Ranking Abstraction as Companion to Predicate Abstraction

Amir Pnueli

New York University and Weizmann Institute of Sciences

Taipei, October 2005

Joint work with

Ittai Balaban, Yonit Kesten, Lenore Zuck

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 $\mathcal{D}^\alpha \models \psi^\alpha$.

The question considered here is whether we can find 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 $\Diamond\Box\,(y > 0)$ for program INCREASE.

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

The abstraction mapping α is specified by the defining expression:

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

where $\textit{sign}(y)$ 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$^\alpha$:

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

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

$$\psi^\alpha: \quad \Diamond\Box\,(Y = +1),$$

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

$$\text{INCREASE} \models \Diamond\Box\,(y > 0)$$

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** $y_1 = y_2 = 0$

$$P_1 :: \left[\begin{array}{l} \ell_0 : \textbf{loop forever do} \\ \left[\begin{array}{ll} \ell_1 : & \textbf{Non-Critical} \\ \ell_2 : & y_1 := y_2 + 1 \\ \ell_3 : & \textbf{await } y_2 = 0 \ \vee \ y_1 < y_2 \\ \ell_4 : & \textbf{Critical} \\ \ell_5 : & y_1 := 0 \end{array}\right] \end{array}\right]$$

$$\|$$

$$P_2 :: \left[\begin{array}{l} m_0 : \textbf{loop forever do} \\ \left[\begin{array}{ll} m_1 : & \textbf{Non-Critical} \\ m_2 : & y_2 := y_1 + 1 \\ m_3 : & \textbf{await } y_1 = 0 \ \vee \ y_2 \leq y_1 \\ m_4 : & \textbf{Critical} \\ m_5 : & y_2 := 0 \end{array}\right] \end{array}\right]$$

The temporal properties for program BAKERY-2 are

$$\begin{array}{lll} \psi_{exc} & : & \square \neg(at\_\ell_4 \ \wedge \ at\_m_4) \\ \psi_{acc} & : & \square \ (at\_\ell_2 \ \rightarrow \ \diamondsuit \ at\_\ell_4), \end{array}$$

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 $B_{y_1=0} = B_{y_2=0} = 1, B_{y_1<y_2} = 0$

$$P_1 :: \left[\begin{array}{l} \ell_0 : \textbf{loop forever do} \\ \left[\begin{array}{ll} \ell_1 : & \textbf{Non-Critical} \\ \ell_2 : & (B_{y_1=0}, B_{y_1<y_2}) := (0,0) \\ \ell_3 : & \textbf{await } B_{y_2=0} \ \vee \ B_{y_1<y_2} \\ \ell_4 : & \textbf{Critical} \\ \ell_5 : & (B_{y_1=0}, B_{y_1<y_2}) := (1, \neg B_{y_2=0}) \end{array}\right] \end{array}\right]$$

$$\|$$

$$P_2 :: \left[\begin{array}{l} m_0 : \textbf{loop forever do} \\ \left[\begin{array}{ll} m_1 : & \textbf{Non-Critical} \\ m_2 : & (B_{y_2=0}, B_{y_1<y_2}) := (0,1) \\ m_3 : & \textbf{await } B_{y_1=0} \ \vee \ \neg B_{y_1<y_2} \\ m_4 : & \textbf{Critical} \\ m_5 : & (B_{y_2=0}, B_{y_1<y_2}) := (1,0) \end{array}\right] \end{array}\right]$$

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 $\psi : \square\, p$ be an invariance property such that $\mathcal{D} \models \square\, p$. Then there exists a finitary abstraction mapping α such that $\mathcal{D}^\alpha \models \square\, \underline{\alpha}(p)$.

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.

$$\begin{array}{ll} & y\text{: } \textbf{natural} \\ \ell_0 : & \textbf{while } y > 0 \textbf{ do} \\ & \left[\begin{array}{ll} \ell_1 : & y := y - 1 \\ \ell_2 : & \textbf{skip} \end{array}\right] \\ \ell_3 : & \end{array}$$

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

$$\begin{array}{ll} & Y\text{: } \{0,+1\} \\ \ell_0 : & \textbf{while } Y = +1 \textbf{ do} \\ & \left[\begin{array}{ll} \ell_1 : & Y := \textbf{if } Y = +1 \textbf{ then } \{0,+1\} \textbf{ else } 0 \\ \ell_2 : & \textbf{skip} \end{array}\right] \\ \ell_3 : & \end{array}$$

This abstracted program may diverge!

Solution: Augment with a Non-Constraining Progress Monitor

y: **natural**

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

– LOOP – — MONITOR M_y –

Forming the cross product, we obtain:

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

Abstracting the Augmented System

We obtain the program

$$\begin{array}{ll} & Y \ : \{0, +1\} \\ & \textit{dec} \ : \{-1, 0, 1\} \\ & \textbf{compassion } (\textit{dec} > 0, \textit{dec} < 0) \\ \ell_0 : & \textbf{while } Y = +1 \textbf{ do} \\ & \left[\begin{array}{llll} \ell_1 : & (Y, \textit{dec}) & := & \left(\begin{array}{ll} \textbf{if} & Y = +1 \\ \textbf{then} & (\{+1, 0\}, 1) \\ \textbf{else} & (0, 0) \end{array}\right) \\ \ell_2 : & \textit{dec} & := & 0 \end{array}\right] \\ \ell_3 : & \end{array}$$

Which always terminates.

A More Complicated Case

Sometimes we need a more complex progress measure:

$$\begin{array}{ll} & y\text{: } \textbf{natural} \\ \ell_0 : & \textbf{while } y > 1 \textbf{ do} \\ & \left[\begin{array}{ll} \ell_1 : & y := y - 2 \\ \ell_2 : & y := \{y + 1, y\} \\ \ell_3 : & \textbf{skip} \end{array}\right] \\ \ell_4 : & \end{array}$$

To prove termination of this program we augment it by the monitor:

$$\begin{array}{ll} \textbf{define} & \delta = y + \textit{at\_}\ell_2 \\ \textit{dec} & : \{-1, 0, 1\} \\ \textbf{compassion} & (\textit{dec} > 0, \textit{dec} < 0) \\ m_0 : & \textbf{always do} \\ & \quad \textit{dec} := \textit{sign}(\delta - \delta') \end{array}$$

Complicated Case Continued

Augmenting and abstracting, we get:

$$\begin{array}{ll} & Y \ : \{0, \textit{one}, \textit{large}\} \\ & \textit{dec} \ : \{-1, 0, 1\} \\ & \textbf{compassion } (\textit{dec} > 0, \textit{dec} < 0) \\ \ell_0 : & \textbf{while } Y = \textit{large} \textbf{ do} \\ & \left[\begin{array}{llll} \ell_1 : & (Y, \textit{dec}) & := & (\textit{sub2}(Y), 1) \\ \ell_2 : & (Y, \textit{dec}) & := & \{(\textit{add1}(Y), 0), (Y, 1)\} \\ \ell_3 : & \textit{dec} & := & 0 \end{array}\right] \\ \ell_4 : & \end{array}$$

where,

$$\textit{sub2}(Y) = \textbf{if } Y \in \{0, \textit{one}\} \textbf{ then } 0 \textbf{ else } \{0, \textit{one}, \textit{large}\}$$

$$\textit{add1}(Y) = \textbf{if } Y = 0 \textbf{ then } \textit{one} \textbf{ else } \textit{large}$$

This program always terminates

Verification by Augmented Finitary Abstraction - The AFA Method

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

- Optionally choose a non-constraining progress monitor FDS M and let $\mathcal{A} = \mathcal{D} \ ||| \ M$. 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 AFA 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.

Can Abstraction Replace Deduction?

Yes, as shown by the completeness theorems.

Can Abstraction Replace Deduction?

Yes, as shown by the completeness theorems.

Yes, but why bother?

Based on the completeness theorems, it appears as though we first construct a deductive proof and then dress it up as abstraction.

Compare the efforts required for the application of the two methods:

Deduction	Abstraction
Provide inductive assertion strengthening candidate invariant. For liveness, provide ranking function.	Provide abstraction mapping. For liveness, provide augmenting monitor.
Establish validity of premises, using a theorem prover	Compute abstraction of system+property, using decision procedures.

The right question to ask is:

Should Abstraction Replace Deduction?

or, in other words,

What do we gain by such a replacement?

In other Namely, what is the value added by abstraction?

For the Case of Predicate Abstraction

A possible answer is:

It is often the case that the user can identify (or conjecture) the possible constituents of an inductive assertion, but does not know what is the precise boolean combination of these constituents which may form such an inductive assertion.

We leave it to the model checker to use BDD or SAT techniques in order to identify the best boolean combination.

Part of the Message of This Talk

In perfect analogy,

It is often the case that the user can identify (or conjecture) a set of possible constituents, but does not know how to combine them into an global ranking function.

We leave it to the model checker to form the correct combination (or prove the liveness property even without such explicit formation).

An Illustrative Example

Consider the following program NESTED-LOOPS:

$$
\begin{array}{l}
\qquad\qquad x, y\text{: } \textbf{natural} \\
\ell_0 : \ x := ? \\
\ell_1 : \ \textbf{while } x > 0 \textbf{ do} \\
\qquad \left[\begin{array}{l}
\ell_2 : \ y := ? \\
\ell_3 : \ \textbf{while } y > 0 \textbf{ do} \\
\qquad \left[\begin{array}{l}
\ell_4 : \ y := y - 1 \\
\ell_5 : \ \textbf{skip}
\end{array}\right] \\
\ell_6 : \ x := x - 1 \\
\ell_7 : \ \textbf{skip}
\end{array}\right] \\
\ell_8 :
\end{array}
$$

A deductive termination proof of this program may be based on the ranking function

$$(at\_\ell_0, \quad 5 \cdot x + 4 \cdot at\_\ell_7 + 3 \cdot at\_\ell_1 + 2 \cdot at\_\ell_2 + at\_\ell_{3..5}, \quad 3 \cdot y + 2 \cdot at\_\ell_5 + at\_\ell_3)$$

whose core constituents are x and y.

The Augmented-Abstraction Version

We augment the system with monitors for the ranking functions x, y, and abstract the domain of x, y into $\{0, +1\}$. This yields:

$$
\begin{array}{l}
\qquad \begin{array}{ll}
X, Y : & \{0, +1\} \\
\mathit{decx}, \mathit{decy} : & \{-1, 0, 1\} \\
\textbf{compassion} & (\mathit{decx} > 0, \mathit{decx} < 0), \quad (\mathit{decy} > 0, \mathit{decy} < 0)
\end{array} \\
\ell_0 : \ (X, Y, \mathit{decx}, \mathit{decy}) := (?, Y, ?, 0) \\
\ell_1 : \ \textbf{while } X = +1 \textbf{ do} \\
\quad \left[\begin{array}{l}
\ell_2 : \ (X, Y, \mathit{decx}, \mathit{decy}) := (X, ?, 0, ?) \\
\ell_3 : \ \textbf{while } Y = +1 \textbf{ do} \\
\quad \left[\begin{array}{l}
\ell_4 : \ (X, Y, \mathit{decx}, \mathit{decy}) := \left(\begin{array}{l}\textbf{if } Y = 0 \textbf{ then } (X, 0, 0, 0) \textbf{ else} \\ \{(X, +1, 0, 1), (X, 0, 0, 1)\}\end{array}\right) \\
\ell_5 : \ \mathit{decy} := 0
\end{array}\right] \\
\ell_6 : \ (X, Y, \mathit{decx}, \mathit{decy}) := \left(\begin{array}{l}\textbf{if } X = 0 \textbf{ then } (0, Y, 0, 0) \textbf{ else} \\ \{(+1, Y, 1, 0), (0, Y, 1, 0)\}\end{array}\right) \\
\ell_7 : \ \mathit{decx} := 0;
\end{array}\right] \\
\ell_8 :
\end{array}
$$

Model checking this program, we find that it always terminates.

Main Features of Predicate Abstraction

Can be used for the automatic verification of some LTL (all invariance) properties of infinite-state systems.

- Has a heuristic for an initial selection of a predicate base: Include all atomic formulas appearing in the program and property.
- Has a heuristic for refining the abstraction (expanding the predicate base), as a result of a spurious counter example.
- Does not require the specification of an inductive invariant. Sufficient to provide the constituents from which such an invariant can be constructed by a boolean combination.
- Can be used to derive the best inductive invariant expressible over the predicate base: Abstract, compute $Reach(P_A)$, and then concretize.

In Comparison, Ranking Abstraction

Can be used, in conjunction with predicate abstraction, for the automatic verification of all LTL properties (in particular, termination) of infinite-state systems.

- Has a heuristic for an initial selection of a ranking core: Include all variables and expressions which consistently increase (decrease) within loops. Specifically, loop indices.
- Has a heuristic for refining the predicate or ranking abstraction (expanding the predicate base or ranking core), as a result of a spurious counter example.
- Does not require the specification of a global ranking function. Sufficient to provide the constituents from which such a function can be constructed by a lexicographic tupling.
- Can be used to derive the best global ranking function expressible over the ranking core: Use recursive SCC's analysis.

A Counter-Example Guided Refinement of a Joint Abstraction

An abstract counter example of a liveness property has the form of a lasso:

$$\begin{tikzcd} S_0 \arrow[r] & \bullet\bullet\bullet \arrow[r] & S_k \arrow[r] & \bullet\bullet\bullet \arrow[r] & S_n \arrow[ll, bend right] \end{tikzcd}$$

As a first step, we attempt to concretize this sequence into a program trace

$\sigma: \quad s_0, \ldots, s_k, \ldots, s_n, s_{n+1}$

such that $S_i = \alpha(s_i)$, for $i \leq n$, and $S_k = \alpha(s_{n+1})$. There are three possible outcomes to this attempt:

1. We succeed to find a concretization such that $s_{n+1} = s_k$. In this case, there exists a concrete counter example and the property is invalid over the original system. In all other cases, the counter example is spurious.
2. The concretization is blocked at state s_i, $i \leq n$, such that s_i has no concrete successor belonging to $\alpha^{-1}(S_{i+1})$. In this case, apply regular predicate abstraction refinement (e.g. [BPR'02]).
3. The concretization completes, but $s_{n+1} \neq s_k$. In this case, apply ranking refinement. A loop has been concretized into a spiral.

Ranking Refinement

Recall the structure of the abstract counter example.

$$\begin{tikzcd} S_0 \arrow[r] & \bullet\bullet\bullet \arrow[r] & S_k \arrow[r] & \bullet\bullet\bullet \arrow[r] & S_n \arrow[ll, bend right] \end{tikzcd}$$

Assume that the labels of states $S_k, \ldots, S_n$ are $\ell_k, \ldots, \ell_n$. Form the (concrete) transition relation $\rho_{k..n,k}$ defined by

$\rho_{k..n,k}: \quad \rho(\ell_k, \ell_{k+1}) \circ \cdots \circ \rho(\ell_{n-1}, \ell_n) \circ \rho(\ell_n, \ell_k)$

This transition relation relates the values of variables in states s_k and s_{n+1} such that there exists a computation segment $s_k, \ldots, s_n, s_{n+1}$ passing through the sequence of labels $\ell_k, \ldots, \ell_n, \ell_k$, respectively.

Also form the assertion $\varphi_k = S_k[(p_1, \ldots, p_r)/(B_1, \ldots, B_r)]$ obtained by viewing abstract state S_k as a boolean expression over the abstract variables $B_1, \ldots, B_r$ and then substituting the predicate p_i for each occurrence of variable B_i. This assertion characterizes all the concrete states which are abstracted into S_k.

Expanding the Ranking Core

A sufficient condition which guarantees that the obtained lasso cannot be concretized into an infinite computation is that the relation $\rho_{k..n,k}$ be well founded over all φ_k-states. Hence we search for a variable or an expression δ, such that

$$\varphi_k \wedge \rho_{k..n,k} \quad \rightarrow \quad \delta > \delta'$$

Heuristics such as the ones expounded in [PR'04] can be used in order to identify such expressions δ.

Having found such a δ, we add it to the ranking core. Abstract and try again.

Example

Reconsider a version of program NESTED-LOOPS:

$$\begin{array}{ll}
 & x, y\text{: } \textbf{natural initially } x = y = 0 \\
\ell_0: & x := ? \\
 & \textbf{while } x > 0 \textbf{ do} \\
 & \left[\begin{array}{ll} \ell_1: & y := ? \\ & \textbf{while } y > 0 \textbf{ do} \\ & [\ \ell_2: \ y := y - 1\ ] \\ \ell_3: & x := x - 1 \end{array}\right] \\
\ell_4: &
\end{array}$$

Apply joint abstraction with $\{X = \mathit{sign}(x), Y = \mathit{sign}(y), \mathit{decy} = \mathit{sign}(y - y')\}$. Note that the ranking core is incomplete.

The Abstracted program

With the abstraction $\{X = \mathit{sign}(x), Y = \mathit{sign}(y), \mathit{decy} = \mathit{sign}(y - y')\}$, we obtain:

$$\begin{array}{ll}
 & \begin{array}{ll} X, Y: : & \{0, +1\} \textbf{ initially } X = Y = 0 \\ \mathit{decy}: & \{-1, 0, 1\} \textbf{ initially } \mathit{decy} = 0 \\ \textbf{compassion} & (\mathit{decy} > 0, \mathit{decy} < 0) \end{array} \\
\ell_0: & X := \{0, 1\} \\
 & \textbf{while } X = 1 \textbf{ do} \\
 & \left[\begin{array}{ll} \ell_1: & (Y, \mathit{decy}) := (?, ?) \\ & \textbf{while } Y = 1 \textbf{ do} \\ & [\ell_2: \ (Y, \mathit{decy}) := \textbf{if } Y = 0 \textbf{ then } (0, 0) \textbf{ else } \{(0, 1), (1, 1)\}] \\ \ell_3: & X := \textbf{if } X = 0 \textbf{ then } 0 \textbf{ else } \{0, 1\} \end{array}\right] \\
\ell_4: &
\end{array}$$

Model checking this program for termination, we obtain the following counter-example lasso:

$S_0 : \langle \Pi : \ell_0, X : 0, Y : 0, \mathit{Decy} : 0 \rangle,$
$S_1 : \langle \Pi : \ell_1, X : 1, Y : 0, \mathit{Decy} : 0 \rangle, \quad S_2 : \langle \Pi : \ell_2, X : 1, Y : 1, \mathit{Decy} : -1 \rangle,$
$S_3 : \langle \Pi : \ell_3, X : 1, Y : 0, \mathit{Decy} : 1 \rangle, \quad S_4 = S_1$

Concretizing and Refining

Concretizing the abstract trace

$S_0 : \langle \Pi : \ell_0, X : 0, Y : 0, \mathit{Decy} : 0 \rangle,$
$S_1 : \langle \Pi : \ell_1, X : 1, Y : 0, \mathit{Decy} : 0 \rangle, \quad S_2 : \langle \Pi : \ell_2, X : 1, Y : 1, \mathit{Decy} : -1 \rangle,$
$S_3 : \langle \Pi : \ell_3, X : 1, Y : 0, \mathit{Decy} : 1 \rangle, \quad S_4 = S_1$

we obtain:

$s_0 : \langle \pi : \ell_0, x : 0, y : 0, \mathit{decy} : 0 \rangle,$
$s_1 : \langle \pi : \ell_1, x : 4, y : 0, \mathit{decy} : 0 \rangle, \quad s_2 : \langle \pi : \ell_2, x : 4, y : 1, \mathit{decy} : -1 \rangle,$
$s_3 : \langle \pi : \ell_3, x : 4, y : 0, \mathit{decy} : 1 \rangle, \quad s_4 : \langle \pi : \ell_1, x : 3, y : 0, \mathit{decy} : 0 \rangle$

We therefore compute $\varphi_1 : x > 0 \wedge y = 0$ and $\rho_{1..3,1} : x' = x - 1 \wedge x' > 0$. A natural choice for additional rank is $\delta = x$ whose descent is implied by $\rho_{1..3,1}$.

A Global Ranking Function From a Terminating Program

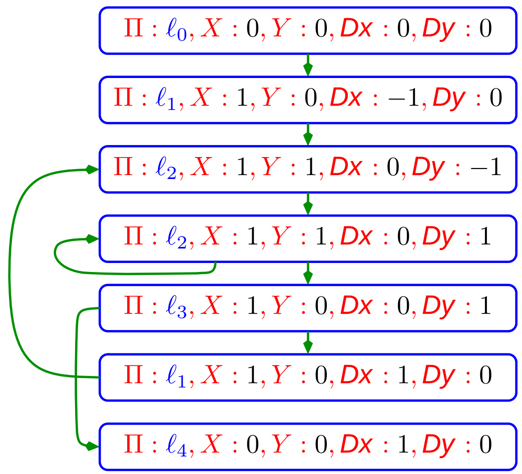
We will show how to extract a global ranking function from an abstract terminating program. Assume that we constructed a state-transition graph containing all the reachable states of the abstracted program.

The extraction algorithm can be described as follows:

- Decompose into MSCC's, Sort topologically, and Rank sequentially.
- For each non-singular component:
  - Identify a compassion req. $(\mathit{decx}_i > 0, \mathit{decx}_i < 0)$ violated by the component.
  - Add x_i to the ranking tuple.
  - Remove all edges entering $(\mathit{decx}_i > 0)$-nodes.
  - Return to top for recursive processing of remaining subgraph.

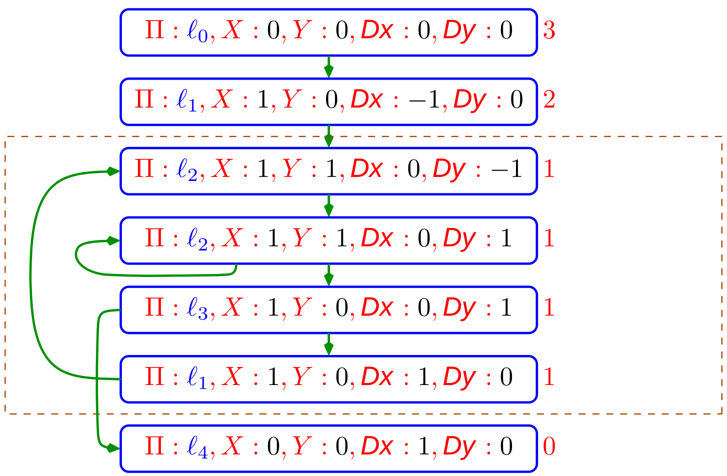
Example

Analyzing abstracted program NESTED-LOOPS with ranking core consisting of $\{x, y\}$, the program always terminates. The resulting state transition graph is:



Decompose, Sort, and Rank

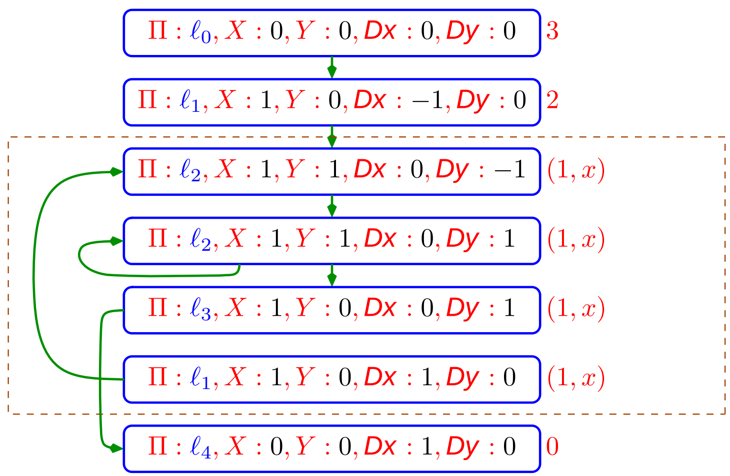
MSCC's decomposition, topologically sorting, and sequentially ranking, yields:



Non-singular component is unfair w.r.t $(\mathit{Dx} > 0, \mathit{Dx} < 0)$.

Add x to Ranking

Add x to ranking, and remove edges entering $(\mathit{Dx} > 0)$-nodes.



Note that component is no longer strongly connected.

Decompose, Sort, and Rank Subgraph

Applying the decomposition+ranking to the unraveled subgraph yields:

$\Pi:\ell_0, X:0, Y:0, \mathit{Dx}:0, \mathit{Dy}:0$	3
$\Pi:\ell_1, X:1, Y:0, \mathit{Dx}:-1, \mathit{Dy}:0$	2
$\Pi:\ell_2, X:1, Y:1, \mathit{Dx}:0, \mathit{Dy}:-1$	$(1,x,2)$
$\Pi:\ell_2, X:1, Y:1, \mathit{Dx}:0, \mathit{Dy}:1$	$(1,x,1)$
$\Pi:\ell_3, X:1, Y:0, \mathit{Dx}:0, \mathit{Dy}:1$	$(1,x,0)$
$\Pi:\ell_1, X:1, Y:0, \mathit{Dx}:1, \mathit{Dy}:0$	$(1,x,3)$
$\Pi:\ell_4, X:0, Y:0, \mathit{Dx}:1, \mathit{Dy}:0$	0

Note that the non-singular component is unfair w.r.t $(\mathit{Dy} > 0, \mathit{Dy} < 0)$.

Add y to the Ranking

Processing the $\langle \Pi:\ell_2, X:1, Y:1, \mathit{Dx}:0, \mathit{Dy}:1\rangle$ component, we add y to its ranking and remove all incoming edges. This yields:

$\Pi:\ell_0, X:0, Y:0, \mathit{Dx}:0, \mathit{Dy}:0$	3
$\Pi:\ell_1, X:1, Y:0, \mathit{Dx}:-1, \mathit{Dy}:0$	2
$\Pi:\ell_2, X:1, Y:1, \mathit{Dx}:0, \mathit{Dy}:-1$	$(1,x,2)$
$\Pi:\ell_2, X:1, Y:1, \mathit{Dx}:0, \mathit{Dy}:1$	$(1,x,1,y)$
$\Pi:\ell_3, X:1, Y:0, \mathit{Dx}:0, \mathit{Dy}:1$	$(1,x,0)$
$\Pi:\ell_1, X:1, Y:0, \mathit{Dx}:1, \mathit{Dy}:0$	$(1,x,3)$
$\Pi:\ell_4, X:0, Y:0, \mathit{Dx}:1, \mathit{Dy}:0$	0

The resulting graph is acyclic, implying that the algorithm terminated.

The Final Global Ranking

Summarizing all that was accumulated, yields the following global ranking:

$\Pi:\ell_0, X:0, Y:0, \mathit{Dx}:0, \mathit{Dy}:0$	3
$\Pi:\ell_1, X:1, Y:0, \mathit{Dx}:-1, \mathit{Dy}:0$	2
$\Pi:\ell_2, X:1, Y:1, \mathit{Dx}:0, \mathit{Dy}:-1$	$(1,x,2)$
$\Pi:\ell_2, X:1, Y:1, \mathit{Dx}:0, \mathit{Dy}:1$	$(1,x,1,y)$
$\Pi:\ell_3, X:1, Y:0, \mathit{Dx}:0, \mathit{Dy}:1$	$(1,x,0)$
$\Pi:\ell_1, X:1, Y:0, \mathit{Dx}:1, \mathit{Dy}:0$	$(1,x,3)$
$\Pi:\ell_4, X:0, Y:0, \mathit{Dx}:1, \mathit{Dy}:0$	0

Padding to the Right

If necessary, we can make all tuples to be of length 4, by adding zeros to the right.

$\Pi:\ell_0, X:0, Y:0, \mathit{Dx}:0, \mathit{Dy}:0$	$(3,0,0,0)$
$\Pi:\ell_1, X:1, Y:0, \mathit{Dx}:-1, \mathit{Dy}:0$	$(2,0,0,0)$
$\Pi:\ell_2, X:1, Y:1, \mathit{Dx}:0, \mathit{Dy}:-1$	$(1,x,2,0)$
$\Pi:\ell_2, X:1, Y:1, \mathit{Dx}:0, \mathit{Dy}:1$	$(1,x,1,y)$
$\Pi:\ell_3, X:1, Y:0, \mathit{Dx}:0, \mathit{Dy}:1$	$(1,x,0,0)$
$\Pi:\ell_1, X:1, Y:0, \mathit{Dx}:1, \mathit{Dy}:0$	$(1,x,3,0)$
$\Pi:\ell_4, X:0, Y:0, \mathit{Dx}:1, \mathit{Dy}:0$	$(0,0,0,0)$

Conclusions

- Ranking abstraction should be considered as an inseparable companion to predicate abstraction. Only their combination can verify the full set of LTL properties.
- We call upon implementors of abstraction-based software verification systems, such as SLAM and BLAST, to enhance the proving power of their systems by adding the component of ranking abstraction.
- Like predicate abstraction, ranking abstraction is easier to apply than its deductive counterpart, because it is sufficient to provide only the constituents and let the model checker figure out their right combination.
- We should not consider abstraction as replacing deduction, but rather as complementing and enhancing deduction.
- Never pay too much attention to completeness theorems. They may provide a misleading view of the usefulness of a method.