Shape Analysis by Ranking Abstraction

Amir Pnueli

New York University and Weizmann Institute of Sciences (Emeritus)

Symposium on Automatic Heap Anlysis AHA'07, Berlin, July 2007

Joint work with

Ittai Balaban, Yonit Kesten, Lenore Zuck

Ranking Abstraction

A method which combines predicate abstraction with augmentation of the system by progress monitors.

We restrict our attention to cases in which the abstractions can be computed automatically, using symbolic (BDD) representation.

AAV: Abstraction Aided Verification

An Obvious idea:

- Abstract system S into S_A – a simpler system, but admitting more behaviors.
- Verify property for the abstracted system S_A.
- Conclude that property holds for the concrete system.

Approach is particularly impressive when abstracting an infinite-state system into a finite-state one.

Technically, Define the methodology of Verification by Finitary Abstraction (VFA) as follows:

To prove $\mathcal{D} \models \psi$,

- Abstract $\mathcal{D}$ into a finite-state system $\mathcal{D}^\alpha$ and the specification ψ into a propositional LTL formula ψ^α.
- Model check $\mathcal{D}^\alpha \models \psi^\alpha$.

We look for instantiations of this general methodology which are sound and (relatively) complete.

Finitary Abstraction

Based on the notion of abstract interpretation [CC77].

Let Σ denote the set of states of an FDS $\mathcal{D}$ – the concrete states. Let $\alpha : \Sigma \mapsto \Sigma_A$ be a mapping of concrete into abstract states. α is finitary if Σ_A is finite.

We consider abstraction mappings which are presented by a set of equations $\alpha : (u_1 = E_1(V), \ldots, u_n = E_n(V))$ (or more compactly, $V_A = \mathcal{E}_\alpha(V)$), where $V_A = \{u_1, \ldots, u_n\}$ are the abstract state variables and V are the concrete variables.

Lifting a State Abstraction to Assertions

For an abstraction mapping $\alpha : V_A = \mathcal{E}_\alpha(V)$ and an assertion $p(V)$, we can lift the state abstraction α to abstract p:

- The expanding α-abstraction (over approximation) of p is given by

$$\overline{\alpha}(p):\quad \exists V : V_A = \mathcal{E}_\alpha(V) \ \wedge\ p(V) \qquad\qquad \|\overline{\alpha}(p)\| = \{\alpha(s) \mid s \in \|p\|\}$$

  An abstract state S belongs to $\|\overline{\alpha}(p)\|$ iff there exists some concrete state $s \in \alpha^{-1}(S)$ such that $s \in \|p\|$.

Sound Joint Abstraction

For a positive normal form temporal formula ψ, we define ψ^α to be the formula obtained by replacing every (maximal) state sub-formula $p \in \psi$ by $\underline{\alpha}(p) = \neg\overline{\alpha}(\neg p)$.

For an FDS $\mathcal{D} = \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$, we define the α-abstracted version $\mathcal{D}^\alpha = \langle V_A, \Theta^\alpha, \rho^\alpha, \mathcal{J}^\alpha, \mathcal{C}^\alpha \rangle$, where

$$\begin{array}{lcl} \Theta^\alpha & = & \overline{\alpha}(\Theta) \\ \rho^\alpha & = & \overline{\overline{\alpha}}(\rho) \\ \mathcal{J}^\alpha & = & \{\overline{\alpha}(J) \mid J \in \mathcal{J}\} \\ \mathcal{C}^\alpha & = & \{(\underline{\alpha}(p), \overline{\alpha}(q)) \mid (p, q) \in \mathcal{C}\} \end{array}$$

Soundness:

If α is an abstraction mapping and $\mathcal{D}$ and ψ are abstracted according to the recipes presented above, then

$$\mathcal{D}^\alpha \models \psi^\alpha \qquad \text{implies} \qquad \mathcal{D} \models \psi.$$

Example: Program INCREASE

Consider the program

$$\begin{array}{c} y : \textbf{integer initially } y = 0 \\ \left[\begin{array}{ll} \ell_0 : & \textbf{while } y \geq 0 \textbf{ do} \quad [\ell_1 : \ y := y + 1] \\ \ell_2 : & \end{array}\right] \end{array}$$

Assume we wish to verify the property $\diamondsuit \square (y > 0)$ for program INCREASE.

Introduce the abstract variable $Y : \{-1, 0, +1\}$.

The abstraction mapping α is specified by the defining expression:

$$\alpha : \quad [Y = \mathit{sign}(y)]$$

where $\mathit{sign}(y)$ is defined to be -1, 0, or 1, according to whether y is negative, zero, or positive, respectively.

The Abstracted Version

With the mapping α, we obtain the abstract version of INCREASE, called INCREASE$^{\alpha}$:

$$Y: \quad \{-1, 0, +1\} \text{ initially } Y = 0$$

$$\left[\begin{array}{l} \ell_0: \quad \textbf{while } Y \in \{0,1\} \textbf{ do} \quad \left[\ell_1: \; Y := \left(\begin{array}{ll} \textbf{if} & Y = -1 \\ \textbf{then} & \{-1, 0\} \\ \textbf{else} & +1 \end{array}\right) \right] \\ \ell_2: \end{array}\right]$$

The original invariance property ψ: $\diamondsuit \square \, (y > 0)$, is abstracted into:

$$\psi^{\alpha}: \quad \diamondsuit \square \, (Y = +1),$$

which can be model-checked over INCREASE$^{\alpha}$, yielding INCREASE$^{\alpha} \models \diamondsuit \square \, (Y = +1)$, from which we infer

$$\text{INCREASE} \models \diamondsuit \square \, (y > 0)$$

Predicate Abstraction

Let $p_1, p_2, \ldots, p_k$ be a set of assertions (state formulas) referring to the data (non-control) state variables. We refer to this set as the predicate base. Usually, we include in the base all the atomic formulas appearing within conditions in the program P and within the temporal formula ψ.

Following [GS97], define a predicate abstraction to be an abstraction mapping of the form

$$\alpha: \quad \{B_{p_1} = p_1, B_{p_2} = p_2, \ldots, B_{p_k} = p_k\}$$

where $B_{p_1}, B_{p_2}, \ldots, B_{p_k}$ is a set of abstract boolean variables, one corresponding to each assertion appearing in the predicate base.

Example: Program BAKERY-2

local y_1, y_2 : **natural initially** $y_1 = y_2 = 0$

```
P_1 ::  [ ℓ_0 : loop forever do
            [ ℓ_1 :  Non-Critical
              ℓ_2 :  $y_1 := y_2 + 1$
              ℓ_3 :  await $y_2 = 0 \lor y_1 < y_2$
              ℓ_4 :  Critical
              ℓ_5 :  $y_1 := 0$                    ] ]

      ||

P_2 ::  [ m_0 : loop forever do
            [ m_1 :  Non-Critical
              m_2 :  $y_2 := y_1 + 1$
              m_3 :  await $y_1 = 0 \lor y_2 \leq y_1$
              m_4 :  Critical
              m_5 :  $y_2 := 0$                    ] ]
```

The temporal properties for program BAKERY-2 are

ψexc : $\square\, \neg(at\_\ell_4 \land at\_m_4)$

ψacc : $\square\, (at\_\ell_2 \rightarrow \diamond\, at\_\ell_4)$,

Abstracting Program BAKERY-2

Define abstract variables $B_{y_1=0}$, $B_{y_2=0}$, and $B_{y_1<y_2}$.

```
local $B_{y_1=0}, B_{y_2=0}, B_{y_1<y_2}$: boolean
      where $B_{y_1=0} = B_{y_2=0} = 1, B_{y_1<y_2} = 0$

$P_1$ :: [ $\ell_0$ :loop forever do
          [ $\ell_1$ :  Non-Critical
            $\ell_2$ :  $(B_{y_1=0}, B_{y_1<y_2}) := (0, 0)$
            $\ell_3$ :  await $B_{y_2=0} \lor B_{y_1<y_2}$
            $\ell_4$ :  Critical
            $\ell_5$ :  $(B_{y_1=0}, B_{y_1<y_2}) := (1, \neg B_{y_2=0})$ ] ]

$\|$

$P_2$ :: [ $m_0$ :loop forever do
          [ $m_1$ :  Non-Critical
            $m_2$ :  $(B_{y_2=0}, B_{y_1<y_2}) := (0, 1)$
            $m_3$ :  await $B_{y_1=0} \lor \neg B_{y_1<y_2}$
            $m_4$ :  Critical
            $m_5$ :  $(B_{y_2=0}, B_{y_1<y_2}) := (1, 0)$ ] ]
```

The abstracted properties can now be model-checked.

The Question of Completeness

We have claimed above that the VFA method is sound. How about completeness?

Completeness means that, for every FDS $\mathcal{D}$ and temporal property ψ such that $\mathcal{D} \models \psi$, there exists a finitary abstraction mapping α such that $\mathcal{D}^\alpha \models \psi^\alpha$.

At this point we can only claim completeness for the special case that ψ is an invariance property.

Claim 1. [Completeness for Invariance Properties]
Let $\mathcal{D}$ be an FDS and $\psi : \Box\, p$ be an invariance property such that $\mathcal{D} \models \Box\, p$. Then there exists a finitary abstraction mapping α such that $\mathcal{D}^\alpha \models \Box\, \underline{\alpha}(p)$.

In fact, the proof shows that there always exists a predicate abstraction validating the invariance property.

Inadequacy of State Abstraction for Proving Liveness

Not all properties can be proven by pure finitary state abstraction.

Consider the program LOOP.

```
            $y$: natural
$\ell_0$ :  while $y > 0$ do
              [ $\ell_1$ :  $y := y - 1$ ]
              [ $\ell_2$ :  skip        ]
$\ell_3$ :
```

Termination of this program cannot be proven by pure finitary abstraction. For example, the abstraction $\alpha : \mathbb{N} \mapsto \{0, +1\}$ leads to the abstracted program

```
            $Y$: $\{0, +1\}$
$\ell_0$ :  while $Y = +1$ do
              [ $\ell_1$ :  $Y :=$ if $Y = +1$ then $\{0, +1\}$ else $0$ ]
              [ $\ell_2$ :  skip                                   ]
$\ell_3$ :
```

This abstracted program may diverge!

Solution: Augment with a Non-Constraining Progress Monitor

y: **natural**

$$\left[\begin{array}{l} \ell_0 : \textbf{while } y > 0 \textbf{ do} \\ \qquad \left[\begin{array}{l} \ell_1 : y := y - 1 \\ \ell_2 : \textbf{skip} \end{array}\right] \\ \ell_3 : \end{array}\right] \quad ||| \quad \left[\begin{array}{l} \quad \textit{dec} \quad : \{-1, 0, 1\} \\ \quad \textbf{compassion} \\ \qquad (\textit{dec} > 0, \textit{dec} < 0) \\ \textbf{always do} \\ m_0 : \ \textit{dec} := \textit{sign}(y - y') \end{array}\right]$$

— LOOP — — MONITOR M_y —

Forming the cross product, we obtain:

$$\boxed{\begin{array}{l} \qquad\qquad y \qquad : \textbf{natural} \\ \qquad\qquad \textit{dec} \quad : \{-1, 0, 1\} \\ \qquad\qquad \textbf{compassion } (\textit{dec} > 0, \textit{dec} < 0) \\ \ell_0 : \quad \textbf{while } y > 0 \textbf{ do} \\ \qquad \left[\begin{array}{llcl} \ell_1 : & (y, \textit{dec}) & := & (y - 1, \textit{sign}(y - y')) \\ \ell_2 : & \textit{dec} & := & \textit{sign}(y - y') \end{array}\right] \\ \ell_3 : \end{array}}$$

Abstracting the Augmented System

We obtain the program

```
        $Y$                 : $\{0, +1\}$
        $dec$               : $\{-1, 0, 1\}$
        compassion  $(dec > 0, dec < 0)$

$\ell_0$ :  while $Y = +1$ do
     [ $\ell_1$ : $(Y, dec)$ := $\begin{pmatrix} \textbf{if} & Y = +1 \\ \textbf{then} & (\{+1, 0\}, 1) \\ \textbf{else} & (0, 0) \end{pmatrix}$ ]
     [ $\ell_2$ :       $dec$ := $0$                               ]
$\ell_3$ :
```

Which always terminates.

Verification by Augmented Finitary Abstraction - The Ranking Abstraction Method

To verify that ψ is $\mathcal{D}$-valid,

- Optionally choose a non-constraining progress monitor FDS M and let $\mathcal{A} = \mathcal{D} \mid\mid\mid M$. In case this step is skipped, let $\mathcal{A} = \mathcal{D}$.
- Choose a finitary state abstraction mapping α and calculate $\mathcal{A}^\alpha$ and ψ^α according to the sound recipes.
- Model check $\mathcal{A}^\alpha \models \psi^a$.
- Infer $\mathcal{D} \models \psi$.

Claim 2. *The Ranking Abstraction method is complete, relative to deductive verification [KP00].*

That is, whenever there exists a deductive proof of $\mathcal{D} \models \psi$, we can find a finitary abstraction mapping α and a non-constraining progress monitor M, such that $\mathcal{A}^\alpha \models \psi^a$. Constructs α and M are derived from the deductive proof.

Computing the Abstraction within a Symbolic Model Checker

For some simple infinite domains, it is possible to decide satisfiability (validity) by boolean methods. For such domains, it is possible to compute and apply the abstraction, all within a single session of the model checker.

A Poor Man's Decision Procedure

Consider system variables:

N :	**natural**
$z_1, \ldots, z_n$:	$1..N$

An EA-assertion is a formula of the form $\varphi : \exists \vec{x} \forall \vec{y}.p(\vec{x}, \vec{y}, \vec{u})$, where p is a boolean combination of atomic formulas of the form $z_i < z_j$.

Small Model Theorem

Claim 3.
Assertion $\varphi = \exists \vec{x} \forall \vec{y}. p(\vec{x}, \vec{y}, \vec{u})$ *is satisfiable iff* φ *is satisfiable in a model of size* $\leq N_0 = |\vec{x}| + |\vec{u}|$.

Proof:
Assume φ is satisfiable in a model M_1 of size $N_1 > N_0$. We show how to construct a satisfying model M_2 of size $N_2 \leq N_0$.

Let $v_1 < v_2 < \cdots < v_k$ be all the distinct values assumed by $\vec{x}, \vec{u}$ in model M_1. Obviously $k \leq N_0$. We construct a model M_2 of size k.

For every $z \in \vec{x} \cup \vec{u}$, let

$$M_2[z] = j \qquad \text{iff} \qquad M_1[z] = v_j$$

We can show that $M_2 \models \varphi$ ∎

Truncation of Infinite-state Systems

Let P be a (potentially) infinite-state program. Denote by $\lfloor P \rfloor_N$ the N-truncated version of P obtained by restricting all integer variables to the subrange $-N..N$ (or $0..N$ for naturals) and all array bounds to N.

Let α be a finitary abstraction mapping and R a ranking augmentation which is a conjunction of expression of the form $\mathit{inc}' = \mathit{sign}(\delta' - \delta)$.

An abstraction problem (P, α, R) is called truncatable if there exists a natural $N > 0$ such that

$$\alpha(P \mid\mid\mid R) \quad \sim \quad \alpha(\lfloor P \rfloor_N \mid\mid\mid R)$$

Claim 4. *If all assertions occurring within α, R, and the* FDS *of P are EA-assertions, then (P, α, δ) is truncatable.*

The value of N is determined as he maximal $|\vec{x}| + |\vec{u}|$ over all abstraction formulas.

Abstracting Assertions and Transition relations

Let us recall the formulas expressing abstractions of assertions and transition relations. Assume that the abstraction mapping is given by $\alpha : V_A = \mathcal{E}_\alpha(V)$ and let $p(V)$ be an assertion over the concrete variables. The α-abstraction of p is given by:

$$\alpha(p)(V_A) : \quad \exists V \; (V_A = \mathcal{E}_\alpha(V) \;\wedge\; p(V))$$

Next, consider a transition relation $\rho(V, V')$. The α-abstraction of ρ is given by:

$$\alpha(\rho)(V_A, V'_A) : \qquad \exists V, V' : (V_A = \mathcal{E}_\alpha(V) \;\wedge\; V'_A = \mathcal{E}_\alpha(V') \;\wedge\; \rho(V, V'))$$

Shape Analysis by Ranking-Abstraction

Consider the REVERSE program:

$$\ell_0 : y := \textit{null}; \quad \ell_1 : \textbf{while } x \neq \textit{null} \textbf{ do } \ell_2 : (x, y, x.n) := (x.n, x, y); \quad \ell_3 :$$

We define the predicate *reach*(u, v) which means that there is a chain of *next*-links leading from the node pointed to by u to the node to which v points.

One of the properties we would like to prove is

$$\textit{at\_}\ell_0 \wedge t \neq \textit{null} \wedge \textit{reach}(x, t) \rightarrow \Box\, (\textit{at\_}\ell_3 \rightarrow \textit{reach}(y, t))$$

We therefore assume the initial condition $\Theta : t \neq \textit{null} \wedge \textit{reach}(x, t)$, and verify the invariance property

$$\Box\, (\textit{at\_}\ell_3 \rightarrow \textit{reach}(y, t))$$

As a predicate base we take the following predicates:

$$x = \textit{null}, \;\; t = \textit{null}, \;\; \textit{reach}(x, t), \;\; \textit{reach}(y, t)$$

and use the following abstraction mapping:

$$\textit{x\_null} = (x = \textit{null}), \quad \textit{t\_null} = (t = \textit{null}), \quad \textit{r\_xt} = \textit{reach}(x, t), \quad \textit{r\_yt} = \textit{reach}(y, t)$$

The Abstracted Program

This leads to the following abstract program:

```
$\mathit{x\_null}, \mathit{t\_null}, \mathit{r\_xt}, \mathit{r\_yt}$ : boolean where $\mathit{x\_null} = \mathit{t\_null} = 0,\ \mathit{r\_xt} = 1$
$\ell_0$ : $\mathit{r\_yt} := \mathit{t\_null}$
$\ell_1$ : while $\neg\mathit{x\_null}$ do
      $\ell_2$ : [ $(\mathit{r\_xt}, \mathit{r\_yt}) :=$ case
                       $\neg\mathit{r\_xt} \wedge \neg\mathit{r\_yt}$ : $(0,0)$
                       $\neg\mathit{r\_xt} \wedge \mathit{r\_yt}$ : $\{(0,1),(1,1)\}$
                       $1$ : $\{(0,1),(1,0),(1,1)\}$
                   esac
           $\mathit{x\_null} :=$ if $\mathit{r\_xt}$ then $0$ else $\{0,1\}$ ]
$\ell_3$ :
```

It is not difficult to verify (say by model checking) that

$$\Pi = 3 \ \Rightarrow\ \mathit{r\_yt}$$

Proving Progress Properties

Reconsider the REVERSE program:

$$\ell_0 : y := \textit{null}; \quad \ell_1 : \textbf{while } x \neq \textit{null} \textbf{ do } \ell_2 : (x, y, x.n) := (x.n, x, y); \quad \ell_3 :$$

A relevant progress property of this program is that of termination, which can be specified as

$$1 \Rightarrow \Diamond (\pi = 3)$$

A possible way of solving the problem is that of augmentation, composing the system with the following progress monitor:

$$\left[\begin{array}{l} \textit{inc} : \{-1, 0, 1\} \\ \textbf{compassion } (\textit{inc} < 0, \textit{inc} > 0) \\ \textbf{loop forever do} \\ \quad \textit{inc} := \textit{sign}(|\{i \mid \textit{reach}'(x, i)\}| - |\{j \mid \textit{reach}(x', j)\}|) \end{array}\right]$$

It is possible to show that there exists a small model theorem which allows us to compute the abstracted system by truncating the heap at size 7.

First Attempt at Verification

```
*** Property is NOT VALID ***
Counter-Example Follows:
---- State no. 1 =
AS.Pi = 0,    AS.xnull = 0,  AS.rxt = 1,   AS.ryt = 0,
AS.tnull = 0, AS.inc = -1,
---- State no. 2 =
AS.Pi = 1,    AS.xnull = 0,  AS.rxt = 1,   AS.ryt = 0,
AS.tnull = 0, AS.inc = 0,
---- State no. 3 =
AS.Pi = 2,    AS.xnull = 0,  AS.rxt = 1,   AS.ryt = 0,
AS.tnull = 0, AS.inc = 0,
---- State no. 4 =
AS.Pi = 1,    AS.xnull = 0,  AS.rxt = 0,   AS.ryt = 1,
AS.tnull = 0, AS.inc = -1,
---- State no. 5 =
AS.Pi = 2,    AS.xnull = 0,  AS.rxt = 0,   AS.ryt = 1,
AS.tnull = 0, AS.inc = 0,
```

Repeating Period

```
---- State no. 6 =
AS.Pi = 2,     AS.xnull = 0,  AS.rxt = 0,    AS.ryt = 1,
AS.tnull = 0, AS.inc = 0,
---- State no. 7 =
AS.Pi = 1,     AS.xnull = 0,  AS.rxt = 0,    AS.ryt = 1,
AS.tnull = 0, AS.inc = -1,
---- State no. 8 =
AS.Pi = 2,     AS.xnull = 0,  AS.rxt = 0,    AS.ryt = 1,
AS.tnull = 0, AS.inc = 0,
---- State no. 9 =
AS.Pi = 1,     AS.xnull = 0,  AS.rxt = 1,    AS.ryt = 1,
AS.tnull = 0, AS.inc = 1,
---- State no. 10 =
AS.Pi = 2,     AS.xnull = 0,  AS.rxt = 1,    AS.ryt = 1,
AS.tnull = 0, AS.inc = 0,
---- State no. 11 =
AS.Pi = 1,     AS.xnull = 0,  AS.rxt = 0,    AS.ryt = 1,
AS.tnull = 0, AS.inc = -1,
```

What Went Wrong?

According to the counter-example, it is possible to take a transition which causes the number of nodes reachable from x to increase. To see how this is possible, we performed the following:

```
>> Let t1:= _s[1].t[1];
>> Let st := x!=nil & t!=nil;
>> Print st & t1 & next(inc)=1;
pi = 2,1       x = 4,4        y = 2,4        t = 4,4
Next[1] = 0,0  Next[2] = 0,0  Next[3] = 0,0  Next[4] = 4,2
inc = ,1
```

This shows that the number of x-reachable nodes can increase if one of these nodes participate in a cycle.

To avoid this, we add the predicate

$$r\_xn = reach(x, null)$$

and added $reach(x, null)$ to the initial condition.

Now it ran successfully!!!

Automatic Refinement

An important element of finitary abstraction methods is the CEGAR methodology (Counter-Example Guided Refinement) in which, we use spurious counter examples in order to improve the abstraction.

In pure predicate abstraction, the CEGAR paradigm produces an improved abstraction α.

For Ranking Abstraction based on (α, Δ), an application of a CEGAR step may yield:

- A true counter example.
- An improved α.
- An enhanced ranking core $\widetilde{\Delta} = \Delta \cup \{\Delta_k\}$.

The paper A Modular Ranking Abstraction describes the application of this extended refinement process.

Characteristics of Our Approach to Shape Analysis

The main characteristics of our approach are:

- Computation of the abstract system is precise and is induced by lifting the state abstraction mapping α.
- Due to ranking abstraction, the method can verify arbitrary LTL properties, not only safety properties.
- We strongly rely on the reachability predicate *reach*(x, y).
- The method has its own version of the CEGAR paradigm.
- To achieve the above, we restrict our attention to shapes that have a decidable EA theory. In particular, theories possessing a small model property.

A Repertoire of Shapes to Which the Method is Applicable

So far, we successfully applied the method to the following classes of shapes:

- Structures in which every node has a single successor link.
- Single parent structures, including ordered and unordered trees.
- Some variants of cascaded single parent structures.

Conclusions and Future Research

- For cases to which the method is applicable, it is very effective. However, the set of these cases is restricted.
- Should consider a combination in which we may relax the requirement of precise complete abstraction and allow over-approximation together with precise abstraction.
- Find an appropriate SAT-based variant.