SAS'97

COALESCING CONDITIONAL BRANCHES INTO EFFICIENT INDIRECT JUMPS

Gang-Ryung Uh **David B. Whalley**

Computer Science Department

Florida State University

Tallahassee, FL 32306-4019
USA

1. Motivation
2. Detecting Sequence of Branches to Coalesce
3. Constructing the Jump Table
4. Efficiently Performing the Indirect Jump
5. Estimating the Benefits
6. Transforming the Control Flow
7. Results
8. Future Work
9. Conclusions

Traditional Indirect Jump Transformation

- For a multiway selection statement, a compiler front end determines whether or not to produce the indirect jump operation as intermediate code.

Two Disadvantages

1. When the decision is made, the underlying hardware information is not available.
2. Other high level control statements cannot be exploited by the indirect jump transformation.

Original

```
for (sp = line; *sp; sp++) {
   switch (*sp) {
   case 'p':  ...

   case 'k': ...
   ...
  }
}
```

Restructured

```
for (sp = line;  ; sp++) {
   switch (*sp) {
   case '\0':   goto out;

   case 'p':

   case 'k':
   ...
}
out:
```

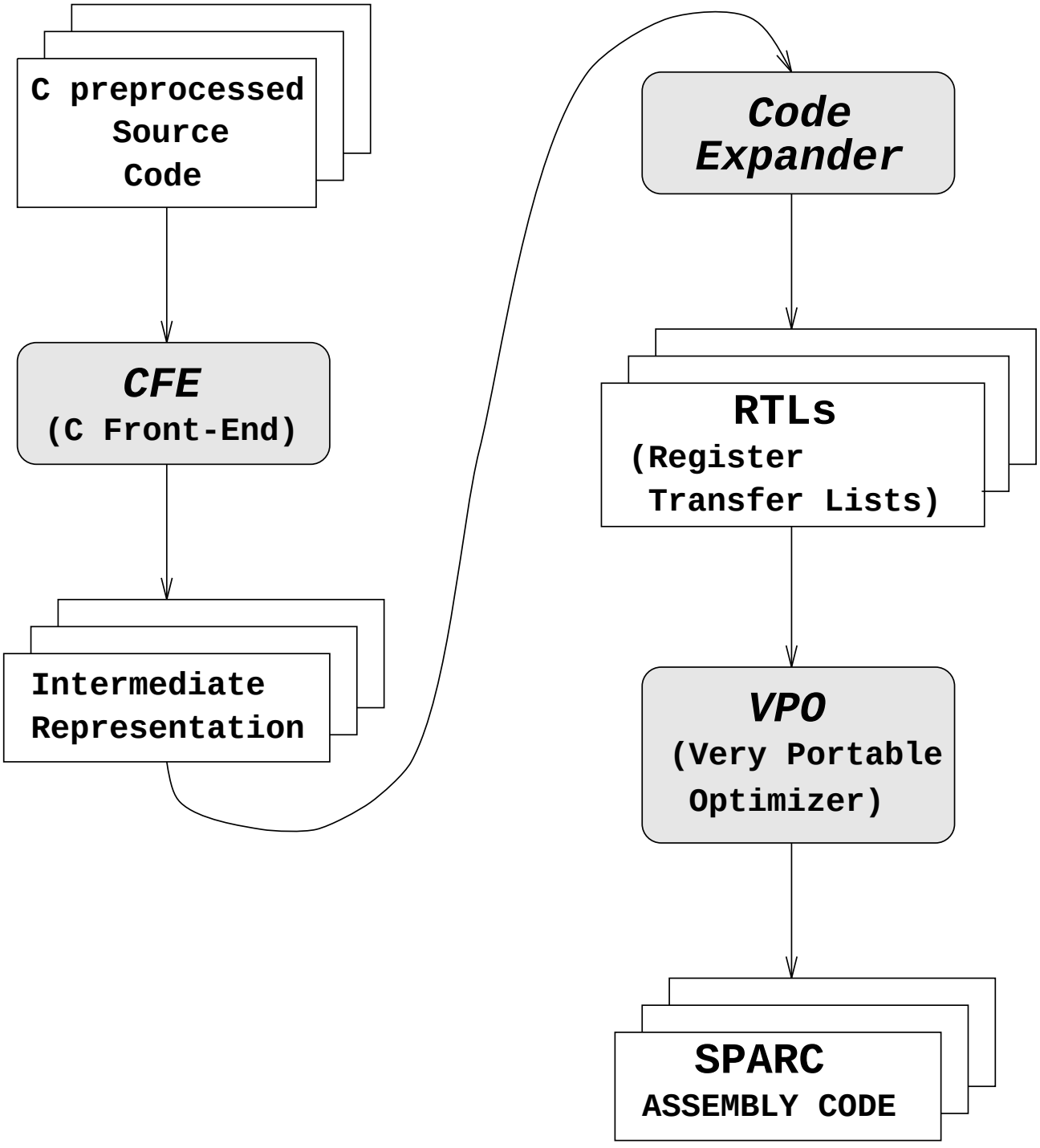
Original

```
if ((c = *sp++) == 0)
    goto cerror;
if (c == '<') { ... }
if (c == '>') { ... }
if (c == '(') { ... }
if (c == ')') { ... }
if (c >= '1' && c <= '9') { ... }
…
```

Restructured

```
c = *sp++;
switch (c) {
case 0:      goto cerror;

case '<':       ...

case '>':       ...

case '(':       ...

case ')':       ...

case '1': case '2': case '3':
case '4': case '5': case '6':
case '7': case '8': case '9':
                ...
default:        ...
}
```

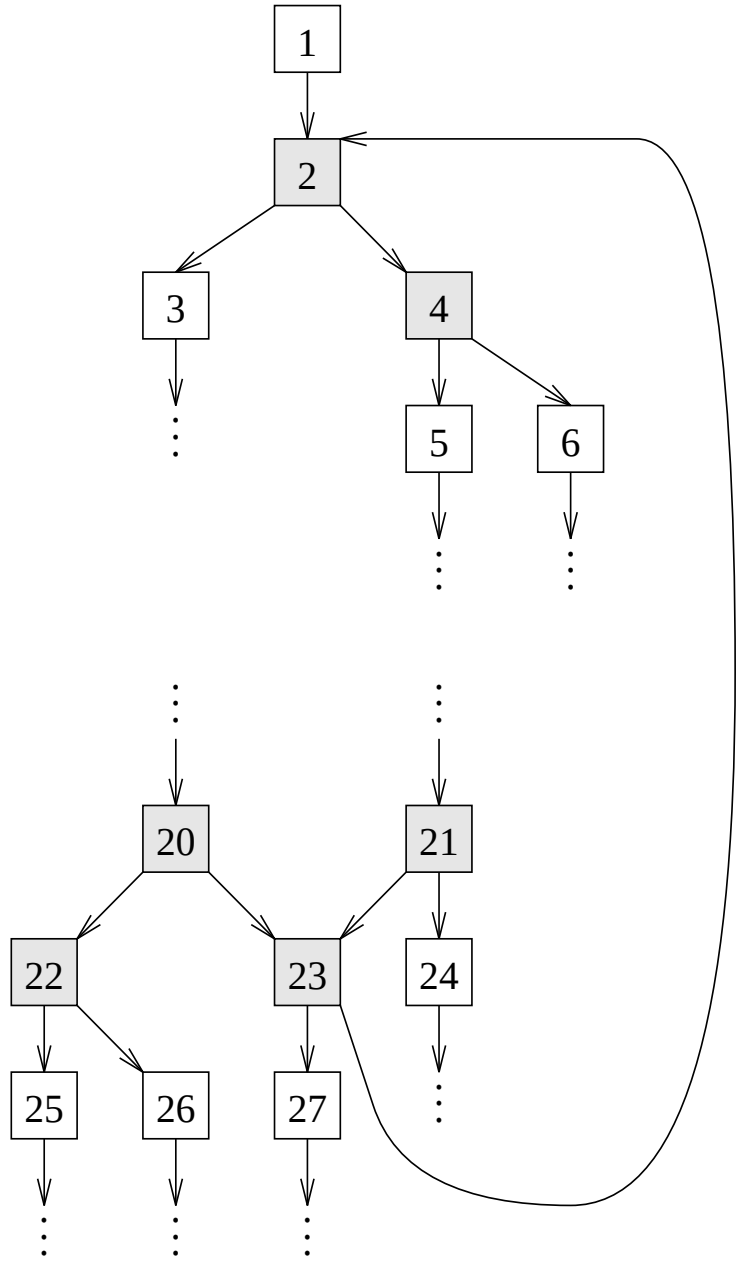
Compiler Organization

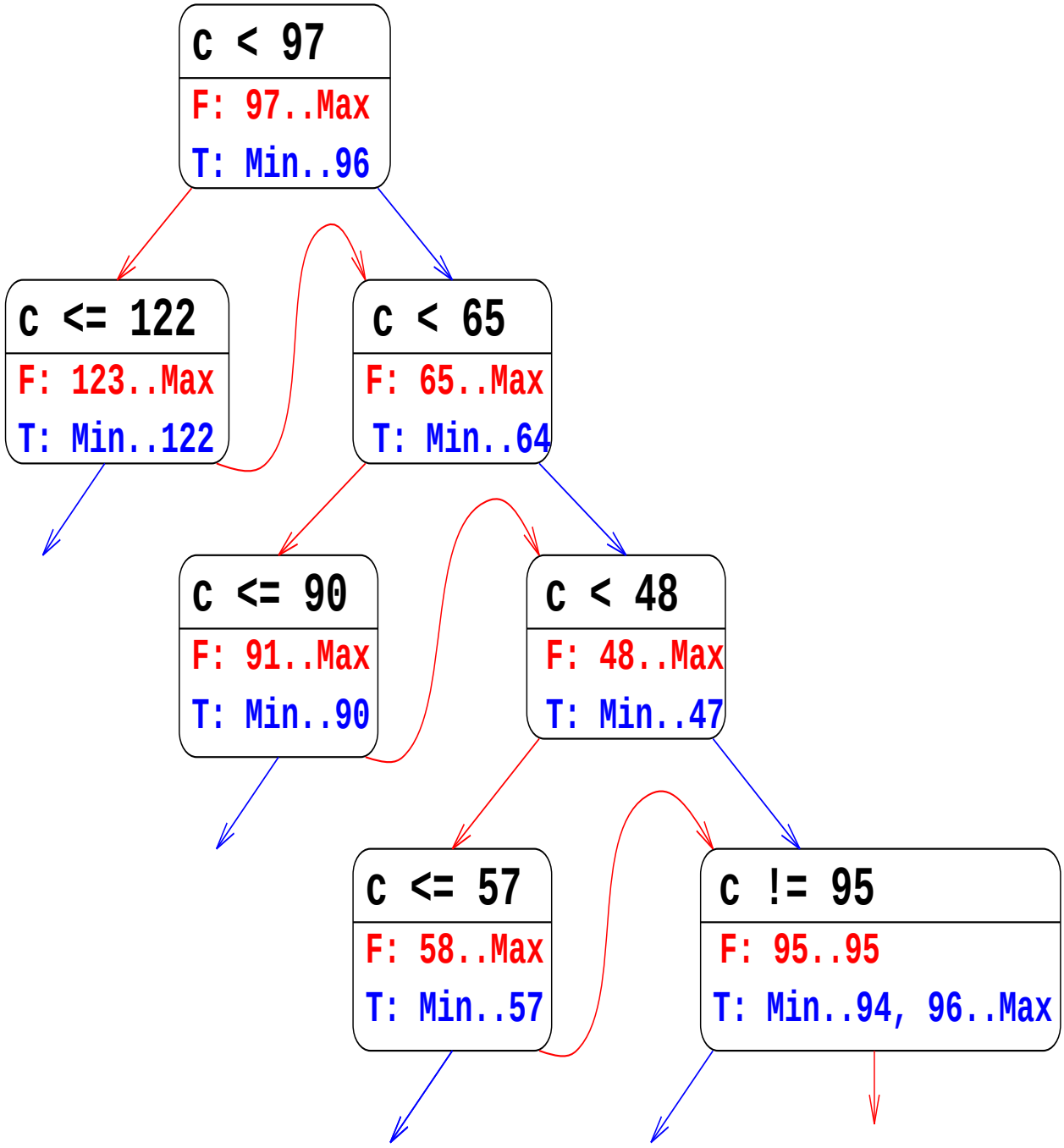


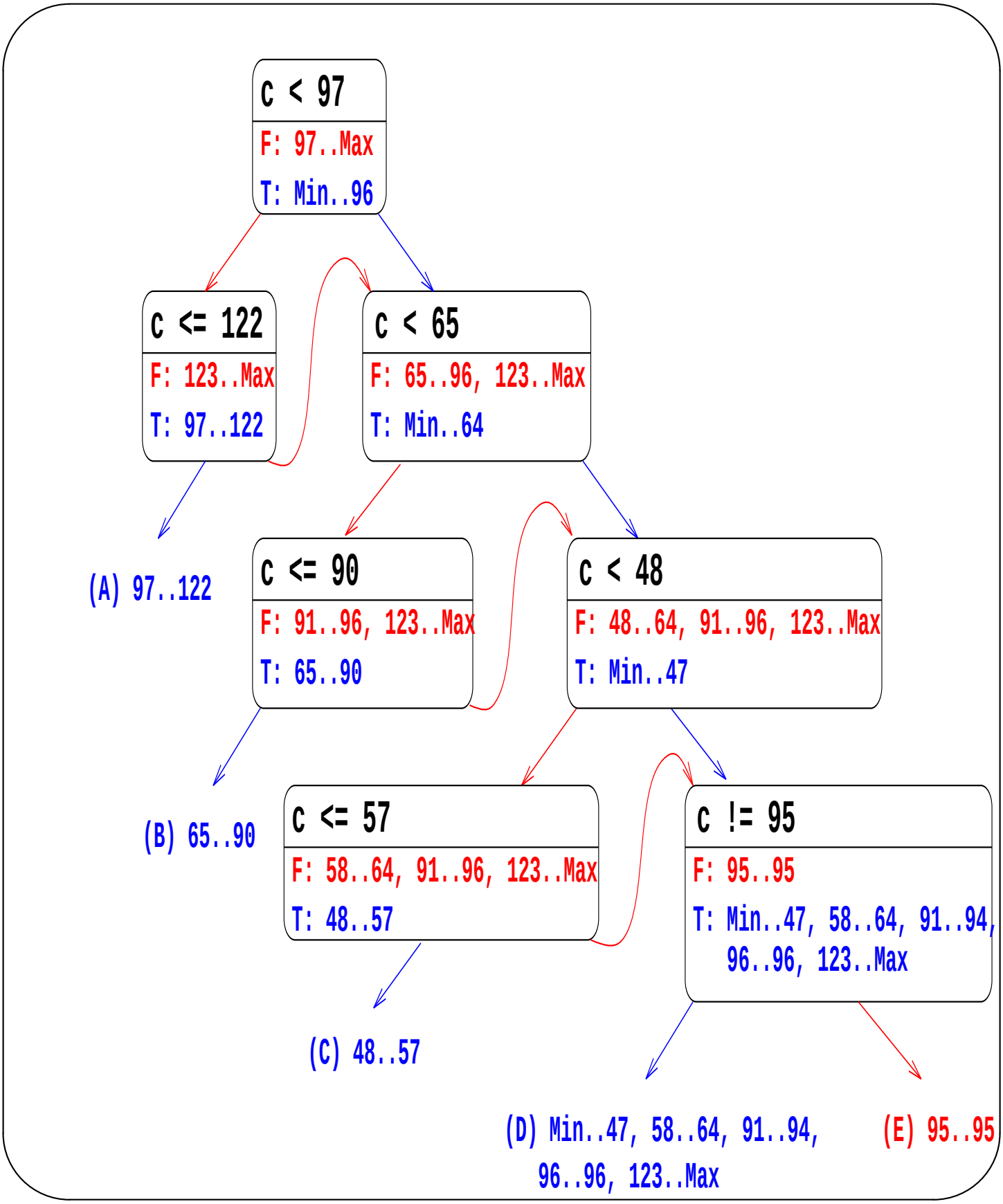
VPO

```
Branch Chaining
...
Instruction Selection
DO {
  Register Allocation
  ...
  Loop Optimizations:
    Code Motion
    Recurrences
    Loop Strength Reduction
    Induction Variable Elimination
    If (First Pass)
      BRANCH COALESCING
  Useless Jump Elimination
  ...
  Instruction Selection
} WHILE (change)
BRANCH COALESCING
...
```

Detect a Contiguous Sequence of Conditional Branches That Can Be Potentially Coalesced

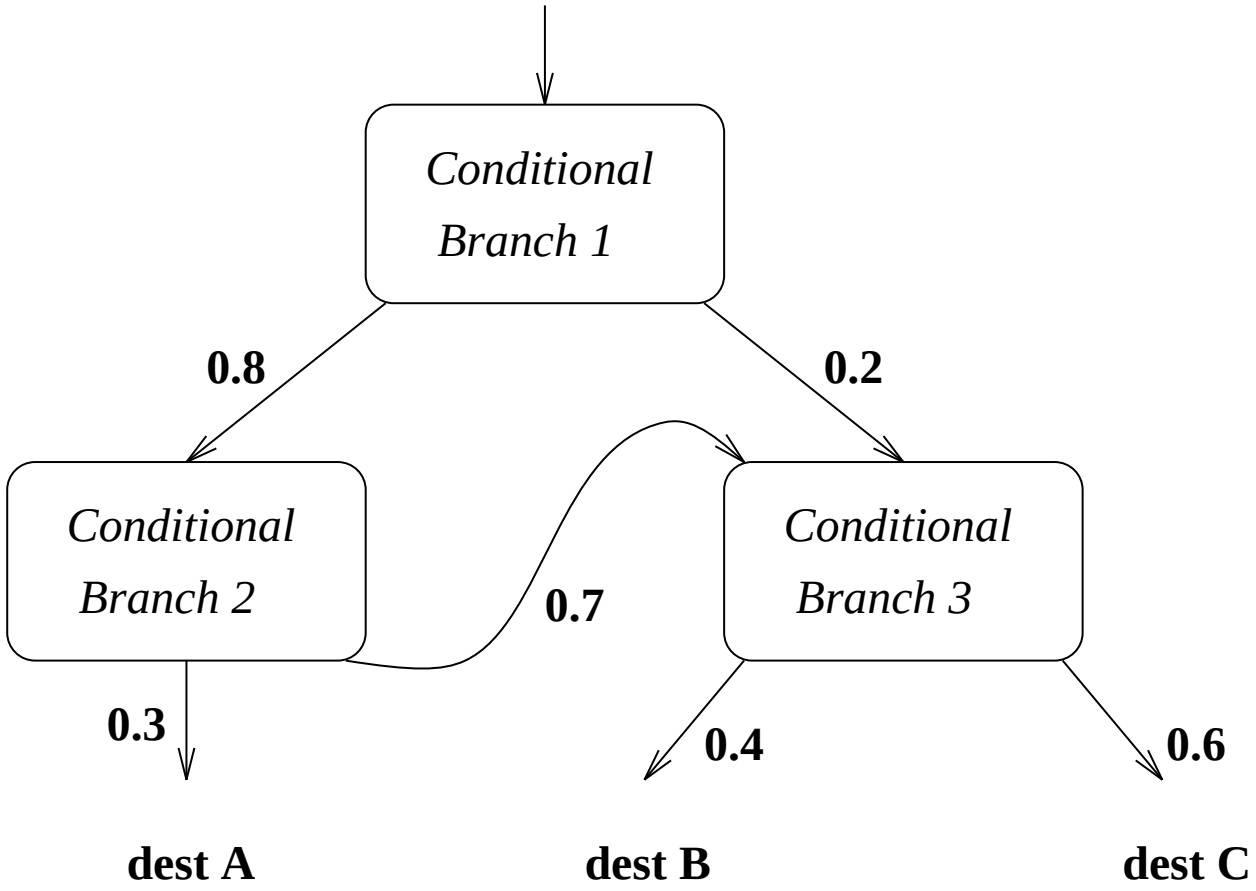



c < 97
F: 97..Max
T: Min..96
c <= 122
F: 123..Max
T: Min..122
c < 65
F: 65..Max
T: Min..64
c <= 90
F: 91..Max
T: Min..90
c < 48
F: 48..Max
T: Min..47
c <= 57
F: 58..Max
T: Min..57
c != 95
F: 95..95
T: Min..94, 96..Max


c < 97
F: 97..Max
T: Min..96
c <= 122
F: 123..Max
T: 97..122
c < 65
F: 65..96, 123..Max
T: Min..64
(A) 97..122
c <= 90
F: 91..96, 123..Max
T: 65..90
c < 48
F: 48..64, 91..96, 123..Max
T: Min..47
(B) 65..90
c <= 57
F: 58..64, 91..96, 123..Max
T: 48..57
c != 95
F: 95..95
T: Min..47, 58..64, 91..94, 96..96, 123..Max
(C) 48..57
(D) Min..47, 58..64, 91..94, 96..96, 123..Max
(E) 95..95

Cost of Branches

$$Average\_Cost = \frac{2 * \sum_{k=1}^{K} (N_{P_k} * W_{P_k})}{K}$$



Traditional Approach for Performing the Indirect Jump Operation

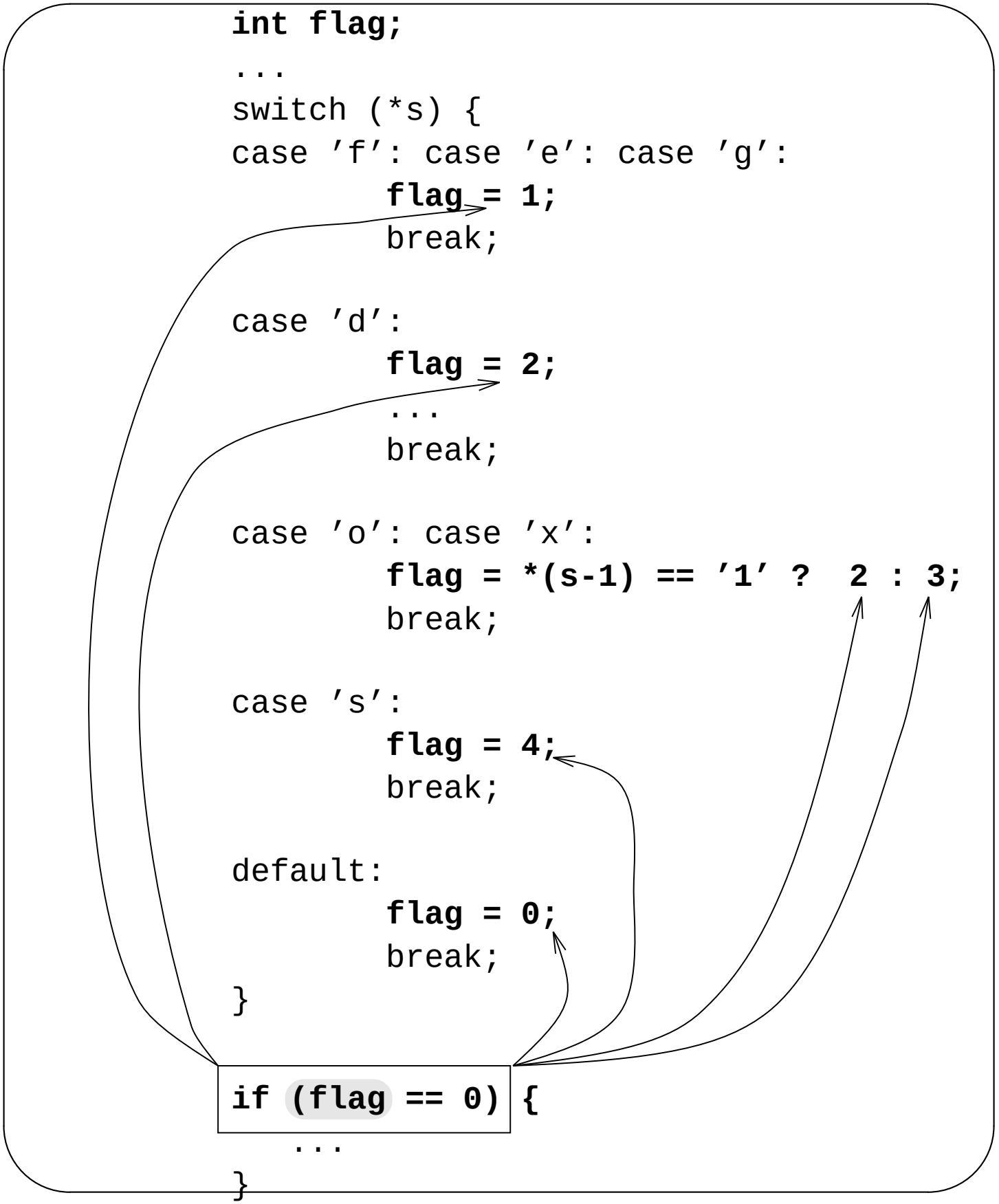
```
r[8]=r[10]-32;
IC=r[8]?45;
PC=ICh,L18;
r[20]=HI[L01];
r[20]=r[20]|LO[L01];
r[8]=r[8]<<2;
r[8]=M[r[8]+r[20]];
PC=r[8];

L01:
.word  L27   # Lowest case value 32
...
.word  L24   # Highest case value 77
```

Padding the Front of the Jump Table

```
IC=r[8]?77;
PC=ICh0,L18;
r[8]=r[8]<<2;
r[8]=M[r[8]+r[20]];
PC=r[8];

L01:
.word  L18    # Target for 0
.word  L18    # Target for 1
...           ...
.word  L18    # Target for 31
.word  L27    # Target for 32
...           ...
.word  L24    # Target for 77
```

Avoiding Initial Range Check

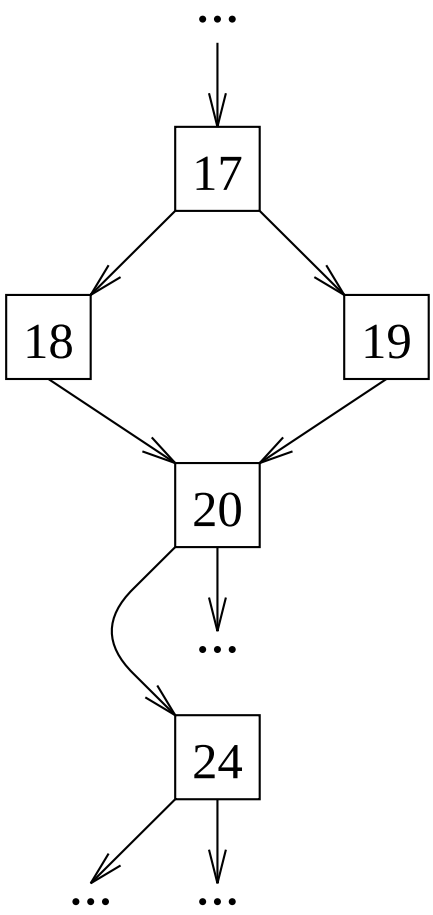
```
r[8]=r[8]<<2;
r[8]=M[r[8]+r[20]];
PC=r[8];

L01:
.word  L18    # Target for 0
.word  L18    # Target for 1
...           ...
.word  L18    # Target for 31
.word  L27    # Target for 32
...           ...
.word  L24    # Target for 77
.word  L18    # Target for 78
...
.word  L18    # Target for 255
```


int flag;
...
switch (*s) {
case 'f': case 'e': case 'g':
flag = 1;
break;
case 'd':
flag = 2;
...
break;
case 'o': case 'x':
flag = *(s-1) == '1' ? 2 : 3;
break;
case 's':
flag = 4;
break;
default:
flag = 0;
break;
}
if (flag == 0) {
...
}
...

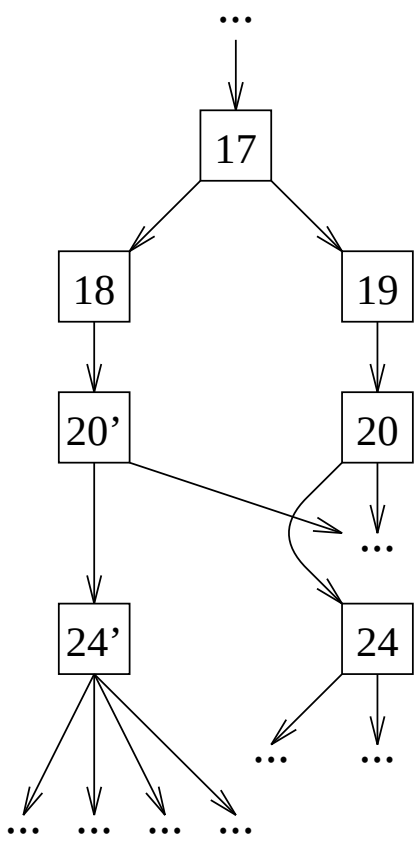
(a) RTLs in Original Blocks

```
...
17. ...
18. ...
    r[8]=B[r[9]]&255;
    ...
19. ...
    CALL _filbuf();
    ...
20. r[10]=r[8];
    ...
...
24. ...
    IC=r[10]?32;
    PC=IC<=0,L66;
...
```

(b) Original Flow



(c) After Replication



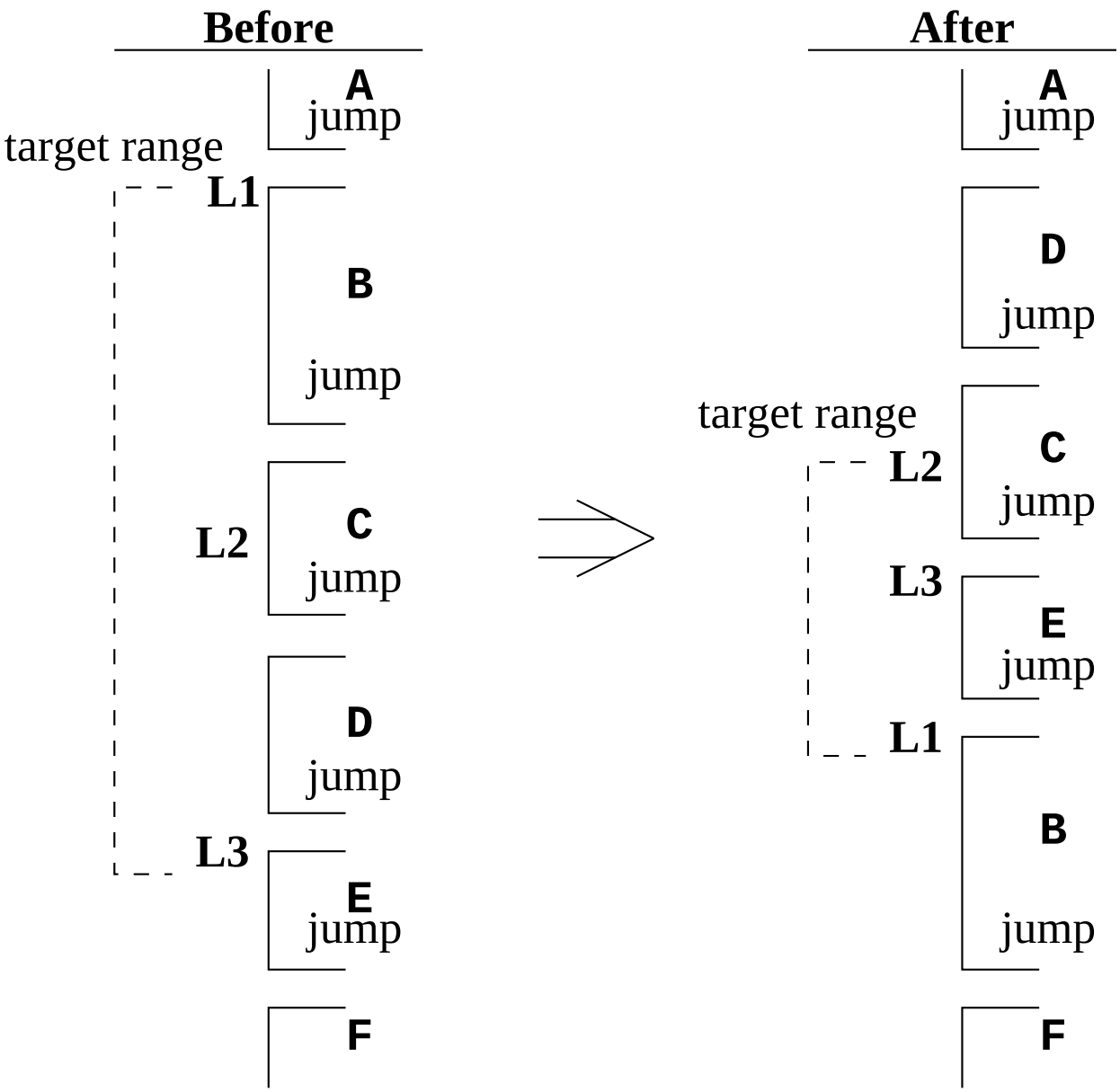
(c) RTLs

```
20'. r[10]=r[8];
     ...
24'. ...
     r[10]=r[10]<<2;
     r[10]=M[r[10]+r[20]];
     PC=r[10];
```

Byte Displacements

```
# r[20] :  jump table address (L01)
# r[22] :  mid-point address  (L02)

r[8]=M[r[8]+r[20]];
PC=r[8]+r[22];
.seg  ‘‘data’’
L01:
.byte  L18-L02
.byte  L18-L02
...
.byte  L18-L02
.byte  L27-L02
...
.byte  L24-L02
.byte  L18-L02
...
.byte  L18-L02
.align 4
.seg  ‘‘text’’
```

Compacting Target Ranges



Dynamic Instructions

Program	None	Original	Coalescing
awk	13,666,952	−0.294%	**−2.145%**
cb	19,739,127	−12.976%	**−20.613%**
cpp	30,985,306	−37.421%	**−37.960%**
ctags	74,316,425	−0.536%	**−10.974%**
deroff	15,513,244	−0.195%	**−1.028%**
grep	11,810,070	−21.620%	**−24.370%**
hyphen	19,535,372	0.000%	**−0.783%**
join	3,552,801	0.000%	**0.102%**
lex	10,052,031	−0.230%	**−0.566%**
nroff	25,118,855	−0.155%	**−0.015%**
pr	78,106,755	0.000%	**−7.801%**
ptx	22,679,653	0.000%	**−7.891%**
sdiff	17,582,760	0.000%	**0.022%**
sed	17,872,507	−6.375%	**−6.629%**
sort	18,921,766	0.000%	**−31.298%**
wc	17,860,086	0.000%	**-17.853%**
yacc	25,658,688	−0.194%	**−0.303%**
average	24,880,729	−4.707%	**−10.006%**

Future Work

- Interprocedural Analysis
- Profiling

Conclusion

- Feasible
- Fewer Dynamic Instructions
- Less Cache Work