A Scalable Segmented Decision Tree Abstract Domain

Dedicated to Amir

Patrick Cousot
CIMS-NYU & ENS

joint work with Radhia Cousot and Laurent Mauborgne
ENS & CNRS IMDEA & ENS

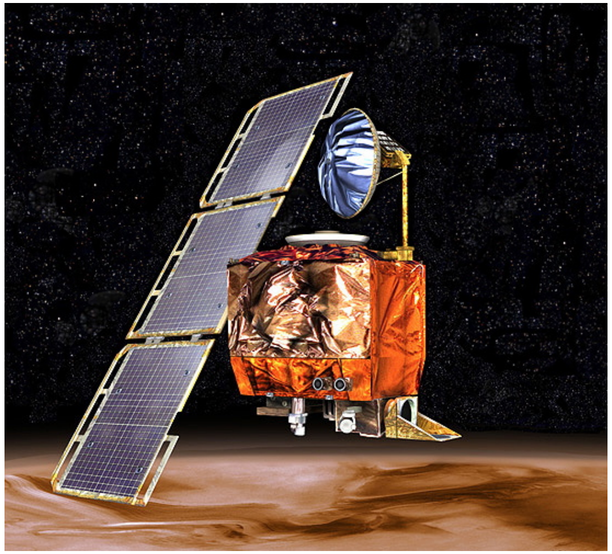
Amir Pnueli Memorial Symposium
CIMS, NYU, New York, May 7—9, 2010

Motivation

Computer scientists have made great contributions to the failure of complex systems

Ariane 5.01 failure
(overflow)

Patriot failure
(float rounding)



Mars orbiter loss
(unit error)

- On-board checking the presence of bugs is great!
- Proving their absence automatically by static analysis is even better!!!

Static analysis

- Automatic static analysis is extremely easy, but for several serious problems:
  - Formally defining the semantics of programming languages and machines
  - Minimizing efforts of developers and end-users
  - Scaling up with enough precision

Making static analysis very easy

- Choose a simple semantic model (e.g. transition systems)
- Choose a uniform representation of properties (e.g. *terms* in deductive methods, *BDDs* in model-checking)
- Problems:
  - Manuel assistance, and/or
  - Combinatorial explosion, and/or
  - Non-termination, and/or
  - Unsoundness, and/or
  - Imprecision (models are not programs)

Origin of the combinatorial explosion: disjunctions

- We have to compute iteratively

$$\mathbf{lfp}_{\perp}^{\sqsubseteq} F$$

where $\quad F \triangleq \bigsqcup_{i \in \Delta} F_i \quad$ is continuous on a cpo

that is

$$\mathbf{lfp}_{\perp}^{\sqsubseteq} F = X^{\omega} = \bigsqcup_{n \geqslant 0} X^n = \underbrace{\bigsqcup_{n \geqslant 0} \bigsqcup_{i_1, \ldots, i_n \in \Delta^n}}_{\text{combinatorial explosion!}} F_{i_1} \circ \ldots \circ F_{i_n}(\perp)$$

Abstract interpretation

- Sound approximations of disjunctions (Galois connection, widening/narrowing, etc)
- Abstract domains (efficient machine representation of a class of abstract program properties & efficient algorithms for implementing abstract operations and transformers)
- Abstract domain functors for combining abstract domains (e.g. reduced product, reduced cardinal power, etc)

Contribution

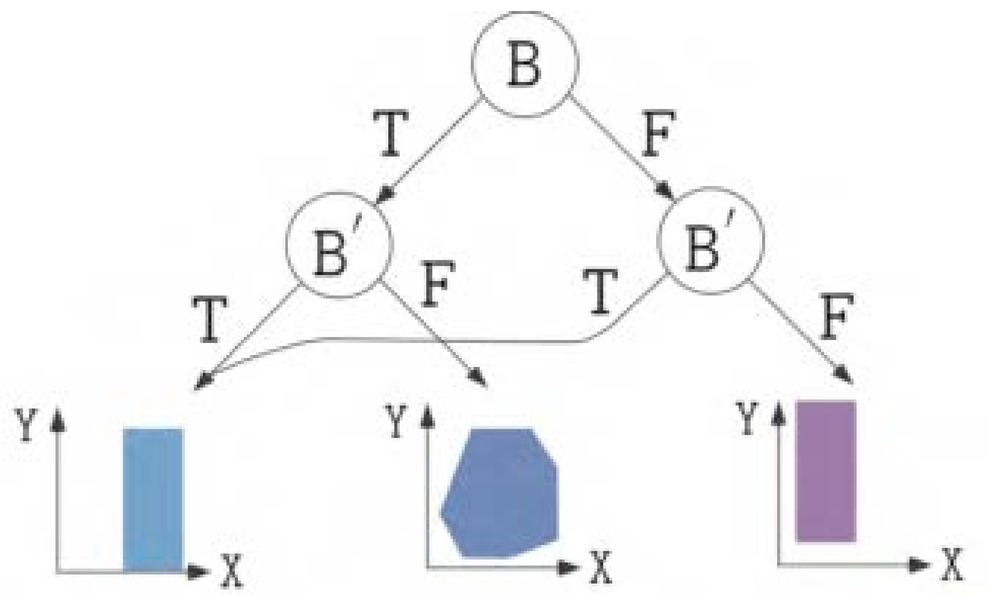
Segmented decision tree functor

- A new abstract domain functor generalizing
  - The binary decision tree functor (L. Mauborgne)
  - The array segmentation functor (P. Cousot, R. Cousot & F. Logozzo)

to approximate disjunctions efficiently with reasonable expressivity

The binary decision tree functor

```
/* boolean.c */
typedef enum {F=0,T=1} BOOL;
BOOL B;
void main () {
  unsigned int X, Y;
  while (1) {
    ...
    B = (X == 0);
    ...
    if (!B) {
      Y = 1 / X;
    }
    ...
  }
}
```



The boolean decision tree abstract domain functor is parameterized by the maximal height of the decision tree (an analyzer option) and the abstract domain at the leaves

Implemented in Astrée, http://www.astree.ens.fr/, http://www.absint.com/astree/

The array segmentation functor

```
          int n = 10;
          int i, A[n];
          i = 0;
/* 1: */
          while /* 2: */ (i < n) {
             p2 = [ A: <{0} [0,0] {i}? [-oo,+oo] {n,10}?>
                     i: [0,+oo] n: [10,10] ]
/* 3: */
             A[i] = 0;
/* 4: */
             i = i + 1;
/* 5: */
          }
/* 6: */
p6 = [ A: <{0} [0,0] {n,10,i}> i: [10 +oo] n: [10,10] ]
```

Array A is initialized to 0

Loop invariant at /* 2 */:

if `i = 0;` then
block is empty (so array A is not initialized)
else if `i` > 0 then
A[0] = ... = A[`i`-1] = 0
else (* `i` < 0 *)
Impossible

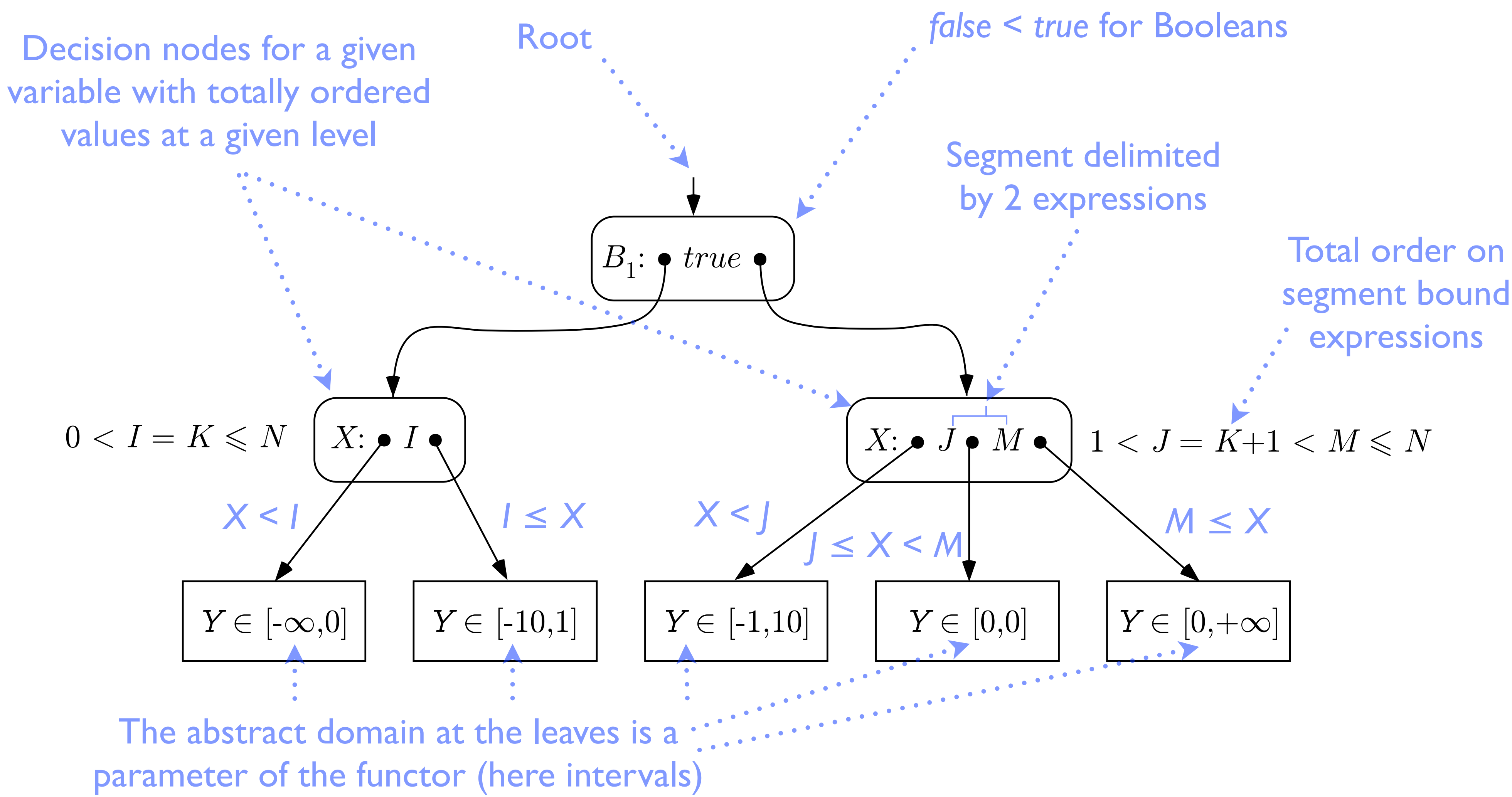
Implemented in Clousot, http://research.microsoft.com/apps/pubs/default.aspx?id=70614

The segmented decision tree functor

The segmented decision tree functor

1) Abstract properties

An example of segmented decision tree



This segmented decision tree encodes the fact that if B_1 is false (i.e. $B_1 < true$) then if $X < I$ then Y is non-positive while if $X \geqslant I$ then $-10 \leqslant Y \leqslant 1$. Similarly, if B_1 is true (i.e. $B_1 \geqslant true$) then either $X < J$ and $-1 \leqslant Y \leqslant 10$, or $J \leqslant X < M$ and Y is null, or $X > M$ and Y is non-negative.

The segmented decision tree abstract functor

$$\mathbb{T}((\mathbb{D}, <_{\mathbb{D}}), \mathbb{E}, D_c, D_\ell)$$

Domain of totally ordered variables

Domain of canonical bound expressions

Domain of ordering side conditions

Domain of leaves

Controling costs

- The time and memory cost of relational abstract domains grows polynomially/exponentially in the number *n* of variables
- For segmented decision trees:
  - Limit the bound expressions to a simple canonical form (e.g. octagons)
  - Limit the height of trees (e.g. 3/4)
  - Variable packing (*) for side expressions

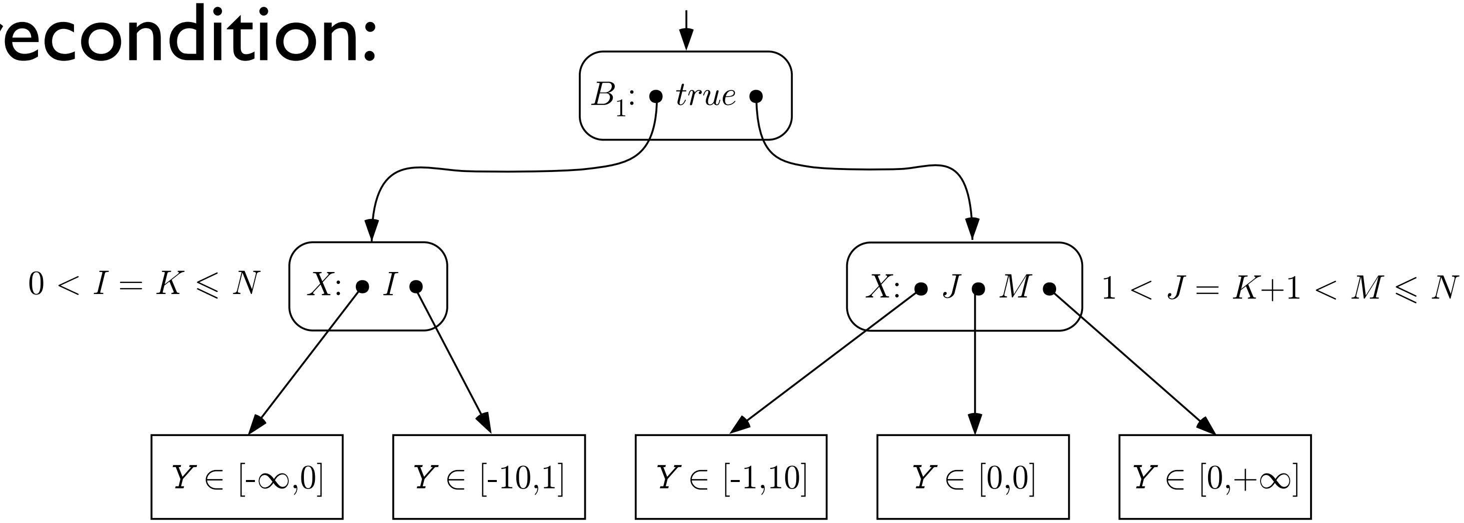
(*) a simple and cheap pre-analysis that groups interdependant variables into packs, leaving unrelated variables in separate packs

The segmented decision tree functor

II) Abstract operations

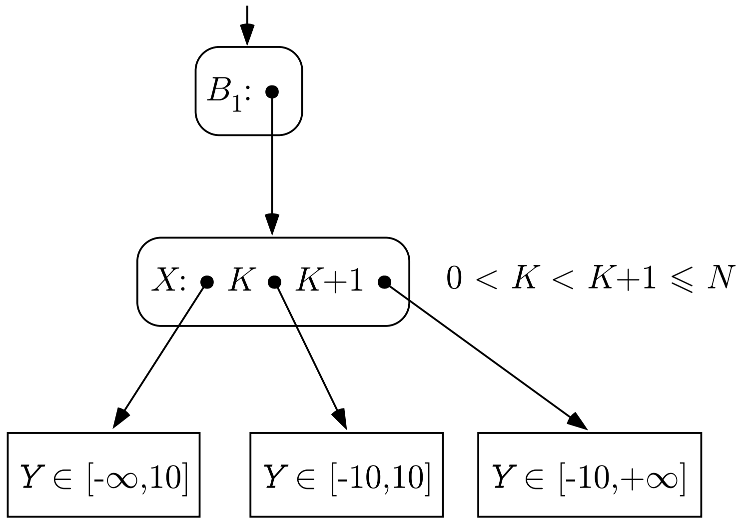
Segment unification

- Abstract precondition:

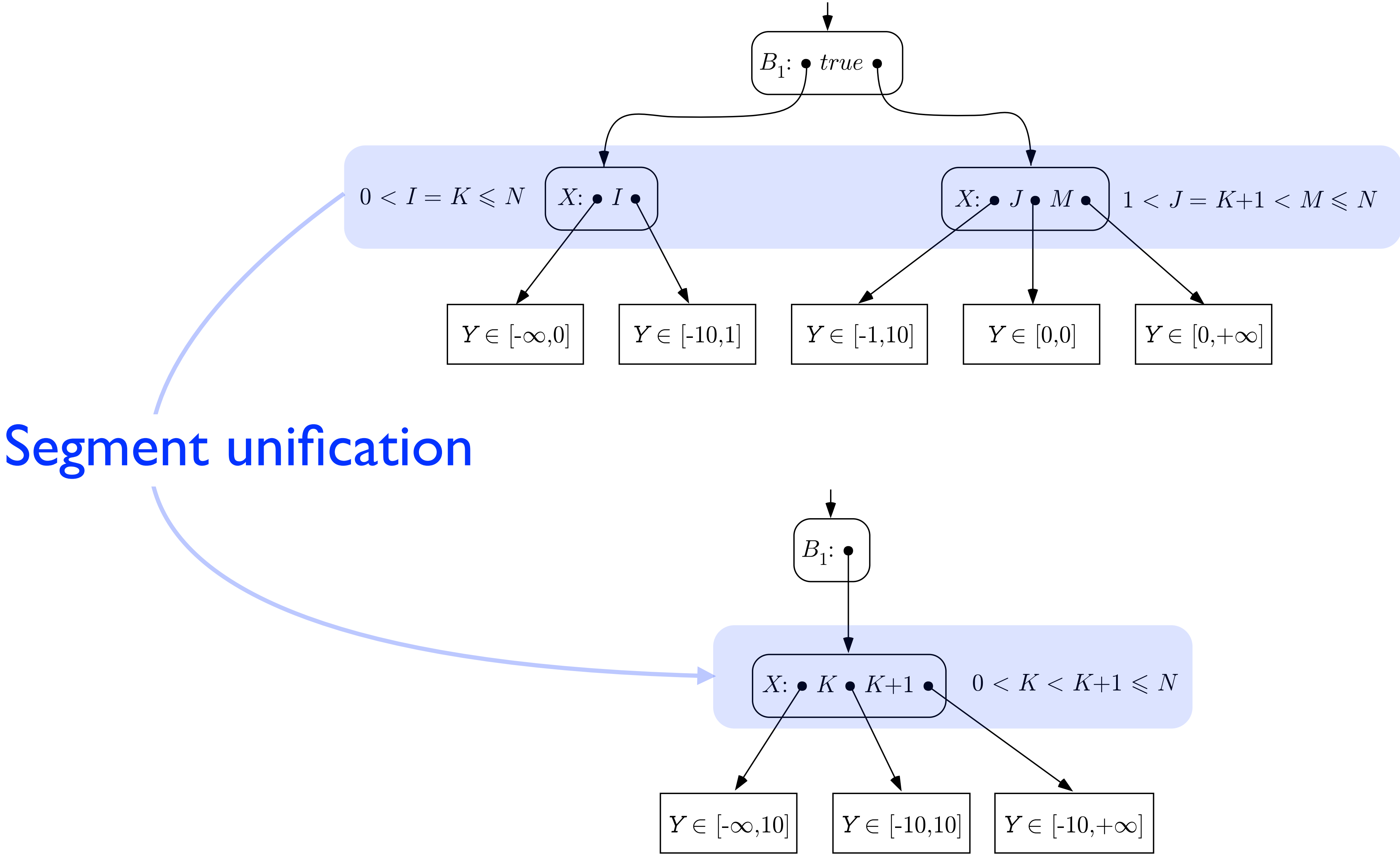


- Assignment: B_1 = ?

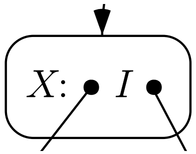
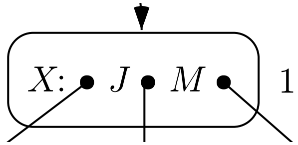
- Abstract postcondition:



Segment unification (cont'd)



- Given two segments to unify:

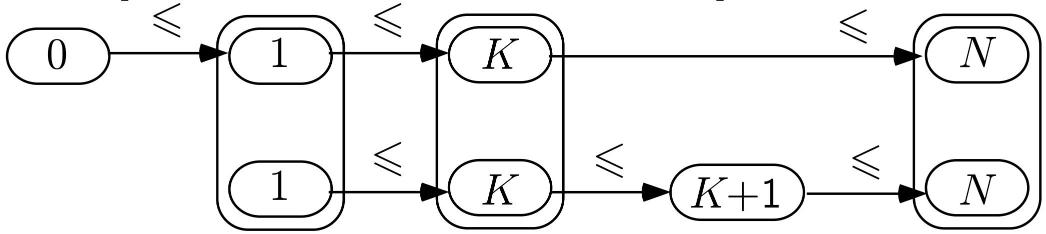
$0 < I = K \leqslant N$  and  $1 < J = K{+}1 < M \leqslant N$

- Build pre-orders with bounds and side conditions

$$0 \xrightarrow{\leqslant} 1 \xrightarrow{\leqslant} I, K \xrightarrow{\leqslant} N$$

$$1 \xrightarrow{\leqslant} J\text{-}1, K \xrightarrow{\leqslant} J, K{+}1 \xrightarrow{\leqslant} M\text{-}1 \xrightarrow{\leqslant} M \xrightarrow{\leqslant} N$$

- Eliminate expressions not comparable in both pre-orders:



- Choose a maximal chain (valid in both pre-orders)

$$0 \xrightarrow{\leqslant} 1 \xrightarrow{\leqslant} K \xrightarrow{\leqslant} K{+}1 \xrightarrow{\leqslant} N$$

- Keep representatives of bounds in either segment

$$\xrightarrow{\leqslant} K \xrightarrow{\leqslant} K{+}1 \xrightarrow{\leqslant} \qquad \big(0,\ 1 \text{ and } N \text{ did not appear}\big)$$

- Merge the corresponding sub-trees

Union, intersection, comparison, etc

- Unify segmentations
- Perform operation segmentwise at the leaves

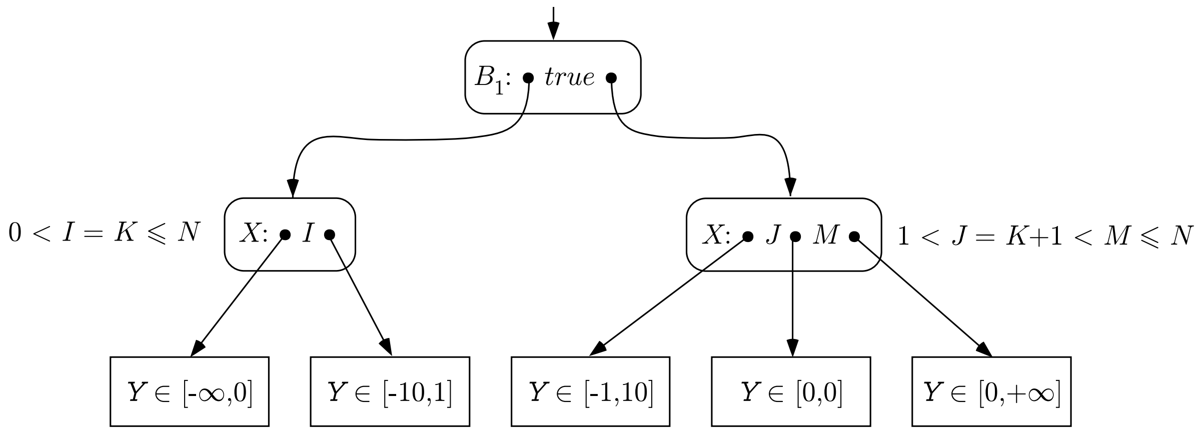
Widening

- Unify segments using only common expressions in both segmentations
- Use the side-conditions and leave abstract domain widenings
- The number of expressions in segmentations can only decrease and each segment is widened

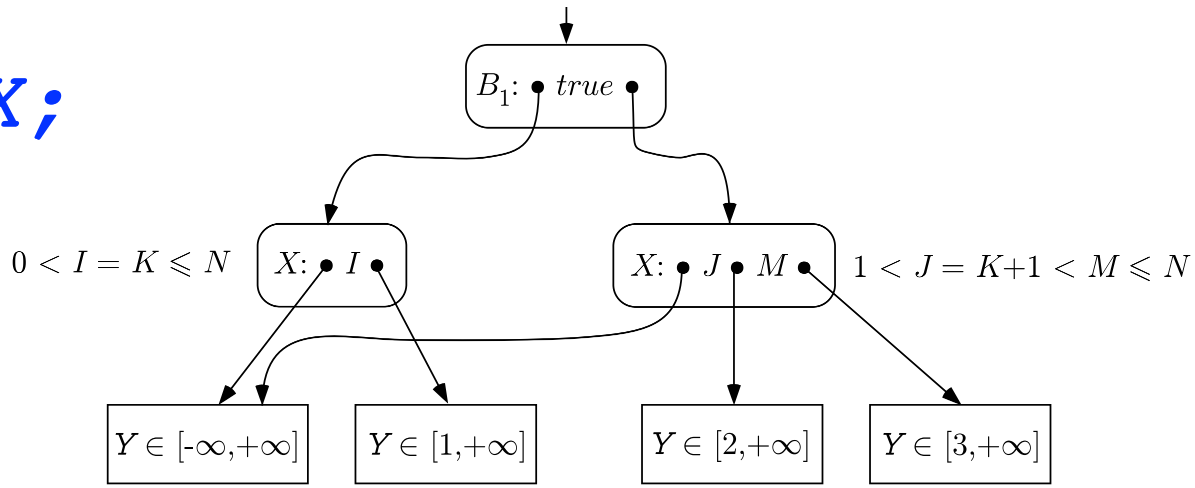
$\Rightarrow$ termination

Assignment to leave variables

- Determine the feasible paths
- Perform assignments at the leaves (opportunistic sharing)




```
Y = X;
```

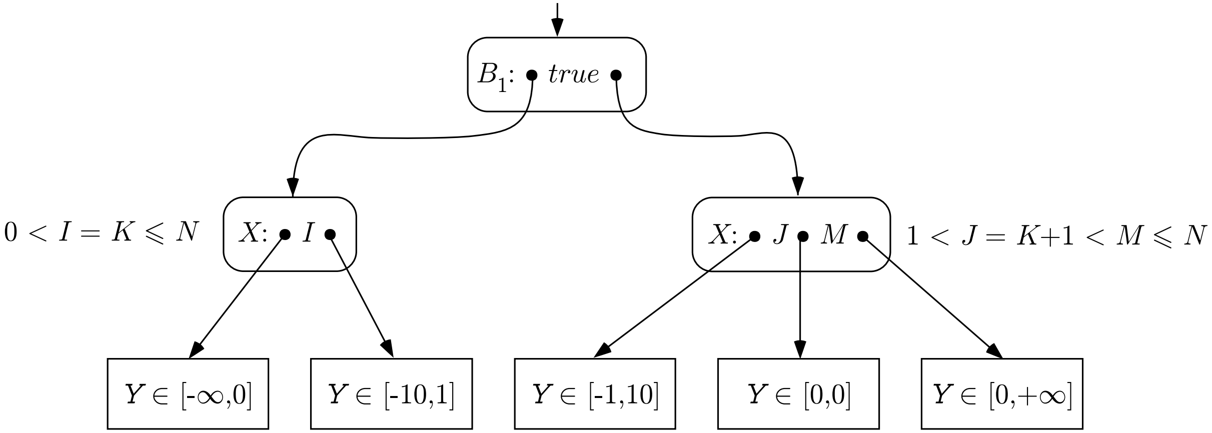


Assignments to variables in segment bounds

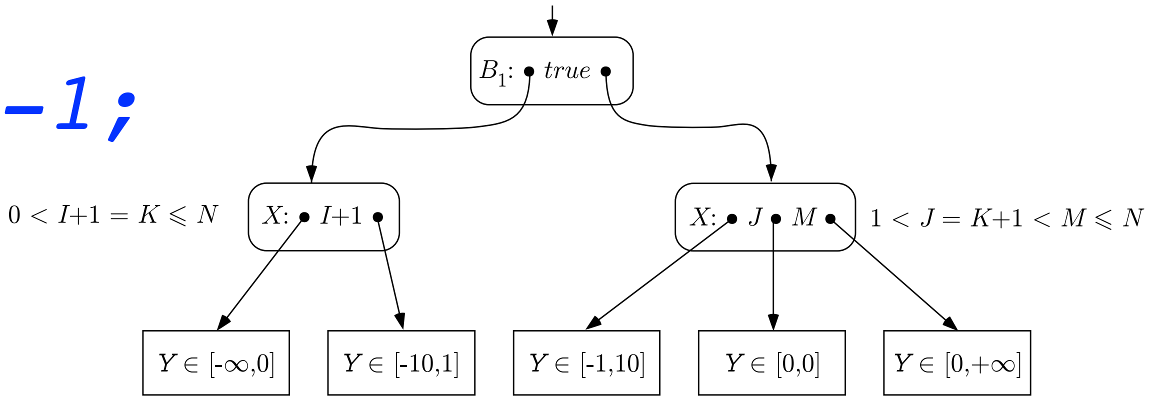
- Invertible assignments:

  $i' = f(i) \Rightarrow i = f^{-1}(i')$

- Replace i by $f^{-1}(i)$ in each segment bound expression (and side conditions)



`I = I-1;`



Assignments to variables in segment bounds

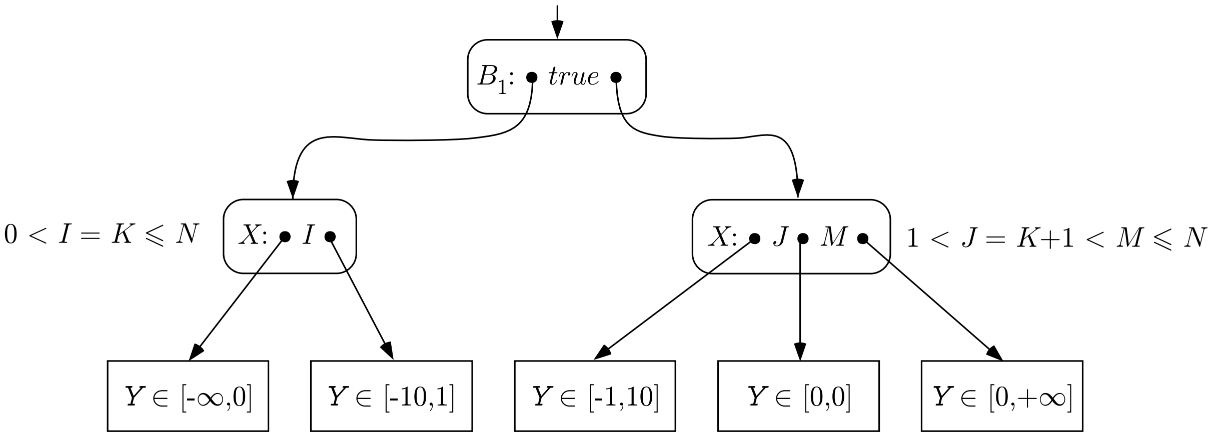
- Non-invertible assignments:
  - Replace expressions with that variable by an equal one, if any in side condition
  - Otherwise eliminate the segment bounds and merge segments
  - Take assignment into account in side conditions

Assignments to decision variables

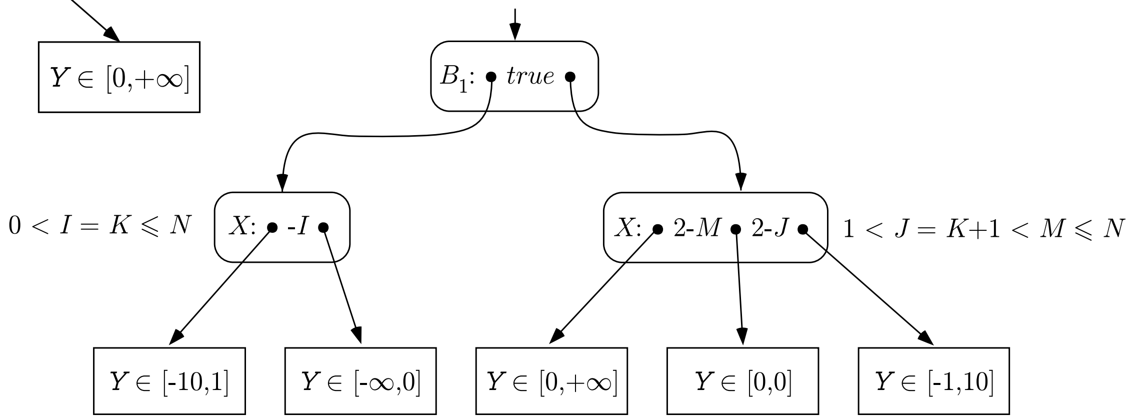
- Invertible assignments:

  $$x' = f(x) \Rightarrow x = f^{-1}(x')$$

- Replace $e \leq x$ by $e \leq f^{-1}(x')$ that is
  - $f(e) \leq x'$ when f increasing
  - $f(e) \geq x'$ when f decreasing

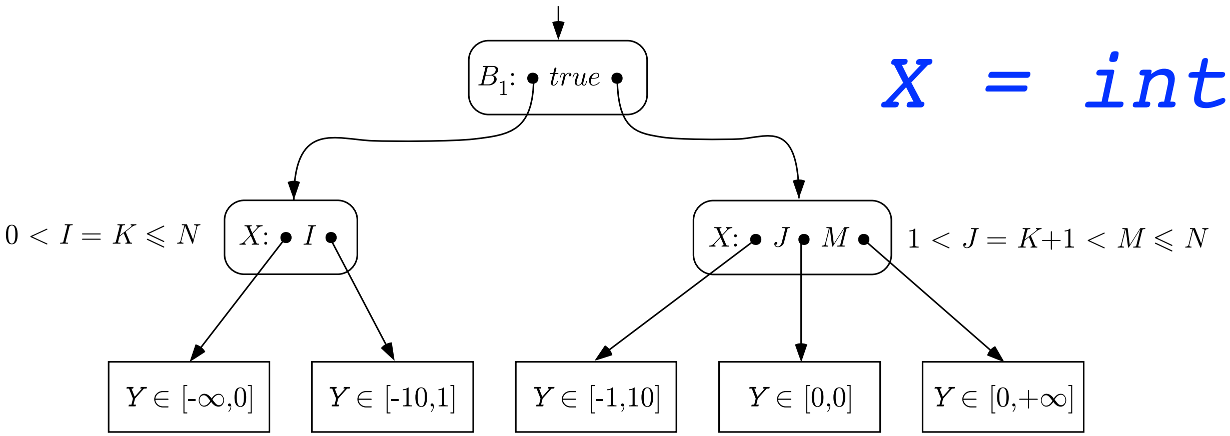


`X = int(B1)-X;`

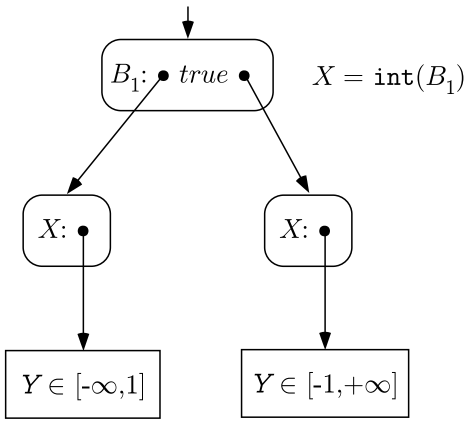


Assignments to decision variables

- Non-invertible assignment:
  - merge segments related to assigned variable
  - possible preserve information in side-conditions




```
X = int(B1);
```



Abstracting functions and arrays

- `f(x1,...,xn)` : values at leaves are function of side-conditions on decision variables `x1,...,xn`

$\sin x,\ x \in [0, 2\pi]$ **is** $[\![\, x\,\{0 \leqslant x \leqslant 2\pi\} : \ (\!|\sin x : [0,1]\,|\!) \ \pi \ (\!|\sin x : [-1,0]\,|\!) \,]\!]$

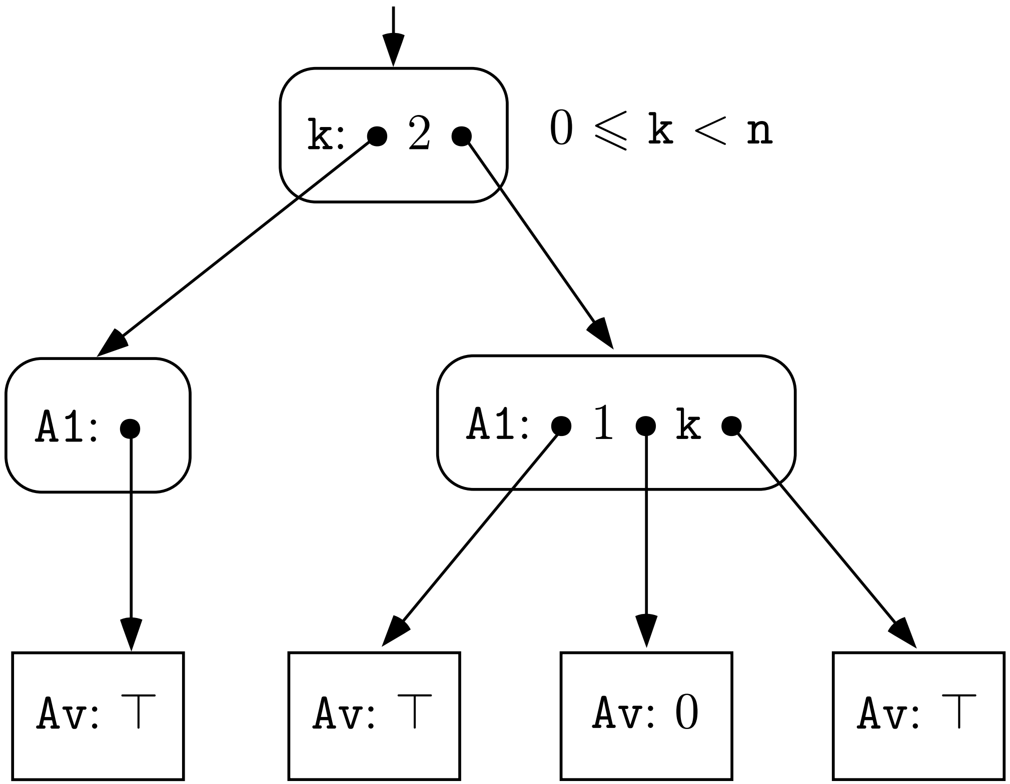
- Arrays `A` map the indexes (denoted `Ai` for dimension `i`, `i = 1,...,n`) to values (denoted `Av`)

Examples

Partial array initialization

```
          int n; /* n > 0 */
          int k, A[n];
/* 0: */ k = 0;
/* 1: */  while /* 2: */ (k < n) {
/* 3: */     if (k > 0) {
/* 4: */        A[k] = 0;
/* 5: */      };
/* 6: */      k = k+1;
/* 7: */  };
/* 8: */
```

Loop invariant at /* 3 */:



Partial matrix initialization

```
             int m, n; /* m, n > 0 */
             int i, j, M[m,n];
/*  0: */    i = 0;
/*  1: */    while /* 2: */ (i < m) {
/*  3: */       j = i+1;
/*  4: */       while /* 5: */ (j < n) {
/*  6: */          M[i,j] = 0;
/*  7: */          j = j+1;
/*  8: */       };
/*  9: */       i = i+1;
/* 10: */    };
/* 11: */
```

$$\texttt{11:}\quad \llbracket \texttt{M1}\,\{0 < \texttt{m} = \texttt{i}\} : \llbracket \texttt{M2} : \ (\!| \texttt{Mv} : \top |\!) \ \texttt{M1} + 1 \ (\!| \texttt{Mv} : 0 |\!) \ \rrbracket \rrbracket$$

⊤	0	0	0	0	0	0	0	0	0
⊤	⊤	0	0	0	0	0	0	0	0
⊤	⊤	⊤	0	0	0	0	0	0	0
⊤	⊤	⊤	⊤	0	0	0	0	0	0
⊤	⊤	⊤	⊤	⊤	0	0	0	0	0
⊤	⊤	⊤	⊤	⊤	⊤	0	0	0	0

The analysis computation is automatic, precise and efficient

0: $\llbracket \mathtt{M1} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket \rrbracket$ ≀program precondition: i, j, and A uninitialized∫

ℓ: $\bot$ ≀$\ell = 1, \ldots, 11$, infimum∫

1:,2:,3: $\llbracket \mathtt{M1}\,\{\mathtt{i} = 0\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket \rrbracket$ ≀0: with $\mathtt{i} = 0$, $\mathtt{i} < \mathtt{m}^8$∫

4:,5:,6: $\llbracket \mathtt{M1}\,\{\mathtt{i} = 0, \mathtt{j} = \mathtt{i}+1 = 1 < \mathtt{n}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket \rrbracket$

≀3: with j = i+1;, $\mathtt{j} < \mathtt{n}$∫

7: $\llbracket \mathtt{M1}\,\{\mathtt{i} = 0, \mathtt{j} = \mathtt{i}+1 = 1 < \mathtt{n}\} :$
$\llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ \mathtt{j}\ (\!| \mathtt{Mv} : 0 |\!)\ \mathtt{j}+1\ (\!| \mathtt{Mv} : \top |\!) \rrbracket\ \mathtt{i}+1\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket$
$\rrbracket$ ≀6: with M[i,j] = 0;∫

8: $\llbracket \mathtt{M1}\,\{\mathtt{i} = 0, \mathtt{j} = \mathtt{i}+2 = 2 \leqslant \mathtt{n}\} :$
$\llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ \mathtt{j}-1\ (\!| \mathtt{Mv} : 0 |\!)\ \mathtt{j}\ (\!| \mathtt{Mv} : \top |\!) \rrbracket\ \mathtt{i}+1\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket$
$\rrbracket$ ≀7: with j = j+1;∫

4: $\sqcup_t$ 8: $\llbracket \mathtt{M1}\,\{\mathtt{i} = 0, \mathtt{i}+1 \leqslant \mathtt{j} \leqslant \mathtt{i}+2 \leqslant \mathtt{n}\} :$
$\llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!)\ \mathtt{j}\ (\!| \mathtt{Mv} : \top |\!) \rrbracket\ \mathtt{i}+1\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket$
$\rrbracket$ ≀join of 4: and 8:∫

5: $\llbracket \mathtt{M1}\,\{\mathtt{i} = 0, \mathtt{i}+1 \leqslant \mathtt{j} \leqslant \mathtt{n}\} :$
$\llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!)\ \mathtt{j}\ (\!| \mathtt{Mv} : \top |\!) \rrbracket\ \mathtt{i}+1\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket$
$\rrbracket$ ≀5: ∇ (4: $\sqcup_t$ 8:)^9∫

9: $\llbracket \mathtt{M1}\,\{\mathtt{i} = 0, \mathtt{i}+1 \leqslant \mathtt{j} = \mathtt{n}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!)\ \mathtt{i}+1$
$\llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket \rrbracket$ ≀5: and $\mathtt{j} \geqslant \mathtt{n}$∫

10: $\llbracket \mathtt{M1}\,\{\mathtt{i} = 1, \mathtt{i} \leqslant \mathtt{j} = \mathtt{n}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket \rrbracket$
≀9: and i = i+1;∫

1: $\sqcup_t$ 10: $\llbracket \mathtt{M1}\,\{\mathtt{i} = 1, \mathtt{i} \leqslant \mathtt{j} = \mathtt{n}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket \rrbracket$
≀join of 1: and 10:∫

2: $\llbracket \mathtt{M1}\,\{0 \leqslant \mathtt{i}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket \rrbracket$
≀2: ∇ (1: $\sqcup_t$ 10:)∫

3: $\llbracket \mathtt{M1}\,\{0 \leqslant \mathtt{i} < \mathtt{m}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket \rrbracket$
≀2: and j < n∫

4:,5:,6: $\llbracket \mathtt{M1}\,\{0 \leqslant \mathtt{i} < \mathtt{m}, \mathtt{j} = \mathtt{i}+1 < \mathtt{n}\} :$
$\llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket$
$\rrbracket$ ≀3:, j = i+1; and j < n∫

7: $\llbracket \mathtt{M1}\,\{0 \leqslant \mathtt{i} < \mathtt{m}, \mathtt{j} = \mathtt{i}+1 < \mathtt{n}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}$
$\llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ \mathtt{j}\ (\!| \mathtt{Mv} : 0 |\!)\ \mathtt{j}+1\ (\!| \mathtt{Mv} : \top |\!) \rrbracket\ \mathtt{i}+1\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket$
$\rrbracket$ ≀6: and M[i, j] = 0;∫

8: $\llbracket \mathtt{M1}\,\{0 \leqslant \mathtt{i} < \mathtt{m}, \mathtt{j} = \mathtt{i}+2 \leqslant \mathtt{n}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!)\ \mathtt{n} \rrbracket\ \mathtt{i}$
$\llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ \mathtt{j}-1\ (\!| \mathtt{Mv} : 0 |\!)\ \mathtt{j}\ (\!| \mathtt{Mv} : \top |\!) \rrbracket\ \mathtt{i}+1\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket$
$\rrbracket$ ≀7: with j = j+1;∫

4: $\sqcup_t$ 8: $\llbracket \mathtt{M1}\,\{0 \leqslant \mathtt{i} < \mathtt{m}, \mathtt{i}+1 \leqslant \mathtt{j} \leqslant \mathtt{i}+2 \leqslant \mathtt{n}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}$
$\llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ \mathtt{i}+1\ (\!| \mathtt{Mv} : 0 |\!)\ \mathtt{j}\ (\!| \mathtt{Mv} : \top |\!) \rrbracket\ \mathtt{i}+1\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket$
$\rrbracket$ ≀join of 4: and 8:∫

5: $\llbracket \mathtt{M1}\,\{0 \leqslant \mathtt{i} < \mathtt{m}, \mathtt{i}+1 \leqslant \mathtt{j} \leqslant \mathtt{n}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}$
$\llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ \mathtt{i}+1\ (\!| \mathtt{Mv} : 0 |\!)\ \mathtt{j}\ (\!| \mathtt{Mv} : \top |\!) \rrbracket\ \mathtt{i}+1\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket$
$\rrbracket$ ≀5: ∇ (4: $\sqcup_t$ 8:)∫

9: $\llbracket \mathtt{M1}\,\{0 \leqslant \mathtt{i} < \mathtt{m}, \mathtt{i}+1 \leqslant \mathtt{j} = \mathtt{n}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}$
$\llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ \mathtt{i}+1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}+1\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket$
$\rrbracket$ ≀5: and $\mathtt{j} \geqslant \mathtt{n}$∫

10: $\llbracket \mathtt{M1}\,\{0 < \mathtt{i} \leqslant \mathtt{m}, \mathtt{i} \leqslant \mathtt{j} = \mathtt{n}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ 1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}-1$
$\llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ \mathtt{i}\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket$
$\rrbracket$ ≀9: and i = i+1;∫

1: $\sqcup_t$ 10: $\llbracket \mathtt{M1}\,\{0 \leqslant \mathtt{i} \leqslant \mathtt{m}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ \mathtt{M1}+1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket \rrbracket$
≀join of 1: and 10:(segments unification yields $1 \leq \mathtt{M1}+1 \leq i$ for subtree merges)∫

2: $\llbracket \mathtt{M1}\,\{0 \leqslant \mathtt{i}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ \mathtt{M1}+1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket\ \mathtt{i}\ \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!) \rrbracket \rrbracket$
≀2: $\sqsubseteq$ (1: $\sqcup_t$ 10:), stabilization at a fixpoint∫

11: $\llbracket \mathtt{M1}\,\{0 < \mathtt{m} = \mathtt{i}\} : \llbracket \mathtt{M2} : (\!| \mathtt{Mv} : \top |\!)\ \mathtt{M1}+1\ (\!| \mathtt{Mv} : 0 |\!) \rrbracket \rrbracket$
≀2: and $\mathtt{i} \geqslant \mathtt{m}$, program postcondition.∫

Conclusion

Abstract domain (functors)

- Abstract domains efficiently encode classes of program properties and operations on these properties
- The approach requires more work than universal representations but is much more efficient
- Abstract domain functors combine abstract domains to produce many instanciated powerful abstract domain at various levels of cost/precision
- Key to scalability with precision in abstract interpretation

The End, Thank You