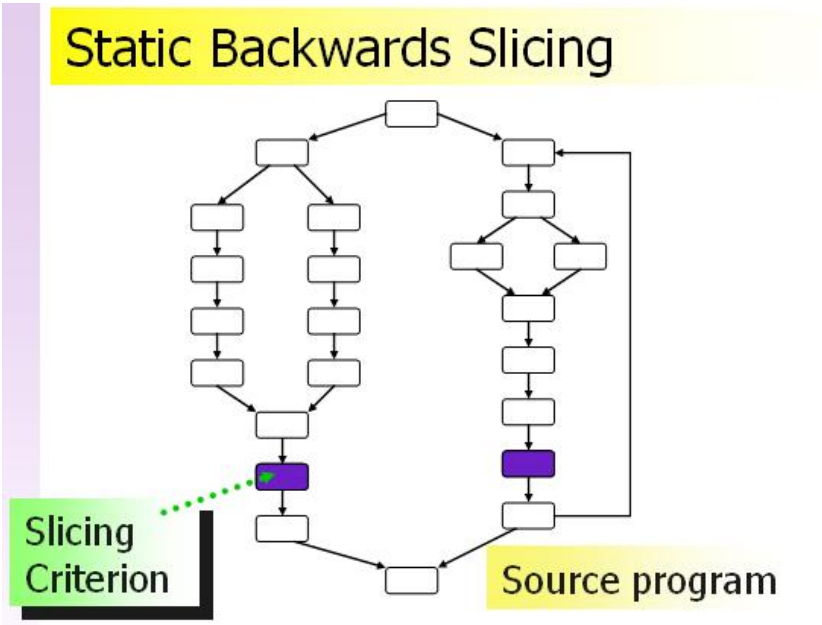
Slicing Modern Program Structures While Eliminating Irrelevant Loops

Torben Amtoft

Kansas State University

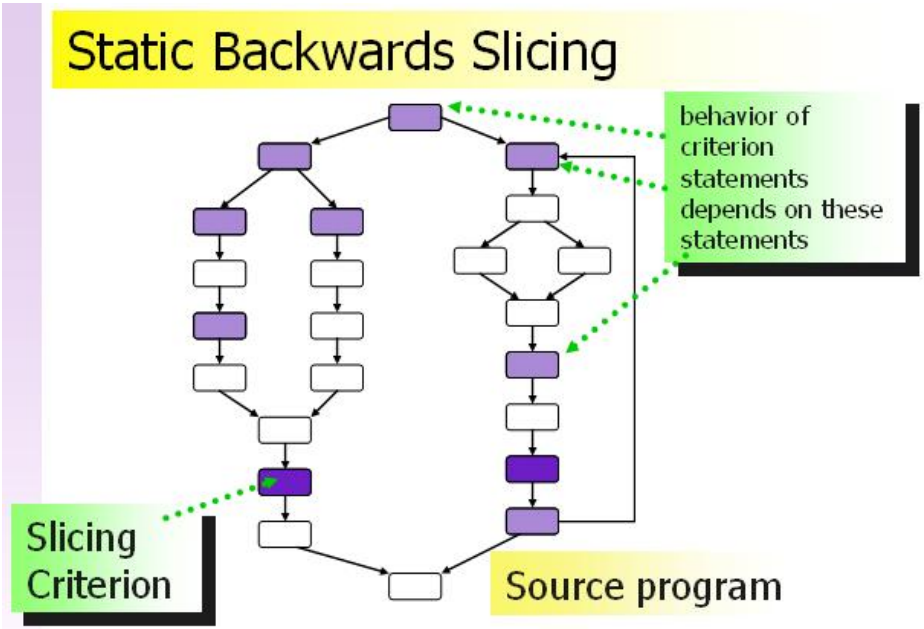
December 2009

What is Slicing?



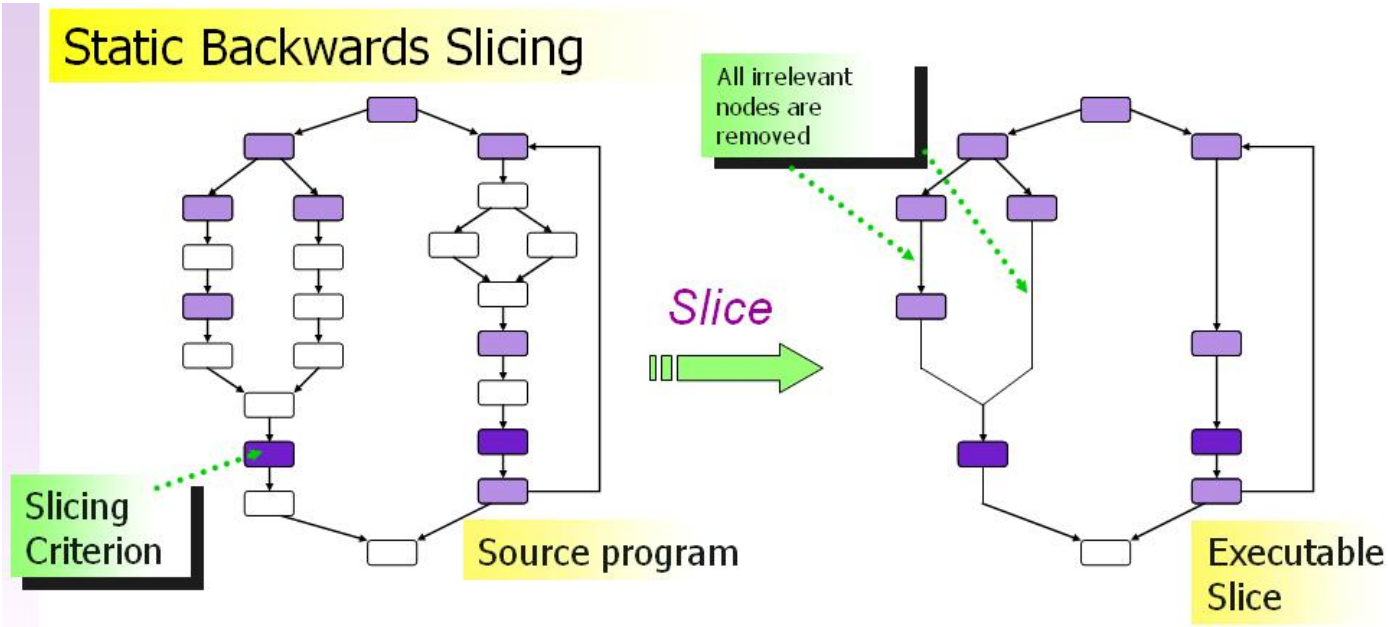
Pick one or more program points of interest (called the slicing criterion)

What is Slicing?



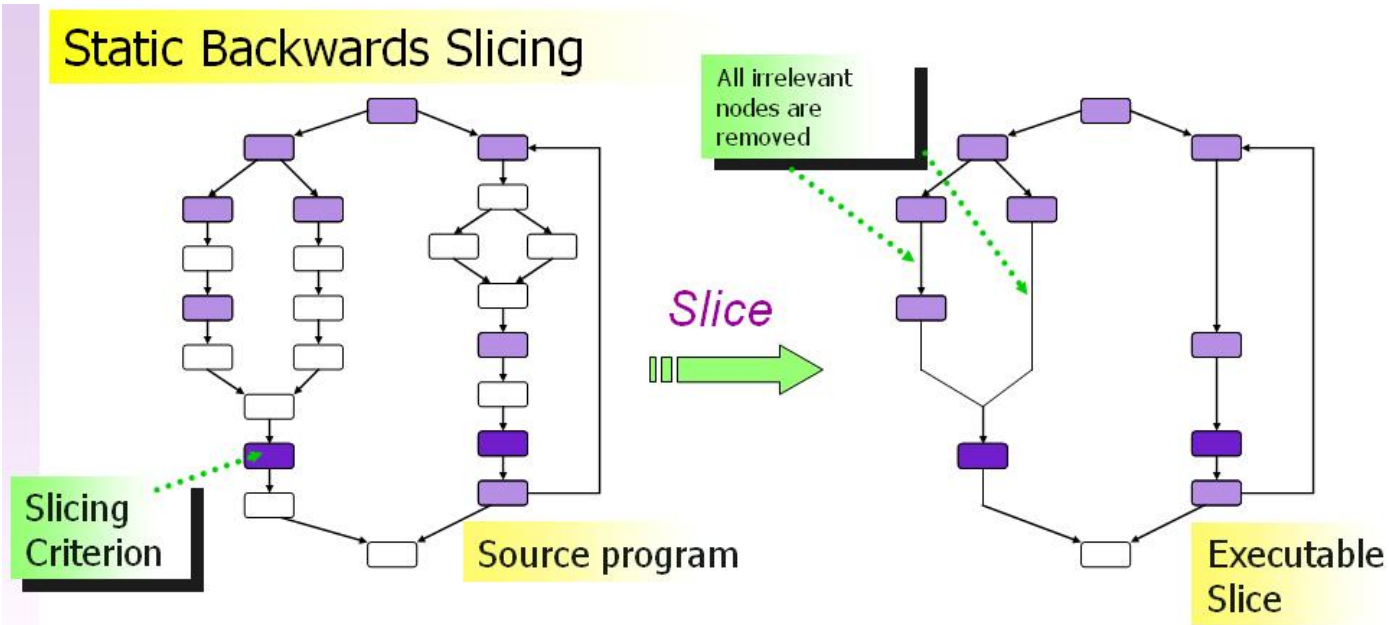
Walk backwards to find nodes that C depend on (called the slice set)

What is Slicing?



Remove irrelevant nodes
(or replace them by dummy nodes)

What is Slicing?



Applications include

- compiler optimizations
- debugging
- model checking
- protocol understanding

Challenge

A definition of control dependency that

1. is applicable to arbitrary CFGs

Challenge

A definition of control dependency that

1. is applicable to arbitrary CFGs, including those that
   - are irreducible (as for state charts)

Challenge

A definition of control dependency that

1. is applicable to arbitrary CFGs, including those that
   - are irreducible (as for state charts)
   - have no end node (as for reactive systems)

Challenge

A definition of control dependency that

1. is applicable to arbitrary CFGs, including those that
   - are irreducible (as for state charts)
   - have no end node (as for reactive systems)
2. produces slices that are likely to be manageable

Challenge

A definition of control dependency that

1. is applicable to arbitrary CFGs, including those that
   - are irreducible (as for state charts)
   - have no end node (as for reactive systems)
2. produces slices that are likely to be manageable

Our proposed solution meets these challenges and is correct in the sense that

- the observable behavior of source program is prefix of observable behavior of sliced program.

Challenge

A definition of control dependency that

1. is applicable to arbitrary CFGs, including those that
   - are irreducible (as for state charts)
   - have no end node (as for reactive systems)
2. produces slices that are likely to be manageable

Our proposed solution meets these challenges and is correct in the sense that

- the observable behavior of source program is prefix of observable behavior of sliced program.
- this is expressed using simulation

Challenge

A definition of control dependency that

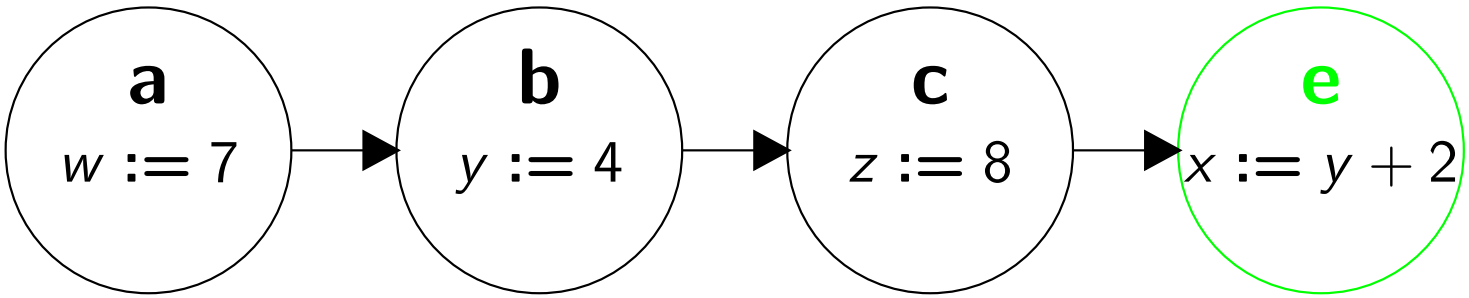
1. is applicable to arbitrary CFGs, including those that
   - are irreducible (as for state charts)
   - have no end node (as for reactive systems)
2. produces slices that are likely to be manageable

Our proposed solution meets these challenges and is correct in the sense that

- the observable behavior of source program is prefix of observable behavior of sliced program.
- this is expressed using simulation
- correctness proof easily fits conference page limits

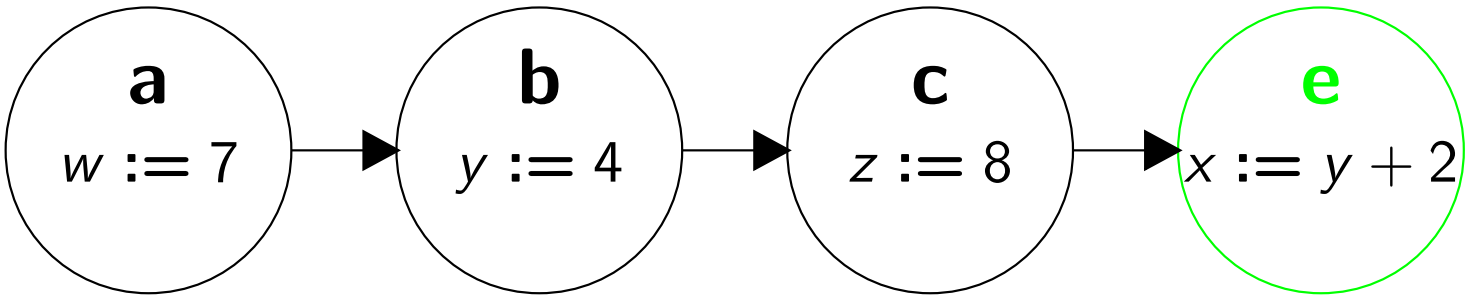
Data Dependence

Source: final value of $x = 6$



Data Dependence

Source: final value of $x = 6$

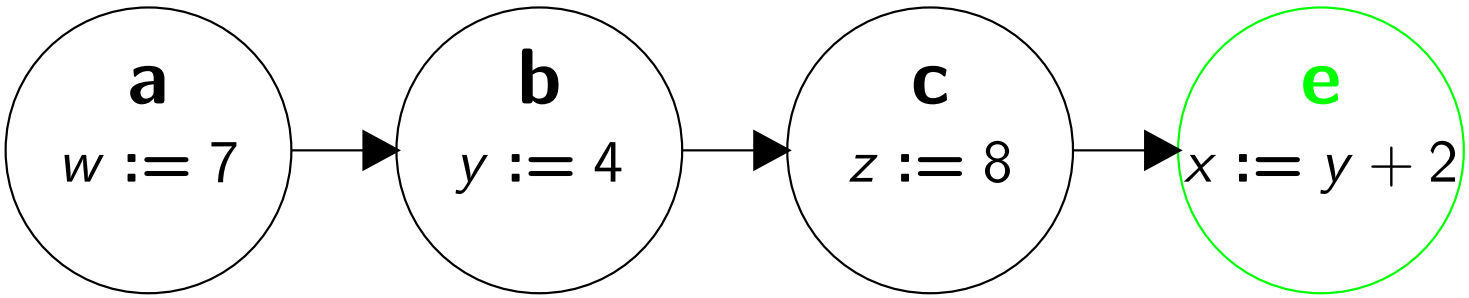


e is data dependent on b, written $b \xrightarrow{dd} e$, since y is

- used in e, defined in b, not defined in between

Data Dependence

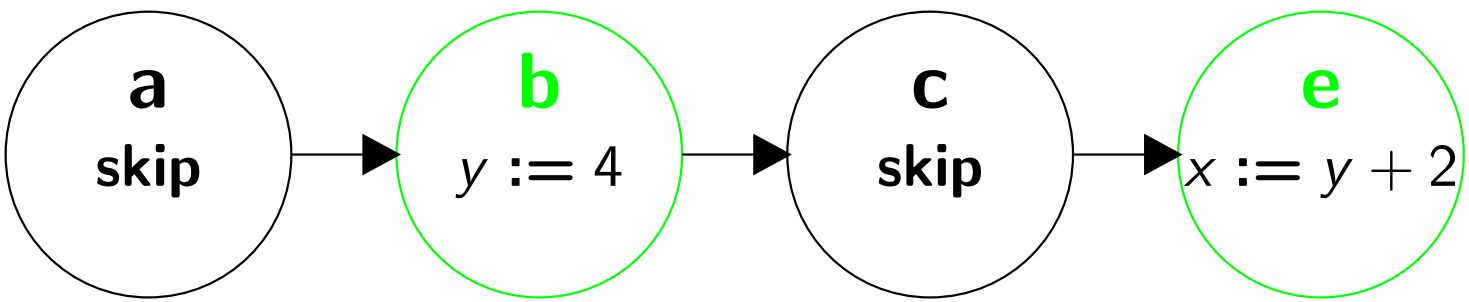
Source: final value of $x = 6$



e is data dependent on b, written $b \xrightarrow{dd} e$, since y is

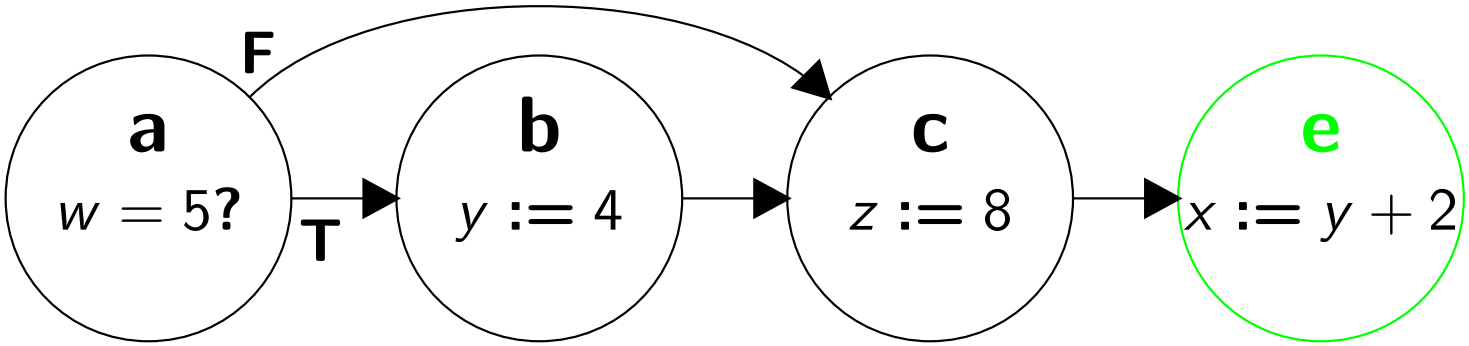
- used in e, defined in b, not defined in between

Slice: final value of $x = 6$



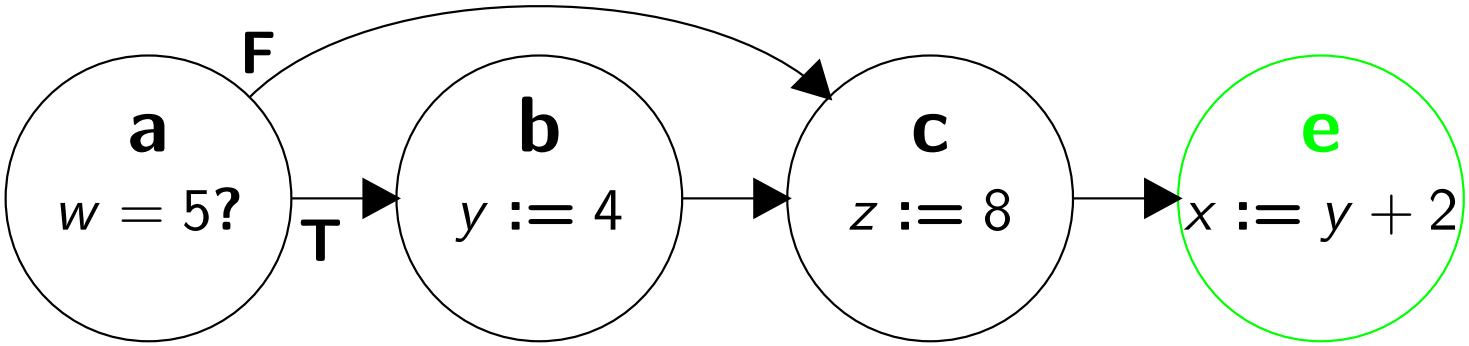
Control Dependence (CD)

Source: final value of $x = 6$ if $w = 5$, else 2



Control Dependence (CD)

Source: final value of $x = 6$ if $w = 5$, else 2

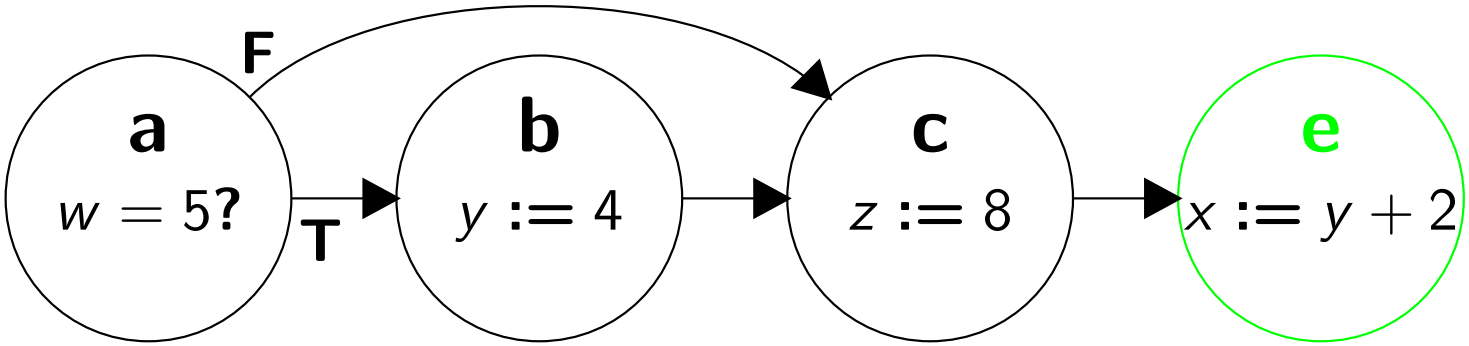


$b \xrightarrow{dd} e$ and b is control dependent on a, $a \xrightarrow{cd} b$, since

- a is not strictly postdominated by b: path $[a..e] \not\ni b$
- in $]a..b[$, all nodes postdominated by b (vacuously)

Control Dependence (CD)

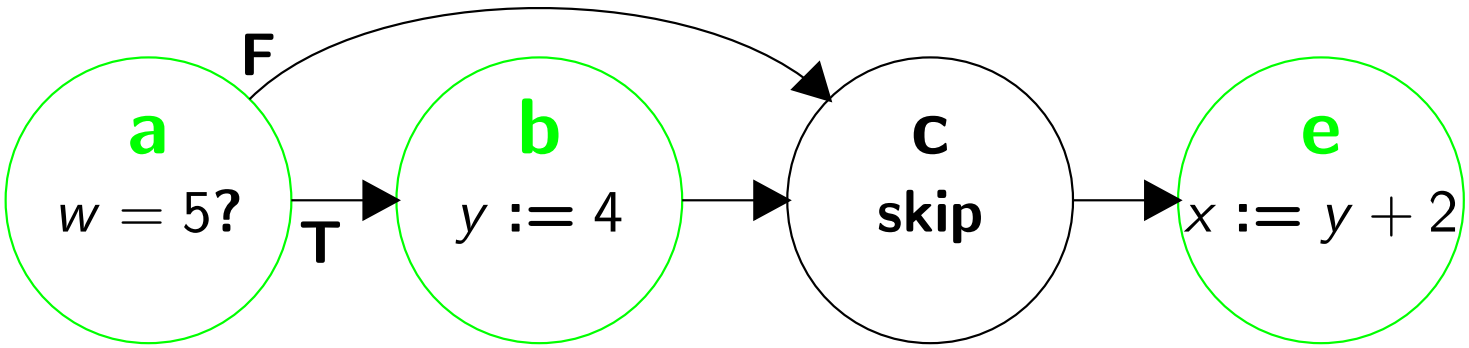
Source: final value of $x = 6$ if $w = 5$, else 2



$b \xrightarrow{dd} e$ and b is control dependent on a, $a \xrightarrow{cd} b$, since

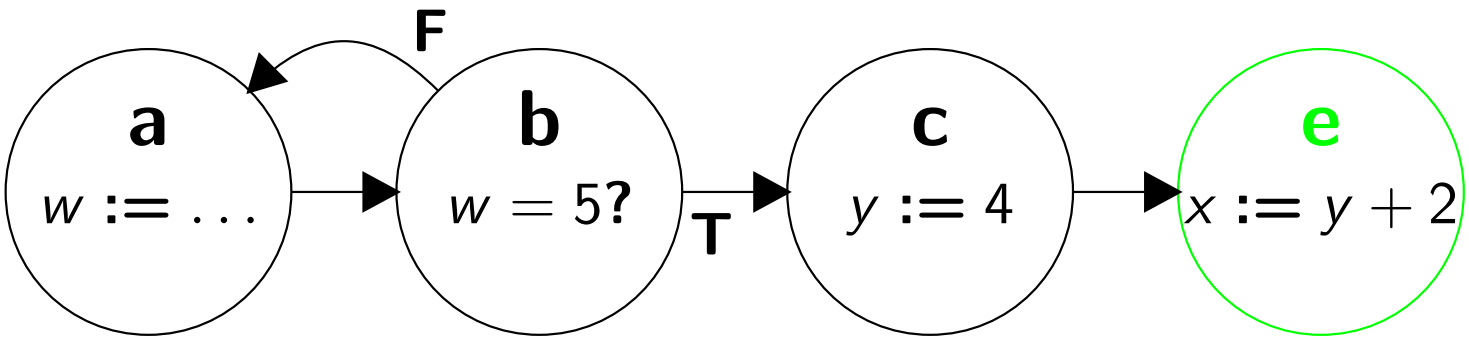
- a is not strictly postdominated by b: path $[a..e] \not\ni b$
- in $]a..b[$, all nodes postdominated by b (vacuously)

Slice: final value of $x = 6$ if $w = 5$, else 2



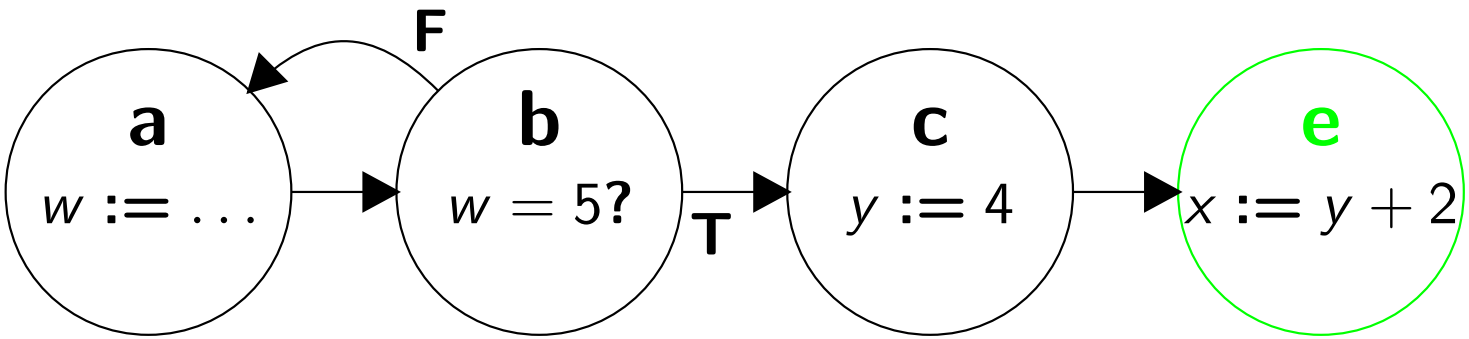
Weak Control Dependence (WCD)

Source: final value of $x = 6$ or $\perp$



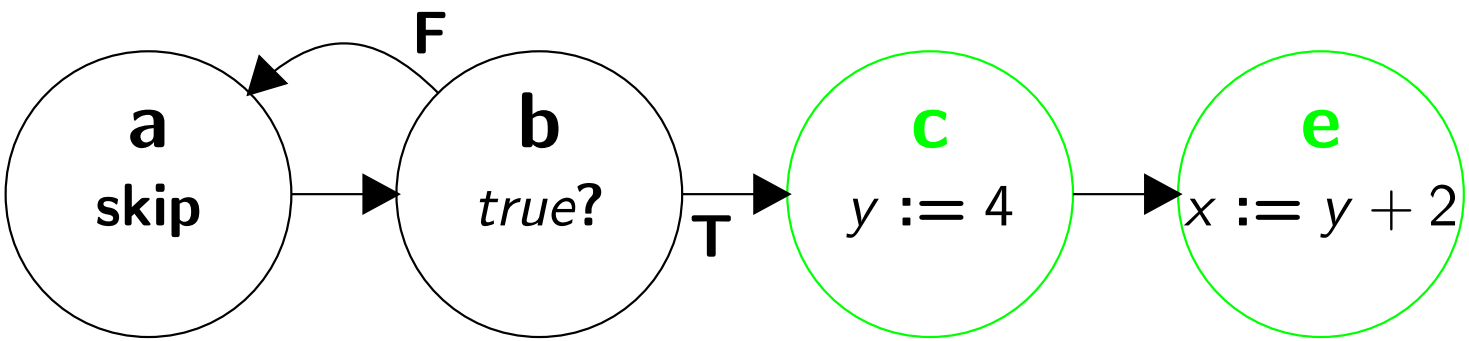
Weak Control Dependence (WCD)

Source: final value of $x = 6$ or $\bot$



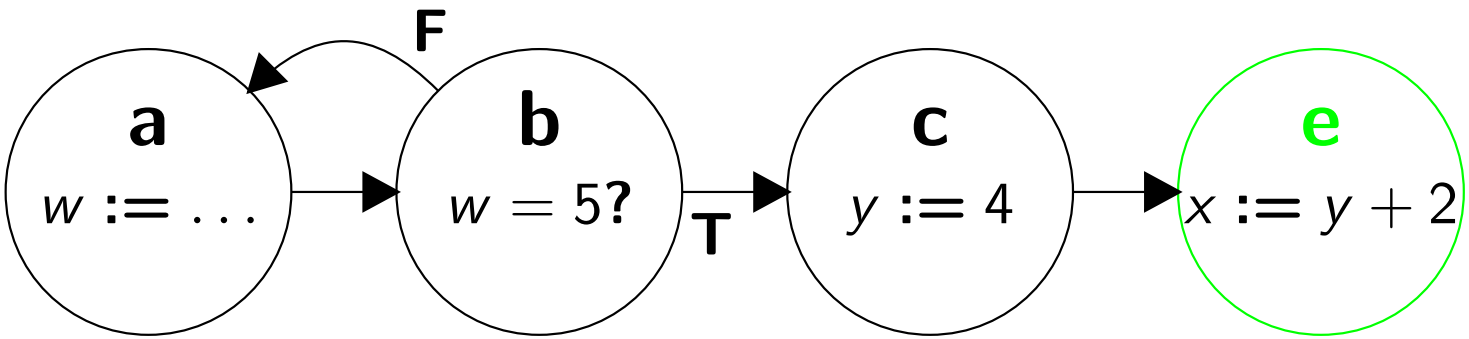
$c \xrightarrow{dd} e$ but $b \overset{cd}{\not\to} c$, since c strictly postdominates b

Slice: final value of $x = 6$



Weak Control Dependence (WCD)

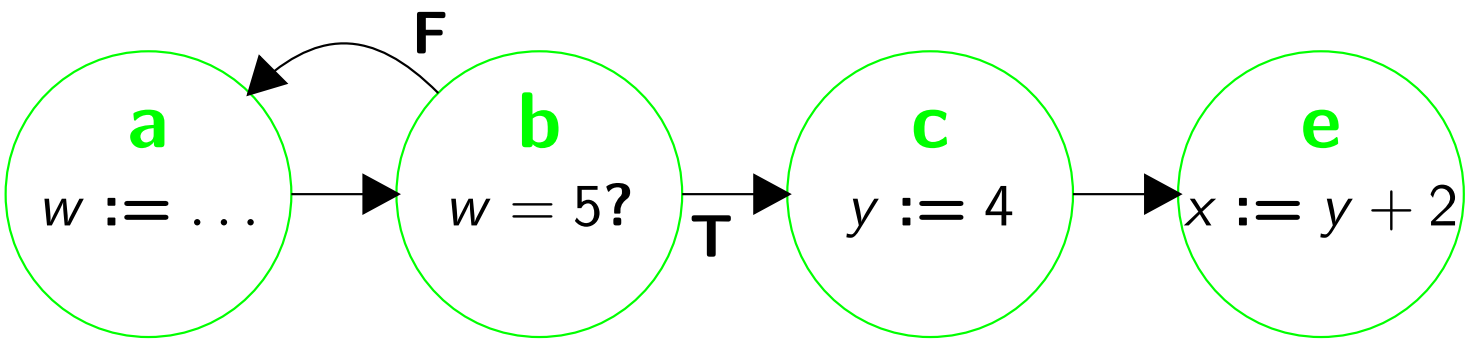
Source: final value of $x = 6$ or $\bot$



$c \xrightarrow{dd} e$ but $b \not\xrightarrow{cd} c$, since c strictly postdominates b

But c is weakly control dependent on b, $b \xrightarrow{wcd} c$ (though no $[b..e]$ bypasses c, b may cause indefinite delay of c)

Slice: final value of $x = 6$ or $\bot$



Removing End Node Assumption

- The definitions so far assume the existence of a (unique) end node.
- recall Challenge 1: extend the definition of control dependency to be applicable even to CFGs that may not have an end node

Removing End Node Assumption

- The definitions so far assume the existence of a (unique) end node.
- recall Challenge 1: extend the definition of control dependency to be applicable even to CFGs that may not have an end node

Ranganath et al [ESOP'05 & TOPLAS'07] proposed a new definition, saying that h is control dependent on a iff

- from one of a's successors, h cannot be avoided forever: all maximal paths go through h.
- from another of a's successors, h may be avoided forever: there exists a maximal path not going through h.

Removing End Node Assumption

- The definitions so far assume the existence of a (unique) end node.
- recall Challenge 1: extend the definition of control dependency to be applicable even to CFGs that may not have an end node

Ranganath et al [ESOP'05 & TOPLAS'07] proposed a new definition, saying that h is control dependent on a iff

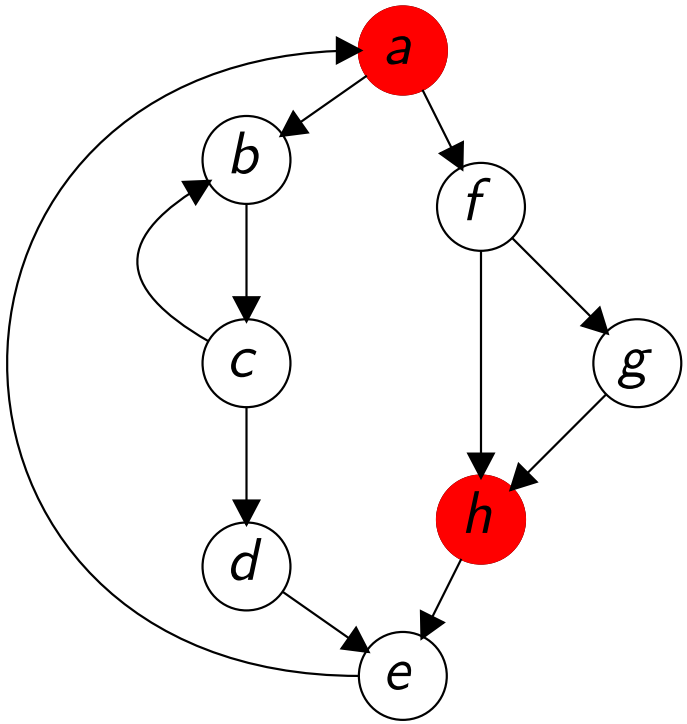
- from one of a's successors, h cannot be avoided forever: all maximal paths go through h.
- from another of a's successors, h may be avoided forever: there exists a maximal path not going through h.

This is called NTSCD (non-termination sensitive CD)

- Conservative Extension: in the special case where the CFG has an end node, NTSCD is equivalent to Podgurski & Clarke's weak control dependence.

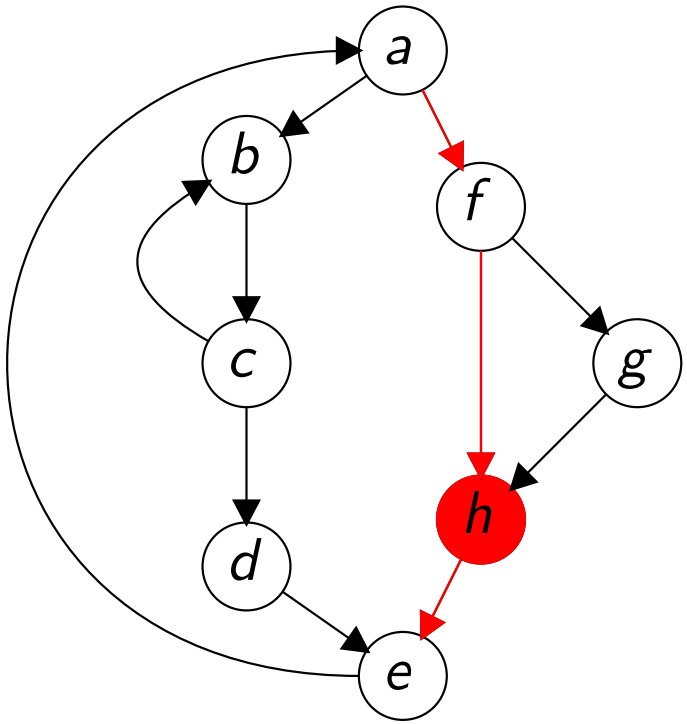
Illustrating NTSCD

h NTSCD on a



Illustrating NTSCD

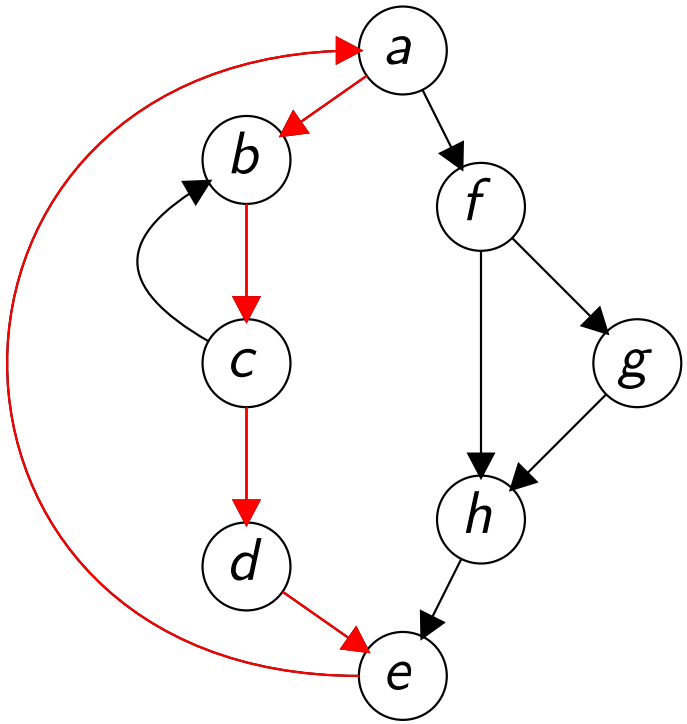
h NTSCD on a



If a selects f then h cannot be avoided

Illustrating NTSCD

h NTSCD on a

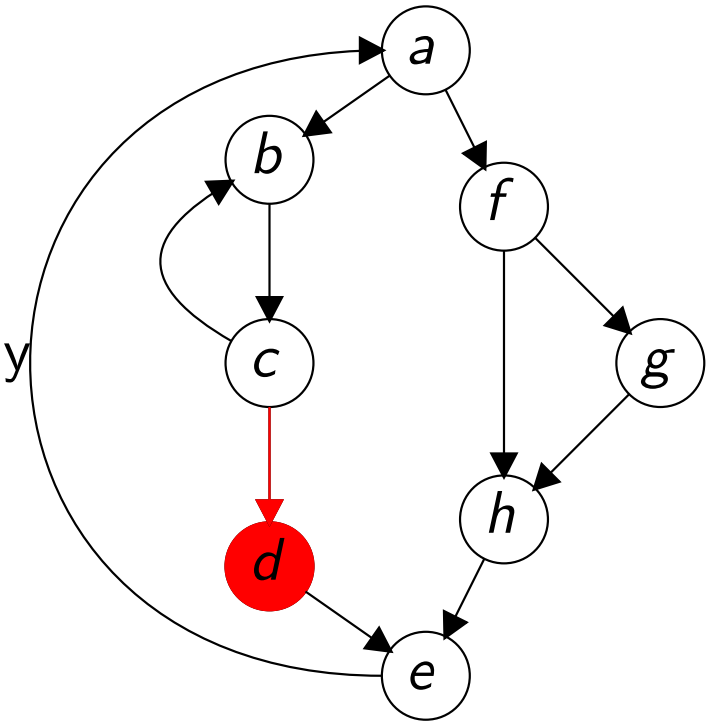


If a selects b then h can be avoided forever

Illustrating NTSCD

Termination is Preserved

d NTSD on c
so inner loop not sliced away

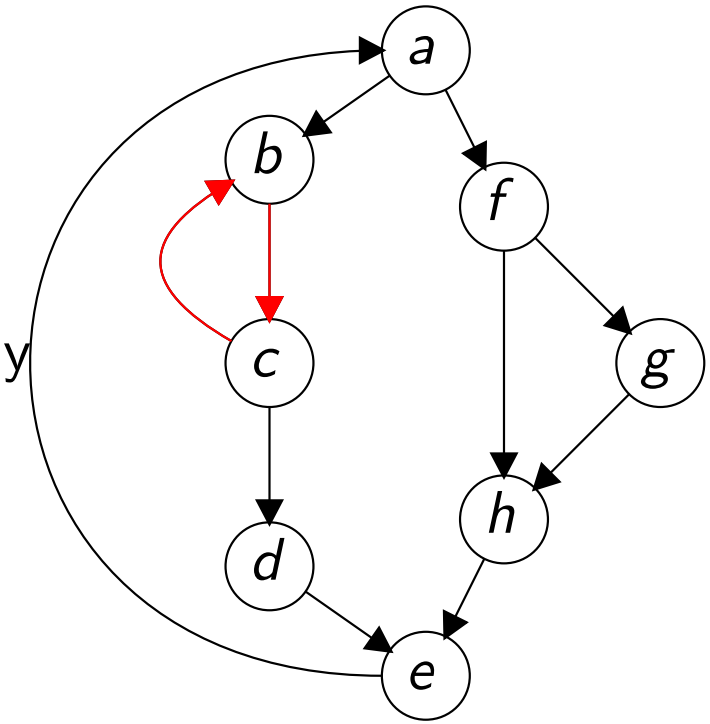


If c selects d then d cannot be avoided

Illustrating NTSCD

Termination is Preserved

d NTSD on c
so inner loop not sliced away



If c selects b then d can be avoided forever

Correctness of NTSCD

We aim at (weak) bisimulation:

- if a node can do an observable action in the source program, it can do so in the sliced program
- if a node can do an observable action in the sliced program, it can do so in the source program
- Also, for stores to be bisimilar, they must agree on "relevant" variables.

Correctness of NTSCD

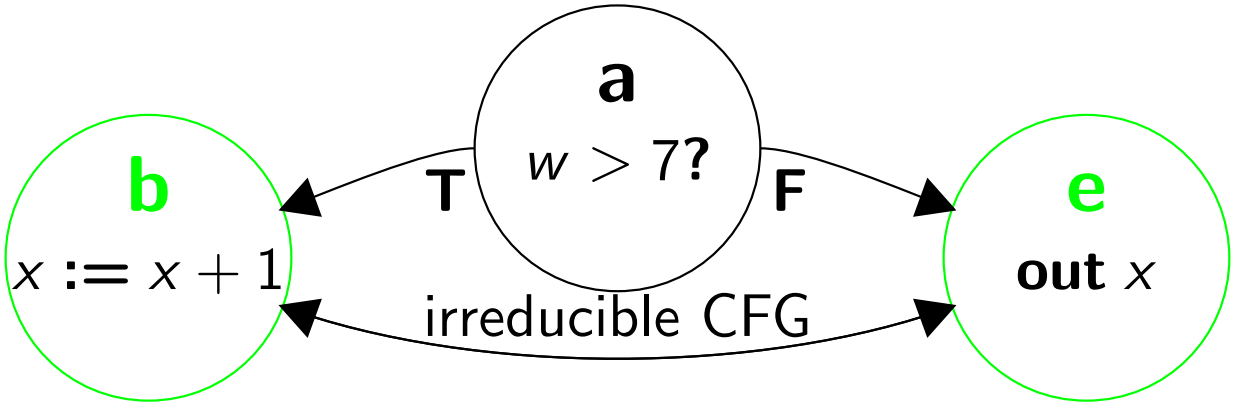
We aim at (weak) bisimulation:

- if a node can do an observable action in the source program, it can do so in the sliced program
- if a node can do an observable action in the sliced program, it can do so in the source program
- Also, for stores to be bisimilar, they must agree on "relevant" variables.

If the CFG is reducible, we can prove that slicing based on NTSCD gives rise to a bisimulation. Recall that a CFG is reducible if its edges can be split into

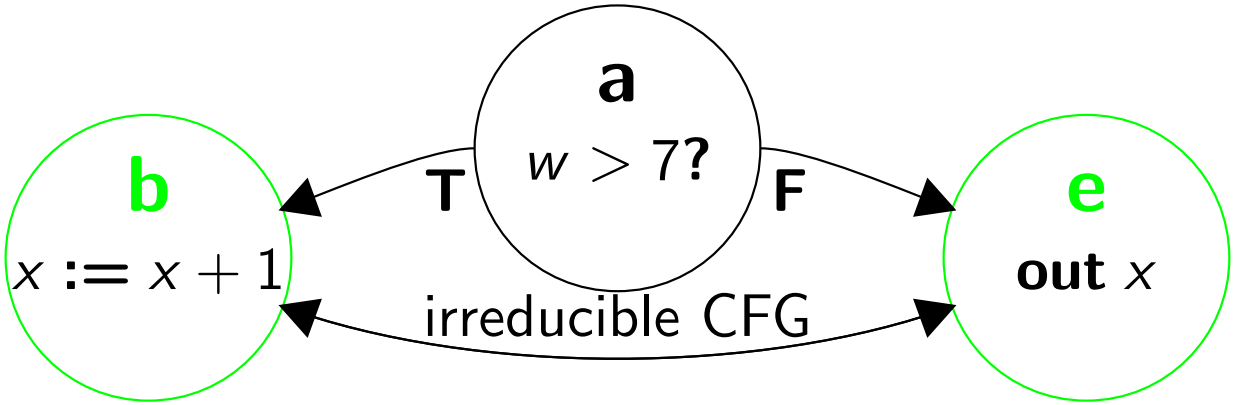
- forward edges which form a DAG;
- back edges where the target dominates the source

Irreducible Graphs



Observable behavior: 1,2,3,. . . if $w > 7$, else 0,1,2,. . .

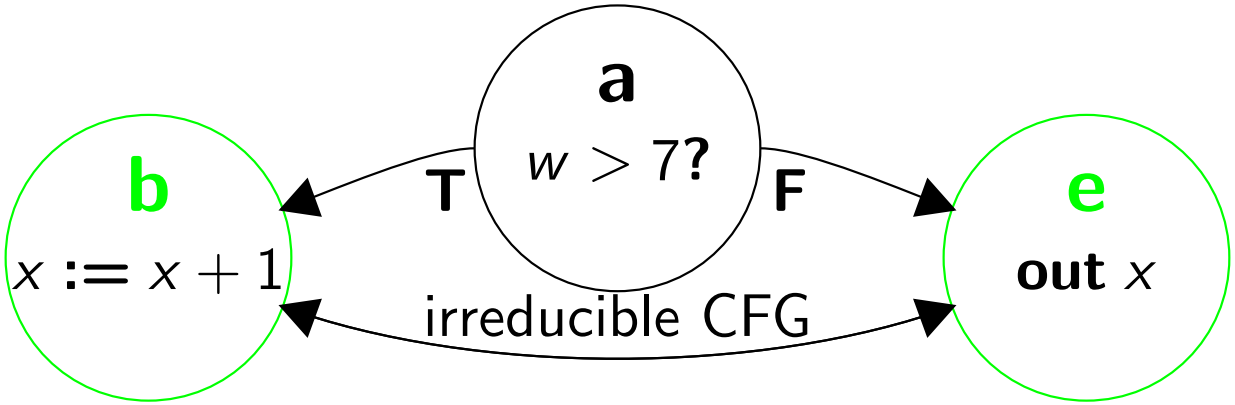
Irreducible Graphs



Observable behavior: 1,2,3,... if $w > 7$, else 0,1,2,...

- Thus a should be in slice set, but hard to see how $a \overset{?cd}{\longrightarrow} b$ or $a \overset{?cd}{\longrightarrow} e$ since all paths from a hit b and e.

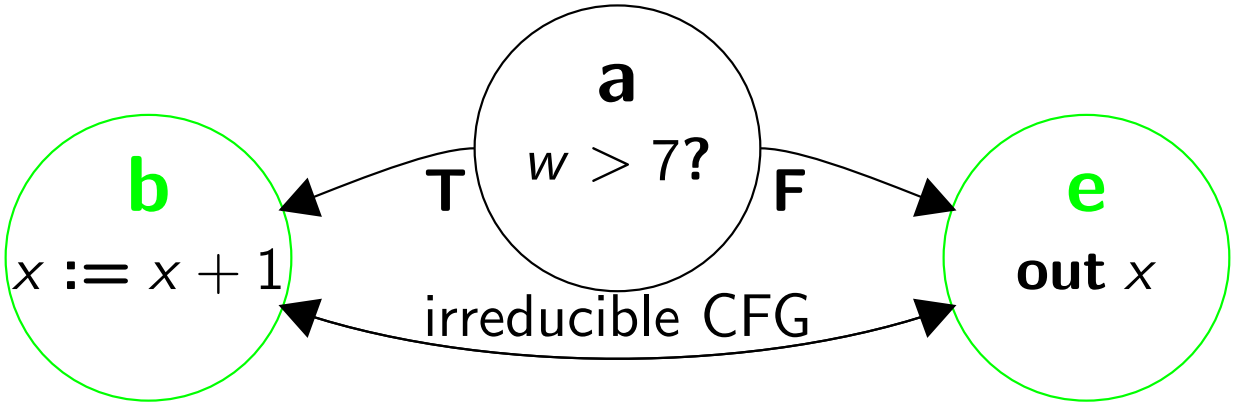
Irreducible Graphs



Observable behavior: 1,2,3,... if $w > 7$, else 0,1,2,...

- Thus a should be in slice set, but hard to see how $a \xrightarrow{?cd} b$ or $a \xrightarrow{?cd} e$ since all paths from a hit b and e.
- Instead we need a ternary relation, expressing that the order of b,c depends on a.

Irreducible Graphs



Observable behavior: 1,2,3,... if $w > 7$, else 0,1,2,...

- Thus a should be in slice set, but hard to see how $a \xrightarrow{?cd} b$ or $a \xrightarrow{?cd} e$ since all paths from a hit b and e.
- Instead we need a ternary relation, expressing that the order of b,c depends on a.

Theorem For an arbitrary CFG (with/without end nodes, reducible or not), the sliced program is bisimular to the source program, provided the slice set is closed under

- data dependency
- NTSCD
- a certain kind of "order dependency"

Assessment

The approach by Ranganath et al

- provides first theoretical foundation for slicing of CFGs that may have zero end nodes
- generalizes weak control dependence, and hence provides for termination-preserving slicing

Assessment

The approach by Ranganath et al

- provides first theoretical foundation for slicing of CFGs that may have zero end nodes
- generalizes weak control dependence, and hence provides for termination-preserving slicing

Slices are typically larger, as they must include all nodes that influence guards of potential loops

- great, if you are slicing to preserve liveness properties for model checking
- not so great, if slicing for program understanding

Assessment

The approach by Ranganath et al

- provides first theoretical foundation for slicing of CFGs that may have zero end nodes
- generalizes weak control dependence, and hence provides for termination-preserving slicing

Slices are typically larger, as they must include all nodes that influence guards of potential loops

- great, if you are slicing to preserve liveness properties for model checking
- not so great, if slicing for program understanding

We may like to discard nodes that may influence loops if they do not impact observable computation

- thus we should generalize $\stackrel{cd}{\longrightarrow}$ rather than $\stackrel{wcd}{\longrightarrow}$
- as was done by something called NTICD, but which was not helpful to establish the correctness of slicing

Weak Order Dependence (WOD)

Recall Challenge 2: find a dependence relation that

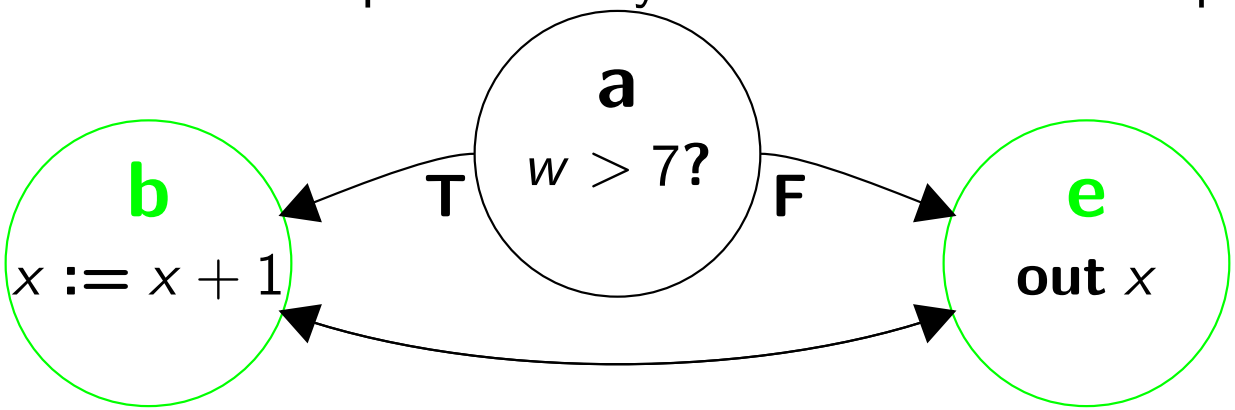
- slices away irrelevant loops
- does not assume the existence of an end node

Weak Order Dependence (WOD)

Recall Challenge 2: find a dependence relation that

- slices away irrelevant loops
- does not assume the existence of an end node

We can accomplish this by a variant of order dependency:



e & b weakly order dependent on a, $a \overset{wod}{\longrightarrow} b, e$,iff:

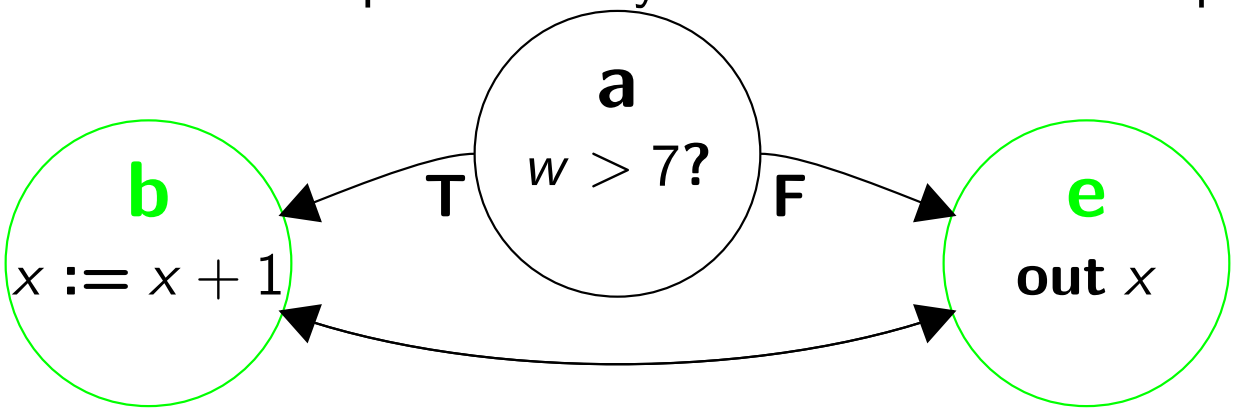
- path $[a..b] \not\ni e$ and path $[a..e] \not\ni b$
- a has successor x such that either
  - b is reachable from x and all $[x..e]$ contain b, or
  - e is reachable from x and all $[x..b]$ contain e.

Weak Order Dependence (WOD)

Recall Challenge 2: find a dependence relation that

- slices away irrelevant loops
- does not assume the existence of an end node

We can accomplish this by a variant of order dependency:



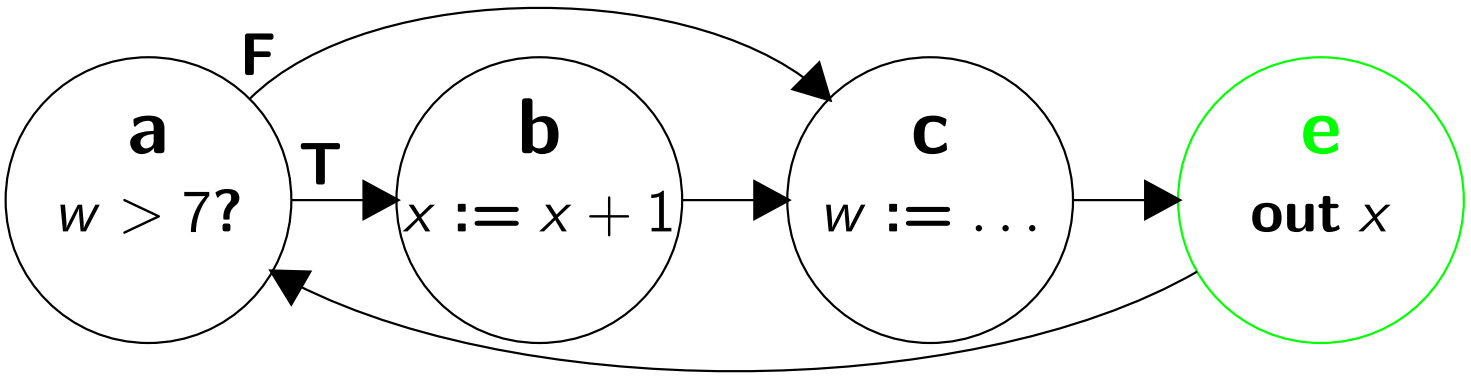
e & b weakly order dependent on a, $a \overset{wod}{\longrightarrow} b, e$,iff:

- path $[a..b] \not\ni e$ and path $[a..e] \not\ni b$
- a has successor x such that either
  - b is reachable from x and all $[x..e]$ contain b, or
  - e is reachable from x and all $[x..b]$ contain e.

We shall see that $\overset{wod}{\longrightarrow}$, though ternary, subsumes standard control dependency.

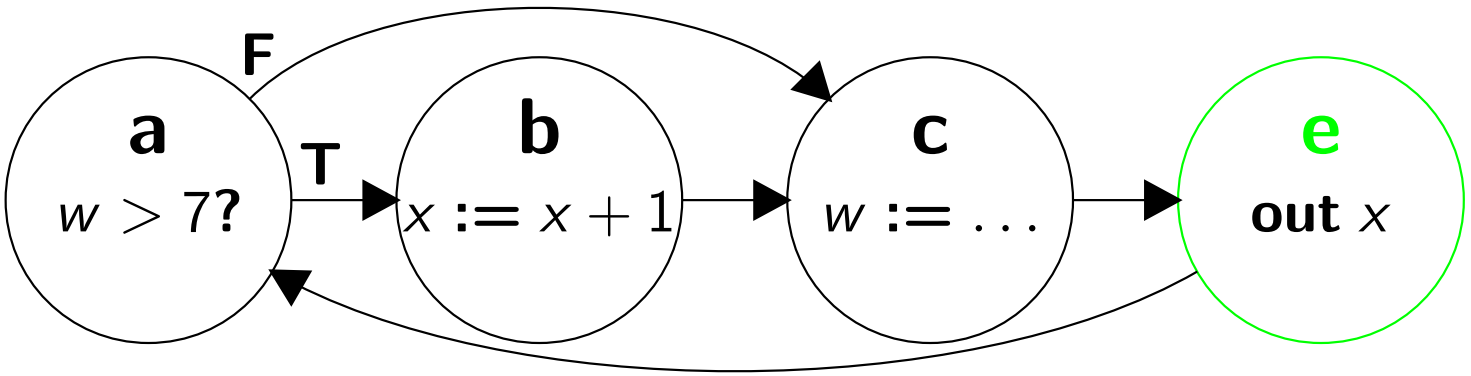
Example: WOD for Conditionals

Source: observable behavior of the form
1,1,2,2,2,3,4,4,. . .

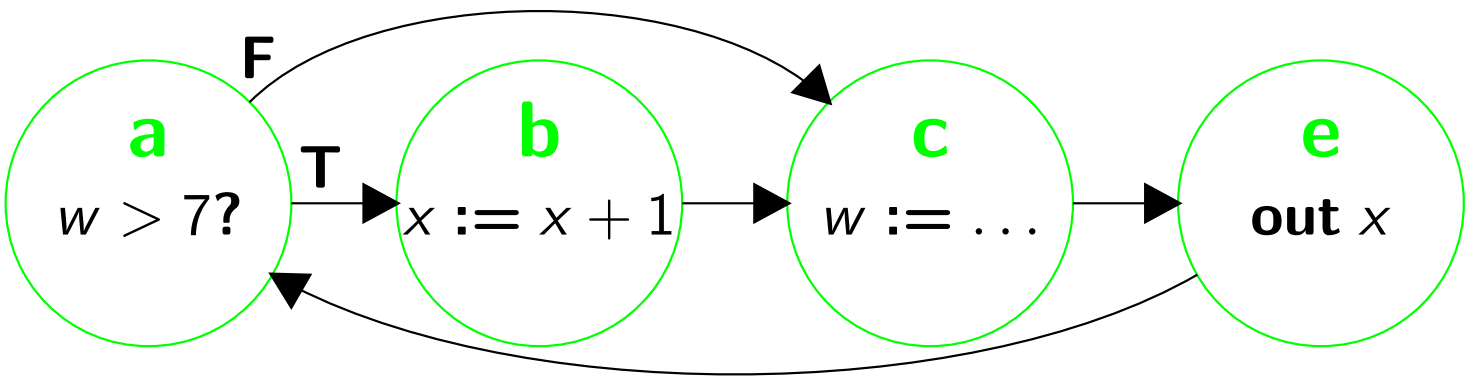


Example: WOD for Conditionals

Source: observable behavior of the form
1,1,2,2,2,3,4,4,. . .

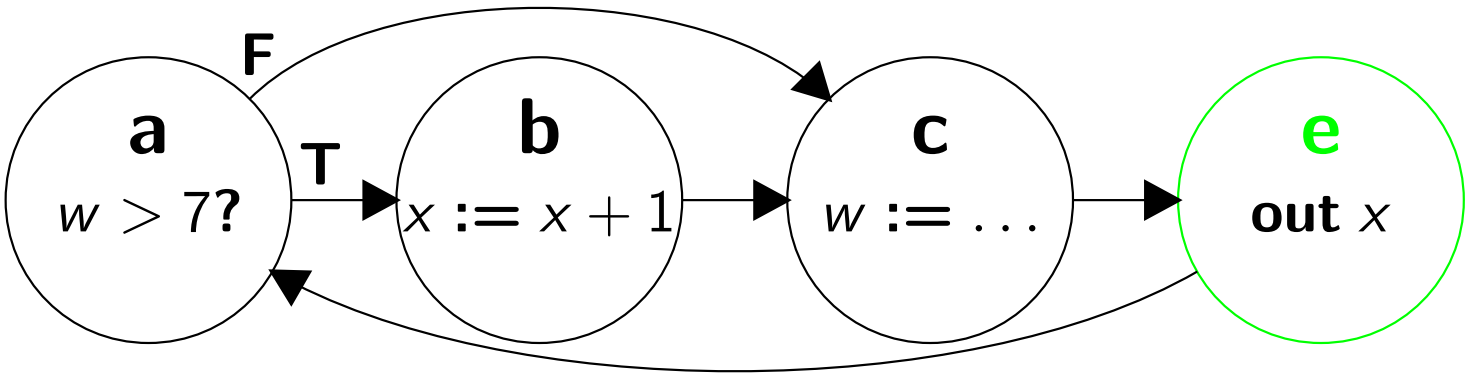


$b \overset{dd}{\rightarrow} e$ and $c \overset{dd}{\rightarrow} a \overset{wod}{\rightarrow} b, e$



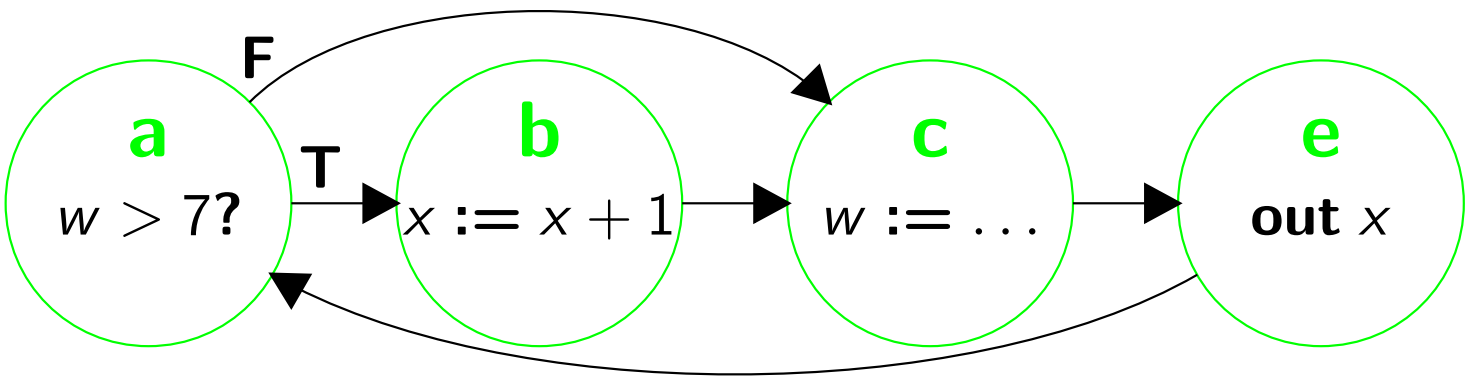
Example: WOD for Conditionals

Source: observable behavior of the form
1,1,2,2,2,3,4,4,...



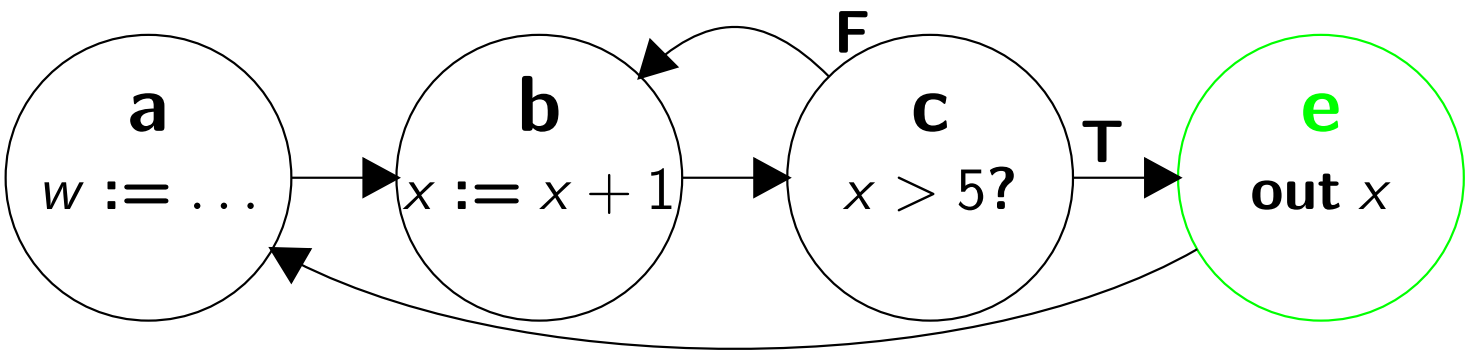
$b \overset{dd}{\rightarrow} e$ and $c \overset{dd}{\rightarrow} a \overset{wod}{\rightarrow} b, e$ since

- path $[a..b] \not\ni e$ and path $[a..e] \not\ni b$
- a has successor b with $[b..b]$ and all $[b..e]$ contain b



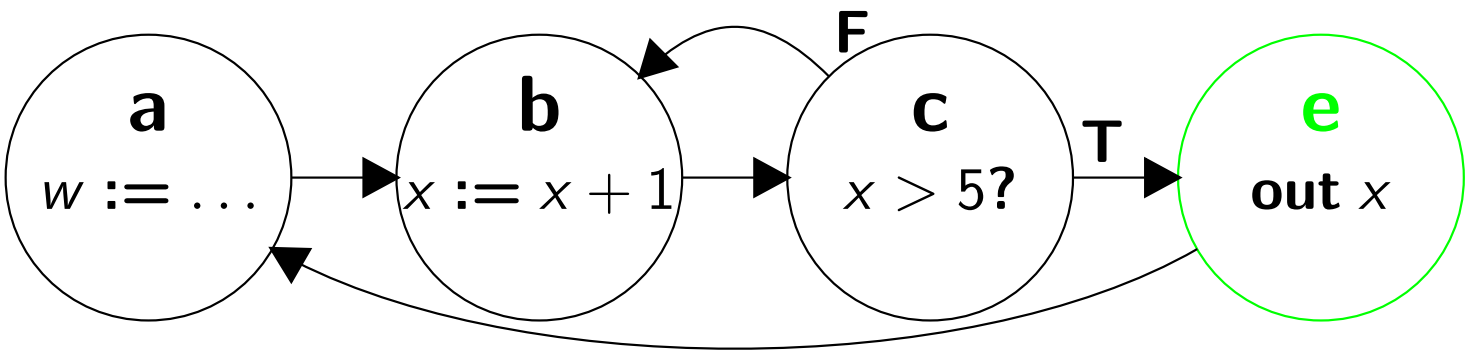
Example: WOD for Relevant Loops

Source: observable behavior 6,7,8,9,10,...



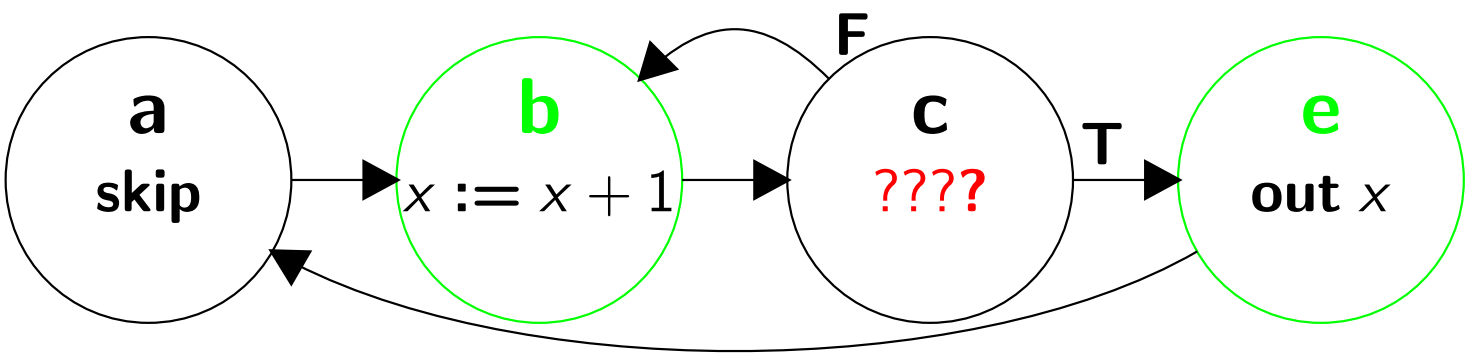
Example: WOD for Relevant Loops

Source: observable behavior 6,7,8,9,10,...



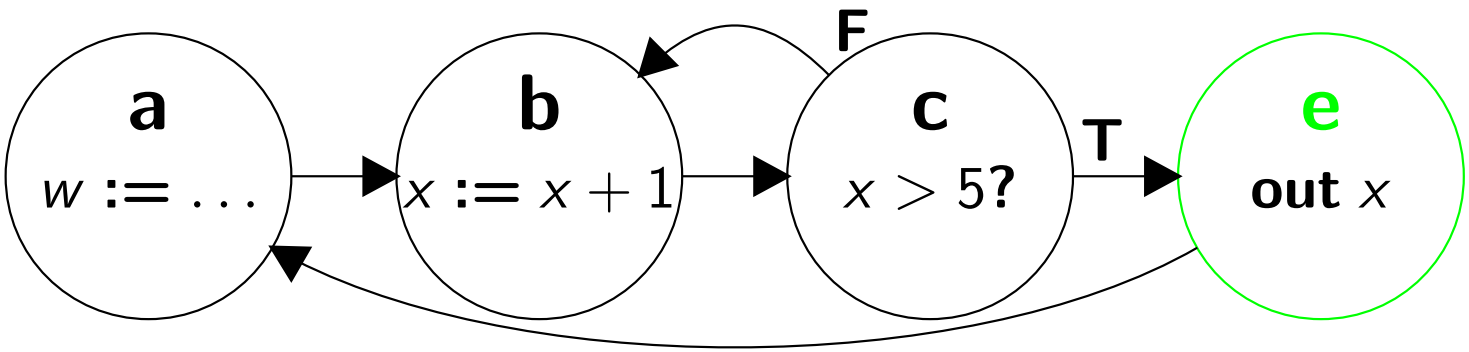
Slice: observable behavior if guard wrongly sliced away:

- 1,2,3,4,5,... if replaced by *true*
- ϵ if replaced by *false*



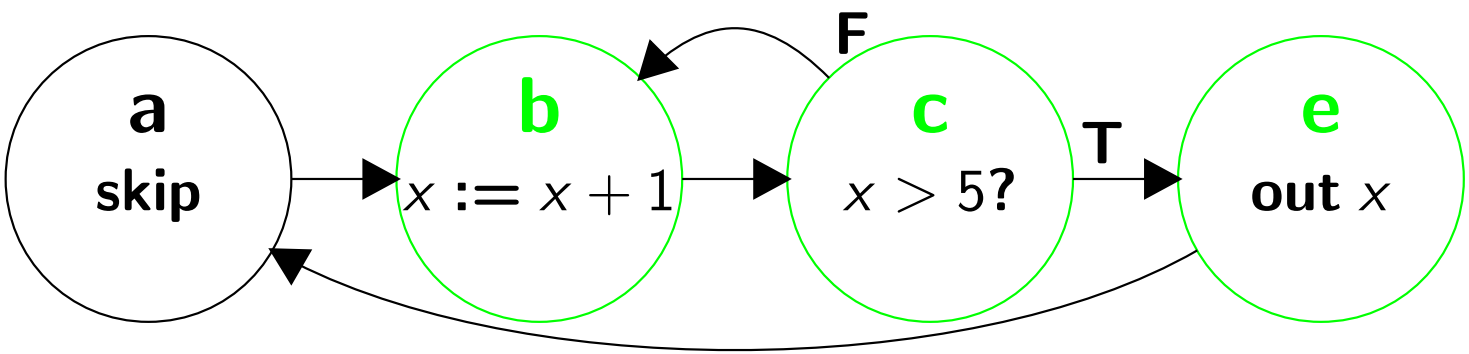
Example: WOD for Relevant Loops

Source: observable behavior 6,7,8,9,10,...



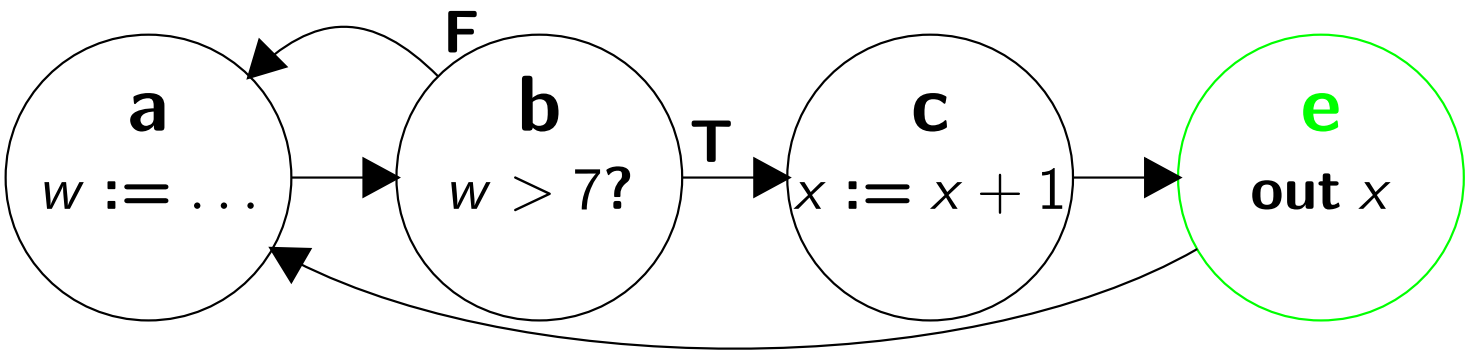
$c \overset{wod}{\rightarrow} b, e$ since

- path $[c..b] \not\ni e$ and path $[c..e] \not\ni b$
- c has successor b with $[b..b]$ and all $[b..e]$ contain b



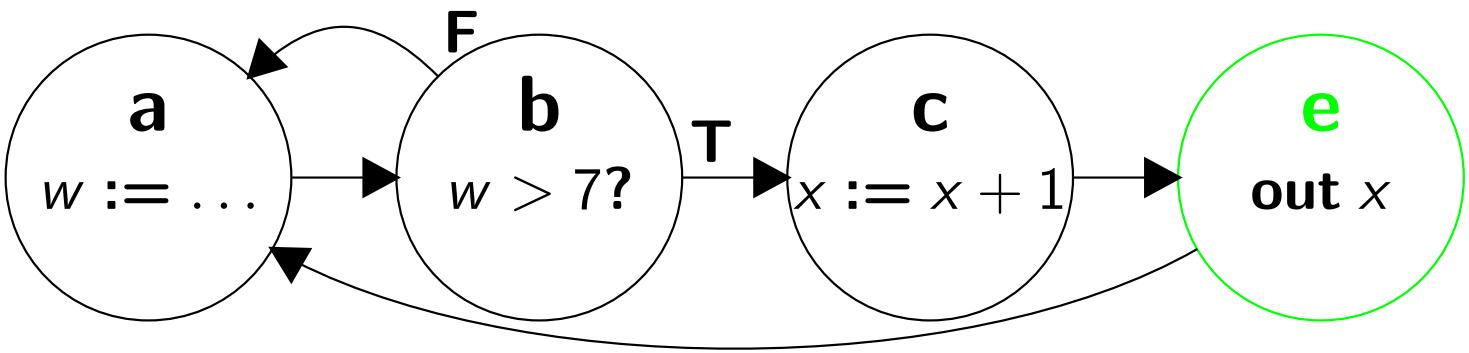
Example: WOD for Irrelevant Loops

Source: observable behavior 1,2,3,4,. . . or prefix thereof

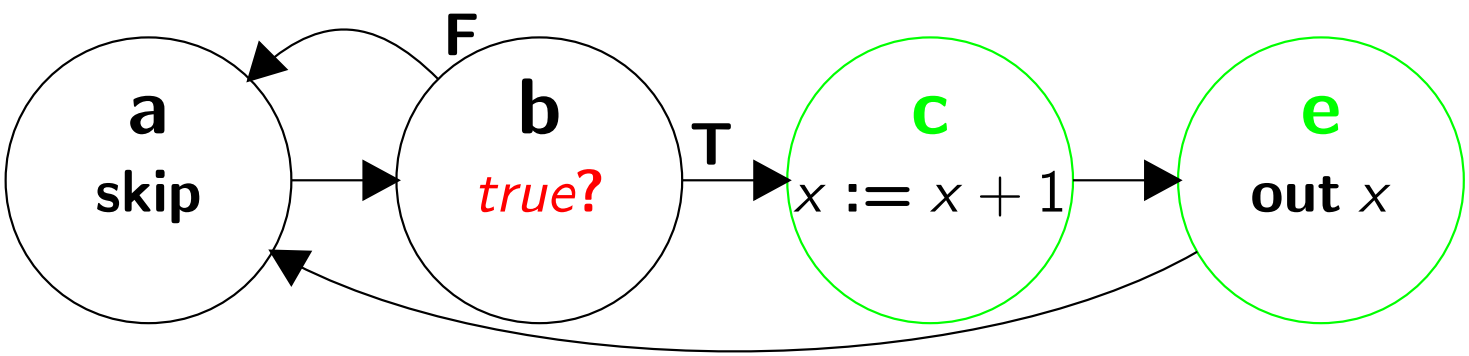


Example: WOD for Irrelevant Loops

Source: observable behavior 1,2,3,4,. . . or prefix thereof

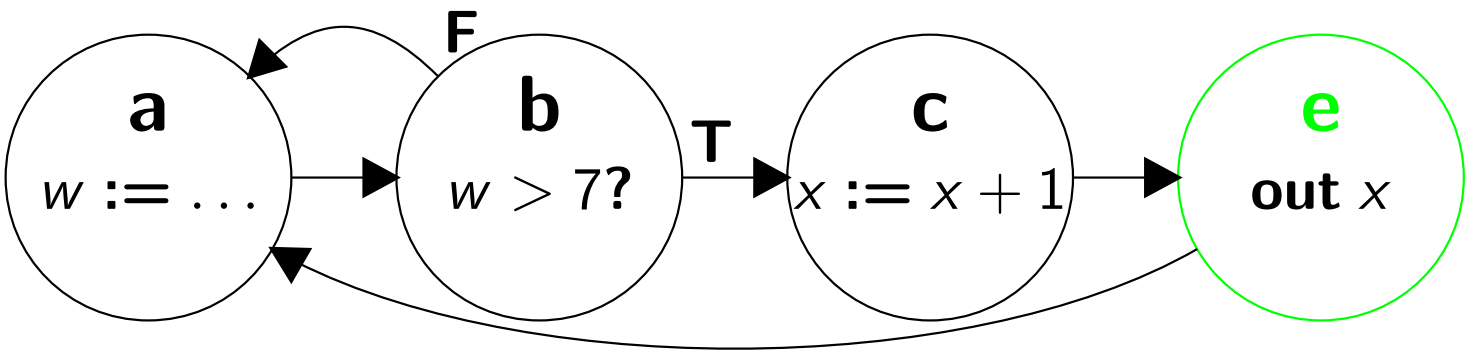


New guard chosen to get closer to next observable.
Slice: observable behavior 1,2,3,4,. . .



Example: WOD for Irrelevant Loops

Source: observable behavior 1,2,3,4,. . . or prefix thereof

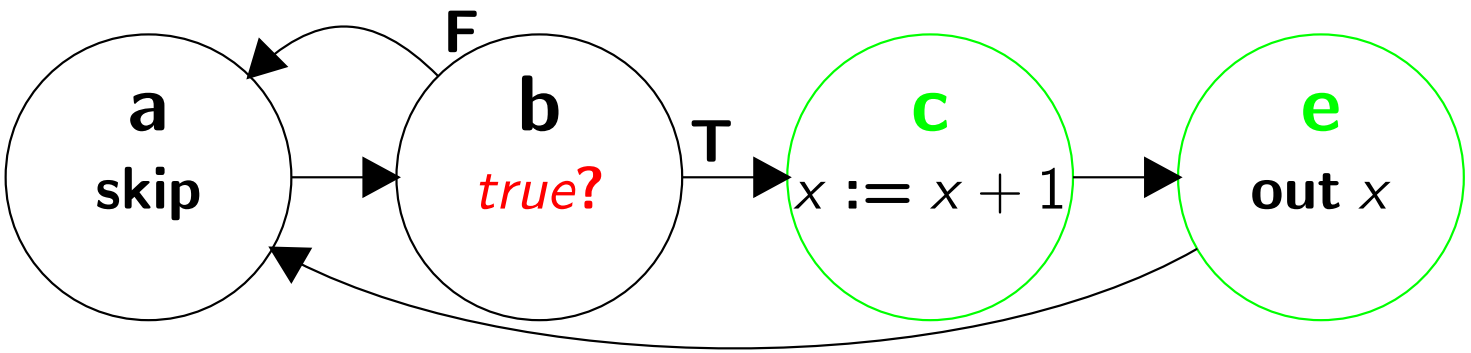


$b \overset{wod}{\not\mapsto} c, e$ since

- no path $[b..e] \not\ni c$

New guard chosen to get closer to next observable.

Slice: observable behavior 1,2,3,4,. . .



Relating WOD to CD

Conservative extension For a CFG with an end node which is part of slicing criterion, the closures of $\overset{wod}{\longrightarrow}$ and $\overset{cd}{\longrightarrow}$ coincide.

Relating WOD to CD

Conservative extension For a CFG with an end node which is part of slicing criterion, the closures of $\overset{wod}{\longrightarrow}$ and $\overset{cd}{\longrightarrow}$ coincide.

Alternative characterization Current work with Mark Harman et al from King's College captures the intution that we need a ternary relation because of one the three nodes must play the role as "local pseudo exit node":

Relating WOD to CD

Conservative extension For a CFG with an end node which is part of slicing criterion, the closures of $\overset{wod}{\longrightarrow}$ and $\overset{cd}{\longrightarrow}$ coincide.

Alternative characterization Current work with Mark Harman et al from King's College captures the intution that we need a ternary relation because of one the three nodes must play the role as "local pseudo exit node":

- Assume a CFG where all nodes are reachable from each other (as is often the case)
- for each node x, if we remove all outgoing edges from x, we get a graph $G_{\rightarrow x}$ having x as end node.

Relating WOD to CD

Conservative extension For a CFG with an end node which is part of slicing criterion, the closures of $\overset{wod}{\longrightarrow}$ and $\overset{cd}{\longrightarrow}$ coincide.

Alternative characterization Current work with Mark Harman et al from King's College captures the intution that we need a ternary relation because of one the three nodes must play the role as "local pseudo exit node":

- Assume a CFG where all nodes are reachable from each other (as is often the case)
- for each node x, if we remove all outgoing edges from x, we get a graph $G_{\rightarrow x}$ having x as end node.
- If $a \overset{wod}{\longrightarrow} b, c$ in G then either
  - $a \overset{cd}{\longrightarrow} b$ in $G_{\rightarrow c}$, or
  - $a \overset{cd}{\longrightarrow} c$ in $G_{\rightarrow b}$.

Relating WOD to CD

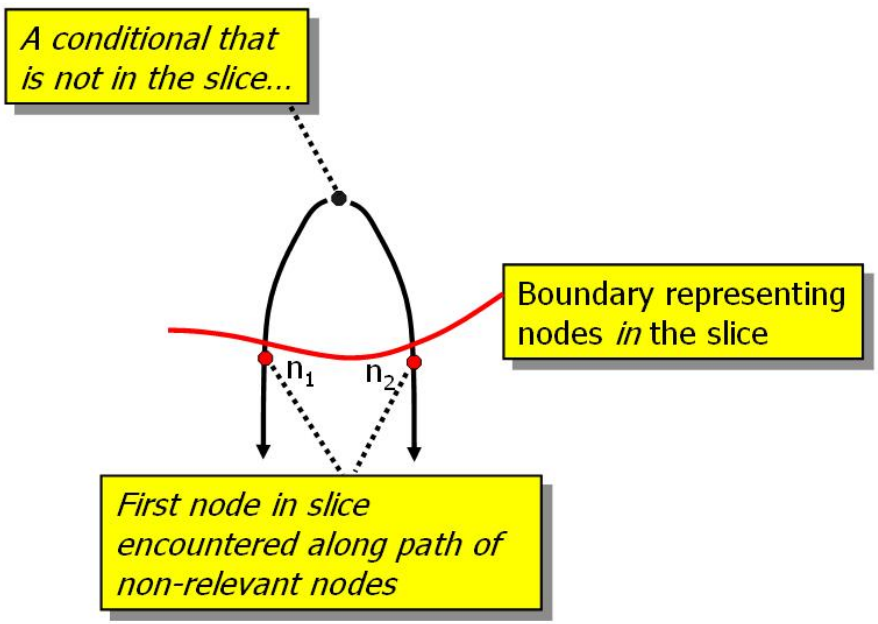
Conservative extension For a CFG with an end node which is part of slicing criterion, the closures of $\overset{wod}{\longrightarrow}$ and $\overset{cd}{\longrightarrow}$ coincide.

Alternative characterization Current work with Mark Harman et al from King's College captures the intution that we need a ternary relation because of one the three nodes must play the role as "local pseudo exit node":

- Assume a CFG where all nodes are reachable from each other (as is often the case)
- for each node x, if we remove all outgoing edges from x, we get a graph $G_{\rightarrow x}$ having x as end node.
- If $a \overset{wod}{\longrightarrow} b, c$ in G then either
  - $a \overset{cd}{\longrightarrow} b$ in $G_{\rightarrow c}$, or
  - $a \overset{cd}{\longrightarrow} c$ in $G_{\rightarrow b}$.
- if $a \overset{cd}{\longrightarrow} c$ in $G_{\rightarrow b}$ with $a \neq c$ then $a \overset{wod}{\longrightarrow} b, c$ in G.

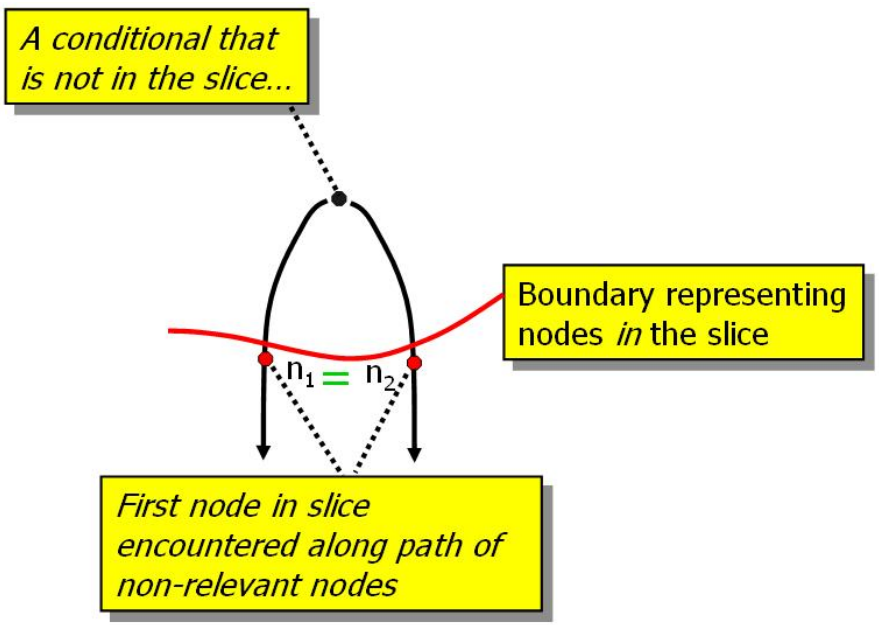
Unique Next Observable

- For the correctness proof, we shall assume the set of observables is closed under $\overset{dd}{\longrightarrow}$ and $\overset{wod}{\longrightarrow}$.
- Key advantage of being closed under $\overset{wod}{\longrightarrow}$: at most one next observable.



Unique Next Observable

- For the correctness proof, we shall assume the set of observables is closed under $\overset{dd}{\longrightarrow}$ and $\overset{wod}{\longrightarrow}$.
- Key advantage of being closed under $\overset{wod}{\longrightarrow}$: at most one next observable.



Unique Next Observable

- For the correctness proof, we shall assume the set of observables is closed under $\overset{dd}{\longrightarrow}$ and $\overset{wod}{\longrightarrow}$.
- Key advantage of being closed under $\overset{wod}{\longrightarrow}$: at most one next observable.

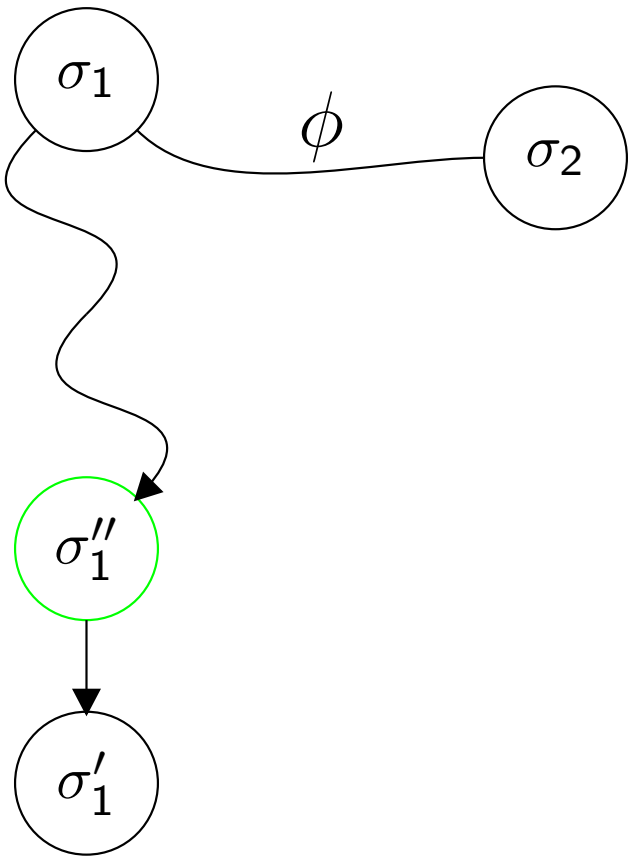
Recent classification [Sebastian Danicic]: all slices in the literature computes one of the two below:

- the least set where all nodes have at most one next observable.
  This is weak commitment, as computed by $\overset{cd}{\longrightarrow}$, and for general CFGs by $\overset{wod}{\longrightarrow}$.
- the least set where all nodes have at most one next observable, and satisfy that all nodes either
  - eventually will reach an observable, or
  - always avoids an observable

  This is strong commitment, as computed by $\overset{wcd}{\longrightarrow}$, and for general CFGs by $\overset{ntscd}{\longrightarrow} + \overset{\text{“}dod\text{”}}{\longrightarrow}$.

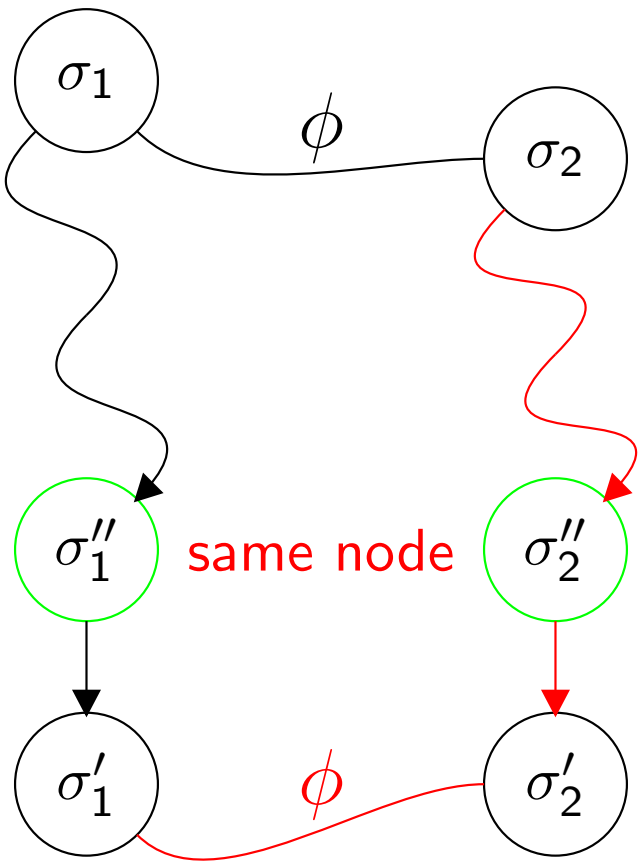
Weak Simulation

- Slicing Correctness expressed using weak simulation
- A weak simulation relates source states to slice states
- A program state is an node and a store



Weak Simulation

- Slicing Correctness expressed using weak simulation
- A weak simulation relates source states to slice states
- A program state is an node and a store



Relevant Variables

A variable x is relevant at n if

- x is used at an observable node q
- there exists a path $[n..q[$ along which x is not defined

Relevant Variables

A variable x is relevant at n if

- x is used at an observable node q
- there exists a path $[n..q[$ along which x is not defined

Facts:

- this definition is independent of whether one considers the source or the slice
- if two nodes have the same next observable, they have the same relevant variables

Defining a Concrete Weak Simulation

We define $(n_1, \sigma_1)\ R\ (n_2, \sigma_2)$ if

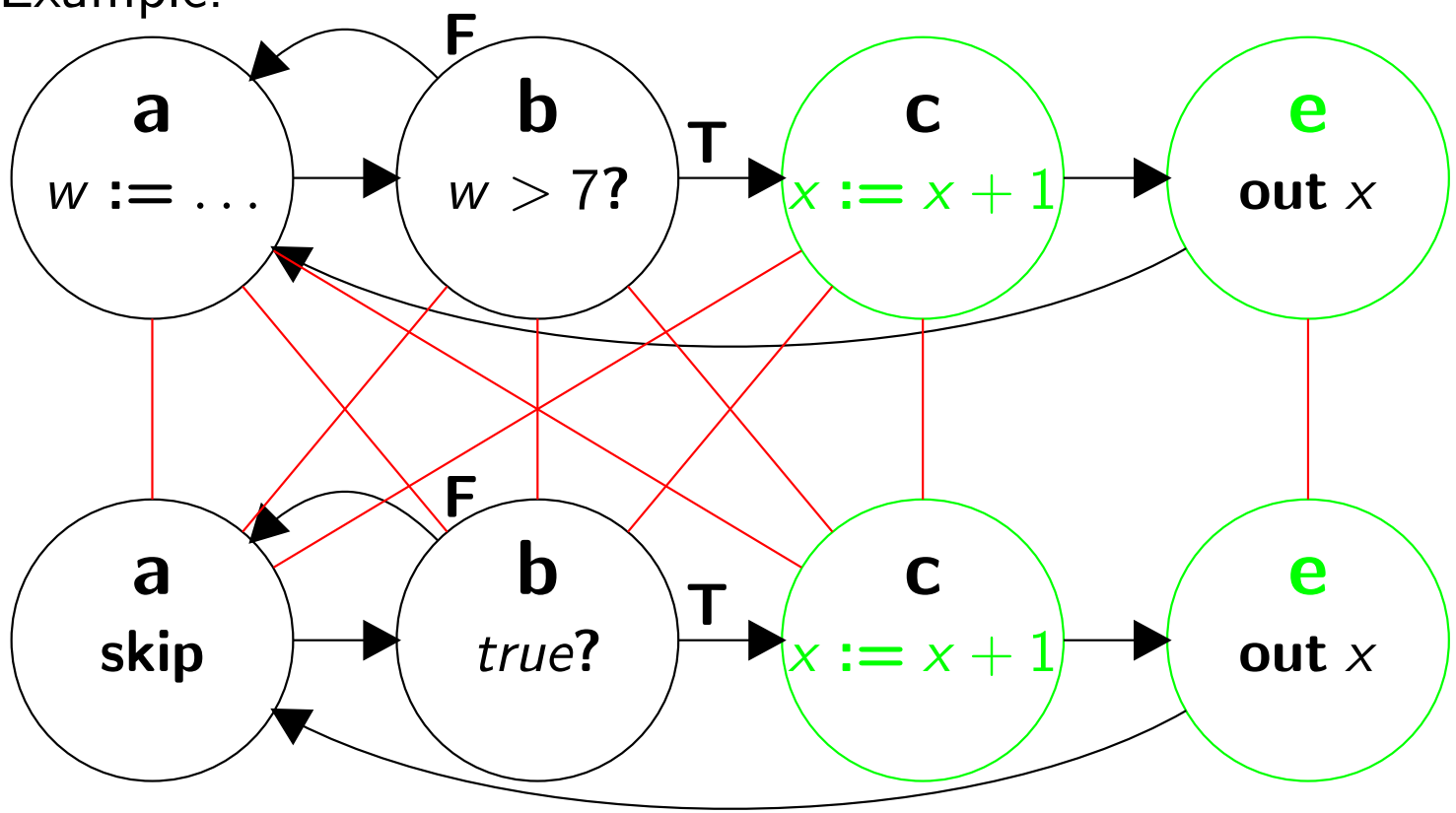
- n_1 and n_2 have the same next observable
- σ_1 and σ_2 agree on the relevant variables

Defining a Concrete Weak Simulation

We define $(n_1, \sigma_1)\ R\ (n_2, \sigma_2)$ if

- n_1 and n_2 have the same next observable
- σ_1 and σ_2 agree on the relevant variables

Example:



Correctness

Theorem

- R is a weak simulation.

Thus any observable action by the source can be matched by the slice, but not necessarily vice versa.

Related Work

Ball & Horwitz, 1993

- assumes unique end node
- correctness expressed using pointwise history
- one history is allowed to be prefix of other

Hatcliff et al, HOSC 2000

- assumes unique end node
- used for model checking, implying
  - sequencing of observables must be preserved
  - liveness properties must be preserved
- detailed correctness proof, assuming termination of both source and slice

Hatcliff et al, SAS 1999

- considers multi-threading
- proposes bisimulation as correctness property
- does not work out the details

Assessment

Foundation for slicing which

- handles CFGs that arise in modeling reactive systems
- produces manageable slices
- allows crisp correctness proof

Assessment

Foundation for slicing which

- handles CFGs that arise in modeling reactive systems
- produces manageable slices
- allows crisp correctness proof

Open question: Is (bi)simulation a strictly stronger correctness criterion than "pointwise" (bi)simulation?

If so, we might modify $\overset{wod}{\longrightarrow}$ to allow for independent assignments to be swapped.

Future Work

- Interprocedural analysis
- Multi-threading
- Connect to information flow analysis (Hammer & Krinke & Snelting)