Fault-based Testing

Reading assignment

- JUnit is a regression testing framework written by Erich Gamma and Kent Beck.
- JUnit Test Infected: Programmers Love Writing Tests , http://junit.sourceforge.net/doc/testinfected/testing.htm
- References
  - JUnit, http://junit.sourceforge.net/
  - http://www.junit.org/index.htm

Structural Test Data Selection

- Random
- Coverage based
  - Control flow
  - Data flow
- Fault-based
  - Error (fault) seeding
    - e.g., mutation testing
  - Fault constraints
    - E.g., RELAY
- Error-based (Failure-based)
  - domain and computation based
    - use representations created by symbolic execution

} Dependence based analysis

Fault Based Techniques

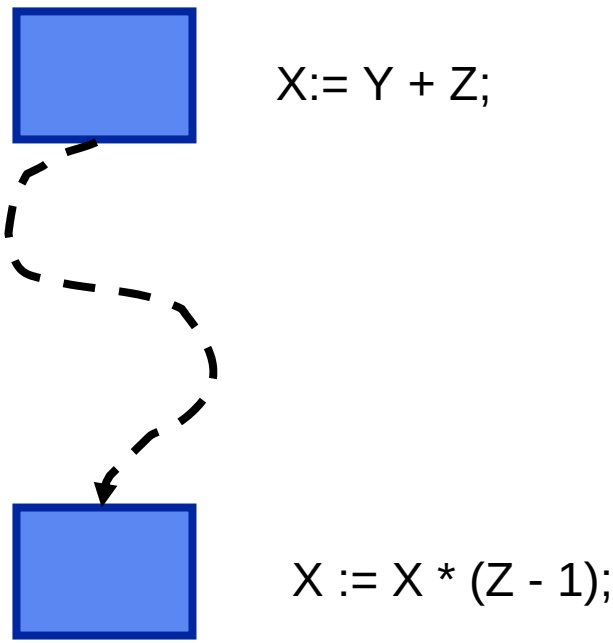
- For each statement try to select test data that will expose faults at that statement
  - Mutation testing monitors effectiveness
  - Fault constraints --instead of monitoring if the selected test data kills a mutant, determine the necessary conditions to guarantee that the fault is revealed if it exists

Remember: comments on dependence based testing coverage

- for selecting test cases
  - syntactic dependence alone is not adequate
    - the number of syntactic dependencies in a program can be quadratic in the number of statements
    - a given syntactic dependence may be demonstrated by (infinitely) many paths
    - propagation of a fault through a particular path may depend on the selection of input data
      ⇒must use semantic information

Can we do better than this?????

Can exercise a dependence relationship but not reveal the fault



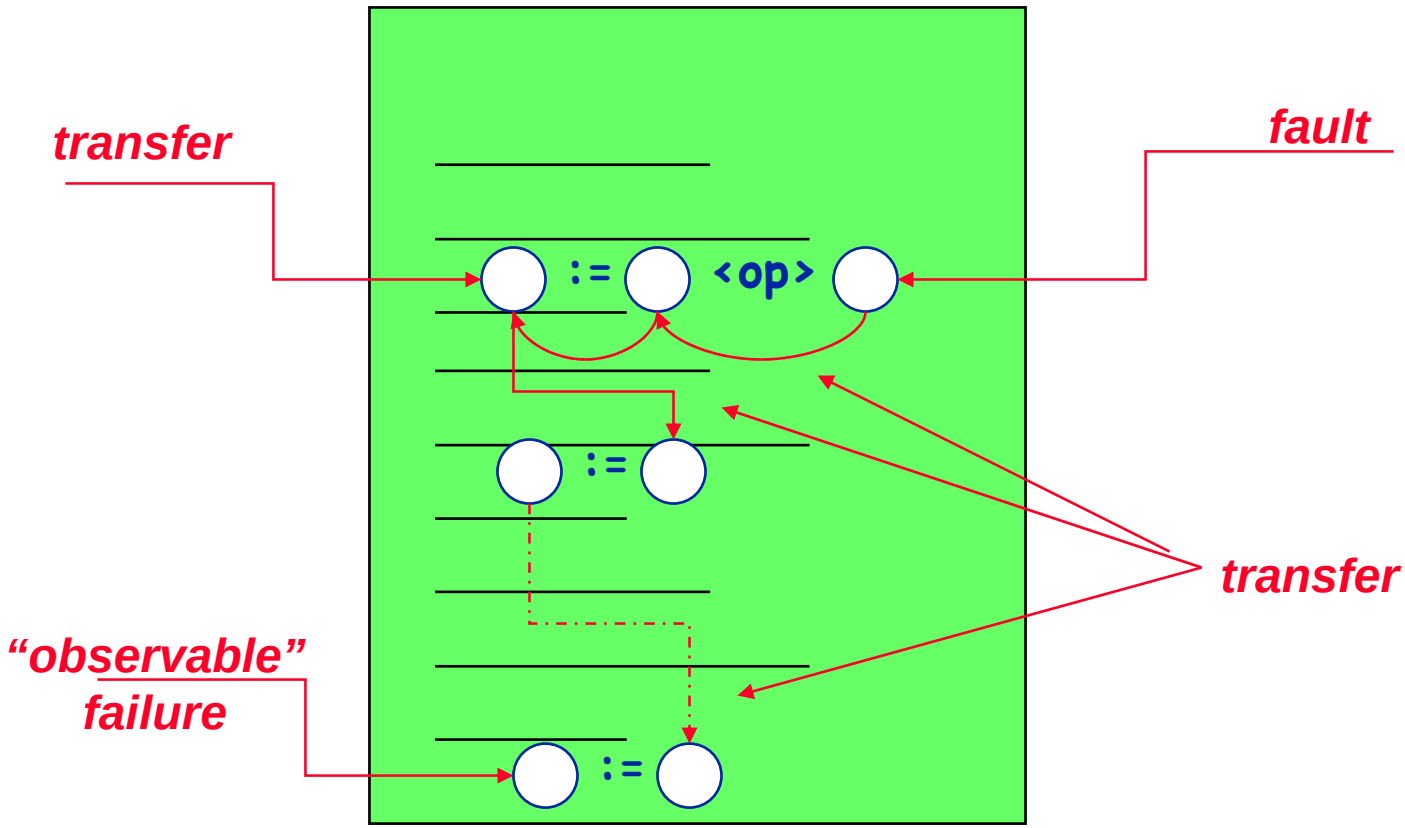
Relay Model

- Selective semantic information + syntactic dependency information
  - origination of a fault
  - computational transfer of a fault
  - propagation of a fault (based on data and control flow)
- Define necessary and sufficient conditions for detecting certain classes of faults

Overview of Relay Model

- origination
  - introduction of potential failure at smallest (valued) subexpression containing fault
- transfer
  - "movement" of potential failure in program
    - Within the originating statement
      - computational transfer
    - From one statement to the next
      - data dependence transfer
      - control dependence transfer

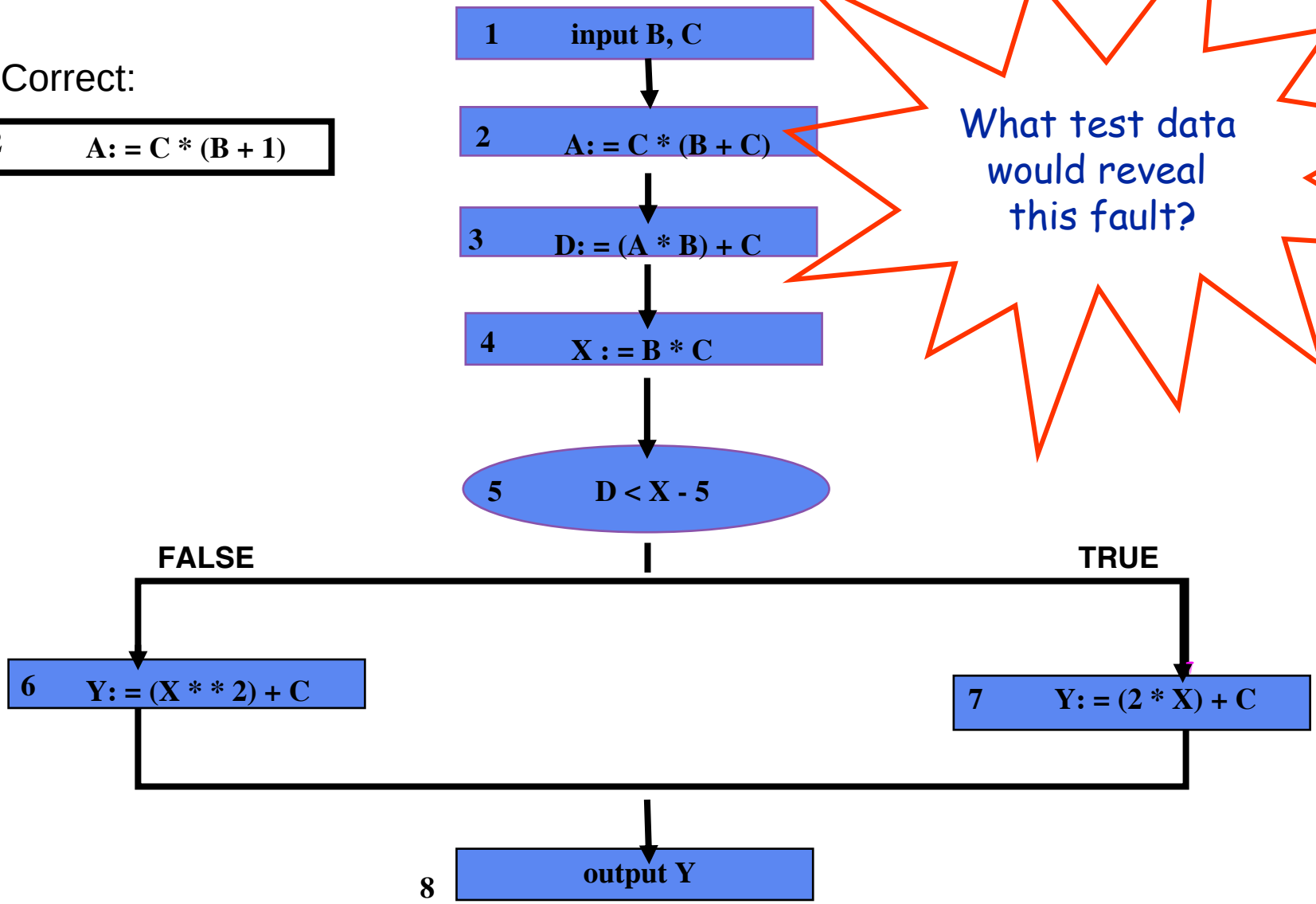
Relay Model



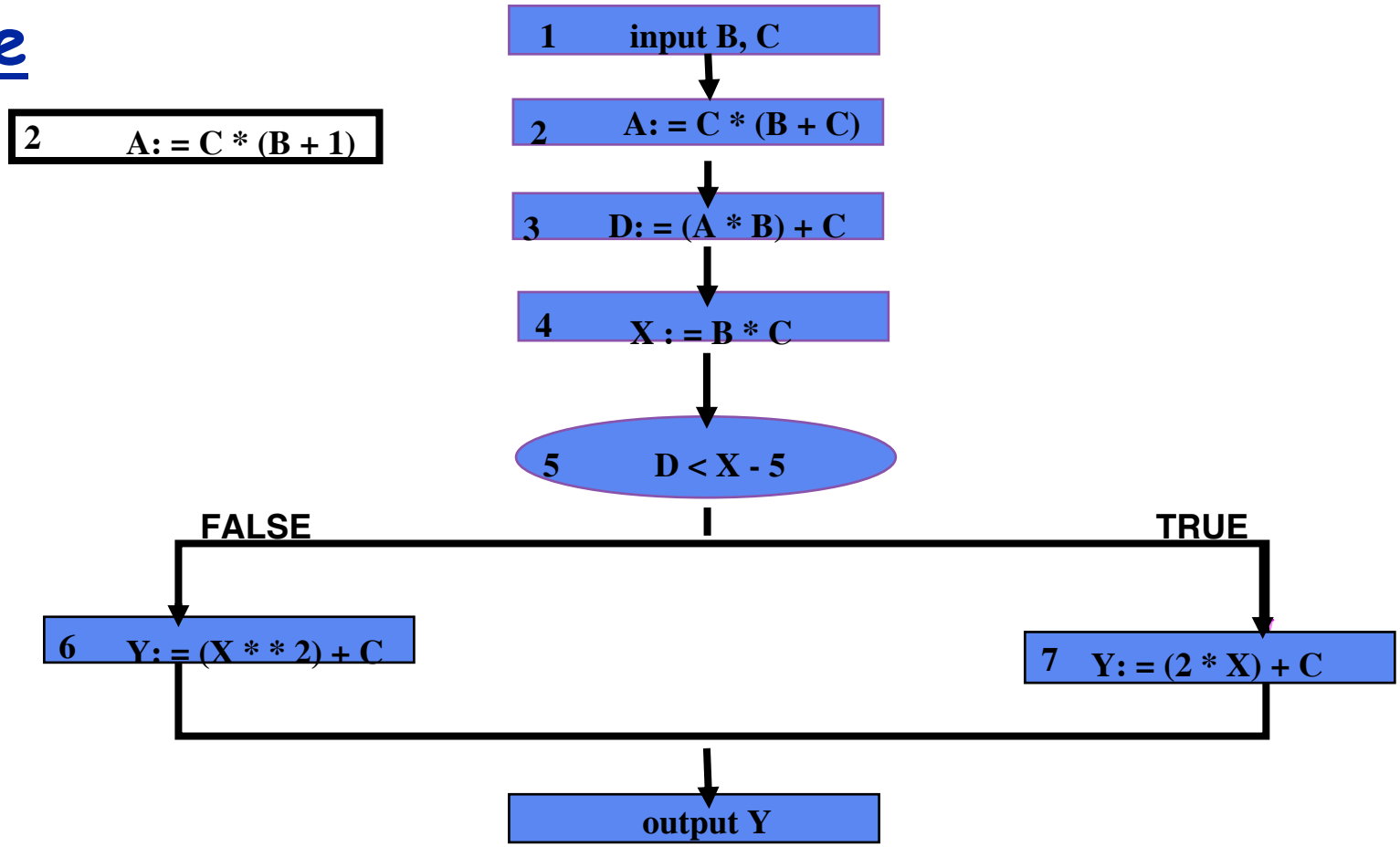
Example

Correct:

2 A: = C * (B + 1)



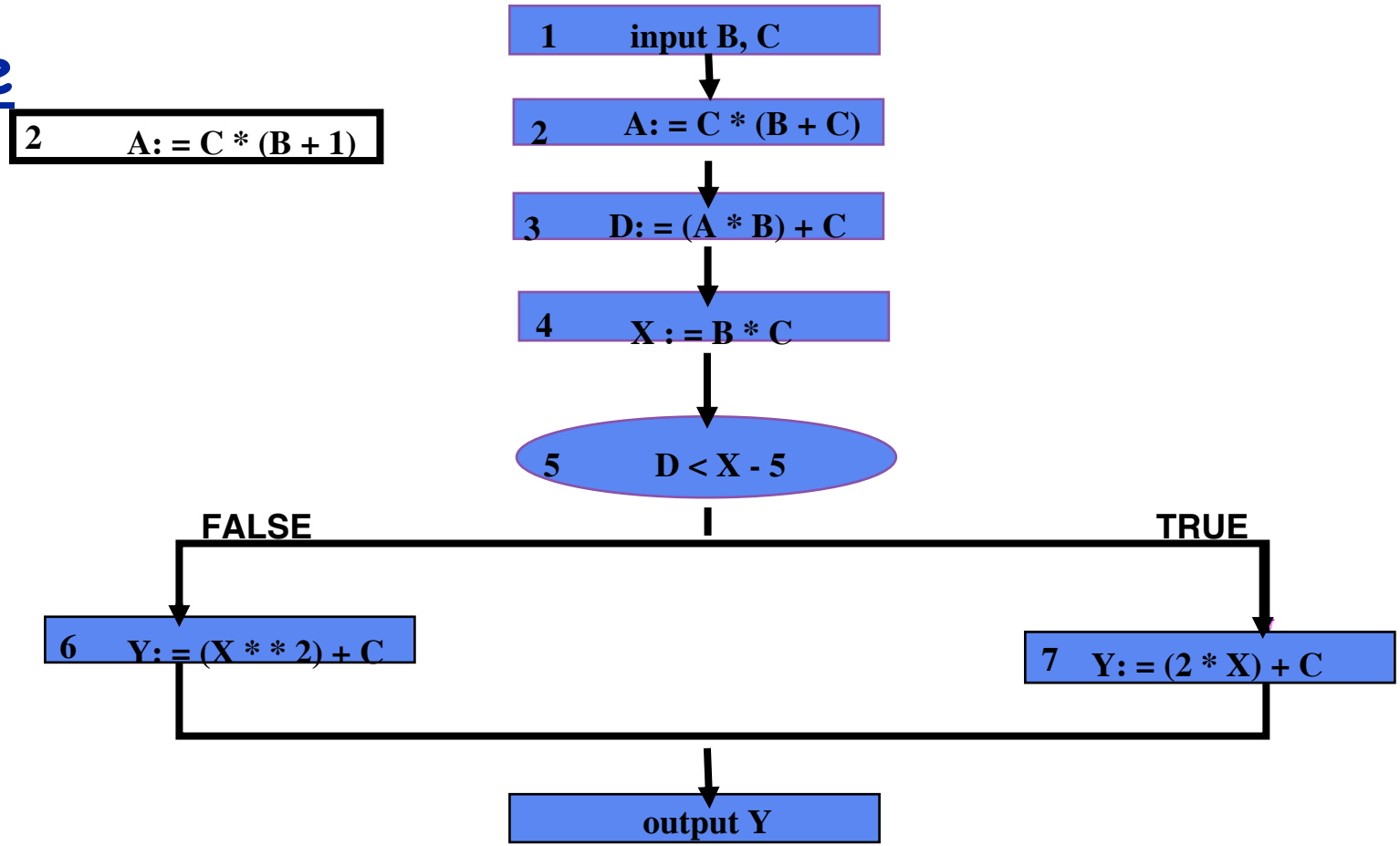
Example



	module	t. c.		exp	*a*	*d*	*x*	*d < x - 5*	*y*	*output*
		b	c							
B+C	faulty	1	1	2	2	3	1	F	2	2
B+1	correct	1	1	2	2	3	1	F	2	2

NO ORIGINATION OF POTENTIAL FAILURE AT NODE 2

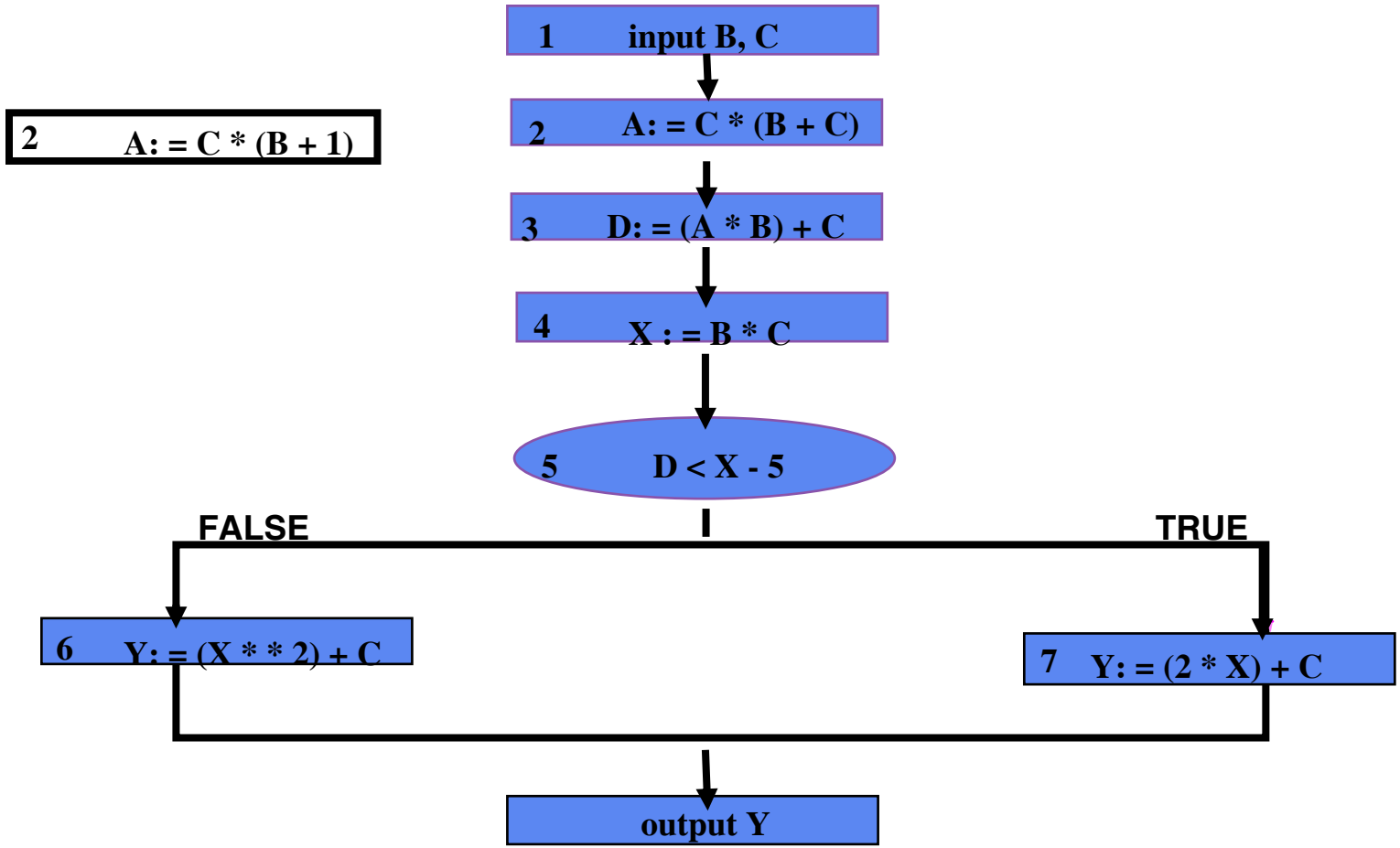
Example



	module	t. c.		exp	*a*	*d*	*x*	*d < x - 5*	*y*	*output*
		b	c							
B+C	faulty	1	0	1	0	0	0	F	0	0
B+1	correct	1	0	2	0	0	0	F	0	0

ORIGINATION (NODE 2),
NO COMPUTATIONAL TRANSFER AT NODE 2

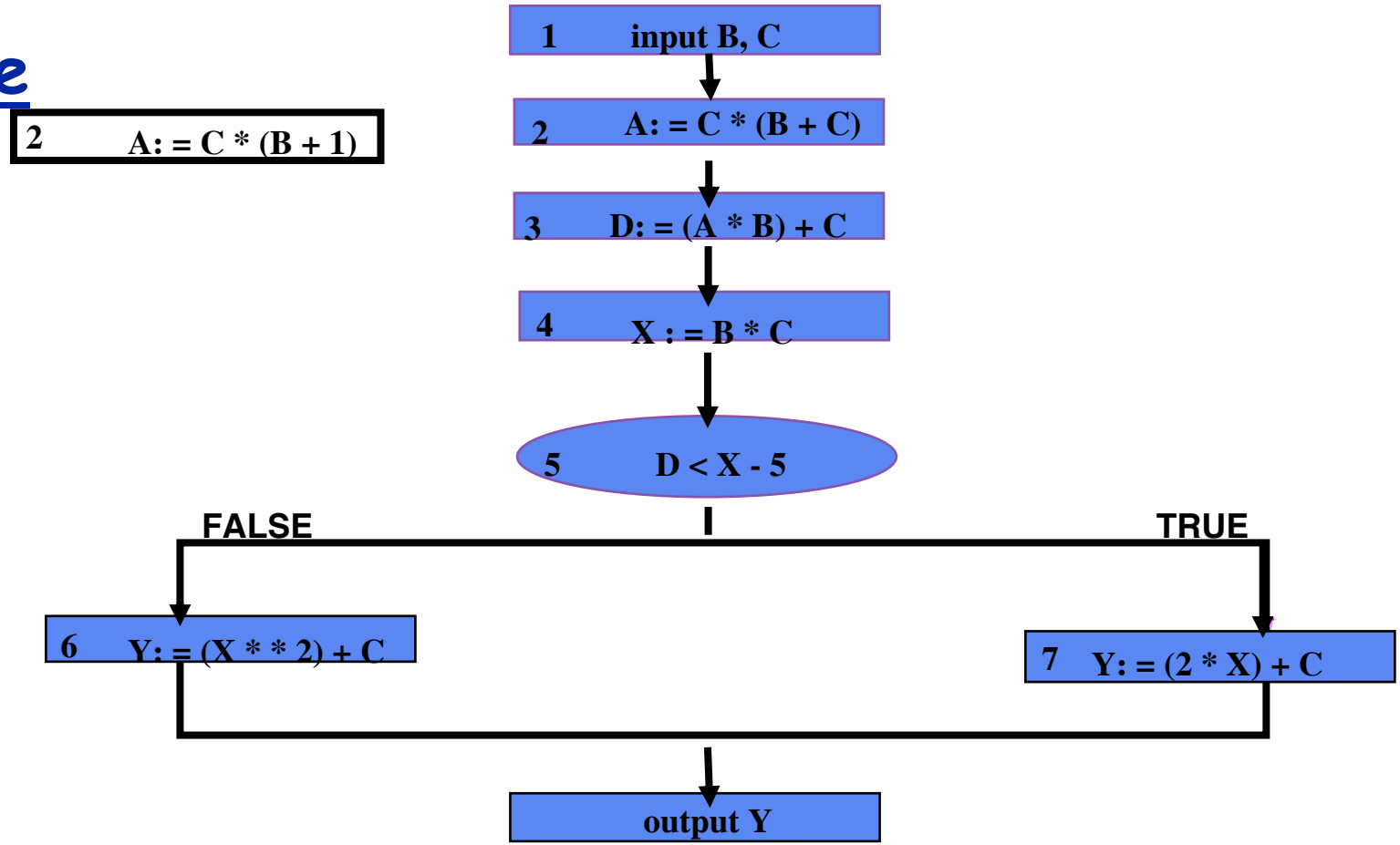
Example



	module	t. c.		exp	*a*	*d*	*x*	*d < x - 5*	*y*	*output*
		b	c							
B+C	faulty	0	3	3	9	3	0	F	3	3
B+1	correct	0	3	1	3	3	0	F	3	3

ORIGINATION, COMP. TRANSFER (NODE 2)
NO DATA DEPENDENCE TRANSFER AT NODE 3

Example

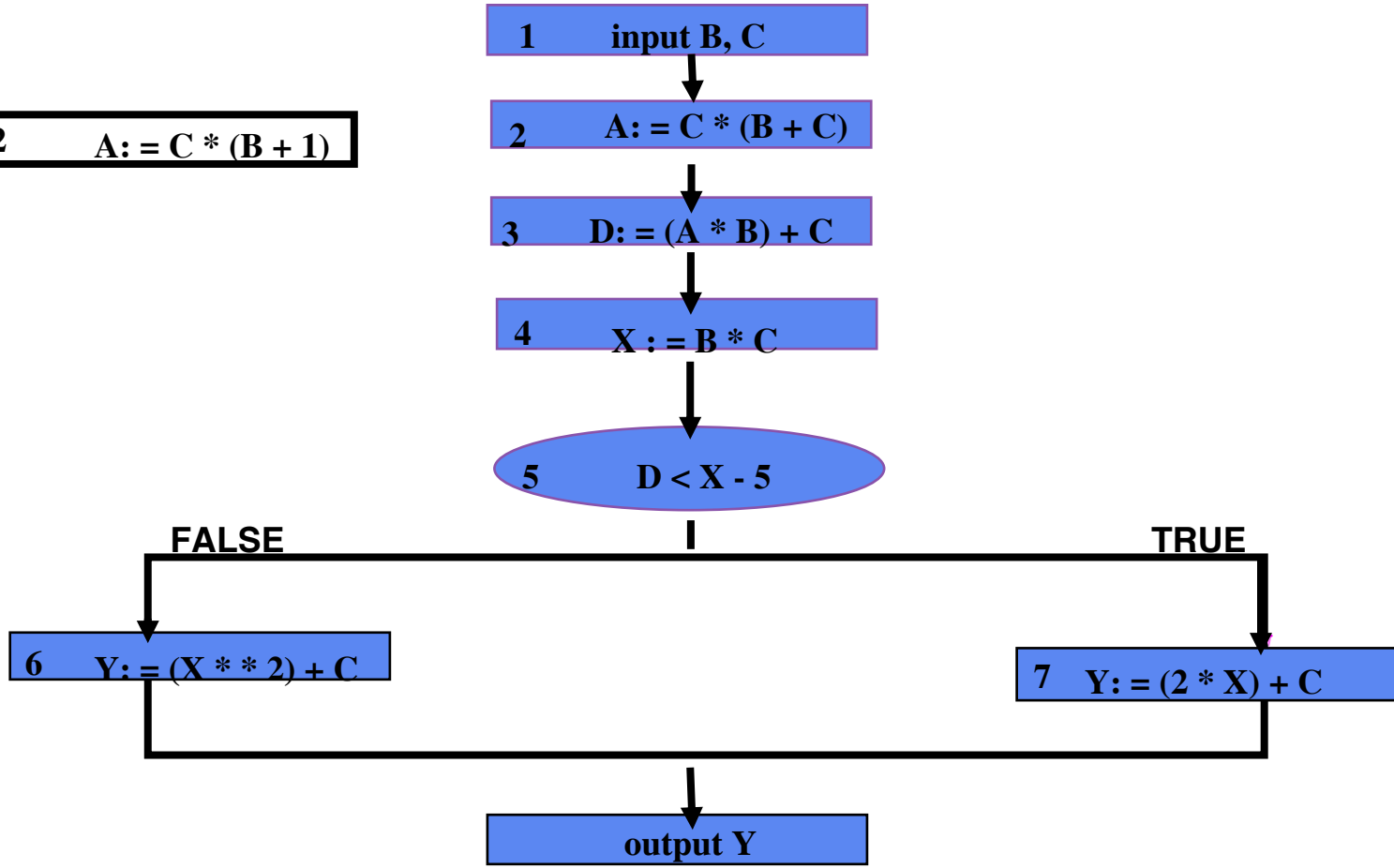


	module	t. c.		exp	*a*	*d*	*x*	*d < x - 5*	*y*	*output*
		b	c							
B+C	faulty	2	3	5	15	33	6	F	39	39
B+1	correct	2	3	3	9	21	6	F	39	39

ORIGINATION, COMP. TRANSFER (NODE 2)
DATA DEPENDENCE TRANSFER (NODE 3)
NO DATA DEPENDENCE TRANSFER AT NODE 5

Example

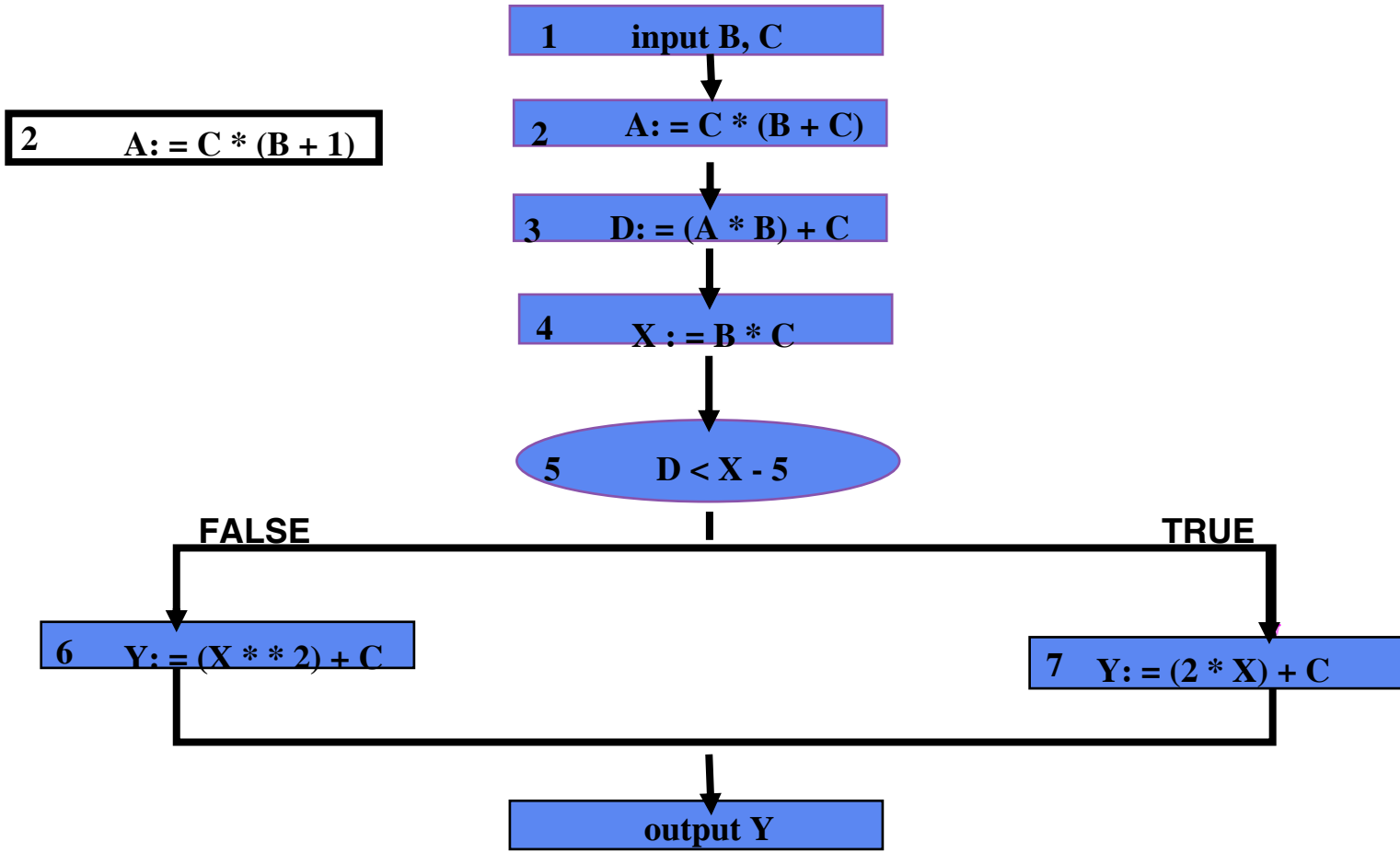
2 A: = C * (B + 1)



	module	t. c. b	t. c. c	exp	*a*	*d*	*x*	*d < x - 5*	*y*	*output*
B+C	faulty	-2	-1	-3	3	-7	2	T	3	3
B+1	correct	-2	-1	-1	1	-3	2	F	3	3

ORIGINATION, COMP. TRANSFER (NODE 2)
DATA DEPENDENCE TRANSFER (NODES 3, 5)
NO CONTROL DEPENDENCE TRANSFER AT NODE 7 and 6

Example



	module	t. c. b	t. c. c	exp	*a*	*d*	*x*	*d < x - 5*	*y*	*output*
B+C	faulty	1	-3	-2	6	3	-3	F	6	6
B+1	correct	1	-3	2	-6	-9	-3	T	-9	-9

ORIGINATION, COMP. TRANSFER (NODE 2)
DATA DEPENDENCE TRANSFER (NODES 3, 5)
CONTROL DEPENDENCE TRANSFER (NODE 6)
FAILURE AT NODE 8

To Guarantee Detection

Step 1: guarantee introduction of potential failure at statement containing hypothetical fault

- origination condition
- computational transfer conditions at statement
- called original state potential failure condition

Step 2: guarantee transfer of potential failure along information flow to some output

- called transfer set condition

Step 1a

- origination condition
  - guarantees introduction of potential failure in smallest subexpression
  - exp ≠ exp*
  - defined for fault
  - suppose c*(b+1) instead of c*(b+c)

$$\Rightarrow c \neq 1$$

Step 1b

- computational transfer condition for a statement
  - exp1 <op> exp2 ≠ exp1' <op> exp2
  - defined for operator and fault

    e.g., (b + c) ≠ (b + 1) ⇒ c ≠ 1
  - many are fault independent

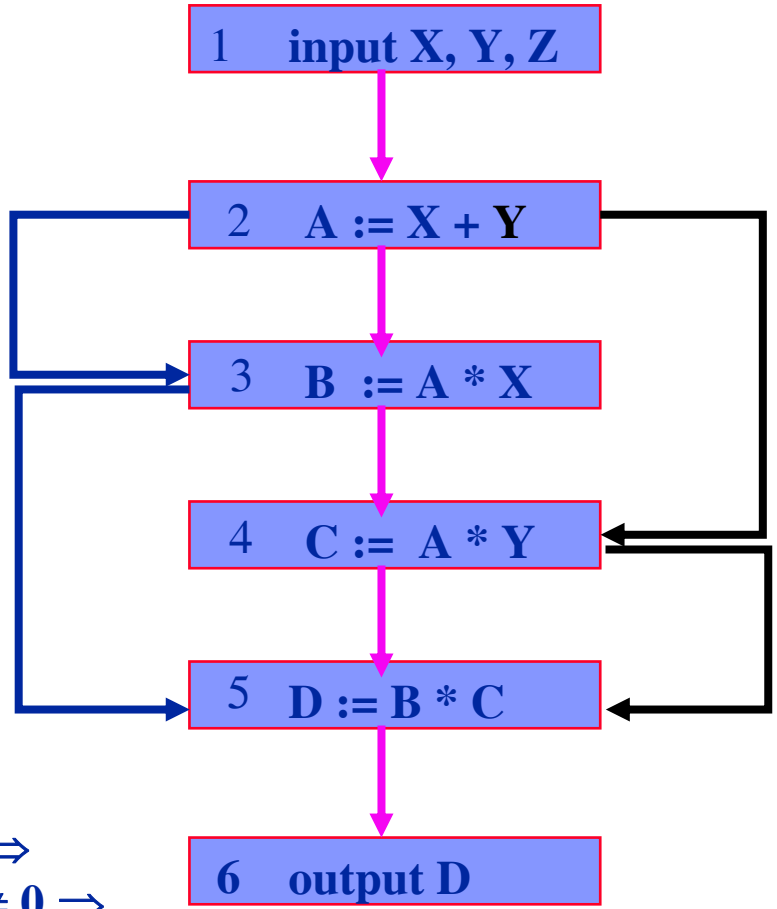
    c * (exp) ≠ c * (exp')

    ⇒ c ≠ 0

Step 2

- Information flow transfer
  - combines data dependence and control dependence transfer
  - occurs along information flow chains
  - to guarantee transfer from (hypothetically) faulty node to output must guarantee transfer along transfer set
    - collection of information flow chains that can be executed together

Simpler Example



2* A: = X + Z

Y ≠ Z

Y ≠ Z

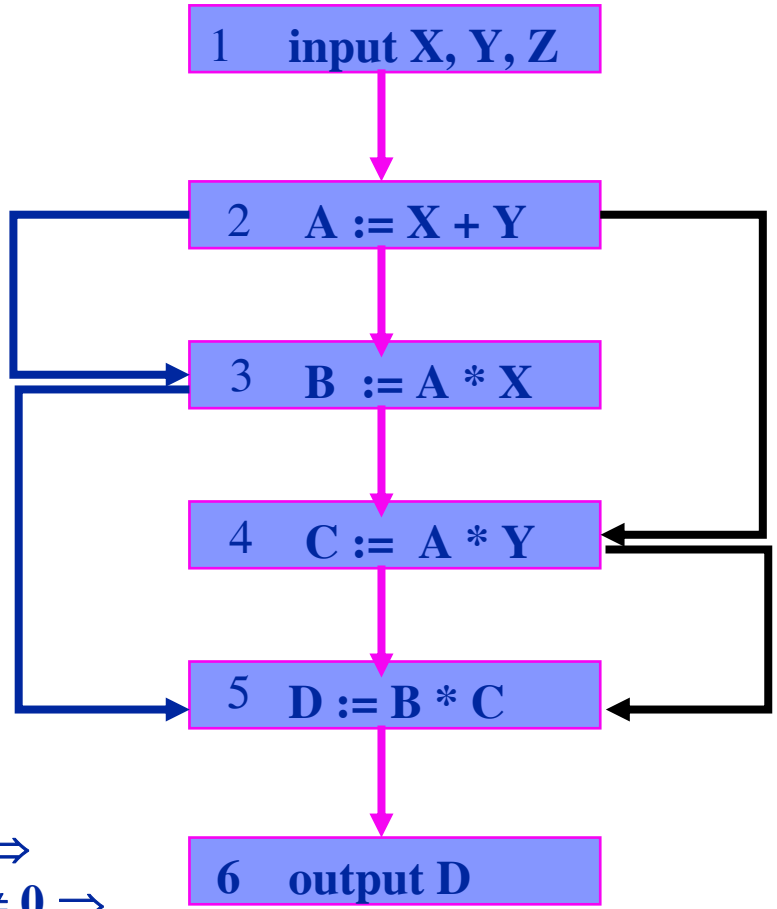
X ≠ 0

Y ≠ 0

C ≠ 0 ⇒
A * Y ≠ 0 ⇒
Y ≠ 0 Λ A ≠ 0 ⇒
Y ≠ 0 Λ X +Y ≠ 0 ⇒
Y ≠ 0 Λ X ≠ –Y

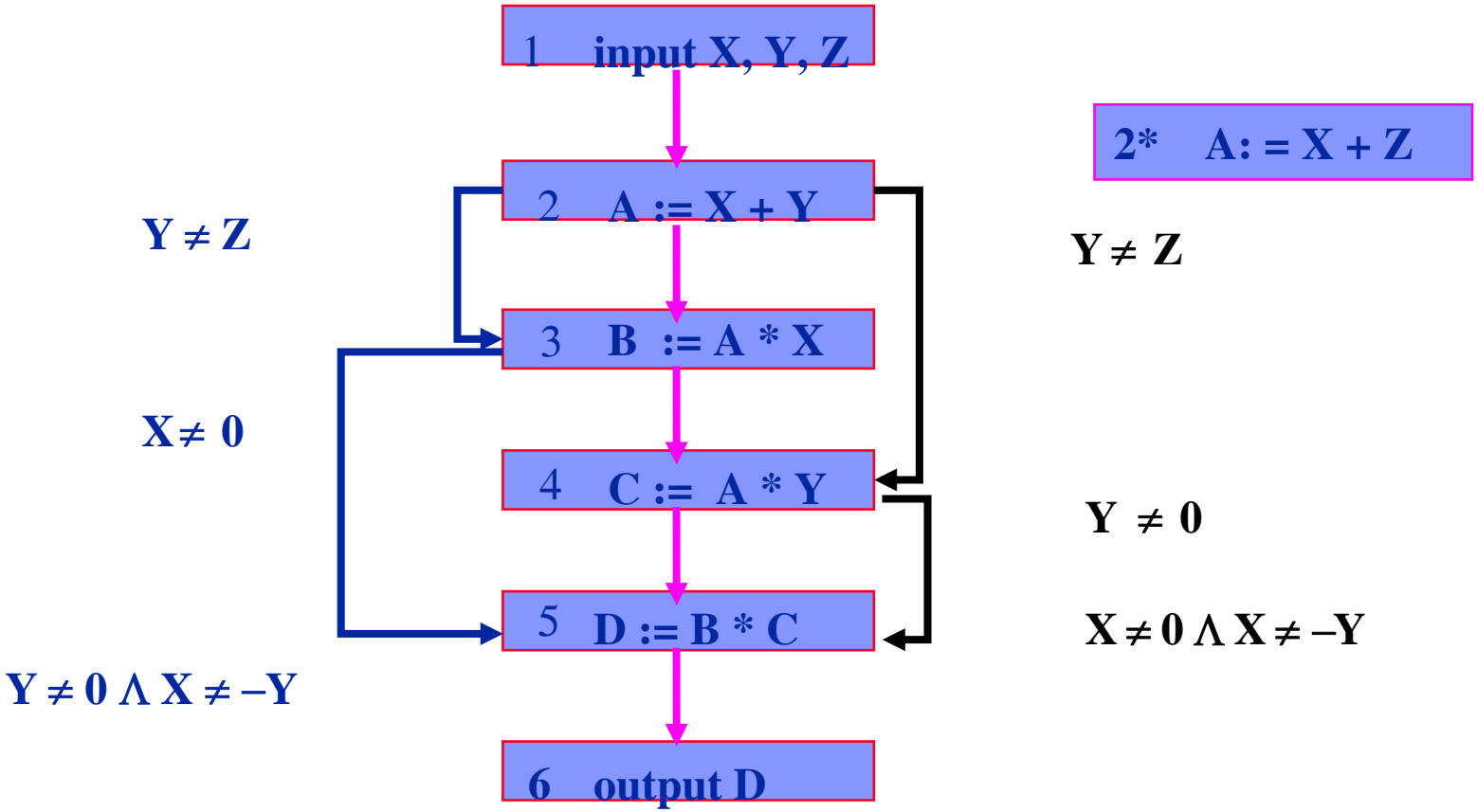
B ≠ 0 ⇒
A * X ≠ 0 ⇒
X ≠ 0 Λ A ≠ 0 ⇒
X ≠ 0 Λ X +Y ≠ 0 ⇒
X ≠ 0 Λ X ≠ –Y

Necessary but not sufficient?



	module	test case			*a*	*b*	*c*	*d*	*output*
		x	y	z					
x+y	faulty	1	-3	1	-2	-2	6	-12	-12
x+z	correct	1	-3	1	2	2	-6	-12	-12

Not Necessary!



		module	test case				*a*	*b*	*c*	*d*	*output*
			x	y	z						
x+y	1	faulty	1	-3	1		-2	-2	6	-12	-12
x+z		correct	1	-3	1		2	2	-6	-12	-12
x+y	2	faulty	1	-1	1		0	0	0	0	0
x+z		correct	1	-1	1		2	2	-2	-4	-4

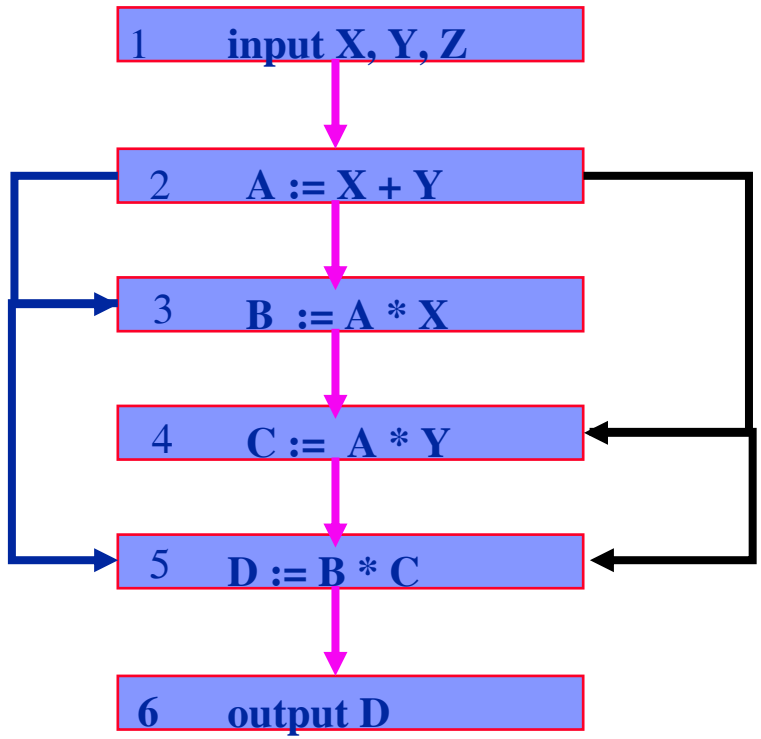
Transfer Condition

- condition that guarantees transfer
  - must know points of interaction
    - places where two or more potential failures come together
- Transfer set defines locations of potential interaction
    - Notation: (U_n, V_m) means faulty value for variable U at node n transfers to variable V at node m
- Transfer route defines chains of transfer set elements that can be combined to form a path

Example

- Transfer Set =

  $\{(A_2, B_3), (B_3, D_5), (D_5, out_6), (A_2, C_4) (C_4, D_5)\}$



Notation: (U_n, V_m) means faulty value for variable U at node n transfers to variable V at node m

Construction of Transfer Route

- different ways to transfer along same set, depending on which portions of chains transfer and which do not
- a transfer route is a subset of the nodes in a transfer set where transfer does and does not occur
- a transfer route defines where actual interactions occur

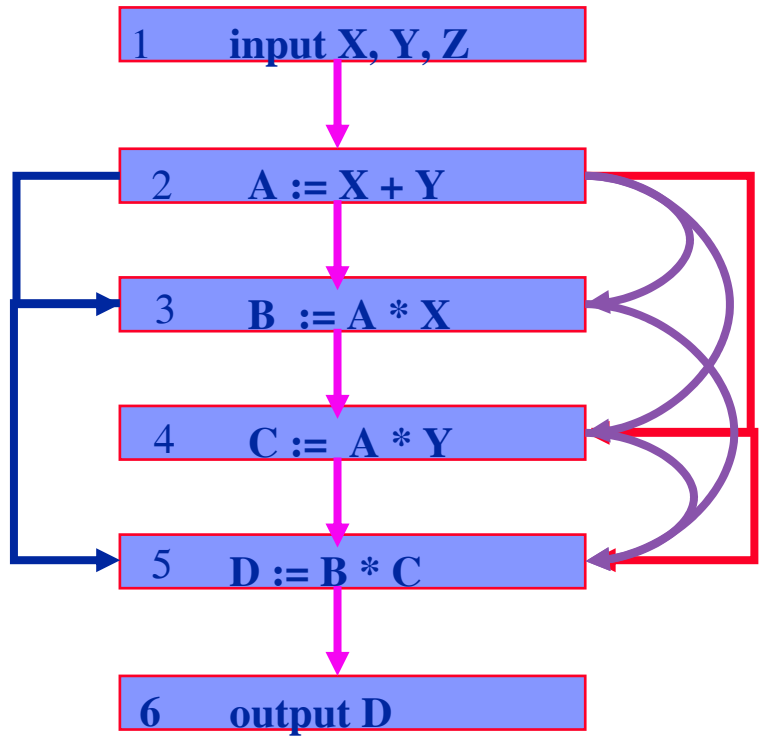
Example

- Transfer Set =

  $\{ (A_2, B_3), (B_3, D_5), (D_5, out_6), (A_2, C_4), (C_4, D_5)\}$

- Transfer Routes

  1. (A transfers to B at 3) and
     (A does not transfer to C at 4)
     and (B transfers to D at 5)
  2. (A does not transfer to B at 3)
     and (A transfers to C at 4)
     and (C transfers to D at 5)
  3. (A transfers to B at 3) and
     (A transfers to C at 4) and
     (B and C transfer to D at 5)

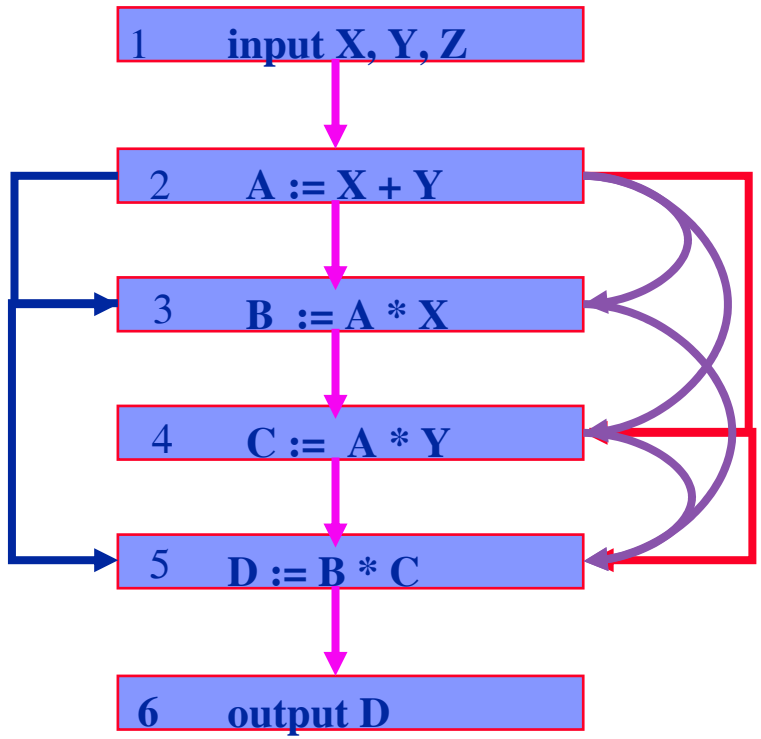


Transfer Condition

- 1. Path Condition
  - guarantees execution of a particular transfer route
    - must guarantee execution of nodes in chain as well as def-clear paths between nodes
- 2. Transfer Route Condition
  - guarantees transfer for particular transfer route
    - computational transfer conditions at nodes in transfer route where transfer does occur
    - complement of computational transfer conditions at nodes where transfer does not occur

Transfer Routes for Example

1. (A transfers to B at 3) and
   (A does not transfer to C at 4) and
   (B transfers to D at 5)
2. (A does not transfer to B at 3) and
   (A transfers to C at 4) and
   (C transfers to D at 5)
3. (A transfers to B at 3) and
   (A transfers to C at 4) and
   (B and C transfer to D at 5)

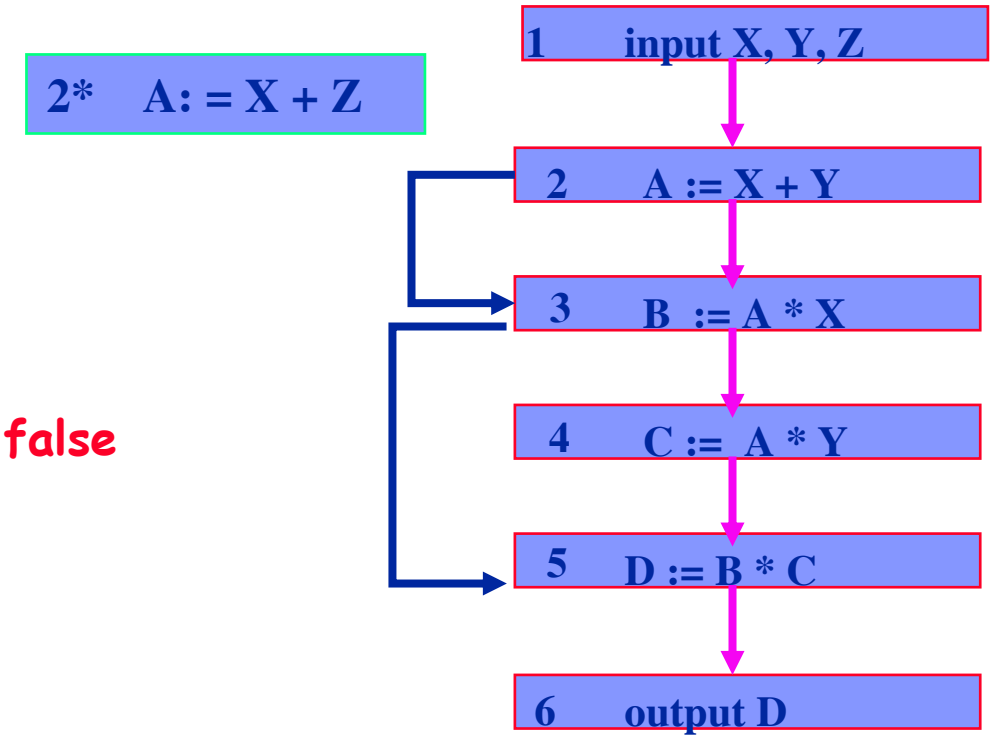


Condition for First Transfer Route

(A transfers to B at 3) and
(A does not transfer to C at 4) and
(B transfers to D at 5)

- Transfer Route Conditions:

$x \neq 0 \wedge y = 0 \wedge c \neq 0 \Rightarrow$
$x \neq 0 \wedge y = 0 \wedge a*y \neq 0 \Rightarrow$ false



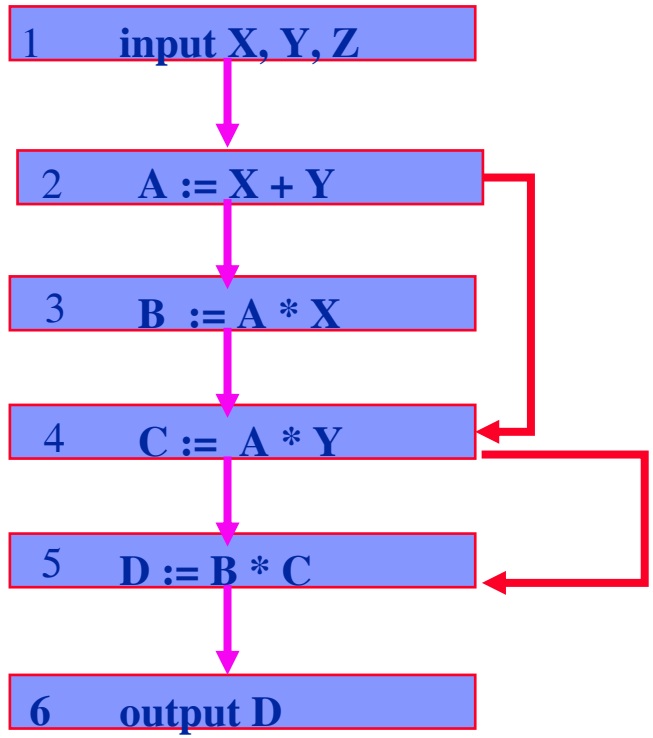
Condition for Second Transfer Route

(A does not transfer to B at 3) and
(A transfers to C at 4) and
(C transfers to D at 5)

- Transfer Route Conditions:

$x = 0 \wedge y \neq 0 \wedge b \neq 0 \Rightarrow$
$x = 0 \wedge y \neq 0 \wedge a*x \neq 0 \Rightarrow$ false

2* A: = X + Z



Condition for Third Transfer Route

(A transfers to B at 3) and
(A transfers to C at 4) and
(B and C transfer to D at 5)

- **Transfer Route Conditions:**

$x \neq 0 \wedge y \neq 0 \wedge b * c \neq b' * c' \Rightarrow$

$x \neq 0 \wedge y \neq 0 \wedge (a*x)(a*y) \neq (a'*x)(a'*y) \Rightarrow$

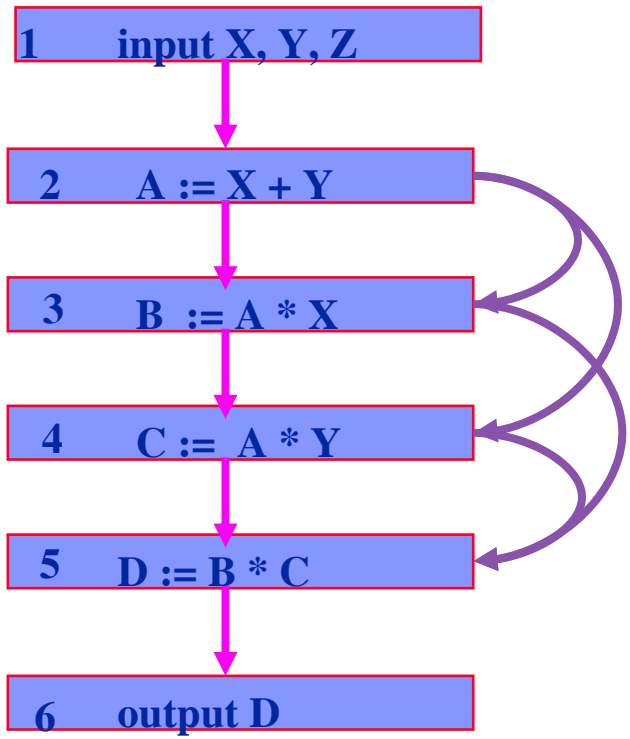
$x \neq 0 \wedge y \neq 0 \wedge$

$(x+y)x(x+y)y \neq (x+z)x(x+z)y \Rightarrow$

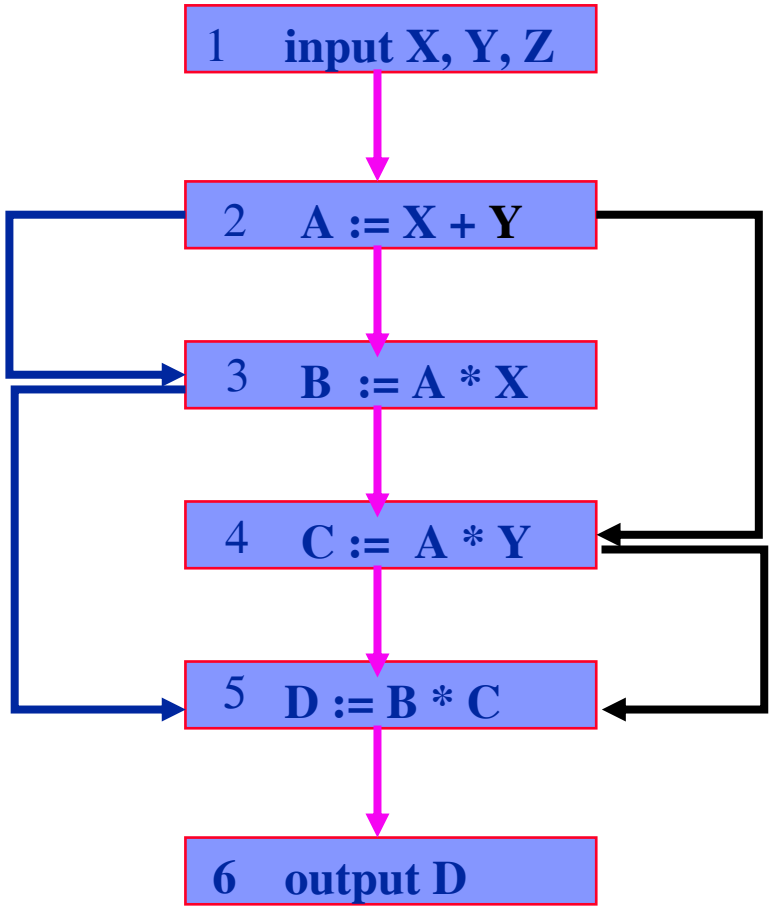
$x \neq 0 \wedge y \neq 0 \wedge (x+y)^2 \neq (x+z)^2 \Rightarrow$

$x \neq 0 \wedge y \neq 0 \wedge y \neq z$

test case: x=1, y= 1, z=2 satisfies the conditions and causes the fault to be revealed



Example



2* A: = X + Z

A= 2
B= 2
C= 4
D= 8

A= 3
B= 3
C= 3
D= 9

test case: x=1, y= 1, z=2 satisfies the conditions $x \neq 0 \wedge y \neq 0 \wedge y \neq z$
and causes the fault to be revealed

Failure condition

failure condition =

original state potential failure condition and transfer condition

if test data satisfies failure condition (fc) and failure → fault

if test data satisfies fc and no failure→ no fault

if can't satisfy fc → try another transfer set

if can't satisfy fc for all transfer sets → no fault

Relay Fault Based Approach

- recognizes what is needed to transfer to output
- other fault based techniques:
  - do not deal with how to select test data that transfers
  - may recognize need to transfer but provide no guidance in test data selection (assume transfer "usually" occurs)
  - do not consider control dependence
  - none discuss interactions for a single fault/multiple faults -- they assume that there is a single fault or if there is more than one that there is no interaction

Relay Fault Based Approach

- defines what is needed to reveal a fault at a statement
  - a general procedure that could be applied to any "atomic" fault
- defines what is needed to propagate erroneous values to output
  - a very negative result!
  - if interaction is not accounted for, then the constraints are neither necessary nor sufficient
  - assumptions about single faults are now very questionable
  - can not assume constraints are necessary