SMT-Style Program Analysis with Value-based Refinements

Vijay D'Silva   Leopold Haller   Daniel Kröning

UNIVERSITY OF OXFORD

NSV-3
July 15, 2010

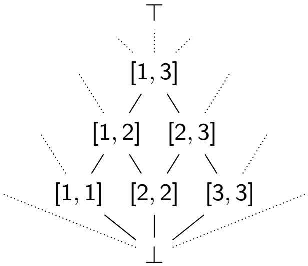
Outline

Imprecision in Abstract Interpretation

- Abstract interpretation sound but not complete.
- Incompleteness manifests in imprecision during the analysis.

Imprecision in Abstract Interpretation

- Abstract interpretation sound but not complete.
- Incompleteness manifests in imprecision during the analysis.



Example: Domain of Intervals

Imprecisions in the Domain

Imprecision in join

```
x:=*;
if(x > 5)
  y := -1;
else
  y := 1;

assert(y != 0);
```

Imprecisions in the Domain

Imprecision in join

```
x:=*;
if(x > 5)
  y := -1;
else
  y := 1;

assert(y != 0);
```

$\longrightarrow y \in [-1, -1], x \in [6, \infty]$

Imprecisions in the Domain

Imprecision in join

```
x:=*;
if(x > 5)
  y := -1;
else
  y := 1;

assert(y != 0);
```

$\texttt{y := -1;} \longrightarrow y \in [-1, -1], x \in [6, \infty]$

$\texttt{y := 1;} \longrightarrow y \in [1, 1], x \in [-\infty, 5]$

Imprecisions in the Domain

Imprecision in join

```
x:=*;
if(x > 5)
  y := -1;
else
  y := 1;

assert(y != 0);
```

$\longrightarrow y \in [-1,-1], x \in [6,\infty]$ (after `y := -1;`)

$\longrightarrow y \in [1,1], x \in [-\infty,5]$ (after `y := 1;`)

$\longrightarrow y \in [-1,1]$ (at `assert(y != 0);`)

The disjunction $y = 1 \vee y = -1$ cannot be expressed as an interval.

Imprecisions in the Domain

Imprecision in transformer

```
x:=y;
if(x > 5)
 assert(y > 5);
```

Imprecisions in the Domain

Imprecision in transformer

```
x:=y;
if(x > 5)
  assert(y > 5);
```

$\longrightarrow \top$

Imprecisions in the Domain

Imprecision in transformer

```
x:=y;
if(x > 5)
  assert(y > 5);
```

$\longrightarrow \top$

$\longrightarrow x \in [6, \infty]$

Imprecisions in the Domain

Imprecision in transformer

```
x:=y;
if(x > 5)
  assert(y > 5);
```

$\longrightarrow \top$ (beside `x:=y;`)

$\longrightarrow x \in [6, \infty]$ (beside `assert(y > 5);`)

Intervals cannot express relational information.

Imprecisions in the Analysis

Imprecision in widening

```
while(x < 50000)
{
  x++;

  if(y < x)
    y++;
}
```

$x \in [0, 0], y \in [0, 0]$

$x \in [0, 1], y \in [0, 1]$

widening

$x \in [0, \infty], y \in [0, \infty]$

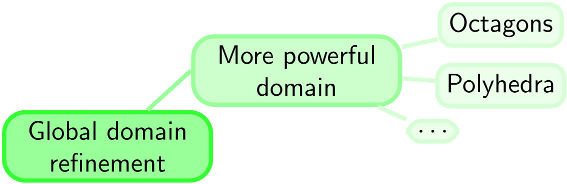
$x \in [50000, 50000]$, $y \in [0, \infty]$

Precision can be lost in the the analysis
Refinement of widening studied by, e.g., Gulavani et. al (TACAS 2008), Wang et al. (CAV 2007)

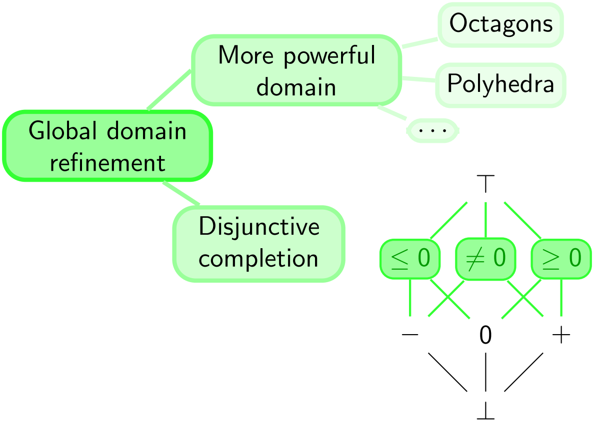
Refining Abstract Domains

Global domain refinement

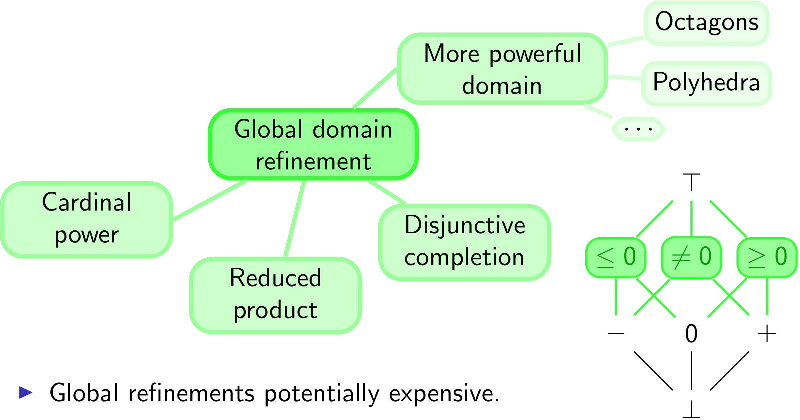
Refining Abstract Domains



Refining Abstract Domains



Refining Abstract Domains



- Global refinements potentially expensive.
- How can we locally refine an abstract domain?

Trace Partitioning

- Trace partitioning allows for flexible and local refinement

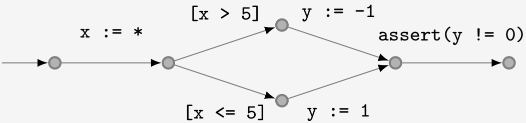
Trace Partitioning

- Trace partitioning allows for flexible and local refinement
  - Consider separately different sets of traces through a program
  - Similar to case splits in a mathematical proof.

Trace Partitioning

- Trace partitioning allows for flexible and local refinement
  - Consider separately different sets of traces through a program
  - Similar to case splits in a mathematical proof.

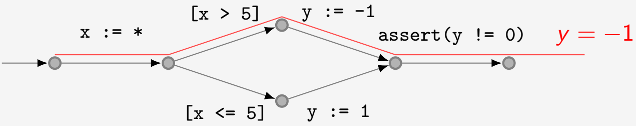
Control-flow based trace partitioning



Trace Partitioning

- Trace partitioning allows for flexible and local refinement
  - Consider separately different sets of traces through a program
  - Similar to case splits in a mathematical proof.

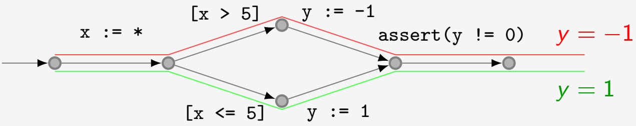
Control-flow based trace partitioning



Trace Partitioning

- Trace partitioning allows for flexible and local refinement
  - Consider separately different sets of traces through a program
  - Similar to case splits in a mathematical proof.

Control-flow based trace partitioning



Trace Partitioning

- Wide range of partitionings possible
  - control flow,
  - values of variables,
  - number of iterations through a loop, etc.

Trace Partitioning

- Wide range of partitionings possible
  - control flow,
  - values of variables,
  - number of iterations through a loop, etc.

Value-based partitioning

```
x:=y;
if(x > 5)
  assert(y > 5);
```

Trace Partitioning

- Wide range of partitionings possible
  - control flow,
  - values of variables,
  - number of iterations through a loop, etc.

Value-based partitioning

```
assume(y > 5);
         x:=y;
         if(x > 5)
           assert(y > 5);
```

$y > 5$

Trace Partitioning

- Wide range of partitionings possible
  - control flow,
  - values of variables,
  - number of iterations through a loop, etc.

Value-based partitioning

```
assume(y > 5);        assume(y <= 5);
          x:=y;
          if(x > 5)
            assert(y > 5);
```

$y > 5$ $\quad\quad$ $\perp$

Finding Partitioning Functions

- Trace partitioning allows one to refine the precision of an analysis down to explicit exploration of all traces.

Finding Partitioning Functions

- Trace partitioning allows one to refine the precision of an analysis down to explicit exploration of all traces.

The main question is:

Finding Partitioning Functions

- Trace partitioning allows one to refine the precision of an analysis down to explicit exploration of all traces.

The main question is:
How can we find a good partitioning?

Finding Partitioning Functions

- Trace partitioning allows one to refine the precision of an analysis down to explicit exploration of all traces.

The main question is:
How can we find a good partitioning?

- Precise enough to prove the property, and
- abstract enough to be efficient.

Finding Partitioning Functions

- Leino and Logozzo (APLAS 2005): Value-based trace partitionings based on counter examples
- Gulavani et al. (TACAS 2008): DAG-based Exploration of control-flow paths inside loops with splitting on demand.
- Gulwani et al. (PLDI 2009): Control-flow refinement for bounds analysis.
- Harris et al. (POPL 2010): Satisfiability Modulo Path Programs

Value-based Trace Partitionings

- If the abstract transformer $\hat{F}$ is too imprecise, find a set of transformers $\hat{F}_1, \ldots, \hat{F}_k$, such that

$$\bigcup_{1 \le i \le k} \gamma(\mu X.\ \hat{F}_i(X)) \ \supseteq \ \mu X.\, F(X)$$

Value-based Trace Partitionings

- If the abstract transformer $\hat{F}$ is too imprecise, find a set of transformers $\hat{F}_1, \ldots, \hat{F}_k$, such that

$$\bigcup_{1 \leq i \leq k} \gamma(\mu X.\ \hat{F}_i(X)) \supseteq \mu X.\, F(X)$$

- This can be done by clipping the analysis by an abstract element:

$$\hat{F}_i = \hat{F} \sqcap a_i$$

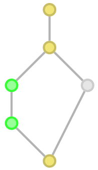
Value-based Trace Partitionings

- If the abstract transformer $\hat{F}$ is too imprecise, find a set of transformers $\hat{F}_1, \ldots, \hat{F}_k$, such that

$$\bigcup_{1 \leq i \leq k} \gamma(\mu X.\ \hat{F}_i(X)) \supseteq \mu X.\ F(X)$$

- This can be done by clipping the analysis by an abstract element:

$$\hat{F}_i = \hat{F} \sqcap a_i$$



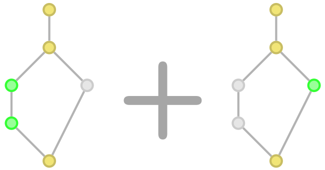
Value-based Trace Partitionings

- If the abstract transformer $\hat{F}$ is too imprecise, find a set of transformers $\hat{F}_1, \ldots, \hat{F}_k$, such that

$$\bigcup_{1 \le i \le k} \gamma(\mu X.\ \hat{F}_i(X)) \ \supseteq \ \mu X.\ F(X)$$

- This can be done by clipping the analysis by an abstract element:

$$\hat{F}_i = \hat{F} \sqcap a_i$$



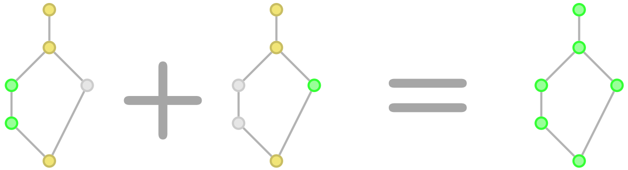
Value-based Trace Partitionings

- If the abstract transformer $\hat{F}$ is too imprecise, find a set of transformers $\hat{F}_1, \ldots, \hat{F}_k$, such that

$$\bigcup_{1 \leq i \leq k} \gamma(\mu X.\ \hat{F}_i(X)) \supseteq \mu X.\, F(X)$$

- This can be done by clipping the analysis by an abstract element:

$$\hat{F}_i = \hat{F} \sqcap a_i$$



Value-based Trace Partitionings

New question:

Value-based Trace Partitionings

New question:

How can we find such a set of elements $a_1, \ldots, a_k$*?*

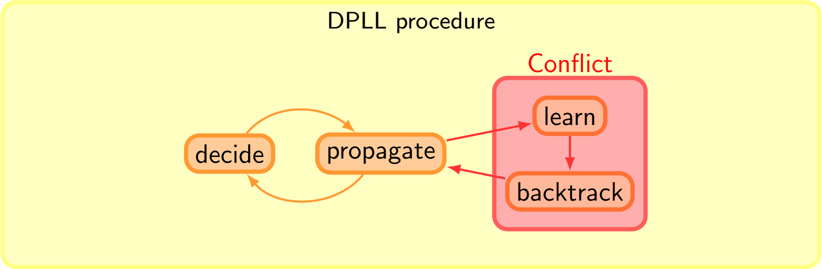
Value-based Trace Partitionings

New question:

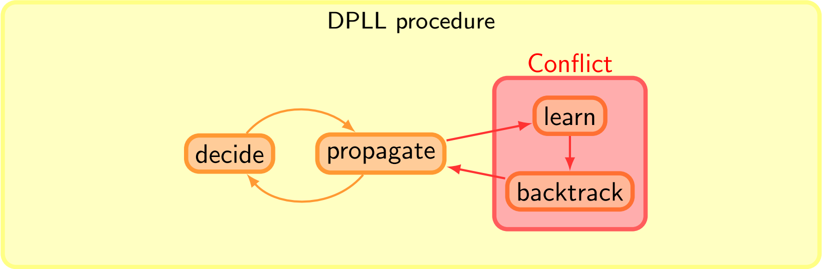
How can we find such a set of elements $a_1, \ldots, a_k$?

Use the search architecture of a SAT solver!

DPLL framework

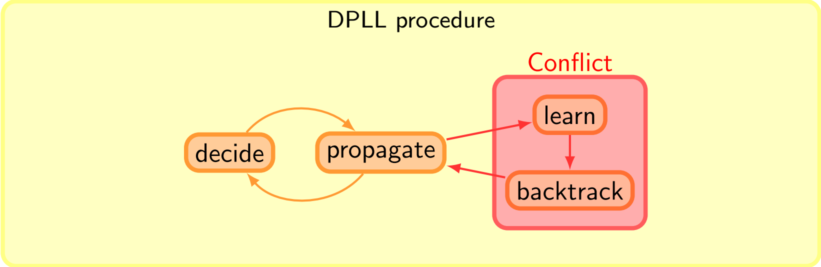


DPLL framework



- Main phases of the DPLL procedure:

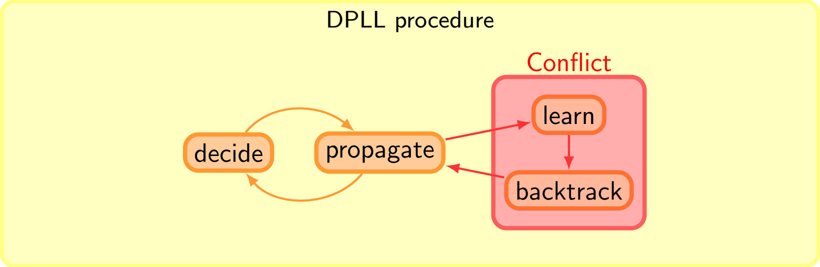
DPLL framework



- Main phases of the DPLL procedure:

  Decision Assume a value for an undetermined variable

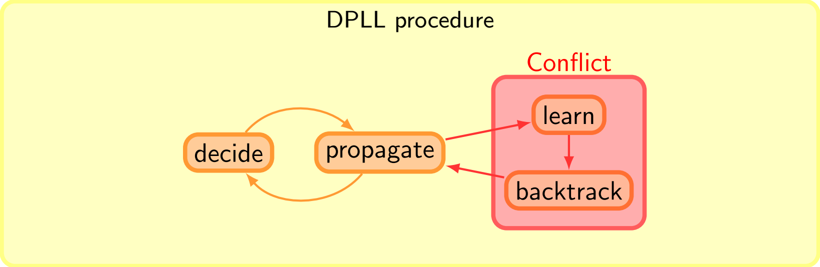
DPLL framework



- Main phases of the DPLL procedure:

Decision	Assume a value for an undetermined variable
Propagation	Deduce implied variable values

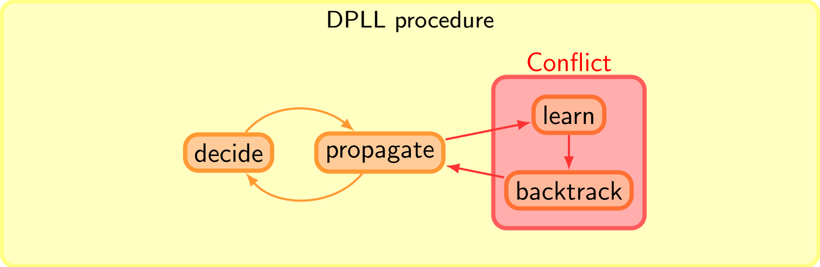
DPLL framework



- Main phases of the DPLL procedure:

Decision	Assume a value for an undetermined variable
Propagation	Deduce implied variable values
Learning	Learn reason for conflict and backtrack

DPLL framework



- Main phases of the DPLL procedure:

Decision	Assume a value for an undetermined variable
Propagation	Deduce implied variable values
Learning	Learn reason for conflict and backtrack

DPLL Procedure

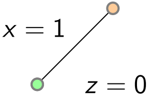
Is $\phi(x, y, z)$ satisfiable?

DPLL Procedure

Is $\phi(x, y, z)$ satisfiable?

DPLL Procedure

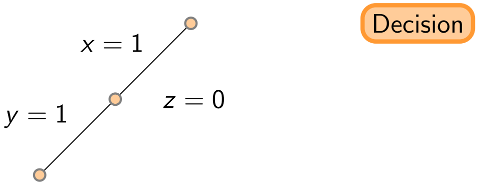
Is $\phi(x, y, z)$ satisfiable?



Propagation

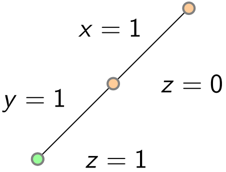
DPLL Procedure

Is $\phi(x, y, z)$ satisfiable?



DPLL Procedure

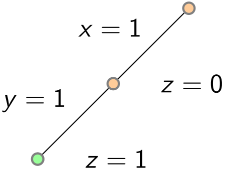
Is $\phi(x, y, z)$ satisfiable?



Propagation

DPLL Procedure

Is $\phi(x, y, z)$ satisfiable?

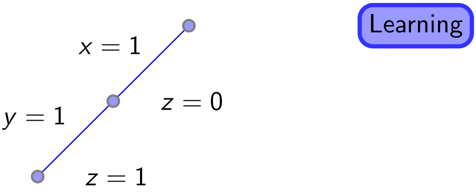


Propagation

Conflict

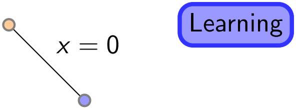
DPLL Procedure

Is $\phi(x, y, z)$ satisfiable?

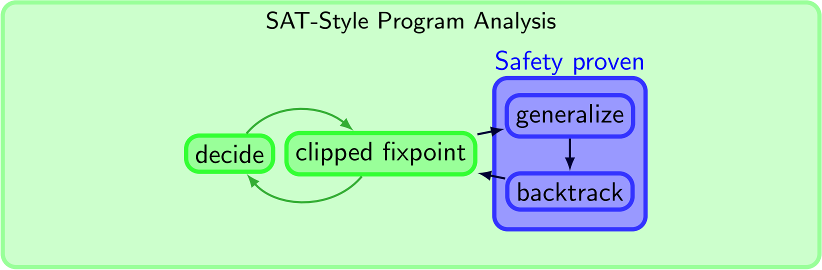


DPLL Procedure

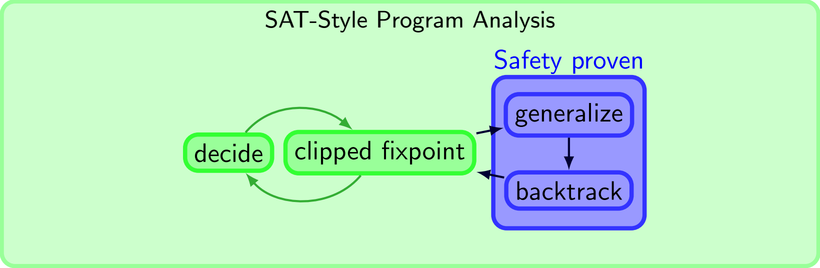
Is $\phi(x, y, z)$ satisfiable?



SAT-Style Program Analysis

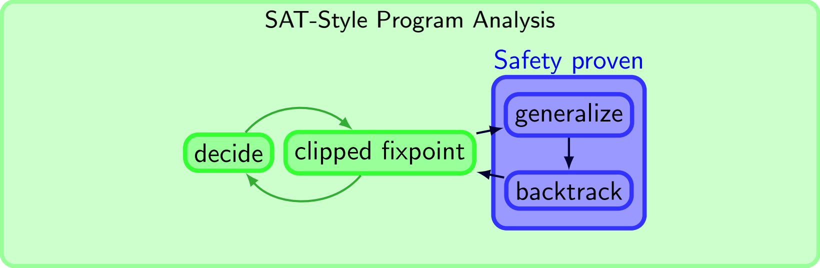


SAT-Style Program Analysis



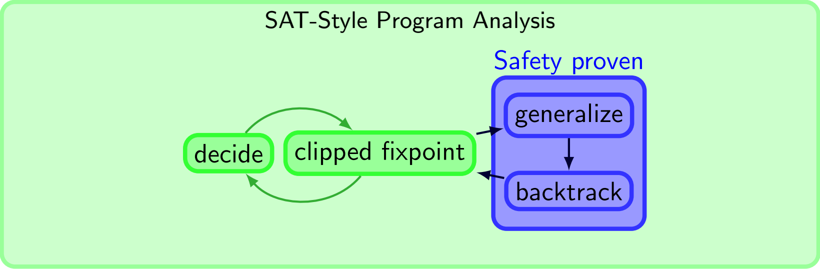
Decision Refine current element a by $a' \sqsubset a$

SAT-Style Program Analysis



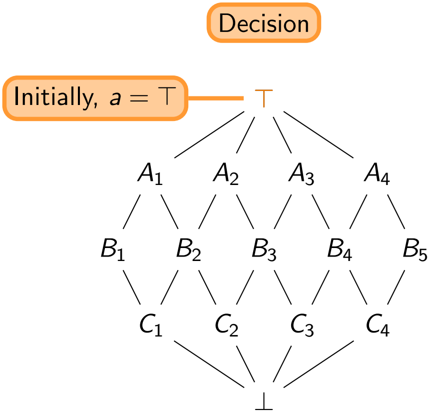
Decision	Refine current element a by $a' \sqsubset a$
Propagation	Compute clipped fixpoint $\mu X. \hat{T}(X) \sqcap a'$

SAT-Style Program Analysis

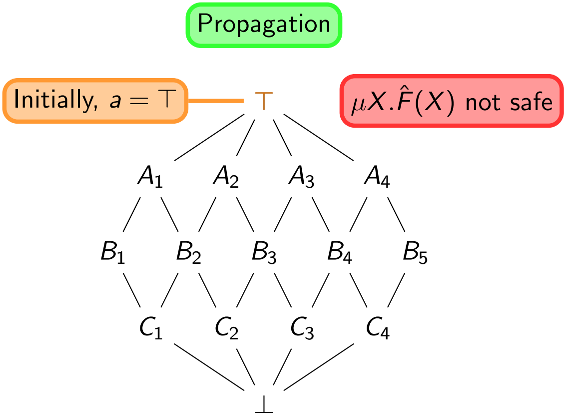


Decision	Refine current element a by $a' \sqsubset a$
Propagation	Compute clipped fixpoint $\mu X.\hat{T}(X) \sqcap a'$
Learning	Find $a'' \sqsupseteq a'$, such that $\mu X.\hat{F}(X) \sqcap a''$ is safe.

SAT-Style Program Analysis



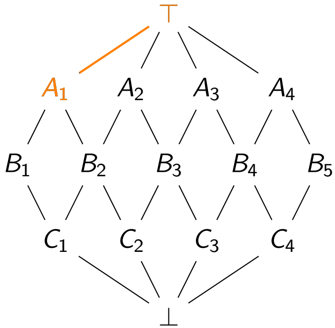
SAT-Style Program Analysis



SAT-Style Program Analysis

Decision

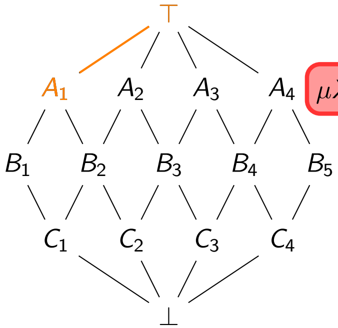
Decision: refine a



SAT-Style Program Analysis

Propagation

Decision: refine a

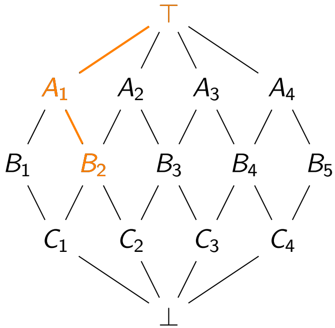


$\mu X.(\hat{F}(X) \sqcap A_1)$ not safe

SAT-Style Program Analysis

Decision

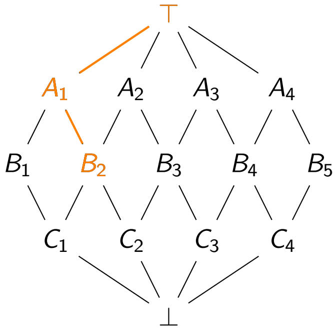
Decision: refine a



SAT-Style Program Analysis

Propagation

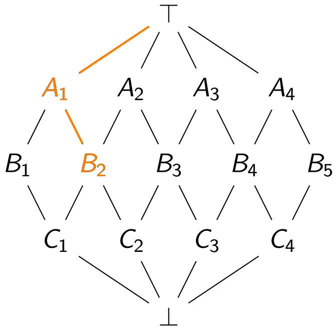
Decision: refine a



$\mu X.(\hat{F}(X) \sqcap B_2)$ safe

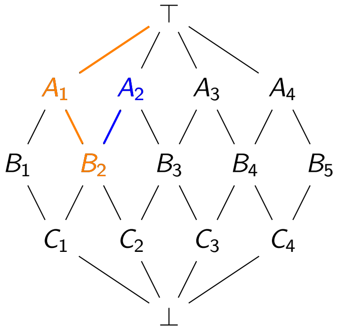
SAT-Style Program Analysis

Generalization



SAT-Style Program Analysis

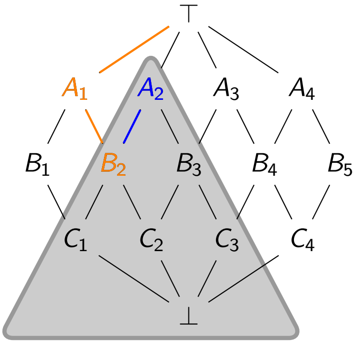
Generalization



$\mu X.\hat{F}(X) \sqcap A_2$ safe

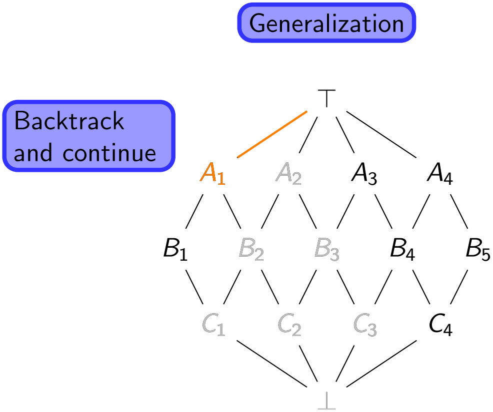
SAT-Style Program Analysis

Generalization



$\mu X.\hat{F}(X) \sqcap A_2$ safe

SAT-Style Program Analysis



Comments on Analysis

- When can we efficiently prove safety with this?

Comments on Analysis

- When can we efficiently prove safety with this?
  - When there is a small and finite number of elements $a_1, \ldots, a_k$ such that the fixpoints $\mu X.(\hat{F}(X) \sqcap a_i)$ can be put together to form a concrete postfixpoint.

Comments on Analysis

- When can we efficiently prove safety with this?
  - When there is a small and finite number of elements $a_1, \ldots, a_k$ such that the fixpoints $\mu X.(\hat{F}(X) \sqcap a_i)$ can be put together to form a concrete postfixpoint.
- Specific implementation issues:
  - Generalization step
  - Decision heuristic

Value-based Refinement for Intervals

We have created a preliminary instantiation of this framework for the domain of intervals.

Value-based Refinement for Intervals

We have created a preliminary instantiation of this framework for the domain of intervals.

Decision:
Choose an initial assignment for all variables

Value-based Refinement for Intervals

We have created a preliminary instantiation of this framework for the domain of intervals.

Decision:
Choose an initial assignment for all variables

Propagation:
Compute forward interpretation for this initial value

Value-based Refinement for Intervals

We have created a preliminary instantiation of this framework for the domain of intervals.

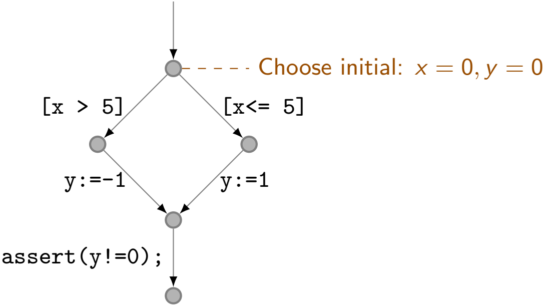
Decision:
Choose an initial assignment for all variables

Propagation:
Compute forward interpretation for this initial value

Generalization and Learning:
Generalize the result by locally generalizing intervals. Remove generalized initial values from selection pool

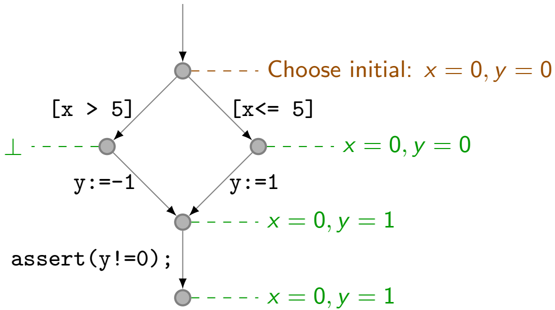
Example 1

Decision



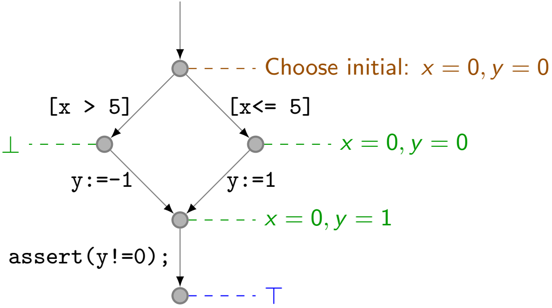
Example 1

Propagation



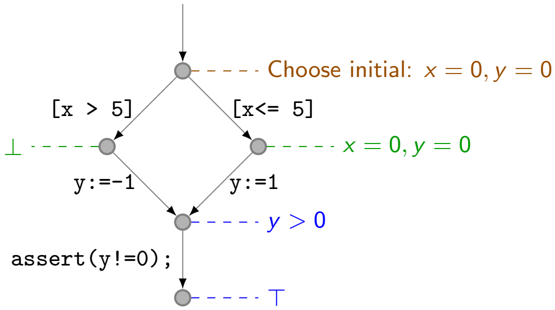
Example 1

Generalization



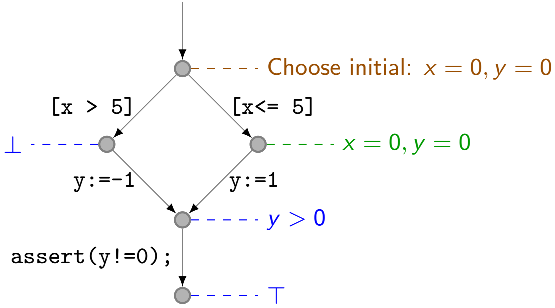
Example 1

Generalization



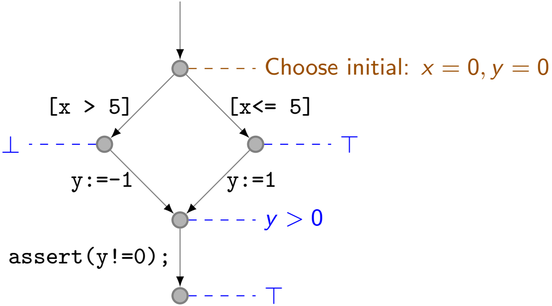
Example 1

Generalization



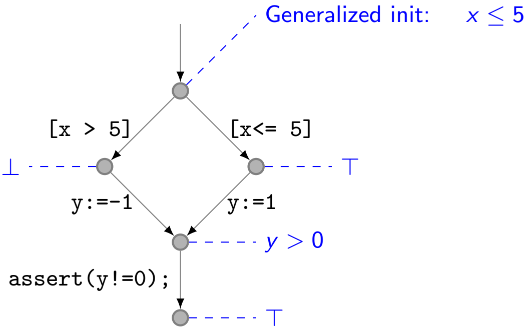
Example 1

Generalization



Example 1

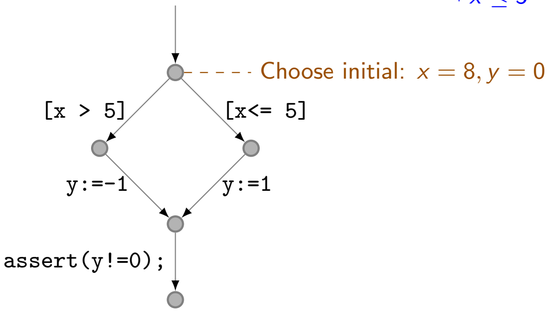
Generalization



Example 1

Decision

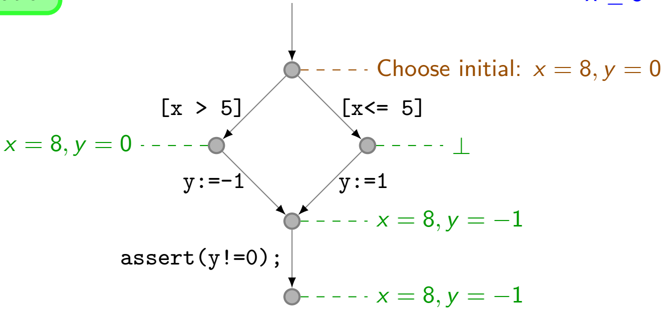
$\neg x \leq 5$



Example 1

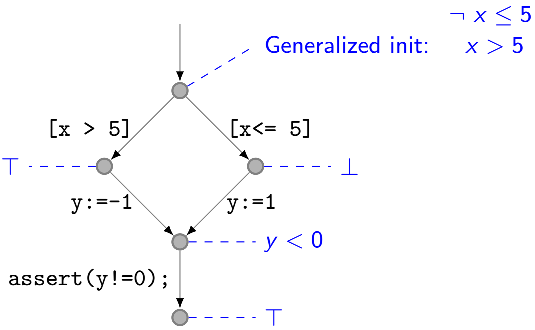
Propagation

$\neg\ x \leq 5$



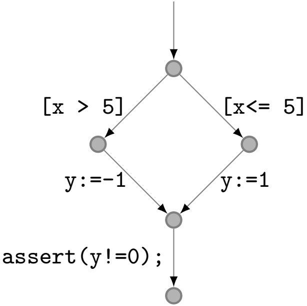
Example 1

Generalization



Example 1

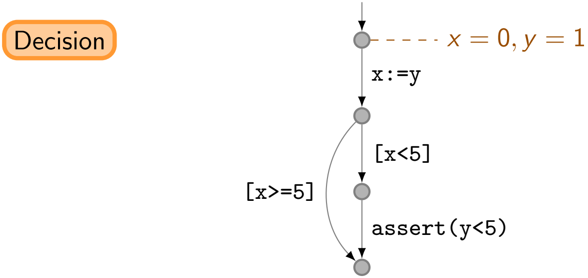
Generalization



$\neg\ x \leq 5$

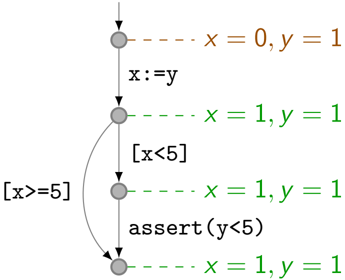
$\neg\ x > 5$

Example 2



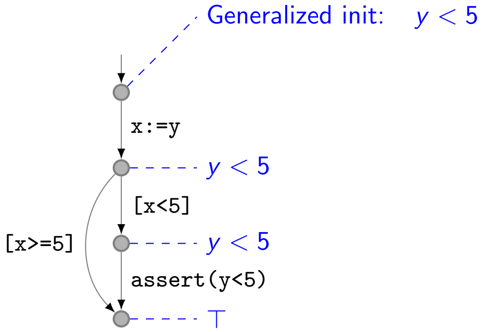
Example 2

Propagation



Example 2

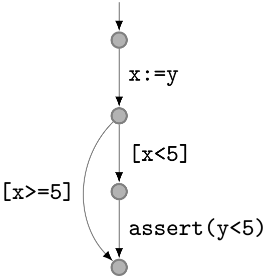
Generalization



Example 2

$\neg y < 5$

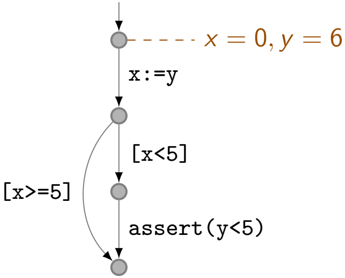
Generalization



Example 2

$\neg y < 5$

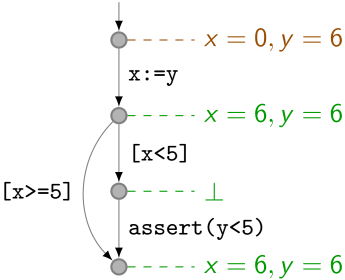
Decision



Example 2

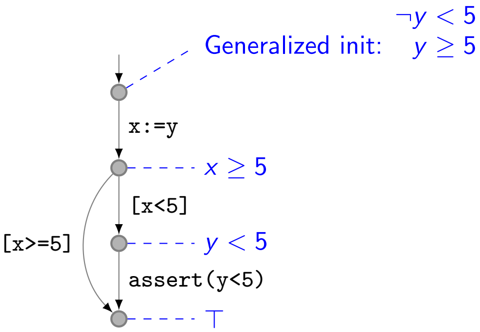
$\neg y < 5$

Propagation



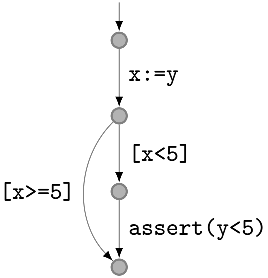
Example 2

Generalization



Example 2

$\neg y < 5$
$\neg y \geq 5$



Notes on Implementation

- Initial values chosen by call to a SAT solver.

Notes on Implementation

- Initial values chosen by call to a SAT solver.
- Generalization uses local repair (SMPP):
  - Set every location to $\top$
  - For each invalid triple $\{pre\}$ `stmt` $\{post\}$
    - repair with $\{pre\}$ from forward analysis.
    - generalize using search on bounds.

Notes on Implementation

- Initial values chosen by call to a SAT solver.
- Generalization uses local repair (SMPP):
  - Set every location to $\top$
  - For each invalid triple $\{pre\}$ `stmt` $\{post\}$
    - repair with $\{pre\}$ from forward analysis.
    - generalize using search on bounds.
- Generalization step:

$0 \leq a \leq 5, b > 5, c < 10$ Repair using SAT solver

```
assert(a <= 10 || a >= -10)
```

$b > 5$

Notes on Implementation

- Initial values chosen by call to a SAT solver.
- Generalization uses local repair (SMPP):
  - Set every location to $\top$
  - For each invalid triple $\{pre\}$ `stmt` $\{post\}$
    - repair with $\{pre\}$ from forward analysis.
    - generalize using search on bounds.
- Generalization step:

$0 \leq a \leq 5, b > 5, c < 10$	Repair using SAT solver
`assert(a <= 10 || a >= -10)`	Increase bounds by search
$b > 5$	

Notes on Implementation

- Initial values chosen by call to a SAT solver.
- Generalization uses local repair (SMPP):
  - Set every location to $\top$
  - For each invalid triple $\{pre\}$ `stmt` $\{post\}$
    - repair with $\{pre\}$ from forward analysis.
    - generalize using search on bounds.
- Generalization step:

$0 \leq a \leq \infty, b > 5, c < 10$	Repair using SAT solver
`assert(a <= 10 || a >= -10)`	Increase bounds by search
$b > 5$	

Notes on Implementation

- Initial values chosen by call to a SAT solver.
- Generalization uses local repair (SMPP):
  - Set every location to $\top$
  - For each invalid triple $\{pre\}$ `stmt` $\{post\}$
    - repair with $\{pre\}$ from forward analysis.
    - generalize using search on bounds.
- Generalization step:

$0 \leq a \leq \infty, b > 5, c < 10$	Repair using SAT solver
`assert(a <= 10 || a >= -10)`	Increase bounds by search
$b > 5$	

Notes on Implementation

- Initial values chosen by call to a SAT solver.
- Generalization uses local repair (SMPP):
  - Set every location to $\top$
  - For each invalid triple $\{pre\}$ `stmt` $\{post\}$
    - repair with $\{pre\}$ from forward analysis.
    - generalize using search on bounds.
- Generalization step:

$-10 \leq a \leq \infty, b > 5, c < 10$	Repair using SAT solver
`assert(a <= 10 || a >= -10)`	Increase bounds by search
$b > 5$	

Preliminary benchmarks

- Selection of *NEC Small Static Analysis Benchmarks* (slightly modified)
- Interval analysis too imprecise in all cases

Preliminary benchmarks

- Selection of *NEC Small Static Analysis Benchmarks* (slightly modified)
- Interval analysis too imprecise in all cases

Inst.	**# paths (SCC-decomp.)**	**runtime (s)**	**iterations**
inf1.c	36	*	*
inf2.c	12	0.7	5
inf3.c	16	0.9	4
inf4.c	1080	*	*
inf5.c	28	2.1	19
inf6.c	32	0.9	4
inf7.c	27	1.7	7
inf8.c	40	3.3	9

Preliminary benchmarks

- Selection of *NEC Small Static Analysis Benchmarks* (slightly modified)
- Interval analysis too imprecise in all cases

Inst.	# paths (SCC-decomp.)	runtime (s)	iterations
inf1.c	36	*	*
inf2.c	12	0.7	5
inf3.c	16	0.9	4
inf4.c	1080	*	*
inf5.c	28	2.1	19
inf6.c	32	0.9	4
inf7.c	27	1.7	7
inf8.c	40	3.3	9

- Does not work if fully relational information is required (inf1.c,inf4.c)

```
assume(x > y);
assert(x > y);
```

Current Work

- Extending the prototype into a tool

Current Work

- Extending the prototype into a tool
- Move towards a fully SAT-style analyzer

Current Work

- Extending the prototype into a tool
- Move towards a fully SAT-style analyzer
- Handling of floating-point numbers

Current Work

- Extending the prototype into a tool
- Move towards a fully SAT-style analyzer
- Handling of floating-point numbers
- Move to more powerful domains

Current Work

- Extending the prototype into a tool
- Move towards a fully SAT-style analyzer
- Handling of floating-point numbers
- Move to more powerful domains
- Use trace partitioning and SMT/SAT-style analysis as “glue“ to combine a static analyzer with a bounded model checker.