Improving Performance by Branch Reordering

by

Minghui Yang and David Whalley
Florida State University

and

Gang-Ryung Uh
Lucent Technologies

Outline of Presentation

- Motivation
- Detecting a Reorderable Sequence
- Selecting the Sequence Ordering
- Applying the Transformation
- Results
- Future Work

Example Sequence of Comparisons with the Same Variable

```
while ((c=getchar())
        != EOF)
  if (c == '\n')
      X;
  else if (c == ' ')
      Y;
  else
      Z;
```

(a) Original Code Segment

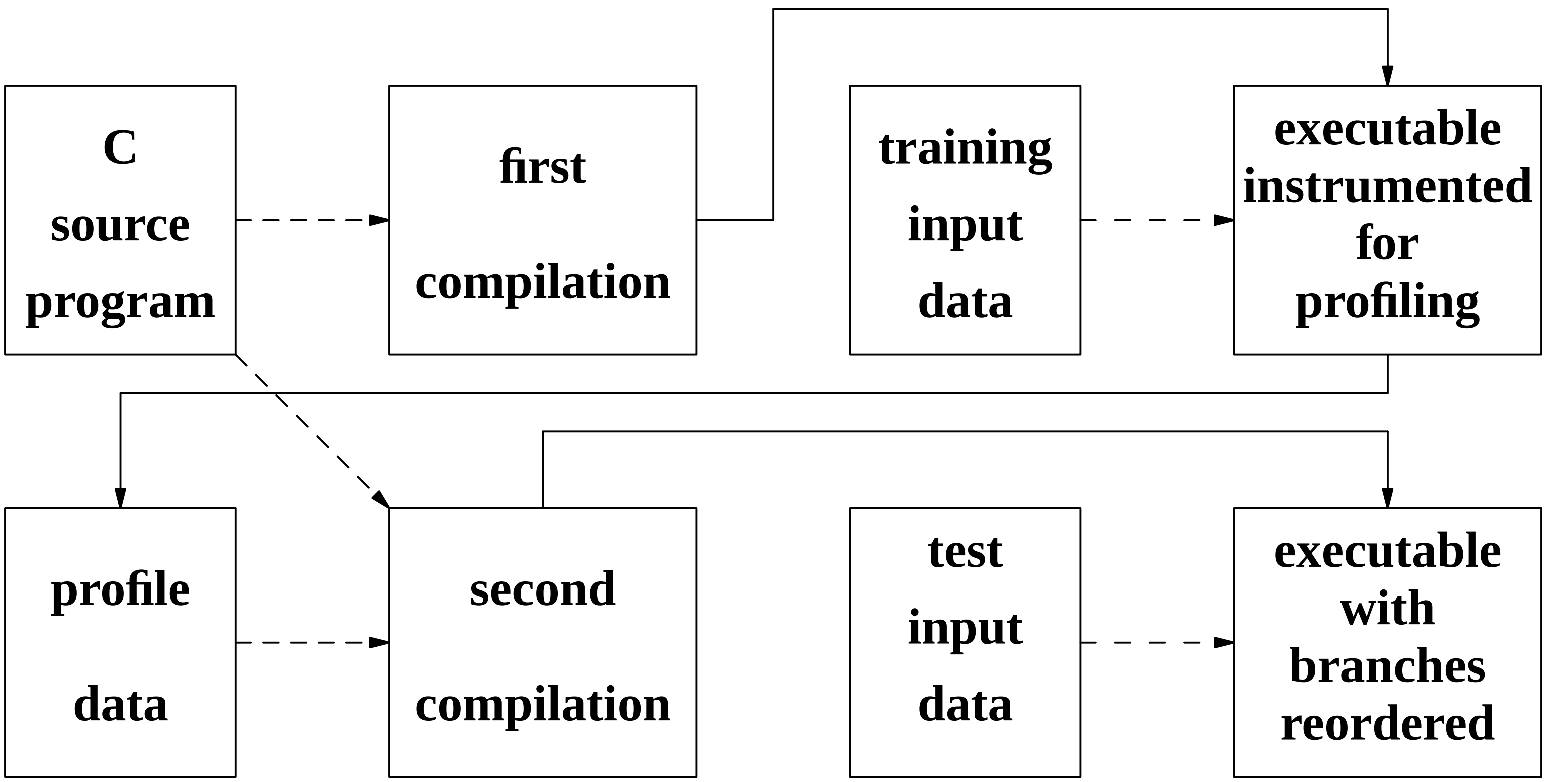
```
while (1) {
  c = getchar();
  if (c == ' ')
      Y;
  else if (c == '\n')
      X;
  else if (c == EOF)
      break;
  else
      Z;
}
```

(b) Conventional Reordering

```
while (1) {
  c = getchar();
  if (c > ' ')
      goto def;
  else if (c == ' ')
      Y;
  else if (c == '\n')
      X;
  else if (c == EOF)
      break;
  else
def:  Z;
}
```

(c) Improved Reordering

Overview of Compilation Process for Branch Reordering



Ranges and Corresponding Range Conditions

Form	Range	Range Condition
1	c..c	v == c
2	MIN..c	v <= c
3	c..MAX	v >= c
4	c1..c2	c1 <= v && v <= c2

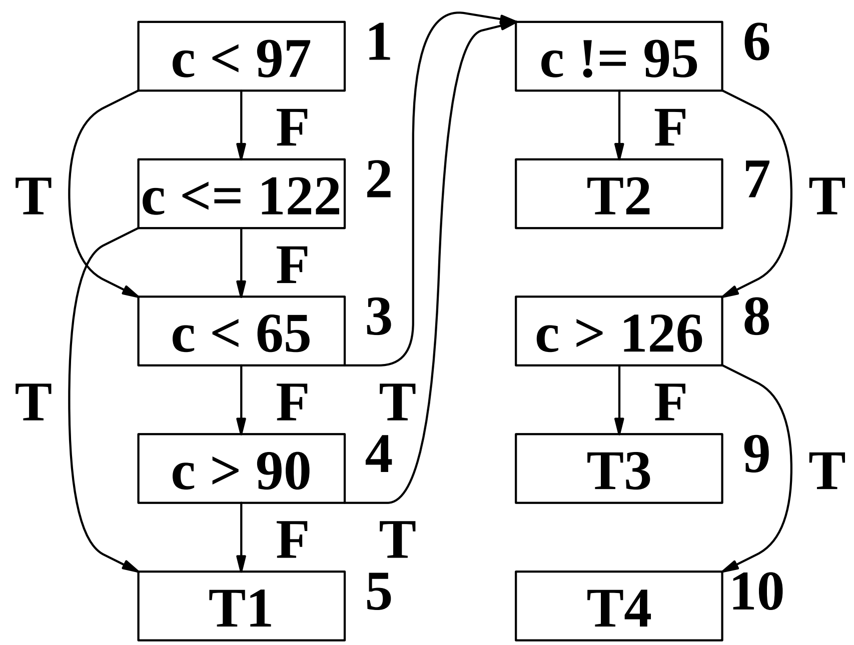
Requirements for a Sequence to Be Reorderable

- All the ranges in the sequence are nonoverlapping.
- The sequence can only be entered through the first range condition.
- The sequence has no side effects.
- Each range condition can only contain comparisons and branches.

Example of Detecting Range Conditions

```
if (c>='a' && c<='z' ||
     c>='A' && c<='Z')
   T1;
else if (c=='_')
   T2;
else if (c<='~')
   T3;
else
   T4;
```

(a) C Code Segment



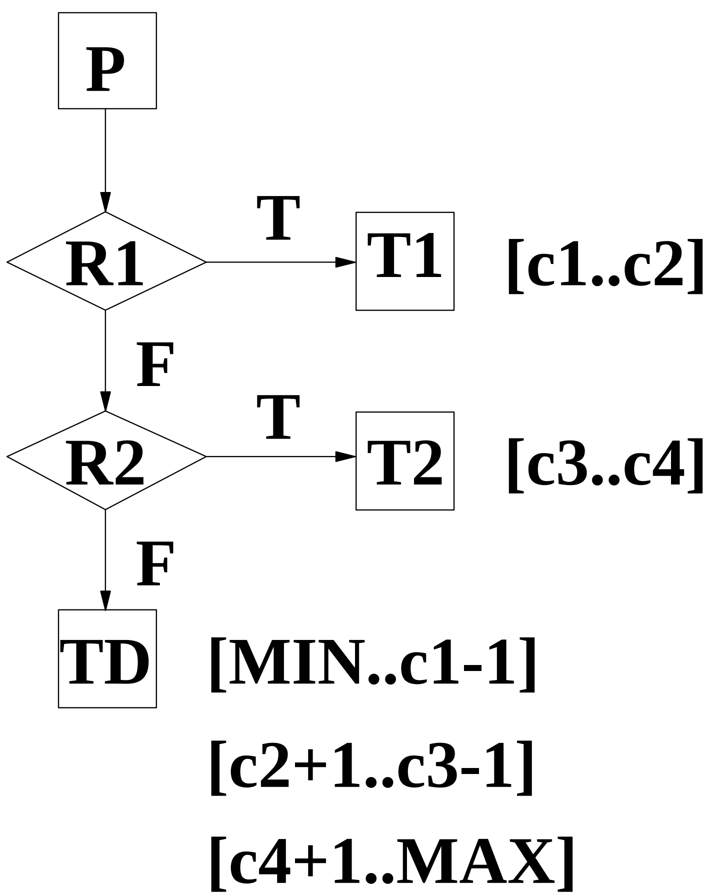
(b) Control Flow

Example of Detecting Range Conditions (cont.)

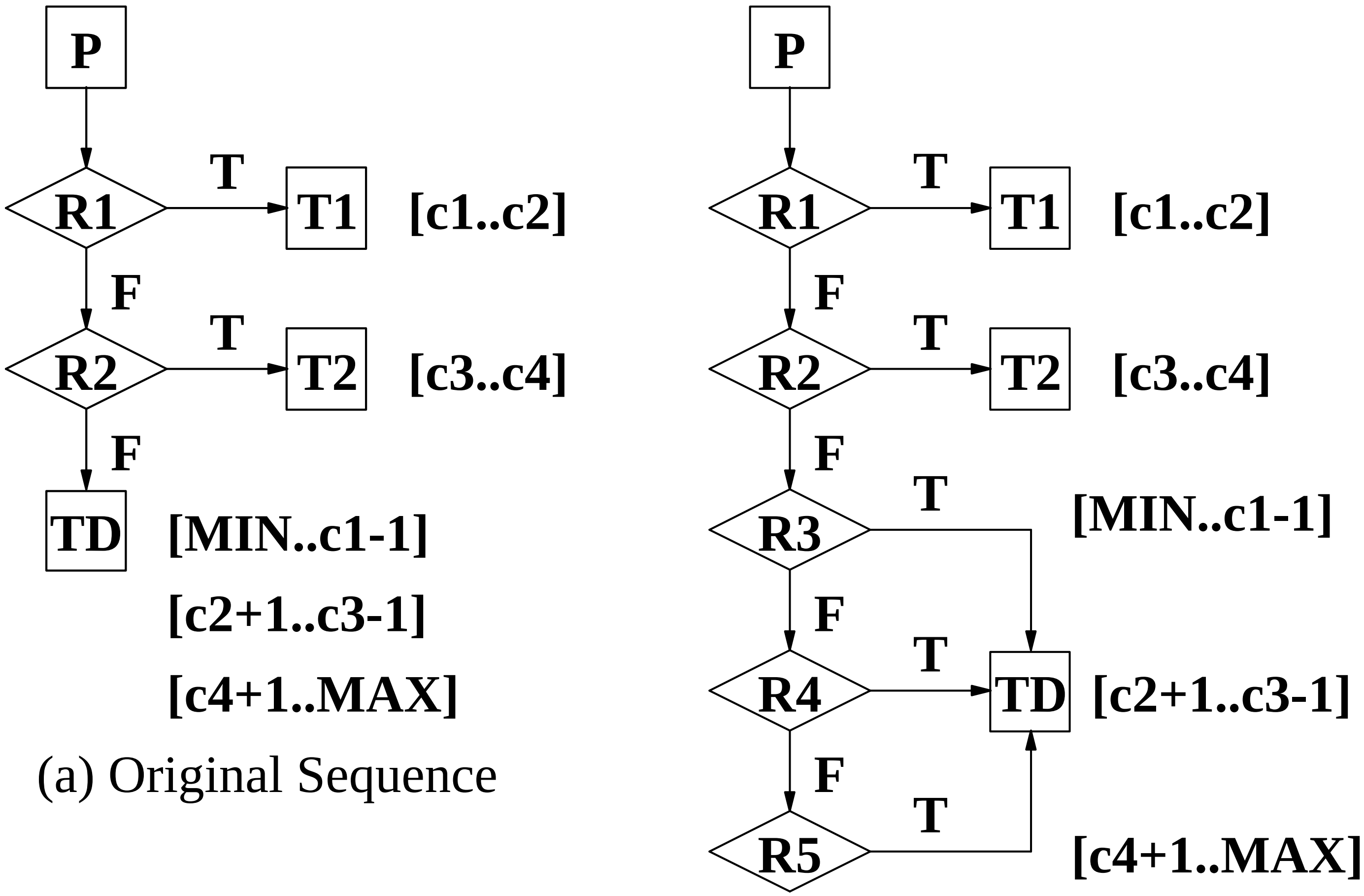
Blocks	Range	Target
1,2	[97..122]	T1
3,4	[65..90]	T1
6	[95..95]	T2
8	[127..MAX]	T4

Explicit and Default Ranges

- An ***explicit range*** is a range that is checked by a range condition.
- A ***default range*** is a range that is not checked by a range condition.



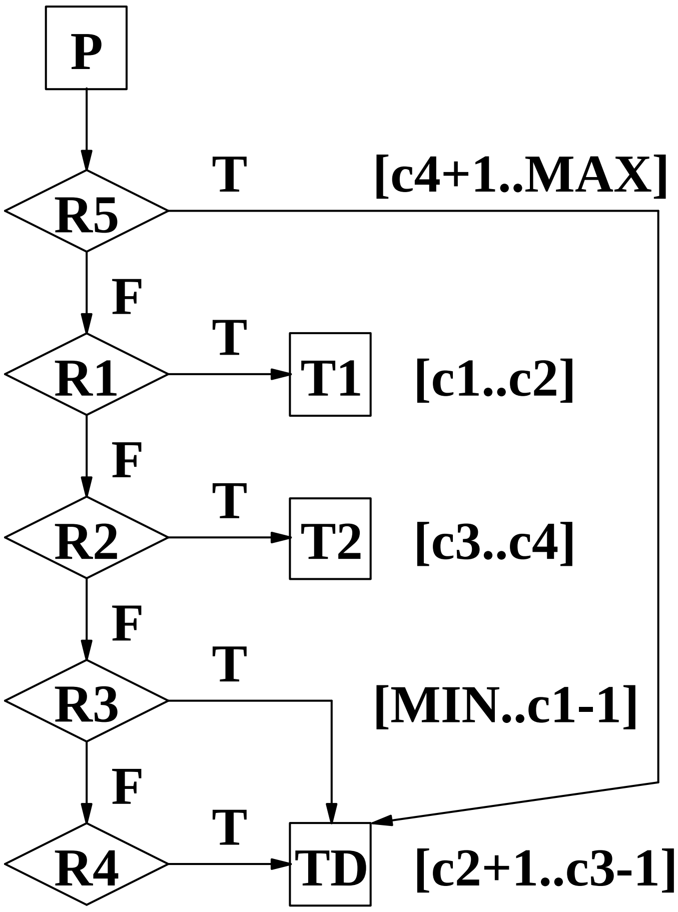
Example of Reordering Range Conditions



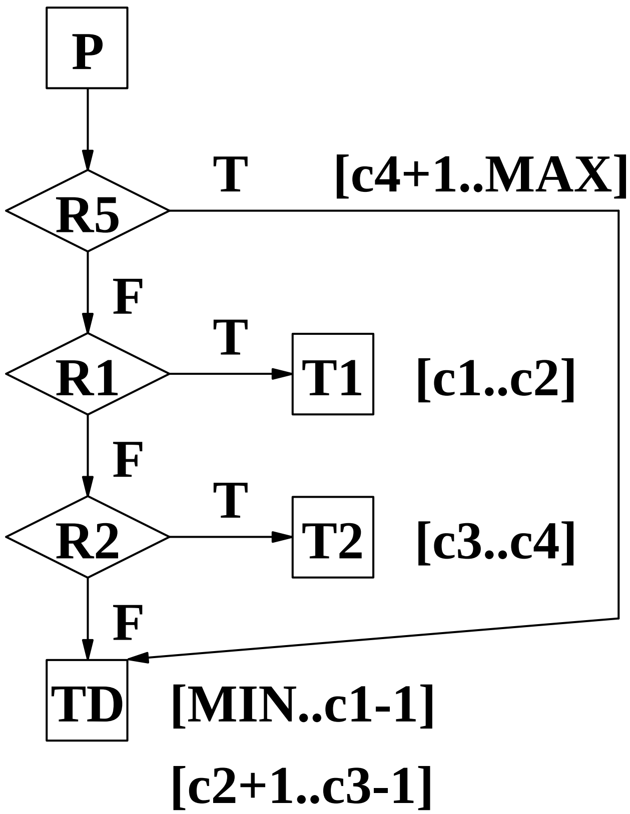
(a) Original Sequence

(b) Equivalent Original Sequence

Example of Reordering Range Conditions (cont.)



(c) Reordered Sequence



(d) Equivalent Reordered Sequence

Sequence Cost Equations

pi is the probability that Ri will exit the sequence.

ci is the cost of testing Ri.

$$Explicit\_Cost([R_1, \ldots, R_n]) \\ = p_1 c_1 + p_2(c_1 + c_2) + \cdots + p_n(c_1 + c_2 + \cdots + c_n)$$

The optimal order of a sequence of explicit range conditions is achieved by sorting them in descending order of ***pi***/***ci***.

$$Cost([R_1, \ldots, R_n]) = Explicit\_Cost([R_1, \ldots, R_n]) + \\ (1 - (p_1 + \cdots + p_n))(c_1 + \cdots + c_n)$$

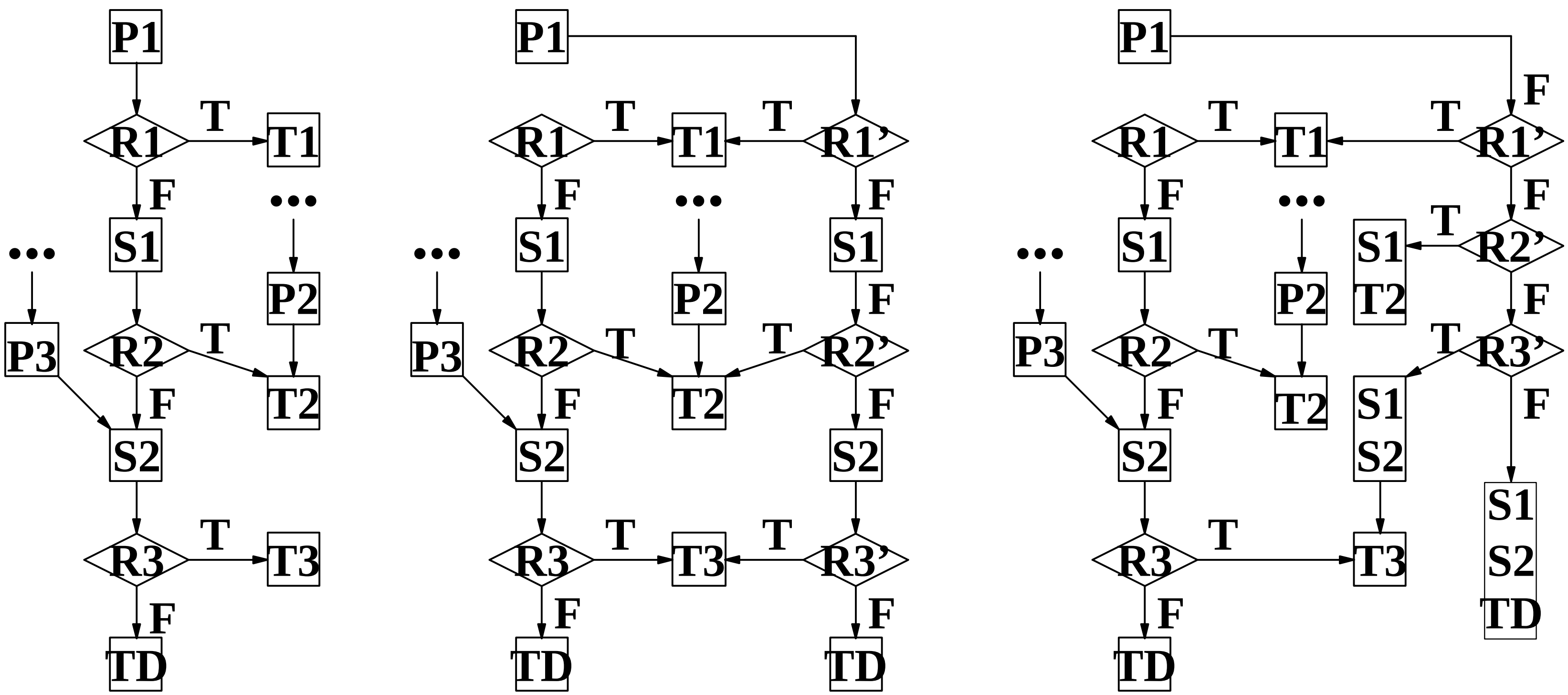
Selecting the Sequence Ordering

- We need to select one of *t* targets as the default.
- A potential default target having *m* ranges could have 2^m-1 combinations of ranges that do not have to be explicitly checked.
- We used the ordering p1/c1 ≥ ... ≥ pm/cm to select the lowest cost from only *m* combinations of default range conditions for each target.

  {Rm}, {Rm-1,Rm}, ..., {R1, ..., Rm}

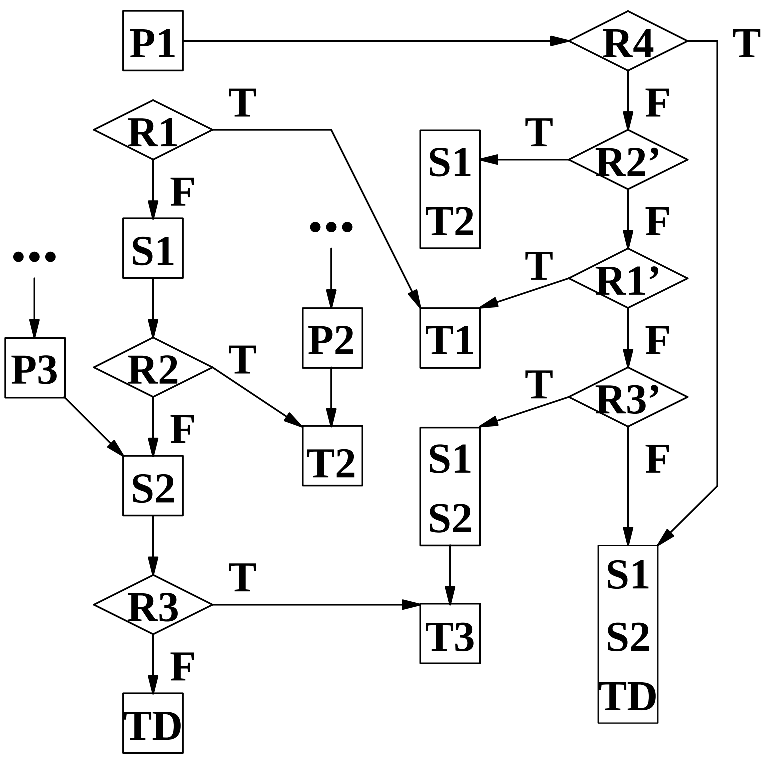
- The minimum cost among the *t* targets is selected.
- Only the cost of *n* sequences are considered, where *n* is the total number of ranges for all of the targets.

Applying the Reordering Transformation

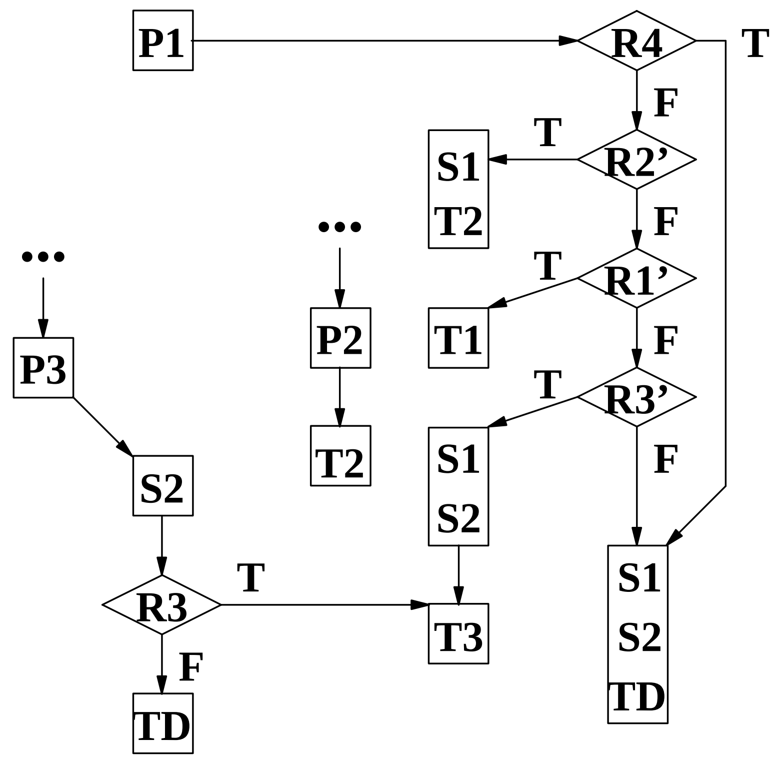


(a) Original Sequence (b) After Duplicating the Sequence (c) After Eliminating Intervening Side Effects

Applying the Reordering Transformation (cont.)



(d) After Reordering Range Conditions



(e) After Dead Code Elimination

Heuristics Used for Translating *switch* Statements

Term	Definition
n	Number of cases in a **`switch`** statement.
m	Number of possible values between the first and last case.

Heuristic Set	Indirect Jump	Binary Search	Linear Search
I	$n \geq 4$ && $m \leq 3n$	!indirect_jump && $n \geq 8$	!indirect_jump && !binary_search
II	$n \geq 16$ && $m \leq 3n$	!indirect_jump && $n \geq 8$	!indirect_jump && !binary_search
III	never	never	always

Dynamic Frequency Measurements

Switch Translation Heuristics	Program	Original Insts	Reordered Insts	Reordered Branches
Set I	awk	13,611,150	-2.02%	-4.19%
	cb	17,100,927	-7.65%	-15.46%
	cpp	18,883,104	-0.13%	-0.19%
	ctags	71,889,513	-9.10%	-14.72%
	deroff	15,460,307	-1.53%	-2.63%
	grep	9,256,749	-3.60%	-8.31%
	hyphen	18,059,010	+3.42%	+3.40%
	join	3,552,801	-1.68%	-2.12%
	lex	10,005,018	-4.56%	-10.39%
	nroff	25,307,809	-2.48%	-6.35%
	pr	73,051,342	-16.25%	-29.96%
	ptx	20,059,901	-9.18%	-13.28%
	sdiff	14,558,535	-16.09%	-37.03%
	sed	14,229,310	-1.16%	-2.03%
	sort	23,146,400	-47.20%	-57.38%
	wc	25,818,199	-15.05%	-26.26%
	yacc	25,127,817	-0.25%	-0.44%
	average	23,477,465	-7.91%	-13.37%
Set II	average	23,510,571	-8.37%	-14.30%
Set III	average	24,556,842	-12.72%	-20.75%

Execution Time

Machine	Heuristic Set	Average Execution Time
SPARC IPC	I	-4.94%
SPARC 20	I	-5.57%
SPARC Ultra I	II	-2.88%

Future Work

- Using Binary Search Instead of Linear Search
- Contrasting Various Semi-static Search Methods
  - — Linear Search
  - — Binary Search
  - — Jump Table
  - — Combinations of Methods
- Reordering Branches with a Common Successor