ACM 1998 Workshop on
Java for High-Performance Network Computing

CONVERTING C POINTERS TO JAVA REFERENCES

Erik Demaine

Department of Computer Science
University of Waterloo, Waterloo, Ontario, Canada
eddemaine@uwaterloo.ca

http://daisy.uwaterloo.ca/~eddemain/papers/Java98

February 28, 1998

OUTLINE

- Motivation
  - C pointers versus references
- Related work
- Our work
  - General conversion
  - Various improvements
- Conclusion

MOTIVATION

- C is popular
  - Lots of code written in it
- Developers want to extend their programs
  - Exploit features of new programming languages
  - Need to convert code to the language
- Example:
  - Port program to Java for demonstration on WWW
- C and Java have similar syntax
- Main difficulty: converting pointers into references

C POINTERS

- C pointers are overly flexible
- Can represent
  - Basic value
  - Array
  - Structure
  - Element of array or structure
  - Byte
- Can be
  - Cast into integers or pointers of other types
  - Split into pieces and later rejoined
  - Offset by arbitrary amounts (pointer arithmetic)
  - Compared for inequality
- Unnecessary flexible
  - Dangerous
  - Difficult to debug

REFERENCES

- Examples:
  - Java
  - Modula, Pascal
  - Perl 5
  - Fortran 90
  - ML
- Limited access to references themselves
  - Assign to a reference
  - Compare references for equality
- Often no dereference operation
  - Use of reference is implicitly about target
- Safer and simpler

RELATED WORK

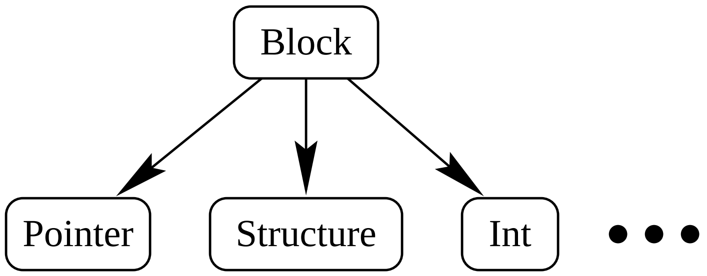
- "C++ to Java"
  - c2j (1996): written in C++
  - c2j++ (1997): written in Java
  - Ignore pointers: remove all dereferences
  - Often correct for objects and pointers to objects
  - Never correct for arrays

OUR WORK

- Specifically on converting pointers to references
  - Focus: Scientific applications
  - Example: SuperLU (1997): 15,000 lines of C
- Start with general conversion
- Optimize common cases
  - Crucial to implement arrays efficiently
  - Goal: Convert C arrays into Java arrays
  - Similarly, convert C structures into Java objects

GENERAL CONVERSION

- Could build C runtime environment at user level
  - Create duplicate representation of heap
  - Hence understand pointers to objects within heap



- Can navigate through heap
  - Can do pointer arithmetic
  - Supports untyped pointers
- Cannot use binary representations of values
  - No union types
  - Cannot cast between integers and pointers

EXAMPLE OF GENERAL CONVERSION

```
main () {
  int a[4] = {1,2,3,0};
  printf ("%d\n", sum (a));
}
int sum (int *a) {
  int s = 0;
  while (*a != 0)
    s += *(a++);
  return s;
}
```

```
main () {
  Int[] a_ = {new Int (1), new Int (2),
    new Int (3), new Int (0)};
  Struct a = new Struct (a_);
  System.out.println (sum (new Pointer (a)));
}
int sum (Pointer a) {
  int s = 0;
  while (((Int) a.target).val != 0) {
    s += ((Int) a.target).val;
    a.increment ();
  }
  return s;
}
```

EXAMPLE OF IMPROVED CONVERSION

```
main () {
  int a[4] = {1,2,3,0};
  printf ("%d\n", sum (a));
}
int sum (int *a) {
  int s = 0;
  while (*a != 0)
    s += *(a++);
  return s;
}
```

```
main () {
  int[] a = {1,2,3,0};
  System.out.println (sum (a));
}
int sum (int[] a) {
  int a_i = 0;
  int s = 0;
  while (a[a_i] != 0)
    s += a[a_i++];
  return s;
}
```

OPTIMIZATIONS

- **Indexable arrays**: most general pointer representation
  - Common to use C pointer as "alias" to element of array
  - Convert C pointer into Java array plus integer index

C		Java
`int a[5];`	$\mapsto$	`int[] a = new int[5];`
`int *p = a;`	$\mapsto$	`int[] p = a;`
		`int p_i = 0;`
`p++;`	$\mapsto$	`p_i++;`
`*p = 5;`	$\mapsto$	`p[p_i] = 5;`
`p[2] = 10;`	$\mapsto$	`p[p_i+2] = 10;`
`f (p);`	$\mapsto$	`f (p, p_i);`
`f (a);`	$\mapsto$	`f (a, 0);`

- **Arrays**: if index is always zero, discard it
  - Maps C arrays to Java arrays

- **Call by reference**: pointer always to basic value
  - Either
    - ☐ Array of size one
    - ☐ Object with one entry
  - Typically objects more efficient
  - May vary in the future

STATIC ANALYSIS

Pointer	Desired conversion
To element of an array	Array plus index
Representing an array	Array
To a basic value	Object (or possibly 1-element array)

- Statically analyze code to determine best guarantees on kinds of pointers
- Major unknown is pointers passed to functions

FUNCTION COPYING

- At call side, should have better "bound" on kind of pointer
- For each type of call (mix of pointer kinds), duplicate function
  - Optimize each function copy by exploiting known bounds on pointer kinds

```
void f (int *i) {
  *i = 5;
}
```

⟶

```
void f (Int i) {
  i.val = 5;
}

void f (int[] i) {
  i[0] = 5;
}

void f (int[] i, int i_i) {
  i[i_i] = 5;
}
```

C		Java
`int i;`	↦	`Int i;`
`f (&i);`	↦	`f (i);`
`int a[5];`	↦	`int[] a = new int[5];`
`f (a);`	↦	`f (a);`
`int *p = &a[2];`	↦	`int[] p = a;`
		`int p_i = 2;`
`f (p);`	↦	`f (p, p_i);`

- Worst case: Quadratic code expansion

CONCLUSION

- Can achieve complete and automatic C-to-Java conversion
  - Provides a general framework
- Optimizations improve "common" cases
  - Arrays $\rightarrow$ arrays
  - Structures $\rightarrow$ objects
  - Occasional "bad" case should not affect other cases
- The better written the code, the better the conversion
- Similar ideas apply to any pointer-to-reference conversion
  - For example, C to Fortran 90
- Future work:
  - Implement C-to-Java converter
  - Extend to C++ as source language