mathcal{S}} : \mathcal{L} \to \mu$ induced by μ evaluates any $\varphi \in \mathcal{L}$ to an abstract value $[\![\varphi]\!]^{\mu}_{\mathcal{S}} \in \mu$. The semantics $[\![\cdot]\!]^{\mu}_{\mathcal{S}}$ is compositionally defined by interpreting any $p \in \mathit{AP}$ and $f \in \mathit{Op}$ as best correct approximations on the abstract domain μ of their concrete interpretations $\boldsymbol{p}$ and $\boldsymbol{f}$:

$$[\![p]\!]^\mu_{\mathcal{S}} = \mu(\boldsymbol{p}) \text{ and } [\![f(\varphi_1, ..., \varphi_n)]\!]^\mu_{\mathcal{S}} = \mu(\boldsymbol{f}([\![\varphi_1]\!]^\mu_{\mathcal{S}}, ..., [\![\varphi_n]\!]^\mu_{\mathcal{S}})).$$

Intuitively, the partition $P_{\mathcal{L}}$ induced by $\mathcal{L}$ is an abstraction of the semantics $[\![\cdot]\!]_{\mathcal{S}}$. We make this observation precise as follows. We define an abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ as strongly preserving for $\mathcal{L}$ (w.r.t. $\mathcal{S}$) when for any $S \in \wp(\Sigma)$ and $\varphi \in \mathcal{L}$: $\mu(S) \subseteq [\![\varphi]\!]^\mu_{\mathcal{S}} \Leftrightarrow S \subseteq [\![\varphi]\!]_{\mathcal{S}}$. As shown in [16], it turns out that this generalizes the notion of strong preservation from partitions to abstract domains because, by exploiting the above isomorphism between partitions and partitioning abstract domains, it turns out that P is s.p. for $\mathcal{L}$ w.r.t. $\mathcal{S}$ iff $\mathrm{pcl}(P)$ is s.p. for $\mathcal{L}$ w.r.t. $\mathcal{S}$. This provides the right framework for viewing strong preservation as a forward completeness property. Given a state space Σ, we associate to any set $S \subseteq \wp(\Sigma)$ a set of atomic propositions $\mathit{AP}_S \overset{\mathrm{def}}{=} \{p_X \mid X \in S\}$ and a corresponding labeling $\ell_S \overset{\mathrm{def}}{=} \lambda s \in \Sigma. \{p_X \in \mathit{AP}_S \mid s \in X\}$. In particular, this can be done for any abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ by viewing μ as a set of sets. Hence, given a state space Σ and an interpretation function $I : \mathit{Op} \to \mathrm{Fun}(\wp(\Sigma))$, any abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ determines the semantic structure $\mathcal{S}_\mu = (\Sigma, I, \mathit{AP}_\mu, \ell_\mu)$. The following result shows that strongly preserving shells indeed coincide with forward complete shells.

Theorem 3.2 ([16]). *Let $\mathcal{L}$ be closed under infinite logical conjunction. Then, for any $\mu \in \mathrm{uco}(\wp(\Sigma))$, $\mathrm{S}_{\boldsymbol{Op}}(\mu)$ is the most abstract domain that refines μ and is strongly preserving for $\mathcal{L}$ w.r.t. $\mathcal{S}_\mu$.*

This allows to characterize the coarsest s.p. partition $P_{\mathcal{L}}$ as a forward complete shell when $\mathcal{L}$ is closed under logical conjunction and negation.

Corollary 3.3 ([16]). *Let $\mathcal{L}$ be closed under infinite logical conjunction and negation. Then, $P_{\mathcal{L}} = \mathrm{par}(\mathrm{S}_{\boldsymbol{Op}}(\mathrm{pcl}(P_\ell)))$.*

4 GPT: A Generalized Paige-Tarjan Refinement Algorithm

In order to emphasize the ideas leading to our generalized Paige-Tarjan algorithm, let us first sketch how some relevant points in PT can be viewed and generalized from an abstract interpretation perspective.

***A New Perspective of* PT.** Consider a finite Kripke structure $(\Sigma, R, \mathit{AP}, \ell)$. In the following, we denote $\mathrm{Part}(\Sigma)$ simply by Part and pre_R by pre. As a consequence of Theorem 3.2, we showed in [16] that the output $\mathrm{PT}(P)$ of the Paige-Tarjan algorithm on input $P \in \mathrm{Part}$ is the abstraction through the map par of the forward $\{\mathrm{pre}, \complement\}$-complete shell of $\mathrm{pcl}(P)$, i.e. $\mathrm{PT}(P) = \mathrm{par}(\mathrm{S}_{\{\mathrm{pre},\complement\}}(\mathrm{pcl}(P)))$. Thus, $\mathrm{PT}(P)$ computes the partition abstraction of the most abstract domain that refines $\mathrm{pcl}(P)$ and is strongly preserving for Hennessy-Milner logic HML, namely, by Corollary 3.3, $\mathrm{PT}(P)$ computes the coarsest s.p. partition P_{HML}. On the other hand, Lemma 2.1 gives a constructive characterization of forward complete shells, meaning that it provides an iterative algorithm for computing a shell $\mathrm{S}_F(\mu)$: begin with $\rho = \top_{\mathrm{uco}(\wp(\Sigma))}$ and iteratively, at each step, compute $F_\mu(\rho)$

until a fixpoint is reached. This scheme could be in particular applied for computing $\mathrm{S}_{\{\mathrm{pre},\mathtt{C}\}}(\mathrm{pcl}(P))$. However, note that the algorithm induced by Lemma 2.1 is far from being efficient: at each step $F_\mu(\rho)$ always re-computes the images $f(\vec{s})$ that have been already computed at the previous step (cf. Example 2.2). Thus, in our abstract interpretation view, PT is an algorithm that computes *a particular abstraction of a particular forward complete shell*. Our goal is to analyze the basic steps of the PT algorithm in order to investigate whether it can be generalized from an abstract interpretation perspective to an algorithm that computes *a generic abstraction of a generic forward complete shell*. We isolate in our abstract interpretation framework the following key points concerning the PT algorithm. Let $P \in \mathrm{Part}$ be any partition.

(i) $\mathrm{PTsplit}(S,P) = \mathrm{par}(\mathcal{M}(\mathrm{pcl}(P) \cup \{\mathrm{pre}(S)\})) = P \curlywedge \{\mathrm{pre}(S), \mathtt{C}(\mathrm{pre}(S))\} = P \curlywedge \mathrm{par}(\mathcal{M}(\{\mathrm{pre}(S)\}))$.
(ii) $\mathrm{PTrefiners}(P) = \{S \in \mathrm{pcl}(P) \mid \mathrm{par}(\mathcal{M}(\mathrm{pcl}(P) \cup \{\mathrm{pre}(S)\})) \prec P\}$.
(iii) P is PT stable iff $\{S \in \mathrm{pcl}(P) \mid \mathrm{par}(\mathcal{M}(\mathrm{pcl}(P) \cup \{\mathrm{pre}(S)\})) \prec P\} = \varnothing$.

Point (i) provides a characterizaztion of the PT splitting step as best correct approximation on the abstract domain Part of the following domain refinement operation: given $S \subseteq \Sigma$, $\mathrm{refine}_{\mathrm{pre}}(S,\cdot) \stackrel{\mathrm{def}}{=} \lambda\mu.\mathcal{M}(\mu \cup \{\mathrm{pre}(S)\}) : \mathrm{uco}(\wp(\Sigma)) \rightarrow \mathrm{uco}(\wp(\Sigma))$. In turn, Points (ii) and (iii) yield a characterization of PTrefiners and PT stability based on this best correct approximation on Part of $\mathrm{refine}_{\mathrm{pre}}(S,\cdot)$. Thus, if $\mathrm{refine}^{\mathrm{Part}}_{\mathrm{pre}} : \mathrm{Part} \rightarrow \mathrm{Part}$ denotes the best correct approximation of $\mathrm{refine}_{\mathrm{pre}}(S,\cdot)$ on Part, we may view PT as follows.

```
while $\{T \in \mathrm{pcl}(P) \mid \mathrm{refine}^{\mathrm{Part}}_{\mathrm{pre}}(T,P) \prec P\} \neq \varnothing$ do
  choose $S \in \{T \in \mathrm{pcl}(P) \mid \mathrm{refine}^{\mathrm{Part}}_{\mathrm{pre}}(T,P) \prec P\}$;
  $P := \mathrm{refine}^{\mathrm{Part}}_{\mathrm{pre}}(S,P)$;
endwhile
```

In the following, we generalize this view of PT to generic abstract domains of $\mathrm{uco}(\wp(\Sigma))$ and isolate some conditions ensuring the correctness of this generalized algorithm.

Generalizing **PT.** We generalize Points (i)-(iii) above as follows. Let $F \subseteq \mathrm{Fun}(\wp(\Sigma))$. We define a family of refinement operators of abstract domains $\mathrm{refine}_f : \wp(\Sigma)^{\mathrm{ar}(f)} \rightarrow (\mathrm{uco}(\wp(\Sigma)) \rightarrow \mathrm{uco}(\wp(\Sigma)))$ indexed on functions $f \in F$ and tuples of sets $\vec{S} \in \wp(\Sigma)^{\mathrm{ar}(f)}$:

(i) $\mathrm{refine}_f(\vec{S},\mu) \stackrel{\mathrm{def}}{=} \mathcal{M}(\mu \cup \{f(\vec{S})\})$.

A tuple $\vec{S}$ is a F-refiner for an abstract domain μ when there exists $f \in F$ such that $\vec{S} \in \mu^{\mathrm{ar}(f)}$ and indeed $\vec{S}$ contributes to refine μ w.r.t. f, i.e., $\mathrm{refine}_f(\vec{S},\mu) \sqsubset \mu$. Thus:

(ii) $\mathrm{Refiners}_f(\mu) \stackrel{\mathrm{def}}{=} \{\vec{S} \in \mu^{\mathrm{ar}(f)} \mid \mathrm{refine}_f(\vec{S},\mu) \sqsubset \mu\}$;
$\mathrm{Refiners}_F(\mu) \stackrel{\mathrm{def}}{=} \cup_{f \in F} \mathrm{Refiners}_f(\mu)$.
(iii) μ is F-stable iff $\mathrm{Refiners}_F(\mu) = \varnothing$.

These simple observations lead us to design the following PT-like algorithm called CPT_F (Concrete PT), parameterized by F, taking as input an abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ and computing the forward F-complete shell of μ.

```
while ($\mathrm{Refiners}_F(\mu) \neq \varnothing$) do
  choose for some $f \in F$, $\vec{S} \in \mathrm{Refiners}_f(\mu)$;
  $\mu := \mathrm{refine}_f(\vec{S},\mu)$;
endwhile                                        $\mathrm{CPT}_F$
```

Lemma 4.1. *Let Σ be finite.* CPT_F *always terminates and, for any* $\mu \in \mathrm{uco}(\wp(\Sigma))$, $\mathrm{CPT}_F(\mu) = \mathrm{S}_F(\mu)$.

Example 4.2. Let us illustrate CPT on $\mu = \{\varnothing, 2, 1234\}$ of Example 2.2.

$$\begin{array}{ll}
\mu_0 = \mu = \{\varnothing, 2, 1234\} & S_0 = \{2\} \in \mathrm{Refiners}_f(\mu_0) \\
\mu_1 = \mathcal{M}(\mu_0 \cup \{f(S_0)\}) = \{\varnothing, 2, 3, 1234\} & S_1 = \{3\} \in \mathrm{Refiners}_f(\mu_1) \\
\mu_2 = \mathcal{M}(\mu_1 \cup \{f(S_1)\}) = \{\varnothing, 2, 3, 4, 1234\} & S_2 = \{1234\} \in \mathrm{Refiners}_f(\mu_2) \\
\mu_3 = \mathcal{M}(\mu_2 \cup \{f(S_2)\}) = \{\varnothing, 2, 3, 4, 234, 1234\} & S_3 = \{234\} \in \mathrm{Refiners}_f(\mu_3) \\
\mu_4 = \mathcal{M}(\mu_3 \cup \{f(S_3)\}) = \{\varnothing, 2, 3, 4, 34, 234, 1234\} & \Rightarrow \ \mathrm{Refiners}_f(\mu_4) = \varnothing
\end{array}$$

Let us note that while in Example 2.2 each step consists in computing the images of f for the sets belonging to the whole domain at the previous step and this gives rise to re-computations, here instead an image $f(S_i)$ is never computed twice because at each step we nondeterministically choose a refiner S and apply f to S. □

Our goal is to provide an abstract version of CPT_F that works on a generic abstraction A of the lattice $\mathrm{uco}(\wp(\Sigma))$. As recalled at the beginning of Section 3, since we aim at designing an algorithm for computing an abstract greatest fixpoint, viz. $\alpha(\mathrm{CPT}_F(\mu))$ for some abstraction map α, we need to approximate this greatest fixpoint computation "from above" instead of "from below" as it happens for least fixpoint computations. Thus, we consider a Galois insertion $(\alpha, \mathrm{uco}(\wp(\Sigma))_{\sqsupseteq}, A_{\geq}, \gamma)$ of an abstract domain $A_{\geq}$ into the dual lattice of abstract domains $\mathrm{uco}(\wp(\Sigma))_{\sqsupseteq}$. We denote by $\geq$ the ordering relation of the abstract domain A, because this makes the concrete and abstract ordering notations uniform. Notice that since we consider a Galois insertion of A into the complete lattice $\mathrm{uco}(\wp(\Sigma))$, by standard results [5], it turns out that A must be a complete lattice as well. Also, we denote by $\rho_A \overset{\text{def}}{=} \gamma \circ \alpha$ the corresponding uco on $\mathrm{uco}(\wp(\Sigma))_{\sqsupseteq}$. For any $f \in F$, the best correct approximation $\mathrm{refine}_f^A : \wp(\Sigma)^{\mathrm{ar}(f)} \to (A \to A)$ of refine_f on A is therefore defined as usual:

(i) $\mathrm{refine}_f^A(\vec{S}, a) \overset{\text{def}}{=} \alpha(\mathrm{refine}_f(\vec{S}, \gamma(a)))$.

Accordingly, abstract refiners and stability go as follows:

(ii) $\mathrm{Refiners}_f^A(a) \overset{\text{def}}{=} \{\vec{S} \in \gamma(a)^{\mathrm{ar}(f)} \mid \mathrm{refine}_f^A(\vec{S}, a) < a\}$;
$\mathrm{Refiners}_F^A(a) \overset{\text{def}}{=} \cup_{f \in F} \mathrm{Refiners}_f^A(a)$.

(iii) An abstract object $a \in A$ is F-stable iff $\mathrm{Refiners}_F^A(a) = \varnothing$.

It is worth remarking that $a \in A$ is F-stable iff $\gamma(a)$ is forward F-complete. We may now define the following abstract version of the above algorithm CPT_F, called GPT_F^A (Generalized PT), parameterized on the abstract domain A. $\mathrm{GPT}_F^A(a)$ computes a sequence of abstract objects $\{a_i\}_{i \in \mathbb{N}}$

```
input: abstract object $a \in A$
while ($\mathrm{Refiners}_F^A(a) \neq \varnothing$) do
  choose for some $f \in F$, $\vec{S} \in \mathrm{Refiners}_f^A(a)$;
  $a := \mathrm{refine}_f^A(\vec{S}, a)$;
endwhile                                   $\mathrm{GPT}_F^A$
```

which is a decreasing chain in A. Thus, in order to ensure termination of GPT_F^A it is enough to consider an abstract domain A satisfying the descending chain condition (DCC). Furthermore, let us remark that correctness for GPT_F^A means that for any $a \in A$,

$\mathrm{GPT}_F(a) = \alpha(\mathrm{S}_F(\gamma(a)))$. Note that, by Lemma 2.1, $\alpha(\mathrm{S}_F(\gamma(a))) = \alpha(\mathrm{gfp}(F_{\gamma(a)}))$. It should be clear that correctness for GPT is somehow related to backward completeness in abstract interpretation. In fact, if the abstract domain A is backward complete for $F_\mu = \lambda\rho.\mu \sqcap F^{\mathrm{uco}}(\rho)$ then, by Lemma 2.1, $\alpha(\mathrm{gfp}(F_\mu)) = \mathrm{gfp}(F_\mu^A)$, where F_μ^A is the best correct approximation of the operator F_μ on the abstract domain A, and $\mathrm{GPT}_F^A(a)$ intuitively is an algorithm for computing $\mathrm{gfp}(F_\mu^A)$. Indeed, the following result shows that GPT_F^A is correct when A is backward complete for F^{uco}, because this implies that A is backward complete for F_μ, for any μ. Moreover, we also isolate the following condition ensuring correctness for GPT_F^A: the forward F-complete shell of any concretization $\gamma(a)$ still belongs to $\gamma(A)$, namely A is forward complete for the forward F-complete shell S_F.

Theorem 4.3. *Let A be DCC and assume that one of the following conditions holds:*

(i) $\rho_A \circ F^{\mathrm{uco}} \circ \rho_A = \rho_A \circ F^{\mathrm{uco}}$.
(ii) $\rho_A \circ \mathrm{S}_F \circ \rho_A = \mathrm{S}_F \circ \rho_A$ *(i.e., $\forall a \in A.\, \mathrm{S}_F(\gamma(a)) \in \gamma(A)$).*

Then, GPT_F^A always terminates and for any $a \in A$, $\mathrm{GPT}_F^A(a) = \alpha(\mathrm{S}_F(\gamma(a)))$.

Corollary 4.4. *Under the hypotheses of Theorem 4.3, for any $a \in A$, $\mathrm{GPT}_F^A(a)$ is the F-stable shell of a.*

Example 4.5. Let us consider again Example 2.2. Recall that an abstract domain $\rho \in \mathrm{uco}(\wp(\Sigma))$ is disjunctive iff for any (possibly empty) $S \subseteq \rho$, $\cup S \in \rho$. We denote by $\mathrm{duco}(\wp(\Sigma))$ the set of disjunctive domains in $\mathrm{uco}(\wp(\Sigma))$. Thus, the disjunctive shell $\mathrm{S}_\cup : \mathrm{uco}(\wp(\Sigma)) \to \mathrm{duco}(\wp(\Sigma))$ maps any ρ to the well-known disjunctive completion $\mathrm{S}_\cup(\rho) = \{\cup S \mid S \subseteq \rho\}$ of ρ (see [5]). It turns out that $\mathrm{duco}(\wp(\Sigma))$ is indeed an abstract domain of $\mathrm{uco}(\wp(\Sigma))_\sqsupseteq$, namely $(\mathrm{S}_\cup, \mathrm{uco}(\wp(\Sigma))_\sqsupseteq, \mathrm{duco}(\wp(\Sigma))_\sqsupseteq, id)$ is a GI.

It turns out that condition (i) of Theorem 4.3 is satisfied for this GI. In fact, by exploiting the fact that, by definition, $f : \wp(\Sigma) \to \wp(\Sigma)$ is additive, it is not hard to verify that $\mathrm{S}_\cup \circ f^{\mathrm{uco}} \circ \mathrm{S}_\cup = \mathrm{S}_\cup \circ f^{\mathrm{uco}}$. Thus, let us apply $\mathrm{GPT}_f^{\mathrm{duco}}$ to the disjunctive abstract domain $\mu_0 = \{\varnothing, 2, 1234\} = \mathrm{S}_\cup(\{2, 1234\}) \in \mathrm{duco}(\wp(\Sigma))$.

$$\begin{aligned}
&\mu_0 = \mu = \{\varnothing, 2, 1234\} && S_0 = \{2\} \in \mathrm{Refiners}_f^{\mathrm{duco}}(\mu_0)\\
&\mu_1 = \mathrm{S}_\cup(\mathcal{M}(\mu_0 \cup \{f(S_0)\})) = \{\varnothing, 2, 3, 23, 1234\} && S_1 = \{3\} \in \mathrm{Refiners}_f^{\mathrm{duco}}(\mu_1)\\
&\mu_2 = \mathrm{S}_\cup(\mathcal{M}(\mu_1 \cup \{f(S_1)\})) &&\\
&\quad = \{\varnothing, 2, 3, 4, 23, 24, 34, 234, 1234\} && \Rightarrow\ \mathrm{Refiners}_f^{\mathrm{duco}}(\mu_2) = \varnothing
\end{aligned}$$

From Example 4.2 we know that $\mathrm{S}_f(\mu_0) = \{\varnothing, 2, 3, 4, 34, 234, 1234\}$. Thus, as expected from Theorem 4.3, $\mathrm{GPT}_f^{\mathrm{duco}}(\mu_0)$ coincides with $\mathrm{S}_\cup(\mathrm{S}_f(\mu_0)) = \{\varnothing, 2, 3, 4, 23, 24, 34, 234, 1234\}$. Note that we reached the abstract fixpoint in two iterations, whereas in Example 4.2 the concrete computation by CPT_f needed four iterations. □

An Optimization of GPT. As pointed out in [15], PT works even if we choose splitters among blocks instead of unions of blocks, i.e., if we replace $\mathrm{PTrefiners}(P)$ with the subset of "block refiners" $\mathrm{PTblockrefiners}(P) \overset{\mathrm{def}}{=} \mathrm{PTrefiners}(P) \cap P$. This can be easily generalized as follows. Given $g \in F$, for any $a \in A$, let $\mathrm{subRefiners}_g^A(a) \subseteq \mathrm{Refiners}_g^A(a)$. We denote by IGPT_F^A (Improved GPT) the version of GPT_F^A where $\mathrm{Refiners}_g^A$ is replaced with $\mathrm{subRefiners}_g^A$.

Corollary 4.6. *Let $g \in F$ be such that, for any $a \in A$, $\mathrm{subRefiners}_g^A(a) = \varnothing \Leftrightarrow \mathrm{Refiners}_g^A(a) = \varnothing$. Then, for any $a \in A$, $\mathrm{GPT}_F^A(a) = \mathrm{IGPT}_F^A(a)$.*

Instantiating* GPT *with Partitions. Let the state space Σ be finite. The following properties (1) and (2) are consequences of the fact that a partitioning abstract domain $\mathrm{pcl}(P)$ is closed under complements, i.e. $X \in \mathrm{pcl}(P)$ iff $\complement(X) \in \mathrm{pcl}(P)$.

(1) $\mathrm{Refiners}_{\complement}^{\mathrm{Part}}(P) = \varnothing$.
(2) For any f and $\vec{S} \in \wp(\Sigma)^{\mathrm{ar}(f)}$, $\mathrm{refine}_f^{\mathrm{Part}}(\vec{S}, P) = P \curlywedge \{f(\vec{S}), \complement(f(\vec{S}))\}$.

Thus, by Point (1), for any $F \subseteq \mathrm{Fun}(\wp(\Sigma))$, a partition $P \in \mathrm{Part}$ is F-stable iff P is $(F \cup \{\complement\})$-stable, that is complements can be left out. Hence, if $F^{\text{-}\complement}$ denotes $F \smallsetminus \{\complement\}$ then $\mathrm{GPT}_F^{\mathrm{Part}}$ may be simplified as follows. Note that the number of iterations of $\mathrm{GPT}_F^{\mathrm{Part}}$ is bounded by the height of the lattice Part, that is by the number of states $|\Sigma|$. Thus, if each refinement step involving some $f \in F$ takes $O(\mathrm{cost}(f))$ time then the time complexity of $\mathrm{GPT}_F^{\mathrm{Part}}$ is bounded by $O(|\Sigma| \max(\{\mathrm{cost}(f) \mid f \in F\}))$.

```
while (Refiners$_{F^{-\complement}}^{\mathrm{Part}}(a) \neq \varnothing$) do
  choose for some $f \in F^{-\complement}$, $\vec{S} \in \mathrm{Refiners}_f^{\mathrm{Part}}(a)$;
  $P := P \curlywedge \{f(\vec{S}), \complement(f(\vec{S}))\}$;
endwhile                                                  $\mathrm{GPT}_F^{\mathrm{Part}}$
```

Let us now consider a language $\mathcal{L}$ with operators in Op and let (Σ, I, AP, ℓ) be a semantic structure for $\mathcal{L}$. If $\mathcal{L}$ is closed under logical conjunction and negation then, for any $\mu \in \mathrm{uco}(\wp(\Sigma))$, $\mathrm{S}_{\boldsymbol{Op}}(\mu)$ is closed under complements and therefore it is a partitioning abstract domain. Thus, condition (ii) of Theorem 4.3 is satisfied. As a consequence of Corollary 3.3 we obtain the following characterization.

Corollary 4.7. *If $\mathcal{L}$ is closed under conjunction and negation then* $\mathrm{GPT}_{\boldsymbol{Op}}^{\mathrm{Part}}(P_\ell) = P_{\mathcal{L}}$.

This provides a parameteric algorithm for computing the coarsest strongly preserving partition $P_{\mathcal{L}}$ induced by a generic language $\mathcal{L}$ including propositional logic.
PT *as an Instance of* GPT. It is now immediate to obtain PT as an instance of GPT. We know that $\mathrm{GPT}_{\{\mathrm{pre},\complement\}}^{\mathrm{Part}} = \mathrm{GPT}_{\mathrm{pre}}^{\mathrm{Part}}$. Moreover, by Points (i) and (ii) above:

$$P \curlywedge \{\mathrm{pre}(S), \complement(\mathrm{pre}(S))\} = \mathrm{PTsplit}(S, P) \ \text{ and } \ \mathrm{Refiners}_{\mathrm{pre}}^{\mathrm{Part}}(P) = \mathrm{PTrefiners}(P).$$

Hence, by Point (iii), it turns out that $P \in \mathrm{Part}$ is PT stable iff $\mathrm{Refiners}_{\mathrm{pre}}^{\mathrm{Part}}(P) = \varnothing$. Thus, the instance $\mathrm{GPT}_{\mathrm{pre}}^{\mathrm{Part}}$ provides *exactly* the PT algorithm. Also, correctness follows from Corollaries 4.4 and 4.7: $\mathrm{GPT}_{\mathrm{pre}}^{\mathrm{Part}}(P)$ is both the coarsest PT stable refinement of P and the coarsest strongly preserving partition P_{HML}.

5 Applications

5.1 Simulation Equivalence and Henzinger et al.'s Algorithm

It is well known that simulation equivalence is an appropriate state equivalence to be used in abstract model checking because it strongly preserves ACTL* and provides a better state-space reduction than bisimulation equivalence. However, computing simulation equivalence is harder than bisimulation [14]. Henzinger et al. [13] provide an algorithm, here called HHK, for computing simulation equivalence which runs in $O(|R||\Sigma|)$-time. As far as time-complexity is concerned, HHK is the best available algorithm for this problem. We show here that HHK can be obtained as an instance of our algorithmic scheme GPT.

Consider a finite Kripke structure $\mathcal{K} = (\Sigma, R, AP, \ell)$ and let $\equiv_{\text{sim}}$ and P_{sim} denote, respectively, simulation equivalence on $\mathcal{K}$ and its corresponding partition. Henzinger et al.'s algorithm maintains, for any state $s \in \Sigma$, a set of states $\text{sim}(s) \subseteq \Sigma$. Initially, $\text{sim}(s) = [s]_\ell$ and at each iteration some $\text{sim}(s)$ is reduced, so that at the end $s \equiv_{\text{sim}} s'$ iff $s \in \text{sim}(s')$ and $s' \in \text{sim}(s)$. The algorithmic scheme HHK is as follows.

```
**for all** $s \in \Sigma$ **do** $\text{sim}(s) := \{s' \in \Sigma \mid \ell(s') = \ell(s)\}$ **endfor**
**while** $(\exists u, v, w \in \Sigma.\ u \in \text{pre}(\{v\})\ \&\ w \in \text{sim}(u)\ \&$
                                        $w \notin \text{pre}(\text{sim}(v)))$ **do**
    $\text{sim}(u) := \text{sim}(u) \smallsetminus \{w\}$;
**endwhile**                                                          HHK
```

Let us show how to cast HHK as an instance of GPT. In this case, we consider the abstraction of $\text{uco}(\wp(\Sigma))$ given by the disjunctive abstract domains $\text{duco}(\wp(\Sigma))$, namely additive closures, that we already defined in Example 4.5. Thus, $\text{S}_\cup : \text{uco}(\wp(\Sigma)) \to \text{duco}(\wp(\Sigma))$ is the disjunctive completion and $(\text{S}_\cup, \text{uco}(\wp(\Sigma))_\sqsupseteq, \text{duco}(\wp(\Sigma))_\sqsupseteq, id)$ is the corresponding Galois insertion. Any disjunctive abstract domain $\rho \in \text{duco}(\wp(\Sigma))$ is completely determined by the images of the singletons $\{s\}$ because, for any $X \in \wp(\Sigma)$, $\rho(X) = \cup_{x \in X} \rho(\{x\})$. Hence, any $\rho \in \text{duco}(\wp(\Sigma))$ can be represented by the set $\{\rho(\{s\})\}_{s \in \Sigma}$, and conversely any set of sets $\mathcal{S} = \{S_s\}_{s \in \Sigma}$ indexed on Σ determines a disjunctive abstract domain that we denote by $\rho_\mathcal{S}$. This shows that HHK can be viewed as an algorithm which maintains and refines a disjunctive abstract domain of $\wp(\Sigma)$ determined by the current $\{\text{sim}(s)\}_{s \in \Sigma}$.

On the other hand, it is known (see e.g. [17–Section 8]) that simulation equivalence on $\mathcal{K}$ coincides with the state equivalence induced by the following language $\mathcal{L}$: $\varphi ::= p \mid \varphi_1 \wedge \varphi_2 \mid \text{EX}\varphi$, namely, $P_{\text{sim}} = P_\mathcal{L}$. Moreover, as already observed in Example 4.5, it turns out that that $\text{S}_\cup \circ \text{pre}^{\text{uco}} \circ \text{S}_\cup = \text{S}_\cup \circ \text{pre}^{\text{uco}}$. Thus, by Theorem 4.3, $\text{GPT}^{\text{duco}}_{\text{pre}}(P_\ell) = \text{S}_\cup(\text{S}_{\text{pre}}(\text{pcl}(P_\ell)))$, and in turn $\text{par}(\text{GPT}^{\text{duco}}_{\text{pre}}(P_\ell)) = \text{par}(\text{S}_\cup(\text{S}_{\text{pre}}(\text{pcl}(P_\ell))))$. By Theorem 3.2, we know that $\text{S}_{\text{pre}}(\text{pcl}(P_\ell))$ is the most abstract domain which is strongly preserving for $\mathcal{L}$. As a consequence, it turns out that $\text{par}(\text{S}_\cup(\text{S}_{\text{pre}}(\text{pcl}(P_\ell)))) = P_\mathcal{L} = P_{\text{sim}}$. Thus, we showed that $\text{GPT}^{\text{duco}}_{\text{pre}}(P_\ell) = P_{\text{sim}}$, namely $\text{GPT}^{\text{duco}}_{\text{pre}}$ allows to compute simulation equivalence.

Even more, it turns out that $\text{GPT}^{\text{duco}}_{\text{pre}}$ exactly coincides with HHK and therefore admits the $O(|R||\Sigma|)$-time implementation described in [13]. In fact, if $\mathcal{S} = \{\text{sim}(s)\}_{s \in \Sigma}$ is the current set of sets maintained by HHK then it is possible to show that: (1) the condition of the while loop in HHK for $\mathcal{S}$ is exactly equivalent to pre-stability for the corresponding disjunctive abstract domain $\rho_\mathcal{S}$; (2) the refinement of $\mathcal{S}$ in HHK is precisely a step of pre-refinement of the additive uco $\rho_\mathcal{S}$ in $\text{GPT}^{\text{duco}}_{\text{pre}}$.

5.2 Stuttering Equivalence and Groote-Vaandrager Algorithm

Behavioural *stuttering*-based equivalences originated as state equivalences induced by languages without a next-time operator [7]. We are interested here in *divergence blind* *stuttering* (dbs for short) equivalence. Given a Kripke structure $\mathcal{K} = (\Sigma, R, AP, \ell)$, we denote by $P_{\text{dbs}} \in \text{Part}(\Sigma)$ the partition corresponding to the largest dbs equivalence on $\mathcal{K}$. We showed in [16] that P_{dbs} coincides with the coarsest strongly preserving partition $P_{\mathcal{L}}$ for the following language $\mathcal{L}$: $\varphi ::= \; p \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \text{EU}(\varphi_1, \varphi_2)$, where the semantics $\mathbf{EU} : \wp(\Sigma)^2 \to \wp(\Sigma)$ of the existential until EU is as usual:

$$\mathbf{EU}(S_1, S_2) = S_2 \cup \{s \in S_1 \mid \exists s_0, ..., s_n \in \Sigma, \text{ with } n \geq 0, \text{ such that (i) } s_0 = s, \\ \text{(ii) } \forall i \in [0, n).\, s_i \in S_1,\; s_i \xrightarrow{R} s_{i+1},\; \text{(iii) } s_n \in S_2\}.$$

Therefore, as a straight instance of Corollary 4.7, it turns out that $\text{GPT}^{\text{Part}}_{\mathbf{EU}}(P_\ell) = P_{\mathcal{L}} = P_{\text{dbs}}$. Groote and Vaandrager [11] designed the following partition refinement algorithm, here denoted by GV, for computing the partition P_{dbs}, where, for $B_1, B_2 \in P$,[1]

```
$P := P_\ell$;
while $\text{GVrefiners}(P) \neq \varnothing$ do
  choose $\langle B_1, B_2\rangle \in \text{GVrefiners}(P)$;
  $P := \text{GVsplit}(\langle B_1, B_2\rangle, P)$;
endwhile                                        GV
```

$$\begin{aligned} \text{GVsplit}(\langle B_1, B_2\rangle, P) &\overset{\text{def}}{=} P \curlywedge \{\mathbf{EU}(B_1, B_2), \complement(\mathbf{EU}(B_1, B_2))\} \\ \text{GVrefiners}(P) &\overset{\text{def}}{=} \{\langle B_1, B_2\rangle \in P \times P \mid \text{GVsplit}(\langle B_1, B_2\rangle, P) \prec P\}. \end{aligned}$$

Groote and Vaandrager show how GV can be efficiently implemented in $O(|R||\Sigma|)$-time. Indeed, it turns out that GV *exactly* coincides with $\text{IGPT}^{\text{Part}}_{\mathbf{EU}}$. This is a consequence of the following two facts:

(1) $\text{GVrefiners}(P) = \varnothing$ iff $\text{Refiners}^{\text{Part}}_{\mathbf{EU}}(P) = \varnothing$;
(2) $\text{GVsplit}(\langle B_1, B_2\rangle, P) = \text{refine}^{\text{Part}}_{\mathbf{EU}}(\langle B_1, B_2\rangle, P)$.

Hence, by Corollary 4.6, Point (1) allows us to exploit the $\text{IGPT}^{\text{Part}}_{\mathbf{EU}}$ algorithm in order to choose refiners for $\mathbf{EU}$ among the pairs of blocks of the current partition, so that by Point (2) we obtain that $\text{IGPT}^{\text{Part}}_{\mathbf{EU}}$ exactly coincides with the GV algorithm.

5.3 A Language Expressing Reachability

Let us consider the following language $\mathcal{L}$ which is able to express propositional logic and reachability: $\varphi ::= p \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \text{EF}\varphi$. Given a Kripke structure (Σ, R, AP, ℓ), the interpretation $\mathbf{EF} : \wp(\Sigma) \to \wp(\Sigma)$ of the reachability operator EF is as usual: $\mathbf{EF}(S) \overset{\text{def}}{=} \mathbf{EU}(\Sigma, S)$. Since $\mathcal{L}$ includes propositional logic, by Corollary 4.7, we have that the instance $\text{GPT}^{\text{Part}}_{\mathbf{EF}}$ allows to compute the coarsest strongly preserving partition $P_{\mathcal{L}}$, namely $\text{GPT}^{\text{Part}}_{\mathbf{EF}}(P_\ell) = P_{\mathcal{L}}$. Also, we may restrict ourselves to "block refiners", that is, $\text{BlockRefiners}^{\text{Part}}_{\mathbf{EF}}(P) = \{B \in P \mid P \curlywedge \{\mathbf{EF}(B), \complement(\mathbf{EF}(B))\} \prec P\}$. In fact, it turns out that $\text{BlockRefiners}^{\text{Part}}_{\mathbf{EF}}(P) = \varnothing$ iff $\text{Refiners}^{\text{Part}}_{\mathbf{EF}}(P) = \varnothing$. Therefore, by exploiting Corollary 4.6, we have that $\text{IGPT}^{\text{Part}}_{\mathbf{EF}}(P_\ell) = P_{\mathcal{L}}$, where $\text{IGPT}^{\text{Part}}_{\mathbf{EF}}$ is as follows. Our implementation of $\text{IGPT}^{\text{Part}}_{\mathbf{EF}}$ exploits the following "stability under refinement"

```
while ($\text{BlockRefiners}^{\text{Part}}_{\mathbf{EF}}(P) \neq \varnothing$) do
  choose $B \in \text{BlockRefiners}^{\text{Part}}_{\mathbf{EF}}(P)$;
  $P := P \curlywedge \{\mathbf{EF}(B), \complement(\mathbf{EF}(B))\}$;
endwhile                                        $\text{IGPT}^{\text{Part}}_{\mathbf{EF}}$
```

[1] In [11], $\text{pos}(B_1, B_2)$ denotes $\mathbf{EU}(B_1, B_2) \cap B_1$.

property: if $Q \preceq P$ and B is a block of both P and Q then $P \curlywedge \{\mathbf{EF}(B), \complement(\mathbf{EF}(B))\} = P$ implies $Q \curlywedge \{\mathbf{EF}(B), \complement(\mathbf{EF}(B))\} = Q$. As a consequence, if some block B of the current partition P_{curr} is not a $\mathbf{EF}$-refiner for P_{curr} and B is also a block of the next partition P_{next} then B cannot be a $\mathbf{EF}$-refiner for P_{next}. This suggests an implementation of $\mathrm{IGPT}^{\mathrm{Part}}_{\mathbf{EF}}$ based on the following points:

(1) to represent the current partition P as a doubly linked *list* of blocks;
(2) to scan *list* from the beginning in order to find block refiners;
(3) when a block B of the current partition P is split in B_1 and B_2 then we remove B from *list* and we append B_1 and B_2 at the end of *list*.

This leads to the following refinement of $\mathrm{IGPT}^{\mathrm{Part}}_{\mathbf{EF}}$.

```
$list$ := list of blocks in $P$;
scan $B$ in $list$
  compute $\mathbf{EF}(B)$;
  $current\_end$ points to the end of $list$;
  scan $B'$ in $list$ up to $current\_end$
    if ($B' \cap \mathbf{EF}(B) \neq \varnothing$ and $B' \smallsetminus \mathbf{EF}(B) \neq \varnothing$) then
      { remove $B'$ from $list$; append $B' \cap \mathbf{EF}(B)$ and $B' \smallsetminus \mathbf{EF}(B)$ to $list$; }
  endscan
endscan
```

It is not hard to devise a practical implementation of this algorithm requiring $O(|R||\Sigma|)$ time. As a preprocessing step we first compute the DAG of the strongly connected components (s.c.c.'s) of the directed graph (Σ, R) that we denote by $\mathrm{DAG}_{(\Sigma,R)}$. This can be done through a well-known algorithm (see e.g. [1]) running in $O(|\Sigma|+|R|)$-time, i.e. by totality of the transition relation R, in $O(|R|)$-time. Moreover, this algorithm returns the s.c.c.'s in $\mathrm{DAG}_{(\Sigma,R)}$ in topological order. This allows us to represent $\mathrm{DAG}_{(\Sigma,R)}$ through an adjacency list where the s.c.c.'s are recorded in reversed topological ordering in a list L. Thus, if S and S' are two s.c.c.'s of (Σ, R) such that there exists $s \in S$ and $s' \in S'$ with $s \xrightarrow{R} s'$ then S follows S' in the list L. By exploiting this representation of $\mathrm{DAG}_{(\Sigma,R)}$ we are able:

(1) to compute $\mathbf{EF}(B)$, for some block $B \in list$, in $O(|R|)$-time;
(2) to execute the inner scan loop in $O(|\Sigma|)$-time.

Hence, each iteration of the outer scan loop costs $O(|R|)$-time, because, by totality of R, $|\Sigma| \leq |R|$. Moreover, it turns out that the number of iterations of the outer scan loop is in $O(|\Sigma|)$. We thus obtain that this implementation of $\mathrm{IGPT}^{\mathrm{Part}}_{\mathbf{EF}}$ runs in $O(|R||\Sigma|)$-time.

6 Related and Future work

Related Work. Dams [6–Chapter 5] presents a generic splitting algorithm which, for a given language $\mathcal{L} \subseteq \mathrm{ACTL}$, computes an abstract model $A \in \mathrm{Abs}(\wp(\Sigma))$ that strongly preserves $\mathcal{L}$. This technique is inherently different from ours, in particular because it is guided by a splitting operation of an abstract state that depends on a given formula of ACTL. Additionally, Dams' methodology does not guarantee optimality of the resulting strongly preserving abstract model, as instead we do, because his algorithm may provide strongly preserving models which are too concrete. Dams [6–Chapter 6] also presents a generic partition refinement algorithm that computes a given (behavioural) state equivalence and generalizes PT (i.e., bisimulation equivalence) and Groote and Vaandrager (i.e., stuttering equivalence) algorithms. This algorithm is parameterized on a notion of splitter corresponding to some state equivalence, while our algorithm is directly parameterized on a given language: the example given in [6] (a "flat" version of CTL-X) seems to indicate that finding the right definition of splitter for some language may be a hard task. Gentilini et al. [8] provide an algorithm that solves a so-called generalized coarsest partition problem, meaning that they generalized PT stability to partitions endowed with an acyclic relation. They show that this technique can be instantiated to obtain a logarithmic algorithm for PT stability and an efficient algorithm for simulation equivalence. This approach is very different from ours since the partition refinement algorithm is not driven by strong preservation w.r.t. some language.

Future Work. GPT is parameteric on a domain of abstract models which is an abstraction of the lattice of abstract domains $\mathrm{Abs}(\wp(\Sigma))$. We instantiated GPT to the lattice $\mathrm{Part}(\Sigma)$ of partitions and to the lattice $\mathrm{DisjAbs}(\wp(\Sigma))$ of disjunctive abstract domains. We plan to investigate whether the GPT scheme can be applied to new domains of abstract models. In particular, models which are abstractions of $\mathrm{Part}(\Sigma)$ could be useful for computing *approximations of strongly preserving partitions*. As an example, if we are interested in reducing only a portion $S \subseteq \Sigma$ of the state space Σ, we may consider the domain $\mathrm{Part}(S)$ as an abstraction of $\mathrm{Part}(\Sigma)$ in order to get strong preservation only on the portion S.

Acknowledgments. This work was partially supported by the FIRB Project "*Abstract interpretation and model checking for the verification of embedded systems*", by the COFIN2002 Project "COVER: *Constraint based verification of reactive systems*" and by the COFIN2004 Project "AIDA: *Abstract Interpretation Design and Applications*".

References

1. A.V. Aho, J.E. Hopcroft and J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.
2. M.C. Browne, E.M. Clarke and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theor. Comp. Sci.*, 59:115-131, 1988.
3. E.M. Clarke, O. Grumberg and D.A. Peled. *Model Checking*. The MIT Press, 1999.
4. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM POPL*, 238-252, 1977.
5. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6th ACM POPL*, 269-282, 1979.
6. D. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD Thesis, Eindhoven Univ., 1996.
7. R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. *J. ACM*, 42(2):458-487, 1995

8. R. Gentilini, C. Piazza and A. Policriti. From bisimulation to simulation: coarsest partition problems. *J. Automated Reasoning*, 31(1):73-103, 2003.
9. R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples and refinements in abstract model checking. In *Proc. 8th SAS*, LNCS 2126:356-373, 2001.
10. R. Giacobazzi, F. Ranzato and F. Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361-416, 2000.
11. J.F. Groote and F. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In *Proc. 17th ICALP*, LNCS 443:626-638, 1990.
12. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137-161, 1985.
13. M.R. Henzinger, T.A. Henzinger and P.W. Kopke. Computing simulations on finite and infinite graphs. In *Proc. 36th FOCS*, 453-462, 1995.
14. A. Kucera and R. Mayr. Why is simulation harder than bisimulation? In *Proc. 13th CONCUR*, LNCS 2421:594-610, 2002.
15. R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973-989, 1987
16. F. Ranzato and F. Tapparo. Strong preservation as completeness in abstract interpretation. In *Proc. 13th ESOP*, LNCS 2986:18-32, 2004.
17. R.J. van Glabbeek. The linear time - branching time spectrum I: the semantics of concrete sequential processes. In *Handbook of Process Algebra*, pp. 3-99, Elsevier, 2001.