Electronic Notes in Theoretical Computer Science 48 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume48.html pp. 1 – 18

Refining and Compressing Abstract Model Checking[1]

Agostino Dovier[2] Roberto Giacobazzi[3]

Dipartimento di Informatica
Università di Verona
Strada Le Grazie 15
37134 Verona (Italy)

Elisa Quintarelli[4]

Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci 32
20133 Milano (Italy)

Abstract

For verifying systems involving a wide number or even an infinite number of states, standard model checking needs approximating techniques to be tractable. Abstract interpretation offers an appropriate framework to approximate models of reactive systems in order to obtain simpler models, where properties of interest can be effectively checked. In this work we study the impact of domain refinements in abstract interpretation based model checking. We consider the universal fragment of the branching time temporal logic CTL* and we characterize the structure of temporal formulae that are verified in new abstract models obtained by refining an abstract domain by means of reduced product and disjunctive completion, or by simplifying the domain by their inverse operations of complementation and least disjunctive bases.

1 Introduction

Model checking has emerged as a successful approach for automated verification of complex reactive systems where properties are typically expressed

[1] The work is partially supported by MURST project: *Certificazione automatica di programmi mediante interpretazione astratta.*

[2] Email:dovier@sci.univr.it

[3] Email:giaco@sci.univr.it

[4] Email:quintare@elet.polimi.it

using temporal logic [11,18] (for instance, to establish the validity of security properties of protocols). However, it is well known that verifying a temporal logic formula against a model, in particular finding all the system states that verify the formula, is in general a hard problem. Recall that in the case of finite states this problem is decidable both for CTL* and for the simpler case of CTL. The complexity of this problem is PSPACE complete for CTL* [11] and linear running time for CTL [1]. Model checking is usually applied to programs that consist of several concurrent processes; the number of states representing the whole program behaviour may grow exponentially in the number of such processes. This problem (known as *state explosion problem*) and the huge complexity for verifying temporal formulae against a model, especially for CTL*, are limiting factors that have to be tackled for any practical use of this technique.

Abstract interpretation is a general theory for approximating the semantics of discrete dynamic systems [4]. This theory offers an appropriate framework to approximate the model of a reactive system in order to obtain a simpler abstract model, over which the properties of interest can be checked for satisfaction. The idea here is that of verifying temporal properties in an abstract model which is systematically derived from the concrete semantics of the system we want to analyze, e.g., by abstracting the information contained in its states. Since the pioneering work on model checking and abstraction by Clarke et al. [2], a number of works have applied this idea to reduce the phenomenon of state explosion (e.g. [9]). However, Abstract Interpretation theory offers a number of methodologies that have not been applied yet in the field of abstract model checking. Many authors recognized in the possibility of modifying abstract models by modifying abstractions a great potential for improving abstract model checking in precision and reducing complexity (e.g., Section 9 in [9]), but few applications of these techniques are known in abstract model checking. On the contrary, this practice is quite common in static program analysis by abstract interpretation. A number of operations have been studied, both in theory and in practice, to compose, decompose, refine and compress abstract domains and analyses (see [12,14] for a survey), providing advanced algebraic methodologies and techniques for tuning analyses in accuracy and costs.

In this work we study the impact of standard domain refinement operations in abstract model checking. The problem is that when a chosen abstract domain turns out to provide a too rough abstract model for verifying a given temporal property of interest, this model can be refined by refining the corresponding abstract domain. Conversely, any operation acting on domains which is devoted to their simplification (decomposition or compression) can play the dual rôle of reducing the complexity of the verification of temporal formulae, provided that the formulae of interest are verified in both abstract and concrete models. In both these situations, the key problem is to study the structure of temporal formulae which are preserved or lost by changing the abstract domain by means of domain refinement or simplification, and in particular the structure of those formulae that are verified in the new model and which were not verified in the former. We consider the universal fragment of the branching time temporal logic CTL* [11] and we characterize the structure of temporal formulae that are verified in a new abstract model obtained either by refining an abstract domain by means of standard operations for domain transformation introduced in [6] (reduced product and disjunctive completion) or by simplifying the domain by means of their inverse operations (complementation for domain decomposition [3] or least disjunctive bases for domain compression [15]). In particular we prove that relevant properties of systems can be checked compositionally by decomposing the abstract models by domain complementation and that disjunctive information is in some cases redundant in abstract model checking of our CTL* fragment. This may provide sensible simplification algorithms for improving abstract model checking in complexity yet maintaining accuracy. We will describe an example of the application of our methods to demonstrate the practical impact of domain refinement operations in abstract model checking.

2 Preliminaries

Temporal Logic appears appropriate for describing the time-varying behaviour of reactive systems, e.g. *universal properties* (properties that have to hold along all executions of a program) and *existential properties* (properties that have to hold along some executions), as well as *safety properties* (nothing bad may happen) and *liveness properties* (something good has to happen) [18,19].

2.1 Temporal Logic and Model Checking

In this paper we consider the fragment known as ∀CTL* of the branching time temporal logic CTL* [2,11]: the formulae we deal with are the formulae of CTL* that do not use existential quantifiers. Of course, all the results apply to the universal fragment of the weaker language CTL, as well. In ∀CTL* universal properties are expressed through the path quantifier ∀ ("for all futures") that quantifies over (infinite) execution sequences. The temporal operators G (**G**enerally, always), F (**F**inally, sometime), X (ne**X**t time), and U (**U**ntil) express properties of a single execution sequence. Precisely, given a set *Prop* of propositions, the set *Lit* of *literals* is defined as $Lit = Prop \cup \{\neg q \mid q \in Prop\} \cup \{\mathsf{true}, \mathsf{false}\}$. *State formulae* ϕ and *Path formulae* ψ are inductively defined by the following grammar, where $p \in Lit$:

$$\text{state formulae:} \quad \phi ::= p \mid \phi \wedge \phi \mid \phi \vee \phi \mid \forall \psi$$

$$\text{path formulae:} \quad \psi ::= \phi \mid \psi \wedge \psi \mid \psi \vee \psi \mid G\psi \mid F\psi \mid X\psi \mid U(\psi, \psi)$$

A *transition system* is a pair $\langle \Sigma, R \rangle$ consisting of a set Σ of *states* and a

transition relation $R \subseteq \Sigma \times \Sigma$.
A *Kripke structure* is a tuple $\mathcal{K} = \langle \Sigma, R, I, \| \circ \| \rangle$ where $\langle \Sigma, R \rangle$ is a transition system, $I \subseteq \Sigma$ is the set of *initial states*, and $\| \circ \|$: *Lit* $\longrightarrow \wp(\Sigma)$ is the *interpretation function* such that $\| p \| = \{s \in \Sigma \mid s \models p\}$. For $\forall$CTL* the notion of *satisfaction* of a state formula ϕ by a state s ($s \models \phi$) is as usual in modal logic [9]. If $\mathcal{K} = \langle \Sigma, R, I, \| \circ \| \rangle$ is a Kripke structure, we say that $\mathcal{K} \models \varphi$ if and only if $\forall s \in I : s \models \varphi$. Given a temporal formula φ the satisfiability problem for φ is that of finding if there is a Kripke structure $\mathcal{K}$ such that $\mathcal{K} \models \varphi$. In the case of CTL* (hence of $\forall$CTL*) this problem is decidable [11]. For verification purposes, we are interested in the *(global) model checking problem (MCP)*: given $\mathcal{K} = \langle \Sigma, R, I, \| \circ \| \rangle$ and a formula φ, check if $\mathcal{K} \models \varphi$.

2.2 Abstract Interpretation

We assume basic notions of lattice theory [10]. The tuple $\langle C, \leq, \vee, \wedge, \top, \bot \rangle$ denotes a complete lattice C, with ordering $\leq$, *lub* $\vee$, *glb* $\wedge$, greatest element (top) $\top$, and least element (bottom) $\bot$ (i.e. C is a poset $(C, \leq)$ such that any subset X of C has a least upper bound $\vee X$ and a greatest lower bound $\wedge X$). The downward closure of $S \subseteq C$ is defined as $\downarrow S \stackrel{\text{def}}{=} \{x \in C \mid \exists y \in S.\ x \leq y\}$. $\downarrow x$ is a shorthand for $\downarrow \{x\}$, while the upward closure $\uparrow$ is dually defined. We consider here Galois insertion/connection based abstract interpretation [5]. If A and C are posets, and $\alpha : C \xrightarrow{\text{m}} A$ and $\gamma : A \xrightarrow{\text{m}} C$ are monotone functions such that $\forall x \in C.\ x \leq \gamma(\alpha(x))$ and $\forall x \in A.\ \alpha(\gamma(x)) \leq x$, then the quadruple (α, C, A, γ) is called a *Galois Connection* (GC for short) between C and A. The concrete and abstract domains, C and A, are assumed to be complete lattices and are related by abstraction and concretization maps forming a GC (α, C, A, γ). If in addition $\forall a \in A.\ \alpha(\gamma(a)) = a$, then we call (α, C, A, γ) a *Galois Insertion* (GI) of A in C. When (α, C, A, γ) is a GI then each value of the abstract domain A is useful in representing C, because all the elements of A represent distinct members of C, being γ 1-1. Any GC may be lifted to a GI identifying in an equivalence class those values of the abstract domain with the same concretization. This process is known as *reduction* of the abstract domain. Any abstract domain A in a GI (α, C, A, γ) is isomorphic to a subset of the concrete domain C which is a *Moore-family* of C, i.e. $X = \mathcal{M}(X) \stackrel{\text{def}}{=} \{\wedge S \mid S \subseteq X\}$ — where $\wedge \varnothing = \top \in \mathcal{M}(X)$. It turns out that in general an abstract domain A corresponds to a complete meet $(\wedge)$ subsemilattice of C, but, in general, it does not correspond to a complete sublattice of C, since the *lub* induced by A in C— namely $\gamma(\alpha(\vee Y))$ — might be different from that in C (i.e. $\vee Y$). Indeed, the two *lub's* coincide whenever $\gamma(\alpha(C))$ is a complete sublattice of C, which holds iff γ is additive. In this case we say that A is *disjunctive*. The lattice of all Moore families of C, also called the lattice of abstract interpretations of C [6], is denoted $\langle \mathcal{L}_C, \sqsubseteq, \sqcap, \sqcup, C, \{\top\} \rangle$, with C being the bottom abstract domain (most concrete abstraction) and

$\{\top\}$ being the top abstract domain (most abstract abstraction) of C. In this case $A \sqsubseteq B$ iff $B \subseteq A$ as Moore families of C.

3 Abstract Model Checking

We abstract the transition systems with simpler (in any case finite) transition systems, following the lines of Abstract Interpretation. Work in this direction can be found in [2,9,8]. A (concrete) Kripke structure $\mathcal{K} = \langle \Sigma, R, I, \| \circ \| \rangle$ is abstracted by a Kripke structure $\mathcal{K}_\alpha = \langle A, R_\alpha, I_\alpha, \| \circ \|_\alpha \rangle$, where:

- $(\alpha, \wp(\Sigma), A, \gamma)$ is a GI,
- A is the set of abstract states,
- $I_\alpha = \{\alpha(s) \mid s \in I\}$, and
- R_α is defined as follows: for all $a, b \in A$,

$$R_\alpha(a,b) \text{ iff } b \in \left\{ \ \alpha(Y) \ \middle| \ Y \in min \left\{ \ Y' \ \middle| \ R^{\exists\exists}(\gamma(a), Y') \ \right\} \ \right\}$$

where $R^{\exists\exists} \overset{\text{def}}{=} \left\{ \ (X,Y) \ \middle| \ \exists_{x \in X} \exists_{y \in Y} R(x,y) \ \right\}$.

Observe that each abstract value represents a *set* of concrete values.

We say that $p \in Lit$ is satisfied in an abstract state a whenever it is satisfied in all concrete states described by a: $\| \, p \, \|_\alpha \overset{\text{def}}{=} \{a \in A \mid \gamma(a) \subseteq \| \, p \, \|\}$.

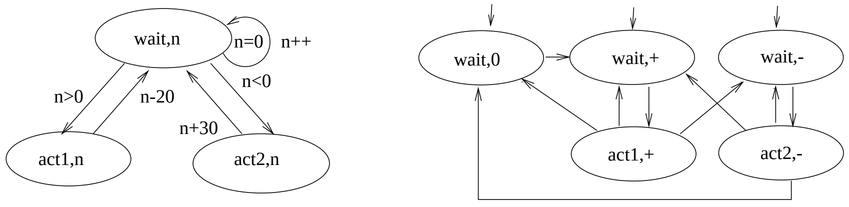

Fig. 1. The concrete and abstract Kripke structures $\mathcal{C}$ and $\mathcal{A}$

The concrete transition system $\mathcal{C}$ of Fig. 1 represents a process that performs the actions *wait*, act_1, and act_2, whose interleaving is regulated by inspecting the value of a variable n ranging in $\mathbb{Z}$. The set of states is the infinite set of pairs $\Sigma = \{wait, act_1, act_2\} \times \mathbb{Z}$, and $I = \{wait\} \times \mathbb{Z}$. Note that the labeled Kripke structure $\mathcal{C}$ represents an infinite transition system whose transitions are not labeled. Consider in a compact way the approximating Kripke structure $\mathcal{A}$ in Fig. 1 given by $A = \{wait, act_1, act_2\} \times \{\emptyset, -, 0, +, \mathbb{Z}\}$ and $I_\alpha = \{(wait, 0), (wait, -), (wait, +)\}$. In this way, we retain the basic actions of the concrete domain and we abstract the infinite part relating to integer numbers by using the domain *Sign* (see Fig. 2). In all the figures we draw only the accessible states from the initial ones in Kripke structures.

Let $\mathcal{K} = \langle \Sigma, R, I, \| \circ \| \rangle$ be a Kripke structure. A *path* in $\mathcal{K}$ is an infinite sequence $\pi = s_0, s_1, \cdots$ of states in Σ such that $s_0 \in I$ and for every $i \in \mathbb{N}$,

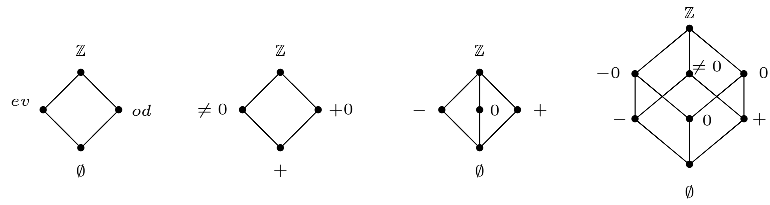

Fig. 2. The domains *Parity*, *Nneg*, *Sign*, and *Sign*$^\vee$.

$R(s_i, s_{i+1})$; π^i denotes s_i. We denote the set of paths of $\mathcal{K}$ by:

$$\Gamma_{\mathcal{K}} = \{\pi \mid \pi^0 \in I \wedge (\forall n \in \mathbb{N})(\pi^n \in \Sigma \wedge R(\pi^n, \pi^{n+1}))\}.$$

Definition 3.1 Given two Kripke structures $\mathcal{A} = \langle A, R_A, I_A, \| \circ \|_A \rangle$ and $\mathcal{B} = \langle B, R_B, I_B, \| \circ \|_B \rangle$, we say that $\mathcal{A}$ is *more precise* than $\mathcal{B}$ (denoted as $\mathcal{A} \preceq \mathcal{B}$) if $\forall \varphi \in$ CTL*. $\mathcal{B} \models \varphi \rightarrow \mathcal{A} \models \varphi$.

If $\mathcal{A} \preceq \mathcal{B}$ and $\mathcal{B} \preceq \mathcal{A}$, then we write $\mathcal{A} \equiv \mathcal{B}$. A first key result on the impact of abstraction on the class of formulae satisfied by a model was proved in [9]. This result, that holds in particular for $\varphi \in \forall$CTL*, justifies the intuitive observation that by abstracting a model we loose precision.

Definition 3.2 Let $\mathcal{C} = \langle \Sigma, R, I, \| \circ \| \rangle$ and $\mathcal{A} = \langle A, R_\alpha, I_\alpha, \| \circ \|_\alpha \rangle$ be the concrete and Abstract Kripke structures obtained by using a GI $(\alpha, \wp(\Sigma), A, \gamma)$. We say that $\mathcal{A}$ is an *abstraction* of $\mathcal{C}$.

Theorem 3.3 ([9]) *Let $\mathcal{C} = \langle \Sigma, R, I, \| \circ \| \rangle$ and $\mathcal{A} = \langle A, R_\alpha, I_\alpha, \| \circ \|_\alpha \rangle$ be a concrete and an Abstract Kripke structures. Then $\mathcal{C} \preceq \mathcal{A}$.*

It is well-known that GIs compose, namely if $(\alpha_1, C, A_1, \gamma_1)$, $(\alpha_2, A_1, A_2, \gamma_2)$ are GIs then $(\alpha_2 \circ \alpha_1, C, A_2, \gamma_1 \circ \gamma_2)$ is a GI. The same holds for abstraction of Kripke structures.

Theorem 3.4 *Let $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ be Kripke structures such that $\mathcal{B}$ is an abstraction of $\mathcal{A}$ and $\mathcal{C}$ is an abstraction of $\mathcal{B}$, then $\mathcal{C}$ is an abstraction of $\mathcal{A}$ and $\mathcal{A} \preceq \mathcal{C}$.*

4 Refining abstract models

In this section we consider the two basic operations of domain refinement introduced in [6]: reduced product and disjunctive completion. A domain refinement (see [14]) is any operation $\mathbb{R} : \mathcal{L}_C^n \longrightarrow \mathcal{L}_C$ such that for all domains $X_i \in \mathcal{L}_C$, $i = 1, \ldots, n$, $\mathbb{R}(X_1, \ldots, X_n) \sqsubseteq X_i$. It is immediate by Theorems 3.3 and 3.4 that if $\mathbb{R}(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ is the Kripke structure obtained by refining the domains in $\mathcal{A}_i$, then $\mathbb{R}(\mathcal{A}_1, \ldots, \mathcal{A}_n) \preceq \mathcal{A}_i$.

4.1 Reduced product model checking

The reduced product operation is basically obtained starting from the cardinal product; the set of pairs is then 'reduced' to obtain a Galois insertion. Recall that given a collection of domains $\{A_i\}_{i\in\Delta}$, all abstracting a given domain C by Galois insertions $(\alpha_i, C, A_i, \gamma_i)_{i\in\Delta}$, then $(\alpha_\sqcap, C, P, \gamma_\sqcap)$ is the reduced product of A_i's, denoted $P = \sqcap_{i\in\Delta} A_i$ if P is isomorphic to the subset of C: $\mathcal{M}(\bigcup_{i\in\Delta}\gamma_i(\alpha_i(C)))$ [6,7]. This operation corresponds to the *glb* operation $\sqcap$ in the lattice of abstract interpretations $\mathcal{L}_C$.

Suppose that a system $\mathcal{C}$ has been abstracted in n different ways, by using abstract interpretation. We assume that $\mathcal{A}_i = \langle A_i, R_{A_i}, I_{A_i}, \| \circ \|_{A_i}\rangle$ is an abstract Kripke structure, for each $i \in \{1, \ldots, n\}$. The following definition formalizes the Kripke structure that can be obtained by combining the abstract state spaces A_i by reduced product. A transition from a to b is allowed in the product structure if a and b are obtained by the meet of states allowing a transition in each component.

Definition 4.1 Let $\Delta = \{1, \ldots, n\}$ and $\forall i \in \Delta$, $\mathcal{A}_i = \langle A_i, R_{A_i}, I_{A_i}, \| \circ \|_{A_i}\rangle$ are abstractions of a Kripke structure $\mathcal{C} = \langle \Sigma, R, I, \| \circ \|\rangle$. The *Product Kripke Structure* is $\sqcap_{i\in\Delta}\mathcal{A}_i = (\sqcap_{i\in\Delta}A_i, R_\sqcap, \sqcap_{i\in\Delta}I_{A_i}, \| \circ \|_\sqcap)$, where $R_\sqcap(a, b)$ iff $(\forall i \in \Delta)((\exists a_i, b_i \in A_i)(R_{A_i}(a_i, b_i) \wedge \alpha_i(\gamma_\sqcap(a)) \leq a_i \wedge \alpha_i(\gamma_\sqcap(b)) \leq b_i))$. $a \in \| p \|_\sqcap$ iff $\gamma_\sqcap(a) \subseteq \| p \|$.

The following result specifies that the reduced product of domains A_i provides a more precise abstract model $\mathcal{A}_\sqcap$, where the conjunction of formulae which can be satisfied in some $\mathcal{A}_i$, can be verified.

Theorem 4.2 *Let $\Delta = \{1, \ldots, n\}$. Suppose that $\mathcal{C} = \langle \Sigma, R, I, \| \circ \|\rangle$ is a Kripke Structure, $\forall i \in \Delta, \mathcal{A}_i = \langle A_i, R_{A_i}, I_{A_i}, \| \circ \|_{A_i}\rangle$ are abstractions of $\mathcal{C}$, and $\mathcal{A}_\sqcap = \langle \sqcap_{i\in\Delta}A_i, R_\sqcap, I_\sqcap, \| \circ \|_\sqcap\rangle$ is the Product Kripke Structure. If $\forall i \in \Delta : \mathcal{A}_i \models \Phi_i$, then $\mathcal{A}_\sqcap \models \bigwedge_{i\in\Delta}\Phi_i$.*

Proof. By contradiction, suppose that $(\forall i \in \Delta)(\mathcal{A}_i \models \Phi_i)$ but $\mathcal{A}_\sqcap \not\models \bigwedge_{i=1}^{n}\Phi_i$, i.e. $(\exists j \in \Delta)(\mathcal{A}_\sqcap \not\models \Phi_j)$. By induction on the structure of the formula Φ_j:

(i) $\Phi_j = p, p \in \mathit{Lit}$. $\mathcal{A}_\sqcap \not\models p \longleftrightarrow (\exists i \in I_\sqcap)(i \not\models p) \longrightarrow \gamma_\sqcap(i) \not\subseteq \| p \|$. For definition of Product Kripke structure $i = i_1 \wedge \cdots \wedge i_j \wedge \cdots \wedge i_n \longrightarrow \gamma_\sqcap(i) = \gamma_\sqcap(i_1 \wedge \cdots \wedge i_j \wedge \cdots \wedge i_n) \subseteq \gamma_j(i_j) \longrightarrow \gamma_j(i_j) \not\subseteq \| p \| \longrightarrow i_j \notin \| p \|_{A_j}$. A contradiction.

(ii) $\Phi_j = Xp$. $\mathcal{A}_\sqcap \not\models \Phi_j \longleftrightarrow (\exists i \in I_\sqcap)(\exists s \in A_\sqcap)(R_\sqcap(i, s) \wedge s \not\models p)$. For definition of Product Kripke structure $\exists j \in \Delta$ such that $(\exists i_j \in I_{A_j})(\exists s_j \in A_j)(R_{A_j}(i_j, s_j) \wedge \alpha_j(\gamma_\sqcap(s)) \leq s_j)$. For definition of GC and Reduced Product $\gamma_\sqcap(s) \subseteq \gamma_j(\alpha_j(\gamma_\sqcap(s)))$. $\gamma_\sqcap(s) \not\subseteq \| p \|$ because $s \not\models p \longrightarrow \gamma_j(\alpha_j(\gamma_\sqcap(s))) \not\subseteq \| p \| \longrightarrow \alpha_j(\gamma_\sqcap(s)) \notin \| p \|_{A_j}$ by definition of $\| \circ \|_{A_j}$. $s_j \not\models p$ because $s_j \geq \alpha_j(\gamma_\sqcap(s))$. Thus, we obtain a contradiction.

(iii) $\Phi_j = \phi_1 \wedge \phi_2$, $\Phi_j = \phi_1 \vee \phi_2$, $\Phi_j = \psi_1 \wedge \psi_2$, $\Phi_j = \psi_1 \vee \psi_2$, $\Phi_j = X\psi$ or $\Phi_j = U(\psi_1, \psi_2)$, the proof is the same as the previous cases.

(iv) $\Phi_j = G\psi$. $\mathcal{A}_\sqcap \not\models \Phi_j \longrightarrow (\exists t \in \Gamma_{\mathcal{A}_\sqcap})((\exists n \in \mathbb{N})(t^n \not\models \psi))$:
- $n = 0 : (\exists i \in I_\sqcap)(i \not\models \psi) \longrightarrow (\exists i_j \in I_{A_j})(i_j \not\models \psi)$ (the proof is the same as the first case). We obtain a contradiction.

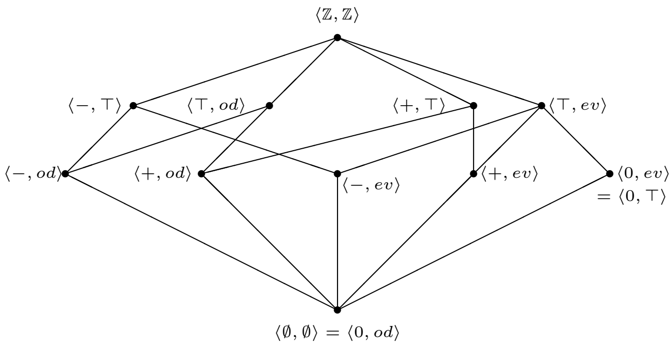
$\langle \mathbb{Z}, \mathbb{Z} \rangle$
$\langle -, \top \rangle$
$\langle \top, od \rangle$
$\langle +, \top \rangle$
$\langle \top, ev \rangle$
$\langle -, od \rangle$
$\langle +, od \rangle$
$\langle -, ev \rangle$
$\langle +, ev \rangle$
$\langle 0, ev \rangle$
$= \langle 0, \top \rangle$
$\langle \emptyset, \emptyset \rangle = \langle 0, od \rangle$

Fig. 3. The abstract domain $Sign \sqcap Parity$.

- $n > 0 : (\exists t \in \Gamma_{\mathcal{A}_\sqcap})((\exists n \in \mathbb{N})(t^n \not\models \psi)) \longrightarrow R(t^{n-1}, t^n) \wedge t^n \not\models \psi \longrightarrow (\exists j \in A)(\exists s_1, s_2 \in A_j)(R(s_1, s_2) \wedge s_2 \not\models \psi)$ (the proof is the same as the second case). We obtain a contradiction.

(v) $\Phi_j = \forall \psi$. $\mathcal{A}_\sqcap \not\models \Phi_j \longrightarrow (\exists t \in \Gamma_{\mathcal{A}_\sqcap})(t \not\models \psi)$. The argument is the same as the previous cases: we find a trace in the Kripke structure $\mathcal{A}_j$ which does not satisfy ψ, a contradiction.

□

Example 4.3 Consider the concrete Kripke structure $\mathcal{C}$ depicted in Fig. 4. It represents a process that performs the action:

- act_1, if the value of a variable n is greater than zero,
- act_2, if the value of the variable n is less than zero.

The value of the variable is modified by the process after the appropriate action is taken. The set of states is the infinite set of pairs $\Sigma = \{wait, act_1, act_2\} \times \mathbb{Z} \setminus \{0\}$, and $I = \{wait\} \times \mathbb{Z} \setminus \{0\}$. A possible approximation is the abstract Kripke Structure $\mathcal{A}_1$ whose set of abstract states is $A_1 = \{wait, act_1, act_2\} \times \{\emptyset, -, +, \mathbb{Z}\}$, and the set of initial states is $I_\alpha = \{(wait, -), (wait, +)\}$. Another approximation is the abstract Kripke Structure $\mathcal{A}_2$ whose set of abstract states is $A_2 = \{wait, act_1, act_2\} \times \{\emptyset, ev, od, \mathbb{Z}\}$, and the set of initial states is $I_\alpha = \{(wait, ev), (wait, od)\}$. The reduced product $Sign \sqcap Parity$ is represented in Fig. 3 and provides the abstract Kripke structure in Fig. 4, where only maximal nodes, corresponding to states with maximal value in the product domain, are depicted (the label "?" reported in the figure means $act_1 \vee act_2$). In this case it is easy to verify that $\mathcal{A}_1 \models \forall G(\neg n \geq 0 \vee Xn > 0)$, $\mathcal{A}_2 \models \forall G(\neg even(n) \vee XXodd(n))$ and $\mathcal{A}_\sqcap \models \forall G((\neg n \geq 0 \vee Xn > 0) \wedge (\neg even(n) \vee XXodd(n)))$ where $\mathcal{A}_\sqcap$ is the Product Kripke structure with domain $Sign \sqcap Parity$.

4.2 Disjunctive model checking

Disjunctive completion was originally introduced to model multiple branches in static program analysis [6,17]. The idea is that a domain is disjunctive if no loss of precision is accumulated by approximating the join operation (e.g.

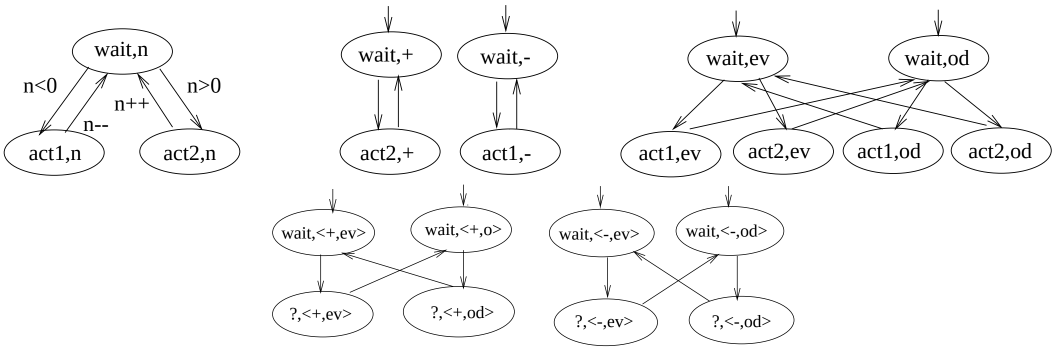

Fig. 4. Kripke structures $\mathcal{A}$, $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_\sqcap$

set-union) in the abstract domain.

Definition 4.4 The *disjunctive completion operator* $\Upsilon_C : \mathcal{L}_C \longrightarrow \mathcal{L}_C$ is defined for any $X \in \mathcal{L}_C$ as follows:

$$\Upsilon_C(X) = \sqcup\{A \in \mathcal{L}_C \mid A \sqsubseteq X \ \wedge \ A \text{ disjunctive}\}.$$

In this section we show that by refining the state space of a Kripke structure via disjunctive completion domain refinement we do not always obtain a more precise model. Recall that if C is a complete lattice, then $x \in C$ is *join-irreducible* if for any $y, z \in C$, if $x = y \vee z$ then $x = y$ or $x = z$ [10]. The set of join-irreducibles of C is denoted by $JI(C)$.

Theorem 4.5 *Let* $\mathcal{A} = \langle A, R_A, I_A, \| \circ \|_A\rangle$ *and* $\mathcal{B} = \langle \mathcal{Y}(A), R_{\Upsilon(A)}, I_{\Upsilon(A)}, \| \circ \|_{\Upsilon(A)}\rangle$ *be Kripke structures abstracting* $\mathcal{C} = \langle \Sigma, R, I, \| \circ \|\rangle$*. If* $I_A \subseteq JI(A)$ *and* $(\forall n \in \mathbb{N})(R_A(s^n, s^{n+1}) \to \{s^n, s^{n+1}\} \subseteq JI(A))$ *then* $\mathcal{A} \equiv \mathcal{B}$*.*

Proof. $JI(\Upsilon(A)) = JI(A)$ ($\Upsilon(A)$ may be different from A only in the information about the join operation between elements of $\wp(\Sigma)$) and thus the most precise abstraction of $\wp(\Sigma)$ in $\Upsilon(A)$ is the same as the abstraction of $\wp(\Sigma)$ in A. □

A characterization of those formulae that are not satisfied in non-disjunctive abstract models will be given later on in Section 5.2.

Example 4.6 By considering the example in Section 3 and by abstracting $\mathcal{C}$ with $Sign^\vee$ we obtain the model $\mathcal{A}$. On the contrary, if we consider the example 5.6 we note that the model computed with $Sign$ is less precise than $\mathcal{A}$ because the $\mathbb{Z}$ element is not join irreducible.

5 Compressing abstract models

In this section we study the impact of two operations of domain simplifications, namely *complementation* and *least disjunctive bases*, in abstract model checking. The idea here is that by reducing domains it is possible to reduce the complexity of temporal models and thus the verification of temporal formulae of interest. A domain simplification is any operation $\mathbb{S} : \mathcal{L}_C^n \longrightarrow \mathcal{L}_C$ such that for all domains $X_i \in \mathcal{L}_C$, $i = 1, \ldots, n$, $X_i \sqsubseteq \mathbb{S}(X_1, \ldots, X_n)$ [14]. It is immediate by Theorems 3.3 and 3.4 that if $\mathbb{S}(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ is the Kripke structure obtained by simplifying the domains A_i, then $\mathcal{A}_i \preceq \mathbb{S}(\mathcal{A}_1, \ldots, \mathcal{A}_n)$. A simplification is a compression if it returns the most abstract domain (when it exists) from which the original domain can be fully reconstructed back by a corresponding refinement: $\mathbb{S} : \mathcal{L}_C \longrightarrow \mathcal{L}_C$ is a compressor for a refinement $\mathbb{R} : \mathcal{L}_C \longrightarrow \mathcal{L}_C$ if $\mathbb{S}(\mathbb{R}(X)) = \mathbb{S}(X)$ and $\mathbb{R}(\mathbb{S}(X)) = \mathbb{R}(X)$ [14].

5.1 Complementing model checking

Complementation is important for domain decomposition, in fact it simplifies verification problems for complex domains, by decomposing them into simpler problems. Domain *complementation* is the inverse operation of reduced product, and corresponds to find, for any domains $A \sqsubseteq B$, the most abstract domain X such that $X \sqcap B = A$, i.e. it is the compressor of the domain refinement $\lambda X.\ X \sqcap B$. The problem of domain decomposition has been solved in [3] providing a systematic method for decomposing abstract domains into simpler factors. Recall that if C is a complete lattice, then $x \in C$ is *meet-irreducible* if for any $y, z \in C$, if $x = y \wedge z$ then $x = y$ or $x = z$. The set of meet-irreducibles of C is denoted by $MI(C)$. We say that C is generated by $MI(C)$ if $C = \mathcal{M}(MI(C))$. The following result provides a characterization of domain complementation in terms of meet-irreducible elements.

Theorem 5.1 ([13]) *Let C be a complete lattice generated by $MI(C)$, and let $(\alpha_A, C, A, \gamma_A)$ and $(\alpha_D, C, D, \gamma_D)$ be such that $A \sqsubseteq D$. Then*

$$A \sim D = \mathcal{M}(MI(A) \setminus D).$$

By applying complementation on the state space of a Kripke structure we obtain a simpler abstract structure which in general does not satisfy some temporal formulae of interest. We study under which conditions on the formulae we obtain a less precise abstract model by decomposing the abstract state space of a transition system.

The following result characterizes the predicates which are not preserved by complementing abstract Kripke structures. By a straightforward induction it is easy to characterize the structure of arbitrary temporal formulae that are not preserved by complementing abstract structures (see the example below).

Theorem 5.2 *Let $\mathcal{C} = \langle \Sigma, R, I, \| \circ \| \rangle$ and $\mathcal{A} = \langle A, R_A, I_A, \| \circ \|_A \rangle$ be a concrete and an abstract Kripke structure. Let $p \in Lit$. $\mathcal{A}^{\sim} = \langle A \sim B, R_{\sim}, I_{\sim}, \| \circ \|_{\sim} \rangle \not\models p$ iff*

$$(\exists s \in I_A)(s \models p \wedge (\forall x \subseteq \uparrow s \cap \mathcal{M}(MI(A) \setminus B))(x \not\models p) \wedge (\exists x \in \uparrow s)(x \not\models p)).$$

Proof. Let $\alpha_{\mathcal{A}} : \wp(\Sigma) \xrightarrow{\text{m}} A$, $\gamma_{\mathcal{A}} : A \xrightarrow{\text{m}} \wp(\Sigma)$, $\alpha_{\mathcal{A}^{\sim}} : \wp(\Sigma) \xrightarrow{\text{m}} A \sim B$, $\gamma_{\mathcal{A}^{\sim}} : A \sim B \xrightarrow{\text{m}} \wp(\Sigma)$ be the abstraction and concretization functions. Let $\hat{s} \in I_A$ be an initial state such that $\hat{s} \models p \wedge (\forall x \subseteq\uparrow \hat{s} \cap \mathcal{M}(MI(A) \setminus B))(x \not\models p) \wedge (\exists x \in\uparrow \hat{s})(x \not\models p))$, this means that $\hat{s} \not\in MI(A) \setminus B$. Let S be $\{s \in A \mid s \subseteq\uparrow \hat{s} \cap \mathcal{M}(MI(A) \setminus B)\}$. $\alpha_{\mathcal{A}^{\sim}}(\gamma_{\mathcal{A}}(\hat{s})) \in S \longrightarrow \alpha_{\mathcal{A}^{\sim}}(\gamma_{\mathcal{A}}(\hat{s})) \not\models p \longrightarrow$ the abstraction of the initial state $\hat{s}$ in $\mathcal{A}^{\sim}$ does not satisfy p, and therefore $\mathcal{A}^{\sim} \not\models p$. □

Example 5.3 Consider the concrete Kripke structure $\mathcal{C}$ in Fig. 5, with $\Sigma = \{wait, act\} \times \mathbb{N}$, and an approximated structure $\mathcal{A}$ with space states $A = \{wait, act\} \times Sign^{\vee}$. We observe that the variables cannot take negative values and thus it is possible to abstract $\mathcal{A}$ in a new structure which does not contain strictly negative information. Note that $Sign^{\vee} \sim \{\mathbb{Z}, -0, -\} = Nneg$ in Fig. 2. $Nneg$ induces an abstract structure $\mathcal{N}$ which abstracts $\mathcal{A}$, i.e. $\mathcal{A} \preceq \mathcal{N}$. It is now easy to verify that $\mathcal{A} \models (n = 0 \vee n > 0)$ (i.e. $\forall i \in I, i \models (n = 0 \vee n > 0)$) instead, $\mathcal{N} \not\models (n = 0 \vee n > 0)$ because the value 0 is abstracted in 0^{+}.

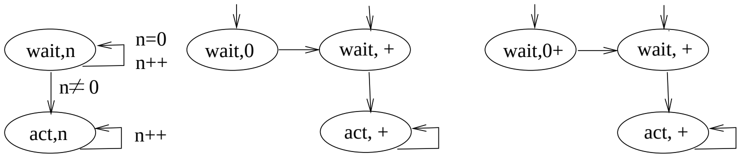

Fig. 5. Kripke structures $\mathcal{C}$, $\mathcal{A}$, and $\mathcal{N}$

5.2 Compressing model checking

The relevance of compression with respect to disjunction relies upon Theorem 4.5 above. In this case it is natural to state the following question: *Is it possible to minimize the disjunctive information in domains in such a way the abstract model be minimal with respect to this information?* In the following we consider the notion of *least disjunctive bases* introduced in [15]. This operation is well defined in most applications of abstract interpretation and returns the most abstract domain which induces, by disjunctive completion, a given disjunctive domain.

Definition 5.4 Given a complete lattice C, $X \in \mathcal{L}_C$ is *disjunctively optimizable* if $\Upsilon_C(\sqcup\{A \in \mathcal{L}_C \mid \Upsilon_C(A) = \Upsilon_C(X)\}) = \Upsilon_C(X)$.

If a domain $A \in \mathcal{L}_C$ is disjunctively optimizable then its *least disjunctive bases* exists and it is denoted by $\Omega_C(A)$ [15]. This is the case when C is a completely distributive lattice generated by its join irreducible elements, in particular when $C = \wp(\Sigma)$. In particular, let $\mathcal{C} = \langle \Sigma, R, I, \| \circ \| \rangle$ be a Kripke structure, then $\Omega_{\wp(\Sigma)}(A) = \mathcal{M}(JI(\Upsilon_{\wp(\Sigma)}(A)))$ for any $A \in \mathcal{L}_{\wp(\Sigma)}$ [15].

The following result characterizes precisely those predicates which are not preserved in the abstract Kripke structures obtained by the least disjunctive bases.

Theorem 5.5 *Let $\mathcal{C} = \langle \Sigma, R, I, \| \circ \| \rangle$ and $\mathcal{A} = \langle A, R_A, I_A, \| \circ \|_A \rangle$ be a concrete and an abstract Kripke structure. Let $p \in Lit$.*
$\Omega(A) = \langle \Omega(A), R_{\Omega(A)}, I_{\Omega(A)}, \| \circ \|_{\Omega(A)} \rangle \not\models p$ iff

$$(\exists s \in I_A)(s \models p \wedge (\forall x \subseteq\uparrow s \cap \mathcal{M}(JI(A)))(x \not\models p) \wedge (\exists x \in\uparrow s)(x \not\models p)).$$

Proof. Let $\alpha_{\mathcal{A}} : \wp(\Sigma) \xrightarrow{\text{m}} A$, $\gamma_{\mathcal{A}} : A \xrightarrow{\text{m}} \wp(\Sigma)$, $\alpha_{\Omega(A)} : \wp(\Sigma) \xrightarrow{\text{m}} \Omega(A)$, $\gamma_{\Omega(A)} : \Omega(A) \xrightarrow{\text{m}} \wp(\Sigma)$ be the abstraction and concretization functions. Let $\hat{s} \in I_A$ be an initial state such that $\hat{s} \models p \wedge (\forall x \subseteq\uparrow \hat{s} \cap \mathcal{M}(JI(A)))(x \not\models p) \wedge (\exists x \in\uparrow \hat{s})(x \not\models p))$, this means that $\hat{s} \notin JI(A)$. Let S be $\{s \in A \mid s \subseteq\uparrow \hat{s} \cap \mathcal{M}(JI(A))\}$. $\alpha_{\Omega(A)}(\gamma_{\mathcal{A}}(\hat{s})) \in S \longrightarrow \alpha_{\Omega(A)}(\gamma_{\mathcal{A}}(\hat{s})) \not\models p \longrightarrow$ the abstraction of the initial state $\hat{s}$ in $\Omega(A)$ does not satisfy $p \longrightarrow \Omega(A) \not\models p$. □

As before, those formulae which are not satisfied in the least disjunctive bases structure can be characterized by a straightforward inductive argument from the predicates not preserved as given in Theorem 5.5.

Example 5.6 Consider the concrete Kripke structure $\mathcal{C}$ in Fig. 6. The set of states is the infinite set of pairs $\Sigma = \{wait, act\} \times \mathbb{Z}$, and $I = \{wait\} \times \{0, \neq 0\}$. Actually, in $\mathcal{C}$ it is not important the integer value of n but the comparison of its value with zero. Consider an approximating Kripke structure $\mathcal{A}$ with domain $A = \{wait, act\} \times Sign^{\vee}$ and a further abstraction $\mathcal{B}$ with domain $B = \{wait, act\} \times Sign$. Note that the abstract Kripke structure $\mathcal{B}$ does not verify all the properties that hold in $\mathcal{A}$. For example, if $\varphi = \forall G(n \neq 0 \vee Xn \neq 0)$, $A \models \varphi$ while $\mathcal{B} \not\models \varphi$ because the value $\neq 0$ is abstracted in $\mathbb{Z}$.

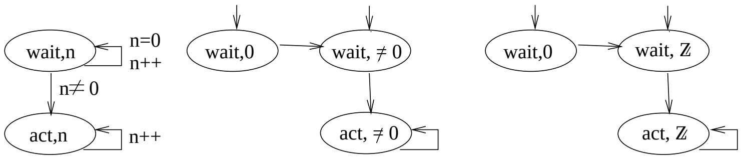

Fig. 6. Kripke structures $\mathcal{C}$, $\mathcal{A}$, and $\mathcal{B}$

6 An example

In this section we consider an example of the application of our methods to demonstrate the practical impact of domain refinement operations in abstract model checking.

The example is drawn from [2]: a concurrent algorithm for sorting an array of n cells containing integer numbers. Avoiding implementation details, the sorting algorithm works as follows: the n cells are numbered consecutively from right to left. The sort proceeds in cycles. During each cycle, exactly half the cells (either all of the odd-numbered cells or all of the even-numbered cells) will be compared with their right neighbour cell. If the value of a cell to be sorted is less than its right neighbour's value then the two values are swapped (for more details on the program see [2], where it is assumed that array cells contain only two values, zero and one). In Fig. 7 we show how the algorithm intuitively works when applied to an array of eight cells. The nodes with an entering arrow are the "active nodes" (i.e. the nodes that control the sort in the current cycle of execution of the program). Note that if in a cycle the odd-position nodes are active, during the next cycle the even-position nodes become active, and vice versa. The algorithm sorts the array in linear time.

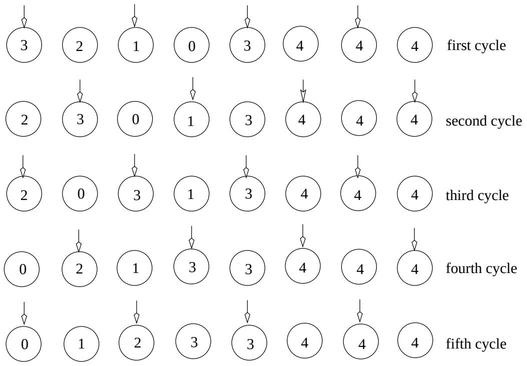

Fig. 7. An execution of the linear sorting array algorithm

This sorting algorithm can be formalized by using a Kripke structure over which it is possible to verify a temporal property that implies that the array is eventually sorted.

For example, in Fig. 8 we represent the Kripke structure for sorting two integer numbers in ascending order (here, as usual, entering arrows denote initial states). The system consists of two concurrent processes that cycle (mutually exclusively) through an infinite sequence of "active" (A) and "non-active" (NA) conditions. Each process swaps its cell with the right neighbour cell only if it is active and the values of the two cells are not in the right order. We want to verify that along every execution the following property holds: *eventually, the value of the first cell is less than or equal to the value of the second cell.*

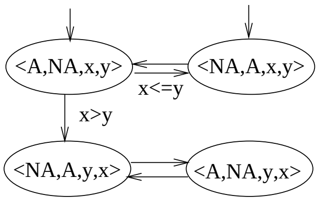

Fig. 8. Kripke structure for linear sorting array program

The state space Σ of this program is the set $\{A, NA\}^2 \times \mathbb{Z}^2$, where the first and the second value in a tuple of Σ are the conditions active or non-active of the two processes, while the last two values are the cells to be sorted. The initial states are $\{\langle A, NA, n, m\rangle, \langle NA, A, n, m\rangle \mid n, m \in \mathbb{Z}\}$. The transitions

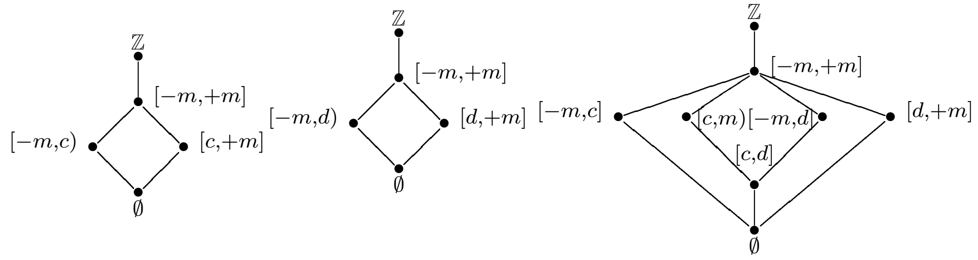

Fig. 9. Abstract domains Int_c, Int_d, and $Int_c \sqcap Int_d$

between two nodes are depicted in Fig. 8: observe that in each state only one process is active.

In order to abstract the Kripke structure in Fig. 8 with an abstract Kripke structure $\mathcal{A}_c = \langle \Sigma_c, R_c, I_c, \ell_c \rangle$ (see Fig. 10) we introduce an abstract domain, which is defined by providing abstractions of the components that form the concrete domain. We choose to leave the components $\{NA, A\}$ the same. Formally, this means that we take an abstract domain containing elements NA and A whose concretizations are $\{NA\}$ and $\{A\}$, respectively. To abstract the integer values to be sorted, we can compute a partitioning with respect to a parameter c (as in [2]). Given $c \in \mathbb{Z}$ we define a GI $(\alpha, \wp(\mathbb{Z}), Int_c, \gamma)$ between $\wp(\mathbb{Z})$ and the abstract domain Int_c in Fig. 9 ($m \in \mathbb{N}$ is a constant playing the rôle of *maxint*). The abstraction function is defined as follows: $\forall S \subseteq \wp(\mathbb{Z})$, $\alpha(S) = [-m, c)$ iff $\forall x \in S\ x < c$; $\alpha(S) = [c, +m]$ iff $\forall x \in S\ x \geq c$; $\alpha(S) = [-m, +m]$ otherwise.

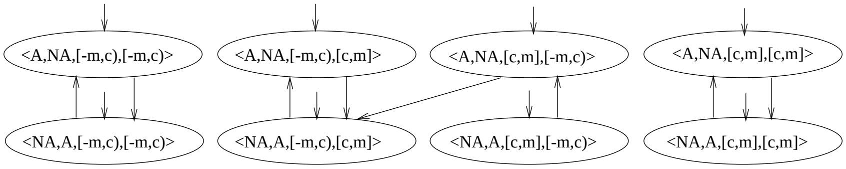

Fig. 10. Abstract Kripke structure $\mathcal{A}_c$ for linear sorting array program

The set Σ_c of abstract states is now defined as follows: $\Sigma_\alpha = \{\mathrm{NA}, \mathrm{A}, \top\}^2 \times Int_c^2$. Its top element is $\langle \top, \top, \mathbb{Z}, \mathbb{Z} \rangle$, while the approximation relation $\preceq$ is the extension of the orderings on each of the four components. It is important to note that the approximation order on the integer components (the Int_c domain in Fig. 9) does not correspond to the obvious order relation ($\leq$) used by the algorithm to sort the integer values (i.e. $[-m, c) \leq [c, +m]$ but $[-m, c) \not\preceq [c, +m]$). The set of abstract initial states is: $I_c = \{\langle A, NA, i_1, i_2 \rangle, \langle NA, A, i_1, i_2 \rangle \mid i_1, i_2 \in \{[-m, c), [c, +m]\}\}$. The abstract transition relations and the abstract interpretation of the concrete predicate '$\leq$' are computed accordingly to the definitions stated in Section 3.

It is easy to check that along every path of the abstract Kripke structure there is a continuation of the path that reaches only "sorted" states, i.e. if there is a value to be sorted that is less than c then it comes before the other value.

The temporal formula corresponding to the property is:

$$\varphi_c \equiv \forall F \, \forall G \, (x_1 < c \vee x_2 \geq c)$$

where x_1 and x_2 are the two values to be sorted. φ_c is tautologically equivalent to:

$$\forall F \forall G((x_1 < c \wedge x_2 < c) \vee x_2 \geq c)$$

Note that if the cells to be sorted by the linear sorting array algorithm are n then the number of abstract states is 2^{n+1}, while the number of concrete states is infinite or $2r^n$, considering a range r of integer number (e.g. $r \simeq 2maxint$). In order to refine the abstract sorting algorithm, it is possible to abstract the original Kripke structure by using two different abstract states spaces, i.e. by partitioning the integer numbers with respect to two different parameters (namely c and d). Consider for example the two domains Int_c and Int_d in Fig. 9 and suppose, without loss of generality, that $c < d$. By means of reduced product operation, we obtain the domain $Int_c \sqcap Int_d$ (see Fig. 9) which allow us to compute a more precise abstraction on integer values. In Fig. 11 it is reported a portion of the Product Kripke structure we obtain by taking into account the new domain $Int_c \sqcap Int_d$, and in particular by combining the state spaces $\Sigma_c = \{\text{NA,A}, \top\}^2 \times Int_c^2$ and $\Sigma_d = \{\text{NA,A}, \top\}^2 \times Int_d^2$ of two different abstract Kripke structures by means of reduced product ($\mathcal{A}_d = \langle \{\text{NA,A}, \top\}^2 \times Int_d^2, R_d, I_d, \ell_d \rangle$ is obtained in the same way as $\mathcal{A}_c$). We illustrate the case where the two values to be sorted may not be in the right order by partitioning the integer numbers with the parameter c, while they are in the correct order if we partition the numbers by using d. This model satisfies the temporal formula $\varphi_c \wedge \varphi_d = \forall F \forall G((x_1 < c \wedge x_2 < c) \vee x_2 \geq c) \wedge \forall F \forall G((x_1 < d \wedge x_2 < d) \vee x_2 \geq d)$, according to Theorem 4.2. Moreover, it satisfies the formula $\varphi_{c \times d} = \forall F \forall G((x_1 < c \wedge x_2 < c) \vee (x_1 < c \wedge c \leq x_2 \leq d) \vee (x_1 < c \wedge x_2 > d) \vee (c \leq x_1 < d \wedge x_2 > d) \vee x_2 > d)$, stating that the two values are ordered also with respect to the new partition of $\mathbb{Z}$.

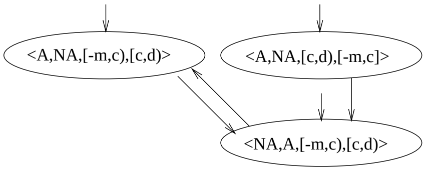

Fig. 11. A part of the Product Kripke structure $\mathcal{A}_c \sqcap \mathcal{A}_d$

Therefore, by using refinement operations, such as reduced product, it is possible to systematically refine abstract models, which approximate a given reactive system, and consequently obtain a new abstract reactive system that automatically verifies a combination (e.g the conjunction for the reduced product operation) of the formulae of interest. We conclude the subsection by giving some remarks on the complexity of the model checking problem that we have taken into account.

The algorithm for determining whether a CTL formula φ is true in the states of a Kripke structure $\langle S, R, P\rangle$ runs in time $O(length(\varphi) \cdot (\mid S \mid + \mid R \mid))$, where $length(\varphi)$ is the number of subformulae of φ (see [2]). Thus, since φ_c is a CTL formula and thus, the problem of checking it on the model $\mathcal{A}_c = \langle \Sigma_c, R_c, I_c, \ell_c\rangle$, has a linear complexity, more precisely the algorithm runs in time proportional to $length(\varphi_c) \cdot (\mid \Sigma_c \mid + \mid R_c \mid) \simeq 2^2 \cdot (2 \cdot 2^2 + 2 \cdot 2^2) = 4 \cdot 2^{2+2}$. Checking φ_d on $\mathcal{A}_d$ is the same problem.

In general, the length of a CTL formula φ which specifies that n integer values are eventually sorted, partitioning $\mathbb{Z}$ into k intervals, is $length(\varphi) \simeq k^n$. The size of the abstract Kripke structure for this particular configuration of the problem is $\mid \Sigma \mid + \mid R \mid \simeq 2k^n + 2k^n$ (the n integers to be sorted may assume k abstract values and in the linear sorting algorithm the "even-cells" or the "odd-cells" are alternatively considered. Moreover, each abstract state has an outgoing edge).

We have also demonstrated that by combining two different abstractions K_1 and K_2 of a given Kripke structure K, such that $K_1 \models \varphi_1$ and $K_2 \models \varphi_2$, we automatically obtain a new abstract Kripke structure $K_1 \sqcap K_2$ which verifies $\varphi_1 \wedge \varphi_2$ (we do not have to check it!). Since $K_1 \sqcap K_2$ is more precise than K_1 and K_2 we are interested in checking the satisfaction of new temporal formulae on its states. The size of $K_1 \sqcap K_2$ is less than or equal to $\mid K_1 \mid \cdot \mid K_2 \mid$, because of the reduction of abstract values with the same concretization (see [6]) (typically, the number is strictly lower: in the case of Fig. 9 we have 8 states instead of 25).

For the sorting problem we want to assure that the integers are eventually in the correct order with respect to the partition induced by the reduced product domain. In our example, we have to check on the model $\mathcal{A}_c \sqcap \mathcal{A}_d$ the satisfaction of the CTL formula $\varphi_{c\times d} = \forall F \forall G((x_1 < c \wedge x_2 < c) \vee (x_1 < c \wedge c \leq x_2 \leq d) \vee (x_1 < c \wedge x_2 > d) \vee (c \leq x_1 < d \wedge c \leq x_2 < d) \vee x_2 > d)$. In this case the model checking problem can be solved in time proportional to $length(\varphi_{c \cdot d}) \cdot (\mid \Sigma_c \mid + \mid R_c \mid) \cdot (\mid \Sigma_d \mid + \mid R_d \mid)$.

To sum up, consider an abstract domain $\Sigma_{c_1} \sqcap \cdots \sqcap \Sigma_{c_j}$, $j > 1$, that is the reduced product of j domains each of them partitions $\mathbb{Z}$ in 2 intervals. On the abstract Kripke structure with state space $\Sigma_{c_1} \sqcap \cdots \sqcap \Sigma_{c_j}$ we can check the satisfaction of the CTL formula $\varphi_{c_1 \times \cdots \times c_j}$ in time proportional to $4 \cdot (j+1)^{2n}$, where n is the number of cells to sort. Moreover, the Theorem 4.2 assures that the Kripke structure satisfies the formula $\varphi_{c_1} \wedge \cdots \wedge \varphi_{c_j}$. If we want to refine again the partition of the integer numbers we compute the abstract state space $\Sigma_{c_1} \sqcap \cdots \sqcap \Sigma_{c_j} \sqcap \Sigma_{c_{j+1}}$ of a new Product Kripke structure, which automatically satisfies the formula $\varphi_{c_1} \wedge \cdots \wedge \varphi_{c_j} \wedge \varphi_{c_{j+1}}$. The complexity of the model checking problem for the formula $\varphi_{c_1 \times \cdots \times c_j \times c_{j+1}}$ increases with a ratio proportional to $(1+\frac{1}{j})^{2n}$ and thus, at each step of the systematic refining process it is possible to check the satisfaction of a more refined formula with a small increment of the complexity.

Abs. Domain	Σ_{c_1}	$\Sigma_{c_1} \sqcap \Sigma_{c_2}$	$\cdots$	$\Sigma_{c_1} \sqcap \cdots \sqcap \Sigma_{c_j}$	$\Sigma_{c_1} \sqcap \cdots \sqcap \Sigma_{c_{j+1}}$
Formula	φ_{c_1}	$\varphi_{c_1} \wedge \varphi_{c_2}$	$\cdots$	$\varphi_{c_1} \wedge \cdots \wedge \varphi_{c_j}$	$\varphi_{c_1} \wedge \cdots \wedge \varphi_{c_{j+1}}$
Complexity	$4 \cdot 2^{2n}$	0	$\cdots$	0	0
Formula	φ_{c_1}	$\varphi_{c_1 \times c_2}$	$\cdots$	$\varphi_{c_1 \times \cdots \times c_j}$	$\varphi_{c_1 \times \cdots \times c_j \times c_{j+1}}$
Ratio	$4 \cdot 2^{2n}$	$(\frac{3}{2})^{2n}$	$\cdots$	$(\frac{j}{j-1})^{2n}$	$(1 + \frac{1}{j})^{2n}$

7 Future works

On the side of domain operations we plan to study the impact of Cousot's *reduced cardinal power* operation [6] and *Heyting completion* [16] for constructing relational abstract model checking. This operation, which does not admit a corresponding compressor, should upgrade domains, and therefore Kripke structures, with implicational information. On the side of Temporal Logic, we plan to generalize our results to arbitrary Temporal Logics. This can be achieved by considering more general μ-calculus, as in [8]. All these results should lead to the definition of a transformer of temporal formulae associated with each abstract domain transformer, and appropriate algorithms for simplifying or refining abstract model checking.

References

[1] Clarke, E. M., E. A. Emerson and A. P. Sistla, *Automatic verification of finite-state concurrent system using temporal logic specification*, ACM Transaction on Programming Languages and Systems **8** (1986), pp. 244–263.

[2] Clarke, E. M., O. Grumberg and D. E. Long, *Model checking and abstraction*, ACM Transaction on Programming Languages and Systems **16** (1994), pp. 1512–1542.

[3] Cortesi, A., G. Filé, R. Giacobazzi, C. Palamidessi and F. Ranzato, *Complementation in Abstract Interpretation*, toplas **19** (1997), pp. 7–47.

[4] Cousot, P., *Abstract Interpretation*, ACM Computing Surveys **28** (1996), pp. 324–328.

[5] Cousot, P. and R. Cousot, *Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints*, in: *Conference Record of the 4th ACM Symposioum on Principles of Programming Languages* (*POPL '77*) (1977), pp. 238–252.

[6] Cousot, P. and R. Cousot, *Systematic design of program analysis frameworks*, in: *Conference Record of the 6th ACM Symposioum on Principles of Programming Languages* (*POPL '79*) (1979), pp. 269–282.

[7] Cousot, P. and R. Cousot, *Inductive definitions, semantics and abstract interpretation*, in: *Conference Record of the 19th ACM Symposioum on Principles of Programming Languages* (*POPL '92*) (1992), pp. 83–94.

[8] Cousot, P. and R. Cousot, *Temporal abstract interpretation*, in: *Conference Record of the ACM Symposioum on Principles of Programming Languages* (*POPL '2000*) (2000), pp. 12–25.

[9] Dams, D., R. Gerth and O. Grumberg, *Abstract interpretation of reactive systems*, ACM Transaction on Programming Languages and Systems **19** (1997), pp. 253–291.

[10] Davey, B. A. and H. A. Priestley, "Introduction to Lattices and Order," Cambridge University Press, Cambridge, U.K., 1990.

[11] Emerson, E. A., *Temporal and modal logic*, in: J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, B: Formal Models and Semantics, Elsevier, Amsterdam and The MIT Press, Cambridge, Mass., 1990 pp. 997–1071.

[12] Filé, G., R. Giacobazzi and F. Ranzato, *A unifying view of abstract domain design*, ACM Computing Surveys **28** (1996), pp. 333–336.

[13] Filé, G. and F. Ranzato, *Complementation of abstract domains made easy*, in: M. Maher, editor, *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming* (*JICSLP '96*) (1996), pp. 348–362.

[14] Giacobazzi, R. and F. Ranzato, *Refining and compressing abstract domains*, in: P. Degano, R. Gorrieri and A. Marchetti-Spaccamela, editors, *Proceedings of the 24th International Colloqium on Automata, Languages and Programming* (*ICALP '97*), Lecture Notes in Computer Science **1256** (1997), pp. 771–781.

[15] Giacobazzi, R. and F. Ranzato, *Optimal domains for disjunctive abstract interpretation*, Science of Computing Programming **32** (1998), pp. 177–210.

[16] Giacobazzi, R. and F. Scozzari, *A logical model for relational abstract domains*, ACM Transaction on Programming Languages and Systems **20** (1998), pp. 1067–1109.

[17] Jensen, T., *Disjunctive strictness analysis*, in: *Proceedings of the 7th IEEE Symposium on Logic in Computer Science* (*LICS '92*) (1992), pp. 174–185.

[18] Manna, Z. and A. Pnueli, "The Temporal Logic of Reactive and Concurrent Systems," Springer-Verlag, Berlin, 1992.

[19] Müller-Olm, M., D. Schmidt and B. Steffen, *Model-checking. A tutorial introduction*, in: G. Filé, editor, *Proceedings of the International Static Analysis Symposium* (*SAS '99*), Lecture Notes in Computer Science **1694** (1999), pp. 330–354.