Equivalence Checking of Program Fragments

```c
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 $\phi$

Satisfiable iff programs non-equivalent

\[
(z = y \land y1 = x \land x1 = z \land ret1 = x1 \times x1) \Rightarrow a
\]

\[
(ret2 = y \times y) \Rightarrow a
\]

\[
(ret1 \neq ret2)
\]

What if we use SAT to check equivalence?
Equivalence Checking
of Program Fragments

```c
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 $\phi$:
Satisfiable iff programs non-equivalent

$( z = y \land y_1 = x \land x_1 = z \land a_1 \land \text{ret}_1 = x_1^2 )$

$\land$

$( a \land \text{ret}_2 = y^2 )$

$\land$

$( a \land \text{ret}_1 \neq \text{ret}_2 )$

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

```c
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 $\phi'$:

$( z = y \land y_1 = x \land x_1 = z \land a_1 \land \text{ret}_1 = \text{sq}(x_1) )$

$\land$

$( a \land \text{ret}_2 = \text{sq}(y) )$

$\land$

$( a \land \text{ret}_1 \neq \text{ret}_2 )$

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

```c
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 \( \phi'' \)

\[
\begin{align*}
& x_1 = \text{store}(x,0,y) \ a \ y_1 = \text{select}(x_1,1) \\
& a \ x_2 = \text{store}(x_1,1,\text{select}(x_1,0)) \\
& a \ \text{ret1} = \text{sq}(\text{select}(x_2,1)) \\
& a \ ( \text{ret2} = \text{sq}(y) ) \\
& a \ ( \text{ret1} \neq \text{ret2} )
\end{align*}
\]

EUF

- Example:
  \[
  g(g(g(x))) = x
  \]
  \[
  a \ g(g(g(g(x)))) = x
  \]
  \[
  a \ g(x) \neq x
  \]
Difference Logic

\[ x_1 \geq x_2 \]
\[ x_3 \geq 0 \]
\[ x_2 + 3 \geq x_1 \]
\[ x_1 + 1 \geq x_3 \]
\[ x_2 + 1 \geq 0 \]
\[ x_4 + 2 \geq 0 \]
\[ x_4 \geq x_2 - 2 \]

Theory of Arrays

- Two main axioms: For all A, i, j, d
  - \( \text{select}(\text{store}(A,i,d), i) = d \)
  - \( \text{select}(\text{store}(A,i,d), j) = \text{select}(A,j) \), 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: