EECS 219C: Formal Methods

Satisfiability Modulo Theories

Examples Used in Lecture

Sanjit A. Seshia

EECS, UC Berkeley

Equivalence Checking of Program Fragments

```
int fun1(int y) {
    int x, z;
    z = y;
    y = x;
    x = z;

   return x*x;
}

int fun2(int y) {
     return y*y;
}
```

SMT formula ϕ
Satisfiable iff programs non-equivalent

$$(z = y \wedge y1 = x \wedge x1 = z \wedge ret1 = x1*x1)$$
$$\wedge$$
$$(ret2 = y*y)$$
$$\wedge$$
$$(ret1 \neq ret2)$$

What if we use SAT to check equivalence?

Equivalence Checking of Program Fragments

```
int fun1(int y) {
    int x, z;
    z = y;
    y = x;
    x = z;

   return x*x;
}

int fun2(int y) {
     return y*y;
}
```

SMT formula ϕ

Satisfiable iff programs non-equivalent

$$(z = y \wedge y1 = x \wedge x1 = z \wedge ret1 = x1*x1)$$
$$\wedge$$
$$(ret2 = y*y)$$
$$\wedge$$
$$(ret1 \neq ret2)$$

Using SAT to check equivalence (w/ Minisat)
- 32 bits for y: Did not finish in over 5 hours
- 16 bits for y: 37 sec.
- 8 bits for y: 0.5 sec.

Equivalence Checking of Program Fragments

```
int fun1(int y) {
    int x, z;
    z = y;
    y = x;
    x = z;

   return x*x;
}

int fun2(int y) {
     return y*y;
}
```

SMT formula ϕ'

$$(z = y \wedge y1 = x \wedge x1 = z \wedge ret1 = sq(x1))$$
$$\wedge$$
$$(ret2 = sq(y))$$
$$\wedge$$
$$(ret1 \neq ret2)$$

Using EUF solver: 0.01 sec

Equivalence Checking of Program Fragments

```
int fun1(int y) {
    int x;
    x = x ^ y;
    y = x ^ y;
    x = x ^ y;

  return x*x;
}

int fun2(int y) {
     return y*y;
}
```

Does EUF still work?

No!
Must reason about bit-wise XOR.

Need a solver for bit-vector arithmetic.

Solvable in less than a sec. with a current bit-vector solver.

Equivalence Checking of Program Fragments

```
int fun1(int y) {
    int x[2];
    x[0] = y;
    y = x[1];
    x[1] = x[0];

   return x[1]*x[1];
}
```

```
int fun2(int y) {
      return y*y;
}
```

How can we express the equivalence checking problem as an SMT formula with arrays?

Equivalence Checking of Program Fragments

```
int fun1(int y) {
    int x[2];
    x[0] = y;
    y = x[1];
    x[1] = x[0];

   return x[1]*x[1];
}
```

```
int fun2(int y) {
      return y*y;
}
```

SMT formula ϕ''

$$[\quad x1 = store(x,0,y) \wedge y1 = select(x1,1)$$
$$\wedge\; x2 = store(x1,1,select(x1,0))$$
$$\wedge\; ret1 = sq(select(x2,1)) \quad]$$
$$\wedge$$
$$(\; ret2 = sq(y)\;)$$
$$\wedge$$
$$(\; ret1 \neq ret2\;)$$

EUF

- Example:

$$
\begin{aligned}
& g(g(g(x))) = x \\
\wedge\ & g(g(g(g(g(x))))) = x \\
\wedge\ & g(x) \neq x
\end{aligned}
$$

Difference Logic

$x_1 \geq x_2$

$x_3 \leq 0$

$x_2 + 3 \geq x_1$

$x_1 + 1 \leq x_3$

$x_2 + 1 \geq 0$

$x_4 + 2 \geq 0$

$x_4 \leq x_2 - 2$

Theory of Arrays

- Two main axioms: For all A, i, j, d
  - select(store(A,i,d), i) = d
  - select(store(A,i,d), j) = select(A,j), if i $\neq$ j
- Decision procedure operates by performing case-splits
- Example:

```
int a[10];
int fun3(int i) {
   int j;
   for(j=0; j<10; j++) a[j] = j;
   assert(a[i] <= 5);
}
```

Theory of Arrays

- Two main axioms: For all A, i, j, d
  - select(store(A,i,d), i) = d
  - select(store(A,i,d), j) = select(A,j), if $i \neq j$
- Decision procedure operates by performing case-splits
- Example:

$a[0] = 0 \wedge a[1] = 1 \wedge a[2] = 2 \wedge \ldots a[9] = 9 \wedge a[i] > 5$