Optimizing Constraint Solving to Better Support Symbolic Execution

Ikpeme Ere te and Alessandro Orso
School of Computer Science – College of Computing
Georgia Institute of Technology

Partially supported by: NSF, IBM, and MSR
Background: Dynamic Symbolic Execution

01. foo(int a, int b, int c, int d) {
  02.  if (c > a)
  03.    int e=d+10
  04.    if (b > 5)
  05.      // do something
  06.    else if (a < e)
  07.      if (b < c)
  08.        // do something
  09.      else
 10.    else
 11.      // do something
 12.    else
 13.    return
 14. }

Inputs: a=4, b= 5, c=6, d=1

Executed branches:

Symbolic state:

Path condition (PC):

DSE:
Background: Dynamic Symbolic Execution

```c
01. foo(int a, int b, int c, int d) {
02.    if (c > a)
03.        int e=d+10
04.    else if (b > 5)
05.        // do something
06.        else if (a < e)
07.            if (b < c)
08.                // do something
09.            else
10.                // do something
11.            else
12.                // do something
13.        return
14.    }
```

Inputs: a=4, b= 5, c=6, d=1

Executed branches:

Symbolic state:
- $a=a_0$, $b=b_0$, $c=c_0$, $d=d_0$

Path condition (PC):

DSE:
Background: Dynamic Symbolic Execution

```c
01. foo(int a, int b, int c, int d) {
02.    if (c > a)  // Highlighted branch
03.       int e = d + 10
04.    if (b > 5)
05.       // do something
06.    else if (a < e)
07.       if (b < c)
08.       // do something
09.    else
10.       // do something
11.    else
12.       // do something
13.    return
14. }
```

**Inputs:** a=4, b=5, c=6, d=1

**Executed branches:**

**Symbolic state:**

a=a₀, b=b₀, c=c₀, d=d₀

**Path condition (PC):**

**DSE:**
Background: Dynamic Symbolic Execution

```c
01. foo(int a, int b, int c, int d) {
02.   if (c > a)  
03.     int e=d+10
04.     if (b > 5)  
05.       // do something
06.     else if (a < e)  
07.       if (b < c)  
08.         // do something
09.     else
10.       // do something
11.     else
12.       // do something
13.   return
14. }
```

**Inputs:** a=4, b=5, c=6, d=1

**Executed branches:** 2T,

**Symbolic state:**

- a=a₀, b=b₀, c=c₀, d=d₀

**Path condition (PC):**

**DSE:**
Background: Dynamic Symbolic Execution

01. foo(int a, int b, int c, int d) {
02.   if (c > a)  
03.     int e = d + 10  
04.     if (b > 5)  
05.       // do something  
06.     else if (a < e)  
07.       if (b < c)  
08.         // do something  
09.       else  
10.         // do something  
11.     else  
12.         // do something  
13.     return  
14. }

**Inputs:** a=4, b=5, c=6, d=1

**Executed branches:** 2T,

**Symbolic state:**
a=a₀, b=b₀, c=c₀, d=d₀

**Path condition (PC):**
(c₀ > a₀)

**DSE:**
Background: Dynamic Symbolic Execution

```c
01. foo(int a, int b, int c, int d) {
    02.  if (c > a)
    03.      int e=d+10
    04.    if (b > 5)
    05.      // do something
    06.    else if (a < e)
    07.      if (b < c)
    08.        // do something
    09.      else
    10.        // do something
    11.    else
    12.        // do something
    13.    return
    14. }
```

**Inputs:** a=4, b=5, c=6, d=1

**Executed branches:** 2T,

**Symbolic state:**
- a=a₀, b=b₀, c=c₀, d=d₀

**Path condition (PC):**
- (c₀ > a₀)

**DSE:**
foo(int a, int b, int c, int d) {
    if (c > a)
        int e = d + 10
    if (b > 5)
        // do something
    else if (a < e)
        // do something
    else
        // do something
    return
}

Inputs: a=4, b=5, c=6, d=1

Executed branches: 2T,

Symbolic state:
a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10

Path condition (PC):
(c_0 > a_0)

DSE:
Background: Dynamic Symbolic Execution

```cpp
01. foo(int a, int b, int c, int d) {
02.  if (c > a)
03.    int e=d+10
04.  if (b > 5)
05.    // do something
06.  else if (a < e)
07.    if (b < c)
08.      // do something
09.    else
10.    // do something
11.  else
12.    // do something
13.  return
14. }
```

**Inputs:** a=4, b= 5, c=6, d=1

**Executed branches:** 2T,

**Symbolic state:**
- a=a₀, b=b₀, c=c₀, d=d₀, e=d₀+10

**Path condition (PC):**
- (c₀ > a₀)

**DSE:**
Background: Dynamic Symbolic Execution

01. foo(int a, int b, int c, int d) {
02.   if (c > a)
03.     int e=d+10
04.     if (b > 5)
05.       // do something
06.   else if (a < e)
07.     if (b < c)
08.       // do something
09.   else
10.     // do something
11.   else
12.     // do something
13.   return
14. }

**Inputs:** a=4, b= 5, c=6, d=1

**Executed branches:** 2T, 4F,

**Symbolic state:**
  a=a₀, b=b₀, c=c₀, d=d₀, e=d₀+10

**Path condition (PC):**
  (c₀ > a₀)

**DSE:**
Background: Dynamic Symbolic Execution

01. foo(int a, int b, int c, int d) {
02.   if (c > a)
03.     int e=d+10
04.     if (b > 5)
05.       // do something
06.   else if (a < e)
07.     if (b < c)
08.       // do something
09.     else
10.     // do something
11.   else
12.     // do something
13. return
14. }

Inputs: a=4, b=5, c=6, d=1

Executed branches: 2T, 4F,

Symbolic state:
  a=a₀, b=b₀, c=c₀, d=d₀, e=d₀+10

Path condition (PC):
  (c₀ > a₀) ∧ (b₀ <= 5)

DSE:
Background: Dynamic Symbolic Execution

01. foo(int a, int b, int c, int d) {
02.   if (c > a)
03.     int e = d + 10
04.     if (b > 5)
05.       // do something
06.     else if (a < e)
07.       if (b < c)
08.         // do something
09.       else
10.       // do something
11.     else
12.     // do something
13.   return
14. }

Inputs: a=4, b=5, c=6, d=1

Executed branches: 2T, 4F,

Symbolic state:
   a=a₀, b=b₀, c=c₀, d=d₀, e=d₀+10

Path condition (PC):
   (c₀ > a₀) ∧ (b₀ <= 5)

DSE:
Background: Dynamic Symbolic Execution

01. foo(int a, int b, int c, int d) {
02. if (c > a)
03.   int e = d + 10
04.   if (b > 5)
05.     // do something
06. else if (a < e)
07.   if (b < c)
08.     // do something
09. else
10.   // do something
11. else
12.   // do something
13. return
14. }

Inputs: a=4, b=5, c=6, d=1

Executed branches: 2T, 4F, 6T,

Symbolic state:
  a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10

Path condition (PC):
  (c_0 > a_0) \land (b_0 <= 5)

DSE:
Background: Dynamic Symbolic Execution

01. foo(int a, int b, int c, int d) {
02.  if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.      if (b < c)
08.        // do something
09.      else
10.        // do something
11.    else
12.      // do something
13.  return
14. }

**Inputs:** a=4, b= 5, c=6, d=1

**Executed branches:** 2T, 4F, 6T,

**Symbolic state:**
\[ a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10 \]

**Path condition (PC):**
\[ (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \]

**DSE:**
Background: Dynamic Symbolic Execution

Inputs: a=4, b=5, c=6, d=1

Executed branches: 2T, 4F, 6T,

Symbolic state:
\[ a=a_0, \ b=b_0, \ c=c_0, \ d=d_0, \ e=d_0+10 \]

Path condition (PC):
\[ (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \]

DSE:
Background: Dynamic Symbolic Execution

```
01. foo(int a, int b, int c, int d) {
02.   if (c > a)
03.     int e=d+10
04.   else if (b > 5)
05.     // do something
06.   else if (a < e)
07.     if (b < c)
08.     // do something
09.   else
10.     // do something
11.   else
12.     // do something
13.   return
14. }
```

**Inputs:** a=4, b=5, c=6, d=1

**Executed branches:** 2T, 4F, 6T, 7T

**Symbolic state:**
\[
a = a_0, \ b = b_0, \ c = c_0, \ d = d_0, \ e = d_0 + 10
\]

**Path condition (PC):**
\[(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10)\]

**DSE:**
Background: Dynamic Symbolic Execution

```c
01. foo(int a, int b, int c, int d) {
02.  if (c > a)
03.    int e=d+10
04.    if (b > 5)
05.      // do something
06.    else if (a < e)
07.    else
08.      // do something
09.  return
10. }
```

**Inputs:** a=4, b=5, c=6, d=1

**Executed branches:** 2T, 4F, 6T, 7T

**Symbolic state:**
```
a=a_0, b=b_0, c=c_0, d=d_0, e=d_0+10
```

**Path condition (PC):**
```
(c_0 > a_0) \land (b_0 <= 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0)
```

**DSE:**
Background: Dynamic Symbolic Execution

01. `foo(int a, int b, int c, int d) {`
02. if (c > a)
03.   int e = d + 10
04. if (b > 5)
05.   // do something
06. else if (a < e)
07.   if (b < c)
08.     // do something
09.     else
10.   else
11.     // do something
12.   else
13.   // do something
14. return
15. }`

**Inputs:** a=4, b=5, c=6, d=1

**Executed branches:** 2T, 4F, 6T, 7T

**Symbolic state:**
\[ a = a_0, b = b_0, c = c_0, d = d_0, e = d_0 + 10 \]

**Path condition (PC):**
\[ (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \]

**DSE:**
Background: Dynamic Symbolic Execution

```c
01. foo(int a, int b, int c, int d) {
    02. if (c > a)
    03.    int e=d+10
    04.    if (b > 5)
    05.       // do something
    06.    else if (a < e)
    07.       if (b < c)
    08.          // do something
    09.       else
    10.       // do something
    11.    else
    12.    // do something
    13.    return
    14. }
```

**Inputs:** a=4, b= 5, c=6, d=1

**Executed branches:** 2T, 4F, 6T, 7T

**Symbolic state:**
- a=a₀, b=b₀, c=c₀, d=d₀, e=d₀+10

**Path condition (PC):**
- \((c_0 > a_0) \land (b_0 <= 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0)\)

**DSE:**
Background: Dynamic Symbolic Execution

01. foo(int a, int b, int c, int d) {  
02.   if (c > a)  
03.     int e=d+10  
04.   if (b > 5)  
05.     // do something  
06.   else if (a < e)  
07.     if (b < c)  
08.       // do something  
09.     else  
10.       // do something  
11.   else  
12.     // do something  
13.   return  
14. }

Inputs: a=4, b=5, c=6, d=1

Executed branches: 2T, 4F, 6T, 7T

Symbolic state:  
   a=a₀, b=b₀, c=c₀, d=d₀, e=d₀+10

Path condition (PC):  
   (c₀ > a₀) ∧ (b₀ <= 5) ∧ (a₀ < d₀ + 10) ∧ (b₀ < c₀)

DSE:  
   (c₀ > a₀) ∧ (b₀ <= 5) ∧ (a₀ < d₀ + 10)
Background: Dynamic Symbolic Execution

01. foo(int a, int b, int c, int d) {
02.   if (c > a)
03.     int e=d+10
04.     if (b > 5)
05.       // do something
06.     else if (a < e)
07.       if (b < c)
08.         // do something
09.       else
10.       // do something
11.     else
12.     // do something
13.   return
14. }

Inputs: a=4, b= 5, c=6, d=1

Executed branches: 2T, 4F, 6T, 7T

Symbolic state:
   a=a₀, b=b₀, c=c₀, d=d₀, e=d₀+10

Path condition (PC):
   (c₀ > a₀) ∧ (b₀ <= 5) ∧ (a₀ < d₀ + 10) ∧ (b₀ < c₀)

DSE:
   (c₀ > a₀) ∧ (b₀ <= 5) ∧ (a₀ < d₀ + 10) ∧ (b₀ >= c₀)
Background: Dynamic Symbolic Execution

```c
01. foo(int a, int b, int c, int d) {
02.   if (c > a)
03.     int e=d+10
04.     if (b > 5)
05.       // do something
06.     else if (a < e)
07.       if (b < c)
08.         // do something
09.       else
10.       // do something
11.     else
12.     // do something
13.   return
14. }
```

**Inputs:** a=4, b=5, c=6, d=1

**Executed branches:** 2T, 4F, 6T, 7T

**Symbolic state:**
\[
\begin{align*}
a &= a_0, \\ b &= b_0, \\ c &= c_0, \\ d &= d_0, \\ e &= d_0 + 10
\end{align*}
\]

**Path condition (PC):**
\[(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0)\]

**DSE:**
\[
\begin{align*}
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \\
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 \geq d_0 + 10) \\
(c_0 > b_0) \land (b_0 > 5) \\
(c_0 \leq b_0)
\end{align*}
\]
Symbolic Execution
Symbolic Execution

Program
Symbolic Execution

Program

Symbolic executor
Symbolic Execution

Program  
Symbolic executor
Symbolic Execution
Symbolic Execution
Symbolic Execution

Program

Symbolic executor

PC

$\text{c}_1 \land \ldots \land \text{c}_n$
Symbolic Execution

Program

Symbolic executor

SMT solver

PC $c_1 \land \ldots \land c_n$
Symbolic Execution

Program

Symbolic executor

SMT solver

PC $c_1 \land \ldots \land c_n$

Sat/Unsat (solution)

PC $c_1 \land \ldots \land c_n$
Symbolic Execution and SMT Solving

Symbolic executor

SMT solver

PC: $c_1 \land \ldots \land c_n$

Sat/Unsat (solution)
Symbolic Execution and SMT Solving

Symbolic executor

SMT solver

PC
\( c_1 \land ... \land c_n \)

Sat/Unsat (solution)
Symbolic Execution and SMT Solving

Symbolic executor

PC
\( c_1 \land ... \land c_n \)

Sat/Unsat (solution)
What Are We Missing?

- **Context information** (e.g., existence of previous solutions for similar PCs)
- **Domain knowledge** (e.g., programs’ specific properties)
State of the Art

• Some techniques present initial solutions (domain-based constraint optimizations)

• But:

  • What is the effectiveness of these techniques?
  
  • What other techniques could be used?
  
  • Would symbolic execution actually benefit from these techniques?
Our Goal

• Initial investigation of these questions by

• proposing a novel constraint optimization technique for dynamic symbolic execution: DomainReduce

• performing an empirical evaluation to assess new and existing optimizations empirically
DomainReduce: Intuitive View

PC

\( c_1 \land \cdots \land c_n \)

Sat/Unsat (solution)

SMT solver
DomainReduce: Intuitive View

$\text{PC} = c_1 \land \ldots \land c_n$

Sat/Unsat (solution)

SMT solver
DomainReduce: Intuitive View

PC: $c_1 \land ... \land c_n$

Sat/Unsat (solution)

SMT solver
DomainReduce: Intuitive View

Restrict domain of constraints to be solved by leveraging solutions of similar PCs
DomainReduce: Intuitive View

Restrict domain of constraints to be solved by leveraging solutions of similar PCs

PC: $c_1 \land \ldots \land c_n$

SMT solver

Sat/Unsat (solution)
DomainReduce: Intuitive View

Restrict domain of constraints to be solved by leveraging solutions of similar PCs

PC: $c_1 \land \ldots \land c_n$

Sat/Unsat (solution)

SMT solver
DomainReduce: Intuitive View

Restrict domain of constraints to be solved by leveraging solutions of similar PCs.
DomainReduce: Intuitive View

Restrict domain of constraints to be solved by leveraging solutions of similar PCs
DomainReduce: Intuitive View

Restrict domain of constraints to be solved by leveraging solutions of similar PCs

PC
C1 ∧ ... ∧ Cn

Sat/Unsat (solution)

SMT solver

✔

✔
DomainReduce: Intuitive View

Restrict domain of constraints to be solved by leveraging solutions of similar PCs

Trade-off speed/likelihood of finding solutions
DomainReduce Example
(with dependencies)
DomainReduce Example
(with dependencies)

\((c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0)\)
DomainReduce Example
(with dependencies)

\[
\begin{cases}
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \\
\end{cases}
\]

\[
a_0 = 4, \ b_0 = 5, \ c_0 = 6, \ d_0 = 1
\]
DomainReduce Example
(with dependencies)

\[
\begin{cases}
(c_0 > a_0) \land (b_0 <= 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \\
a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \\
(c_0 > a_0) \land (b_0 <= 5) \land (a_0 < d_0 + 10) \land (b_0 >= c_0)
\end{cases}
\]
DomainReduce Example
(with dependencies)

\(\{(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0)\}\)

\(a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1\)

\(\{(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0)\}\)

With dependencies:
DomainReduce Example
(with dependencies)

\{(c_0 > a_0) \land (b_0 <= 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \}

a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1

(c_0 > a_0) \land (b_0 <= 5) \land (a_0 < d_0 + 10) \land (b_0 >= c_0)

With dependencies:
(c_0 > a_0) \land (b_0 <= 5) \land (a_0 < d_0 + 10) \land (b_0 >= c_0)
DomainReduce Example
(with dependencies)

\[ (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \]

\[ a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \]

\[ (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \]

With dependencies:
\[ (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \]
DomainReduce Example
(with dependencies)

\[
\{ (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \\
\}

a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1

(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0)

With dependencies:

\[
(5 > 4) \land (b_0 \leq 5) \land (4 < 1 + 10) \land (b_0 \geq 6)
\]
DomainReduce Example
(with dependencies)

\[
\{ (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \}
\]

\[
a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1
\]

\[
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0)
\]

With dependencies:

\[
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0)
\]

\[
5 > 4 \land (b_0 \leq 5) \land (4 < 1 + 10) \land (b_0 \geq 6)
\]
DomainReduce Example
(with dependencies)

\[
\begin{align*}
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \\
\{a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1\} \\
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0)
\end{align*}
\]

With dependencies:

\[
\begin{align*}
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \\
(5 > 4) \land (b_0 \leq 5) \land (4 < 1 + 10) \land (b_0 \geq c_0) \\
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0)
\end{align*}
\]
DomainReduce Example
(with dependencies)

\[ \begin{align*}
\{ (c_0 > a_0) & \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \\
& a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \\
& (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \\
\} \\
\text{With dependencies:} \\
& (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \\
& (5 > 4) \land (b_0 \leq 5) \land (4 < 1 + 10) \land (b_0 \geq 6) \\
& (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0)
\end{align*} \]
DomainReduce Example
(with dependencies)

\[
\left\{ (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \right. \\
\left. a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \right. \\
\left. (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \right. \\
\]

With dependencies:

\[
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \\
(5 > 4) \land (b_0 \leq 5) \land (4 < 1 + 10) \land (b_0 \geq 6) \\
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \\
(c_0 > a_0) \land (5 \leq 5) \land (a_0 < d_0 + 10) \land (5 \geq c_0) \\
\]

✘
DomainReduce Example
(with dependencies)

\[
\begin{align*}
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \\
\end{align*}
\]

\(a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1\)

\[
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0)
\]

With dependencies:

\[
\begin{align*}
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \\
(5 > 4) \land (b_0 \leq 5) \land (4 < 1 + 10) \land (b_0 \geq 6) \\
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \\
(c_0 > a_0) \land (5 \leq 5) \land (a_0 < d_0 + 10) \land (5 \geq c_0)
\end{align*}
\]
DomainReduce Example
(without dependencies)

\[
\begin{align*}
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \\
\end{align*}
\]

\[
a_0 = 4, \ b_0 = 5, \ c_0 = 6, \ d_0 = 1
\]

\[
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0)
\]

Without dependencies:

\[
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0)
\]

\[
(5 > 4) \land (b_0 \leq 5) \land (4 < 1 + 10) \land (b_0 \geq 6)
\]

\[
(c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0)
\]

✘
DomainReduce Example
(without dependencies)

\[
\begin{aligned}
\{ & (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \\
 & a_0 = 4, b_0 = 5, c_0 = 6, d_0 = 1 \\
& (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \}
\end{aligned}
\]

Without dependencies:
\[
\begin{aligned}
& (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \\
& (5 > 4) \land (b_0 \leq 5) \land (4 < 1 + 10) \land (b_0 \geq 6) \\
& (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \\
& (c_0 > 4) \land (5 \leq 5) \land (4 < 1 + 10) \land (5 \geq c_0)
\end{aligned}
\]
DomainReduce Example
(without dependencies)

\[
\begin{align*}
\{ & (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 < c_0) \\
a_0 &= 4, \quad b_0 = 5, \quad c_0 = 6, \quad d_0 = 1 \\
& (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \}
\end{align*}
\]

Without dependencies:

\[
\begin{align*}
& (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \\
& (5 > 4) \land (b_0 \leq 5) \land (4 < 1 + 10) \land (b_0 \geq 6) \\
& (c_0 > a_0) \land (b_0 \leq 5) \land (a_0 < d_0 + 10) \land (b_0 \geq c_0) \\
& (c_0 > 4) \land (5 \leq 5) \land (4 < 1 + 10) \land (5 \geq c_0)
\end{align*}
\]
Terminology

- **Target constraint:**
  negated constraint

- **Target variables:**
  variables in negated constraint

- **Direct dependency** $(c_a, c_b)$:
  $\text{vars}(c_a) \cap \text{vars}(c_b) \neq 0$

- **Indirect dependency** $(c_a, c_b)$:
  $\text{vars}(c_a) \cap \text{vars}(c_1) \neq 0$, $\text{vars}(c_1) \cap \text{vars}(c_2) \neq 0$, ..., $\text{vars}(c_n) \cap \text{vars}(c_b) \neq 0$
DomainReduce Algorithm

- $s = 1$
- until sat or $s=\text{max}$ or time limit reached
  - select next subset $TV$ of target variables of size $s$
  - if no more subsets, increase $s$ and reiterate
  - identify variables dependents on $TV$ and add them to $TV$
  - keep variables in $TV$ symbolic
  - concretize all other variables
  - invoke solver
Other Techniques Considered

- **Incremental solving** (Sen et al, 2005)
  - Eliminates irrelevant constraints
  - Analogous to worst case for DomainReduce with dependencies

- **Subsumption** (Godefroid et al, 2008)
  - Eliminates implied constraints in input-bound loops
  - In hindsight, not really a constraint-optimization approach
Empirical Evaluation

**Goal:** Quantitative initial investigation of the usefulness of constraint optimization

**RQ1:** Are constraint optimization techniques effective?

**RQ2:** How do the different techniques compare to each other?
Experimental Infrastructure

- **Tool**
  Customized JFuzz/JPF framework

- **Software subjects**
  HTMLParser
  XMLParser
  K-Nearest Neighbor

- **Solvers**
  CVC3, Z3

- **Data**
  - Ten input sets per subject
  - Over 5,000 real path conditions; ∀ technique and constraint:
    - Number of path conditions solved by the technique
    - Time necessary to solve the condition (10 minutes timeout)
Experimental Infrastructure

- **Tool**
  Customized JFuzz/JPF framework

- **Software subjects**
  HTMLParser
  XMLParser
  K-Nearest Neighbor

- **Solvers**
  CVC3, Z3

- **Data**
  - Ten input sets per subject
  - Over 5,000 real path conditions; ∀ technique and constraint:
    - Number of path conditions solved by the technique
    - Time necessary to solve the condition (10 minutes timeout)

Infrastructure and data freely available online:
http://www.cc.gatech.edu/~ikpeme/software/
## Study Results 1

(\# \text{constraints processed})

<table>
<thead>
<tr>
<th>Subjects</th>
<th>PCs considered</th>
<th>PCs successfully processed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No optimization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cvc3</td>
</tr>
<tr>
<td>HTMLParser</td>
<td>1879</td>
<td>1879</td>
</tr>
<tr>
<td></td>
<td>43+1836</td>
<td>43+1836</td>
</tr>
<tr>
<td>XMLParser</td>
<td>1881</td>
<td>473</td>
</tr>
<tr>
<td></td>
<td>49+424</td>
<td>49+1832</td>
</tr>
<tr>
<td>K-NN</td>
<td>1930</td>
<td>261</td>
</tr>
<tr>
<td></td>
<td>261+0</td>
<td>936+0</td>
</tr>
</tbody>
</table>
Study Results 1
(# constraints processed)

<table>
<thead>
<tr>
<th>Subjects</th>
<th>PCs considered</th>
<th>PCs successfully processed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>unsat+sat</td>
</tr>
<tr>
<td></td>
<td>No optimization</td>
<td>Subsumption</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Incremental solving</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DomainReduce with dependencies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DomainReduce without dependencies</td>
</tr>
<tr>
<td></td>
<td>cvc3</td>
<td>z3</td>
</tr>
<tr>
<td>HTMLParser</td>
<td>1879</td>
<td></td>
</tr>
<tr>
<td></td>
<td>43+1836</td>
<td>43+1836</td>
</tr>
<tr>
<td>XMLParser</td>
<td>1881</td>
<td></td>
</tr>
<tr>
<td></td>
<td>49+424</td>
<td>49+1832</td>
</tr>
<tr>
<td>K-NN</td>
<td>1930</td>
<td></td>
</tr>
<tr>
<td></td>
<td>261+0</td>
<td>936+0</td>
</tr>
</tbody>
</table>

- Results for HTMLParser not compelling
Study Results 1
(# constraints processed)

<table>
<thead>
<tr>
<th>Subjects</th>
<th>PCs considered</th>
<th>PCs successfully processed unsat+sat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No optimization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cvc3</td>
</tr>
<tr>
<td>HTMLParser</td>
<td>1879</td>
<td>1879</td>
</tr>
<tr>
<td>XMLParser</td>
<td>1881</td>
<td>473</td>
</tr>
<tr>
<td>K-NN</td>
<td>1930</td>
<td>261</td>
</tr>
</tbody>
</table>

- Results for HTMLParser not compelling
- Optimizations ineffective for Z3
Study Results 1
(# constraints processed)

<table>
<thead>
<tr>
<th>Subjects</th>
<th>PCs considered</th>
<th>PCs successfully processed</th>
<th>No optimization</th>
<th>Subsumption</th>
<th>Incremental solving</th>
<th>DomainReduce with dependencies</th>
<th>DomainReduce without dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>cvc3</td>
<td>z3</td>
<td>cvc3</td>
<td>z3</td>
<td>cvc3</td>
</tr>
<tr>
<td>HTMLParser</td>
<td>1879</td>
<td>1879</td>
<td>1879</td>
<td>1879</td>
<td>1879</td>
<td>1879</td>
<td>1879</td>
</tr>
<tr>
<td></td>
<td></td>
<td>43+1836</td>
<td>43+1836</td>
<td>43+1836</td>
<td>43+1836</td>
<td>43+1836</td>
<td>43+1836</td>
</tr>
<tr>
<td>XMLParser</td>
<td>1881</td>
<td>473</td>
<td>1881</td>
<td>473</td>
<td>1881</td>
<td>1881</td>
<td>1881</td>
</tr>
<tr>
<td></td>
<td></td>
<td>49+424</td>
<td>49+1832</td>
<td>49+424</td>
<td>49+1832</td>
<td>49+1832</td>
<td>49+1832</td>
</tr>
<tr>
<td>K-NN</td>
<td>1930</td>
<td>261</td>
<td>936</td>
<td>261+0</td>
<td>936</td>
<td>271</td>
<td>937</td>
</tr>
<tr>
<td></td>
<td></td>
<td>936+0</td>
<td></td>
<td></td>
<td>936+0</td>
<td>936+0</td>
<td>937</td>
</tr>
</tbody>
</table>

- Results for HTMLParser not compelling
- Optimizations ineffective for Z3
  - Useless or ineffective, with one exception
### Study Results 1
(# constraints processed)

<table>
<thead>
<tr>
<th>Subjects</th>
<th>No optimization</th>
<th>Subsumption</th>
<th>Incremental solving</th>
<th>DomainReduce with dependencies</th>
<th>DomainReduce without dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cvc3 z3</td>
<td>cvc3 z3</td>
<td>cvc3 z3</td>
<td>cvc3 z3</td>
<td>cvc3 z3</td>
</tr>
<tr>
<td>HTMLParser</td>
<td>1879</td>
<td>1879</td>
<td>1879</td>
<td>1879</td>
<td>1879</td>
</tr>
<tr>
<td></td>
<td>43+1836</td>
<td>43+1836</td>
<td>43+1836</td>
<td>43+1836</td>
<td>43+1836</td>
</tr>
<tr>
<td>XMLParser</td>
<td>473</td>
<td>1881</td>
<td>473</td>
<td>1881</td>
<td>1881</td>
</tr>
<tr>
<td></td>
<td>49+424</td>
<td>49+1832</td>
<td>49+424</td>
<td>49+1832</td>
<td>49+1832</td>
</tr>
<tr>
<td>K-NN</td>
<td>261</td>
<td>936</td>
<td>261</td>
<td>936</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td>261+0</td>
<td>936+0</td>
<td>261+0</td>
<td>936+0</td>
<td>878</td>
</tr>
</tbody>
</table>

- Results for HTMLParser not compelling
- Optimizations ineffective for Z3
  - Useless or ineffective, with one exception
- DomainReduce produces negative results for K-NN (worst case)
### Study Results I

(# constraints processed)

<table>
<thead>
<tr>
<th>Subjects</th>
<th>PCs considered</th>
<th>PCs successfully processed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No optimization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cvc3</td>
</tr>
<tr>
<td>HTMLParser</td>
<td>1879</td>
<td>1879</td>
</tr>
<tr>
<td></td>
<td></td>
<td>43+1836</td>
</tr>
<tr>
<td>XMLParser</td>
<td>1881</td>
<td>473</td>
</tr>
<tr>
<td></td>
<td></td>
<td>49+424</td>
</tr>
<tr>
<td>K-NN</td>
<td>1930</td>
<td>261</td>
</tr>
<tr>
<td></td>
<td></td>
<td>261+0</td>
</tr>
</tbody>
</table>

- Results for HTMLParser not compelling
- Optimizations ineffective for Z3
  - Useless or ineffective, with one exception
- DomainReduce produces negative results for K-NN (worst case)
- Optimizations effective for CVC3
Study Results 1
(# constraints processed)

- Results for HTMLParser not compelling
- Optimizations ineffective for Z3
  - Useless or ineffective, with one exception
  - DomainReduce produces negative results for K-NN (worst case)
- Optimizations effective for CVC3
  - Small improvement for K-NN
Study Results 1
(# constraints processed)

- Results for HTMLParser not compelling
- Optimizations ineffective for Z3
  - Useless or ineffective, with one exception
  - DomainReduce produces negative results for K-NN (worst case)
- Optimizations effective for CVC3
- Small improvement for K-NN
- Dramatic improvement for XMLParser (25% → 100%)

<table>
<thead>
<tr>
<th>Subjects</th>
<th>PCs considered</th>
<th>No optimization</th>
<th>Subsumption</th>
<th>Incremental solving</th>
<th>DomainReduce with dependencies</th>
<th>DomainReduce without dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>cvc3</td>
<td>z3</td>
<td>cvc3</td>
<td>z3</td>
<td>cvc3</td>
</tr>
<tr>
<td>HTMLParser</td>
<td>1879</td>
<td>1879</td>
<td>43+1836</td>
<td>1879</td>
<td>43+1836</td>
<td>1879</td>
</tr>
<tr>
<td></td>
<td>49+424</td>
<td>1881</td>
<td>49+1832</td>
<td>473</td>
<td>49+424</td>
<td>1881</td>
</tr>
<tr>
<td>K-NN</td>
<td>1930</td>
<td>261</td>
<td>936</td>
<td>261</td>
<td>936</td>
<td>271</td>
</tr>
<tr>
<td></td>
<td>261+0</td>
<td>936+0</td>
<td>936+0</td>
<td>261+0</td>
<td>936+0</td>
<td>271+0</td>
</tr>
</tbody>
</table>
Study Results 2
(time to process constraints)
Study Results 2
(time to process constraints)
Study Results 2
(time to process constraints)
Study Results 2
(time to process constraints)
Study Results 2
(time to process constraints)
Study Results 2
(time to process constraints)
Study Results 2
(time to process constraints)
Study Results 2
(time to process constraints)

Analogous results for HTML Parser
Study Results 2
(time to process constraints)
Study Results 2
(time to process constraints)

- K-NN
  - All but one optimizations provided no benefits (timeout or unsat after a long time)
  - DomainReduce with no dependencies finds solutions for less constraints, but very quickly, for K-NN and CVC3
Study Results 2
(time to process constraints)

- **K-NN**
  - All but one optimizations provided no benefits (timeout or unsat after a long time)
  - DomainReduce with no dependencies finds solutions for less constraints, but very quickly, for K-NN and CVC3
- **HTMLParser and XMLParser**
  - Almost all optimizations improve efficiency of constraint solvers dramatically (several orders of magnitude)
Symbolic Execution and SMT Solving

```
Symbolic executor

PC: C_1 \land ... \land C_n

Sat/Unsat (solution)

Study Results (\# constraints processed)

<table>
<thead>
<tr>
<th>Subjects</th>
<th>PCs considered</th>
<th>No optimization</th>
<th>Subsumption</th>
<th>Incremental solving</th>
<th>DomainReduce with dependencies</th>
<th>DomainReduce without dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTMLParser</td>
<td>1879</td>
<td>1879</td>
<td>1879</td>
<td>1879</td>
<td>1879</td>
<td>1879</td>
</tr>
<tr>
<td>XMLParser</td>
<td>1881</td>
<td>49+424</td>
<td>49+1832</td>
<td>49+424</td>
<td>49+1832</td>
<td>49+1832</td>
</tr>
<tr>
<td>K-NN</td>
<td>1930</td>
<td>261+0</td>
<td>936+0</td>
<td>261+0</td>
<td>936+0</td>
<td>261+0</td>
</tr>
</tbody>
</table>
```

- Results for HTMLParser uninteresting
- Optimizations ineffective for Z3,
- Useless or ineffective with one exception
- DomainReduce even produce negative results for K-NN (worst case)
- Optimizations effective for CVC3
- Small improvement for K-NN
- Dramatic improvement for XMLParser (25% \to 100%)

Study Results (time to process constraints)

```
Analogous results for HTML Parser
```

DomainReduce: Intuitive View

Restrict domain of constraints to be solved by leveraging solution of similar PC

Tradeoff speed/likelihood of finding solution
Future work

- More experiments (subjects, solvers, configurations)
- Investigate why optimizations work/don’t work
- Apply optimizations in parallel
- More sophisticated optimizations (program structure or properties)
- Tighter integration