Fixpoint-Guided Abstraction Refinements

Patrick Cousot, Pierre Ganty, and Jean-François Raskin, presented by Samuel Pastva

Terminology recap: Transition systems

Transition system $\mathcal{T} = (S, I, T)$ *(states, initial states, transition relation)*

Forward and backward concrete semantics:

Existential:

$$post(X) = \{s' \in S \mid \exists s.(s, s') \in T \wedge s \in X\}$$
$$pre(X) = \{s \in S \mid \exists s'.(s, s') \in T \wedge s' \in X\}$$

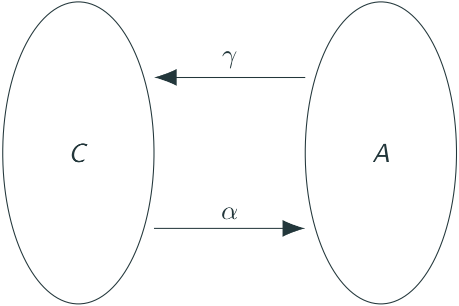
Universal:

$$\widetilde{post}(X) = \{s' \in S \mid \forall s.(s, s') \in T \Rightarrow s \in X\}$$
$$\widetilde{pre}(X) = \{s \in S \mid \forall s'.(s, s') \in T \Rightarrow s' \in X\}$$

Terminology recap: Abstraction

Concrete domain (C) | Galois Connection | Abstract domain (A)

$\langle 2^S, \subseteq, \cap, \cup, S, \emptyset \rangle$ | | $\langle A, \sqsubseteq, \sqcap, \sqcup, \top_A, \bot_A \rangle$



Terminology recap: Abstraction

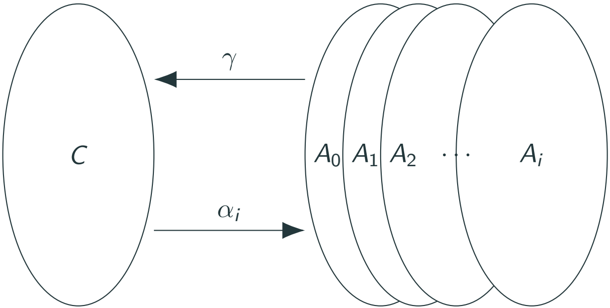
Concrete domain (C)

Galois Connection

Abstract domain family (A_i)

$\langle 2^S, \subseteq, \cap, \cup, S, \emptyset \rangle$

$\langle A_i, \sqsubseteq, \sqcap, \sqcup, \top_{A_i}, \bot_{A_i} \rangle$



Abstraction

We consider two fixpoint functions on A_i:

$\mathcal{R}_i(I, R)$ — states in A_i reachable from I inside R.

$$\mathcal{R}_i(I, R) = \mathit{lfp}(\lambda X.\alpha_i(R \cap (I \cup \mathit{post}(\gamma(X)))))$$

$\mathcal{S}_i(S)$ — states in A_i which are stuck in (cannot escape) S.

$$\mathcal{S}_i(S) = \mathit{gfp}(\lambda X.\alpha_i(S \cap \widetilde{\mathit{pre}}(\gamma(X))))$$

For a correct A_i, both $\mathcal{R}_i$ and $\mathcal{S}_i$ are over-approximations of their exact counterparts ($\mathcal{S}_i$ works because $\cap$ is defined on the concrete domain).

Fixpoint checking problem and abstraction

Given a transition system $\mathcal{T} = (S, I, T)$ **and a state set** Z **decide if** $lfp(\lambda X.(I \cup post(X))) \subseteq Z$.

In other words — decide if I cannot escape Z.

Boolean and Moore closed domains

We assume C is Boolean closed and A_i are finite and Moore closed:

Moore closed set M: closed under **intersection**. (meet on lattices)

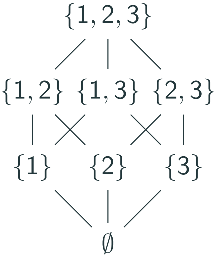
- Moore closure operator $\mathcal{M}(X)$.

Boolean closed set B: closed under **intersection, union and complement**. (meet/join/??? on lattices)

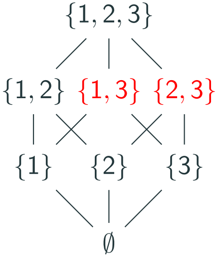
- Boolean closure operator $\mathcal{B}(X)$.

Predicate abstraction produces Boolean closed domains. Moore domains are less restrictive.

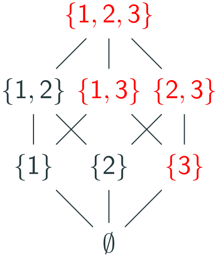
Domain 2^S is obviously Boolean (and Moore) closed.



Is the red domain Moore closed?



Is the red domain Boolean closed?



Algorithm

$Z_0 \leftarrow Z$;
for $i = 0\ 1\ 2\ 3\ ...$ **do**
    $R_i = \mathcal{R}_i(I, Z_i)$;
    if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
        return OK;
    else
        $S_i = \mathcal{S}_i(R_i)$;
        if $\alpha_i(I) \sqsubseteq S_i$ **then**
            $Z_{i+1} =$ **(?) refined** S_i;
            $A_{i+1} =$ **(?) refined** A_i;
        else
            return NOK;
        end
    end
end

Algorithm

$Z_0 \leftarrow Z$;

for $i = 0\ 1\ 2\ 3\ \ldots$ **do**

  $R_i = \mathcal{R}_i(I, Z_i)$;

  if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**

    return OK;

  else

    $S_i = \mathcal{S}_i(R_i)$;

    if $\alpha_i(I) \sqsubseteq S_i$ **then**

      $Z_{i+1} =$ **(?) refined** S_i;

      $A_{i+1} =$ **(?) refined** A_i;

    else

      return NOK;

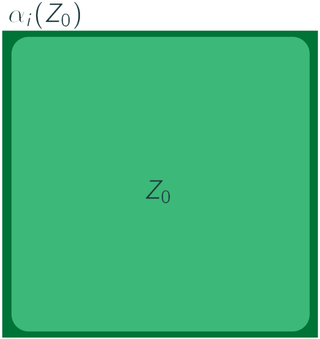
    end

  end

end

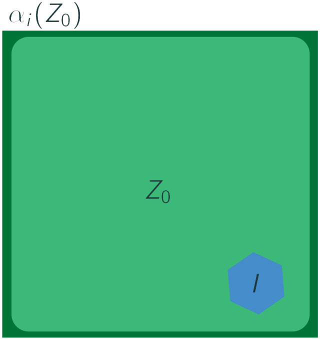
Algorithm

$Z_0 \leftarrow Z$;
for $i = 0\ 1\ 2\ 3\ \ldots$ **do**
    $R_i = \mathcal{R}_i(I, Z_i)$;
    if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
        return OK;
    else
        $S_i = \mathcal{S}_i(R_i)$;
        if $\alpha_i(I) \sqsubseteq S_i$ **then**
            $Z_{i+1} =$ **(?) refined** S_i;
            $A_{i+1} =$ **(?) refined** A_i;
        else
            return NOK;
        end
    end
end



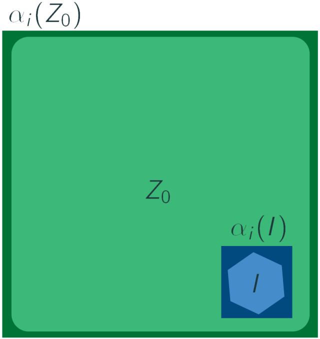
Algorithm

$Z_0 \leftarrow Z$;
for $i = 0\ 1\ 2\ 3\ \dots$ **do**
  $R_i = \mathcal{R}_i(I, Z_i)$;
  if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
    return OK;
  else
    $S_i = \mathcal{S}_i(R_i)$;
    if $\alpha_i(I) \sqsubseteq S_i$ **then**
      $Z_{i+1} =$ **(?) refined** S_i;
      $A_{i+1} =$ **(?) refined** A_i;
    else
      return NOK;
    end
  end
end



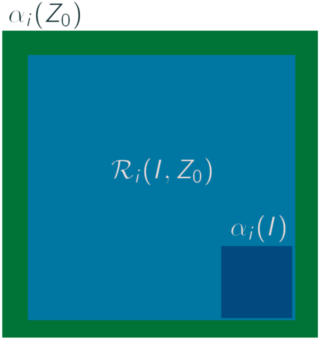
Algorithm

$Z_0 \leftarrow Z$;
for $i = 0\ 1\ 2\ 3\ \ldots$ **do**
  $R_i = \mathcal{R}_i(I, Z_i)$;
  if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
    return OK;
  else
    $S_i = \mathcal{S}_i(R_i)$;
    if $\alpha_i(I) \sqsubseteq S_i$ **then**
      $Z_{i+1} =$ **(?) refined** S_i;
      $A_{i+1} =$ **(?) refined** A_i;
    else
      return NOK;
    end
  end
end



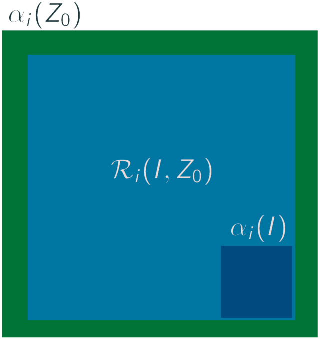
Algorithm

$Z_0 \leftarrow Z$;
for i = *0 1 2 3 ...* **do**
  $R_i = \mathcal{R}_i(I, Z_i)$;
  if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
    return OK;
  else
    $S_i = \mathcal{S}_i(R_i)$;
    if $\alpha_i(I) \sqsubseteq S_i$ **then**
      Z_{i+1} = **(?) refined** S_i;
      A_{i+1} = **(?) refined** A_i;
    else
      return NOK;
    end
  end
end



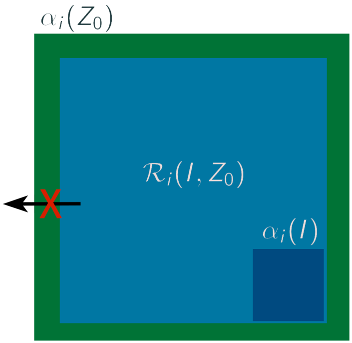
Algorithm

$Z_0 \leftarrow Z$;
for $i = 0\ 1\ 2\ 3\ \ldots$ **do**
  $R_i = \mathcal{R}_i(I, Z_i)$;
  if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
    return OK;
  else
    $S_i = \mathcal{S}_i(R_i)$;
    if $\alpha_i(I) \sqsubseteq S_i$ **then**
      $Z_{i+1} =$ **(?) refined** S_i;
      $A_{i+1} =$ **(?) refined** A_i;
    else
      return NOK;
    end
  end
end



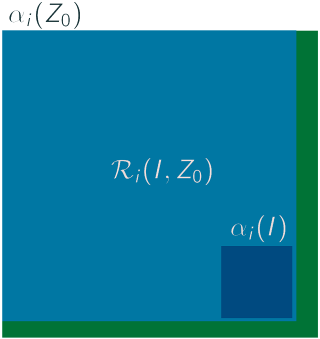
Algorithm

$Z_0 \leftarrow Z$;
for $i = 0\ 1\ 2\ 3\ \ldots$ **do**
  $R_i = \mathcal{R}_i(I, Z_i)$;
  if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
    return OK;
  else
    $S_i = \mathcal{S}_i(R_i)$;
    if $\alpha_i(I) \sqsubseteq S_i$ **then**
      $Z_{i+1} =$ **(?) refined** S_i;
      $A_{i+1} =$ **(?) refined** A_i;
    else
      return NOK;
    end
  end
end



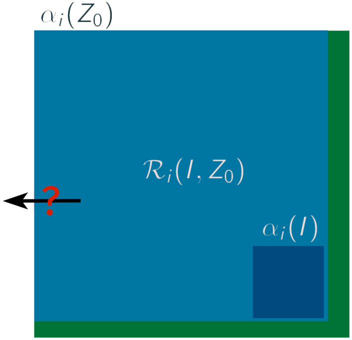
Algorithm

$Z_0 \leftarrow Z$;
for $i = 0\ 1\ 2\ 3\ \ldots$ **do**
$\quad R_i = \mathcal{R}_i(I, Z_i)$;
$\quad$ **if** $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
$\quad\quad$ **return OK**;
$\quad$ **else**
$\quad\quad S_i = \mathcal{S}_i(R_i)$;
$\quad\quad$ **if** $\alpha_i(I) \sqsubseteq S_i$ **then**
$\quad\quad\quad Z_{i+1} =$ **(?) refined** S_i;
$\quad\quad\quad A_{i+1} =$ **(?) refined** A_i;
$\quad\quad$ **else**
$\quad\quad\quad$ **return NOK**;
$\quad\quad$ **end**
$\quad$ **end**
end



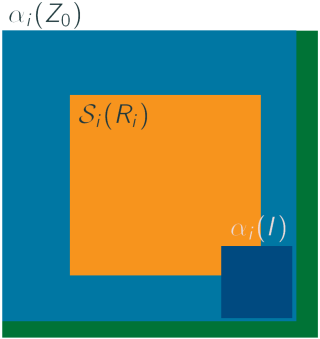
Algorithm

$Z_0 \leftarrow Z$;
for $i = 0\ 1\ 2\ 3\ \ldots$ **do**
  $R_i = \mathcal{R}_i(I, Z_i)$;
  if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
    return OK;
  else
    $S_i = \mathcal{S}_i(R_i)$;
    if $\alpha_i(I) \sqsubseteq S_i$ **then**
      $Z_{i+1} =$ **(?) refined** S_i;
      $A_{i+1} =$ **(?) refined** A_i;
    else
      return NOK;
    end
  end
end



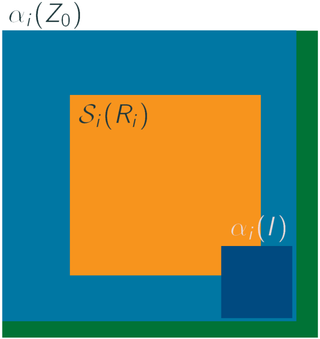
Algorithm

$Z_0 \leftarrow Z$;
for $i = 0\ 1\ 2\ 3\ \ldots$ **do**
    $R_i = \mathcal{R}_i(I, Z_i)$;
    if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
        return OK;
    else
        $S_i = \mathcal{S}_i(R_i)$;
        if $\alpha_i(I) \sqsubseteq S_i$ **then**
            $Z_{i+1} =$ **(?) refined** S_i;
            $A_{i+1} =$ **(?) refined** A_i;
        else
            return NOK;
        end
    end
end



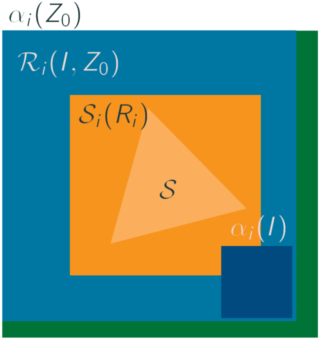
Algorithm

$Z_0 \leftarrow Z$;
for $i = 0\ 1\ 2\ 3\ \ldots$ **do**
  $R_i = \mathcal{R}_i(I, Z_i)$;
  if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
    return OK;
  else
    $S_i = \mathcal{S}_i(R_i)$;
    if $\alpha_i(I) \sqsubseteq S_i$ **then**
      $Z_{i+1} =$ **(?) refined** S_i;
      $A_{i+1} =$ **(?) refined** A_i;
    else
      return NOK;
    end
  end
end



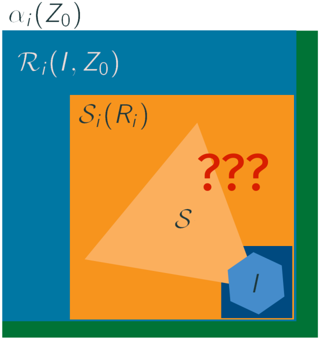
Algorithm

$Z_0 \leftarrow Z$;
for $i = 0\ 1\ 2\ 3\ \ldots$ **do**
  $R_i = \mathcal{R}_i(I, Z_i)$;
  if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
    return OK;
  else
    $S_i = \mathcal{S}_i(R_i)$;
    if $\alpha_i(I) \sqsubseteq S_i$ **then**
      $Z_{i+1} =$ **(?) refined** S_i;
      $A_{i+1} =$ **(?) refined** A_i;
    else
      return NOK;
    end
  end
end



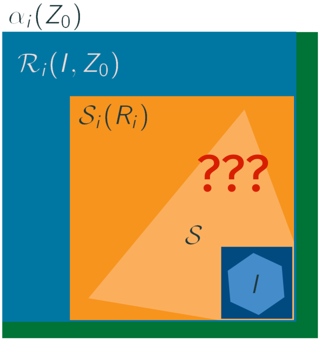
Algorithm

$Z_0 \leftarrow Z$;
for $i = 0\ 1\ 2\ 3\ \ldots$ **do**
  $R_i = \mathcal{R}_i(I, Z_i)$;
  if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
    return OK;
  else
    $S_i = \mathcal{S}_i(R_i)$;
    if $\alpha_i(I) \sqsubseteq S_i$ **then**
      $Z_{i+1} =$ **(?) refined** S_i;
      $A_{i+1} =$ **(?) refined** A_i;
    else
      return NOK;
    end
  end
end



Algorithm

$Z_0 \leftarrow Z$;
for $i = 0\ 1\ 2\ 3\ \ldots$ **do**
  $R_i = \mathcal{R}_i(I, Z_i)$;
  if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
    return OK;
  else
    $S_i = \mathcal{S}_i(R_i)$;
    if $\alpha_i(I) \sqsubseteq S_i$ **then**
      $Z_{i+1} =$ **(?) refined** S_i;
      $A_{i+1} =$ **(?) refined** A_i;
    else
      return NOK;
    end
  end
end



Algorithm

$Z_0 \leftarrow Z$;

for $i = 0\ 1\ 2\ 3\ \ldots$ **do**

  $R_i = \mathcal{R}_i(I, Z_i)$;

  if $\alpha_i(I \cup post(\gamma(R_i))) \sqsubseteq \alpha_i(Z_i)$ **then**

    return OK;

  else

    $S_i = \mathcal{S}_i(R_i)$;

    if $\alpha_i(I) \sqsubseteq S_i$ **then**

      $Z_{i+1} =$ **(?) refined** S_i;

      $A_{i+1} =$ **(?) refined** A_i;

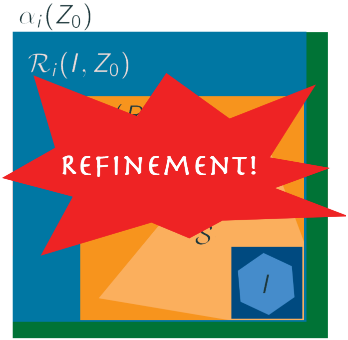
    else

      return NOK;

    end

  end

end



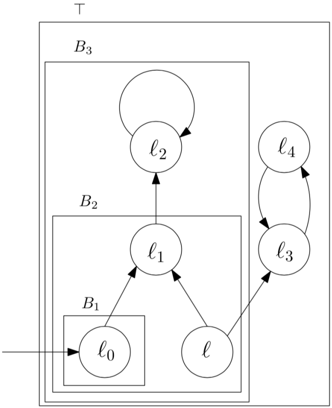
Refinement

Idea: Refine using states in S_i distinguishable in one step.

$$Z_{i+1} = \gamma(S_i) \cap \widetilde{pre}(\gamma(S_i))$$
$$\gamma(A_{i+1}) = \mathcal{M}(\{Z_{i+1}\} \cup \gamma(A_i))$$

Refinement

$$Z_{i+1} = \gamma(S_i) \cap \widetilde{pre}(\gamma(S_i))$$
$$\gamma(A_{i+1}) = \mathcal{M}(\{Z_{i+1}\} \cup \gamma(A_i))$$



$$I = \{l_0\} \quad Z_0 = B_3$$
$$R_0 = B_3 \quad S_0 = B_3$$
$$\top \in \alpha_0(post(\gamma(R_0))) \Rightarrow \text{cannot say OK}$$
$$\alpha_0(I) \sqsubseteq S_0 \Rightarrow \text{cannot say NOK}$$
$$Z_1 = \{l_0, l_1, l_2\}$$
$$A_i = A_0 \cup \{Z_1, \{l_0, l_1\}\}$$
$$R_1 = Z_1$$
$$\alpha_1(post(\gamma(R_1))) \sqsupseteq \alpha_1(Z_1) \Rightarrow \text{OK}$$

Additional results

- The algorithm can be enhanced using fixpoint computation acceleration (under-approximation of transitive closure of transition relation).

Additional results

- The algorithm can be enhanced using fixpoint computation acceleration (under-approximation of transitive closure of transition relation).
- Using Boolean instead of Moore closure has the same power, but asymptotically worse complexity.

Additional results

- The algorithm can be enhanced using fixpoint computation acceleration (under-approximation of transitive closure of transition relation).
- Using Boolean instead of Moore closure has the same power, but asymptotically worse complexity.
- If CEGAR can compute the result, so can the fixpoint refinement.

Additional results

- The algorithm can be enhanced using fixpoint computation acceleration (under-approximation of transitive closure of transition relation).
- Using Boolean instead of Moore closure has the same power, but asymptotically worse complexity.
- If CEGAR can compute the result, so can the fixpoint refinement.
- There exist systems where CEGAR does not terminate but fixpoint refinement does.

Experiments

At the present time, no implementation of Alg. 1 is available...