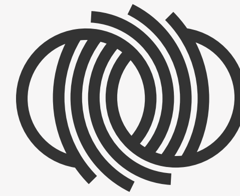
Imperial College London



SOFTWARE RELIABILITY GROUP

Constraint Solving in DSE:

THE GOOD THE BAD AND THE UGLY

Andrea Mattavelli

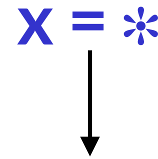
Department of Computing
Imperial College London

Dynamic Symbolic Execution

```
int bad_abs(int x) {
  if (x < 0)
    return -x;
  if (x == 1234)
    return -x;
  return x;
}
```

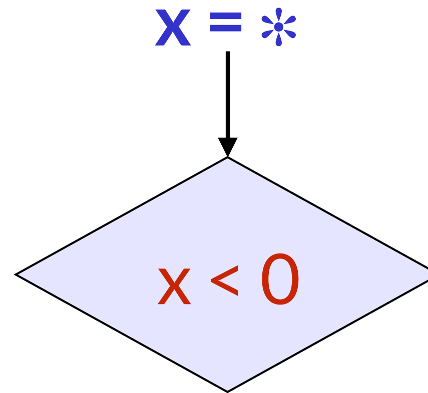
Dynamic Symbolic Execution

```
int bad_abs(int x) {
  if (x < 0)
    return -x;
  if (x == 1234)
    return -x;
  return x;
}
```



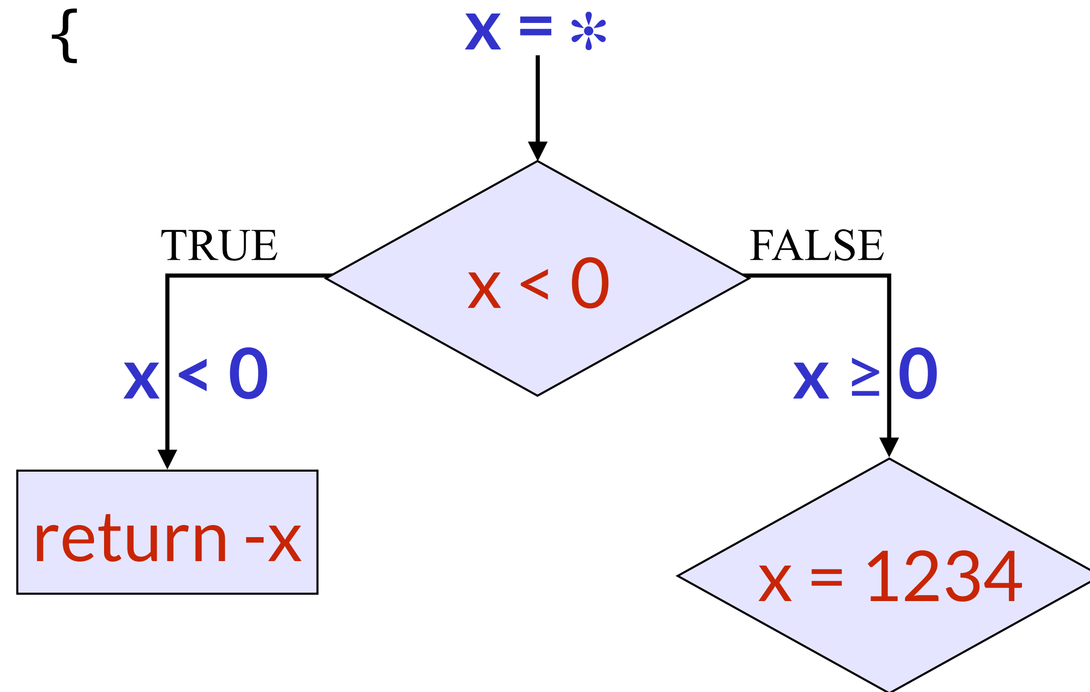
Dynamic Symbolic Execution

```
int bad_abs(int x) {
  if (x < 0)
    return -x;
  if (x == 1234)
    return -x;
  return x;
}
```



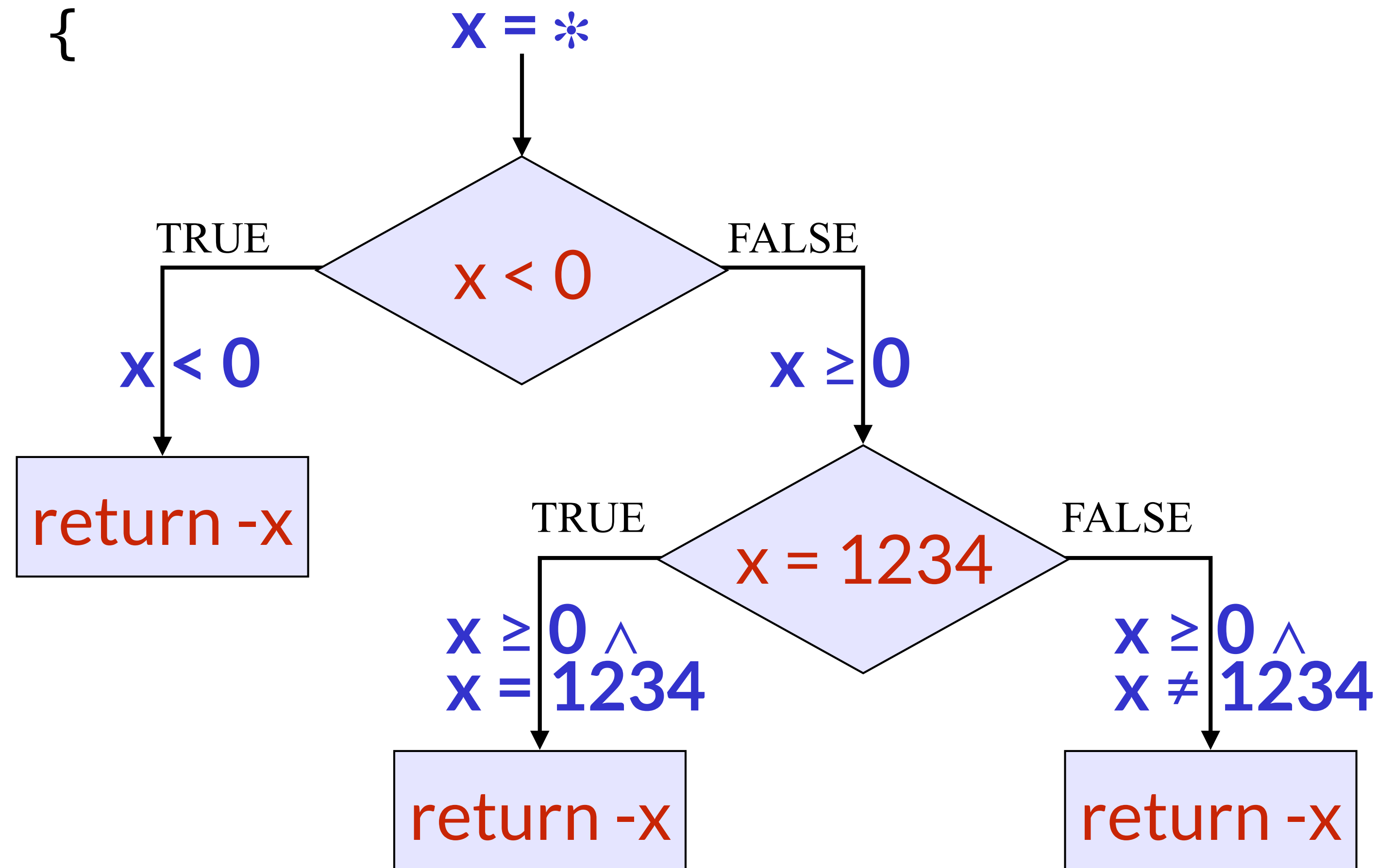
Dynamic Symbolic Execution

```
int bad_abs(int x) {
  if (x < 0)
    return -x;
  if (x == 1234)
    return -x;
  return x;
}
```



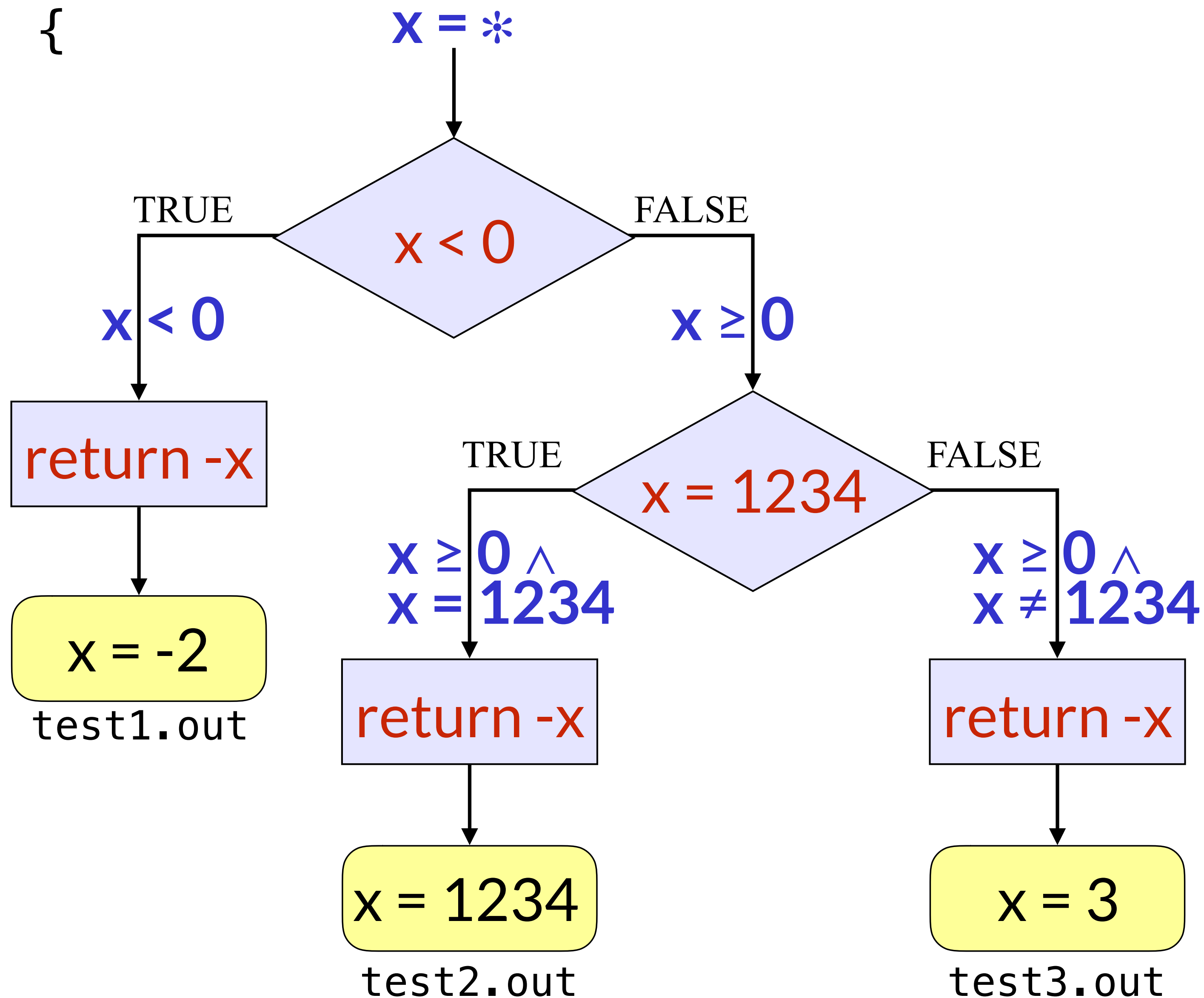
Dynamic Symbolic Execution

```
int bad_abs(int x) {
  if (x < 0)
    return -x;
  if (x == 1234)
    return -x;
  return x;
}
```



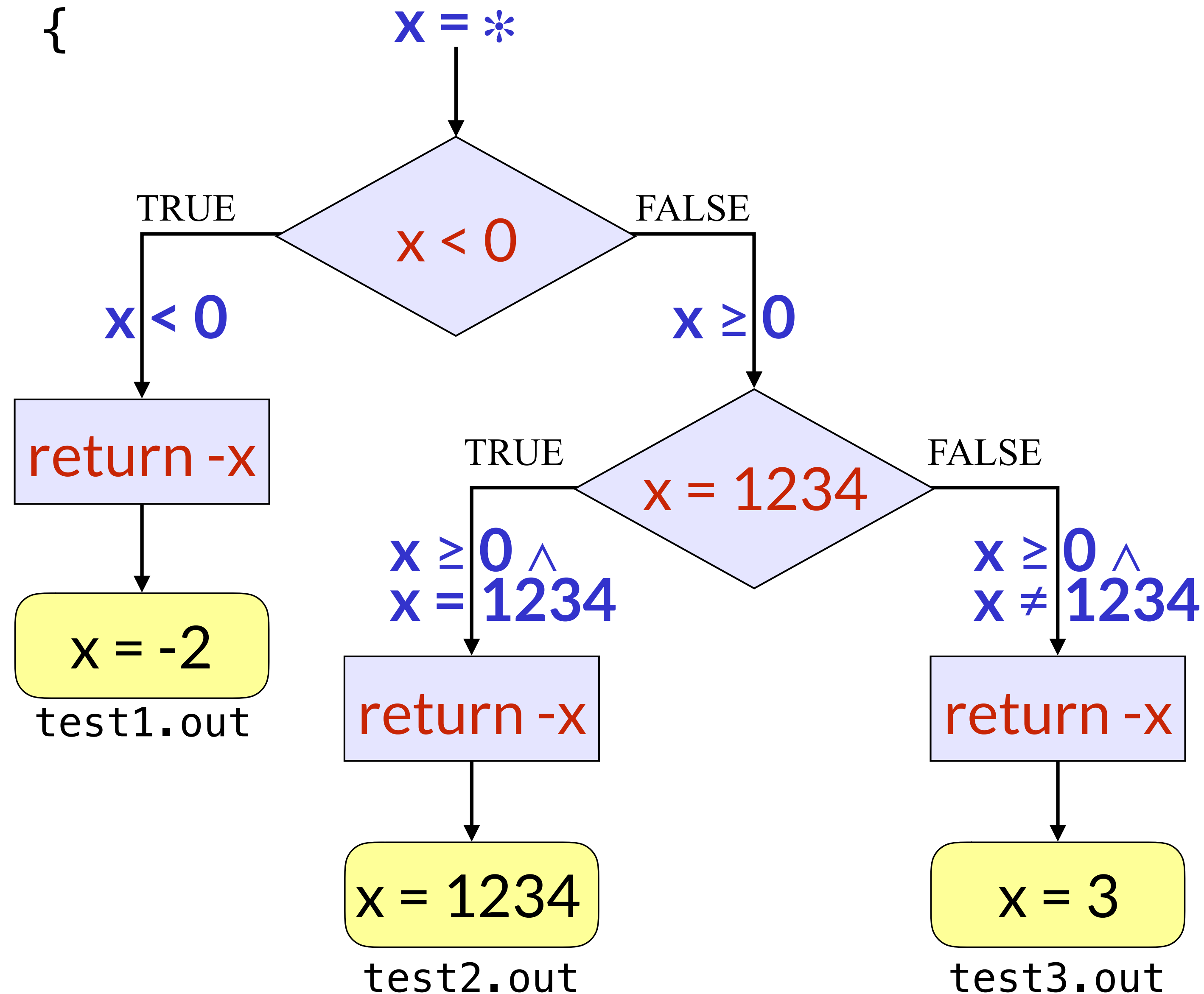
Dynamic Symbolic Execution

```
int bad_abs(int x) {
  if (x < 0)
    return -x;
  if (x == 1234)
    return -x;
  return x;
}
```



Constraint Solving & DSE

```
int bad_abs(int x) {
  if (x < 0)
    return -x;
  if (x == 1234)
    return -x;
  return x;
}
```



The Good: Checks and Accuracy

The Good: All-Value Checks

Implicit checks before each dangerous operation

- Null-pointer dereferences
- Buffer overflows
- Division/modulo by zero
- Asserts violations

All-value checks!

Errors are found if **any** buggy value exists on that path!

The Good: All-Value Checks

Implicit checks before each dangerous operation

- Null-pointer dereferences
- Buffer overflows
- Division/modulo by zero
- Asserts violations

```
int foo(unsigned k) {
   int a[4] = {3, 1, 0, 4};
   k = k % 4;
   return a[a[k]];
}
```

All-value checks!

Errors are found if **any** buggy value exists on that path!

The Good: All-Value Checks

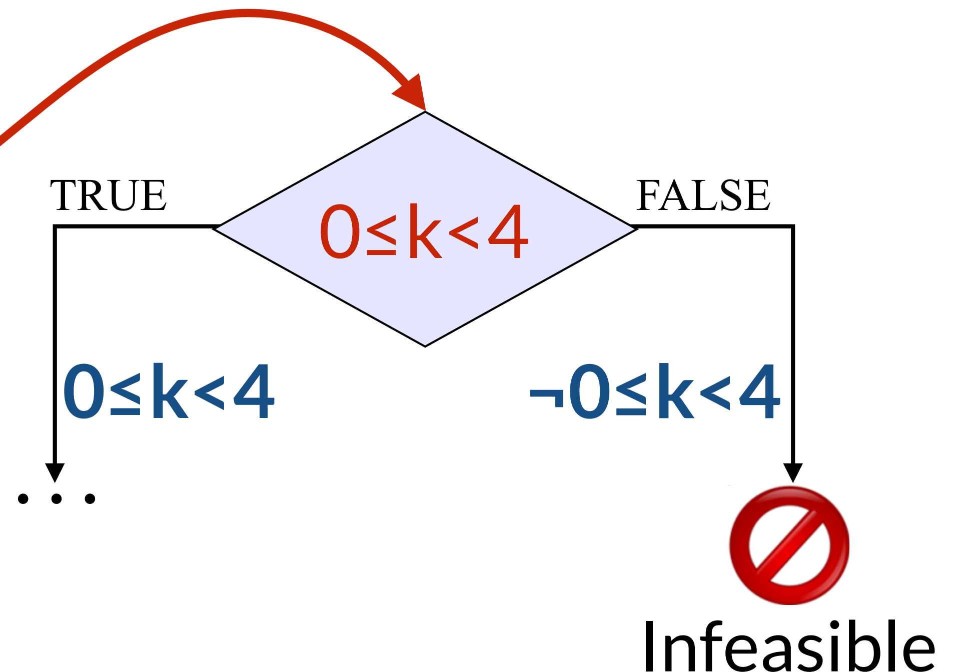
Implicit checks before each dangerous operation

- Null-pointer dereferences
- Buffer overflows
- Division/modulo by zero
- Asserts violations

All-value checks!

Errors are found if **any** buggy value exists on that path!

```
int foo(unsigned k) {
   int a[4] = {3, 1, 0, 4};
   k = k % 4;
   return a[a[k]];
}
```



The Good: All-Value Checks

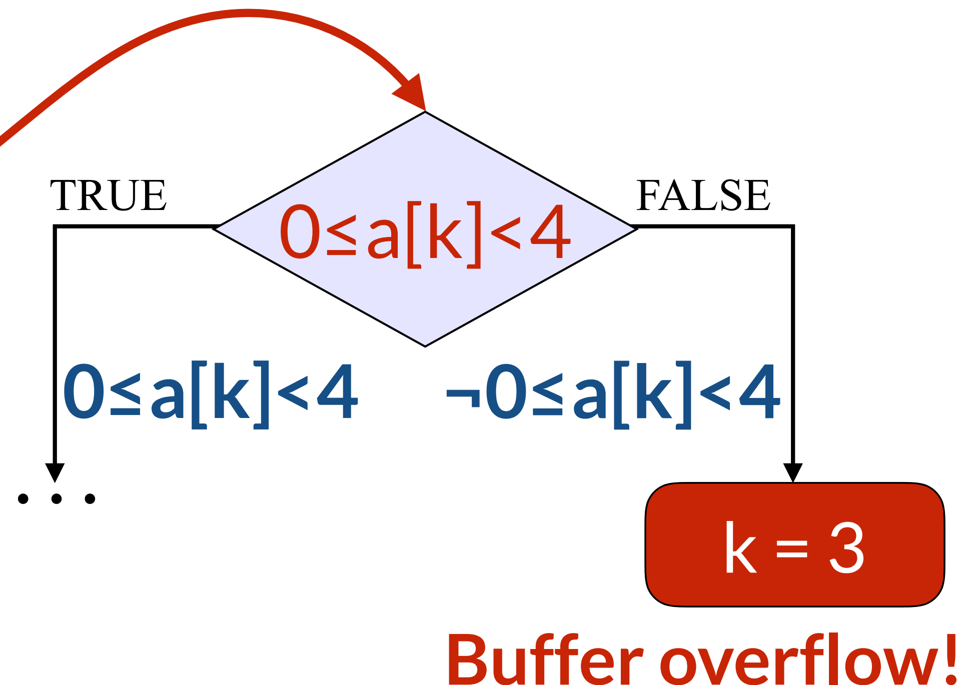
Implicit checks before each dangerous operation

- Null-pointer dereferences
- Buffer overflows
- Division/modulo by zero
- Asserts violations

All-value checks!

Errors are found if **any** buggy value exists on that path!

```
int foo(unsigned k) {
    int a[4] = {3, 1, 0, 4};
    k = k % 4;
    return a[a[k]];
}
```



The Good: Accuracy

Need constraint solver that allows bit-level modeling of memory:

- Systems code often observes the **same bytes** in **different ways**
- Bugs in systems code are often triggered by **corner cases**

The Good: Accuracy

Need constraint solver that allows bit-level modeling of memory:

- Systems code often observes the **same bytes** in **different ways**
- Bugs in systems code are often triggered by **corner cases**

```
char buf[N]; // symbolic
struct pkt1 { char x,y,v,w; int z; } *pa = (struct pkt1*) buf;
struct pkt2 { unsigned i, j; }       *pb = (struct pkt2*) buf;
if (pa[2].v < 0) { assert(pb[2].i >= 1<<23); }
```

The Good: Accuracy

Need constraint solver that allows bit-level modeling of memory:

- Systems code often observes the **same bytes** in **different ways**
- Bugs in systems code are often triggered by **corner cases**

```
char buf[N]; // symbolic
struct pkt1 { char x,y,v,w; int z; } *pa = (struct pkt1*) buf;
struct pkt2 { unsigned i, j; }       *pb = (struct pkt2*) buf;
if (pa[2].v < 0) { assert(pb[2].i >= 1<<23); }
```

```
buf: ARRAY OF BITVECTOR(8)
SBVLT(buf[18], 0x00)
BVGE(buf[19]@buf[18]@buf[17]@buf[16], 0x00800000)
```

The Bad: Scalability Challenges

The Bad: Scalability Challenges

Application	Instrs/s	Queries/s	Solver %
[	695	7.9	97.8
base64	20,520	42.2	97.0
chmod	5,360	12.6	97.2
comm	222,113	305.0	88.4
csplit	19,132	63.5	98.3
dircolors	1,019,795	4,251.7	98.6
echo	52	4.5	98.8
env	13,246	26.3	97.2
factor	12,119	22.6	99.7
join	1,033,022	3,401.2	98.1
ln	2,986	24.5	97.0
mkdir	3,895	7.2	96.6
Avg:	**196,078**	**675.5**	**97.1**

Palikareva and Cadar, "Multi-solver Support in Symbolic Execution", CAV 2013

The Bad: Scalability Challenges

Application	Instrs/s	Queries/s	Solver %
[	695	7.9	97.8
base64	20,520	42.2	97.0
chmod	5,360	12.6	97.2
comm	222,113	305.0	88.4
csplit	19,132	63.5	98.3
dircolors	1,019,795	4,251.7	98.6
echo	52	4.5	98.8
env	13,246	26.3	97.2
factor	12,119	22.6	99.7
join	1,033,022	3,401.2	98.1
ln	2,986	24.5	97.0
mkdir	3,895	7.2	96.6
Avg:	**196,078**	**675.5**	**97.1**

Palikareva and Cadar, "Multi-solver Support in Symbolic Execution", CAV 2013

The Bad: Scalability Challenges

Eliminating irrelevant constraints

- In practice, each branch usually depends on a small number of variables

```
…
…
if (x < 10) {
    …
}
```

PC: $x + y > 10 \wedge$
$z \,\&\, {-z} = z$

$x < 10$?

Cadar, Dunbar, and Engler, "KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs", OSDI 2008

The Bad: Scalability Challenges

Eliminating irrelevant constraints

- In practice, each branch usually depends on a small number of variables

```
…
…
if (x < 10) {
     …
}
```

PC: $x + y > 10 \wedge$

~~**z & z z**~~

$x < 10$?

Cadar, Dunbar, and Engler, "KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs", OSDI 2008

The Bad: Scalability Challenges

Caching Solutions

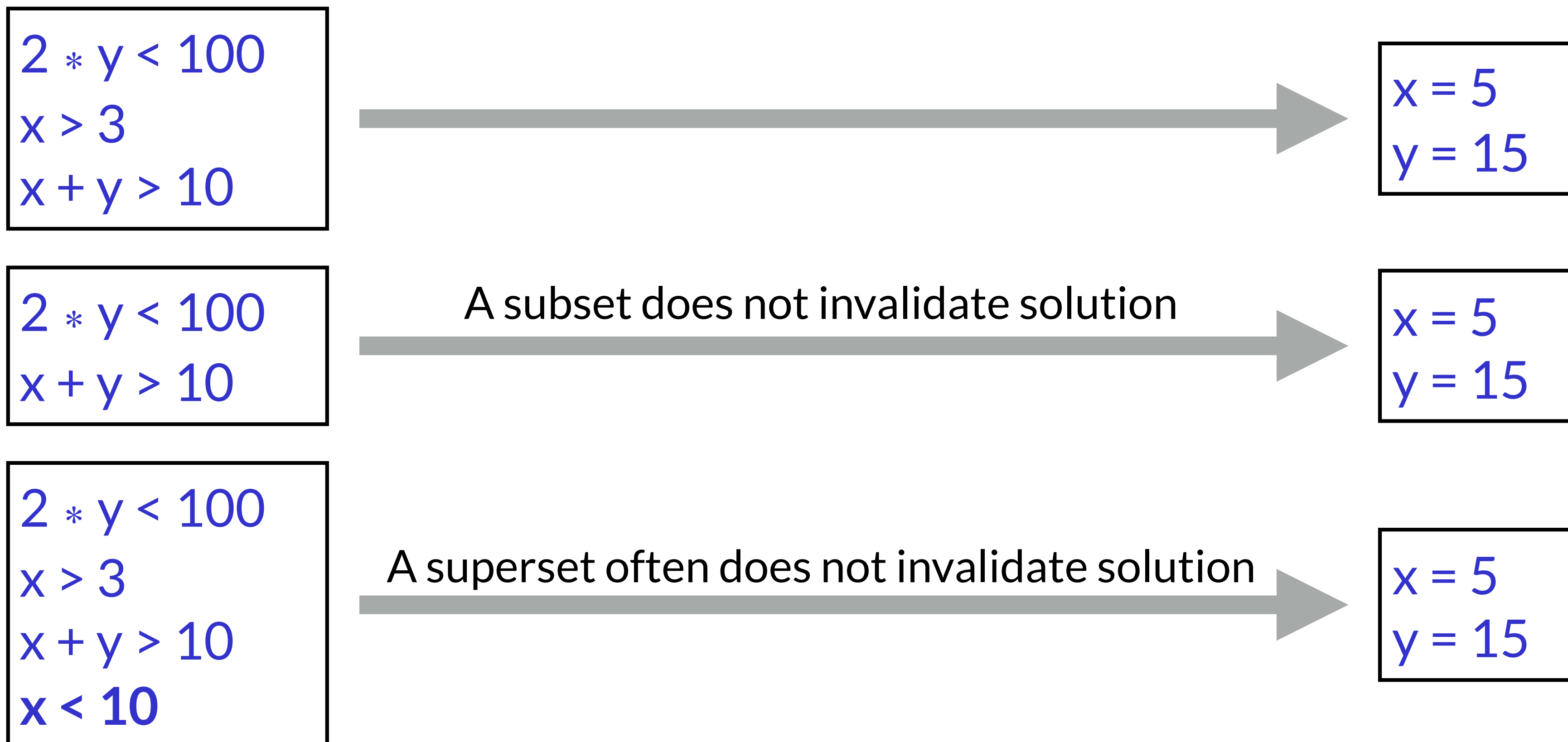
- Static set of branches: lots of similar constraint sets

Cadar, Dunbar, and Engler, "KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs", OSDI 2008

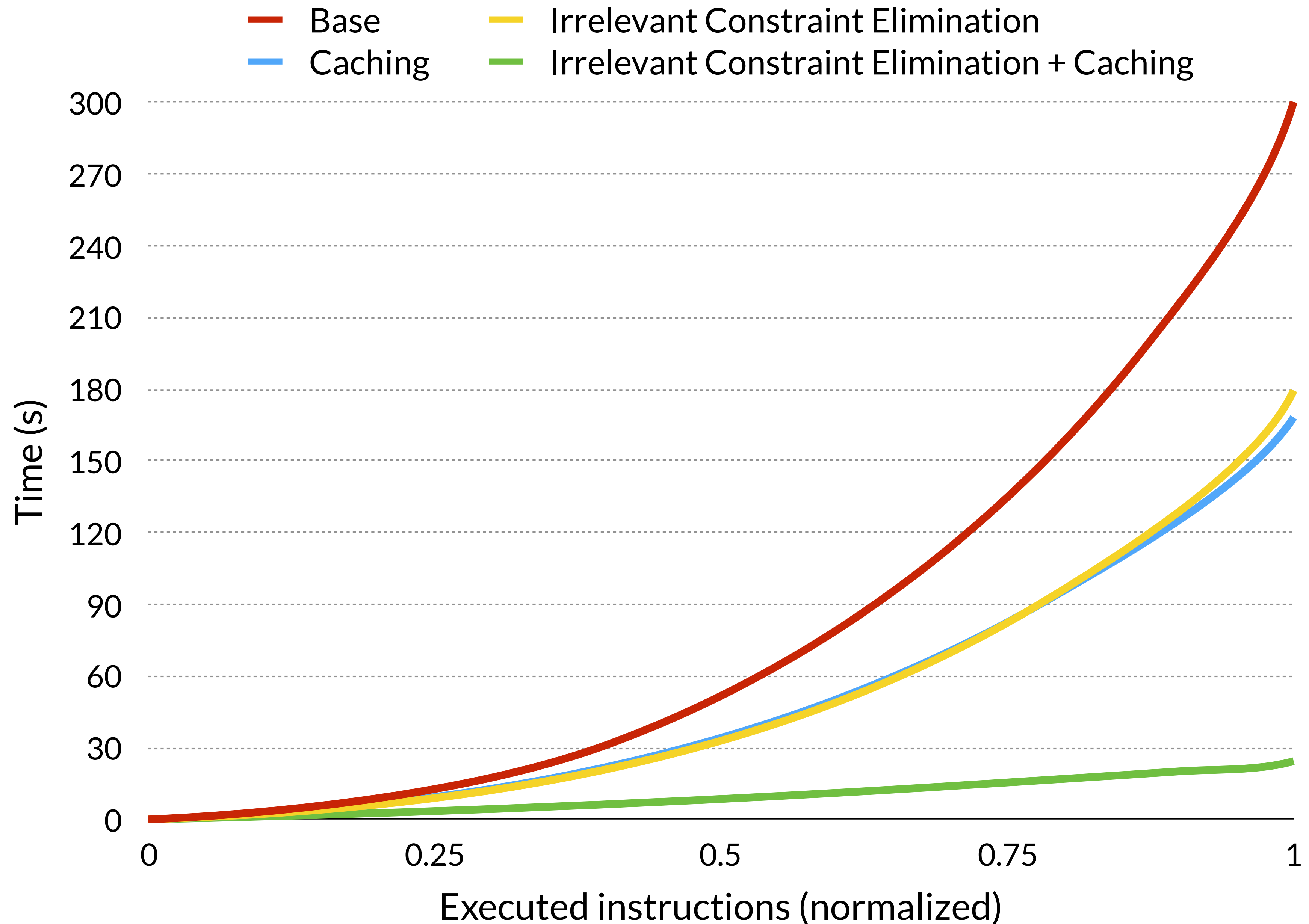
The Bad: Scalability Challenges

Caching Solutions

- Static set of branches: lots of similar constraint sets



The Bad: Scalability Challenges



The Ugly: Reasoning about Arrays

Joint work with Cristian Cadar, David Perry, Xiangyu Zhang

The Ugly: Reasoning about Arrays

Arrays are **pervasive** in software

- strings
- hash tables
- vectors
- pointer operations

Many programs generate **large** constraints involving arrays with **symbolic indexes**

The Ugly: Reasoning about Arrays

```
char b64[256] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
-1, -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};

unsigned isBase64(unsigned k) {
  if (k > 255)
      return -1;
  if (b64[k] >= 0)
      return 1;
  else return 0;
}
```

base64 from the GNU Coreutils collection

The Ugly: Reasoning about Arrays

```
char b64[256] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
-1, -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};

unsigned isBase64(unsigned k) {
  if (k > 255)
      return -1;
  if (b64[k] >= 0)
      return 1;
  else return 0;
}
```

base64 from the GNU Coreutils collection

The Ugly: Reasoning about Arrays

```
char b64[256] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
-1, -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};

unsigned isBase64(unsigned k) {
  if (k > 255)
      return -1;
  if (b64[k] >= 0)
      return 1;
  else return 0;
}
```

base64 from the GNU Coreutils collection

The Ugly: Reasoning about Arrays

```
char b64[256] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
-1, -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};

unsigned isBase64(unsigned k) {
   if (k > 255)
         return -1;
   if (b64[k] >= 0)
         return 1;
   else return 0;
}
```

base64 from the GNU Coreutils collection

The Ugly: Reasoning about Arrays

```
b64[0]=-1 ∧ b64[1]=-1 ∧ b64[2]=-1 ∧ b64[3]=-1 ∧ b64[4]=-1 ∧ b64[5]=-1 ∧ b64[6]=-1 ∧ b64[7]=-1 ∧ b64[8]=-1 ∧ b64[9]=-1 ∧ b64[10]=-1 ∧ b64[11]=-1 ∧ b64[12]=-1 ∧ b64[13]=-1 ∧
b64[14]=-1 ∧ b64[15]=-1 ∧ b64[16]=-1 ∧ b64[17]=-1 ∧ b64[18]=-1 ∧ b64[19]=-1 ∧ b64[20]=-1 ∧ b64[21]=-1 ∧ b64[22]=-1 ∧ b64[23]=-1 ∧ b64[24]=-1 ∧ b64[25]=-1 ∧ b64[26]=-1 ∧
b64[27]=-1 ∧ b64[28]=-1 ∧ b64[29]=-1 ∧ b64[30]=-1 ∧ b64[31]=-1 ∧ b64[32]=-1 ∧ b64[33]=-1 ∧ b64[34]=-1 ∧ b64[35]=-1 ∧ b64[36]=-1 ∧ b64[37]=-1 ∧ b64[38]=-1 ∧ b64[39]=-1 ∧
b64[40]=-1 ∧ b64[41]=-1 ∧ b64[42]=-1 ∧ b64[43]=62 ∧ b64[44]=-1 ∧ b64[45]=-1 ∧ b64[46]=-1 ∧ b64[47]=63 ∧ b64[48]=52 ∧ b64[49]=53 ∧ b64[50]=54 ∧ b64[51]=55 ∧ b64[52]=56 ∧
b64[53]=57 ∧ b64[54]=58 ∧ b64[55]=59 ∧ b64[56]=60 ∧ b64[57]=61 ∧ b64[58]=-1 ∧ b64[59]=-1 ∧ b64[60]=-1 ∧ b64[61]=-1 ∧ b64[62]=-1 ∧ b64[63]=-1 ∧ b64[64]=-1 ∧ b64[65]=0 ∧
b64[66]=1 ∧ b64[67]=2 ∧ b64[68]=3 ∧ b64[69]=4 ∧ b64[70]=5 ∧ b64[71]=6 ∧ b64[72]=7 ∧ b64[73]=8 ∧ b64[74]=9 ∧ b64[75]=10 ∧ b64[76]=11 ∧ b64[77]=12 ∧ b64[78]=13 ∧ b64[79]=14 ∧
b64[80]=15 ∧ b64[81]=16 ∧ b64[82]=17 ∧ b64[83]=18 ∧ b64[84]=19 ∧ b64[85]=20 ∧ b64[86]=21 ∧ b64[87]=22 ∧ b64[88]=23 ∧ b64[89]=24 ∧ b64[90]=25 ∧ b64[91]=-1 ∧ b64[92]=-1 ∧
b64[93]=-1 ∧ b64[94]=-1 ∧ b64[95]=-1 ∧ b64[96]=-1 ∧ b64[97]=26 ∧ b64[98]=27 ∧ b64[99]=28 ∧ b64[100]=29 ∧ b64[101]=30 ∧ b64[102]=31 ∧ b64[103]=32 ∧ b64[104]=33 ∧ b64[105]=34 ∧
b64[106]=35 ∧ b64[107]=36 ∧ b64[108]=37 ∧ b64[109]=38 ∧ b64[110]=39 ∧ b64[111]=40 ∧ b64[112]=41 ∧ b64[113]=42 ∧ b64[114]=43 ∧ b64[115]=44 ∧ b64[116]=45 ∧ b64[117]=46 ∧
b64[118]=47 ∧ b64[119]=48 ∧ b64[120]=49 ∧ b64[121]=50 ∧ b64[122]=51 ∧ b64[123]=-1 ∧ b64[124]=-1 ∧ b64[125]=-1 ∧ b64[126]=-1 ∧ b64[127]=-1 ∧ b64[128]=-1 ∧ b64[129]=-1 ∧
b64[130]=-1 ∧ b64[131]=-1 ∧ b64[132]=-1 ∧ b64[133]=-1 ∧ b64[134]=-1 ∧ b64[135]=-1 ∧ b64[136]=-1 ∧ b64[137]=-1 ∧ b64[138]=-1 ∧ b64[139]=-1 ∧ b64[140]=-1 ∧ b64[141]=-1 ∧
b64[142]=-1 ∧ b64[143]=-1 ∧ b64[144]=-1 ∧ b64[145]=-1 ∧ b64[146]=-1 ∧ b64[147]=-1 ∧ b64[148]=-1 ∧ b64[149]=-1 ∧ b64[150]=-1 ∧ b64[151]=-1 ∧ b64[152]=-1 ∧ b64[153]=-1 ∧
b64[154]=-1 ∧ b64[155]=-1 ∧ b64[156]=-1 ∧ b64[157]=-1 ∧ b64[158]=-1 ∧ b64[159]=-1 ∧ b64[160]=-1 ∧ b64[161]=-1 ∧ b64[162]=-1 ∧ b64[163]=-1 ∧ b64[164]=-1 ∧ b64[165]=-1 ∧
b64[166]=-1 ∧ b64[167]=-1 ∧ b64[168]=-1 ∧ b64[169]=-1 ∧ b64[170]=-1 ∧ b64[171]=-1 ∧ b64[172]=-1 ∧ b64[173]=-1 ∧ b64[174]=-1 ∧ b64[175]=-1 ∧ b64[176]=-1 ∧ b64[177]=-1 ∧
b64[178]=-1 ∧ b64[179]=-1 ∧ b64[180]=-1 ∧ b64[181]=-1 ∧ b64[182]=-1 ∧ b64[183]=-1 ∧ b64[184]=-1 ∧ b64[185]=-1 ∧ b64[186]=-1 ∧ b64[187]=-1 ∧ b64[188]=-1 ∧ b64[189]=-1 ∧
b64[190]=-1 ∧ b64[191]=-1 ∧ b64[192]=-1 ∧ b64[193]=-1 ∧ b64[194]=-1 ∧ b64[195]=-1 ∧ b64[196]=-1 ∧ b64[197]=-1 ∧ b64[198]=-1 ∧ b64[199]=-1 ∧ b64[200]=-1 ∧ b64[201]=-1 ∧
b64[202]=-1 ∧ b64[203]=-1 ∧ b64[204]=-1 ∧ b64[205]=-1 ∧ b64[206]=-1 ∧ b64[207]=-1 ∧ b64[208]=-1 ∧ b64[209]=-1 ∧ b64[210]=-1 ∧ b64[211]=-1 ∧ b64[212]=-1 ∧ b64[213]=-1 ∧
b64[214]=-1 ∧ b64[215]=-1 ∧ b64[216]=-1 ∧ b64[217]=-1 ∧ b64[218]=-1 ∧ b64[219]=-1 ∧ b64[220]=-1 ∧ b64[221]=-1 ∧ b64[222]=-1 ∧ b64[223]=-1 ∧ b64[224]=-1 ∧ b64[225]=-1 ∧
b64[226]=-1 ∧ b64[227]=-1 ∧ b64[228]=-1 ∧ b64[229]=-1 ∧ b64[230]=-1 ∧ b64[231]=-1 ∧ b64[232]=-1 ∧ b64[233]=-1 ∧ b64[234]=-1 ∧ b64[235]=-1 ∧ b64[236]=-1 ∧ b64[237]=-1 ∧
b64[238]=-1 ∧ b64[239]=-1 ∧ b64[240]=-1 ∧ b64[241]=-1 ∧ b64[242]=-1 ∧ b64[243]=-1 ∧ b64[244]=-1 ∧ b64[245]=-1 ∧ b64[246]=-1 ∧ b64[247]=-1 ∧ b64[248]=-1 ∧ b64[249]=-1 ∧
b64[250]=-1 ∧ b64[251]=-1 ∧ b64[252]=-1 ∧ b64[253]=-1 ∧ b64[254]=-1 ∧ b64[255]=-1 ∧
b64k ≥ 0 ∧ k < 256 ∧
k=0→b64k=b64[0] ∧ k=1→b64k=b64[1] ∧ k=2→b64k=b64[2] ∧ k=3→b64k=b64[3] ∧ k=4→b64k=b64[4] ∧ k=5→b64k=b64[5] ∧ k=6→b64k=b64[6] ∧ k=7→b64k=b64[7] ∧ k=8→b64k=b64[8] ∧
k=9→b64k=b64[9] ∧ k=10→b64k=b64[10] ∧ k=11→b64k=b64[11] ∧ k=12→b64k=b64[12] ∧ k=13→b64k=b64[13] ∧ k=14→b64k=b64[14] ∧ k=15→b64k=b64[15] ∧ k=16→b64k=b64[16] ∧ k=17→b64k=b64[17]
∧ k=18→b64k=b64[18] ∧ k=19→b64k=b64[19] ∧ k=20→b64k=b64[20] ∧ k=21→b64k=b64[21] ∧ k=22→b64k=b64[22] ∧ k=23→b64k=b64[23] ∧ k=24→b64k=b64[24] ∧ k=25→b64k=b64[25] ∧
k=26→b64k=b64[26] ∧ k=27→b64k=b64[27] ∧ k=28→b64k=b64[28] ∧ k=29→b64k=b64[29] ∧ k=30→b64k=b64[30] ∧ k=31→b64k=b64[31] ∧ k=32→b64k=b64[32] ∧ k=33→b64k=b64[33] ∧
k=34→b64k=b64[34] ∧ k=35→b64k=b64[35] ∧ k=36→b64k=b64[36] ∧ k=37→b64k=b64[37] ∧ k=38→b64k=b64[38] ∧ k=39→b64k=b64[39] ∧ k=40→b64k=b64[40] ∧ k=41→b64k=b64[41] ∧
k=42→b64k=b64[42] ∧ k=43→b64k=b64[43] ∧ k=44→b64k=b64[44] ∧ k=45→b64k=b64[45] ∧ k=46→b64k=b64[46] ∧ k=47→b64k=b64[47] ∧ k=48→b64k=b64[48] ∧ k=49→b64k=b64[49] ∧
k=50→b64k=b64[50] ∧ k=51→b64k=b64[51] ∧ k=52→b64k=b64[52] ∧ k=53→b64k=b64[53] ∧ k=54→b64k=b64[54] ∧ k=55→b64k=b64[55] ∧ k=56→b64k=b64[56] ∧ k=57→b64k=b64[57] ∧
k=58→b64k=b64[58] ∧ k=59→b64k=b64[59] ∧ k=60→b64k=b64[60] ∧ k=61→b64k=b64[61] ∧ k=62→b64k=b64[62] ∧ k=63→b64k=b64[63] ∧ k=64→b64k=b64[64] ∧ k=65→b64k=b64[65] ∧
k=66→b64k=b64[66] ∧ k=67→b64k=b64[67] ∧ k=68→b64k=b64[68] ∧ k=69→b64k=b64[69] ∧ k=70→b64k=b64[70] ∧ k=71→b64k=b64[71] ∧ k=72→b64k=b64[72] ∧ k=73→b64k=b64[73] ∧
k=74→b64k=b64[74] ∧ k=75→b64k=b64[75] ∧ k=76→b64k=b64[76] ∧ k=77→b64k=b64[77] ∧ k=78→b64k=b64[78] ∧ k=79→b64k=b64[79] ∧ k=80→b64k=b64[80] ∧ k=81→b64k=b64[81] ∧
k=82→b64k=b64[82] ∧ k=83→b64k=b64[83] ∧ k=84→b64k=b64[84] ∧ k=85→b64k=b64[85] ∧ k=86→b64k=b64[86] ∧ k=87→b64k=b64[87] ∧ k=88→b64k=b64[88] ∧ k=89→b64k=b64[89] ∧
k=90→b64k=b64[90] ∧ k=91→b64k=b64[91] ∧ k=92→b64k=b64[92] ∧ k=93→b64k=b64[93] ∧ k=94→b64k=b64[94] ∧ k=95→b64k=b64[95] ∧ k=96→b64k=b64[96] ∧ k=97→b64k=b64[97] ∧
k=98→b64k=b64[98] ∧ k=99→b64k=b64[99] ∧ k=100→b64k=b64[100] ∧ k=101→b64k=b64[101] ∧ k=102→b64k=b64[102] ∧ k=103→b64k=b64[103] ∧ k=104→b64k=b64[104] ∧ k=105→b64k=b64[105] ∧
k=106→b64k=b64[106] ∧ k=107→b64k=b64[107] ∧ k=108→b64k=b64[108] ∧ k=109→b64k=b64[109] ∧ k=110→b64k=b64[110] ∧ k=111→b64k=b64[111] ∧ k=112→b64k=b64[112] ∧ k=113→b64k=b64[113] ∧
k=114→b64k=b64[114] ∧ k=115→b64k=b64[115] ∧ k=116→b64k=b64[116] ∧ k=117→b64k=b64[117] ∧ k=118→b64k=b64[118] ∧ k=119→b64k=b64[119] ∧ k=120→b64k=b64[120] ∧ k=121→b64k=b64[121] ∧
k=122→b64k=b64[122] ∧ k=123→b64k=b64[123] ∧ k=124→b64k=b64[124] ∧ k=125→b64k=b64[125] ∧ k=126→b64k=b64[126] ∧ k=127→b64k=b64[127] ∧ k=128→b64k=b64[128] ∧ k=129→b64k=b64[129] ∧
k=130→b64k=b64[130] ∧ k=131→b64k=b64[131] ∧ k=132→b64k=b64[132] ∧ k=133→b64k=b64[133] ∧ k=134→b64k=b64[134] ∧ k=135→b64k=b64[135] ∧ k=136→b64k=b64[136] ∧ k=137→b64k=b64[137] ∧
k=138→b64k=b64[138] ∧ k=139→b64k=b64[139] ∧ k=140→b64k=b64[140] ∧ k=141→b64k=b64[141] ∧ k=142→b64k=b64[142] ∧ k=143→b64k=b64[143] ∧ k=144→b64k=b64[144] ∧ k=145→b64k=b64[145] ∧
k=146→b64k=b64[146] ∧ k=147→b64k=b64[147] ∧ k=148→b64k=b64[148] ∧ k=149→b64k=b64[149] ∧ k=150→b64k=b64[150] ∧ k=151→b64k=b64[151] ∧ k=152→b64k=b64[152] ∧ k=153→b64k=b64[153] ∧
k=154→b64k=b64[154] ∧ k=155→b64k=b64[155] ∧ k=156→b64k=b64[156] ∧ k=157→b64k=b64[157] ∧ k=158→b64k=b64[158] ∧ k=159→b64k=b64[159] ∧ k=160→b64k=b64[160] ∧ k=161→b64k=b64[161] ∧
k=162→b64k=b64[162] ∧ k=163→b64k=b64[163] ∧ k=164→b64k=b64[164] ∧ k=165→b64k=b64[165] ∧ k=166→b64k=b64[166] ∧ k=167→b64k=b64[167] ∧ k=168→b64k=b64[168] ∧ k=169→b64k=b64[169] ∧
k=170→b64k=b64[170] ∧ k=171→b64k=b64[171] ∧ k=172→b64k=b64[172] ∧ k=173→b64k=b64[173] ∧ k=174→b64k=b64[174] ∧ k=175→b64k=b64[175] ∧ k=176→b64k=b64[176] ∧ k=177→b64k=b64[177] ∧
k=178→b64k=b64[178] ∧ k=179→b64k=b64[179] ∧ k=180→b64k=b64[180] ∧ k=181→b64k=b64[181] ∧ k=182→b64k=b64[182] ∧ k=183→b64k=b64[183] ∧ k=184→b64k=b64[184] ∧ k=185→b64k=b64[185] ∧
k=186→b64k=b64[186] ∧ k=187→b64k=b64[187] ∧ k=188→b64k=b64[188] ∧ k=189→b64k=b64[189] ∧ k=190→b64k=b64[190] ∧ k=191→b64k=b64[191] ∧ k=192→b64k=b64[192] ∧ k=193→b64k=b64[193] ∧
k=194→b64k=b64[194] ∧ k=195→b64k=b64[195] ∧ k=196→b64k=b64[196] ∧ k=197→b64k=b64[197] ∧ k=198→b64k=b64[198] ∧ k=199→b64k=b64[199] ∧ k=200→b64k=b64[200] ∧ k=201→b64k=b64[201] ∧
k=202→b64k=b64[202] ∧ k=203→b64k=b64[203] ∧ k=204→b64k=b64[204] ∧ k=205→b64k=b64[205] ∧ k=206→b64k=b64[206] ∧ k=207→b64k=b64[207] ∧ k=208→b64k=b64[208] ∧ k=209→b64k=b64[209] ∧
k=210→b64k=b64[210] ∧ k=211→b64k=b64[211] ∧ k=212→b64k=b64[212] ∧ k=213→b64k=b64[213] ∧ k=214→b64k=b64[214] ∧ k=215→b64k=b64[215] ∧ k=216→b64k=b64[216] ∧ k=217→b64k=b64[217] ∧
k=218→b64k=b64[218] ∧ k=219→b64k=b64[219] ∧ k=220→b64k=b64[220] ∧ k=221→b64k=b64[221] ∧ k=222→b64k=b64[222] ∧ k=223→b64k=b64[223] ∧ k=224→b64k=b64[224] ∧ k=225→b64k=b64[225] ∧
k=226→b64k=b64[226] ∧ k=227→b64k=b64[227] ∧ k=228→b64k=b64[228] ∧ k=229→b64k=b64[229] ∧ k=230→b64k=b64[230] ∧ k=231→b64k=b64[231] ∧ k=232→b64k=b64[232] ∧ k=233→b64k=b64[233] ∧
k=234→b64k=b64[234] ∧ k=235→b64k=b64[235] ∧ k=236→b64k=b64[236] ∧ k=237→b64k=b64[237] ∧ k=238→b64k=b64[238] ∧ k=239→b64k=b64[239] ∧ k=240→b64k=b64[240] ∧ k=241→b64k=b64[241] ∧
k=242→b64k=b64[242] ∧ k=243→b64k=b64[243] ∧ k=244→b64k=b64[244] ∧ k=245→b64k=b64[245] ∧ k=246→b64k=b64[246] ∧ k=247→b64k=b64[247] ∧ k=248→b64k=b64[248] ∧ k=249→b64k=b64[249] ∧
k=250→b64k=b64[250] ∧ k=251→b64k=b64[251] ∧ k=252→b64k=b64[252] ∧ k=253→b64k=b64[253] ∧ k=254→b64k=b64[254] ∧ k=255→b64k=b64[255]
```

The Ugly: Reasoning about Arrays

```
char equivClass[256] = {0, 1, 1, .. 10, 11, 12, 13 .. 1, 1, , 1};
char accept[298]     = {0, 0, .. 41, 32, 28,  ..   23, 2, 2, 0};
unsigned check[580]  = {0, 1, 1, .. 141, 142, 144, .. 297, 297};
unsigned base[302]   = {0, 0, .. 412, 422, .. 120, 395, 398};
unsigned def[302]    = {0, 297,  .. 63, 300,   .. 297, 297, 297};
char meta[54]        = {0, 1, 1, .. 1, 3, 3, .. 3, 3, 1, 1, 1};
unsigned next[580]   = {0, 6, 7, .. 132, 133,  .. 297, 297};

Void tokenMatch(char *input) {
  unsigned currState = 0;
  char charPtr = input;
  do {
    char currClass = equivClass[*charPtr];
    if(accept[currState]) {
      lastAcceptState = currState;
      lastAcceptPos = charPtr;
    }
    while(check[base[currState] + currClass] != currState) {
      currState = def[currState];
      if(currState >= 298)
        currClass = meta[currClass];
    }
    currState = next[base[currState] + currClass];
    ++charPtr;
  } while(base[currState] != 526);
}
```

GNU BC

The Ugly: Reasoning about Arrays

```
char equivClass[256] = {0, 1, 1, .. 10, 11, 12, 13 .. 1, 1, , 1};
char accept[298]     = {0, 0, .. 41, 32, 28,  ..   23, 2, 2, 0};
unsigned check[580]  = {0, 1, 1, .. 141, 142, 144, .. 297, 297};
unsigned base[302]   = {0, 0, .. 412, 422, .. 120, 395, 398};
unsigned def[302]    = {0, 297,  .. 63, 300,   .. 297, 297, 297};
char meta[54]        = {0, 1, 1, .. 1, 3, 3, .. 3, 3, 1, 1, 1};
unsigned next[580]   = {0, 6, 7, .. 132, 133,  .. 297, 297};

Void tokenMatch(char *input) {
  unsigned currState = 0;
  char charPtr = input;
  do {
    char currClass = equivClass[*charPtr];
    if(accept[currState]) {
      lastAcceptState = currState;
      lastAcceptPos = charPtr;
    }
    while(check[base[currState] + currClass] != currState) {
      currState = def[currState];
      if(currState >= 298)
        currClass = meta[currClass];
    }
    currState = next[base[currState] + currClass];
    ++charPtr;
  } while(base[currState] != 526);
}
```

KLEE 1h DFS mode → 20 paths

GNU BC

The Ugly: Reasoning about Arrays

```
char equivClass[256] = {0, 1, 1, .. 10, 11, 12, 13 .. 1, 1, , 1};
char accept[298]     = {0, 0, .. 41, 32, 28,  ..   23, 2, 2, 0};
unsigned check[580]  = {0, 1, 1, .. 141, 142, 144, .. 297, 297};
unsigned base[302]   = {0, 0, .. 412, 422, .. 120, 395, 398};
unsigned def[302]    = {0, 297,  .. 63, 300,   .. 297, 297, 297};
char meta[54]        = {0, 1, 1, .. 1, 3, 3, .. 3, 3, 1, 1, 1};
unsigned next[580]   = {0, 6, 7, .. 132, 133,  .. 297, 297};

Void tokenMatch(char *input) {
  unsigned currState = 0;
  char charPtr = input;
  do {
    char currClass = equivClass[*charPtr];
    if(accept[currState]) {
      lastAcceptState = currState;
      lastAcceptPos = charPtr;
    }
    while(check[base[currState] + currClass] != currState) {
      currState = def[currState];
      if(currState >= 298)
        currClass = meta[currClass];
    }
    currState = next[base[currState] + currClass];
    ++charPtr;
  } while(base[currState] != 526);
}
```

KLEE 1h DFS mode → 828 paths

GNU BC

The Ugly: Reasoning about Arrays

```
char b64[256] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
-1, -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};

unsigned isBase64(unsigned k) {
  if (k > 255)
      return -1;
  if (b64[k] >= 0)
      return 1;
  else return 0;
}
```

base64 from the GNU Coreutils collection

The Ugly: Reasoning about Arrays

```
char b64[256] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
-1, -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};

unsigned isBase64(unsigned k) {
  if (k > 255)
      return -1;
  if (b64[k] >= 0)
      return 1;
  else return 0;
}
```

base64 from the GNU Coreutils collection

The Ugly: Reasoning about Arrays

```c
char b64[256] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
-1, -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};

unsigned isBase64(unsigned k) {
  if (k > 255)
      return -1;
  if (b64[k] >= 0)
      return 1;
  else return 0;
}
```

base64 from the GNU Coreutils collection

The Ugly: Reasoning about Arrays

```
char b64[256] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
-1, -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};

unsigned isBase64(unsigned k) {
  if (k > 255)
      return -1;
  if (k==43|(47≤k≤57)|(65≤k≤90)|(97≤k≤122))
      return 1;
  else return 0;
}
```

base64 from the GNU Coreutils collection

The Ugly: Reasoning about Arrays

```
char b64[256] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
-1, -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};

unsigned isBase64(unsigned k) {
   if (k > 255)
       return -1;
   if (k==43|(47≤k≤57)|(65≤k≤90)|(97≤k≤122))
       return 1;
   else return 0;
}
```

Index-based transformation

base64 from the GNU Coreutils collection

The Ugly: Reasoning about Arrays

```c
char b64[256] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
-1, -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};

b64[k]
```

The Ugly: Reasoning about Arrays

```
char b64[256] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
-1, -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};

b64[k]
```

The Ugly: Reasoning about Arrays

```
char b64[256] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
-1, -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};
```

```
b64[k] ──────→ ITE(0≤k≤42|44≤k≤46|58≤k≤64|91≤k≤96|123≤k≤255,
                    -1,
                    ITE(k==43, 62,
                               ITE(…
```

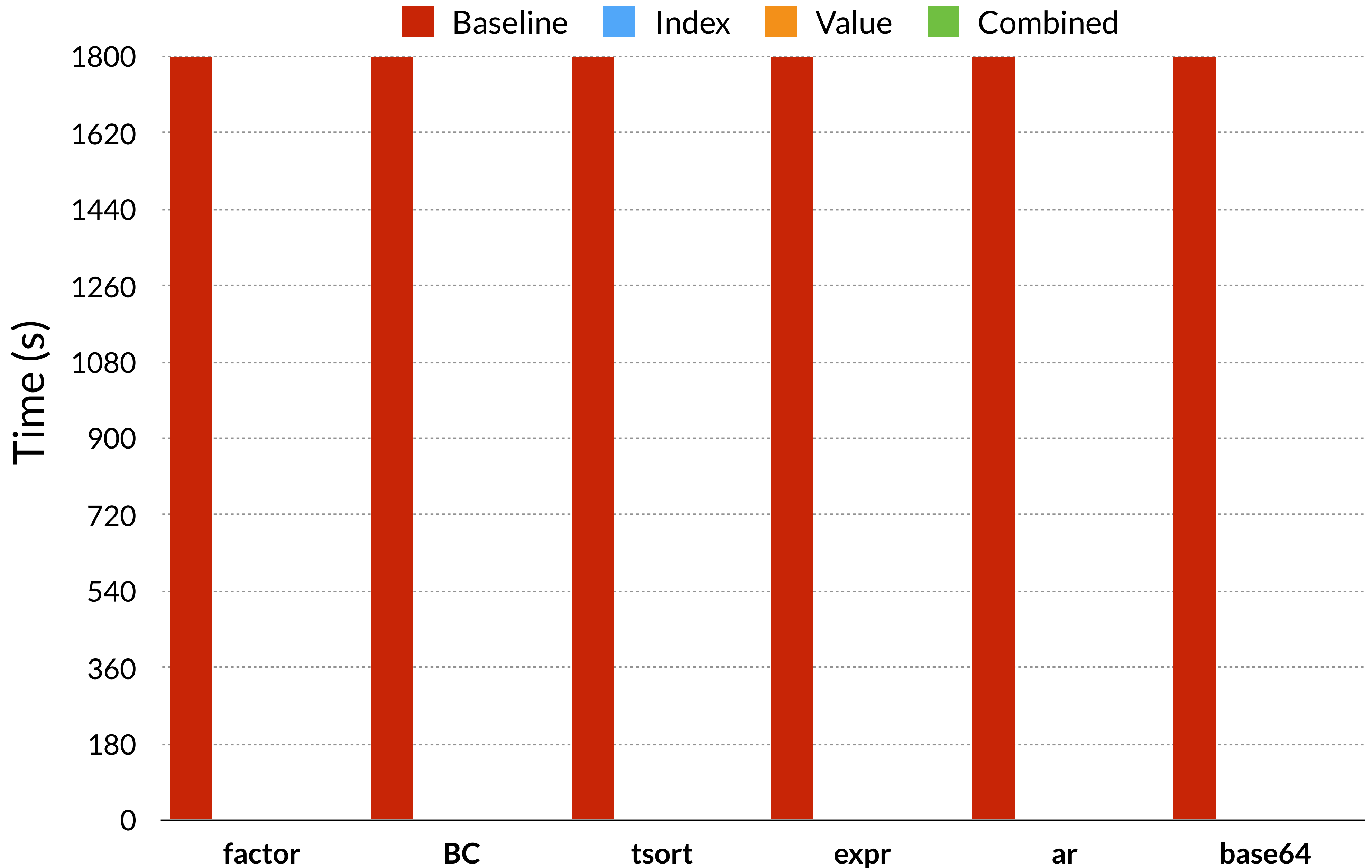
The Ugly: Reasoning about Arrays

```
char b64[256] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, 3, 4, 5, 6, 7,
8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1,
-1, -1, -1, -1, -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};
```

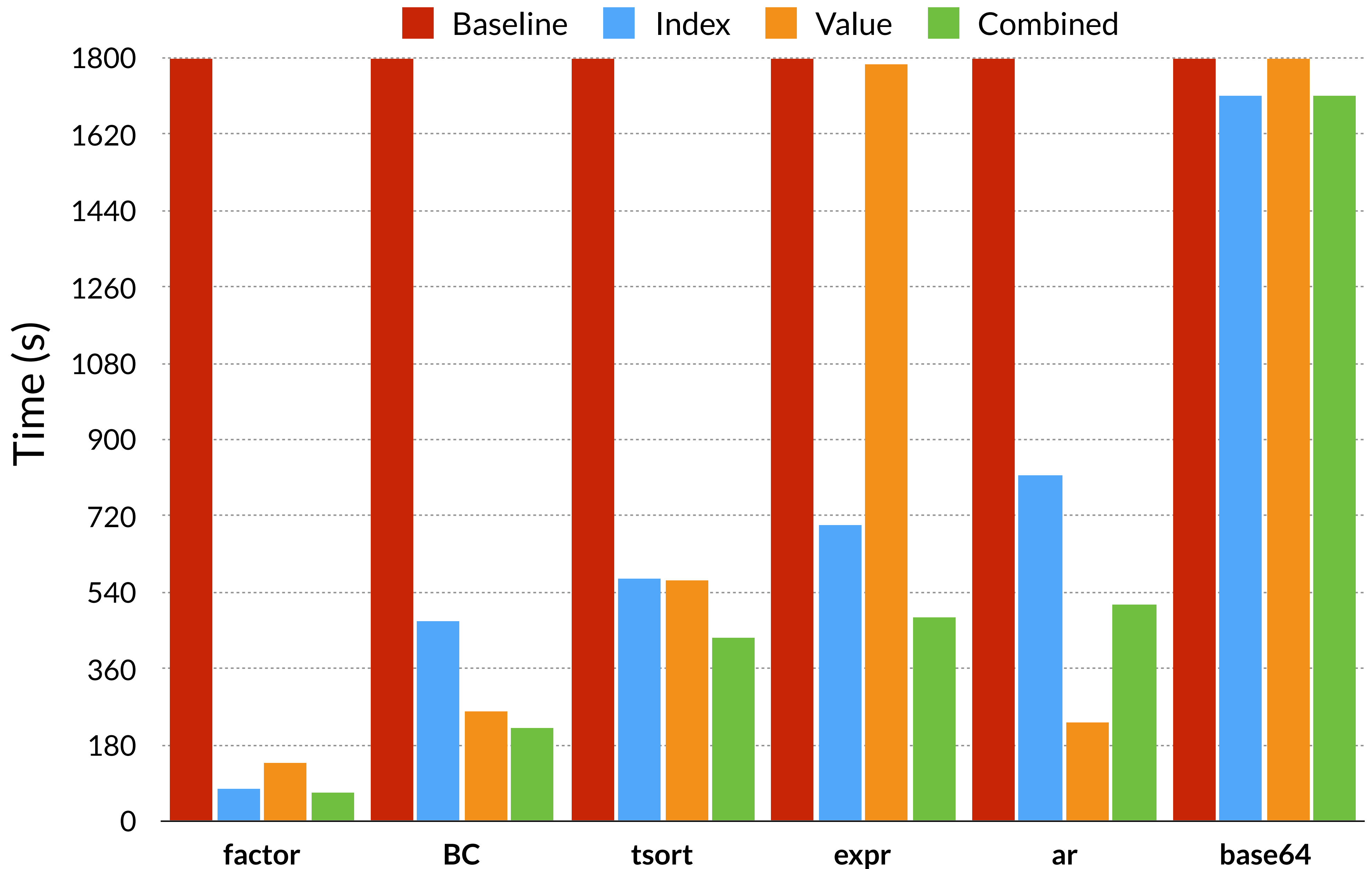
```
b64[k] ──────► ITE(0≤k≤42|44≤k≤46|58≤k≤64|91≤k≤96|123≤k≤255,
                    -1,
                    ITE(k==43, 62,
                               ITE(…
```

Value-based transformation

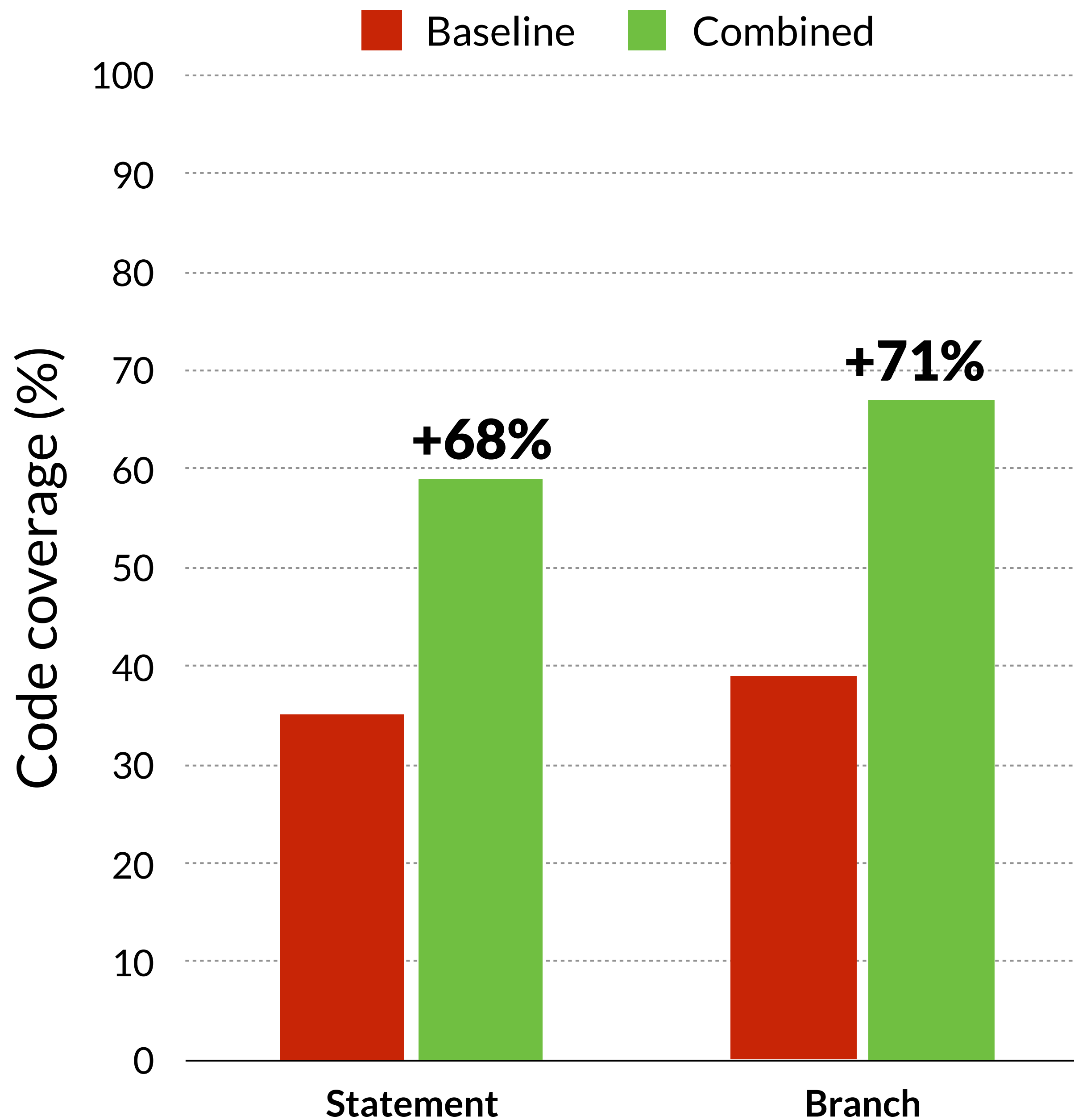
The Ugly: Reasoning about Arrays



The Ugly: Reasoning about Arrays



The Ugly: Reasoning about Arrays



The Good: Checks and Accuracy

The Bad: Scalability Challenges

The Ugly: Reasoning about Arrays

~~The Ugly:~~ Reasoning about Arrays