Generation of Test Data Structures Using Constraint Logic Programming

Valerio Senni[1], Fabio Fioravanti[2]

[1]Department of Computer Science, Systems and Production
University of Rome 'Tor Vergata', Italy
[2]Department of Sciences - University 'G. d'Annunzio' of Chieti-Pescara, Italy

Tests & Proofs 2012

Software testing

- Increasing **confidence** in software correctness

Software testing

- Increasing **confidence** in software correctness
- Focus: how to **generate** input-output pairs

  to check the behavior of a **function/module**

```
Input = generate()
Output = my_function(Input)
if !oracle_accept(Output) then print "bug"
```

Software testing

- Increasing **confidence** in software correctness
- Focus: how to **generate** input-output pairs

  to check the behavior of a **function/module**

```
Input = generate()
Output = my_function(Input)
if !oracle_accept(Output) then print "bug"
```

- **White-box** (code-aware)
  select input-output pairs to **maximize** a code **coverage** criteria
- **Black-box** (model-based)
  select input-output pairs using a given **model** (specification)

Developing Generators for Black-box Testing

Black-box : select input-output pairs w.r.t. a given **model**

(A) **imperative** (direct generation) [e.g., ASTGen]
specifies how to compute tests

PROS

- **fast** (ad-hoc)

CONS

- **high effort** in development
- **error-prone**

(B) **declarative** (filtering-based) [e.g., Korat, TestEra]
specifies which tests to compute

PROS

- **low effort** in development
- **correctness** improved

CONS

- **slow** (generate-and-test)

Bounded-Exhaustive Testing (BET)

Small scope hypothesis [Jackson]:

> *a large portion of faults is likely to be revealed*
> *already by testing all inputs up to a small scope*

BET: testing software for **all valid inputs** up to a given **size bound**

Challenge: efficient & correct generators of complex data structures

Applications: testing of

- data-structures manipulating programs
- web-based applications (accepting XML, PHP, HTML, . . . inputs)
- . . .

Constraint Logic Programming -based Generation

Constraint Logic Programming :

- **declarative** programming
- **constraint** solving
- **symbolic** execution
- checkpoint-based exploration (**backtracking** + **unification**)

} efficient & correct generators of complex data structures

- Constraint-based
  [DeMillo-Offutt '91, Meudec (ATGen) '01, Euclide '09]
- Constraint Logic Programming -based
  [PathCrawler '05, Charreteur-Botella-Gotlieb '01, jPET '11]

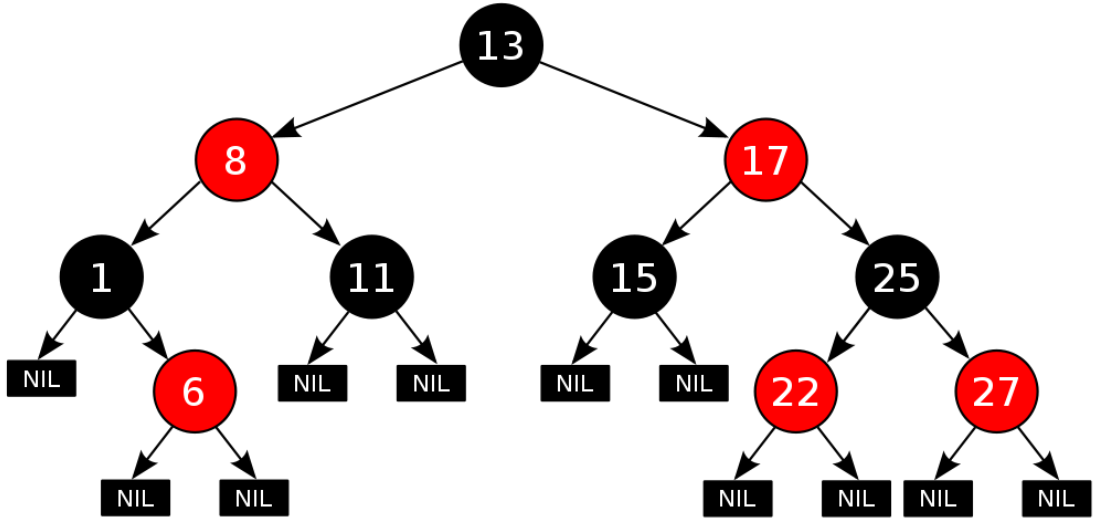
Plan

1. Running Example and Korat
2. Use of the Constrain-and-Generate Paradigm
3. Optimizations

Red-Black Tree

Nodes are:

- colored (red or black)
- marked (by a key)



A binary tree satisfying the following invariants:

(I_1) **red** nodes have **black children**

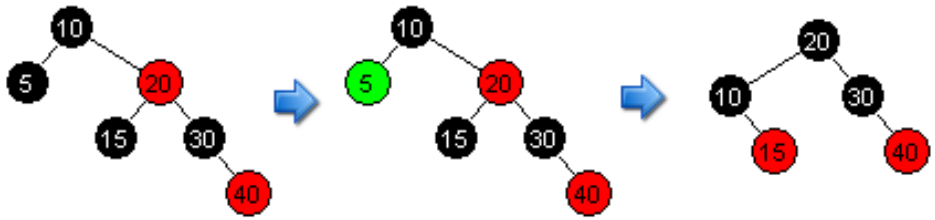
(I_2) every **root-to-leaf path** has the same **number of black nodes**

(I_3) keys are **ordered** left-to-right

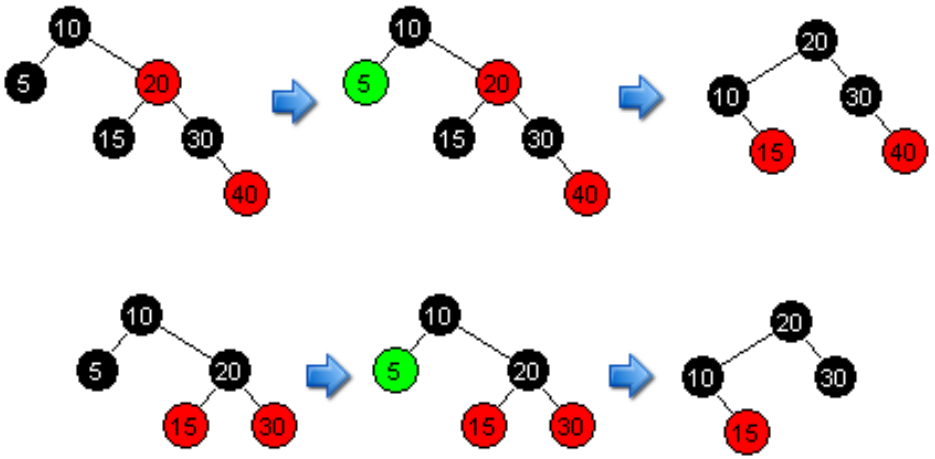
Efficient worst-case insertion/deletion/retrieval

Red-Black Tree - Node Removal

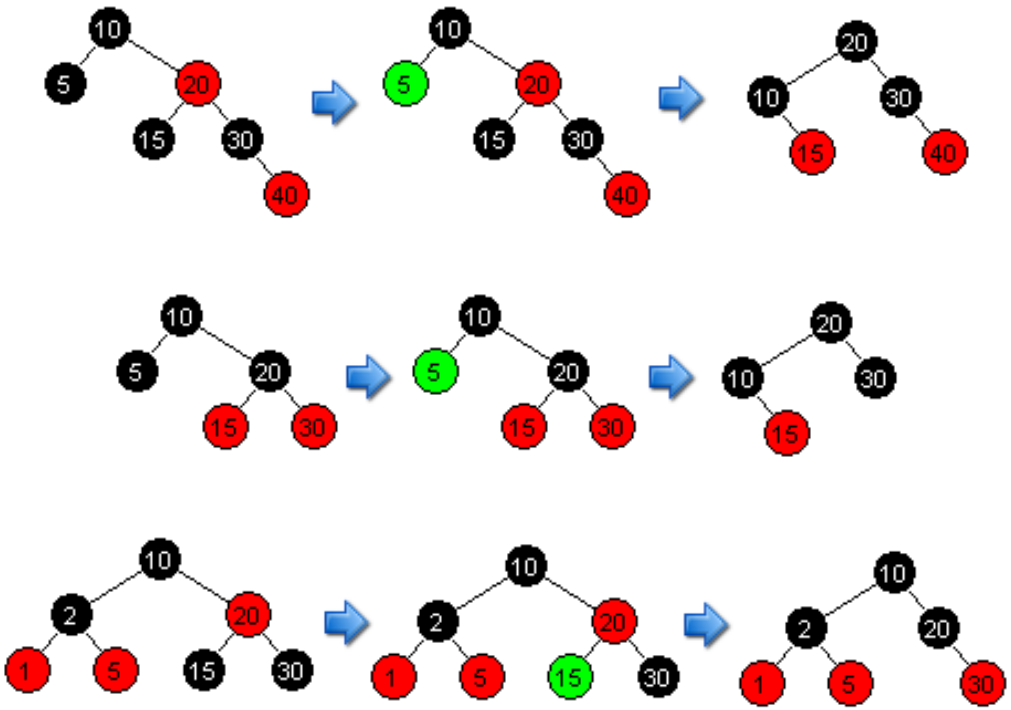
Red-Black Tree - Node Removal



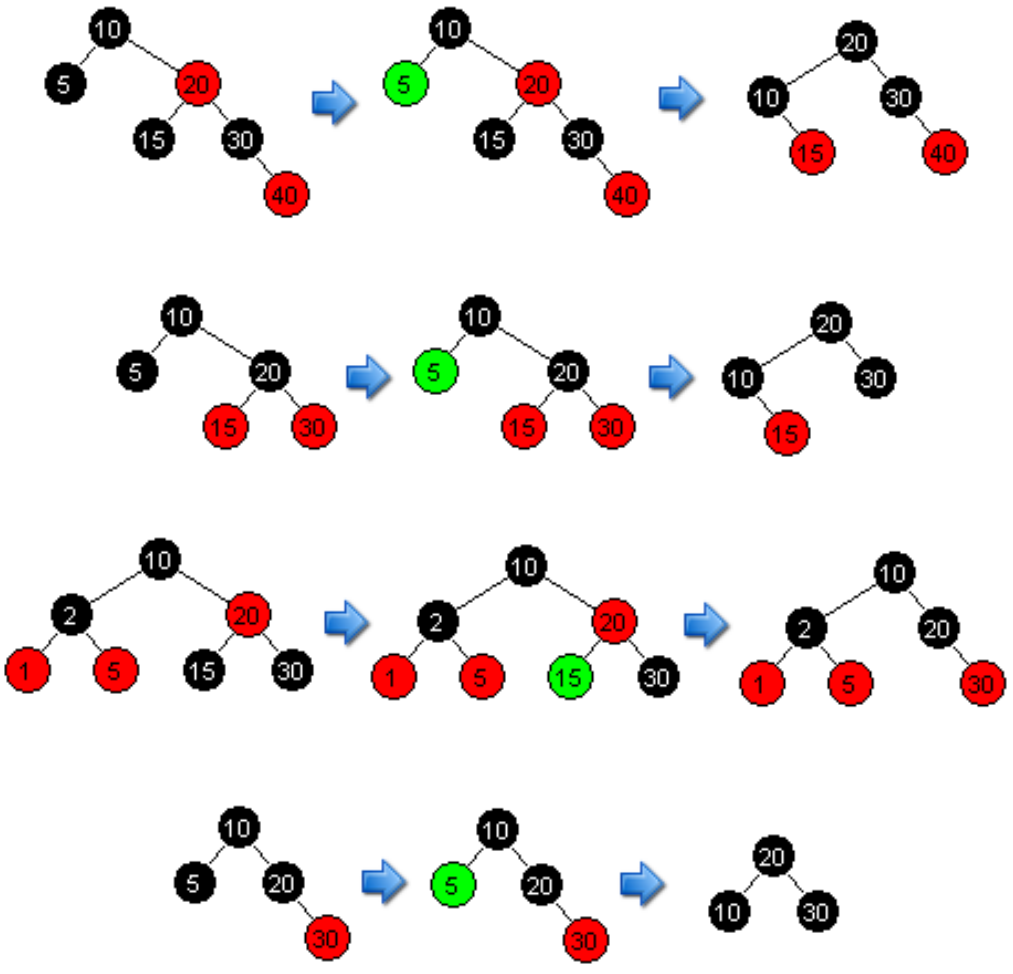
Red-Black Tree - Node Removal



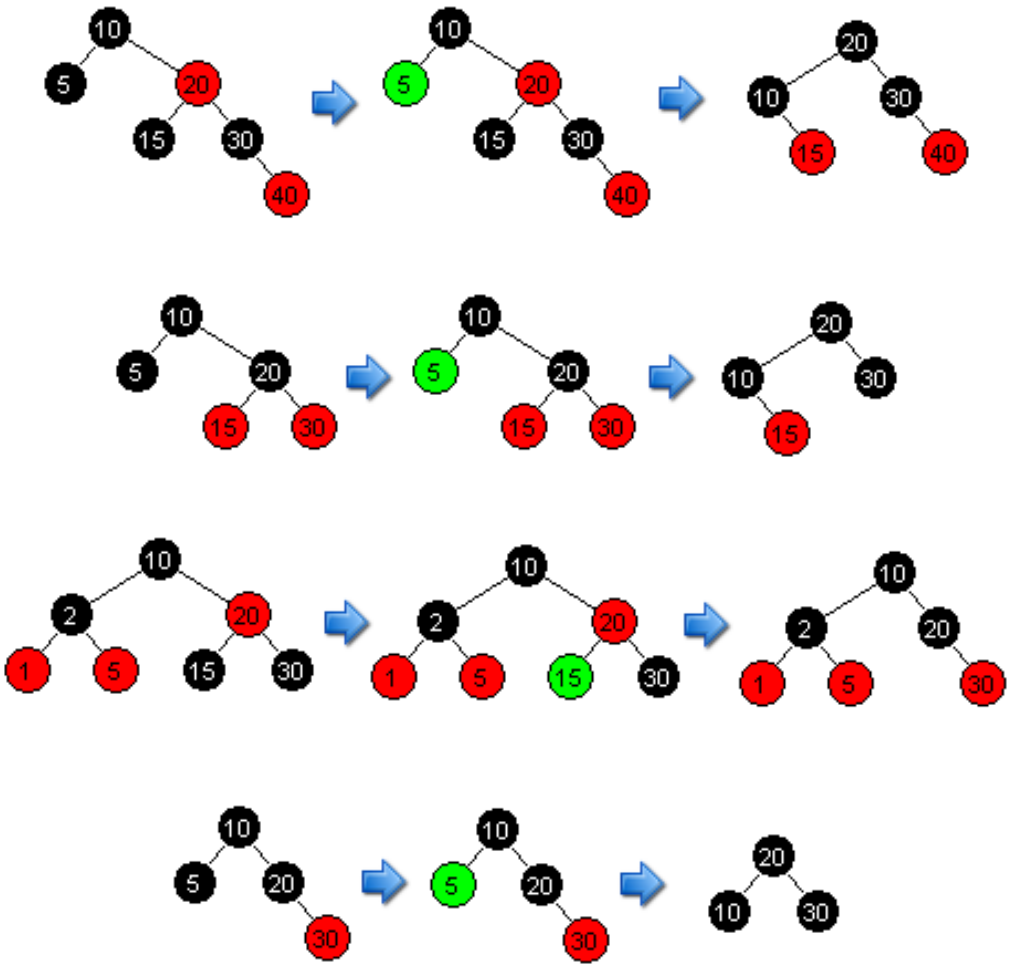
Red-Black Tree - Node Removal



Red-Black Tree - Node Removal



Red-Black Tree - Node Removal



$\sim$ 300 line of Java code for node deletion...

KORAT: filter-based test-case generation

```
public class RedBlackTree {
  Node root;
  int size;
  class Node {
    int key, value;
    Node left, right, parent;
    int color;
  }
}
```

```
boolean repOK() { ... }
```

1. build a tuple of the variables
2. fix a finite domain for variables
3. enumerate concrete tuples
4. select valid structures using repOK
5. pruning irrelevant instances by monitoring fields accessed by repOK

Keys in {0,1,2,3}
Size = 3 $\langle root, size, \underbrace{key_1, value_1, left_1, right_1, parent_1, color_1}_{\text{Node 1}}, \underbrace{\ldots}_{\text{Node 2}}, \underbrace{\ldots}_{\text{Node 3}} \rangle$

domain	2147483648	(combinatorial)	$color_1, color_2, color_3 \in \{\texttt{RED},\texttt{BLACK}\}$
explored	248	(with pruning)	$left_1, left_2, left_3 \in \{\texttt{null},1,2,3\}$
found	12	(desired)	$key_1, key_2, key_3 \in \{0,1,2,3\}$ ...

KORAT on Red-Black Trees

size	*domain*	*explored*	*found*	time (s)
0	1	1	1	0.20
1	128	10	4	0.21
2	236196	52	6	0.22
3	2147483648	248	12	0.23
4	$6.10*10^{13}$	1521	40	0.30
5	$4.21*10^{18}$	6827	84	0.46
6	$6.01*10^{23}$	25727	140	0.58
7	$1.58*10^{29}$	109749	280	0.95
8	$7.12*10^{34}$	481398	576	1.91
9	$5.12*10^{40}$	2184788	1220	6.12
10	$5.61*10^{46}$	10693872	2860	30.01
11	$8.99*10^{52}$	54605852	7032	168.33
12	$2.04*10^{59}$	279610078	16848	878.13
13	$6.37*10^{65}$	1416658806	37912	5634.82

- Korat has an **efficient pruning** mechanism
- The problem is **inherently hard**
- How do we scale-up?

Constraint Logic Programming

Programs as sets of rules (clauses) of the form:

$\mathtt{H}$ `:-` $c \wedge \mathtt{B}$ (meaning, $\mathtt{H}$ holds **if** c is satisfiable in $\mathcal{T}$ and $\mathtt{B}$ holds)

Example:

```
ordered([]).
ordered([X]).
ordered([X1,X2|L]) :- X1 ≤ X2 ∧ ordered([X2|L]).
```

$X_1 \leq X_2$: solver for $\mathcal{T}$; `ordered([X2|L])`: resolution

Query evaluation:

$d \wedge \mathtt{G} \implies_{\rho}^{k} c_1 \wedge \ldots \wedge c_n \wedge \square$ (with $c_1 \wedge \ldots \wedge c_n$ $\mathcal{T}$-satisfiable) (and $\rho = \vartheta_1 \cdot \ldots \cdot \vartheta_k$)

$\implies_{\vartheta}^{1}$ =

1. $d \wedge \mathtt{G} = d \wedge \mathtt{A} \wedge \mathtt{R}$
2. find unifying head $(\mathtt{A} = \mathtt{H})\vartheta$
3. rewrite into $(d \wedge c \wedge \mathtt{B} \wedge \mathtt{R})\vartheta$

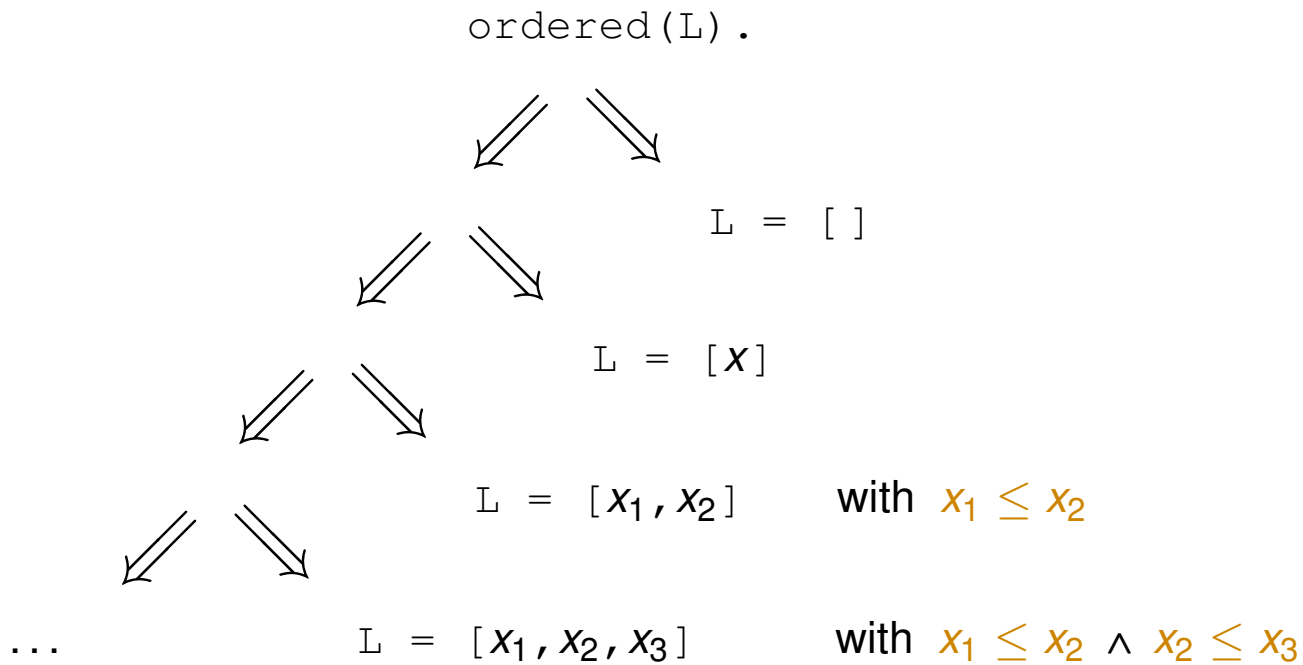
CLP Evaluation for Test Generation

```
ordered([]).
ordered([X]).
ordered([X1,X2|L]) :- X1 ≤ X2 ∧ ordered([X2|L]).
```

$X_1 \leq X_2$: solver for $\mathcal{T}$

`ordered([X2|L])`: resolution

As a generator:



A CLP-based Encoding of Red-Black Trees

```
rbtree(T,MinSize,MaxSize,NumKeys) :-
    % Preamble
      MinSize#=<S, S#=<MaxSize, fd_labeling([S]),
      length(Keys,S), length(Colors,S), Max#=NumKeys-1,
      fd_domain(Keys,0,Max), fd_domain(Colors,0,1),
    % Symbolic Definition
      lbt(T,S,Keys,[]),                  % data structure shape
      pi(T,D), ci(T,Colors,[]),          % filters
      ordered(T,0,NumKeys),              % filters
    % Instantiation
      fd_labeling(Keys), fd_labeling(Colors).
```

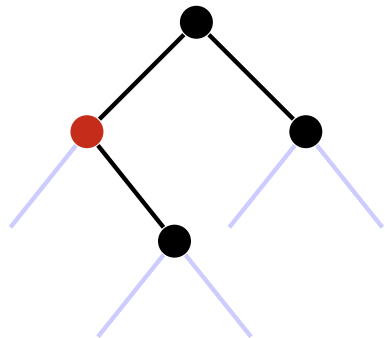
`lbt(T,S,Keys,[])`	**if**	`T` is labeled binary tree with `S` nodes
`pi(T,D)`	**if**	the tree `T` satisfies the **path invariant**
`ci(T,Colors)`	**if**	the tree `T` satisfies the **color invariant**
`ordered(T,0,NumKeys)`	**if**	the labels (keys) in `T` are **ordered** left-to-right
`fd_labeling(X)`	**if**	the variable `X` is instantiated to a feasible value

On Specifying Invariants

KORAT

```
private boolean path_invariant(){
   int numberOfBlack = -1;
   workList = new java.util.LinkedList();
   workList.add(new Pair(root, 0));

   while (!workList.isEmpty()) {
      Pair p = (Pair) workList.removeFirst();
      Node e = p.e;
      int n = p.n;
      if (e != null && e.color == BLACK)
         n++;
      if (e == null) {
         if (numberOfBlack == -1)
            numberOfBlack = n;
         else if (numberOfBlack != n)
            return debug("failure");
      } else {
         workList.add(new Pair(e.left, n));
         workList.add(new Pair(e.right, n));
      }
   }
   return true;
}
```

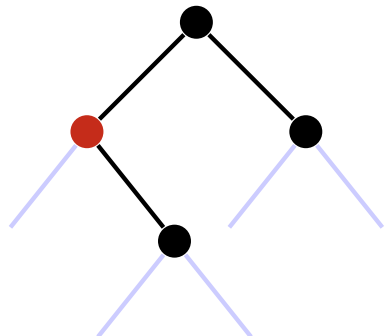


On Specifying Invariants

```
private boolean path_invariant(){
   int numberOfBlack = -1;
   workList = new java.util.LinkedList();
   workList.add(new Pair(root, 0));

   while (!workList.isEmpty()) {
      Pair p = (Pair) workList.removeFirst();
      Node e = p.e;
      int n = p.n;
      if (e != null && e.color == BLACK)
         n++;
      if (e == null) {
         if (numberOfBlack == -1)
            numberOfBlack = n;
         else if (numberOfBlack != n)
            return debug("failure");
      } else {
         workList.add(new Pair(e.left, n));
         workList.add(new Pair(e.right, n));
      }
   }
   return true;
}
```

KORAT



`numberOfBlack`, `workList`

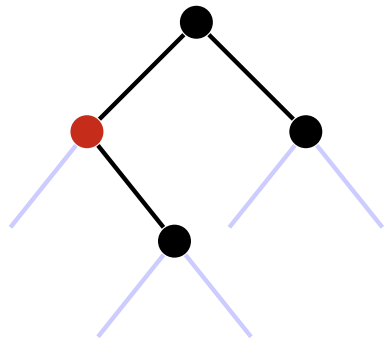
-1	(root,0)
-1	(l,1), (r,1)
-1	(**e**,**1**), (lr,1), (r,1)
1	(lr,1), (r,1)
1	(**e**,**2**), (e,2), (r,1)

On Specifying Invariants

```
private boolean path_invariant(){
   int numberOfBlack = -1;
   workList = new java.util.LinkedList();
   workList.add(new Pair(root, 0));

   while (!workList.isEmpty()) {
      Pair p = (Pair) workList.removeFirst();
      Node e = p.e;
      int n = p.n;
      if (e != null && e.color == BLACK)
         n++;
      if (e == null) {
         if (numberOfBlack == -1)
            numberOfBlack = n;
         else if (numberOfBlack != n)
            return debug("failure");
      } else {
         workList.add(new Pair(e.left, n));
         workList.add(new Pair(e.right, n));
      }
   }
   return true;
}
```

KORAT



`numberOfBlack`, `workList`

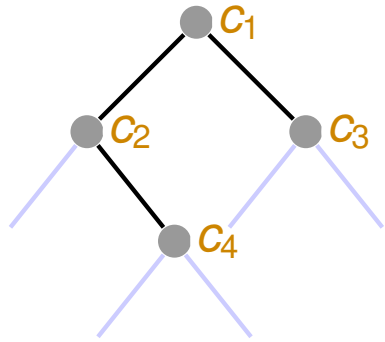
-1	(root,0)
-1	(l,1), (r,1)
-1	(**e**,**1**), (lr,1), (r,1)
1	(lr,1), (r,1)
1	(**e**,**2**), (e,2), (r,1)

$1 \neq 2$ **fail**

On Specifying Invariants

```
private boolean path_invariant(){
   int numberOfBlack = -1;
   workList = new java.util.LinkedList();
   workList.add(new Pair(root, 0));

   while (!workList.isEmpty()) {
      Pair p = (Pair) workList.removeFirst();
      Node e = p.e;
      int n = p.n;
      if (e != null && e.color == BLACK)
         n++;
      if (e == null) {
         if (numberOfBlack == -1)
            numberOfBlack = n;
         else if (numberOfBlack != n)
            return debug("failure");
      } else {
         workList.add(new Pair(e.left, n));
         workList.add(new Pair(e.right, n));
      }
   }
   return true;
}
```



$c_1, \ldots, c_4$ in {**0**,**1**}

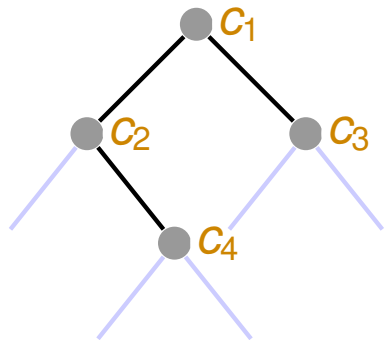
CLP

```
1.  pi(          e,D) :- D#=0.
2.  pi(t(C,K,L,R),D) :- ND#>=0, D#=ND+C, pi(L,ND), pi(R,ND).
```

On Specifying Invariants

```
private boolean path_invariant(){
   int numberOfBlack = -1;
   workList = new java.util.LinkedList();
   workList.add(new Pair(root, 0));

   while (!workList.isEmpty()) {
      Pair p = (Pair) workList.removeFirst();
      Node e = p.e;
      int n = p.n;
      if (e != null && e.color == BLACK)
         n++;
      if (e == null) {
         if (numberOfBlack == -1)
            numberOfBlack = n;
         else if (numberOfBlack != n)
            return debug("failure");
      } else {
         workList.add(new Pair(e.left, n));
         workList.add(new Pair(e.right, n));
      }
   }
   return true;
}
```



$c_1, \ldots, c_4$ in $\{\mathbf{0}, \mathbf{1}\}$

$$c_1 + c_2 = c_1 + c_2 + c_4 = c_1 + c_3$$

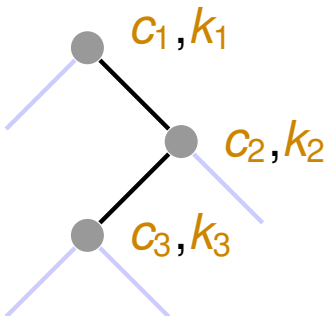
CLP

```
1.  pi(          e,D) :- D#=0.
2.  pi(t(C,K,L,R),D) :- ND#>=0, D#=ND+C, pi(L,ND), pi(R,ND).
```

Shape Rejection

`Tree ::= e | Color × Key × Tree × Tree` with **`Color`** in $\{0, 1\}$ (red, black) and **`Key`** in $\{0, \ldots, \text{MaxKey}\}$

Size 3 (a possible solution):



$\wedge$ structure shape

$$\begin{array}{c} c_1 = c_1 + c_2 \\ \wedge \\ c_1 + c_2 = c_1 + c_2 + c_3 \end{array}$$

path invariant

$\wedge$

$$\begin{array}{c} c_1 + c_2 > 0 \\ \wedge \\ c_2 + c_3 > 0 \end{array}$$

color invariant

$\wedge$

$$\begin{array}{c} k_1 < k_2 \\ \wedge \\ k_1 < k_3 \\ \wedge \\ k_3 < k_2 \end{array}$$

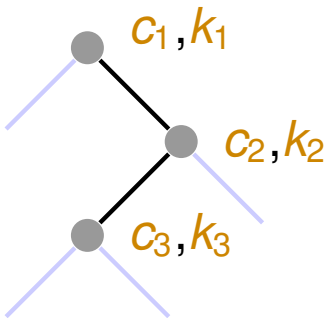
ordering

Shape Rejection

`Tree ::= e | Color × Key × Tree × Tree`

with `Color` in $\{0, 1\}$ (red, black)
and `Key` in $\{0, \ldots, \text{MaxKey}\}$

Size 3 (a possible solution):



$$\wedge \quad \begin{array}{c} c_1 = c_1 + c_2 \\ \wedge \\ c_1 + c_2 = c_1 + c_2 + c_3 \end{array} \quad \wedge \quad \begin{array}{c} c_1 + c_2 > 0 \\ \wedge \\ c_2 + c_3 > 0 \end{array} \quad \wedge \quad \begin{array}{c} k_1 < k_2 \\ \wedge \\ k_1 < k_3 \\ \wedge \\ k_3 < k_2 \end{array}$$

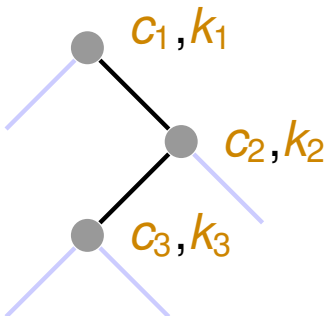
structure shape | path invariant | color invariant | ordering

lbt(T,S,Keys,[ ]) ∧ pi(T,D) ∧ ci(T,Colors) ∧ ordered(T,. . .)

Shape Rejection

`Tree ::= e | Color × Key × Tree × Tree` with **`Color`** in $\{0, 1\}$ (red, black) and **`Key`** in $\{0, \ldots, \text{MaxKey}\}$

Size 3 (a possible solution):



$\wedge$ $\begin{array}{c} c_1 = c_1 + c_2 \\ \wedge \\ c_1 + c_2 = c_1 + c_2 + c_3 \end{array}$ $\wedge$ $\begin{array}{c} c_1 + c_2 > 0 \\ \wedge \\ c_2 + c_3 > 0 \end{array}$ $\wedge$ $\begin{array}{c} k_1 < k_2 \\ \wedge \\ k_1 < k_3 \\ \wedge \\ k_3 < k_2 \end{array}$ UNFEASIBLE

structure shape | path invariant | color invariant | ordering

lbt(T,S,Keys,[ ]) $\wedge$ pi(T,D) $\wedge$ ci(T,Colors) $\wedge$ ordered(T,. . .)

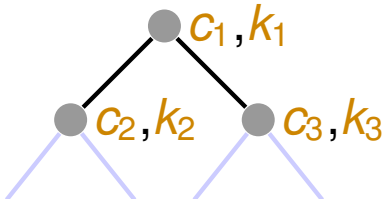
No instantiation possible $\Rightarrow$ $c_2 = 0 \wedge c_3 = 0$

Shape Concretization

`Tree ::= e |` **`Color`** $\times$ **`Key`** $\times$ `Tree` $\times$ `Tree`

with **`Color`** in $\{0, 1\}$ (red, black)
and **`Key`** in $\{0, \ldots, \text{MaxKey}\}$

Size 3 (another solution) :



$$\bigwedge \quad c_1 + c_2 = c_1 + c_3 \quad \bigwedge \quad \begin{array}{c} c_1 + c_2 > 0 \\ \wedge \\ c_1 + c_3 > 0 \end{array} \quad \bigwedge \quad \begin{array}{c} k_2 < k_1 \\ \wedge \\ k_1 < k_3 \end{array}$$

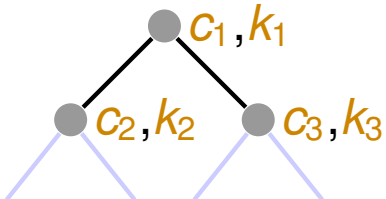
FEASIBLE

Instantiations:

Shape Concretization

`Tree ::= e | Color × Key × Tree × Tree` with **Color** in $\{0, 1\}$ (red, black) and **Key** in $\{0, \ldots, \text{MaxKey}\}$

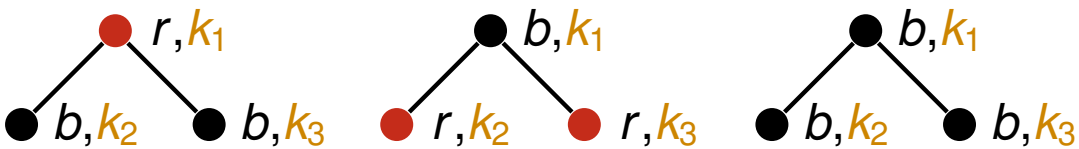
Size 3 (another solution) :



$$\wedge \quad c_1 + c_2 = c_1 + c_3 \quad \wedge \quad \begin{array}{c} c_1 + c_2 > 0 \\ \wedge \\ c_1 + c_3 > 0 \end{array} \quad \wedge \quad \begin{array}{c} k_2 < k_1 \\ \wedge \\ k_1 < k_3 \end{array}$$

FEASIBLE

Instantiations:

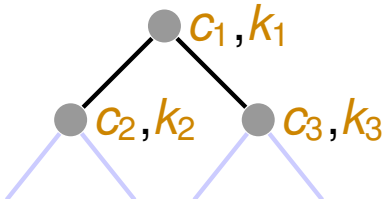


Shape Concretization

`Tree ::= e | Color × Key × Tree × Tree`

with `Color` in $\{0, 1\}$ (red, black)
and `Key` in $\{0, \ldots, \text{MaxKey}\}$

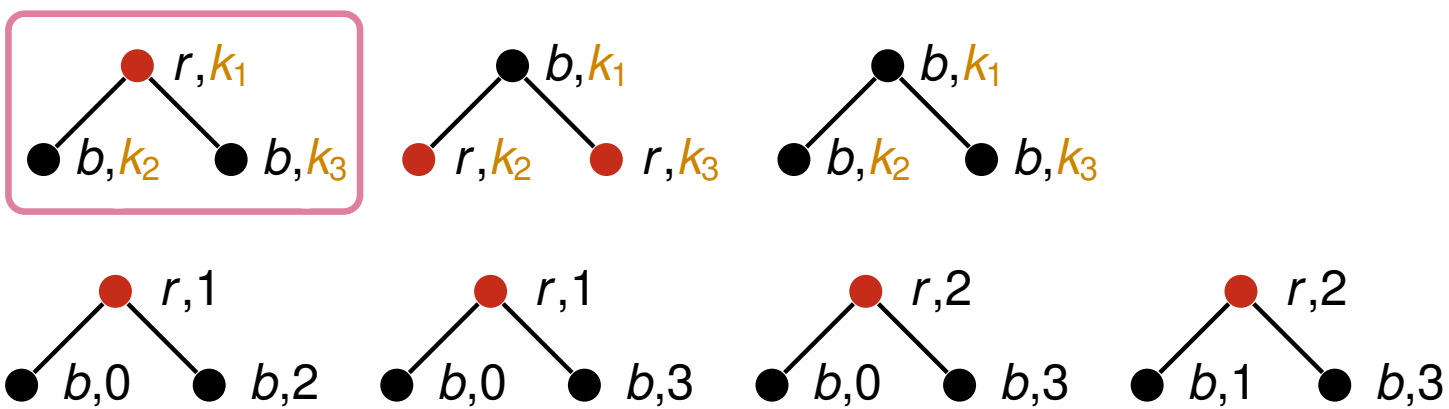
Size 3 (another solution) :



$$\wedge \quad c_1 + c_2 = c_1 + c_3 \quad \wedge \quad \begin{array}{c} c_1 + c_2 > 0 \\ \wedge \\ c_1 + c_3 > 0 \end{array} \quad \wedge \quad \begin{array}{c} k_2 < k_1 \\ \wedge \\ k_1 < k_3 \end{array}$$

FEASIBLE

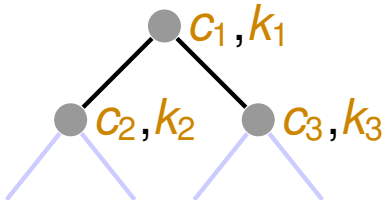
Instantiations:



Shape Concretization

`Tree ::= e | Color × Key × Tree × Tree` with **Color** in $\{0, 1\}$ (red, black) and **Key** in $\{0, \ldots, \text{MaxKey}\}$

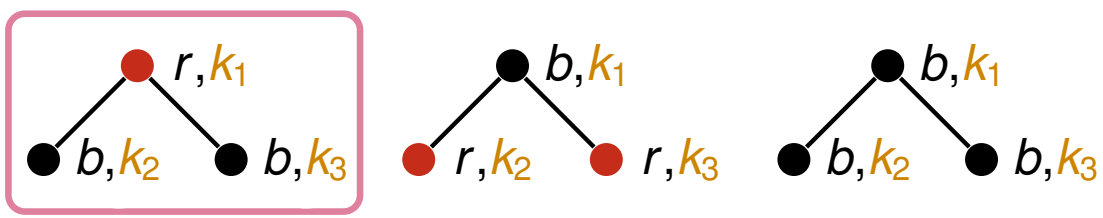
Size 3 (another solution) :



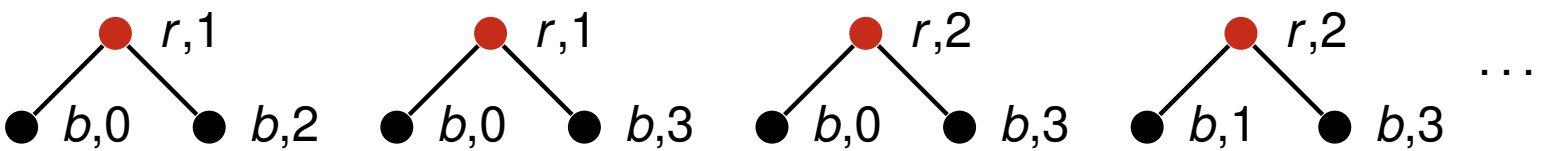
$$\wedge \quad c_1 + c_2 = c_1 + c_3 \quad \wedge \quad \begin{array}{c} c_1 + c_2 > 0 \\ \wedge \\ c_1 + c_3 > 0 \end{array} \quad \wedge \quad \begin{array}{c} k_2 < k_1 \\ \wedge \\ k_1 < k_3 \end{array}$$

FEASIBLE

Instantiations:



5 shapes
1 feasible
12 instantiations



. . .

we let the solver choose an instantiation order to minimize backtracking

KORAT

size	*domain*	*explored*	*found*	time (s)
0	1	1	1	0.20
1	128	10	4	0.21
2	236196	52	6	0.22
3	2147483648	248	12	0.23
4	$6.10*10^{13}$	1521	40	0.30
5	$4.21*10^{18}$	6827	84	0.46
6	$6.01*10^{23}$	25727	140	0.58
7	$1.58*10^{29}$	109749	280	0.95
8	$7.12*10^{34}$	481398	576	1.91
9	$5.12*10^{40}$	2184788	1220	6.12
10	$5.61*10^{46}$	10693872	2860	30.01
11	$8.99*10^{52}$	54605852	7032	168.33
12	$2.04*10^{59}$	279610078	16848	878.13
13	$6.37*10^{65}$	1416658806	37912	5634.82

Constrain-and-Generate

size	*domain*	symb.expl.	*feasible*	*found*	time (s)
0	1	1	1	1	0.01
1	4	1	1	4	0.01
2	72	2	2	6	0.01
3	2560	5	1	12	0.01
4	140000	14	4	40	0.01
5	10450944	42	6	84	0.01
6	993898752	132	12	140	0.01
7	$1.15*10^{11}$	429	29	280	0.01
8	$1.57*10^{14}$	1430	40	576	0.03
9	$2.49*10^{15}$	4862	46	1220	0.11
10	$4.46*10^{17}$	16796	76	2860	0.40
11	$8.94*10^{19}$	58786	190	7032	1.63
12	$1.98*10^{22}$	208012	456	16848	6.20
13	$4.83*10^{24}$	742900	904	37912	23.31

KORAT

Graphs

size	*domain*	*explored*	*found*	time (s)
0	1	1	1	0.20
1	128	10	4	0.21
2	236196	52	6	0.22
3	2147483648	248	12	0.23
4	$6.10*10^{13}$	1521	40	0.30
5	$4.21*10^{18}$	6827	84	0.46
6	$6.01*10^{23}$	25727	140	0.58
7	$1.58*10^{29}$	109749	280	0.95
8	$7.12*10^{34}$	481398	576	1.91
9	$5.12*10^{40}$	2184788	1220	6.12
10	$5.61*10^{46}$	10693872	2860	30.01
11	$8.99*10^{52}$	54605852	7032	168.33
12	$2.04*10^{59}$	279610078	16848	878.13
13	$6.37*10^{65}$	1416658806	37912	5634.82

Constrain-and-Generate

Trees

size	*domain*	symb.expl.	*feasible*	*found*	time (s)
0	1	1	1	1	0.01
1	4	1	1	4	0.01
2	72	2	2	6	0.01
3	2560	5	1	12	0.01
4	140000	14	4	40	0.01
5	10450944	42	6	84	0.01
6	993898752	132	12	140	0.01
7	$1.15*10^{11}$	429	29	280	0.01
8	$1.57*10^{14}$	1430	40	576	0.03
9	$2.49*10^{15}$	4862	46	1220	0.11
10	$4.46*10^{17}$	16796	76	2860	0.40
11	$8.94*10^{19}$	58786	190	7032	1.63
12	$1.98*10^{22}$	208012	456	16848	6.20
13	$4.83*10^{24}$	742900	904	37912	23.31

KORAT

size	*domain*	*explored*	*found*	time (s)
0	1	1	1	0.20
1	128	10	4	0.21
2	236196	52	6	0.22
3	2147483648	248	12	0.23
4	$6.10*10^{13}$	1521	40	0.30
5	$4.21*10^{18}$	6827	84	0.46
6	$6.01*10^{23}$	25727	140	0.58
7	$1.58*10^{29}$	109749	280	0.95
8	$7.12*10^{34}$	481398	576	1.91
9	$5.12*10^{40}$	2184788	1220	6.12
10	$5.61*10^{46}$	10693872	2860	30.01
11	$8.99*10^{52}$	54605852	7032	168.33
12	$2.04*10^{59}$	279610078	16848	878.13
13	$6.37*10^{65}$	1416658806	37912	5634.82

Graphs

Constrain-and-Generate

size	*domain*	symb.expl.	*feasible*	*found*	time (s)
0	1	1	1	1	0.01
1	4	1	1	4	0.01
2	72	2	2	6	0.01
3	2560	5	1	12	0.01
4	140000	14	4	40	0.01
5	10450944	42	6	84	0.01
6	993898752	132	12	140	0.01
7	$1.15*10^{11}$	429	29	280	0.01
8	$1.57*10^{14}$	1430	40	576	0.03
9	$2.49*10^{15}$	4862	46	1220	0.11
10	$4.46*10^{17}$	16796	76	2860	0.40
11	$8.94*10^{19}$	58786	190	7032	1.63
12	$1.98*10^{22}$	208012	456	16848	6.20
13	$4.83*10^{24}$	742900	904	37912	23.31

Trees

Symbolic + instantiation

size	*domain*	*explored*	*found*	time (s)
0	1	1	1	0.20
1	128	10	4	0.21
2	236196	52	6	0.22
3	2147483648	248	12	0.23
4	$6.10*10^{13}$	1521	40	0.30
5	$4.21*10^{18}$	6827	84	0.46
6	$6.01*10^{23}$	25727	140	0.58
7	$1.58*10^{29}$	109749	280	0.95
8	$7.12*10^{34}$	481398	576	1.91
9	$5.12*10^{40}$	2184788	1220	6.12
10	$5.61*10^{46}$	10693872	2860	30.01
11	$8.99*10^{52}$	54605852	7032	168.33
12	$2.04*10^{59}$	279610078	16848	878.13
13	$6.37*10^{65}$	1416658806	37912	5634.82

KORAT

Graphs

Pruning
(ratio ∼**6 oom**)

size	*domain*	symb.expl.	*feasible*	*found*	time (s)
0	1	1	1	1	0.01
1	4	1	1	4	0.01
2	72	2	2	6	0.01
3	2560	5	1	12	0.01
4	140000	14	4	40	0.01
5	10450944	42	6	84	0.01
6	993898752	132	12	140	0.01
7	$1.15*10^{11}$	429	29	280	0.01
8	$1.57*10^{14}$	1430	40	576	0.03
9	$2.49*10^{15}$	4862	46	1220	0.11
10	$4.46*10^{17}$	16796	76	2860	0.40
11	$8.94*10^{19}$	58786	190	7032	1.63
12	$1.98*10^{22}$	208012	456	16848	6.20
13	$4.83*10^{24}$	742900	904	37912	23.31

Constrain-and-Generate

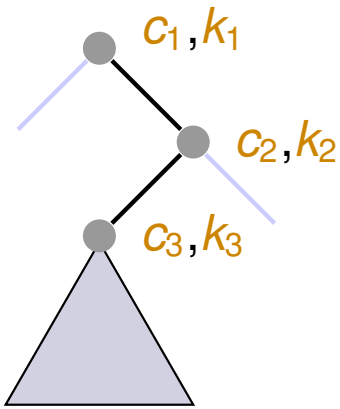
Trees

Symbolic + instantiation

Pruning symbolic
(ratio ∼**6.5 oom**)

Optimizing Generators by Transformation

Size 20



No way of placing the remaining
17 nodes to build a feasible tree

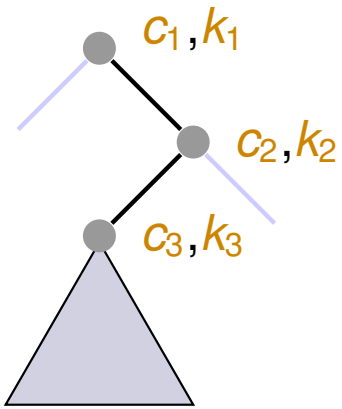
$c_1 = c_1 + c_2 \wedge c_1 = c_1 + c_2 + c_3 + X \wedge$
$X \geq 0 \wedge c_1 + c_2 > 0 \wedge c_2 + c_3 > 0$ UNFEASIBLE

Yet, there would be 35357670 **shapes**
(and corresponding feasiblity tests)

Idea: apply filter earlier

Optimizing Generators by Transformation

Size 20



No way of placing the remaining
17 nodes to build a feasible tree

$c_1 = c_1 + c_2 \wedge c_1 = c_1 + c_2 + c_3 + X \wedge$
$X \geq 0 \wedge c_1 + c_2 > 0 \wedge c_2 + c_3 > 0$ UNFEASIBLE

Yet, there would be 35357670 **shapes**
(and corresponding feasiblity tests)

Idea: apply filter earlier

Program Transformation = **rewrite rules + strategies**

easy to
prove correct

$$\begin{array}{ccccccc} P_0 & \longmapsto & \ldots & \longmapsto & P_n \\ M(P_0) & = & \ldots & = & M(P_n) \end{array}$$

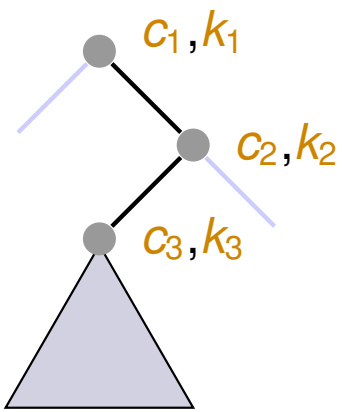
efficient

rule-based
model-preserving
(local) rewriting

A flexible deductive framework: applied to **Verification** & **SW Model Checking**

Optimizing Generators by Transformation

Size 20



No way of placing the remaining
17 nodes to build a feasible tree

$c_1 = c_1 + c_2 \wedge c_1 = c_1 + c_2 + c_3 + X \wedge$
$X \geq 0 \wedge c_1 + c_2 > 0 \wedge c_2 + c_3 > 0$ UNFEASIBLE

Yet, there would be 35357670 **shapes**
(and corresponding feasiblity tests)

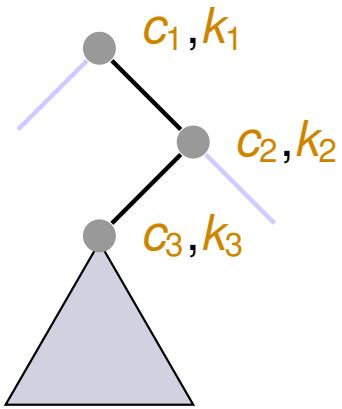
Idea: apply filter earlier

The Synchronization transformation strategy

- **filter promotion + tupling + folding** (local optimization inductively propagated)
- force the generator to have the desired behavior
- reduces don't care nondeterminism (less **failures**)

Optimizing Generators by Transformation

Size 20



No way of placing the remaining
17 nodes to build a feasible tree

$c_1 = c_1 + c_2 \wedge c_1 = c_1 + c_2 + c_3 + X \wedge$
$X \geq 0 \wedge c_1 + c_2 > 0 \wedge c_2 + c_3 > 0$ UNFEASIBLE

Yet, there would be 35357670 **shapes**
(and corresponding feasiblity tests)

Idea: apply filter earlier

The Synchronization transformation strategy

- **filter promotion + tupling + folding** (local optimization inductively propagated)
- force the generator to have the desired behavior
- reduces don't care nondeterminism (less **failures**)

Related techniques : compiling control, co-routining

The synch program

```
1. sync(                              e ,A, B,B, C,C,_,_,0) :- A#=0.
2. sync( t(A,B,          e,           e),C,[B|D],D,[A|E],E,F,G,A) :-
     C#=1, F#=<B, B#<G.
3. sync(t(A,B,           e,t(C,D,E,F)),G,[B|H],I,[A|J],K,L,M,A) :-
     G#>=2, O#=G-1, A+C#>0, L#=<B, B#<M, P#=B+1,
     sync(t(C,D,E,F),O,H,I,J,K,P,M,0).
4. sync(t(A,F,t(B,C,D,E),             e),G,[F|H],I,[A|J],K,L,M,A) :-
     G#>=2, O#=G-1, A+B#>0, L#=<F, F#<M,
     sync(t(B,C,D,E),O,H,I,J,K,L,F,0).
5. sync(t(A,F,t(B,C,D,E),t(G,H,I,J)),K,[F|L],M,[A|N],O,P,Q,R) :-
     K#>=3, S#=K-1, T#>0, U#>0, A+B#>0, A+G#>0, V#>=0,
     R#=V+A, P#=<F, F#<Q, W#=F+1, fdsum(S,T,U),
     sync(t(B,C,D,E),T,L,X,N,Y,P,F,V),
     sync(t(G,H,I,J),U,X,M,Y,O,W,Q,V).
```

The new generator intertwines **tree construction** and **invariant construction**

Experiments

Size	***RB Trees***	Time		
		Original	Synchronized	Korat
6	20	0	0	0.47
7	35	0.01	0	0.63
8	64	0.02	0	1.49
9	122	0.08	0	4.51
10	260	0.29	0.01	21.14
11	586	1.07	0.03	116.17
12	1296	3.98	0.06	-
13	2708	14.85	0.12	-
14	5400	55.77	0.26	-
15	10468	-	0.55	-
...	...	...	...	...
20	279264	-	25.90	-

Size = number of nodes
RB Trees = number of structures found
Time = (in seconds) for generating all the structures

Zero means less than 10 ms and (-) means more than 200 seconds.

Experiments (cont.)

Size	**Sorted Lists**	Time	
		C&G	Korat
8	6435	0.00	0.61
9	24310	0.00	1.08
10	92378	0.02	1.83
11	352716	0.09	6.37
12	1352078	0.36	24.95
13	5200300	1.40	125.73
14	20058300	5.40	-
15	77558760	21.16	-
16	300540195	82.22	-

Size	**Heaparrays**	Time	
		C&G	Korat
6	13139	0.00	0.30
7	117562	0.03	0.86
8	1005075	0.17	3.41
9	10391382	1.66	34.103
10	111511015	17.57	-

Size	**Search Trees**	Time	
		C&G	Korat
7	429	0.01	0.87
8	1430	0.02	4.43
9	4862	0.08	33.99
10	16796	0.28	-
11	58786	1.11	-
12	208012	4.43	-
13	742900	17.68	-
14	2674440	70.75	-

Size	**Disjoint Sets**	Time	
		C&G	Korat
6	203	0.00	1.60
7	877	0.00	37.10
8	4140	0.01	-
9	21147	0.10	-
10	115975	0.61	-
11	678570	3.90	-
12	4213597	26.63	-
13	27644437	189.42	-

No optimizations

From CLP Terms to Java Objects

A solution using Java (de)serialization:

1. generate XML encodings of (found) objects
2. XML description $\longrightarrow$ Java object

   by using libraries such as XStream/Simple

XML $\longleftrightarrow$ Java-objects given at specification-time, inductively

Triggered only on structures of actual interest for testing,
with negligible overhead

Work in Progress

- Experiments with graph-like structures
- Experiments with ASP solvers
- Full automation of the optimization algorithm
- Automatic extraction of CLP generators from contracts
- Testing web-based applications