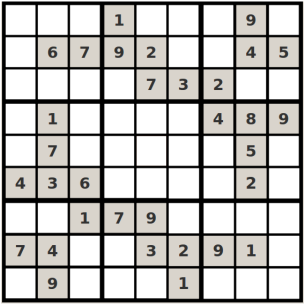
Unifying Execution of Imperative and Declarative Code

Aleksandar Milicevic

Massachusetts Institute of Technology
Cambridge, MA

RQE Defense, MIT
May 03, 2011

Solving Sudoku

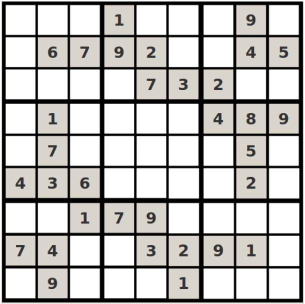


			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

Sudoku puzzle: fill in the empty cells s.t.:

1. cell values are in $\{1, 2, \cdots, 9\}$
2. all rows have distinct values
3. all columns have distinct values
4. all sub-grids have distinct values

Solving Sudoku



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

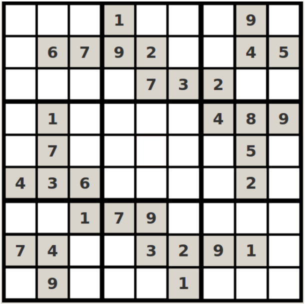
Sudoku puzzle: fill in the empty cells s.t.:

1. cell values are in $\{1,2,\cdots,9\}$
2. all rows have distinct values
3. all columns have distinct values
4. all sub-grids have distinct values

Approaches:

- write a custom (heuristic-based) algorithm [imperative]
- write a set of constraints and use a constraint solver [declarative]

Solving Sudoku



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

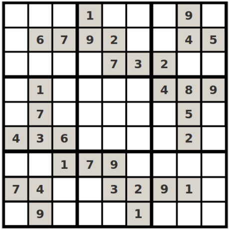
Sudoku puzzle: fill in the empty cells s.t.:

1. cell values are in $\{1, 2, \cdots, 9\}$
2. all rows have distinct values
3. all columns have distinct values
4. all sub-grids have distinct values

Approaches:

- write a custom (heuristic-based) algorithm [imperative]
- write a set of constraints and use a constraint solver [declarative]

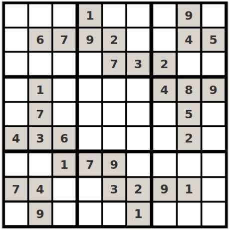
Solving Sudoku with **Squander**



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

Solving Sudoku with **Squander**

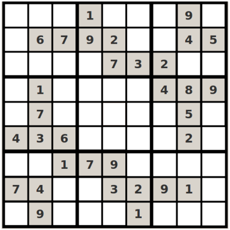
```
public class Sudoku {
  private final int n = 9;
  private final int[][] regions = new int[] {
    new int[] {0, 1, 2},
    new int[] {3, 4, 5},
    new int[] {6, 7, 8}
  };

  private int[][] data = new int[n][n];

  public Sudoku() {}
```



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

Solving Sudoku with **Squander**

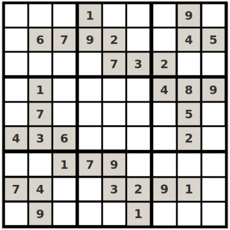
```
public class Sudoku {
  private final int n = 9;
  private final int[][] regions = new int[] {
    new int[] {0, 1, 2},
    new int[] {3, 4, 5},
    new int[] {6, 7, 8}
  };

  private int[][] data = new int[n][n];

  public Sudoku() {}




  public void solve() { Squander.exe(this); }
```



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

Solving Sudoku with **Squander**

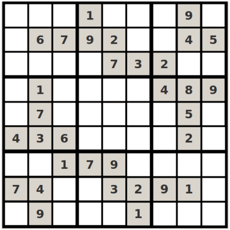
```
public class Sudoku {
  private final int n = 9;
  private final int[][] regions = new int[] {
    new int[] {0, 1, 2},
    new int[] {3, 4, 5},
    new int[] {6, 7, 8}
  };

  private int[][] data = new int[n][n];

  public Sudoku() {}

  @Ensures({
    "all row in {0 ... (this.n - 1)} | this.data[row][int] = {1 ... this.n}",
    "all col in {0 ... (this.n - 1)} | this.data[int][col] = {1 ... this.n}",


  public void solve() { Squander.exe(this); }
```



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

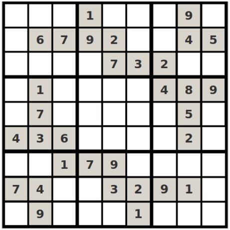
Solving Sudoku with **Squander**

```
public class Sudoku {
  private final int n = 9;
  private final int[][] regions = new int[] {
    new int[] {0, 1, 2},
    new int[] {3, 4, 5},
    new int[] {6, 7, 8}
  };

  private int[][] data = new int[n][n];

  public Sudoku() {}

  @Ensures({
    "all row in {0 ... (this.n - 1)} | this.data[row][int] = {1 ... this.n}",
    "all col in {0 ... (this.n - 1)} | this.data[int][col] = {1 ... this.n}",
    "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})

  public void solve() { Squander.exe(this); }
```



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

Solving Sudoku with **Squander**

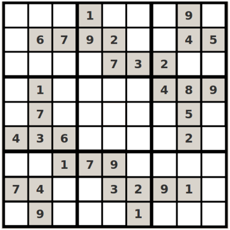


			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

```
public class Sudoku {
  private final int n = 9;
  private final int[][] regions = new int[] {
    new int[] {0, 1, 2},
    new int[] {3, 4, 5},
    new int[] {6, 7, 8}
  };

  private int[][] data = new int[n][n];

  public Sudoku() {}

  @Ensures({
    "all row in {0 ... (this.n - 1)} | this.data[row][int] = {1 ... this.n}",
    "all col in {0 ... (this.n - 1)} | this.data[int][col] = {1 ... this.n}",
    "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
  @Modifies("this.data[int].elems | _<2> = 0")
  public void solve() { Squander.exe(this); }
```

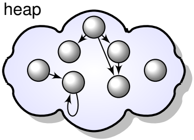
Solving Sudoku with **Squander**

```
public class Sudoku {
  private final int n = 9;
  private final int[][] regions = new int[] {
    new int[] {0, 1, 2},
    new int[] {3, 4, 5},
    new int[] {6, 7, 8}
  };

  private int[][] data = new int[n][n];

  public Sudoku() {}

  @Ensures({
    "all row in {0 ... (this.n - 1)} | this.data[row][int] = {1 ... this.n}",
    "all col in {0 ... (this.n - 1)} | this.data[int][col] = {1 ... this.n}",
    "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
  @Modifies("this.data[int].elems | _<2> = 0")
  public void solve() { Squander.exe(this); }

  public static void main(String[] args) {
    Sudoku s = new Sudoku();
    s.data[0][3] = 1; s.data[0][7] = 9;
    ...
    s.data[8][1] = 9; s.data[8][5] = 1;
    s.solve();
    System.out.println(s);
  }
}
```



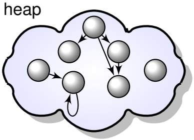
			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

Solving Sudoku with **Squander**



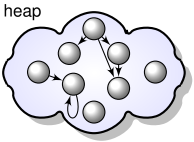
```
@Ensures({
  "all row in {0 ... (this.n − 1)} | this.data[row][int] = {1 ... this.n}",
  "all col in {0 ... (this.n − 1)} | this.data[int][col] = {1 ... this.n}",
  "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
@Modifies("this.data[int].elems | _<2> = 0")
public void solve() { Squander.exe(this); }

public static void main(String[] args) {
  Sudoku s = new Sudoku();
  s.data[0][3] = 1; s.data[0][7] = 9;
  ...
  s.data[8][1] = 9; s.data[8][5] = 1;
  s.solve();
  System.out.println(s);
}
```

Solving Sudoku with **Squander**



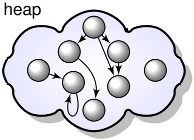
```
@Ensures({
  "all row in {0 ... (this.n − 1)} | this.data[row][int] = {1 ... this.n}",
  "all col in {0 ... (this.n − 1)} | this.data[int][col] = {1 ... this.n}",
  "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
@Modifies("this.data[int].elems | _<2> = 0")
public void solve() { Squander.exe(this); }

public static void main(String[] args) {
  Sudoku s = new Sudoku();
  s.data[0][3] = 1; s.data[0][7] = 9;
  ...
  s.data[8][1] = 9; s.data[8][5] = 1;
  s.solve();
  System.out.println(s);
}
```

Solving Sudoku with **Squander**



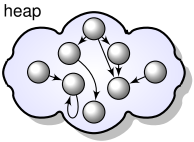
```
@Ensures({
  "all row in {0 ... (this.n − 1)} | this.data[row][int] = {1 ... this.n}",
  "all col in {0 ... (this.n − 1)} | this.data[int][col] = {1 ... this.n}",
  "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
@Modifies("this.data[int].elems | _<2> = 0")
public void solve() { Squander.exe(this); }

public static void main(String[] args) {
  Sudoku s = new Sudoku();
  s.data[0][3] = 1; s.data[0][7] = 9;
  ...
  s.data[8][1] = 9; s.data[8][5] = 1;
  s.solve();
  System.out.println(s);
}
```

Solving Sudoku with **Squander**



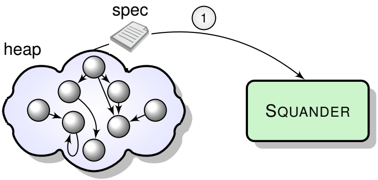
```
@Ensures({
  "all row in {0 ... (this.n − 1)} | this.data[row][int] = {1 ... this.n}",
  "all col in {0 ... (this.n − 1)} | this.data[int][col] = {1 ... this.n}",
  "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
@Modifies("this.data[int].elems | _<2> = 0")
public void solve() { Squander.exe(this); }

public static void main(String[] args) {
  Sudoku s = new Sudoku();
  s.data[0][3] = 1; s.data[0][7] = 9;
  ...
  s.data[8][1] = 9; s.data[8][5] = 1;
  s.solve();
  System.out.println(s);
}
```

Solving Sudoku with **Squander**



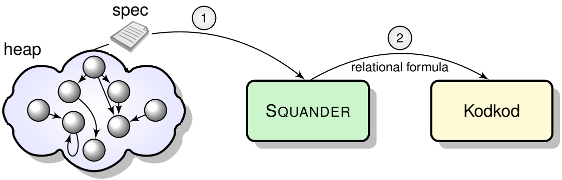
```
@Ensures({
  "all row in {0 ... (this.n − 1)} | this.data[row][int] = {1 ... this.n}",
  "all col in {0 ... (this.n − 1)} | this.data[int][col] = {1 ... this.n}",
  "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
@Modifies("this.data[int].elems | _<2> = 0")
public void solve() { Squander.exe(this); }

public static void main(String[] args) {
  Sudoku s = new Sudoku();
  s.data[0][3] = 1; s.data[0][7] = 9;
  ...
  s.data[8][1] = 9; s.data[8][5] = 1;
  s.solve();
  System.out.println(s);
}
```

Solving Sudoku with **Squander**



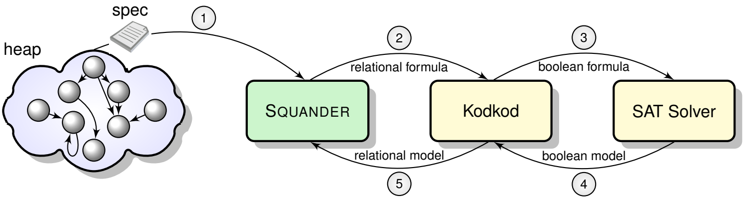
```
@Ensures({
  "all row in {0 ... (this.n - 1)} | this.data[row][int] = {1 ... this.n}",
  "all col in {0 ... (this.n - 1)} | this.data[int][col] = {1 ... this.n}",
  "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
@Modifies("this.data[int].elems | _<2> = 0")
public void solve() { Squander.exe(this); }

public static void main(String[] args) {
  Sudoku s = new Sudoku();
  s.data[0][3] = 1; s.data[0][7] = 9;
  ...
  s.data[8][1] = 9; s.data[8][5] = 1;
  s.solve();
  System.out.println(s);
}
```

Solving Sudoku with **Squander**



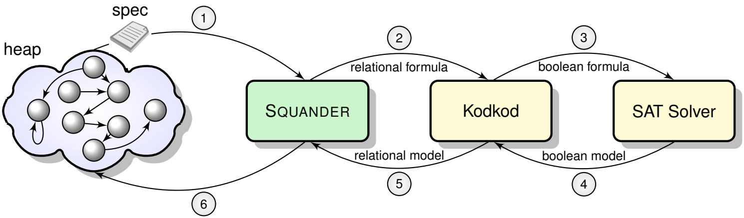
```
@Ensures({
  "all row in {0 ... (this.n − 1)} | this.data[row][int] = {1 ... this.n}",
  "all col in {0 ... (this.n − 1)} | this.data[int][col] = {1 ... this.n}",
  "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
@Modifies("this.data[int].elems | _<2> = 0")
public void solve() { Squander.exe(this); }

public static void main(String[] args) {
  Sudoku s = new Sudoku();
  s.data[0][3] = 1; s.data[0][7] = 9;
  ...
  s.data[8][1] = 9; s.data[8][5] = 1;
  s.solve();
  System.out.println(s);
}
```

Solving Sudoku with **Squander**



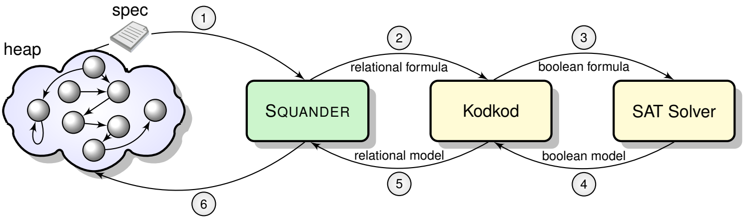
```
@Ensures({
  "all row in {0 ... (this.n - 1)} | this.data[row][int] = {1 ... this.n}",
  "all col in {0 ... (this.n - 1)} | this.data[int][col] = {1 ... this.n}",
  "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
@Modifies("this.data[int].elems | _<2> = 0")
public void solve() { Squander.exe(this); }

public static void main(String[] args) {
  Sudoku s = new Sudoku();
  s.data[0][3] = 1; s.data[0][7] = 9;
  ...
  s.data[8][1] = 9; s.data[8][5] = 1;
  s.solve();
  System.out.println(s);
}
```

Solving Sudoku with **Squander**



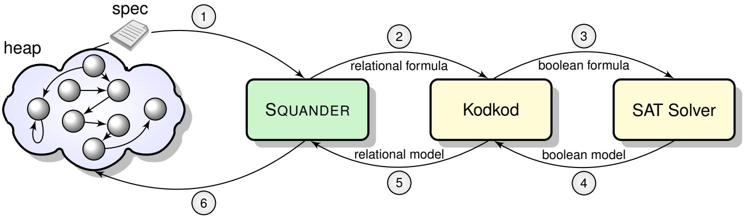
```
@Ensures({
  "all row in {0 ... (this.n - 1)} | this.data[row][int] = {1 ... this.n}",
  "all col in {0 ... (this.n - 1)} | this.data[int][col] = {1 ... this.n}",
  "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
@Modifies("this.data[int].elems | _<2> = 0")
public void solve() { Squander.exe(this); }

public static void main(String[] args) {
  Sudoku s = new Sudoku();
  s.data[0][3] = 1; s.data[0][7] = 9;
  ...
  s.data[8][1] = 9; s.data[8][5] = 1;
  s.solve();
  System.out.println(s);
}
```

Solving Sudoku with **Squander**



```
@Ensures({
  "all row in {0 ... (this.n − 1)} | this.data[row][int] = {1 ... this.n}",
  "all col in {0 ... (this.n − 1)} | this.data[int][col] = {1 ... this.n}",
  "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
@Modifies("this.data[int].elems | _<2> = 0")
public void solve() { Squander.exe(this); }

public static void main(String[] args) {
  Sudoku s = new Sudoku();
  s.data[0][3] = 1; s.data[0][7] = 9;
  ...
  s.data[8][1] = 9; s.data[8][5] = 1;
  s.solve();
  System.out.println(s);
}
```

Solving Sudoku with **Squander**


heap
spec
① SQUANDER
② relational formula
Kodkod
③ boolean formula
SAT Solver
④ boolean model
⑤ relational model
⑥

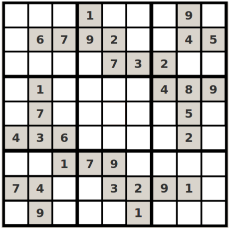
```
@Ensures({
  "all row in {0 ... (this.n - 1)} | this.data[row][int] = {1 ... this.n}",
  "all col in {0 ... (this.n - 1)} | this.data[int][col] = {1 ... this.n}",
  "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
@Modifies("this.data[int].elems | _<2> = 0")
public void solve() { Squander.exe(this); }

public static void main(String[] args) {
  Sudoku s = new Sudoku();
  s.data[0][3] = 1; s.data[0][7] = 9;
  ...
  s.data[8][1] = 9; s.data[8][5] = 1;
  s.solve();
  System.out.println(s);
}
```

Immediate Benefits

- executable Alloy-style specifications for Java
- specify and solve constraint problems “in place”
- no manual translation to/from an external solver

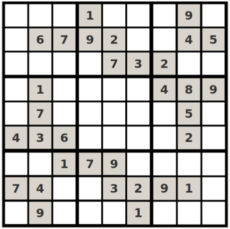
Solving Sudoku with **Alloy Analyzer**

```
abstract sig Number {}
one sig N1,N2,N3,N4,N5,N6,N7,N8,N9 extends Number {}

one sig Global {
    data: Number -> Number -> one Number
}

pred complete [rows:set Number, cols:set Number]{
   Number = Global.data[rows][cols]
}

pred rules {
   all row: Number { complete[row,Number] }
   all col: Number { complete[Number,col] }
   let r1=N1+N2+N3, r2=N4+N5+N6, r3=N7+N8+N9 |
     complete[r1,r1] and complete[r1,r2] and complete[r1,r3] and
     complete[r2,r1] and complete[r2,r2] and complete[r2,r3] and
     complete[r3,r1] and complete[r3,r2] and complete[r3,r3]
}

pred puzzle {
   N1->N4->N1 + N1->N8->N9 +
   ...
   N9->N2->N2 + N9->N6->N1 in Global.data
}

run { rules and puzzle }
```



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

Solving Sudoku with **Kodkod**



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

```
public class Sudoku {
  private Relation Number = Relation.unary("Number");
  private Relation data = Relation.ternary("data");
  private Relation[] regions = new Relation[] {
    Relation.unary("Region1"),
    Relation.unary("Region2"),
    Relation.unary("Region3") };

  public Formula complete(Expression rows, Expression cols) {
    // Number = data[rows][cols]
    return Number.eq(cols.join(rows.join(data))); }

  public Formula rules() {
    // all x,y: Number | lone data[x][y]
    Variable x = Variable.unary("x");
    Variable y = Variable.unary("y");
    Formula f1 = y.join(x.join(data)).lone().
      forAll(x.oneOf(Number).and(y.oneOf(Number)));
    // all row: Number | complete[row, Number]
    Variable row = Variable.unary("row");
    Formula f2 = complete(row, Number).
      forAll(row.oneOf(Number));
    // all col: Number | complete[Number, col]
    Variable col = Variable.unary("col");
    Formula f3 = complete(Number, col).
      forAll(col.oneOf(Number));
    // complete[r1,r1] and complete[r1,r2] and complete[r1,r3] and
    // complete[r2,r1] and complete[r2,r2] and complete[r2,r3] and
    // complete[r3,r1] and complete[r3,r2] and complete[r3,r3]
    Formula rules = f1.and(f2).and(f3);
    for(Relation rx: regions)
      for(Relation ry: regions)
        rules = rules.and(complete(rx,ry));
    return rules;
  }

  public Bounds puzzle() {
    Set<Integer> atoms = new LinkedHashSet<Integer>(9);
    for(int i = 1; i <= 9; i++) { atoms.add(i); }
    Universe u = new Universe(atoms);
    Bounds b = new Bounds(u);
```

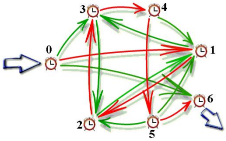
```
    TupleFactory f = u.factory();
    b.boundExactly(Number, f.allOf(1));
    b.boundExactly(regions[0], f.setOf(1, 2, 3));
    b.boundExactly(regions[1], f.setOf(4, 5, 6));
    b.boundExactly(regions[2], f.setOf(7, 8, 9));

    TupleSet givens = f.noneOf(3);
    givens.add(f.tuple(1, 4, 1));
    givens.add(f.tuple(1, 8, 9));
    ...
    givens.add(f.tuple(9, 6, 1));
    b.bound(data, givens, f.allOf(3));
    return b;
  }

  public static void main(String[] args) {
    Solver solver = new Solver();
    solver.options().setSolver(SATFactory.MiniSat);
    Sudoku sudoku = new Sudoku();
    Solution sol = solver.solve(sudoku.rules(), sudoku.puzzle());
    System.out.println(sol);
  }
}
```

SQUANDER– Summary

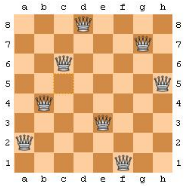
- freely mix imperative code and declarative specifications
- execute specifications as part of a Java program

```
public Sudoku() {}

@Ensures({
  "all row in {0 ... (this.n - 1)} | this.data[row][int] = {1 ... this.n}",
  "all col in {0 ... (this.n - 1)} | this.data[int][col] = {1 ... this.n}",
  "all r1, r2 in this.regions.vals | this.data[r1.vals][r2.vals] = {1 ... this.n}"})
@Modifies("this.data[int].elems | _<2> = 0")
public void solve() { Squander.exe(this); }
```

- conveniently express and solve constraint problems in place
  - even gain performance for certain problems



Hamiltonian Path

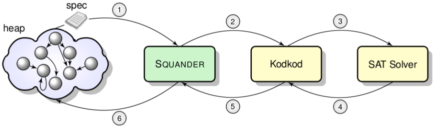


n-Queens

Outline

Framework Overview

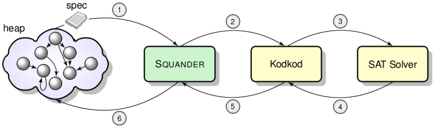
- specification language
- SQUANDER architecture



Outline

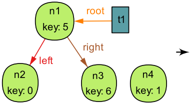
Framework Overview

- specification language
- SQUANDER architecture



Translation

- from Java heap + specs to Kodkod
- minimizing the universe size



BST1:	$\{t_1\}$	**N3**:	$\{n_3\}$	**BST_this**:	$\{t_1\}$
N1:	$\{n_1\}$	**N4**:	$\{n_4\}$	**z**:	$\{n_4\}$
N2:	$\{n_2\}$	**null**:	$\{null\}$	**ints**:	$\{0, 1, 5, 6\}$

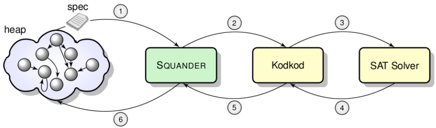
key_pre: $\{(n_1 \rightarrow 5), (n_2 \rightarrow 0), (n_3 \rightarrow 6), (n_4 \rightarrow 1)\}$
root_pre: $\{(t_1 \rightarrow n_1)\}$
left_pre: $\{(n_1 \rightarrow n_2), (n_2 \rightarrow null), (n_3 \rightarrow null), (n_4 \rightarrow null)\}$
right_pre: $\{(n_1 \rightarrow n_3), (n_2 \rightarrow null), (n_3 \rightarrow null), (n_4 \rightarrow null)\}$

root: $\{\}$, $\{t_1\} \times \{n_1, n_2, n_3, n_4\}$
left: $\{\}$, $\{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$
right: $\{\}$, $\{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$

Outline

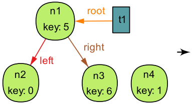
Framework Overview

- specification language
- SQUANDER architecture



Translation

- from Java heap + specs to Kodkod
- minimizing the universe size



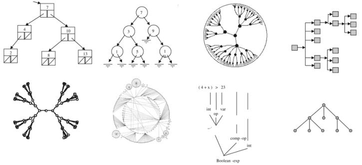
BST1: $\{t_1\}$ | **N3**: $\{n_3\}$ | **BST_this**: $\{t_1\}$
N1: $\{n_1\}$ | **N4**: $\{n_4\}$ | **z**: $\{n_4\}$
N2: $\{n_2\}$ | **null**: $\{null\}$ | **ints**: $\{0, 1, 5, 6\}$

key_pre: $\{(n_1 \to 5), (n_2 \to 0), (n_3 \to 6), (n_4 \to 1)\}$
root_pre: $\{(t_1 \to n_1)\}$
left_pre: $\{(n_1 \to n_2), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$
right_pre: $\{(n_1 \to n_3), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$

root: $\{\}$, $\{t_1\} \times \{n_1, n_2, n_3, n_4\}$
left: $\{\}$, $\{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$
right: $\{\}$, $\{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$

Treatment of Data Abstractions

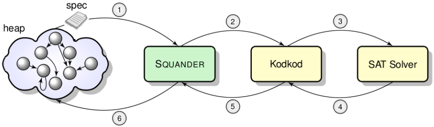
- support for third party library classes (e.g. Java collections)



Outline

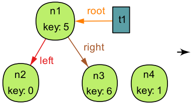
Framework Overview

- specification language
- SQUANDER architecture


spec
heap
SQUANDER
Kodkod
SAT Solver

Translation

- from Java heap + specs to Kodkod
- minimizing the universe size



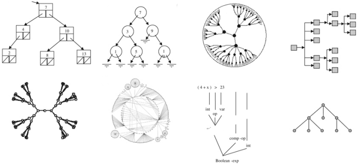
BST1:	$\{t_1\}$	**N3**:	$\{n_3\}$	**BST_this**:	$\{t_1\}$
N1:	$\{n_1\}$	**N4**:	$\{n_4\}$	**z**:	$\{n_4\}$
N2:	$\{n_2\}$	**null**:	$\{null\}$	**ints**:	$\{0, 1, 5, 6\}$

key_pre: $\{(n_1 \rightarrow 5), (n_2 \rightarrow 0), (n_3 \rightarrow 6), (n_4 \rightarrow 1)\}$
root_pre: $\{(t_1 \rightarrow n_1)\}$
left_pre: $\{(n_1 \rightarrow n_2), (n_2 \rightarrow null), (n_3 \rightarrow null), (n_4 \rightarrow null)\}$
right_pre: $\{(n_1 \rightarrow n_3), (n_2 \rightarrow null), (n_3 \rightarrow null), (n_4 \rightarrow null)\}$

root: $\{\}, \quad \{t_1\} \times \{n_1, n_2, n_3, n_4\}$
left: $\{\}, \quad \{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$
right: $\{\}, \quad \{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$

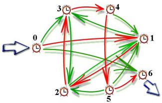
Treatment of Data Abstractions

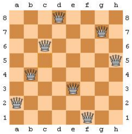
- support for third party library classes (e.g. Java collections)



Evaluation/Case Study

- performance advantages for some puzzles and graph algorithms
- case study: *MIT course scheduler*

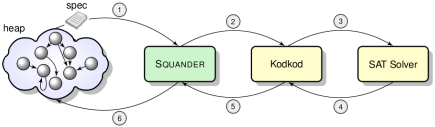




Framework Overview

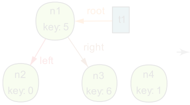
Framework Overview

- specification language
- SQUANDER architecture



Translation

- from Java heap + specs to Kodkod
- minimizing the universe size



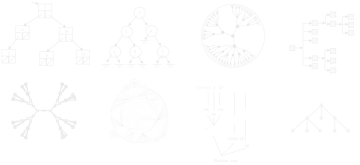
BST1: $\{t_1\}$ | N3: $\{n_3\}$ | BST_this: $\{t_1\}$
N1: $\{n_1\}$ | N4: $\{n_4\}$ | z: $\{n_4\}$
N2: $\{n_2\}$ | null: $\{null\}$ | ints: $\{0, 1, 5, 6\}$

key_pre: $\{(n_1 \to 5), (n_2 \to 0), (n_3 \to 6), (n_4 \to 1)\}$
root_pre: $\{(t_1 \to n_1)\}$
left_pre: $\{(n_1 \to n_2), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$
right_pre: $\{(n_1 \to n_3), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$

root: $\{\}$, $\{t_1\} \times \{n_1, n_2, n_3, n_4\}$
left: $\{\}$, $\{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$
right: $\{\}$, $\{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$

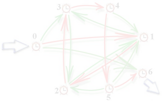
Treatment of Data Abstractions

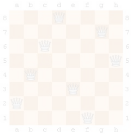
- support for third party library classes (e.g. Java collections)



Evaluation/Case Study

- performance advantages for some puzzles and graph algorithms
- case study: *MIT course scheduler*





Specification Language

Example - Binary Search Tree

```
public class BST {
  private BSTNode root;
}
```

```
public class BSTNode {
  private BSTNode left, right;
  private int key;
}
```

Specification Language

Example - Binary Search Tree

```
public class BST {
  private BSTNode root;
}
```

```
public class BSTNode {
  private BSTNode left, right;
  private int key;
}
```

Annotations

class specification field **@SpecField** ("<fld_decl> | <abs_func>")

Specification Language

Example - Binary Search Tree

```
public class BST {
  private BSTNode root;
}
```

```
public class BSTNode {
  private BSTNode left, right;
  private int key;
}
```

Annotations

class specification field **@SpecField** ("<fld_decl> | <abs_func>")

```
    @SpecField("this.nodes: set BSTNode | this.nodes = this.root.*(left+right) - null")
    public class BST {
```

Specification Language

Example - Binary Search Tree

```
public class BST {
  private BSTNode root;
}
```

```
public class BSTNode {
  private BSTNode left, right;
  private int key;
}
```

Annotations

class specification field **@SpecField** ("<fld_decl> | <abs_func>")

```
@SpecField("this.nodes: set BSTNode | this.nodes = this.root.*(left+right) − null")
public class BST {
```

class invariant **@Invariant** ("<expr>")

Specification Language

Example - Binary Search Tree

```
public class BST {
  private BSTNode root;
}
```

```
public class BSTNode {
  private BSTNode left, right;
  private int key;
}
```

Annotations

class specification field **@SpecField** ("<fld_decl> | <abs_func>")

```
@SpecField("this.nodes: set BSTNode | this.nodes = this.root.*(left+right) - null")
public class BST {
```

class invariant **@Invariant** ("<expr>")

```
@Invariant({
  /* left sorted  */ "all x: this.left.*(left+right) - null | x.key < this.key",
  /* right sorted */ "all x: this.right.*(left+right) - null | x.key > this.key"})
public class BSTNode {
```

Specification Language

Example - Binary Search Tree

```
public class BST {
  private BSTNode root;
}
```

```
public class BSTNode {
  private BSTNode left, right;
  private int key;
}
```

Annotations

class specification field **@SpecField** ("<fld_decl> | <abs_func>")

```
@SpecField("this.nodes: set BSTNode | this.nodes = this.root.*(left+right) − null")
public class BST {
```

class invariant **@Invariant** ("<expr>")

```
@Invariant({
  /* left sorted  */ "all x: this.left.*(left+right) − null | x.key < this.key",
  /* right sorted */ "all x: this.right.*(left+right) − null | x.key > this.key"})
public class BSTNode {
```

method pre-condition **@Requires** ("<expr>")
method post-condition **@Ensures** ("<expr>")
method frame condition **@Modifies** ("<fld> | <filter>")

Specification Language

Example - Binary Search Tree

```
public class BST {
  private BSTNode root;
}
```

```
public class BSTNode {
  private BSTNode left, right;
  private int key;
}
```

Annotations

class specification field **@SpecField** ("<fld_decl> | <abs_func>")

```
    @SpecField("this.nodes: set BSTNode | this.nodes = this.root.∗(left+right) − null")
    public class BST {
```

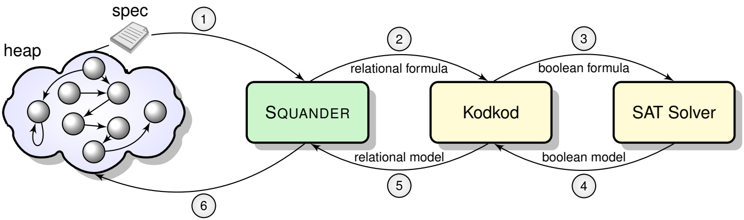
class invariant **@Invariant** ("<expr>")

```
    @Invariant({
      /∗ left sorted  ∗/ "all x: this.left.∗(left+right) − null | x.key < this.key",
      /∗ right sorted ∗/ "all x: this.right.∗(left+right) − null | x.key > this.key"})
    public class BSTNode {
```

method pre-condition **@Requires** ("<expr>")
method post-condition **@Ensures** ("<expr>")
method frame condition **@Modifies** ("<fld> | <filter>")

```
    @Requires("z.key !in this.nodes.key")
    @Ensures ("this.nodes = @old(this.nodes) + z")
    @Modifies("this.root, this.nodes.left | _<1> = null, this.nodes.right | _<1> = null")
    public BST insertNode(BSTNode z) { Squander.exe(this, z); }
```

Framework Overview



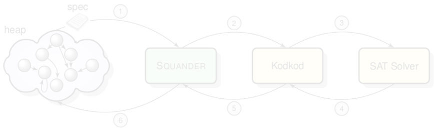
Execution steps

- traverse the heap and assemble the relevant constraints
- translate to Kodkod
    - translate the heap to relations and bounds
    - collect all the specs and assemble a single relational formula
- if a solution is found, update the heap to reflect the solution

Translation

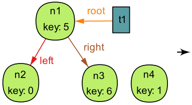
Framework Overview

- specification language
- SQUANDER architecture



Translation

- from Java heap + specs to Kodkod
- minimizing the universe size



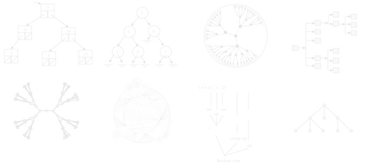
BST1: $\{t_1\}$ | **N3**: $\{n_3\}$ | **BST_this**: $\{t_1\}$
N1: $\{n_1\}$ | **N4**: $\{n_4\}$ | **z**: $\{n_4\}$
N2: $\{n_2\}$ | **null**: $\{null\}$ | **ints**: $\{0, 1, 5, 6\}$

key_pre: $\{(n_1 \to 5), (n_2 \to 0), (n_3 \to 6), (n_4 \to 1)\}$
root_pre: $\{(t_1 \to n_1)\}$
left_pre: $\{(n_1 \to n_2), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$
right_pre: $\{(n_1 \to n_3), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$

root: $\{\}, \quad \{t_1\} \times \{n_1, n_2, n_3, n_4\}$
left: $\{\}, \quad \{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$
right: $\{\}, \quad \{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$

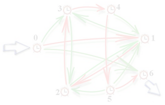
Treatment of Data Abstractions

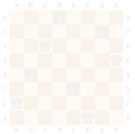
- support for third party library classes (e.g. Java collections)



Evaluation/Case Study

- performance advantages for some puzzles and graph algorithms
- case study: *MIT course scheduler*





From Objects to Relations

The back-end solver — Kodkod

- constraint solver for first-order logic with relations
- SAT-based finite relational model finder
  - finite bounds must be provided for all relations
- designed to be efficient for partial models
  - partial instances are encoded using bounds

From Objects to Relations

The back-end solver — Kodkod

- constraint solver for first-order logic with relations
- SAT-based finite relational model finder
  - finite bounds must be provided for all relations
- designed to be efficient for partial models
  - partial instances are encoded using bounds

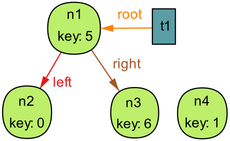
Everything is a relation

					relation name		relation type
classes	⇝	unary relations	**class** C {}	⇝	$\mathscr{R}_{C}$	:	C
objects	⇝	unary relations	**new** C();	⇝	$\mathscr{R}_{C_1}$	:	C
fields	⇝	binary relations	**class** C { A fld ; }	⇝	$\mathscr{R}_{fld}$	:	$C \to A \cup \{\textbf{null}\}$
arrays	⇝	ternary relations	T[]	⇝	$\mathscr{R}_{T[]\_elems}$	:	$T[] \to \textbf{int} \to T \cup \{\textbf{null}\}$

From Objects to Relations

Translation of the `BST.insert` method

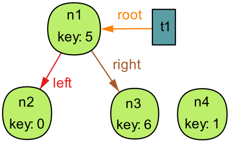
```
@Requires("z.key !in this.nodes.key")
@Ensures ("this.nodes = @old(this.nodes) + z")
@Modifies("this.root, this.nodes.left | _<1> = null, this.nodes.right | _<1> = null")
public BST insertNode(BSTNode z) { Squander.exe(this, z); }
```



From Objects to Relations

Translation of the `BST.insert` method

```
@Requires("z.key !in this.nodes.key")
@Ensures ("this.nodes = @old(this.nodes) + z")
@Modifies("this.root, this.nodes.left | _<1> = null, this.nodes.right | _<1> = null")
public BST insertNode(BSTNode z) { Squander.exe(this, z); }
```



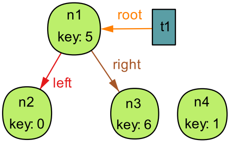
BST1: $\{t_1\}$
N1: $\{n_1\}$
N2: $\{n_2\}$
N3: $\{n_3\}$
N4: $\{n_4\}$
null: $\{null\}$
BST_this: $\{t_1\}$
z: $\{n_4\}$
ints: $\{0,1,5,6\}$

reachable objects

From Objects to Relations

Translation of the `BST.insert` method

```
@Requires("z.key !in this.nodes.key")
@Ensures ("this.nodes = @old(this.nodes) + z")
@Modifies("this.root, this.nodes.left | _<1> = null, this.nodes.right | _<1> = null")
public BST insertNode(BSTNode z) { Squander.exe(this, z); }
```



BST1:	$\{t_1\}$	**N3**:	$\{n_3\}$	**BST_this**:	$\{t_1\}$
N1:	$\{n_1\}$	**N4**:	$\{n_4\}$	**z**:	$\{n_4\}$
N2:	$\{n_2\}$	**null**:	$\{null\}$	**ints**:	$\{0, 1, 5, 6\}$

reachable objects

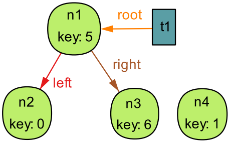
key_pre: $\{(n_1 \rightarrow 5), (n_2 \rightarrow 0), (n_3 \rightarrow 6), (n_4 \rightarrow 1)\}$
root_pre: $\{(t_1 \rightarrow n_1)\}$
left_pre: $\{(n_1 \rightarrow n_2), (n_2 \rightarrow null), (n_3 \rightarrow null), (n_4 \rightarrow null)\}$
right_pre: $\{(n_1 \rightarrow n_3), (n_2 \rightarrow null), (n_3 \rightarrow null), (n_4 \rightarrow null)\}$

pre-state

From Objects to Relations

Translation of the `BST.insert` method

```
@Requires("z.key !in this.nodes.key")
@Ensures ("this.nodes = @old(this.nodes) + z")
@Modifies("this.root, this.nodes.left | _<1> = null, this.nodes.right | _<1> = null")
public BST insertNode(BSTNode z) { Squander.exe(this, z); }
```



reachable objects

BST1: $\{t_1\}$
N1: $\{n_1\}$
N2: $\{n_2\}$
N3: $\{n_3\}$
N4: $\{n_4\}$
null: $\{null\}$
BST_this: $\{t_1\}$
z: $\{n_4\}$
ints: $\{0, 1, 5, 6\}$

pre-state

key_pre: $\{(n_1 \to 5), (n_2 \to 0), (n_3 \to 6), (n_4 \to 1)\}$
root_pre: $\{(t_1 \to n_1)\}$
left_pre: $\{(n_1 \to n_2), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$
right_pre: $\{(n_1 \to n_3), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$

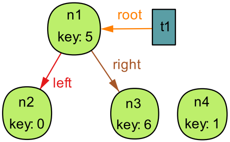
post-state

	lower bound	upper bound
root:	$\{\}$,	$\{t_1\} \times \{n_1, n_2, n_3, n_4, null\}$
left:	$\{n_1 \to n_2\}$,	$\{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4, null\}$
right:	$\{n_1 \to n_3\}$,	$\{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4, null\}$

From Objects to Relations

Translation of the `BST.insert` method

```
@Requires("z.key !in this.nodes.key")
@Ensures ("this.nodes = @old(this.nodes) + z")
@Modifies("this.root, this.nodes.left | _<1> = null, this.nodes.right | _<1> = null")
public BST insertNode(BSTNode z) { Squander.exe(this, z); }
```



BST1: $\{t_1\}$
N1: $\{n_1\}$
N2: $\{n_2\}$
N3: $\{n_3\}$
N4: $\{n_4\}$
null: $\{null\}$
BST_this: $\{t_1\}$
z: $\{n_4\}$
ints: $\{0, 1, 5, 6\}$

reachable objects

key_pre: $\{(n_1 \to 5), (n_2 \to 0), (n_3 \to 6), (n_4 \to 1)\}$
root_pre: $\{(t_1 \to n_1)\}$
left_pre: $\{(n_1 \to n_2), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$
right_pre: $\{(n_1 \to n_3), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$

pre-state

root:	$\{\}$,	$\{t_1\} \times \{n_1, n_2, n_3, n_4, null\}$
left:	$\{n_1 \to n_2\}$,	$\{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4, null\}$
right:	$\{n_1 \to n_3\}$,	$\{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4, null\}$

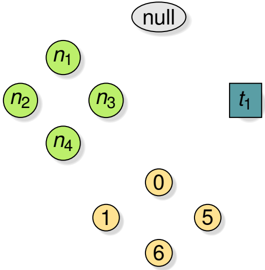
post-state

lower bound upper bound

- **lower bound**: tuples that must be included
- **upper bound**: tuples that may be included
- shrinking the bounds (instead of adding more constraints) leads to more efficient solving

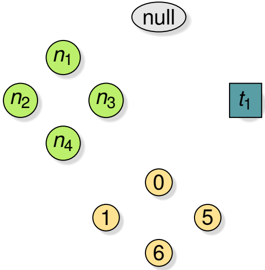
Minimizing the Universe

goal: use fewer Kodkod atoms than heap objects



Minimizing the Universe

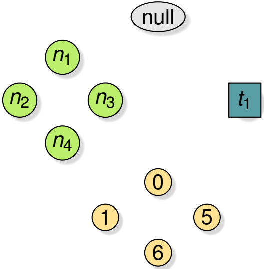
goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective



Minimizing the Universe

goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective

also: must be able to unambiguously restore the heap
→ *instances of the same type must map to distinct atoms*



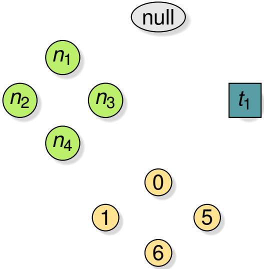
Minimizing the Universe

goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective

also: must be able to unambiguously restore the heap
→ *instances of the same type must map to distinct atoms*

algorithm

1. discover all used types (clusters)



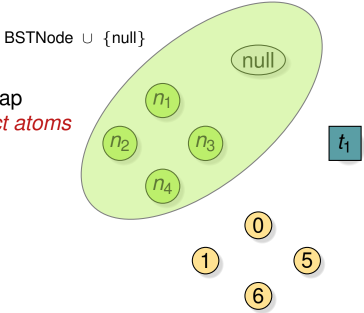
Minimizing the Universe

goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective

also: must be able to unambiguously restore the heap
→ *instances of the same type must map to distinct atoms*

algorithm

1. discover all used types (clusters)



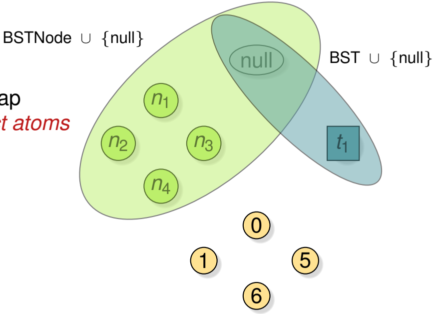
Minimizing the Universe

goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective

also: must be able to unambiguously restore the heap
→ *instances of the same type must map to distinct atoms*

algorithm

1. discover all used types (clusters)



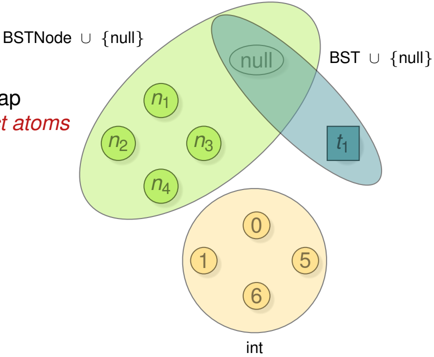
Minimizing the Universe

goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective

also: must be able to unambiguously restore the heap
→ *instances of the same type must map to distinct atoms*

algorithm

1. discover all used types (clusters)



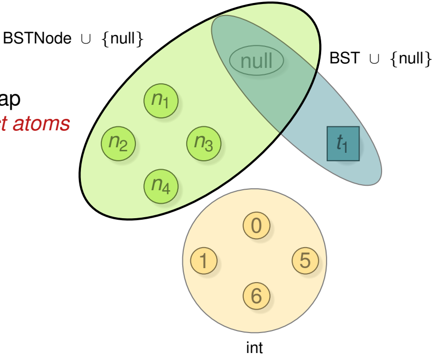
Minimizing the Universe

goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective

also: must be able to unambiguously restore the heap
→ *instances of the same type must map to distinct atoms*

algorithm

1. discover all used types (clusters)
2. find the largest cluster



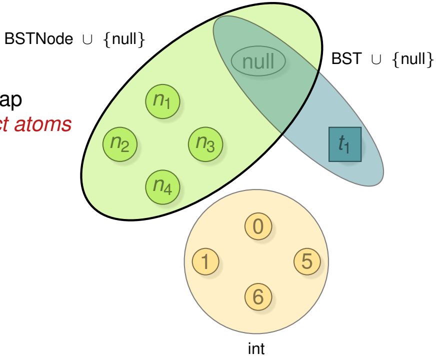
Minimizing the Universe

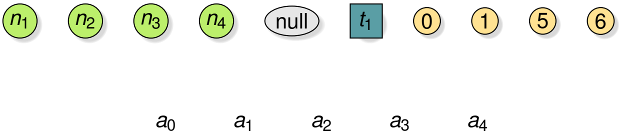
goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective

also: must be able to unambiguously restore the heap
→ *instances of the same type must map to distinct atoms*

algorithm

1. discover all used types (clusters)
2. find the largest cluster
3. create that many atoms





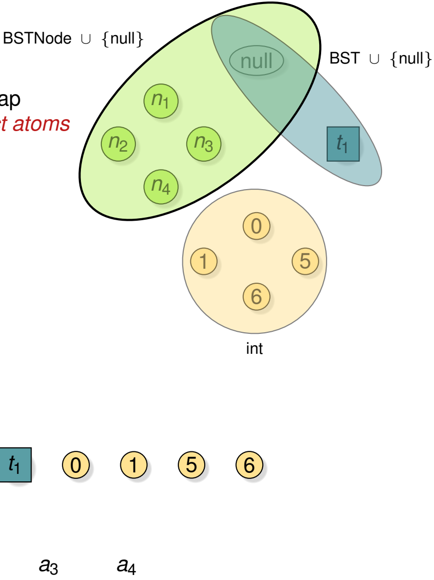
Minimizing the Universe

goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective

also: must be able to unambiguously restore the heap
→ *instances of the same type must map to distinct atoms*

algorithm

1. discover all used types (clusters)
2. find the largest cluster
3. create that many atoms
4. assign atoms to instances



Minimizing the Universe

goal: use fewer Kodkod atoms than heap objects

→ multiple objects must map to same atoms

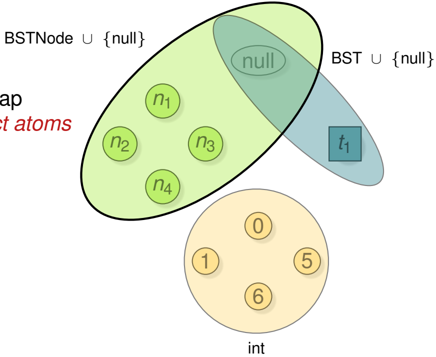
→ mapping from objects to atoms is not injective

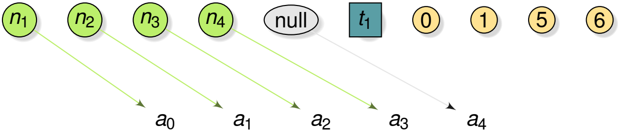
also: must be able to unambiguously restore the heap

→ *instances of the same type must map to distinct atoms*

algorithm

1. discover all used types (clusters)
2. find the largest cluster
3. create that many atoms
4. assign atoms to instances





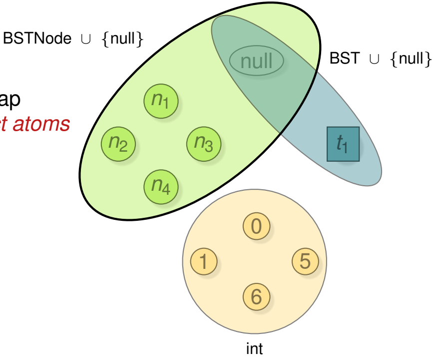
Minimizing the Universe

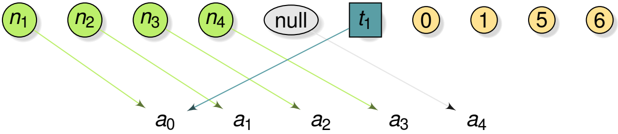
goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective

also: must be able to unambiguously restore the heap
→ *instances of the same type must map to distinct atoms*

algorithm

1. discover all used types (clusters)
2. find the largest cluster
3. create that many atoms
4. assign atoms to instances





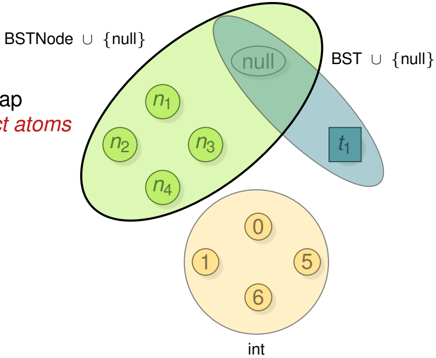
Minimizing the Universe

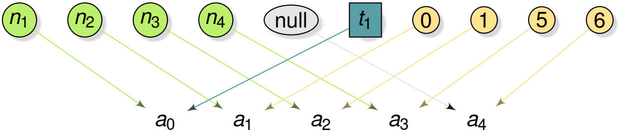
goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective

also: must be able to unambiguously restore the heap
→ *instances of the same type must map to distinct atoms*

algorithm

1. discover all used types (clusters)
2. find the largest cluster
3. create that many atoms
4. assign atoms to instances

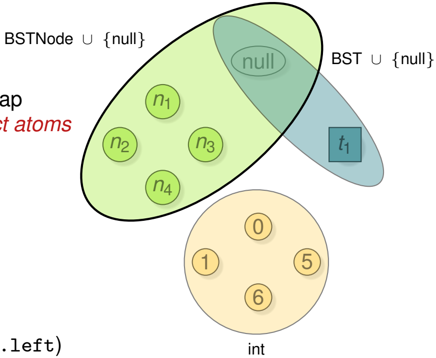




Minimizing the Universe

goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective

also: must be able to unambiguously restore the heap
→ *instances of the same type must map to distinct atoms*

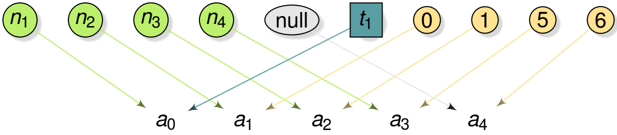


algorithm

1. discover all used types (clusters)
2. find the largest cluster
3. create that many atoms
4. assign atoms to instances

restoring field values (e.g. a_0 for the field `BSTNode.left`)

1. based on the field's type, select its cluster
2. select the instance from that cluster that maps to the given atom



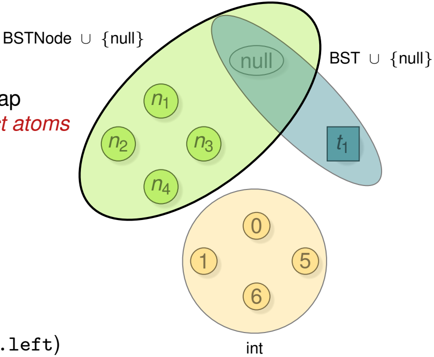
Minimizing the Universe

goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective

also: must be able to unambiguously restore the heap
→ *instances of the same type must map to distinct atoms*

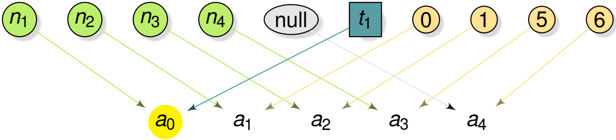
algorithm

1. discover all used types (clusters)
2. find the largest cluster
3. create that many atoms
4. assign atoms to instances



restoring field values (e.g. a_0 for the field `BSTNode.left`)

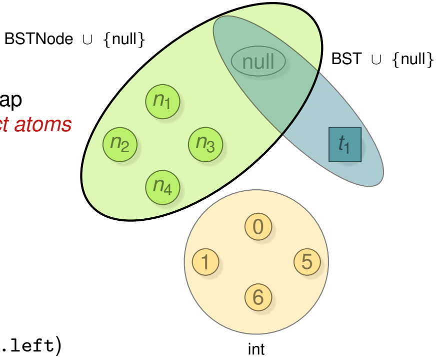
1. based on the field's type, select its cluster
2. select the instance from that cluster that maps to the given atom



Minimizing the Universe

goal: use fewer Kodkod atoms than heap objects
→ multiple objects must map to same atoms
→ mapping from objects to atoms is not injective

also: must be able to unambiguously restore the heap
→ *instances of the same type must map to distinct atoms*

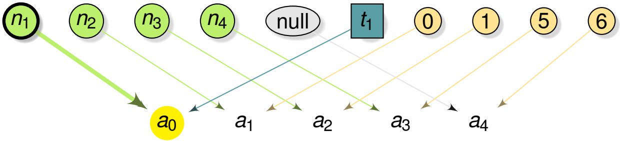


algorithm

1. discover all used types (clusters)
2. find the largest cluster
3. create that many atoms
4. assign atoms to instances

restoring field values (e.g. a_0 for the field `BSTNode.left`)

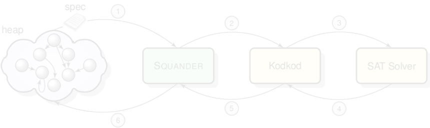
1. based on the field's type, select its cluster
2. select the instance from that cluster that maps to the given atom



Treatment of Data Abstractions

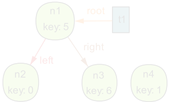
Framework Overview

- specification language
- SQUANDER architecture



Translation

- from Java heap + specs to Kodkod
- minimizing the universe size



→

BST1: $\{t_1\}$ | **N3:** $\{n_3\}$ | **BST_this:** $\{t_1\}$

N1: $\{n_1\}$ | **N4:** $\{n_4\}$ | **z:** $\{n_4\}$

N2: $\{n_2\}$ | **null:** $\{null\}$ | **ints:** $\{0, 1, 5, 6\}$

key_pre: $\{(n_1 \to 5), (n_2 \to 0), (n_3 \to 6), (n_4 \to 1)\}$

root_pre: $\{(t_1 \to n_1)\}$

left_pre: $\{(n_1 \to n_2), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$

right_pre: $\{(n_1 \to n_3), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$

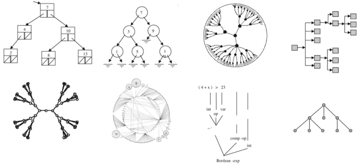
root: $\{\}, \quad \{t_1\} \times \{n_1, n_2, n_3, n_4\}$

left: $\{\}, \quad \{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$

right: $\{\}, \quad \{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$

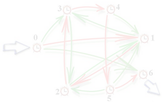
Treatment of Data Abstractions

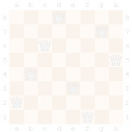
- support for third party library classes (e.g. Java collections)



Evaluation/Case Study

- performance advantages for some puzzles and graph algorithms
- case study: *MIT course scheduler*





User-Defined Abstractions for Library Types

Why is it important to be able to specify library types?

- library classes are ubiquitous
- specs need to refer to the *content* of library types
  (e.g. iterate through set elements, get all keys of a map, etc.)
- we don't want to have to change existing code in order to be able to specify a method

```
class Graph {
  class Node { public int key; }
  class Edge { public Node src, dest; }

  private Set<Node> nodes = new LinkedHashSet<Node>();
  private Set<Edge> edges = new LinkedHashSet<Edge>();

  // how to write a spec for the k-Coloring problem for a graph like this?
  public Map<Node, Integer> k_color(int k) { return Squander.exe(this, k); }
}
```

- we certainly don't want to use concrete fields of `LinkedHashSet` in the spec

User-Defined Abstractions for Library Types

Why is it important to be able to specify library types?

- library classes are ubiquitous
- specs need to refer to the *content* of library types
  (e.g. iterate through set elements, get all keys of a map, etc.)
- we don't want to have to change existing code in order to be able to specify a method

```
class Graph {
  class Node { public int key; }
  class Edge { public Node src, dest; }

  private Set<Node> nodes = new LinkedHashSet<Node>();
  private Set<Edge> edges = new LinkedHashSet<Edge>();

  // how to write a spec for the k-Coloring problem for a graph like this?
  public Map<Node, Integer> k_color(int k) { return Squander.exe(this, k); }
}
```

- we certainly don't want to use concrete fields of `LinkedHashSet` in the spec
- solution:
  - use `@SpecField` to specify abstract data types

User-Defined Abstractions for Library Types

How to support a third party class?

- write a spec file

```
interface Map<K,V> {
  @SpecField("elts: K -> V")

  @SpecField("size: one int | this.size = #this.elts")
  @SpecField("keys: set K | this.keys = this.elts.(V)")
  @SpecField("vals: set V | this.vals = this.elts[K]")

  @Invariant({"all k: K | k in this.elts.V => one this.elts[k]"})
}
```

User-Defined Abstractions for Library Types

How to support a third party class?

- write a spec file

```
interface Map<K,V> {
  @SpecField("elts: K -> V")

  @SpecField("size: one int | this.size = #this.elts")
  @SpecField("keys: set K | this.keys = this.elts.(V)")
  @SpecField("vals: set V | this.vals = this.elts[K]")

  @Invariant({"all k: K | k in this.elts.V => one this.elts[k]"})
}
```

- write an **abstraction** and a **concretization** function

```
public class MapSer implements IObjSer {
  public static final String ELTS = "elts";

  public List<FieldValue> absFunc(JavaScene javaScene, Object obj) {
    Map<Object, Object> map = (Map<Object, Object>) obj;
    // return values for the field "elts": Map -> K -> V
  }

  public Object concrFunc(Object obj, FieldValue fieldValue) {
    // update and return the given object "obj" from
    // the given values of the given abstract field
  }
}
```

User-Defined Abstractions for Library Types

Now we can specify the k-Coloring problem

```
class Graph {
  class Node { public int key; }
  class Edge { public Node src, dest; }

  private Set<Node> nodes = new LinkedHashSet<Node>();
  private Set<Edge> edges = new LinkedHashSet<Edge>();

  @Ensures({
    "return.keys = this.nodes.elts",
    "return.vals in {1 ... k}",
    "all e : this.edges.elts | return.elts[e.src] != return.elts[e.dst]"})
  @Modifies("return.elts")
  @FreshObjects(cls = Map.class, typeParams={Node.class, Integer.class}, num = 1)
  public Map<Node, Integer> color(int k) { return Squander.exe(this, k); }
}
```

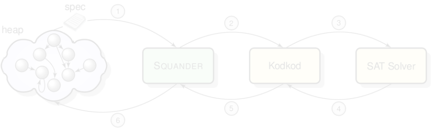
```
interface Set<K> {
  @SpecField("elts: set K")

  @SpecField("size: one int |
    this.size=#this.elts")
}
```

```
interface Map<K,V> {
  @SpecField("elts: K -> V")

  @SpecField("size: one int | this.size = #this.elts")
  @SpecField("keys: set K | this.keys = this.elts.(V)")
  @SpecField("vals: set V | this.vals = this.elts[K]")

  @Invariant({"all k: K | k in this.elts.V => one this.elts[k]"})
}
```

Evaluation/Case Study

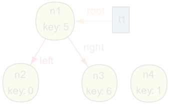
Framework Overview

- specification language
- SQUANDER architecture



Translation

- from Java heap + specs to Kodkod
- minimizing the universe size



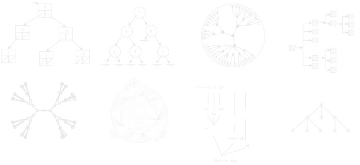
BST1: $\{t_1\}$ | **N3:** $\{n_3\}$ | **BST_this:** $\{t_1\}$
N1: $\{n_1\}$ | **N4:** $\{n_4\}$ | **z:** $\{n_4\}$
N2: $\{n_2\}$ | **null:** $\{null\}$ | **ints:** $\{0, 1, 5, 6\}$

key_pre: $\{(n_1 \to 5), (n_2 \to 0), (n_3 \to 6), (n_4 \to 1)\}$
root_pre: $\{(t_1 \to n_1)\}$
left_pre: $\{(n_1 \to n_2), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$
right_pre: $\{(n_1 \to n_3), (n_2 \to null), (n_3 \to null), (n_4 \to null)\}$

root: $\{\}, \quad \{t_1\} \times \{n_1, n_2, n_3, n_4\}$
left: $\{\}, \quad \{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$
right: $\{\}, \quad \{n_1, n_2, n_3, n_4\} \times \{n_1, n_2, n_3, n_4\}$

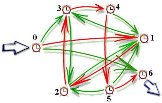
Treatment of Data Abstractions

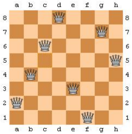
- support for third party library classes (e.g. Java collections)



Evaluation/Case Study

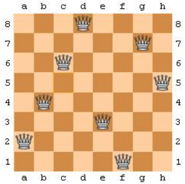
- performance advantages for some puzzles and graph algorithms
- case study: *MIT course scheduler*



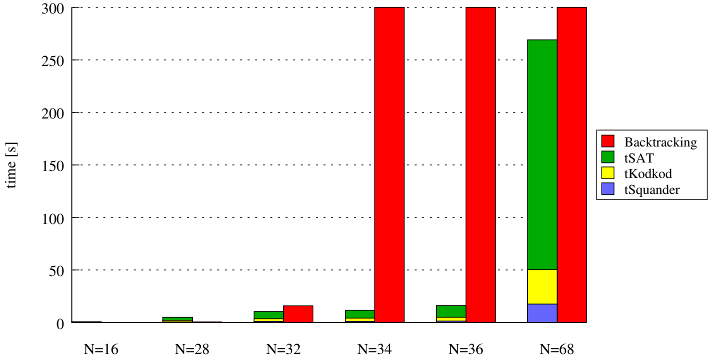


SQUANDER vs Manual Search

N-Queens



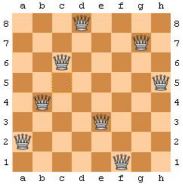
- place N queens on an N×N chess board such that no two queens attack each other

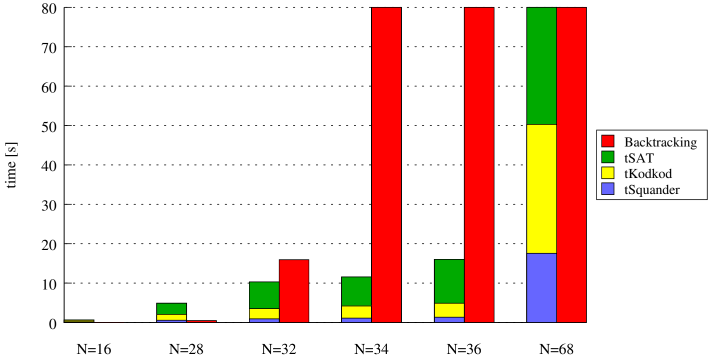


SQUANDER vs Manual Search

N-Queens

- place N queens on an N×N chess board such that no two queens attack each other

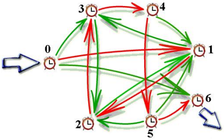




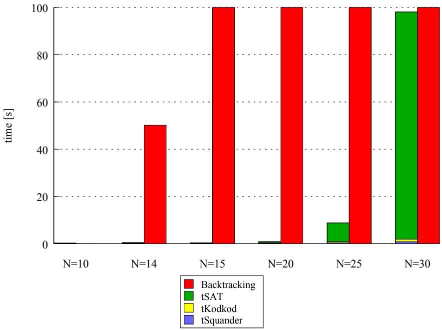
SQUANDER vs Manual Search

Hamiltonian Path

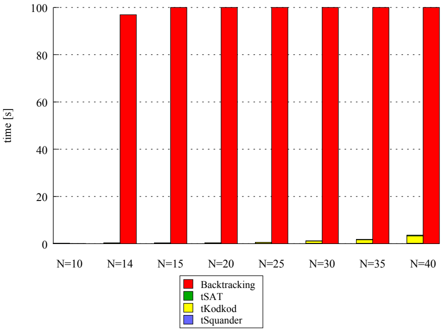
- find a path in a graph that visits all nodes exactly once



Graphs with Hamiltonian path



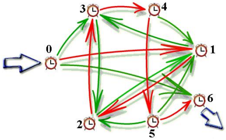
Graphs with no Hamiltonian path



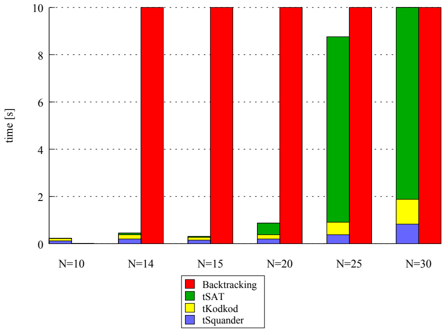
SQUANDER vs Manual Search

Hamiltonian Path

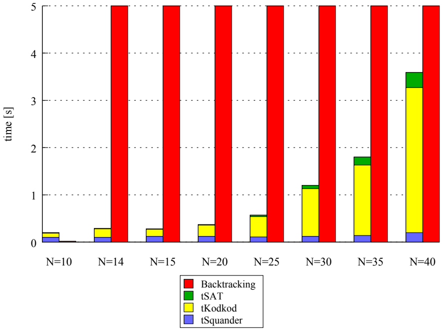
- find a path in a graph that visits all nodes exactly once



Graphs with Hamiltonian path



Graphs with no Hamiltonian path

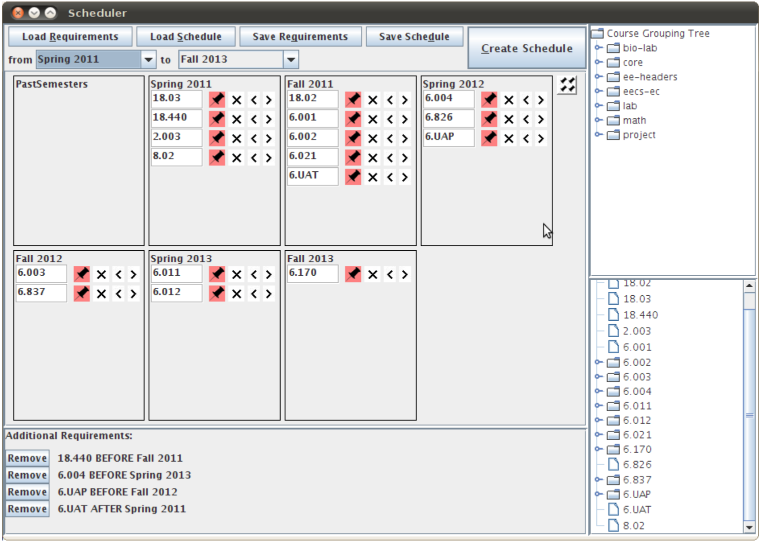


Case Study – Course Scheduler

Course Scheduler

- assign courses to semester to complete graduation requirements
- the program offers around 300 courses, more than 150 of them have prerequisites
- additional requirements:
  - mandatory courses
  - choices from course groups
  - no overlapping between course groups
  - time requirements
  - student specified requirements, etc.

Course Scheduler GUI



from Spring 2011 to Fall 2013

PastSemesters

Spring 2011: 18.03, 18.440, 2.003, 8.02

Fall 2011: 18.02, 6.001, 6.002, 6.021, 6.UAT

Spring 2012: 6.004, 6.826, 6.UAP

Fall 2012: 6.003, 6.837

Spring 2013: 6.011, 6.012

Fall 2013: 6.170

Additional Requirements:

18.440 BEFORE Fall 2011
6.004 BEFORE Spring 2013
6.UAP BEFORE Fall 2012
6.UAT AFTER Spring 2011

Case Study – Course Scheduler

Evaluation questions

Case Study – Course Scheduler

Evaluation questions

- **usability of** SQUANDER **on a real-world constraint problem**
    - an existing implementation retrofitted with SQUANDER
    - didn't have to change the local structure, just annotate classes
    - ... thanks to the **treatment of data abstractions**

Case Study – Course Scheduler

Evaluation questions

- **usability of** SQUANDER **on a real-world constraint problem**
  - an existing implementation retrofitted with SQUANDER
  - didn't have to change the local structure, just annotate classes
  - ... thanks to the **treatment of data abstractions**
- **annotation overhead**
  - only about 30 lines of specs to replace 1500 lines of code
  - ... thanks to the **unified execution environment**

Case Study – Course Scheduler

Evaluation questions

- **usability of SQUANDER on a real-world constraint problem**
  - an existing implementation retrofitted with SQUANDER
  - didn't have to change the local structure, just annotate classes
  - ... thanks to the **treatment of data abstractions**
- **annotation overhead**
  - only about 30 lines of specs to replace 1500 lines of code
  - ... thanks to the **unified execution environment**
- **ability to handle large program heaps**
  - the heap counted almost 2000 objects
  - ... thanks to the **partitioning algorithm**

Case Study – Course Scheduler

Evaluation questions

- **usability of SQUANDER on a real-world constraint problem**
  - an existing implementation retrofitted with SQUANDER
  - didn't have to change the local structure, just annotate classes
  - ... thanks to the **treatment of data abstractions**
- **annotation overhead**
  - only about 30 lines of specs to replace 1500 lines of code
  - ... thanks to the **unified execution environment**
- **ability to handle large program heaps**
  - the heap counted almost 2000 objects
  - ... thanks to the **partitioning algorithm**
- **efficiency**
  - about 5s as opposed to 1s of the original implementation

Limitations

- **boundedness** – SQUANDER can't generate an arbitrary number of new objects; instead the maximum number of new objects must be explicitly specified by the user
- **integers** – integers must also be bounded to a small bitwidth
- **equality** – only referential equality can be used (except for strings)
- **no higher-order expressions** – e.g. can't specify *find the longest path in the graph*; instead must specify the minimum length k, i.e. *find a path in the graph of length at least* k *nodes*
- **unsat core** – if a solution cannot be found, the user is not given any additional information as to why the specification wasn't satisfiable

Acknowledgements

Derek Rayside

Kuat Yessenov

Daniel Jackson

Related Work

Executable Specifications:

- **An Overview of Some Formal Methods for Program Design**, C.A.R. Hoare (*IEEE Computer 1987*)
- **Specifications are not (necessarily) executable**, I. Hayes et al. (*SEJ 1989*)
- **Specifications are (preferably) executable**, N.E. Fuchs (*SEJ 1992*)
- **Programming from Specification**, C. Morgan, PrenticeHall, 1998
- **Agile Specifications**, D. Rayside et al. (*Onward! 2009*)
- **Falling Back on Executable Specifications**, H. Samimi et al. (*ECOOP 2010*)
- **Unified Execution of Imperative and Declarative Code**, A. Milicevic et al. (*ICSE 2011*)

Specification Languages

- **JFSL: JForge Specification Language**, K. Yessenov, MIT 2009
- **Software Abstractions: Logic, Language, and Analysis**, D. Jackson, MIT Press 2006

Programming Languages with Constraint Programming:

- **Jeeves: Programming with Delegation**, J. Yang, MIT, 2010
- **Programming with Quantifiers**, J.P. Near, MIT, 2010

Summary

SQUANDER **framework**

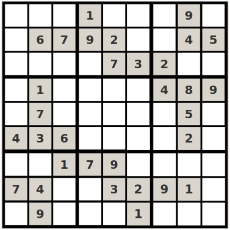
- unified execution of imperative and declarative code
- executable first-order, relational specifications for Java programs
- support for library classes and data abstractions
- ease of writing and solving constraint problems

http://people.csail.mit.edu/aleks/squander

Next steps

- provide better support for debugging
  - when no solution can be found, explain why (with the help of unsat core)
- syntesize code from specifications
  - especially for methods that only traverse the heap
- combine different solvers in the back end
  - an SMT solver would be better at handling large integers

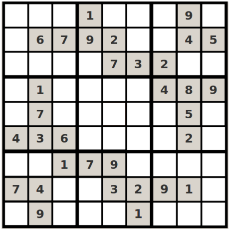
Mixing Imperative and Declarative with SQUANDER



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

Mixing Imperative and Declarative with SQUANDER

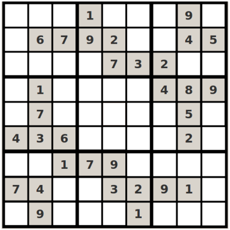
```
static class Cell { int num = 0; } // 0 means empty

@Invariant("all v: int - 0 | lone {c: this.cells.vals | c.num = v}")
static class CellGroup {
  Cell[] cells;
  public CellGroup(int n) { this.cells = new Cell[n]; }
}
```



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

Mixing Imperative and Declarative with SQUANDER

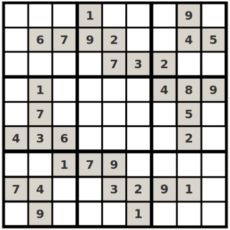
```
static class Cell { int num = 0; } // 0 means empty

@Invariant("all v: int - 0 | lone {c: this.cells.vals | c.num = v}")
static class CellGroup {
  Cell[] cells;
  public CellGroup(int n) { this.cells = new Cell[n]; }
}

public class Sudoku {
  int n;
  CellGroup[] rows, cols, grids;

  public Sudoku(int n) {
      // (1) create CellGroup and Cell objects,
      // (2) establish sharing of Cells between CellGroups
      init(n);
  }
```



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

Mixing Imperative and Declarative with SQUANDER

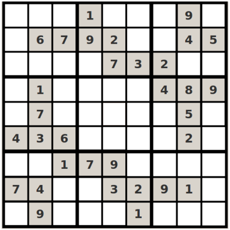
```
static class Cell { int num = 0; } // 0 means empty

@Invariant("all v: int - 0 | lone {c: this.cells.vals | c.num = v}")
static class CellGroup {
  Cell[] cells;
  public CellGroup(int n) { this.cells = new Cell[n]; }
}

public class Sudoku {
  int n;
  CellGroup[] rows, cols, grids;

  public Sudoku(int n) {
      // (1) create CellGroup and Cell objects,
      // (2) establish sharing of Cells between CellGroups
      init(n);
  }

  @Ensures("all c:Cell | c.num > 0 && c.num <= this.n")
  @Modifies("Cell.num | _<1> = 0")
  public void solve() { Squander.exe(this); }
```



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

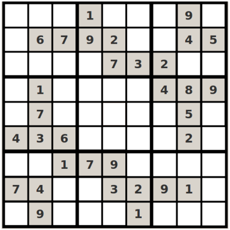
Mixing Imperative and Declarative with SQUANDER

```
static class Cell { int num = 0; } // 0 means empty

@Invariant(”all v: int - 0 | lone {c: this.cells.vals | c.num = v}”)
static class CellGroup {
  Cell[] cells;
  public CellGroup(int n) { this.cells = new Cell[n]; }
}

public class Sudoku {
  int n;
  CellGroup[] rows, cols, grids;

  public Sudoku(int n) {
      // (1) create CellGroup and Cell objects,
      // (2) establish sharing of Cells between CellGroups
      init(n);
  }

  @Ensures(”all c:Cell | c.num > 0 && c.num <= this.n”)
  @Modifies(”Cell.num | _<1> = 0”)
  public void solve() { Squander.exe(this); }

  public static void main(String[] args) {
    Sudoku s = new Sudoku();
    s.rows[0][3].num = 1; s.rows[0][7].num = 9;
    ...
    s.rows[8][1].num = 9; s.rows[8][5].num = 1;
    s.solve();
    System.out.println(s);
  }
}
```



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

Mixing Imperative and Declarative with SQUANDER



			1				9	
	6	7	9	2			4	5
				7	3	2		
	1					4	8	9
	7						5	
4	3	6					2	
		1	7	9				
7	4			3	2	9	1	
	9				1			

```java
static class Cell { int num = 0; } // 0 means empty

@Invariant("all v: int - 0 | lone {c: this.cells.vals | c.num = v}")
static class CellGroup {
  Cell[] cells;
  public CellGroup(int n) { this.cells = new Cell[n]; }
}

public class Sudoku {
  int n;
  CellGroup[] rows, cols, grids;

  public Sudoku(int n) {
      // (1) create CellGroup and Cell objects,
      // (2) establish sharing of Cells between CellGroups
      init(n);
  }

  @Ensures("all c:Cell | c.num > 0 && c.num <= this.n")
  @Modifies("Cell.num | _<1> = 0")
  public void solve() { Squander.exe(this); }

  public static void main(String[] args) {
    Sudoku s = new Sudoku();
    s.rows[0][3].num = 1; s.rows[0][7].num = 9;
    ...
    s.rows[8][1].num = 9; s.rows[8][5].num = 1;
    s.solve();
    System.out.println(s);
  }
}
```

Write more imperative code to make constraints simpler

Minimizing the Universe Size

Relations in Kodkod

$$\underset{\text{a relation of arity } k}{r_k} \overset{\text{in Kodkod}}{\rightsquigarrow} \underset{\text{a matrix of dim } |univ|^k}{M_{|univ| \times |univ| \times \cdots \times |univ|}}$$

Minimizing the Universe Size

Relations in Kodkod

$$\underset{\text{a relation of arity } k}{r_k} \overset{\text{in Kodkod}}{\rightsquigarrow} \underset{\text{a matrix of dim } |univ|^k}{M_{|univ| \times |univ| \times \cdots \times |univ|}}$$

SO

if $|univ| > 1291 \land (\exists_{r_k} \mid k \geq 3)$

Minimizing the Universe Size

Relations in Kodkod

$$\underset{\text{a relation of arity } k}{r_k} \overset{\text{in Kodkod}}{\rightsquigarrow} \underset{\text{a matrix of dim } |univ|^k}{M_{|univ| \times |univ| \times \cdots \times |univ|}}$$

SO

if $|univ| > 1291 \land (\exists_{r_k} \mid k \geq 3)$

$\implies dim(M) > 1291^3 = 2151685171 > \text{Integer.MAX\_VALUE}$

Minimizing the Universe Size

Relations in Kodkod

$$\underset{\text{a relation of arity } k}{r_k} \xrightarrow{\text{in Kodkod}} \underset{\text{a matrix of dim } |univ|^k}{M_{|univ| \times |univ| \times \cdots \times |univ|}}$$

SO

if $|univ| > 1291 \wedge (\exists_{r_k} \mid k \geq 3)$

$\implies dim(M) > 1291^3 = 2151685171 > \text{Integer.MAX\_VALUE}$

$\implies$ can't be represented in Kodkod

Minimizing the Universe Size

Relations in Kodkod

$$\begin{tikzcd} r_k \arrow[r, rightsquigarrow, "\text{in Kodkod}"] \arrow[d, dash] & M_{|univ| \times |univ| \times \cdots \times |univ|} \arrow[d, dash] \\ \text{a relation of arity } k & \text{a matrix of dim } |univ|^k \end{tikzcd}$$

so

if $|univ| > 1291 \wedge (\exists_{r_k} \mid k \geq 3)$

$\implies dim(M) > 1291^3 = 2151685171 >$ Integer.MAX_VALUE

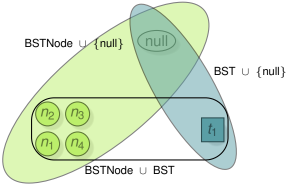
$\implies$ can't be represented in Kodkod

- ternary relations are not uncommon in SQUANDER (e.g. arrays)
- *MIT course scheduler* case study: almost 2000 objects
- **solution**:
  - partitioning algorithm that allows atoms to be shared

Partitioning Algorithm – Discussion

Why is this algorithm sufficient?

- what if we had partitions like this:

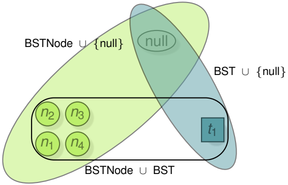


- 5 atoms would not be enough!
- the algorithm would have to discover strongly connected components
- **but**, SQUANDER type checker disallows types like BSTNode ∪ BST

Partitioning Algorithm – Discussion

Why is this algorithm sufficient?

- what if we had partitions like this:



- or a spec like:

"no BSTNode & int"

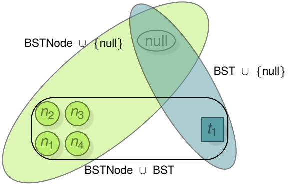
- 5 atoms would not be enough!
- the algorithm would have to discover strongly connected components
- **but**, SQUANDER type checker disallows types like BSTNode ∪ BST

- if nodes and ints shared atoms, then the intersection would not be empty!
- **again**, in Java, such expressions don't make much sense, so SQUANDER disallows them.

Partitioning Algorithm – Discussion

Why is this algorithm sufficient?

- what if we had partitions like this:



- 5 atoms would not be enough!
- the algorithm would have to discover strongly connected components
- **but**, SQUANDER type checker disallows types like BSTNode ∪ BST

- or a spec like:

"no BSTNode & int"

- if nodes and ints shared atoms, then the intersection would not be empty!
- **again**, in Java, such expressions don't make much sense, so SQUANDER disallows them.

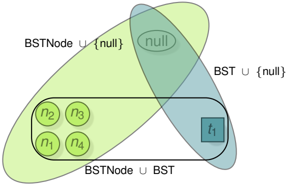
Limitations

- no performance gain

Partitioning Algorithm – Discussion

Why is this algorithm sufficient?

- what if we had partitions like this:



- 5 atoms would not be enough!
- the algorithm would have to discover strongly connected components
- **but**, SQUANDER type checker disallows types like BSTNode ∪ BST

- or a spec like:

"no BSTNode & int"

- if nodes and ints shared atoms, then the intersection would not be empty!
- **again**, in Java, such expressions don't make much sense, so SQUANDER disallows them.

Limitations

- no performance gain
- if a field of type Object is used, this algorithm has no effect
  - everything is a subtype of Object so everything has to go to the same partition