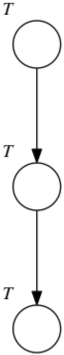
Slicing Abstractions for CPAchecker

Martin Spießl

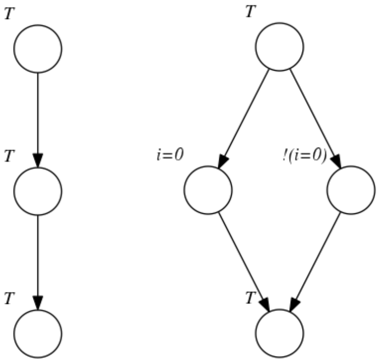
LMU Munich

5. September 2017

Slicing Abstractions Idea

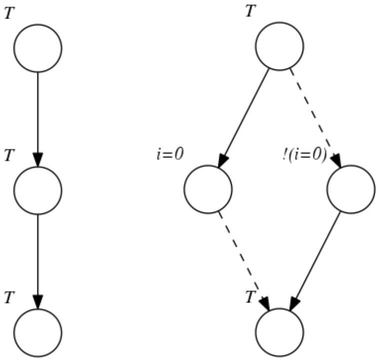


Slicing Abstractions Idea



- Split abstraction state into two states
- Disjunction of the splitted states represent the same concrete states as in the original state $\Rightarrow$ soundness
- Incoming & outgoing edges have to be copied

Slicing Abstractions Idea



- Split abstraction state into two states
- Disjunction of the splitted states represent the same concrete states as in the original state $\Rightarrow$ soundness
- Incoming & outgoing edges have to be copied
- Slice by removing infeasible edges
- disconnected subgraphs can be removed

Slicing Abstractions in Software Model Checking

Brückner, Dräger, Finkbeiner, Wehrheim (2007). Slicing Abstractions

- program counter tracked symbolically
- uses predicate abstraction
- Implementation: SLAB

Ermis, Hoenicke, Podelski (2012). Splitting via Interpolants

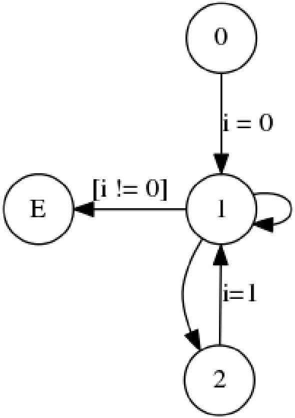
- explicit program counter
- no predicate abstraction computation (comparable to IMPACT)
- uses Large-Block Encoding
- Implementation: Ultimate Kojak

Example Program

example.c

```
0  int i = 0;
1  do {
2     assert i == 0;
3     if (*) {
4        i = 1;
5     }
6  while (true);
```

CFA



Splitting via Interpolants (Kojak)

Basic Steps:

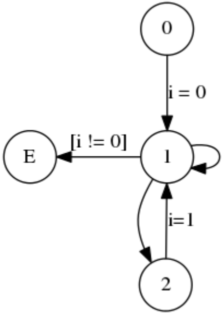
- **Split:** Split states along the error path using interpolants
- **Slice:** Remove infeasible edges

Termination

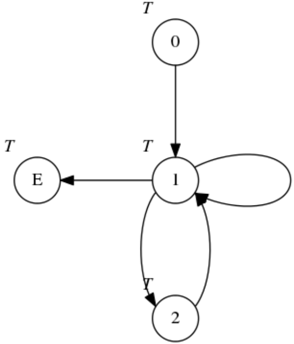
- when finding a feasible counterexample: return false
- when all error states are disconnected: return true

Splitting via Interpolants (Kojak)

CFA

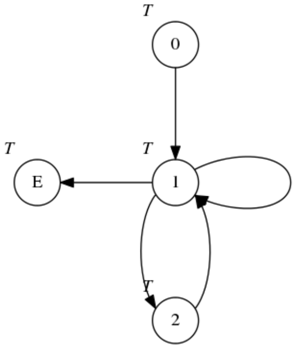


Abstraction

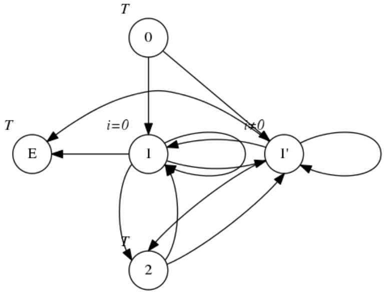


Splitting via Interpolants (Kojak)

Abstraction

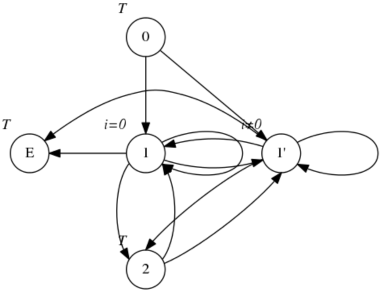


Split

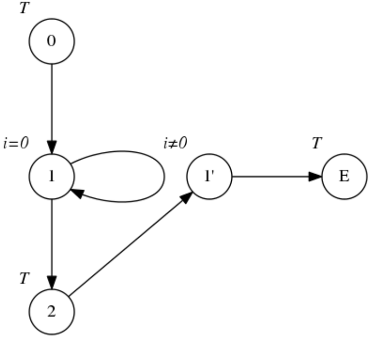


Splitting via Interpolants (Kojak)

Split



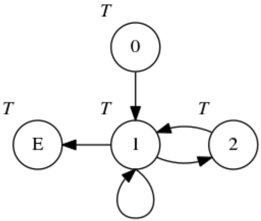
Slice



CPA+ Algorithm Setup for Kojak

Algorithm 1: CPA+ Algorithm (simplified)

while $waitlist \neq \emptyset$ **do**
  pop e from waitlist
  forall e' *with* $e \rightsquigarrow e'$ **do**
    forall $e'' \in reached$ **do**
      $e_{new} := \text{merge}(e', e'')$
      if $e_{new} \neq e''$ **then**
        $\text{waitlist} := (\text{waitlist} \cup \{e_{new}\}) \setminus \{e''\}$
        $\text{reached} := (\text{reached} \cup \{e_{new}\}) \setminus \{e''\}$
    if not $stop(e', reached)$ **then**
      $\text{waitlist} := \text{waitlist} \cup \{e'\}$
      $\text{reached} := \text{reached} \cup \{e'\}$
return $(\text{reached}, \text{waitlist})$

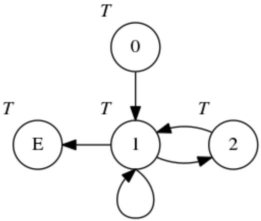


CPA+ Algorithm Setup for Kojak

Algorithm 1: CPA+ Algorithm (simplified)

while $waitlist \neq \emptyset$ **do**
  pop e from waitlist
  forall e' *with* $e \rightsquigarrow e'$ **do**
    forall $e'' \in reached$ **do**
      $e_{new} := \text{merge}(e', e'')$
      if $e_{new} \neq e''$ **then**
        $\text{waitlist} := (\text{waitlist} \cup \{e_{new}\}) \setminus \{e''\}$
        $\text{reached} := (\text{reached} \cup \{e_{new}\}) \setminus \{e''\}$
    if not $stop(e', reached)$ **then**
      $\text{waitlist} := \text{waitlist} \cup \{e'\}$
      $\text{reached} := \text{reached} \cup \{e'\}$
return $(\text{reached}, \text{waitlist})$

- merge operator:
  Merge states at same location in the ARG, but do not re-add them to the waitlist*



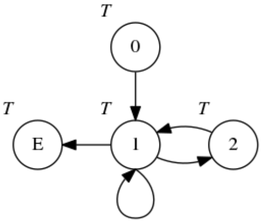
*could also be done using an additional CPA that tracks all parent locations for a state

CPA+ Algorithm Setup for Kojak

Algorithm 1: CPA+ Algorithm (simplified)

while $waitlist \neq \emptyset$ **do**
  pop e from waitlist
  forall e' *with* $e \rightsquigarrow e'$ **do**
    forall $e'' \in reached$ **do**
      $e_{new} := \mathrm{merge}(e', e'')$
      if $e_{new} \neq e''$ **then**
        waitlist $:= (\text{waitlist} \cup \{e_{new}\}) \setminus \{e''\}$
        reached $:= (\text{reached} \cup \{e_{new}\}) \setminus \{e''\}$
    if not $stop(e', reached)$ **then**
      waitlist $:= \text{waitlist} \cup \{e'\}$
      reached $:= \text{reached} \cup \{e'\}$
return (reached, waitlist)

- merge operator: Merge states at same location in the ARG, but do not re-add them to the waitlist*
- $stop_{sep}$ is sufficient, since all states have abstraction formula $\top$

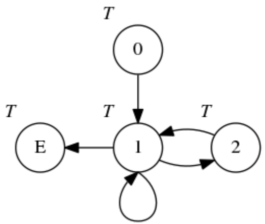


*could also be done using an additional CPA that tracks all parent locations for a state

CPA+ Algorithm Setup for Kojak

Algorithm 1: CPA+ Algorithm (simplified)

while *waitlist* $\neq \emptyset$ **do**
  pop e from waitlist
  forall e' *with* $e \rightsquigarrow e'$ **do**
    forall $e'' \in$ *reached* **do**
      e_{new} := merge (e', e'')
      if $e_{new} \neq e''$ **then**
        waitlist := (waitlist $\cup \{e_{new}\}) \setminus \{e''\}$
        reached := (reached $\cup \{e_{new}\}) \setminus \{e''\}$
    if not *stop* $(e', reached)$ **then**
      waitlist := waitlist $\cup \{e'\}$
      reached := reached $\cup \{e'\}$
return (reached, waitlist)



- merge operator: Merge states at same location in the ARG, but do not re-add them to the waitlist*
- $stop_{sep}$ is sufficient, since all states have abstraction formula $\top$
- Global Refinement: Counterexamples are not refined until the algorithm finishes

*could also be done using an additional CPA that tracks all parent locations for a state

Slicing Abstractions as CEGAR Refinement Strategy

Algorithm 2: CEGAR Algorithm (simplified

```
while l_ERR ∈ reached do
    (reached, waitlist) := CPA+ (reached, waitlist)
    if l_ERR ∈ reached then
        (reached, waitlist) := refine (reached, waitlist)
        if l_ERR ∈ reached then
            return false
    else
        return true
return true
```

Slicing Abstractions as CEGAR Refinement Strategy

Algorithm 2: CEGAR Algorithm (simplified

while $l_{ERR} \in reached$ **do**
    (reached, waitlist) := CPA+ (reached, waitlist)
    if $l_{ERR} \in reached$ **then**
        (reached, waitlist) := refine (reached, waitlist)
        if $l_{ERR} \in reached$ **then**
            return false
    else
        return true
return true

- Formulate Splitting and Slicing as CEGAR Refinement Strategy (called in refine)

Slicing Abstractions as CEGAR Refinement Strategy

Algorithm 2: CEGAR Algorithm (simplified

while $l_{ERR} \in \mathit{reached}$ **do**
    $(\text{reached}, \text{waitlist}) := \text{CPA+}\,(\text{reached}, \text{waitlist})$
    if $l_{ERR} \in \mathit{reached}$ **then**
        $(\text{reached}, \text{waitlist}) := \text{refine}\,(\text{reached}, \text{waitlist})$
        if $l_{ERR} \in \mathit{reached}$ **then**
            return false
    else
        return true
return true

- Formulate Splitting and Slicing as CEGAR Refinement Strategy (called in refine)
- Strategy repeats Splitting and Slicing until feasible counterexample is found or all error states have been removed

Slicing Abstractions as CEGAR Refinement Strategy

Algorithm 2: CEGAR Algorithm (simplified

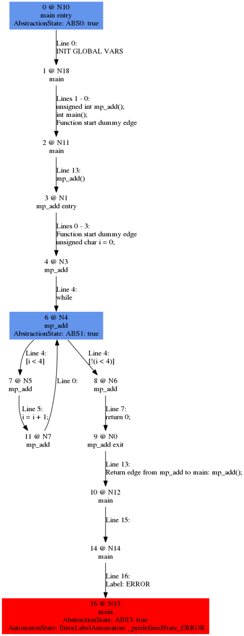
while $l_{ERR} \in$ *reached* **do**
  (reached, waitlist) := CPA+ (reached, waitlist)
  if $l_{ERR} \in$ *reached* **then**
    (reached, waitlist) := refine (reached, waitlist)
    if $l_{ERR} \in$ *reached* **then**
      return false
  else
    return true
return true

- Formulate Splitting and Slicing as CEGAR Refinement Strategy (called in refine)
- Strategy repeats Splitting and Slicing until feasible counterexample is found or all error states have been removed
- $\Rightarrow$ CPA+ and refine will only be called once in this setup

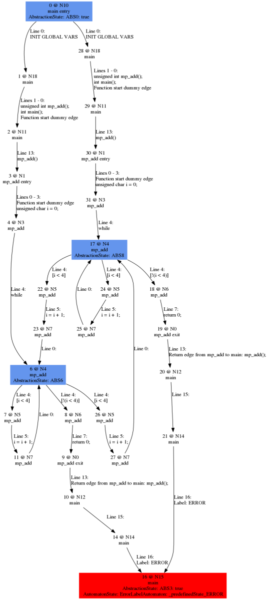
Adjustable-Block Encoding

- ABE as flexible replacement for LBE (used by Kojak)
- PredicateCPA is already used to store abstraction formulas
  $\Rightarrow$ ABE from PredicateCPA can be reused
- However: BlockFormulas from PredicateCPA cannot be used because the abstraction states in the ARG do not form a tree anymore (previous abstraction state will be ambigious)
  $\Rightarrow$ dynamically recalculate them
- Loss of tree shape in ARG causes other problems
- Non-abstraction states have to be copied when splitting states (example on next slide)

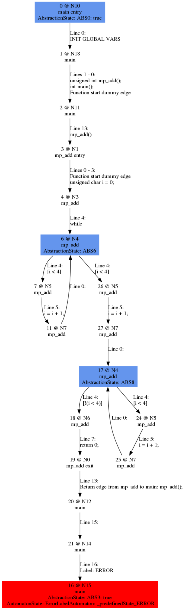
[]



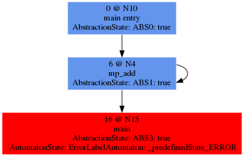
In Refinement after Splitting



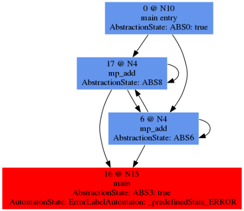
In Refinement after slicing



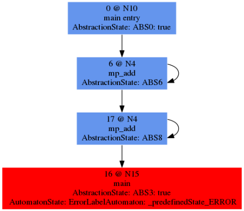
[]



In Refinement after Splitting



In Refinement after slicing



Possible Applications

Slicing Abstractions Refinement Strategy

- Can be used instead of other Refinement Strategies
- Will preserve tree structure in ARG, e.g. in an IMPACT configuration
- Leads to similar or identical results as IMPACT, depending on setting

Kojak-like Analysis

- Can generate (non-inductive) invariants
- Explore different block sizes using ABE
- Explore different optimizations to edge slicing process (reduce number of solver calls)