Configurable Software Verification:

Concretizing the Convergence of Model Checking and Program Analysis

Dirk Beyer[1] Thomas A. Henzinger[2]

Grégory Théoduloz[2]

[1]Simon Fraser University, BC, Canada
[2]EPFL, Switzerland

Introduction

- **Goal**: Tool to combine and reuse different abstract domains

- Model Checking [Clarke/Emerson, Sifakis ’81]
    - E.g., predicate abstraction

- Program Analysis [Cousot/Cousot ’77]
    - E.g., pointer analysis

- In theory: no difference [Steffen ’91, Cousot/Cousot ’95, Schmidt ’98]

- Fine tune the dial between the two extremes

Model Checking

Reached, *Frontier* := { a_0 }
while *Frontier* $\neq \emptyset$ do

pop *a* from *Frontier*
for each *a'* $\in$ post(*a*) do

if $\neg$ stop(*a'*,*Reached*) then add *a'* to *Reached*, *Frontier*

return *Reached*

Configurable Program Analysis

Reached, *Frontier* := { a_0 }
while *Frontier* $\neq \emptyset$ do

pop *a* from *Frontier*
for each *a'* $\in$ post(*a*) do

for each *a''* $\in$ *Reached* do

a := merge(*a'*, *a''*)
if *a* $\neq$ *a''* then replace *a''* in *Reached*, *Frontier* by *a*

if $\neg$ stop(*a'*,*Reached*) then add *a'* to *Reached*, *Frontier*

return *Reached*

Configurable Program Analysis

Abstract interpreter [Cousot & Cousot]:

- concrete system $(C, c_0, \rightarrow)$
  - abstract domain $(A, \top, \bot, \sqsubseteq, \sqcup)$
    - abstraction function α: $C \rightarrow A$
    - concretization function γ: $A \rightarrow 2^C$
- transfer function post: $A \rightarrow 2^A$ s.t. $\cup_{c \in \gamma(a)} \{ c' : c \rightarrow c' \} \subseteq \cup_{a' \in \text{post}(a)} \gamma(a')$

Configurable Program Analysis

Configurable abstract interpreter:

- concrete system $(C, c_0, \rightarrow)$
  - abstract domain $(A, \top, \bot, \sqsubseteq, \sqcup)$
  - abstraction function α: $C \rightarrow A$
  - concretization function γ: $A \rightarrow 2^C$
- transfer function post: $A \rightarrow 2^A$ s.t. $\cup_{c \in \gamma(a)} \{ c' : c \rightarrow c' \} \subseteq \cup_{a' \in post(a)} \gamma(a')$
- merge operator merge: $A \times A \rightarrow A$ s.t. $a' \sqsubseteq$ merge(a, a')

  merge-sep(a, a') = a'

  merge-join(a, a') = $a \sqcup a'$

Configurable Program Analysis

Configurable abstract interpreter:

- concrete system $(C, c_0, \rightarrow)$
- abstract domain $(A, \top, \bot, \sqsubseteq, \sqcup)$
- abstraction function $\alpha: C \rightarrow A$
- concretization function $\gamma: A \rightarrow 2^C$

- transfer function post: $A \rightarrow 2^A$ s.t. $\bigcup_{c \in \gamma(a)} \{ c' : c \rightarrow c' \} \subseteq \bigcup_{a' \in \text{post}(a)} \gamma(a')$

- merge operator merge: $A \times A \rightarrow A$ s.t. $a' \sqsubseteq \text{merge}(a, a')$

  merge-sep(a, a') = a'

  merge-join(a, a') = $a \sqcup a'$

- termination check stop: $A \times 2^A \rightarrow \mathbb{B}$ s.t. if stop(a,R) then $\gamma(a) \subseteq \bigcup_{a' \in R} \gamma(a')$

  stop-sep(a, R) = $(\exists\, a' \in R : a \sqsubseteq a')$

  stop-join(a, R) = $(a \sqsubseteq \bigsqcup R)$

Configurable Program Analysis

Classical software model checking (e.g. SLAM):

- predicate abstraction
- merge-sep (build abstract reachability tree)
- stop-sep (stop at covered leaves)

Classical program analysis (e.g. TVLA):

- shape abstraction
- merge-join (annotate control locations)
- stop-join (stop at fixpoint)

Configurable Program Analysis

Classical software model checking (e.g. SLAM):

- predicate abstraction
- merge-sep (build abstract reachability tree)
- stop-sep (stop at covered leaves)

Classical program analysis (e.g. TVLA):

- shape abstraction
- merge-join (annotate control locations) ← possibly widen
- stop-join (stop at fixpoint)

Composite Program Analysis

- two configurable program analyses D_1 and D_2

- composite transfer function post
- composite merge operator merge
- composite termination check stop

defined from the components of D_1 and D_2 and
two strengthening operators str_i: $A_1 \times A_2 \rightarrow A_i$ such that

$$str_i(a_1,a_2) \sqsubseteq a_i$$

Example: $A_1 = \mathbb{P}$ (predicate abstraction)
$A_2 = \mathbb{S}$ (shape abstraction)

$str_2(a_1,a_2)$ sharpens field information for shapes by using predicate information

Yes, but ...

“You just need to change the abstract domain, and everything can be done by classical program analysis (merge-join; stop-join).”

Yes, but ...

“You just need to change the abstract domain, and everything can be done by classical program analysis (merge-join; stop-join).”

This is true, but misses the point: we want to use existing abstract interpreters as building blocks and

1. parameterize the execution engine
2. combine abstract interpreters

Composite Program Analysis

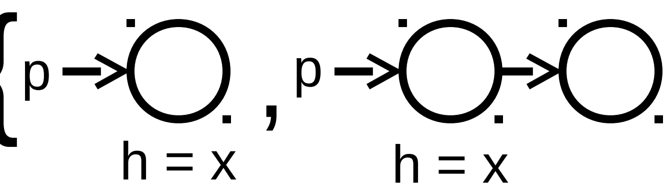
Example: predicate + shape abstraction

Composite analysis from the following domains:

$A_1 = \mathbb{L}$ (locations)
$A_2 = \mathbb{P}$ (predicate abstraction)
$A_3 = \mathbb{S}$ (shape abstraction)

(*loc., predicate cube, set of shape graphs*)

e.g. (4 , x = 4 Æ y < x , {



})

Composite Program Analysis

Example: predicate + shape abstraction

Choices for composite transfer

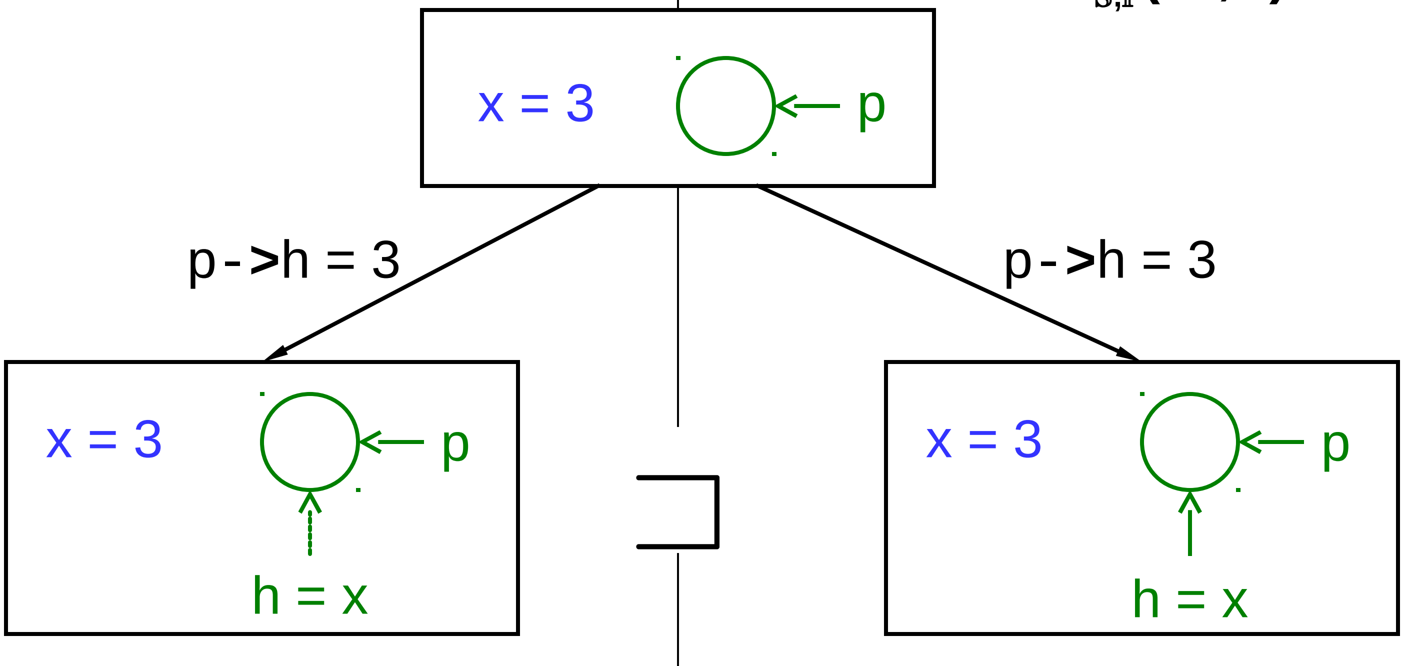
$\mathbb{L} \times \mathbb{P} \times \mathbb{S}$

post-cartesian

(l,r,s) ⇝ (l',r',s')
  iff l $\rightsquigarrow_{\mathbb{L}}$ l' Æ r$\rightsquigarrow_{\mathbb{P}}$r' Æ s$\rightsquigarrow_{\mathbb{S}}$s''

post-strengthened

(l,r,s) ⇝ (l',r',s')
  iff l $\rightsquigarrow_{\mathbb{L}}$ l' Æ r$\rightsquigarrow_{\mathbb{P}}$r' Æ s$\rightsquigarrow_{\mathbb{S}}$s''
  Æ s'= str$_{\mathbb{S},\mathbb{P}}$(s'',r')

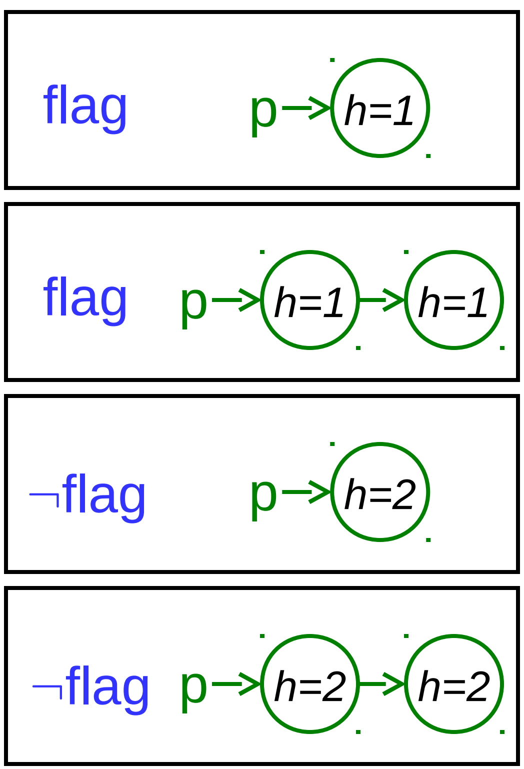


Composite Program Analysis

Example: predicate + shape abstraction $\mathbb{L} \times \mathbb{P} \times \mathbb{S}$

Choices for composite merge

merge-sep

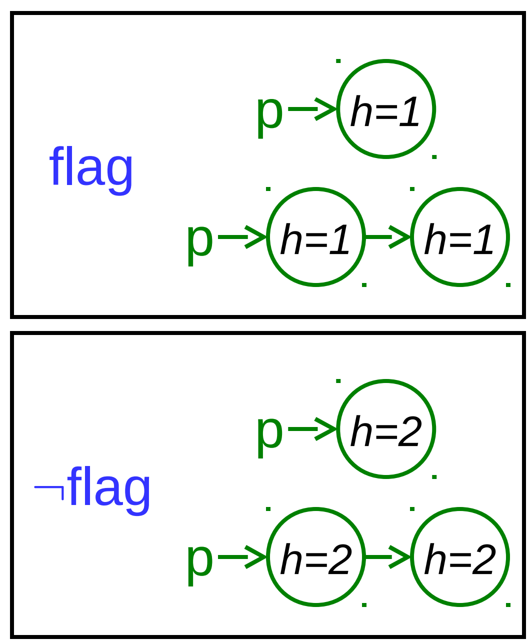


$\sqsubseteq$

merge-pred-join((l,r,s), (l’,r’,s’))

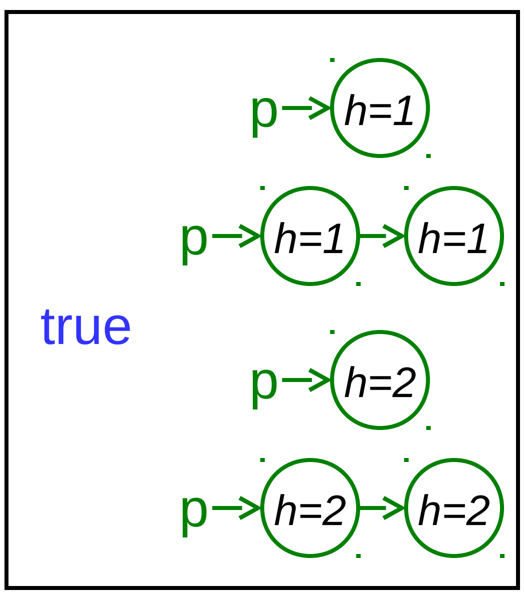
= (l’, r’, merge$_{\mathbb{S}}$(s, s’)) if l = l’ Æ **r = r’**,

(l’, r’, s’) otherwise



$\sqsubset$

merge-join



Composite Program Analysis

Example: predicate + shape abstraction $\mathbb{L} \times \mathbb{P} \times \mathbb{S}$

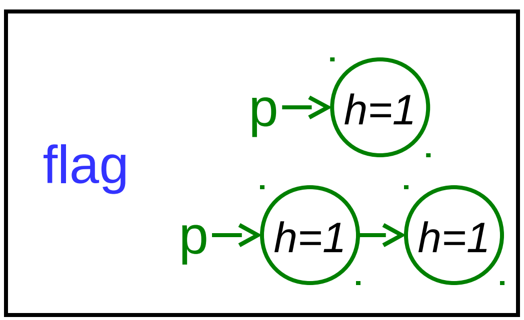
Choices for composite termination check

stop-sep

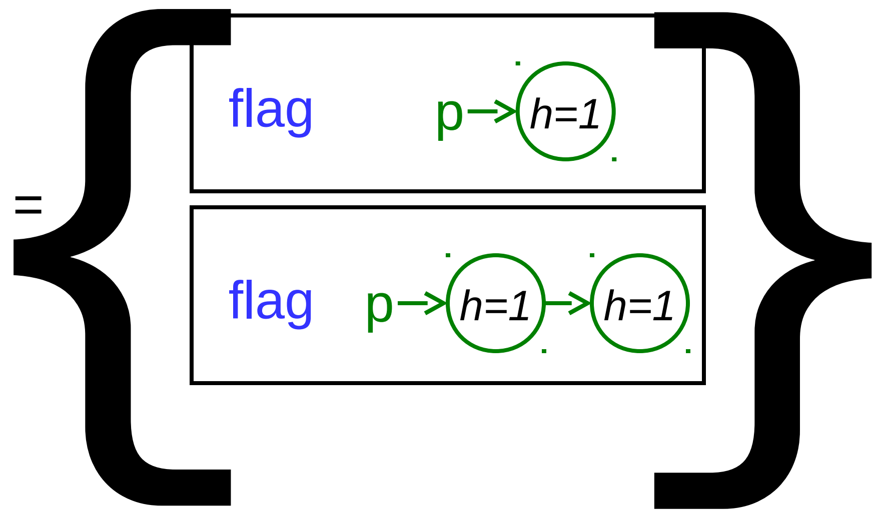
stop-pred-join((l,r,s), R)

$= \quad s \sqsubseteq \bigsqcup_{\mathbb{S}} \{ s' \mid (l,r,s') \in R \}$

a =



R =



stop-sep(a,R) = **NO**

stop-pred-join(a,R) = **YES**

Composite Program Analysis

Example: predicate + shape abstraction

- Lazy Shape Analysis [Beyer/Henzinger/T 2006]

transfer-cartesian	merge-sep	stop-sep

Improvements:

transfer-strengthened	merge-sep	stop-sep
transfer-cartesian	merge-sep	stop-pred-join

- Joining Data-flow with Predicate [Fischer/Jhala/Majumdar 2005]

transfer-cartesian	merge-pred-join	stop-sep

Implementation

- Based on building blocks from BLAST & TVLA

- Input:
  - C program
  - Abstraction definition (i.e., predicates, shape predicates)
  - Parameters to select operators (merge-sep, merge-cov, …)

- Post and strengthening use Simplify

```
List a = (List)malloc(...);
List p = a;
while (non det.) {
  if (flag)
    p->h = 1;
  else
    p->h = 2;
  p->n = (List)malloc(...);
  p = p->n;
}
p->h = 3;

p = a;
if (flag)
  while (p->h == 1) p = p->n;
else
  while (p->h == 2) p = p->n;
assert(p->h == 3);
```

Composite Program Analysis

Example: predicate + shape abstraction

Program	merge-sep stop-sep	merge-prjoin stop-sep	merge-sep stop-join	merge-join stop-join
list_1	0.37 s	0.42 s	0.32 s	0.41 s
list_2	0.85 s	5.24 s	0.86 s	5.36 s
list_4	9.67 s	> 600 s	11.87 s	> 600 s
list_flag	0.49 s	0.69 s	0.46 s	FP
alternating	0.61 s	0.86 s	0.60 s	FP
list_flag2	FP	FP	FP	FP

Verification time of examples for list manipulation program
(using post-cartesian)

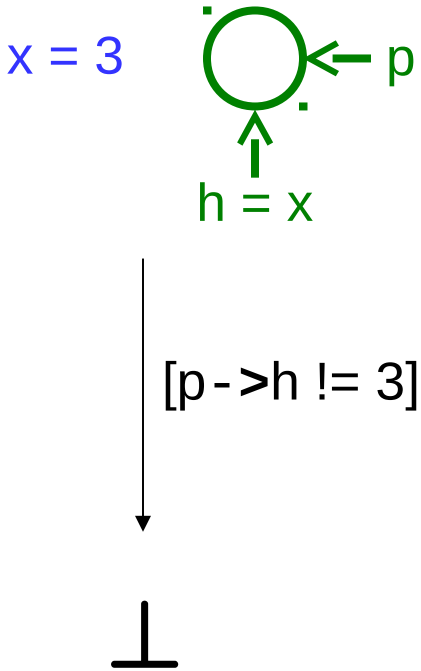
```
List a = (List)malloc(...);
List p = a;
int  x = 3;
while (non det.) {
  if (flag)
    p->h = 1;
  else
    p->h = 2;
  p->n = (List)malloc(...);
  p = p->n;
}
p->h = x;

p = a;
if (flag)
  while (p->h == 1) p = p->n;
else
  while (p->h == 2) p = p->n;
assert(p->h == 3);
```

Composite Program Analysis

Example: predicate + shape abstraction

Program	post-cartesian merge-sep stop-sep	post-strengthened merge-sep stop-sep
list_1	0.37 s	0.41 s
list_2	0.85 s	1.25 s
list_4	9.67 s	15.44 s
list_flag	0.49 s	0.79 s
alternating	0.61 s	0.96 s
list_flag2	FP	0.81 s



Example for which predicated lattice is best

[Fischer, Jhala & Majumdar 05]

$A_1 = \mathbb{L}$ (locations)
$A_2 = \mathbb{P}$ (predicate abstraction)
$A_3 = \mathbb{D}$ **(symbolic values lattice)**

- cartesian post
- merge-pred-join is much better than merge-sep
- stop-sep

Current Work / Summary

- Improve usability and flexibility of the tool

- Refinement (domain and operators)

- Q: Is the tool a model checker that explores states?
  A static analyzer interpreting an abstract program?

  A: **Both!**

?