No warm-up today

We'll start with two announcements and then go back to the circuit.

Announcements:

HW 1 due tonight

+ make sure you can log into gradescope (we'll reset up at 11 pm).
+ when in doubt, show more work.

HW 2 out this afternoon.

Grab activity 5 pdf from the webpage.

~~If you see the same activity as yesterday, change the "11" to a "5" in the URL.~~

Normal Forms and Predicates

CSE 311 Autumn 2020
Lecture 5

Step One

Input: day of the week, Boolean talkToSomeone

Output: The way to get your question answered, according to the following rules:

On M,Tu,W,F if you want to talk, go to office hours

On Th if you want to talk, go to section

Monday through Friday, if you don't want to talk ask on Ed

On Saturday or Sunday, text a friend (whether you want to talk or not)

Take 2 minutes **plan** what your code might look like.

Step One

```
1   if( (day==Monday || day==Tuesday || day==Wednesday || day==Friday) ) {
2       if(talkToSomeone)
3           return "office hours";
4       else
5           return "Ed";
6   }
7   else if(day==Thursday) {
8       if(talkToSomeone)
9           return "section";
10      else
11          return "Ed";
12  }
13  else //day is Saturday or Sunday
14      return "text a friend";
15
```

One possibility (there are many)

Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0			1	
Tuesday	0	0	1	1	1			
Wednesday	0	1	0	0			1	
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				1
Sunday	1	1	0	0				1
Sunday	1	1	0	1				1
---	1	1	1	0				
---	1	1	1	1				

Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0			1	
Tuesday	0	0	1	1	1			
Wednesday	0	1	0	0			1	
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				
Sunday	1	1	0	0				
Sunday	1	1	0	1				
---	1	1	1	0				
---	1	1	1	1				

$\neg d_2 \wedge \neg d_1 \wedge \neg d_0 \wedge s$

$\neg d_2 \wedge \neg d_1 \wedge d_0 \wedge s$

$\neg d_2 \wedge d_1 \wedge \neg d_0 \wedge s$

$d_2 \wedge \neg d_1 \wedge \neg d_0 \wedge s$

$$out_0 = (\neg d_2 \wedge \neg d_1 \wedge \neg d_0 \wedge s) \vee (\neg d_2 \wedge \neg d_1 \wedge d_0 \wedge s) \vee (\neg d_2 \wedge d_1 \wedge \neg d_0 \wedge s) \vee (d_2 \wedge \neg d_1 \wedge \neg d_0 \wedge s)$$

Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0			1	
Tuesday	0	0	1	1	1			
Wednesday	0	1	0	0			1	
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				
Sunday	1	1	0	0				
Sunday	1	1	0	1				
---	1	1	1	0				
---	1	1	1	1				

$d_2'd_1'd_0's$

$d_2'd_1'd_0s$

$d_2'd_1d_0's$

$d_2d_1'd_0's$

$$out_0 = d_2'd_1'd_0's + d_2'd_1'd_0s + d_2'd_1d_0's + d_2d_1'd_0's$$

Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0			1	
Tuesday	0	0	1	1	1			
Wednesday	0	1	0	0			1	
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				
Sunday	1	1	0	0				
Sunday	1	1	0	1				
---	1	1	1	0				
---	1	1	1	1				

$d_2'd_1'd_0's$

$d_2'd_1'd_0s$

$d_2'd_1d_0's$

$d_2d_1'd_0's$

$$out_0 = (d_2'd_1'd_0' + d_2'd_1'd_0 + d_2'd_1d_0' + d_2d_1'd_0')s$$

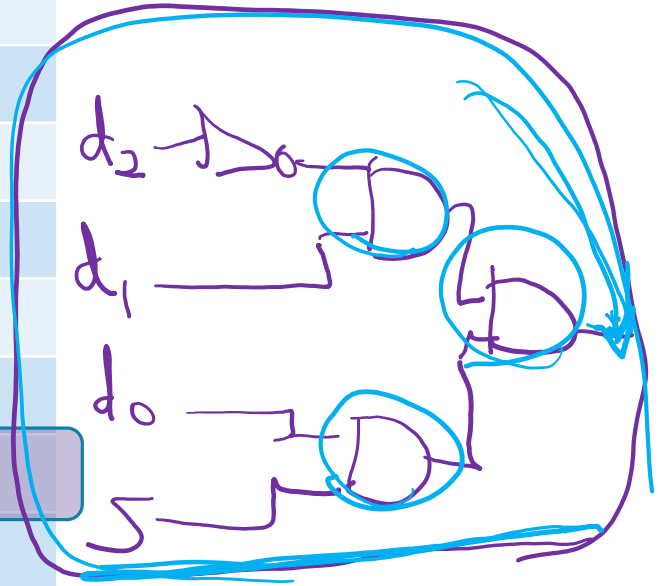
Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0			1	
Tuesday	0	0	1	1	1			
Wednesday	0	1	0	0			1	
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				1
Sunday	1	1	0	0				1
Sunday	1	1	0	1				1
---	1	1	1	0				
---	1	1	1	1				

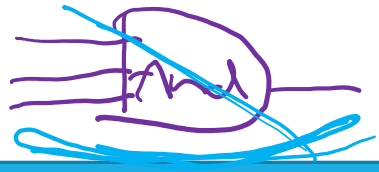
Find the formula for out_1 in both Boolean algebra and propositional logic.

If you have extra time, draw the circuit representation.

Fill out the poll everywhere for Activity Credit!
Go to pollev.com/cse311 and login with your UW identity
Or text cse311 to 22333

Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0			1	
Tuesday	0	0	1	1	1			
Wednesday	0	1	0	0			1	
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				
Sunday	1	1	0	0				
Sunday	1	1	0	1				
---	1	1	1	0				
---	1	1	1	1				





$$out_1 = d_2' d_1 d_0 s$$

$$out_1 = \neg d_2 \wedge d_1 \wedge d_0 \wedge s$$

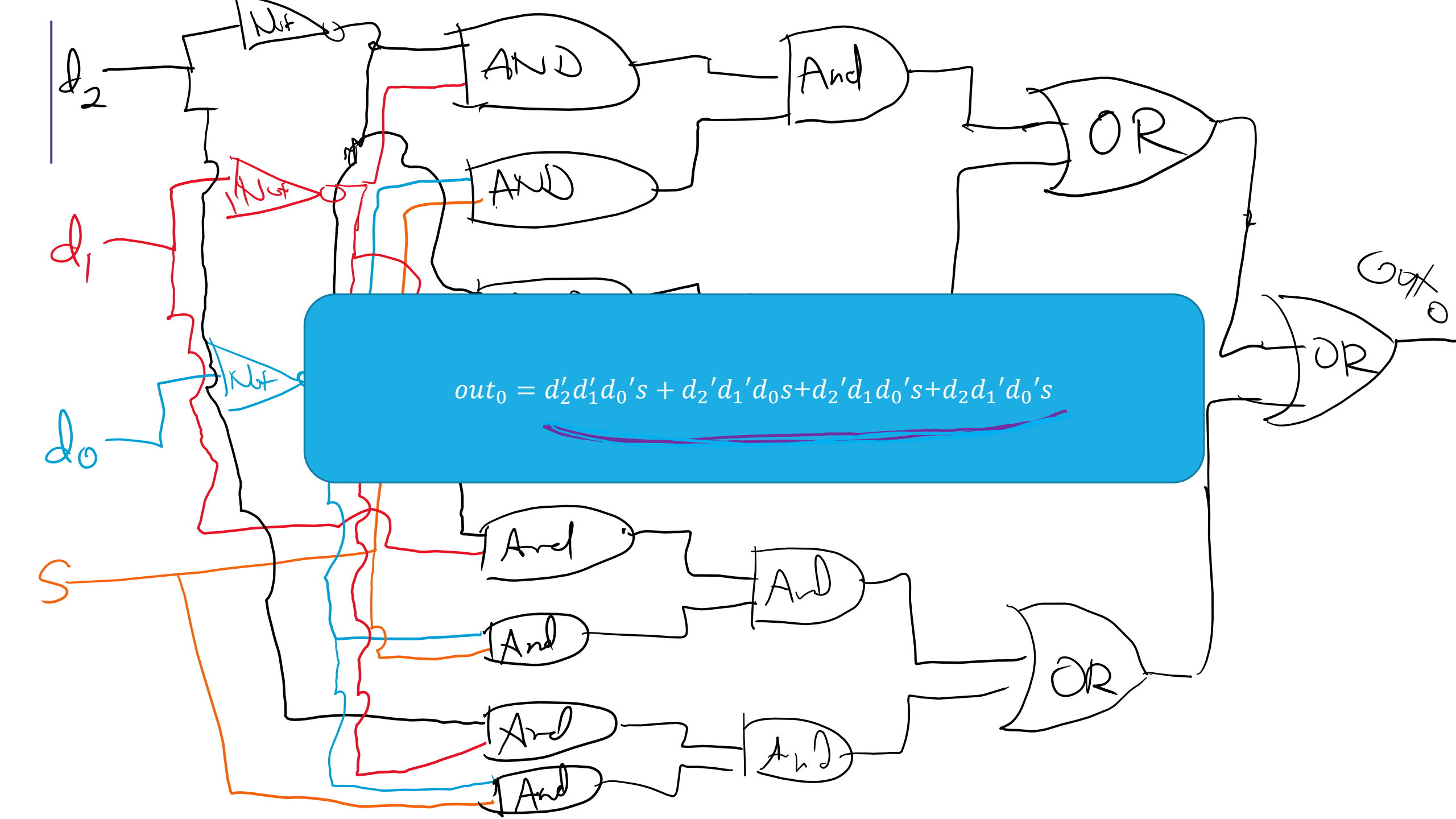
Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0			1	
Tuesday	0	0	1	1	1			
Wednesday	0	1	0	0			1	
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				1
Sunday	1	1	0	0				
Sunday	1	1	0	1				
---	1	1	1	0				
---	1	1	1	1				

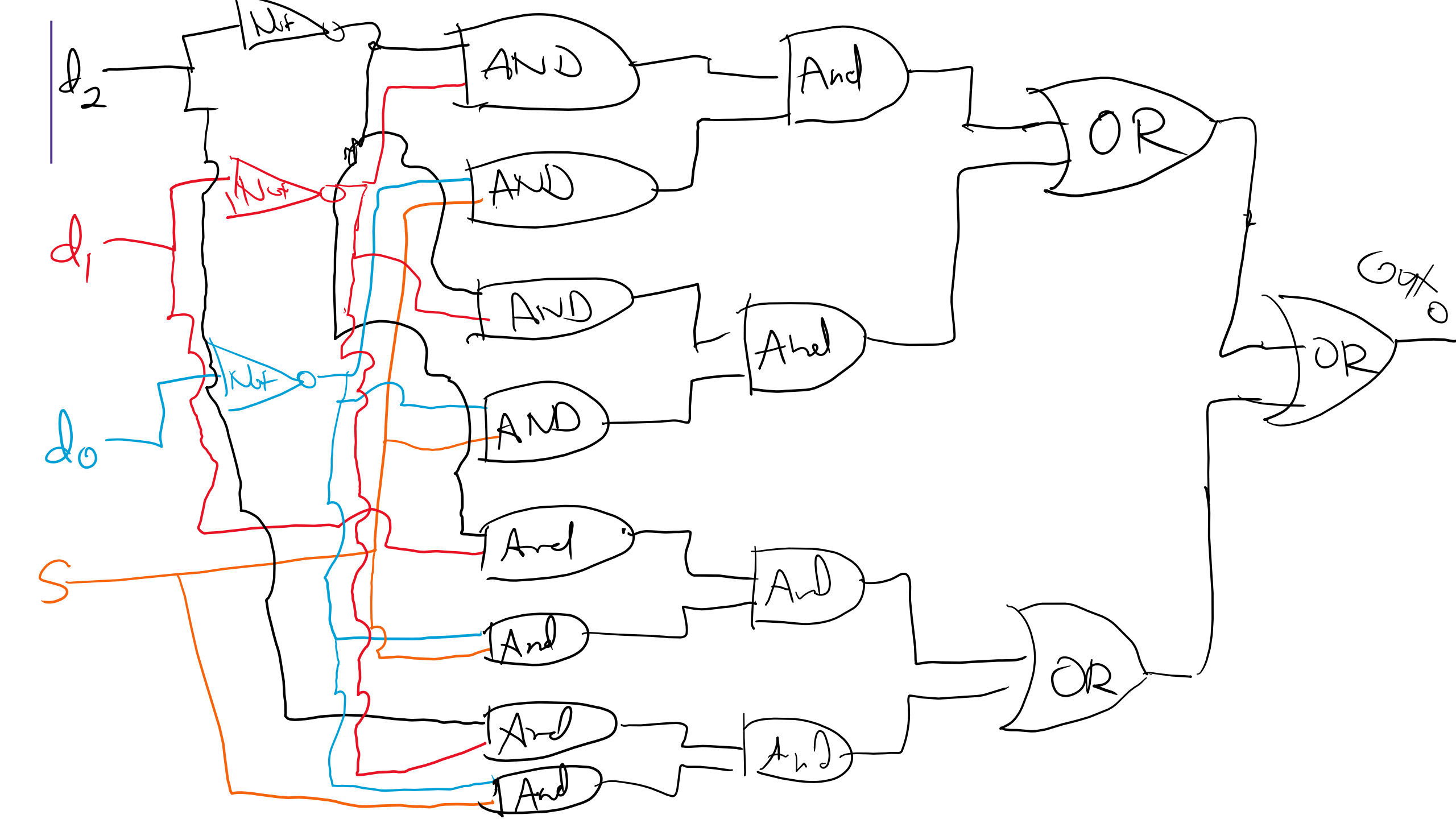
$$out_2 = d_2' d_1' d_0' s' + d_2' d_1' d_0 s' + d_2' d_1 d_0 s' + d_2 d_1' d_0' s'$$

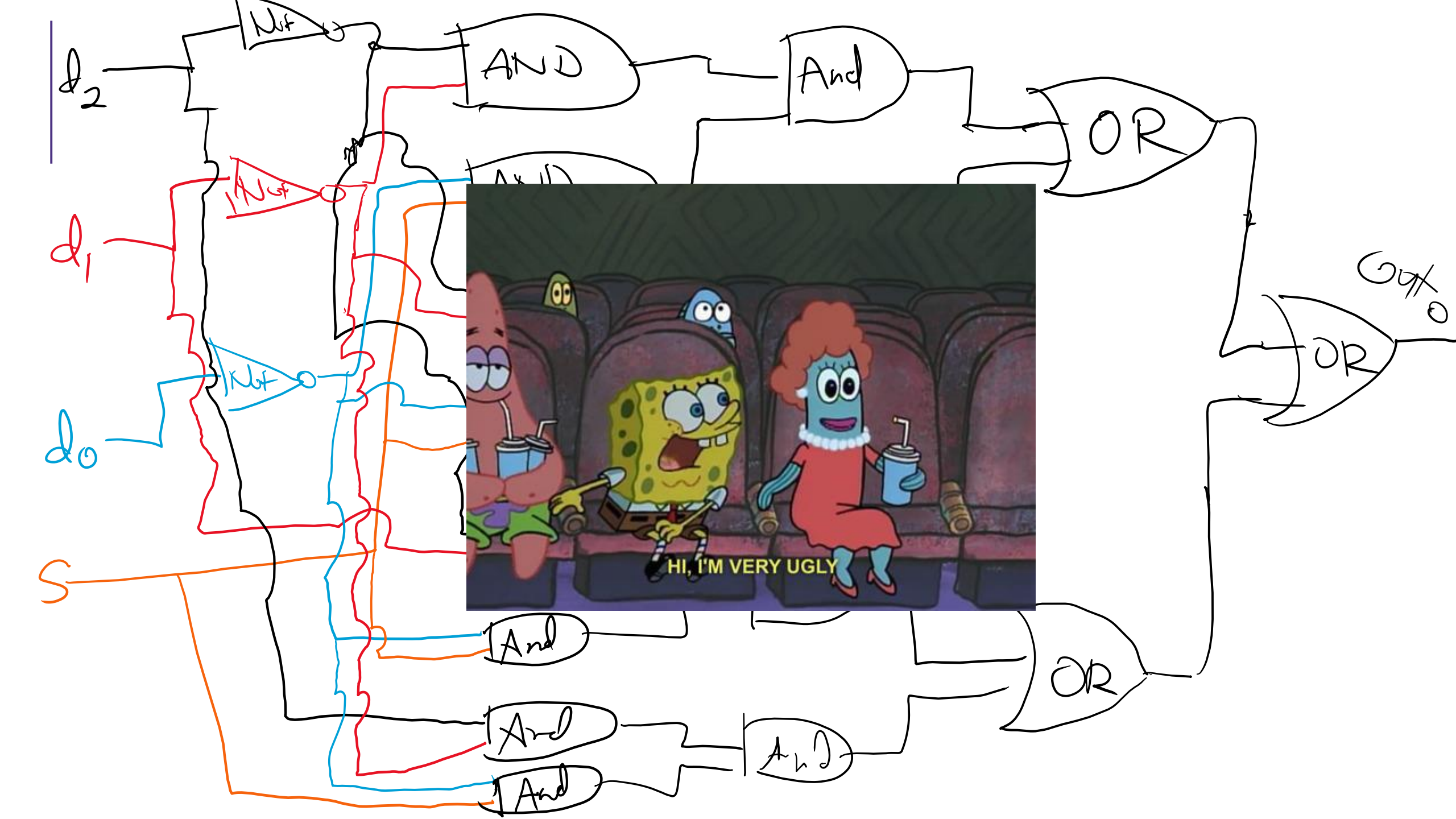
$$out_2 = d_2' s' (d_1' d_0' + d_1' d_0 + d_1 d_0) + d_2 d_1' d_0' s'$$

Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1				
Tuesday	0	0	1	0				
Tuesday	0	0	1	1				
Wednesday	0	1	0	0				
Wednesday	0	1	0	1				
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				1
Sunday	1	1	0	0				1
Sunday	1	1	0	1				1
---	1	1	1	0				
---	1	1	1	1				

$$out_3 = d_2(d_1'd_0 + d_1d_0' + d_1d_0)$$

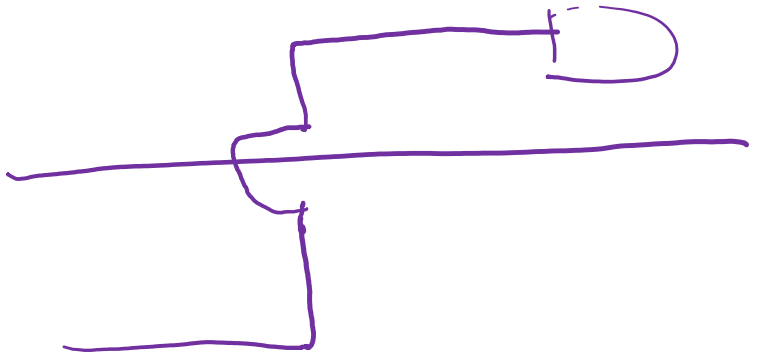

d_2
d_1
d_0
S
Not
Not
Not
AND
AND
And
And
And
And
And
And
OR
OR
OR
Out
$out_0 = d_2'd_1'd_0's + d_2'd_1'd_0s + d_2'd_1d_0's + d_2d_1'd_0's$


d_2
d_1
d_0
S
Not
Not
Not
AND
AND
AND
AND
And
And
And
And
And
And
OR
OR
OR
Out
0

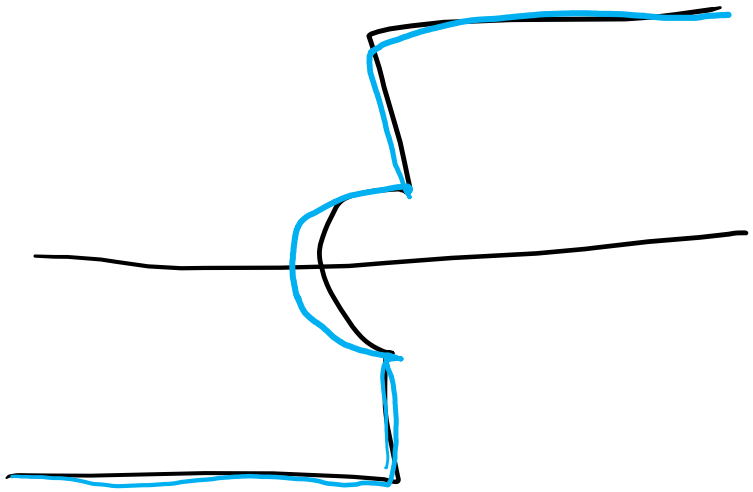

d_2
d_1
d_0
S
Not
Not
Not
AND
And
And
And
And
And
OR
OR
OR
Out
HI, I'M VERY UGLY

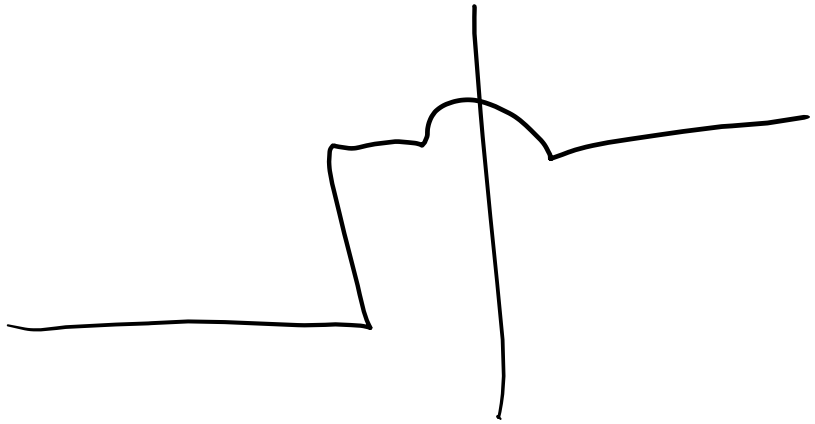
Ick

WOW that's ugly.



Be careful when wires cross – draw one "jumping over" the other.

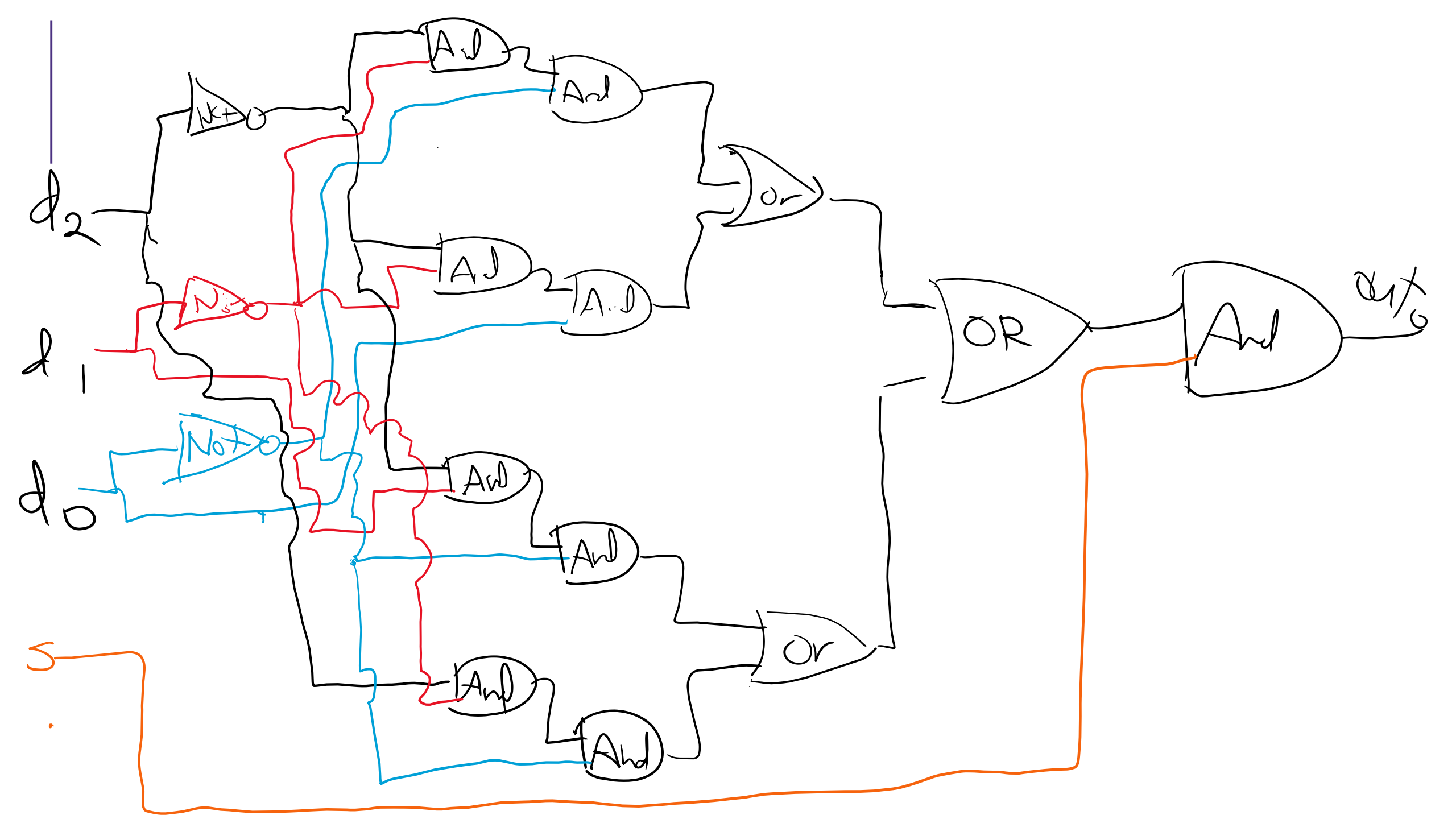




Can we do better

Maybe the factored version will be better?

$$out_0 = (d_2' d_1' d_0' + d_2' d_1' d_0 + d_2' d_1 d_0' + d_2 d_1' d_0')s$$


d_2
d_1
d_0
S
NOT
NOT
NOT
AND
AND
AND
AND
AND
AND
AND
OR
OR
OR
AND
out


Still
HI, I'M VERY UGLY

The Factored Version

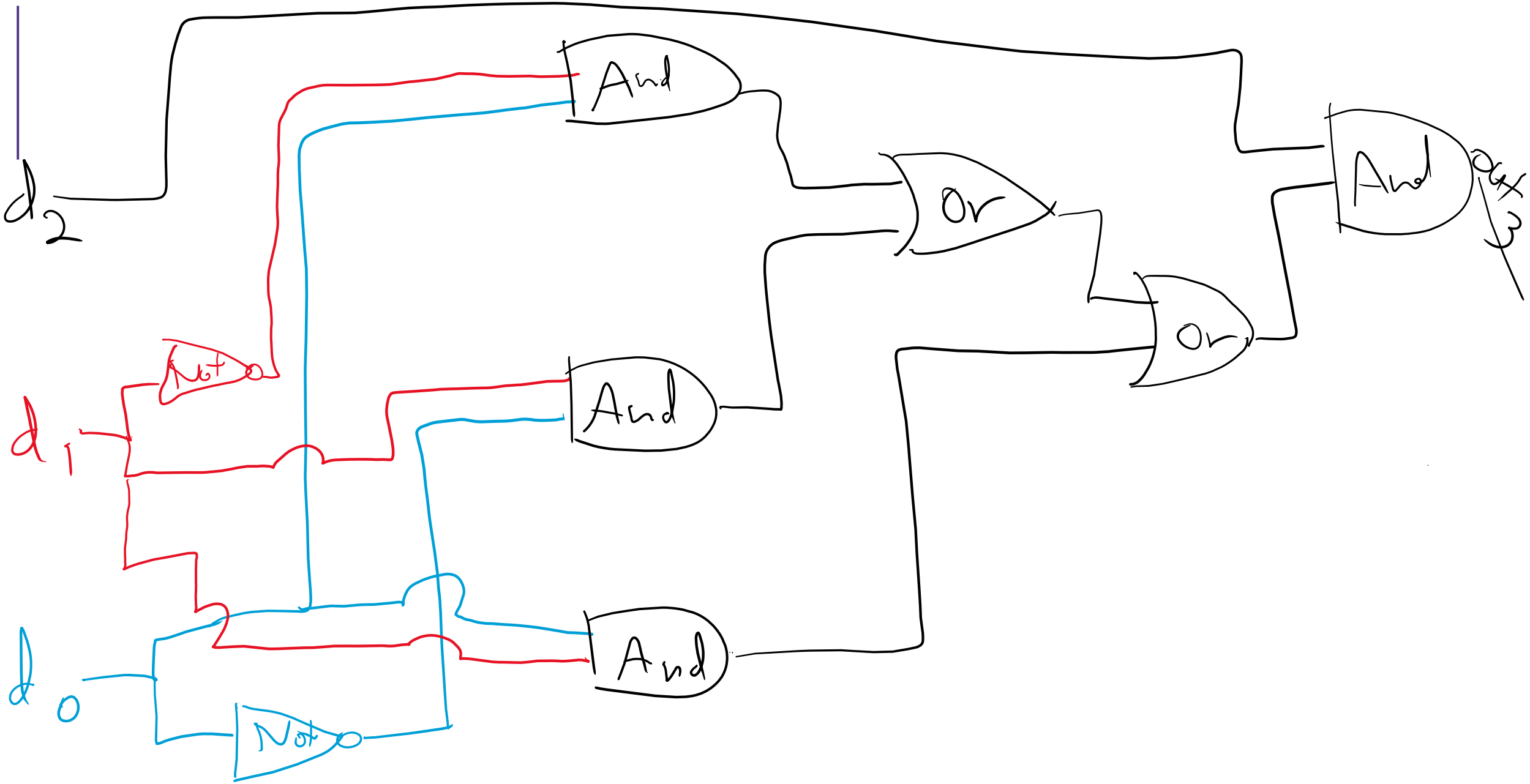
Ehhhhhhh, it's a little better?

Part of the problem here is Robbie's art skills.

Part is some layout choices – commuting the terms might make things prettier.

Most of the problem is just the circuit is complicated.

out_3 is a little better.


d_2
And
Or
And
Out
Not
d_1
And
Or
d_0
And
Not

Can we use these for anything?

Sometimes these concrete formulas lead to easier observations.

For example, we might have noticed we factored out s or s' in three of the four, which suggests switching s first.

```
1  if(talkToSomeone) {
2      if( (day==Monday || day==Tuesday || day==Wednesday || day==Friday) )
3          return "office hours";
4      else if( day==Thursday)
5          return "section";
6      else
7          return "text a friend";
8  }
9  else {
10     if( (day==Monday || day==Tuesday || day==Wednesday || day==Thursday || day==Friday) )
11         return "Ed";
12     else
13         return "text a friend";
14 }
```

Can we use these for anything?

Is this code better? Maybe, maybe not.

It's another tool in your toolkit for thinking about logic

Including logic you write in code!

```
1  if(talkToSomeone) {
2      if( (day==Monday || day==Tuesday || day==Wednesday || day==Friday) )
3          return "office hours";
4      else if( day==Thursday)
5          return "section";
6      else
7          return "text a friend";
8  }
9  else {
10     if( (day==Monday || day==Tuesday || day==Wednesday || day==Thursday || day==Friday) )
11         return "Ed";
12     else
13         return "text a friend";
14 }
```

Takeaways

Yet another notation for propositions.

These are just more representations – there's only **one** underlying set of rules.

Next time: wrap up digital logic and the tool really represent $x > 5$.

Another Proof

Let's prove that $(p \wedge q) \rightarrow (q \vee p)$ is a tautology.

Alright, what are we trying to show?

Another Proof

$$(p \wedge q) \rightarrow (q \vee p) \equiv \neg(p \wedge q) \vee (q \vee p)$$
$$\equiv (\neg p \vee \neg q) \vee (q \vee p)$$
$$\equiv \neg p \vee (\neg q \vee (q \vee p))$$
$$\equiv \neg p \vee \big((\neg q \vee q) \vee p\big)$$
$$\equiv \neg p \vee ((q \vee \neg q) \vee p)$$
$$\equiv \neg p \vee (\mathrm{T} \vee p)$$
$$\equiv \neg p \vee (p \vee \mathrm{T})$$
$$\equiv \neg p \vee \mathrm{T}$$
$$\equiv \mathrm{T}$$

Law of Implication

DeMorgan's Law

It's easier if everything is AND/OR/NOT

Associative (twice)

Gets rid of some parentheses/just a gut feeling

Commutative, Negation

Put $q, \neg q$ next to each other.

Simplify $q \vee \neg q$.

Commutative, Domination, Domination

Simplify until we get T.

We're done!

Proof-writing tip:

Take a step back.

Pause and carefully look at what you have. You might see where to go next…

Another Proof

$(p \wedge q) \rightarrow (q \vee p)$	$\equiv \neg(p \wedge q) \vee (q \vee p)$	Law of implication
	$\equiv (\neg p \vee \neg q) \vee (q \vee p)$	DeMorgan's Law
	$\equiv \neg p \vee (\neg q \vee (q \vee p))$	Associative
	$\equiv \neg p \vee \big((\neg q \vee q) \vee p\big)$	Associative
	$\equiv \neg p \vee ((q \vee \neg q) \vee p)$	Commutative
	$\equiv \neg p \vee (\mathrm{T} \vee p)$	Negation
	$\equiv \neg p \vee (p \vee \mathrm{T})$	Commutative
	$\equiv \neg p \vee \mathrm{T}$	Domination
	$\equiv \mathrm{T}$	Domination

Today

Wrap up digital logic with “standard” ways to read propositions from truth tables.

Predicate

~~Propositional~~ logic – how do we handle logic with more than one “entity”

Canonical Forms

A truth table is a unique representation of a Boolean Function.

If you describe a function, there's only one possible truth table for it.

Given a truth table you can find many circuits and many compound prepositions to represent it.

Think back to when we were developing the law of implication…

Using Our Rules

WOW that was a lot of rules.

Why do we need them? Simplification!

Let's go back to the "law of implication" example.

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

When is the implication true? Just "or" each of the three "true" lines!

$$(p \wedge q) \vee (\neg p \wedge q) \vee (\neg p \wedge \neg q)$$

Also seems pretty reasonable

So is $(p \wedge q) \vee (\neg p \wedge q) \vee (\neg p \wedge \neg q) \equiv (\neg p \vee q)$

i.e. are these both alternative representations of $p \rightarrow q$?

Canonical Forms

A truth table is a unique representation of a Boolean Function.

If you describe a function, there's only one possible truth table for it.

Given a truth table you can find many circuits and many compound prepositions to represent it.

Think back to when we were developing the law of implication…

It would be nice to have a "standard" proposition (or standard circuit) we could always write as a starting point.

So we have a (possibly) shorter way of telling if we have the same function.

Disjunctive Normal Form (DNF)

a.k.a. OR of ANDs

a.k.a Sum-of-Products Form

a.k.a. Minterm Expansion

1. Read the true rows of the truth table
2. AND together all the settings in a given (true) row.
3. OR together the true rows.

Disjunctive Normal Form

p	q	$G(p,q)$
T	T	T
T	F	F
F	T	T
F	F	F

$p \wedge q$

$\neg p \wedge q$

1. Read the true rows of the truth table
2. AND together all the settings in a given (true) row.
3. OR together the true rows.

$$G(p,q) \equiv (p \wedge q) \vee (\neg p \wedge q)$$

DNF

$(q \wedge p) \vee (q \wedge \neg p)$

Another Canonical Form

DNF is a great way to represent functions that are usually false.

If there are only a few true rows, the representation is short.

What about functions that are usually true?

Well G is equivalent to $\neg\neg G$, and $\neg G$ is a function that is usually false.

Let's try taking the Disjunctive Normal Form of $\neg G$ and negating it.

Another Canonical Form

p	q	$G(p,q)$	$\neg G(p,q)$	
T	T	T	F	
T	F	F	T	$p \wedge \neg q$
F	T	T	F	
F	F	F	T	$\neg p \wedge \neg q$

1. Read the true rows of the truth table
2. AND together all the settings in a given (true) row.
3. OR together the true rows.

$$\neg G(p,q) \equiv (p \wedge \neg q) \vee (\neg p \wedge \neg q)$$
$$G(p,q) \equiv \neg[(p \wedge \neg q) \vee (\neg p \wedge \neg q)]$$
$$G(p,q) \equiv [\neg(p \wedge \neg q) \wedge \neg(\neg p \wedge \neg q)]$$
$$G(p,q) \equiv [(\neg p \vee q) \wedge (p \vee q)]$$

This is not in Disjunctive Normal Form! It's something else, though…

Conjunctive Normal Form

a.k.a. AND of ORs

a.k.a. Product-of-Sums Form

a.k.a. Maxterm Expansion

1. Read the false rows of the truth table (of the original function G)
2. OR together ~~all~~ the negations of the settings in the false rows.
3. AND together the false rows.

Or take the DNF of the negation of the function you care about (i.e. of $\neg G$), and distribute the negation.

Normal Forms

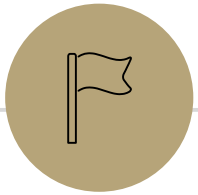
Don't simplify any further! Don't factor anything out (even if you can). The point of the canonical form is we know exactly what it looks like, you might simplify differently than someone else.

Why? Easier to understand for people.

Inside the parentheses are only ORs between the parentheses are only ANDs (or vice versa).

You'll use these more in later courses.

CNF ≡ CSDNF



Predicate Logic

Predicate Logic

So far our propositions have worked great for fixed objects.

What if we want to say "If $x > 10$ then $x^2 > 100$."

$x > 10$ isn't a proposition. Its truth value depends on x.

We need a function that can take in a value for x and output True or False as appropriate.

Predicates

Predicate

A function that outputs true or false.

`Cat(x)` := "`x` is a cat"

`Prime(x)` := "`x` is prime"

`LessThan(x,y)` := "`x<y`"

`Sum(x,y,z)` := "`x+y=z`"

`HasNChars(s,n)` := "string `s` has length `n`"

Numbers and types of inputs can change. Only requirement is output is Boolean.

Analogy

Propositions were like Boolean variables.

What are predicates? Functions that return Booleans

```
public boolean pred(…)
```

Translation

Translation works a lot like when we just had propositions.

Let's try it…

x is prime or x^2 is odd or $x = 2$.

$\texttt{Prime}(x) \vee \texttt{Odd}(x^2) \vee \texttt{Equals}(x, 2)$

Domain of Discourse

x is prime or x^2 is odd or $x = 2$.

$\texttt{Prime}(x) \vee \texttt{Odd}(x^2) \vee \texttt{Equals}(x, 2)$

Can x be 4.5? What about "abc" ?

I never intended you to plug 4.5 or "abc" into x.

When you read the sentence you probably didn't imagine plugging those values in….

Domain of Discourse

x is prime or x^2 is odd or $x = 2$.

$$\texttt{Prime}(x) \vee \texttt{Odd}(x^2) \vee \texttt{Equals}(x, 2)$$

To make sure we **can't** plug in 4.5 for x, predicate logic requires deciding on the types we'll allow

Domain of Discourse

The *types* of inputs allowed in our predicates.

Try it…

What's a possible domain of discourse for these lists of predicates?

1. "x is a cat", "x barks", "x likes to take walks"
2. "x is prime", "x=5" "$x < 20$" "x is a power of two"
3. "x is enrolled in course y", "y is a pre-req for z"

Try it…

What's a possible domain of discourse for these lists of predicates?

1. "x is a cat", "x barks", "x likes to take walks"

   "Mammals", "pets", "dogs and cats", …

2. "x is prime", "$x=5$" "$x < 20$" "x is a power of two"

   "positive integers", "integers", "numbers", …

3. "x is enrolled in course y", "y is a pre-req for z"

   "objects in the university course enrollment system", "university entities", "students and courses", …

More than one domain of discourse might be reasonable…if it might affect the meaning of the statement, we specify it.

Quantifiers

Now that we have variables, let's really use them…

We tend to use variables for two reasons:

1. The statement is true for every x, we just want to put a name on it.
2. There's some x out there that works, (but I might not know which it is, so I'm using a variable).

Quantifiers

We have two extra symbols to indicate which way we're using the variable.

1. The statement is true for every x, we just want to put a name on it.

$\forall x\ (\mathrm{p}(\mathrm{x}) \wedge q(x))$ means "for every x in our domain, $p(x)$ and $q(x)$ both evaluate to true."

2. There's some x out there that works, (but I might not know which it is, so I'm using a variable).

$\exists x(p(x) \wedge q(x))$ means "there is an x in our domain, such that $p(x)$ and $q(x)$ are both true.

Quantifiers

We have two extra symbols to indicate which way we're using the variable.

1. The statement is true for every x, we just want to put a name on it.

$\forall x\ (\mathrm{p}(\mathrm{x}) \wedge q(x))$ means "for every x in our domain, $p(x)$ and $q(x)$ both evaluate to true."

Universal Quantifier

"$\forall x$"

"for each x", "for every x", "for all x" are common translations

Remember: upside-down-A for All.

Quantifiers

$x < 5$

Existential Quantifier

"$\exists x$"

"there is an x", "there exists an x", "for some x" are common translations

Remember: backwards-E for Exists.

2. There's some x out there that works, (but I might not know which it is, so I'm using a variable).

$\exists x(p(x) \wedge q(x))$ means "there is an x in our domain, for which $p(x)$ and $q(x)$ are both true.

$\exists x\,(P(x))$

Translations

"For every x, if x is even, then $x = 2$."

"There are x, y such that $x < y$."

$\exists x\ (\texttt{Odd}(x) \wedge \texttt{LessThan}(x, 5))$

$\forall y\ (\text{Even}(y) \wedge \text{Odd}(y))$

Fill out the poll everywhere for Activity Credit!

Go to pollev.com/cse311 and login with your UW identity
Or text cse311 to 22333

Translations

"For every x, if x is even, then $x = 2$."

$$\forall x(\texttt{Even}(x) \rightarrow \texttt{Equal}(x, 2))$$

"There are x, y such that $x < y$."

$$\exists x \exists y(\texttt{LessThan}(x, y))$$

$$\exists x (\texttt{Odd}(x) \wedge \texttt{LessThan}(x, 5))$$

There is an odd number that is less than 5.

$$\forall y \, (\text{Even}(y) \wedge \text{Odd}(y))$$

All numbers are both even and odd.

Translations

More practice in section and on homework.

Also a reading on the webpage –

An explanation of why "for any" is not a great way to translate ∀ (even though it looks like a good option on the surface)

More information on what happens with multiple quantifiers (we'll discuss more next week).

Evaluating Predicate Logic

"For every x, if x is even, then $x = 2$." / $\forall x(\texttt{Even}(x) \rightarrow \texttt{Equal}(x, 2))$

Is this true?

Evaluating Predicate Logic

"For every x, if x is even, then $x = 2$." / $\forall x(\texttt{Even}(x) \rightarrow \texttt{Equal}(x, 2))$

Is this true?

TRICK QUESTION! It depends on the domain.

Prime Numbers	Positive Integers	Odd integers
True	False	True (vacuously)

One Technical Matter

How do we parse sentences with quantifiers?

What's the "order of operations?"

We will usually put parentheses right after the quantifier and variable to make it clear what's included. If we don't, it's the rest of the expression.

Be careful with repeated variables…they don't always mean what you think they mean.

$\forall x\big(P(x)\big) \wedge \forall x(Q(x))$ are different x's.

More Practice

Let your domain of discourse be fruits.

There is a fruit that is tasty and ripe.

$$\exists x(\texttt{Tasty}(x) \land \texttt{Ripe}(x))$$

For every fruit, if it is not ripe then it is not tasty.

$$\forall x(\neg\texttt{Ripe}(x) \rightarrow \neg\texttt{Tasty}(x))$$

There is a fruit that is sliced and diced.

$$\exists x(\texttt{Sliced}(x) \land \texttt{Diced}(x))$$