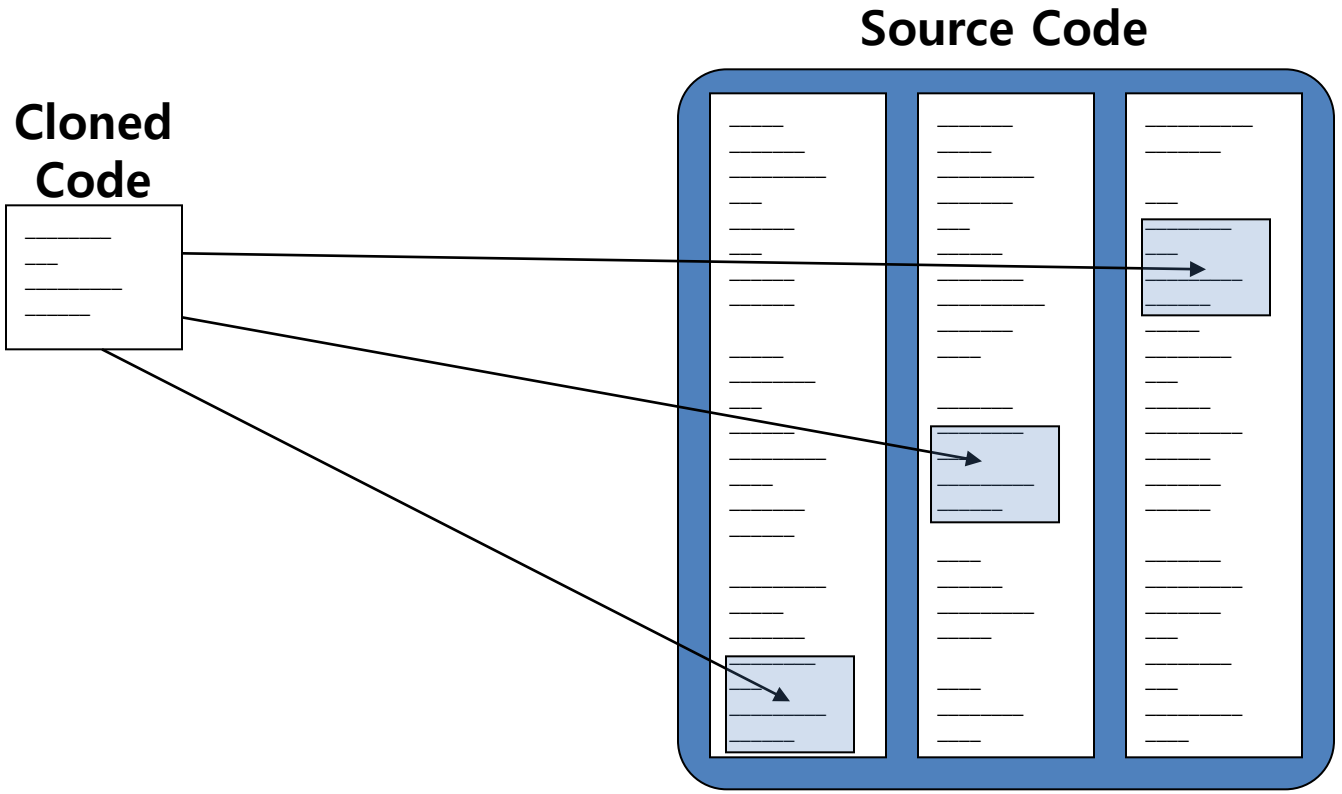
Static Analysis Based Code Clone Detector

Heejung Kim
hjkim@ropas.snu.ac.kr

Jan 8, 2010
ROSAEC Workshop

What is code clones?

- A code fragment which has identical or similar co de fragments in the source code



Existing Detection Techniques

- **Line-based**
  - Dup[Baker, CSS'92]
- **Token-based**
  - CCFinder [Kamiya et al., TSE'02]
- **AST(Abstract Syntax Tree)-based**
  - Deckard [Jiang et al. ICSE'07]
- **PDG(Program Dependency Graph)-based**
  - PDG-DUP[Komondoor and Horwitz SAS'01]

Code Clone Types

- **Textual Similarity : Syntactic equivalence**

```
int f1(int a, int b)
{
    int c;
    if (a > 10)
      c = b + 5;
    else
      c = 10;
    return c;
}
```

```
int f2(int x, int z){
int y;
if (x > 10)
  y = z + 5;
else
  y = 10;
return y;
}
```

Code Clone Types

Functional Similarity : Semantic equivalence

```
1  static PyObject *
2  PyCurses_Init_Color(PyObject *self, PyObject *args)
3  {
4    short color, r, g, b;
5
6    PyCursesInitialised
7    PyCursesInitialisedColor
8
9    switch(PyTuple_Size(args)) {
10   case 4:
11     if (!PyArg_ParseTuple(args, "hhhh;color,r,g,b",
12           &color, &r, &g, &b))
13       return NULL;
14     break;
15   default:
16     PyErr_SetString(PyExc_TypeError,
17         "init_color requires 4 arguments");
18     return NULL;
19   }
20
21   return PyCursesCheckERR(init_color(color, r, g, b),
22           "init_color");
23 }
```

```
1  static PyObject *
2  PyCurses_Init_Pair(PyObject *self, PyObject *args)
3  {
4    short pair, f, b;
5
6    PyCursesInitialised
7    PyCursesInitialisedColor
8
9    if(PyTuple_Size(args) != 3) {
10     PyErr_SetString(PyExc_TypeError,
11         "init_pair requires 3 arguments");
12     return NULL;
13   }
14
15   if(!PyArg_ParseTuple(args, "hhh;pair, f, b",
16       &pair, &f, &b))
17     return NULL;
18
19   return PyCursesCheckERR(init_pair(pair, f, b),
20           "init_pair");
21 }
```

Code Clone Types

¤ Functional Similarity : Semantic equivalence

```
1  static PyObject *
2  PyCurses_Init_Color(PyObject *self, PyObject *args)
3  {
4    short color, r, g, b;
5
6    PyCursesInitialised
7    PyCursesInitialisedColor
8
9    switch(PyTuple_Size(args)) {
10   case 4:
11     if (!PyArg_ParseTuple(args, "hhhh;color,r,g,b",
12           &color, &r, &g, &b))
13       return NULL;
14     break;
15   default:
16     PyErr_SetString(PyExc_TypeError,
17         "init_color requires 4 arguments");
18     return NULL;
19   }
20
21   return PyCursesCheckERR(init_color(color, r, g, b),
22           "init_color");
23 }
```

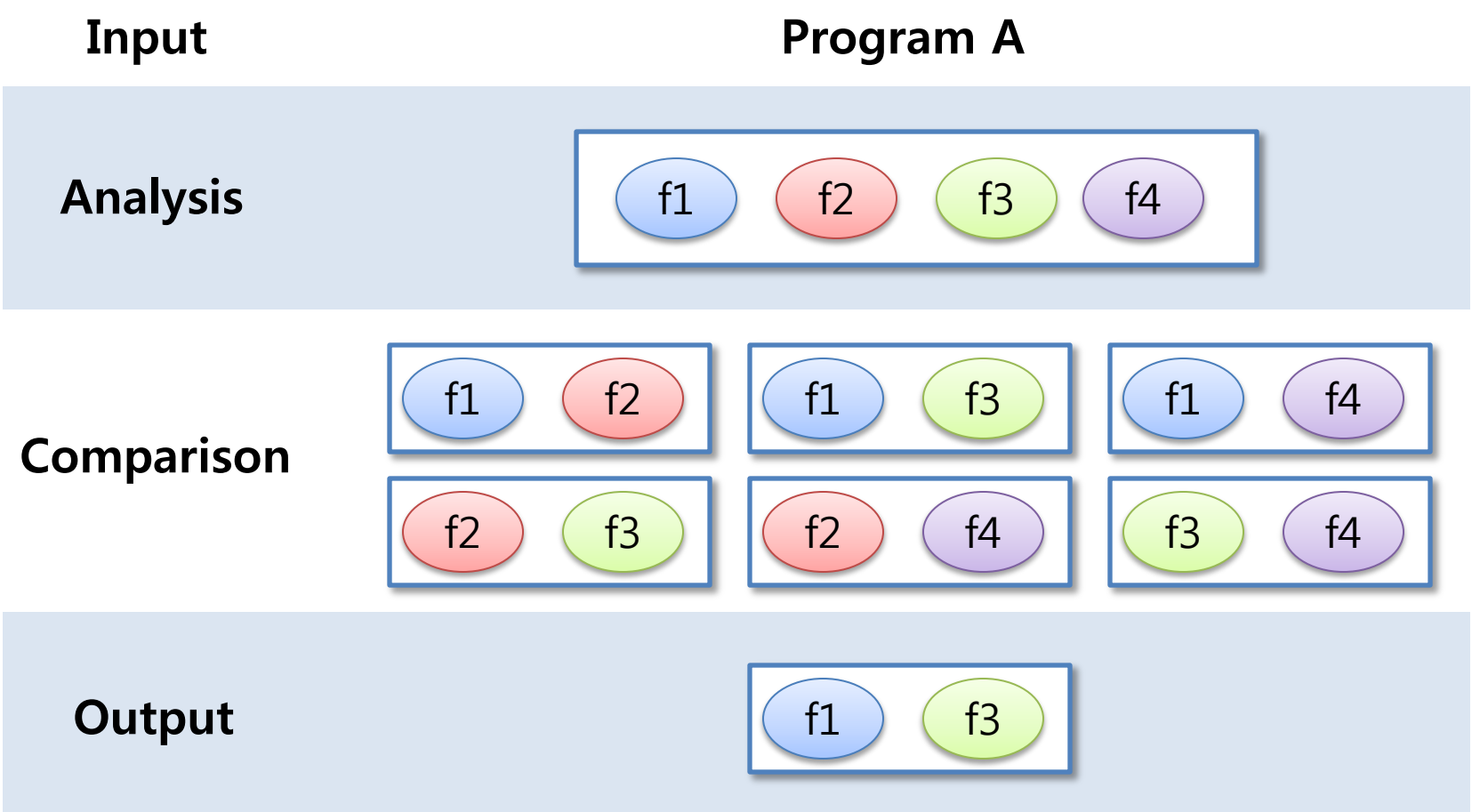
```
1  static PyObject *
2  PyCurses_Init_Pair(PyObject *self, PyObject *args)
3  {
4    short pair, f, b;
5
6    PyCursesInitialised
7    PyCursesInitialisedColor
8
9    switch(PyTuple_Size(args)) {
10   case 3:
11     if (!PyArg_ParseTuple(args, "hhh;pair, f, b",
12           &pair, &f, &b))
13       return NULL;
14     break;
15   default:
16     PyErr_SetString(PyExc_TypeError,
17         "init_pair requires 3 arguments");
18     return NULL;
19   }
20
21   return PyCursesCheckERR(init_pair(pair, f, b),
22           "init_pair");
23 }
```

Motivation

- Find semantic clones!
- If two functions have similar memory states, they would be a code clone
- Use Path-sensitive Mairac to get memory states

Detection Process

¤ Clone Unit : Function



Comparison Algorithm

```
int f1(int a, int b)
{
    int c;
    if (a > 10)
      c = b + 5;
    else
      c = 10;
    return c;
}
```

f1's Memory

Variable a → {[a::true]}

Variable b → {[b::(a>10)]}

Variable c → {[(b+5)::(a>10)],
[10::(a<=10)]}

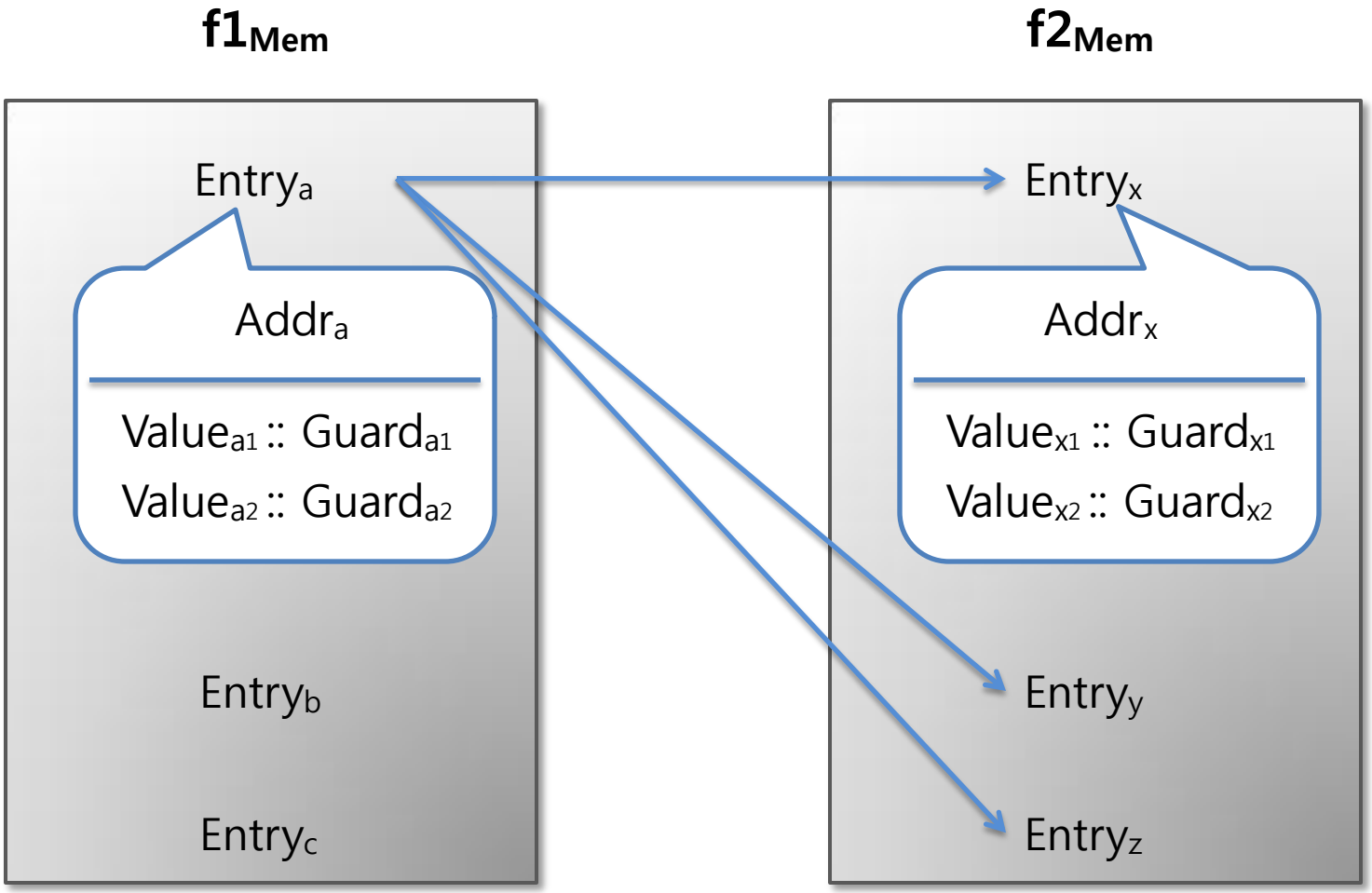
Address — Guarded Values

Entry$_c$ in Memory state : Variable c → {[(b+5)::(a>10)] , [10::(a<=10)]}

Value — Guard

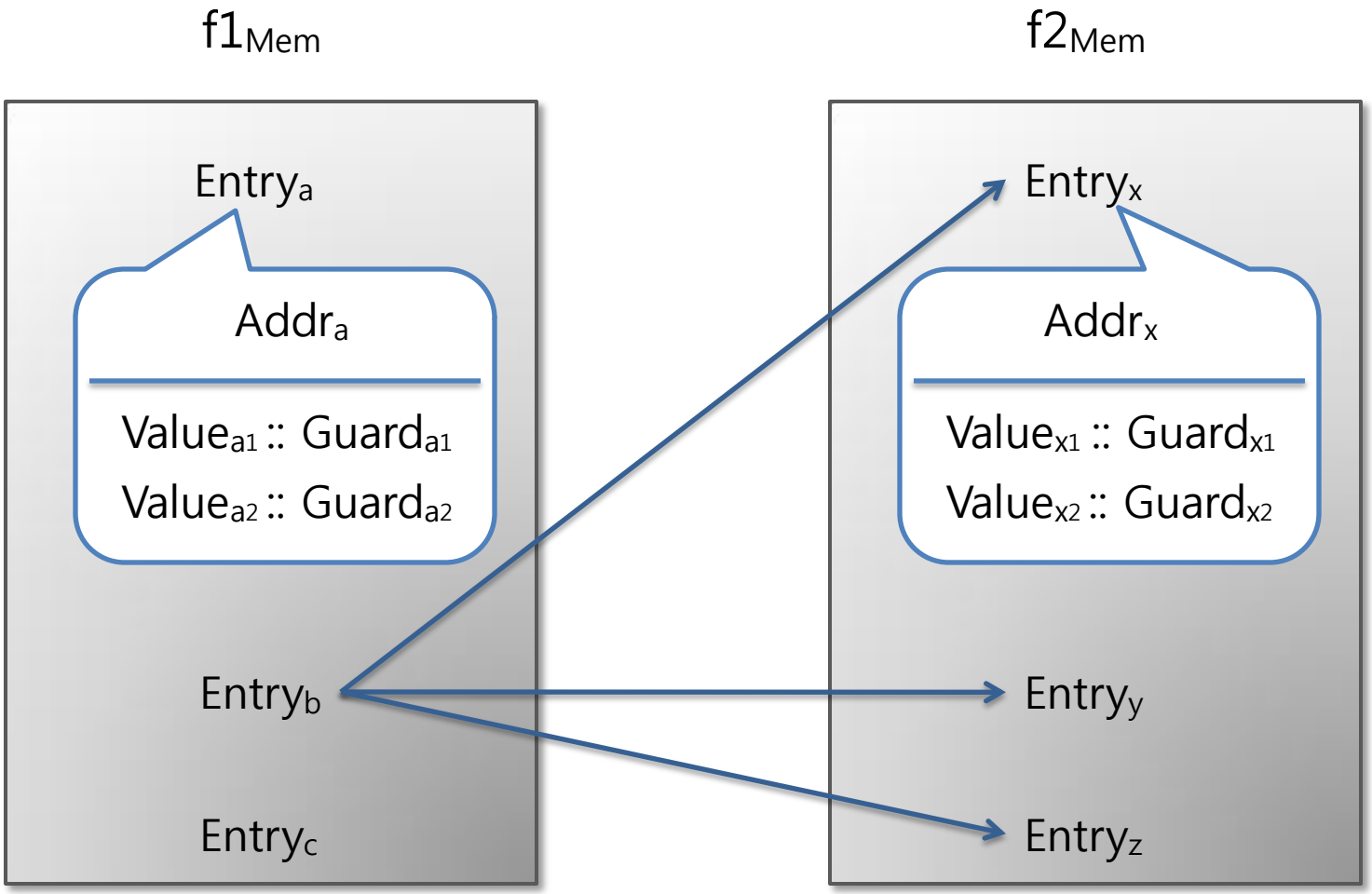
Comparison Algorithm

- **Function f1 vs Function f2**



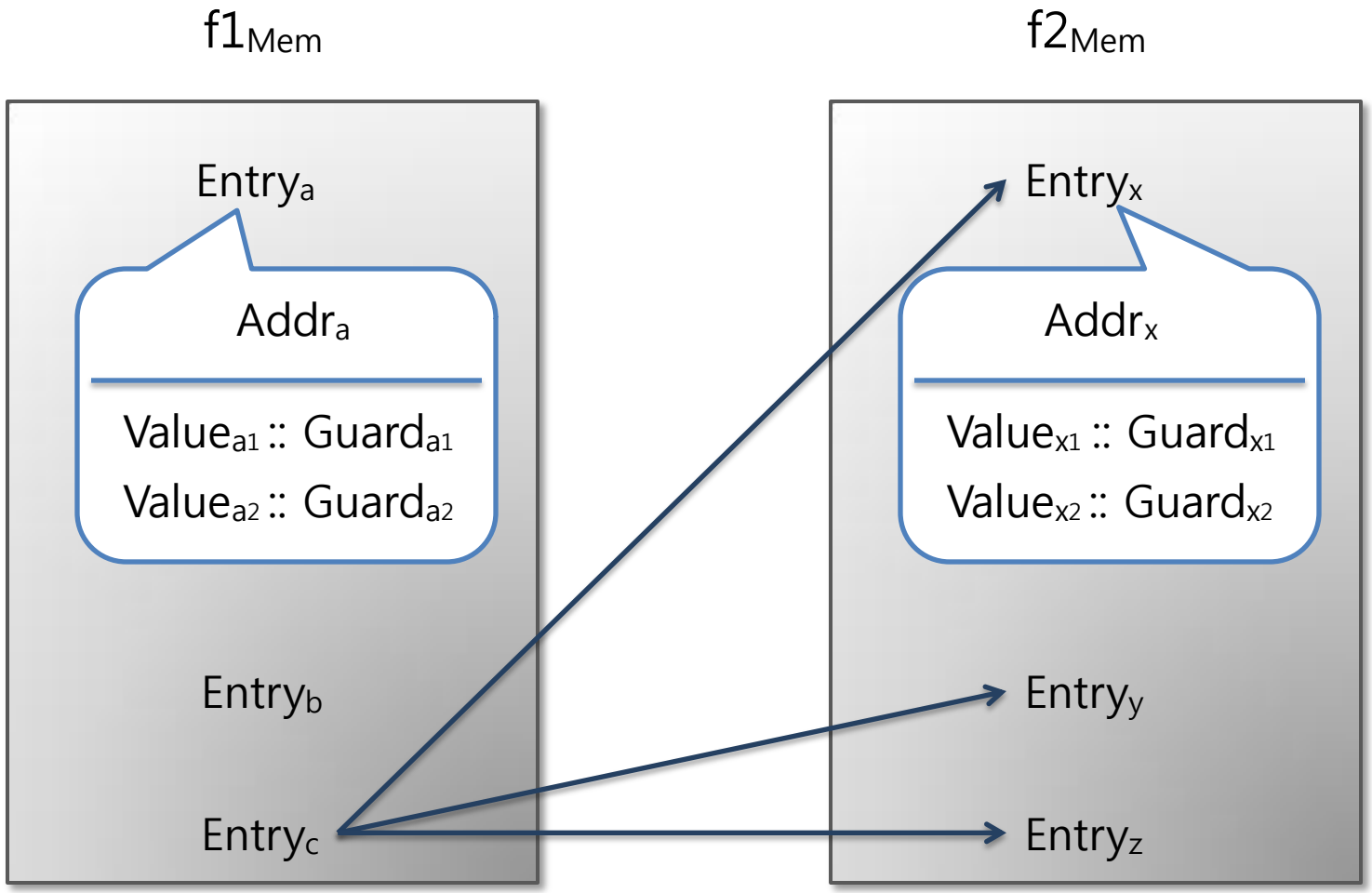
Comparison Algorithm

- **Function f1 vs Function f2**



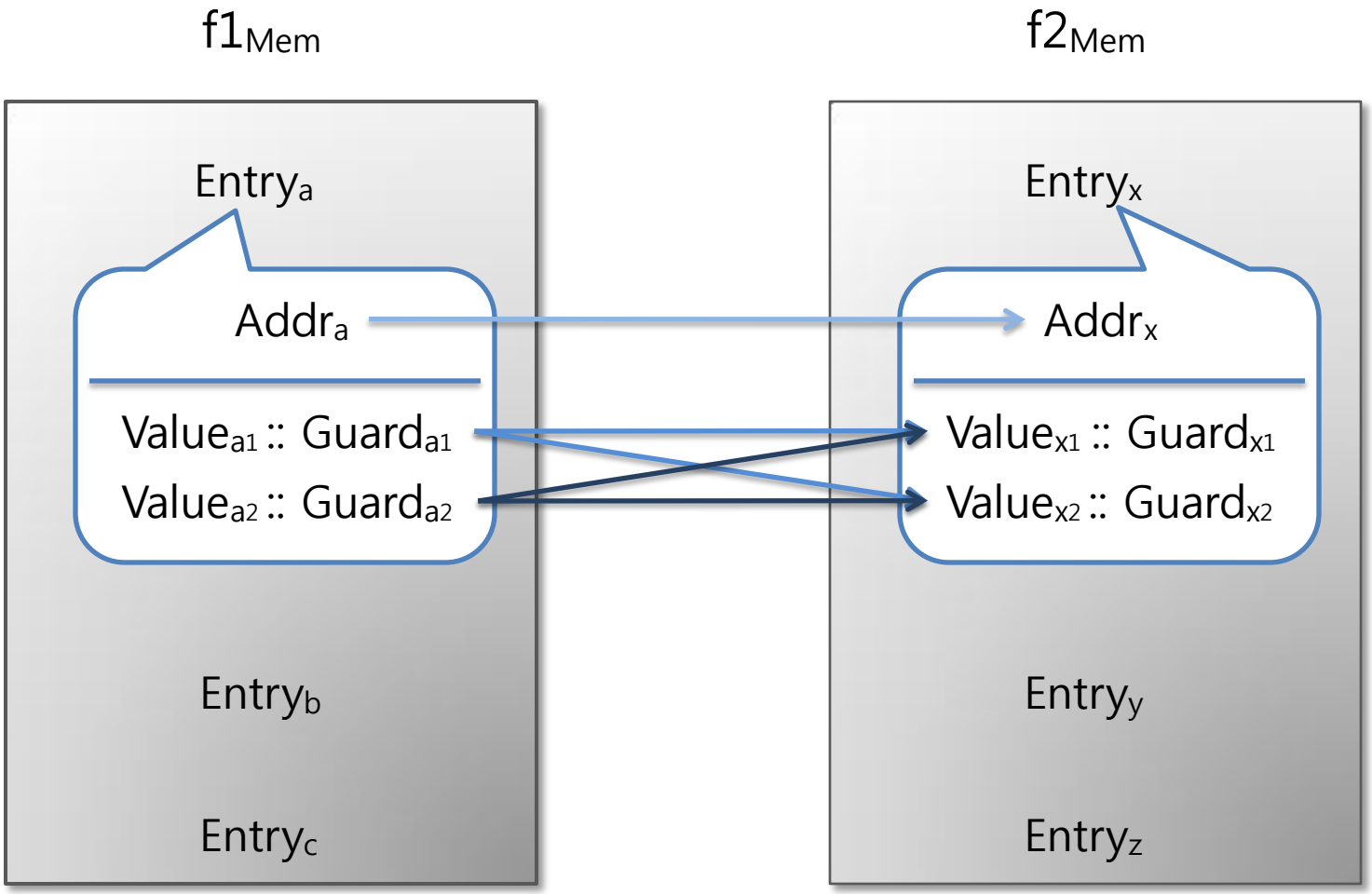
Comparison Algorithm

- **Function f1 vs Function f2**



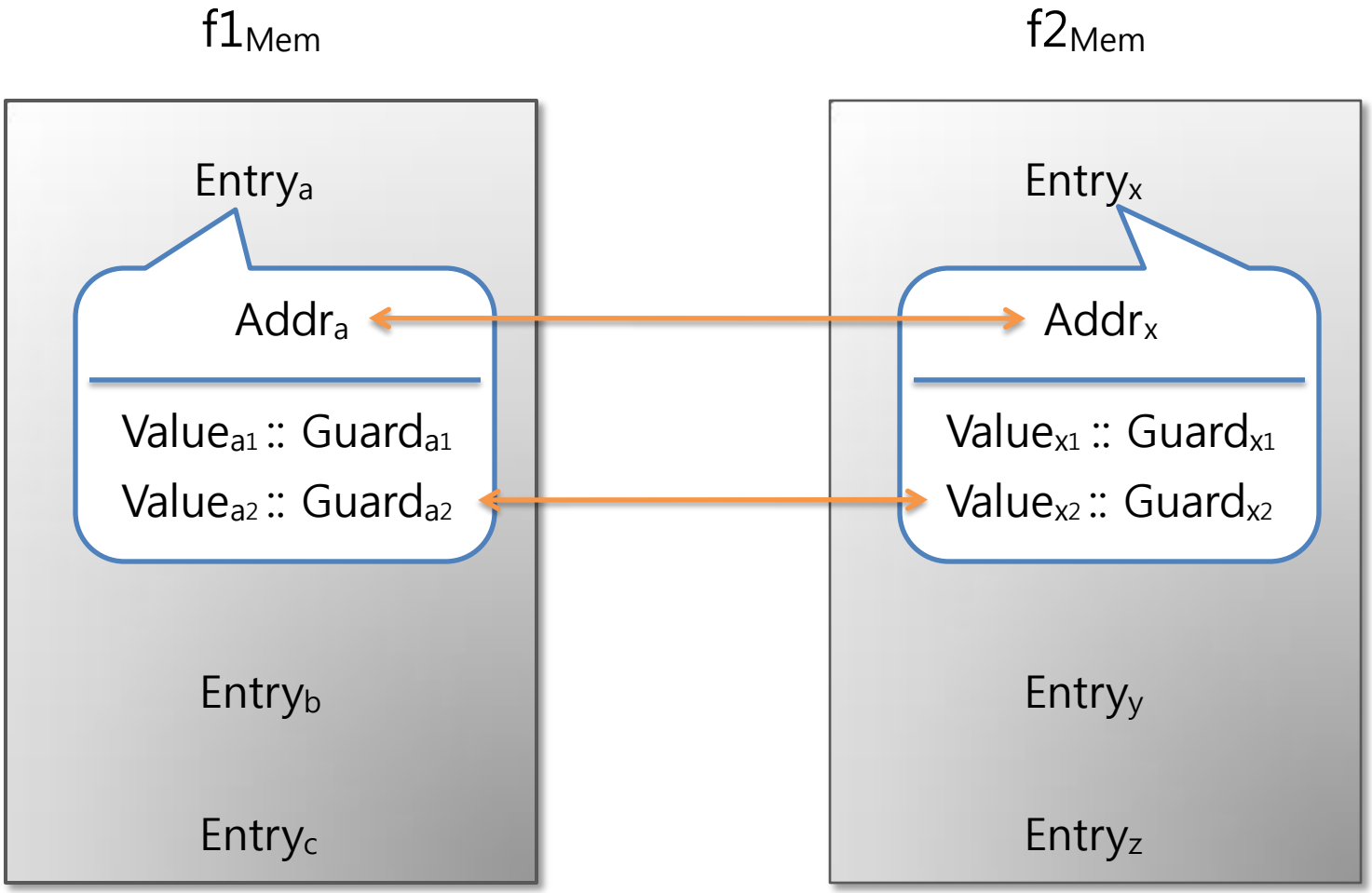
Comparison Algorithm

- **Function f1 vs Function f2**



Comparison Algorithm

- **Use matched point**



Comparison Algorithm

Similarity Measurement

$$S(x,n) = \left(\frac{x}{n}\right)^{\frac{c}{\log n}} \times 100$$

- n = total number of entries
- x = 2**maximize*(sum of matched points)
- c = proper constant (eg. log 50)

- $f1_{mem}$ = { $Entry_a$, $Entry_b$, $Entry_c$ }
- $f2_{mem}$ = { $Entry_x$, $Entry_y$, $Entry_z$ }

	$Entry_a$	$Entry_b$	$Entry_c$
$Entry_x$	0.5	0.1	0.1
$Entry_y$	0.0	0.2	0.9
$Entry_z$	0.2	0.6	0.4

Preliminary Experiment Results

- **Input Source : Python-2.5.1**
  - Size : 435 KLoc, 14836 Kbyte
  - Total functions : 11663

- **For performance comparison**
  - Use Deckard [Jiang et al. ICSE'07]
  - AST(Abstract Syntax Tree)-based clone detection tool
  - Known to be accurate for syntactic clone
  - Meaningful clone/Total output : 60/74

Preliminary Experiment Results

Execution Time & Found clones

- Clone detecting time/Total analysis time : 360/3808 (s)
- Total output (Similarity ≥ 80%) : 258
- Random sampling
  - 90% ≤ Similarity ≤ 100%
    : 30 samples are all clones
    : most of them are syntactic clones
  - 80% ≤ Similarity < 90%
    : 3 samples out of 30 are false positive
    : syntactic & semantic clones are detected

Preliminary Experiment Results

Found clone

```
1  static int
2  _siftdown(PyListObject *heap, Py_ssize_t startpos, Py_ssize_t pos)
3  {
4          PyObject *newitem, *parent;
5          int cmp;
6          Py_ssize_t parentpos;
7
8          assert(PyList_Check(heap));
9          if (pos >= PyList_GET_SIZE(heap)) {
10                 PyErr_SetString(PyExc_IndexError, "index out of range");
11                 return -1;
12         }
13
14         newitem = PyList_GET_ITEM(heap, pos);
15         Py_INCREF(newitem);
16         /* Follow the path to the root, moving parents down until finding
17            a place newitem fits. */
18         while (pos > startpos){
19                 parentpos = (pos - 1) >> 1;
20                 parent = PyList_GET_ITEM(heap, parentpos);
21                 cmp = PyObject_RichCompareBool(parent, newitem, Py_LE);
22                 if (cmp == -1) {
23                         Py_DECREF(newitem);
24                         return -1;
25                 }
26                 if (cmp == 1)
27                         break;
28                 Py_INCREF(parent);
29                 Py_DECREF(PyList_GET_ITEM(heap, pos));
30                 PyList_SET_ITEM(heap, pos, parent);
31                 pos = parentpos;
32         }
33         Py_DECREF(PyList_GET_ITEM(heap, pos));
34         PyList_SET_ITEM(heap, pos, newitem);
35         return 0;
36 }
```

```
1  static int
2  _siftdownmax(PyListObject *heap, Py_ssize_t startpos, Py_ssize_t pos)
3  {
4          PyObject *newitem, *parent;
5          int cmp;
6          Py_ssize_t parentpos;
7
8          assert(PyList_Check(heap));
9          if (pos >= PyList_GET_SIZE(heap)) {
10                 PyErr_SetString(PyExc_IndexError, "index out of range");
11                 return -1;
12         }
13
14         newitem = PyList_GET_ITEM(heap, pos);
15         Py_INCREF(newitem);
16         /* Follow the path to the root, moving parents down until finding
17            a place newitem fits. */
18         while (pos > startpos){
19                 parentpos = (pos - 1) >> 1;
20                 parent = PyList_GET_ITEM(heap, parentpos);
21                 cmp = PyObject_RichCompareBool(newitem, parent, Py_LE);
22                 if (cmp == -1) {
23                         Py_DECREF(newitem);
24                         return -1;
25                 }
26                 if (cmp == 1)
27                         break;
28                 Py_INCREF(parent);
29                 Py_DECREF(PyList_GET_ITEM(heap, pos));
30                 PyList_SET_ITEM(heap, pos, parent);
31                 pos = parentpos;
32         }
33         Py_DECREF(PyList_GET_ITEM(heap, pos));
34         PyList_SET_ITEM(heap, pos, newitem);
35         return 0;
36 }
```

Preliminary Experiment Results

- **Found clone**

```
1  static PyObject *
2  grp_getgrall(PyObject *self, PyObject *ignore)
3  {
4      PyObject *d;
5      struct group *p;
6      if ((d = PyList_New(0)) == NULL)
7          return NULL;
8      setgrent();
9      while ((p = getgrent()) != NULL) {
10         PyObject *v = mkgrent(p);
11         if (v == NULL || PyList_Append(d, v) != 0) {
12             Py_XDECREF(v);
13             Py_DECREF(d);
14             return NULL;
15         }
16         Py_DECREF(v);
17     }
18     endgrent();
19     return d;
20 }
```

```
1  static PyObject *
2  spwd_getspall(PyObject *self, PyObject *args)
3  {
4          PyObject *d;
5          struct spwd *p;
6          if ((d = PyList_New(0)) == NULL)
7                  return NULL;
8          setspent();
9          while ((p = getspent()) != NULL) {
10                 PyObject *v = mkspent(p);
11                 if (v == NULL || PyList_Append(d, v) != 0) {
12                         Py_XDECREF(v);
13                         Py_DECREF(d);
14                         endspent();
15                         return NULL;
16                 }
17                 Py_DECREF(v);
18         }
19         endspent();
20         return d;
21 }
```

Preliminary Experiment Results

Found clone

```
1  void addarc(dfa *d, int from, int to, int lbl)
2  {
3          state *s;
4          arc *a;
5
6          assert(0 <= from && from < d->d_nstates);
7          assert(0 <= to && to < d->d_nstates);
8
9          s = &d->d_state[from];
10         s->s_arc = (arc *)PyObject_REALLOC(s->s_arc, sizeof(arc) * (s->s_narcs + 1));
11         if (s->s_arc == NULL)
12                 Py_FatalError("no mem to resize arc list in addarc");
13         a = &s->s_arc[s->s_narcs++];
14         a->a_lbl = lbl;
15         a->a_arrow = to;
16 }
```

```
1  void addnfaarc(nfa *nf, int from, int to, int lbl)
2  {
3          nfastate *st;
4          nfaarc *ar;
5
6
7          st = &nf->nf_state[from];
8          st->st_arc = (nfaarc *)PyObject_REALLOC(st->st_arc, sizeof(nfaarc) * (st->st_narcs + 1));
9          if (st->st_arc == NULL)
10                 Py_FatalError("out of mem");
11         ar = &st->st_arc[st->st_narcs++];
12         ar->ar_label = lbl;
13         ar->ar_arrow = to;
14 }
```

Q&A

Thank you!