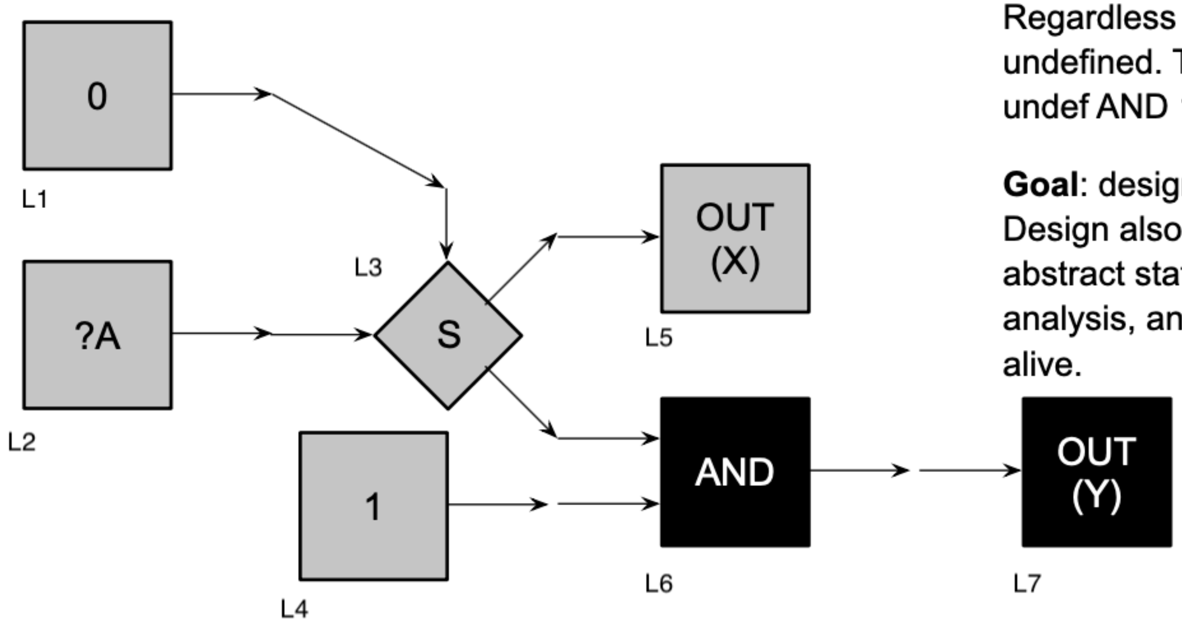
ather, Midsem 2022 dcc888

Question 1

Initial working q1

> **Question 1 [5 Points]** Programs might contain dead code. To see why, consider the program below:
>
> 
>
>
> Regardless of the value of A, the output Y will be always undefined. The AND function is also undefined, because undef AND 1 is always undef.
>
> **Goal**: design a static analysis that marks gates as DEAD. Design also a query IS_DEAD(A), which receives the abstract state of a variable A, as computed by your static analysis, and reports if A MUST be dead or if it MIGHT be alive.

Forward analysis. Must analysis.

The lattice is of $Scope = INPUTS \cup \{0, 1, \bot\}$ with these operations for evaluation $State = Vars \rightarrow Scope$,

- $eval(\sigma, op(Y, X)) = op(eval(\sigma, Y), eval(\sigma, X))$
- $eval(\sigma, S(Y, X)) = (snd|fst)(S(eval(\sigma, Y), eval(\sigma, X)))$
- $eval(\sigma, op(X)) = op(eval(\sigma, X))$
- $eval(\sigma, INPUT(X)) = X$
- $eval(\sigma, INPUT(N, X)) = (N > 0 \implies X) \vee (N \leq 0 \implies \bot)$
- $eval(\sigma, OUTPUT(X)) = X$

op:

- $AND(Y, X)$
    - $AND(0, X) = 0$
    - $AND(1, X) = X$
- $OR(Y, X)$
    - $OR(0, X) = X$
    - $OR(1, X) = 1$
- $S(X, CONTROL) = \{\neg CONTROL \implies [X, \bot], CONROL \implies [\bot, X]\}$
    - $S(X, 0) = [X, \bot]$
    - $S(X, 1) = [\bot, X]$
- $NOT(X) = \neg X$

$Functions:$

- $JOIN(v) = \sqcap_{w \in pred(v)} [\![w]\!]$
- $op(Y, X) : [\![v]\!] = JOIN(v)[eval(JOIN(v), op(Y, X))]$
- $op(X) : [\![v]\!] = JOIN(v)[(eval(JOIN(v), op(X)))]$

Workthrough q1

$[\![L7]\!] = [\![L6]\!] = \perp$
$[\![L6]\!] = snd(AND([\![L4]\!], [\![L3]\!])) = snd(AND(1, [\![L3]\!])) = snd([\![L3]\!]) = snd([A, \perp]) = \perp$
$[\![L4]\!] = INPUT(1) = 1$
$[\![L3]\!] = S([\![L2]\!], [\![L1]\!]) = S(A, [\![L1]\!]) = S(A, 0) = [A, \perp]$
$[\![L2]\!] = INPUT(A) = A$
$[\![L1]\!] = 0$
$[\![L5]\!] = fst([\![L3]\!]) = fst([A, \perp]) = A$

IS_DEAD(V) q1

$IS\_DEAD(V) = if([\![V]\!] == \perp)\ then\ TRUE\ else\ FALSE$

Time & Space q1

IS_DEAD is just time: O(1) & space: O(1) if constructed abstrace states.

Time q1

O(E) for E nodes, must visit each once.

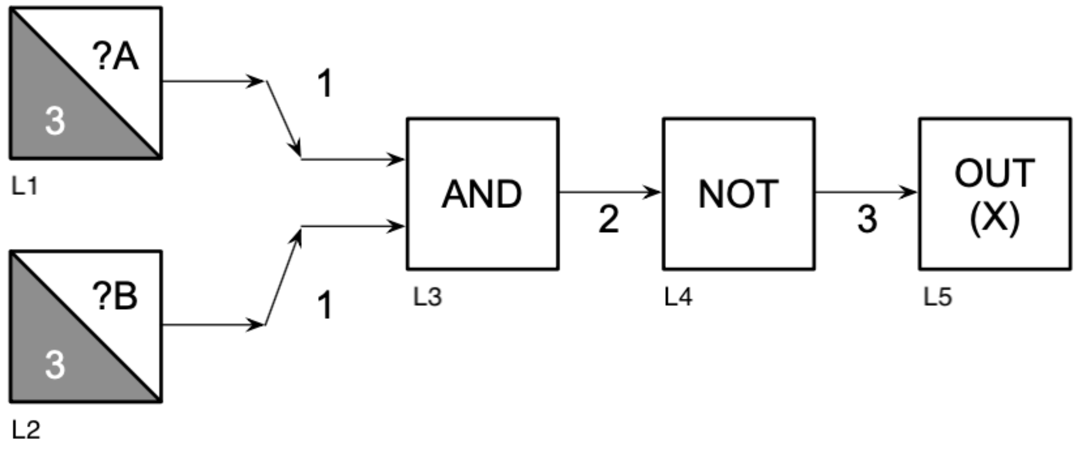
Space q1

Just contains the abstract value of the Node, so O(1) for each Node hence O(E) for E nodes.

Question 2

Initial working q2

Question 2 [5 Points] Edges are *undefined* if they receive an undef value. Edges that hold either zero or one are said to be *well-defined*. Circuits might start with undefined edges, which eventually become well-defined. For instance, in the program below, we show, associated with each edge, the first tick when that edge becomes well-defined.



Goal: design a static analysis to predict the moment when each edge becomes well-defined. If your analysis says that the edge will never become well defined, you must bind that edge to the abstract state INFTY. Otherwise, you must bind that edge to the abstract state [N], whereas N is the first tick when the edge becomes well defined. Design a query WAKE_UP(A), that receives the abstract state of a variable A, and informs the moment when the variable A first became well-defined.

Forward analysis. Must analysis.

$(StartT, MaxT)$, means starts being well-defined at $StartT$ till $StartT + MaxT$ clock tick.

The lattice is of $Scope = (flat(\mathrm{N}) \times$INPUTS_START_N$)$ with these operations for evaluation $State = Vars \rightarrow Scope$,

- $eval(\sigma, op(Y, X)) = op(eval(\sigma, Y), eval(\sigma, X))$
- $eval(\sigma, S(Y, X)) = (snd|fst)(S(eval(\sigma, Y), eval(\sigma, X)))$
- $eval(\sigma, op(X)) = op(eval(\sigma, X))$
- $eval(\sigma, INPUT(X)) = (0, \infty)$
- $eval(\sigma, INPUT(N, X)) = (N > 0 \implies (0, N)) \vee (N \leq 0 \implies (\infty, 0))$

- $eval(\sigma, OUTPUT(XS, XM)) = (XS + 1, XM)$

op:

```
CALC_RANGE((YS, YM), (XS, XM)) =  let range = ([YS .. (YS + YM)] `intersect` [XS .. (XS + XM)])
                                  in  if (length(range) `equal` 0 then return (INFINITY, 0))
                                      else return (min(range) + 1, length(range))
```

- $AND((YS, YM), (XS, XM)) = CALC_{RANGE}((YS, YM), (XS, XM))$
- $OR((YS, YM), (XS, XM)) = CALC_{RANGE}((YS, YM), (XS, XM))$
- $S(X, C) = \{$
    - $\neg C \implies [CALC_{RANGE}(X, C), [\infty, 0]],$
    - $C \implies [[\infty, 0], CALC_{RANGE}(X, C)]\}$
- $NOT((XS, XM)) = (XS + 1, XM)$

$Functions:$

- $JOIN(v) = \sqcap_{w \in pred(v)} [\![w]\!]$
- $op(Y, X) : [\![v]\!] = JOIN(v)[eval(JOIN(v), op(Y, X))]$
- $op(X) : [\![v]\!] = JOIN(v)[(eval(JOIN(v), op(X)))]$

Workthrough q2

$[\![L1]\!] = INPUT(3, A) = (0, 3)$
$[\![L2]\!] = INPUT(3, B) = (0, 3)$
$[\![L3]\!] = AND([\![L1]\!], [\![L1]\!]) = AND((0, 3), (0, 3)) = (1, 3)$
$[\![L4]\!] = NOT([\![L3]\!]) = NOT((1, 3)) = (2, 3)$
$[\![L5]\!] = OUT([\![L4]\!]) = OUT((2, 3)) = (3, 3)$

WAKE_UP(V) q2

$WAKE\_UP(V) = if(fst([\![V]\!]) == \infty)\ then\ TRUE\ else\ FALSE$

Time & Space q2

WAKE_UP is just time: O(1) & space: O(1) if constructed abstrace states.

Time q2

O(E) for E nodes, must visit each once.

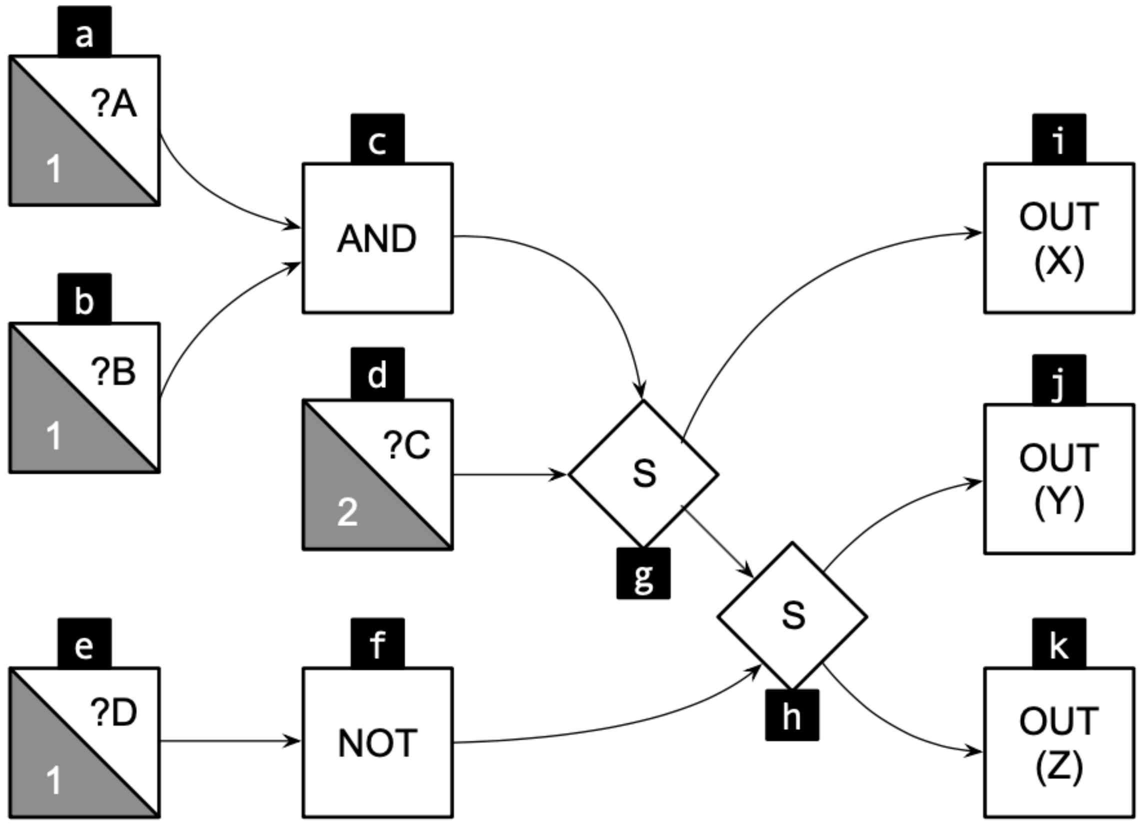
Space q2

Just contains the abstract value of the Node, so O(1) for each Node hence O(E) for E nodes.

Question 3

Initial working q3

Question 3 [5 Points] Some gates can be interpreted in parallel. In other words, whenever there are no dependencies between two gates, they can run simultaneously, because the result of a gate will not influence the result of another one. For an example, see the program below:



In this program, Gate ‘a’ can influence gates ‘c’, ‘g’, ‘i’, ‘h’, ‘j’ and ‘k’, but gate ‘a’ cannot influence gate ‘f’, for instance.

Goal: In this exercise, you must design a static analysis that determines if a gate A can influence the outcome of another gate B. Design your analysis so that, once it runs, the result of the query M_INF(A, B) is true if A can influence the result of B. M_INF must consult the abstract states of A and B, and based on this query, return an answer.

Forward analysis. Must analysis.

The lattice is of $Scope = \wp(Vars)$ with these operations for evaluation $State = \{Vars, \subseteq\}$,

- $eval(\sigma, op(Y, X)) = op(eval(\sigma, Y), eval(\sigma, X))$
- $eval(\sigma, S(Y, X)) = (snd|fst)(S(eval(\sigma, Y), eval(\sigma, X)))$
- $eval(\sigma, op(X)) = op(eval(\sigma, X))$

- $eval(\sigma, INPUT(X)) = X$
- $eval(\sigma, INPUT(N, X)) = (N > 0 \implies X) \vee (N \leq 0 \implies \bot)$
- $eval(\sigma, OUTPUT(X)) = X$

op:

- $AND(Y, X)$
  - $AND(0, X) = \emptyset$
  - $AND(1, X) = X$
- $OR(Y, X)$
  - $OR(0, X) = X$
  - $OR(1, X) = \emptyset$
- $S(X, CONTROL) = \{\neg CONTROL \implies [X, \bot], CONROL \implies [\bot, X]\}$
  - $S(X, 0) = [X, \emptyset]$
  - $S(X, 1) = [\emptyset, X]$
  - $S(X, Y) = [\{X, Y\}, \{X, Y\}]$
- $NOT(X) = X$

$Functions:$

- $JOIN(v) = \sqcup_{w \in pred(v)} [\![w]\!]$
- $op(Y, X) : [\![v]\!] = JOIN(v)[eval(JOIN(v), op(Y, X))]$
- $op(X) : [\![v]\!] = JOIN(v)[(eval(JOIN(v), op(X)))]$

Workthrough q3

$[\![i]\!] = \{i\} \cup [\![g]\!] = \{i, g, c, b, a, d\}$
$[\![j]\!] = \{j\} \cup [\![h]\!] = \{j, h, g, c, b, a, d, f, e\}$
$[\![k]\!] = \{k\} \cup [\![h]\!] = \{k, h, g, c, b, a, d, f, e\}$

$[\![h]\!] = \{h\} \cup [\![g]\!] \cup [\![f]\!] = \{h, g, c, b, a, d, f, e\}$
$[\![f]\!] = \{f\} \cup [\![e]\!] = \{f, e\}$
$[\![e]\!] = \{e\}$
$[\![g]\!] = \{g\} \cup [\![c]\!] \cup [\![d]\!] = \{g, c, b, a, d\}$
$[\![d]\!] = \{d\}$
$[\![c]\!] = \{c\} \cup [\![b]\!] \cup [\![a]\!] = \{c, b, a\}$
$[\![b]\!] = \{b\}$
$[\![a]\!] = \{a\}$

M_INF(V,W) q3

$M\_INF(V, W) = if(\{V\} \subset [\![W]\!])\ then\ TRUE\ else\ FALSE$

Time & Space q3

M_INF is just time: $O(|[\![W]\!]|)$ & space: O(1) if constructed abstrace states.

Time q3

O(E) for E nodes, must visit each once.

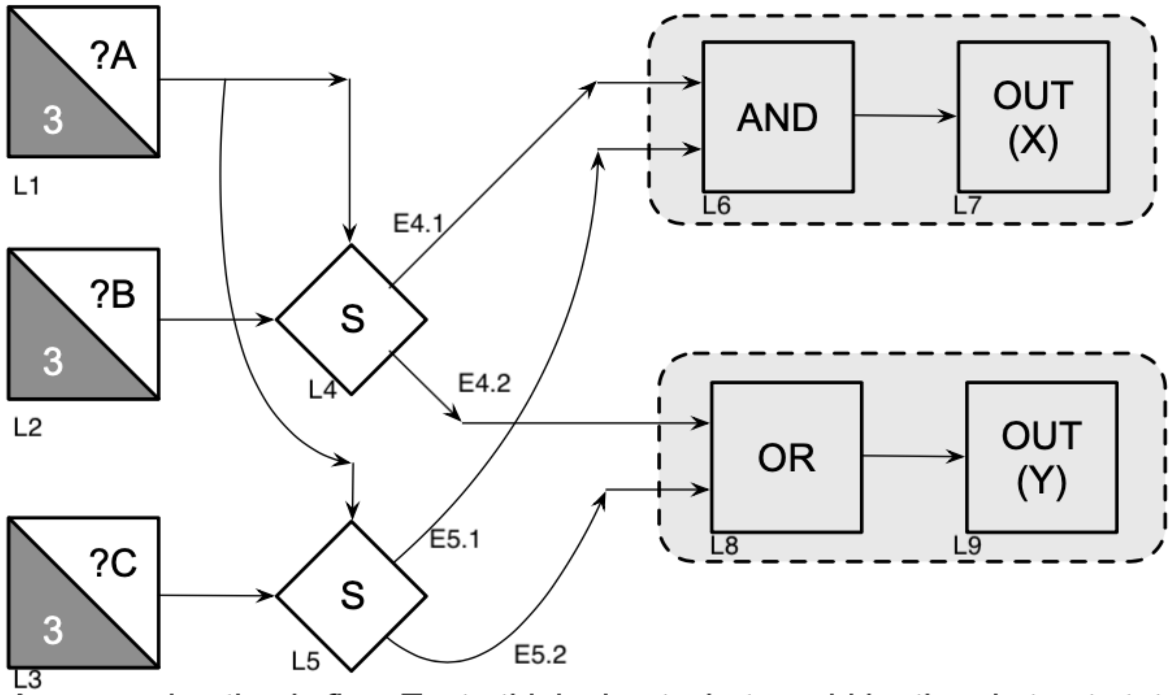
Space q3

Just contains the abstract value of the Node, so O(1) for each Node hence O(E) for E nodes.

Question 4

Initial thoughts q4

Question 4 [5 Points] When porting our hardware description language to an actual hardware, it is necessary to implement the gates using resources, like flip-flops and memory. Some gates can be implemented onto the same resources, because they can never execute together. For instance, consider the program below:



This program computes X = B AND C if A is zero, and Y = B OR C, if A is one. What is interesting about this program is that the gates in the gray regions will never be simultaneously defined. Therefore, these gates can be implemented using the same resources. Or, in other words, there is no problem in allocating these gates onto the same flip-flops, or simulating them in the same memory.

Goal: design an analysis that tells if two gates will never receive well-defined inputs simultaneously. Notice that this is the hardest exercise. You do not need to find the most precise answer to this analysis. An approximation is fine. Try to think about what would be the abstract state associated with each gate, and try to give at least one example where your analysis produces a non-trivial result, e.g., given two gates, A and B, it tells that they might, indeed, overlap. The product of this exericse is a function MIGHT_OVERLAP(A, B) which is true if A and B might be allocated to the same resources, e.g., they are never well-defined at the same clock tick.

- E4.1: selector at L4 bound with $\neg[\![L1]\!]$
- E4.2: selector at L4 bound with $[\![L1]\!]$
- E5.1: selector at L4 bound with $\neg[\![L1]\!]$
- E5.2: selector at L4 bound with $[\![L1]\!]$

Product lattice of selectors and variables, Set Union.

Reject inconsistent state e.g $s_{l4}@[\![L1]\!] \wedge s_{l4}@\neg[\![L1]\!]$.

Simplify the gates using the previous question 1 technique.

Bundle nodes with same abstract states together.

For each bundle switch on the variables needed to power this current bundle, identify other bundles that light up. These will be the bundles that overlap with this one lets call them activeCurr. Remove them from the set of all bundles to get the bundles that do not overlap.

Initial working q4

Forward analysis. Must analysis.

The lattice is of $States = \{SELECTORS \times VARIABLES, \subseteq\}$.

- $eval(\sigma, op(Y, X)) = op(eval(\sigma, Y), eval(\sigma, X))$
- $eval(\sigma, S(Y, X)) = (snd|fst)(S(eval(\sigma, Y), eval(\sigma, X)))$
- $eval(\sigma, op(X)) = op(eval(\sigma, X))$
- $eval(\sigma, INPUT(X)) = X$
- $eval(\sigma, INPUT(N, X)) = (N > 0 \implies X) \vee (N \leq 0 \implies \emptyset)$
- $eval(\sigma, OUTPUT(X)) = \emptyset$

op:

- $AND(Y, X)$
  - $AND(0, X) = \emptyset$
  - $AND(1, X) = X$
  - $AND(Y, X) = X \cup Y$
- $OR(Y, X)$
  - $OR(0, X) = X$
  - $OR(1, X) = \emptyset$

  - $OR(Y, X) = X \cup Y$
- $S_{label}(X, CONTROL) = \{\neg CONTROL \implies (S_{label}, \neg CONTROL), CONROL \implies (S_{label}, CONTROL)\}$
  - $S_{label}(X, 0) = [(S_{label}, \neg CONTROL), \perp]$
  - $S_{label}(X, 1) = [\perp, (S_{label}, CONTROL)]$
- $NOT(X) = \neg X$ (apply inside on to vars, i.e. snd)

$Functions:$

- $JOIN(v) = \sqcap_{w \in pred(v)} [\![w]\!]$
- $op(Y, X) : [\![v]\!] = JOIN(v)[eval(JOIN(v), op(Y, X))]$
- $op(X) : [\![v]\!] = JOIN(v)[(eval(JOIN(v), op(X)))]$

Workthrough q4

Not given.

MIGHT_OVERLAP(V, W) q4

$MIGHT\_OVERLAP(V, W) = if([\![V]\!] \cap [\![W]\!]) == \emptyset \ then \ FALSE \ else \ TRUE$