

kodkod for alloy users

emina torlak and greg dennis

everybody loves alloy



everybody loves alloy, but ...



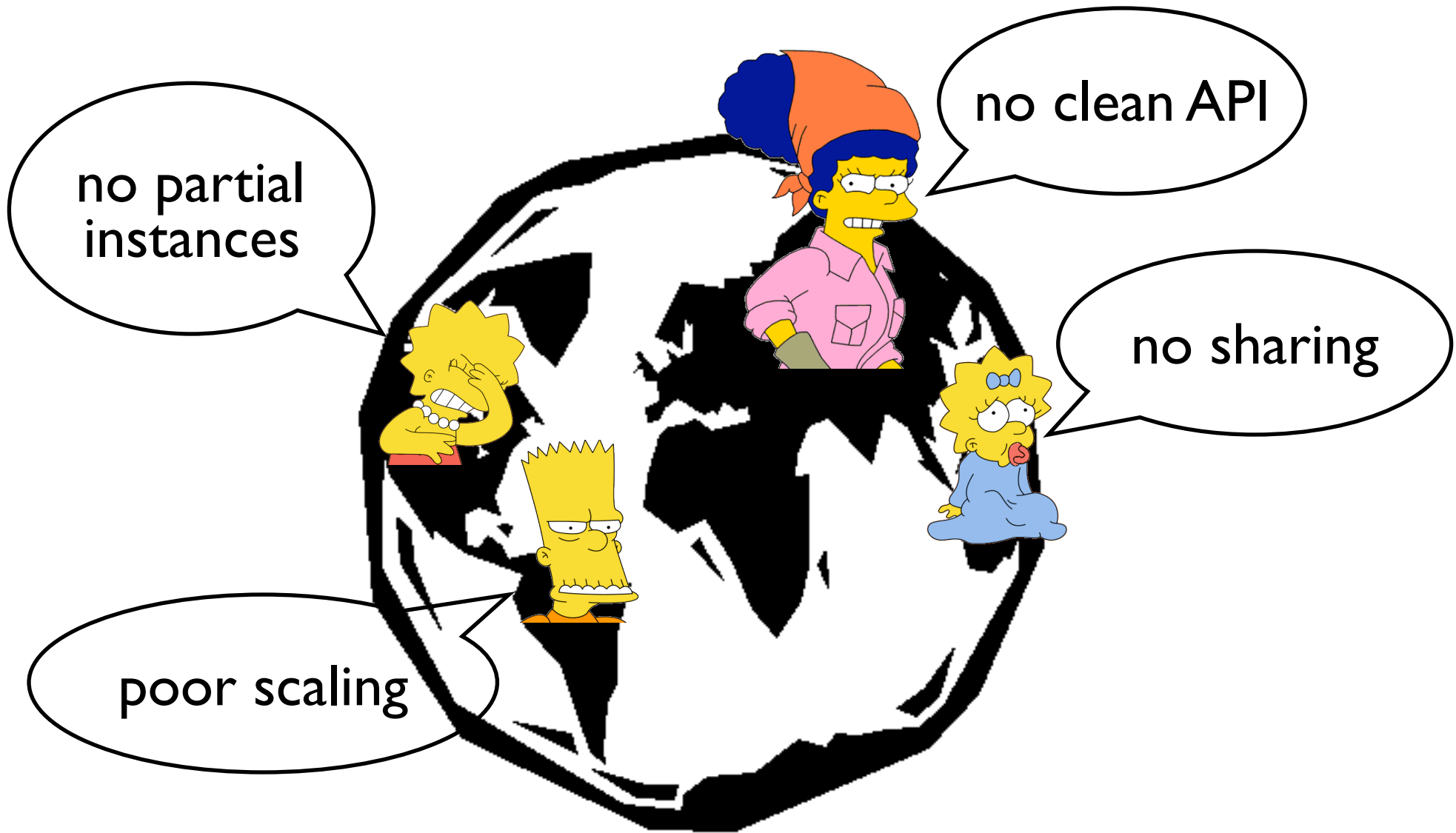
everybody loves alloy, but ...



everybody loves alloy, but ...

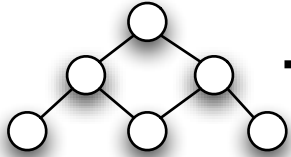


everybody loves alloy, but ...



kodkod

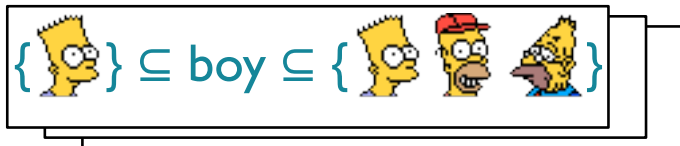
kodkod.ast.Formula



kodkod.engine.Solver



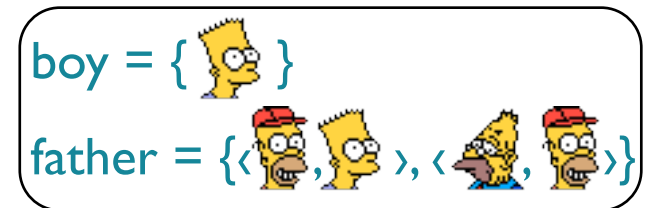
kodkod.instance.Bounds



kodkod.engine.Options







kodkod.engine.Solution







kodkod's spots

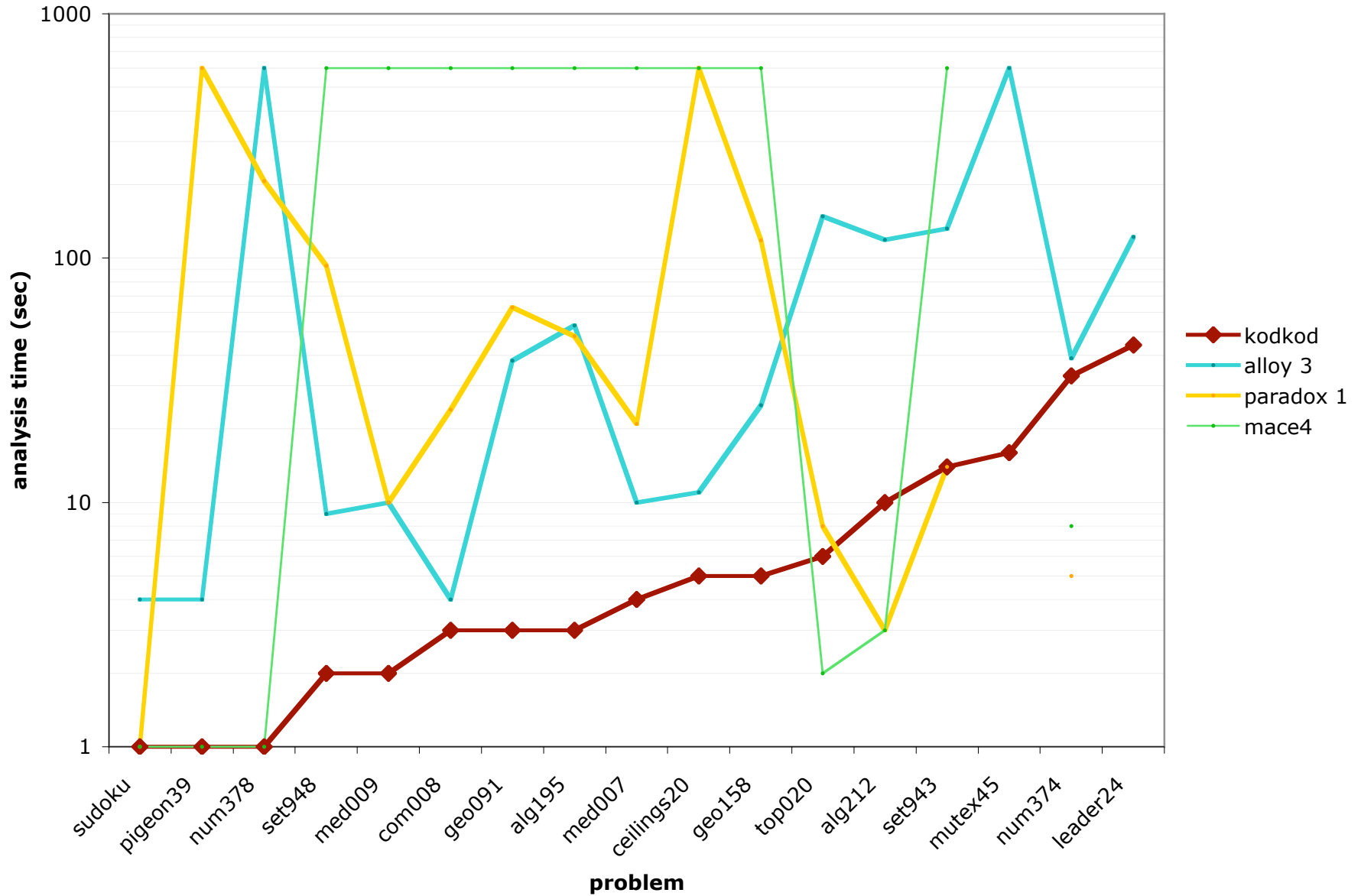
technical

-  new symmetry detection scheme
-  new sparse-matrix translation to boolean
-  new sharing detection mechanism
-  polarity-aware conversion from sat to cnf







usability

-  12K sloc (c.f. 62K sloc for alloy 3 engine)
-  can run purely in java
-  or with a c/c++ sat solver via jni
-  or with any sat solver using sat2004 formats

kodkod pounces



kodkod's friends (so far)

-  alloy 4 (chang, mit)
-  code analysis (dennis & taghdiri, mit)
-  course scheduling (yeung, mit):
<http://optima.csail.mit.edu:8080/scheduler/>
-  access-control policy analysis (fisler, wpi)
-  domain-specific language development and transformation (rouquette, jpl)
-  vaziri, ibm

example: sudoku in alloy

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

```
module sudoku
abstract sig Number { data: Number->Number }
abstract sig Region1, Region2, Region3 extends Number {}
```

```
one sig N1, N2, N3 extends Region1 {}
one sig N4, N5, N6 extends Region2 {}
one sig N7, N8, N9 extends Region3 {}
```

```
pred complete(rows, cols: set Number) {
  Number in cols.(rows.data) }
```

```
pred rules() {
  all x, y: Number | lone y.(x.data)
  all row: Number | complete(row, Number)
  all col: Number | complete(Number, col)
  complete(Region1, Region1)
  ...
  complete(Region3, Region3)
}
```

```
pred puzzle() {
  N1->N1->N1 + ... + N9->N9->N4 in data
}
```

```
pred game() { rules() and puzzle() }
```

```
run game
```

example: sudoku in kodkod

```
module sudoku
abstract sig Number { data: Number->Number }
abstract sig RegionI ... extends Number {}
```

```
pred complete(rows, cols: set Number) {...}
```

```
pred rules() {...}
```

```
pred puzzle() {...}
```

```
run game
```

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

  public Formula complete(Expression rows, Expression cols) {...}

  public Formula rules() {...}

  public Bounds puzzle() {...}

  public static void main(String[] args) {...}
}
```

example: sudoku in kodkod

```
public final class Sudoku {  
    private final Relation Number = Relation.unary("Number");  
    private final Relation data = Relation.ternary("data");  
    private final Relation[] regions = new Relation[] { ... };  
  
    public Formula complete(Expression rows, Expression cols) {  
        return Number.in(cols.join(rows.join(data)));  
    }  
  
    public Formula rules() { ... }  
  
    public Bounds puzzle() { ... }  
  
    public static void main(String[] args) {...}  
}
```

```
pred complete(rows, cols: set Number) {  
    Number in cols.(rows.data) }
```



example: sudoku in kodkod

```
public final class Sudoku {  
    private final Relation Number = Relation.unary("Number");  
    private final Relation data = Relation.ternary("data");  
    private final Relation[] regions = new Relation[] {...};  
  
    public Formula complete(Expression rows, Expression cols) {...}
```

```
pred rules() {  
    all x, y: Number | lone y.(x.data)  
    all row: Number | complete(row, Number)  
    all col: Number | complete(Number, col)  
    complete(Region1, Region1)  
    ...  
    complete(Region3, Region3)  
}
```

```
public Formula rules() {  
    final Variable x = Variable.unary("x");  
    final Variable y = Variable.unary("y");  
    final Formula f1 = y.join(x.join(data)).lone().  
        forAll(x.oneOf(Number).and(y.oneOf(Number)));  
  
    final Variable row = Variable.unary("row");  
    final Formula f2 = complete(row, Number).  
        forAll(row.oneOf(Number));  
  
    final Variable col = Variable.unary("col");  
    final Formula f3 = complete(Number, col).  
        forAll(col.oneOf(Number));  
  
    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() {...}  
  
public static void main(String[] args) {...}  
}
```

example: sudoku in kodkod

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

    public Formula complete(Expression rows, Expression cols) {...}

    public Formula rules() { ... }

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

        final Universe u = new Universe(atoms);
        final Bounds b = new Bounds(u);
        final 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));

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

    public static void main(String[] args) {...}
}
```

```
abstract sig Number { ... }
abstract sig Region1, Region2, Region3
  extends Number {}

one sig N1, N2, N3 extends Region1 {}
one sig N4, N5, N6 extends Region2 {}
one sig N7, N8, N9 extends Region3 {}
```

example: sudoku in kodkod

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

    public Formula complete(Expression rows, Expression cols) {...}

    public Formula rules() { ... }

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

        final Universe u = new Universe(atoms);
        final Bounds b = new Bounds(u);
        final 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));

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

        return b;
    }

    public static void main(String[] args) {...}
}
```

```
abstract sig Number { data: Number->Number }
pred puzzle() {
    N1->N1->N1 + ... + N9->N9->N4 in data
}
```



example: sudoku in kodkod





```
public final class Sudoku {  
    private final Relation Number = Relation.unary("Number");  
    private final Relation data = Relation.ternary("data");  
    private final Relation[] regions = new Relation[] {...};  
  
    public Formula complete(Expression rows, Expression cols) {...}  
  
    public Formula rules() {...}  
  
    public Bounds puzzle() {...}
```

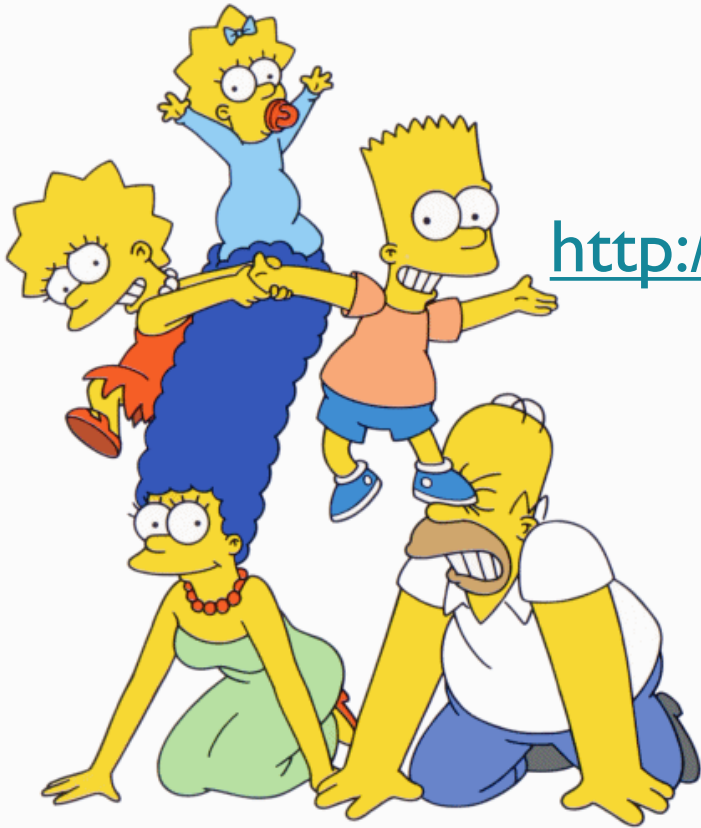
```
pred game() { rules() and puzzle() }  
run game
```

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




```
}
```

future work

-  solution enumeration
-  native support for sequences
-  incremental solving
-  richer support for unsat core extraction



<http://web.mit.edu/emina/www/kodkod.html>

-  kodkod.jar
-  javadocs
-  examples