Problem 2.1

Consider the following formula:

$$\varphi := x = y \rightarrow (x = a \vee x = b)$$

where x, y are variables and a, b are constants.
Describe the class of interpretations that makes this formula valid.

As it was often done in the lecture, assume that = is already interpreted as the usual equality relation. Otherwise variants where = is a constant binary relation evaluating to true could be considered, which (I think) may not be the purpose of this exercise? However, considering = as equality alone is insufficient, as this class of interpretations still contains interpretations where the formula evaluates to false. Hence, the approach chosen here, is to further restrict the class of interpretations by restricting the domain of the interpretations in the class.

Claim: $\mathcal{I} \vDash \varphi$ if and only if $\mathcal{I} \in \{\mathcal{J} | \mathfrak{P}(\mathcal{I})\}$, where $\mathfrak{P}(\mathcal{J})$ holds if and only if = is the equality relation, $|\alpha^{\mathcal{J}}| \leq 1$.

(Note: only one sort has to be considered, as otherwise the formula would not be well formed).

"$\Leftarrow$" For $|\alpha^{\mathcal{I}}| = 1$, for sake of convenience call this element 0. Hence, $\mathcal{I}(x = y \rightarrow (x = a \vee x = b)) = (0 = 0 \Rightarrow (0 = 0 \; or \; 0 = 0)))$.
"$\Rightarrow$" Consider the class of interpretations where $|\alpha^{\mathcal{I}}| > 1$. Hence, for some $\mathcal{I}$ from this class, there are at least two distinct elements 0 and 1. Now, given this fact choose $\mathcal{I}$ from this class to be such that $a^{\mathcal{I}} = b^{\mathcal{I}}$, as well as $x^{\mathcal{I}} = y^{\mathcal{I}}$ and $a^{\mathcal{I}} \neq x^{\mathcal{I}}$. Clearly $\mathcal{I}(x = y \rightarrow (x = a \vee x = b))$ evaluates to false.

Moreover, it is possible to extend the class of interpretations to interpretations with domain 2, if one specifies that a and b ought to be distinct.

Problem 2.2

Consider two axiomatisable theories $\mathcal{T}_1$ and $\mathcal{T}_2$ such that $\Sigma_{\mathcal{T}_1} = \Sigma_{\mathcal{T}_2} = \Sigma$ and $A_{\mathcal{T}_2} \subseteq A_{\mathcal{T}_1}$. Recall that $\Sigma_{\mathcal{T}_i}$ and $A_{\mathcal{T}_1}$ denote, respectively, a signature and set of axioms defining $\mathcal{T}_1$, for $i = 1, 2$. Let F be a formula over signature Σ.

1. If F is valid in $\mathcal{T}_1$, it is also valid in $\mathcal{T}_2$? Prove your answer or give a counterexample.

2. If F is valid in $\mathcal{T}_2$, it is also valid in $\mathcal{T}_1$? Prove your answer or give a counterexample.

Let $\varphi = F$.

1. Let $\mathcal{T}_1 := \mathcal{T}_E$ and $\mathcal{T}_2 := \mathcal{T}_\varnothing$, where $A_\varnothing = \varnothing$, i.e. arbitrary interpretations. Clearly $A_\varnothing \subseteq A_{\mathcal{T}_E}$. Moreover, let $\varphi := a = b \rightarrow (f(a) = f(b))$. Let the corresponding signatures be appropriate with respect to the conditions above and the formula φ. Firstly, as φ is an instance of function congruence, clearly $\mathcal{T}_E \vDash \varphi$. Secondly, choose $\mathcal{I}$ such that there are at least two distinct elements in the domain, which will be called 0 and 1. Moreover, let $f^{\mathcal{I}}$ be a unary constant function mapping to 0 and let $a^{\mathcal{I}} = b^{\mathcal{I}} := 1$ and . Lastly, let $=^{\mathcal{I}}$ be a binary relation such that $=^{\mathcal{I}} := \{(1,1)\}$. Clearly, the premise of the implication in φ is satisfied by $\mathcal{I}$, while the consequent is not. Hence, $\mathcal{I} \nvDash \varphi$ and therefore $\mathcal{T}_\varnothing \nvDash \varphi$.

2. φ is valid in $\mathcal{T}_2$. This means that for every $\mathcal{T}_2$-interpretation $\mathcal{I}_{\mathcal{T}_2}$, it holds that $\mathcal{I}_{\mathcal{T}_2} \vDash \varphi$. Moreover, since $\mathcal{I}_{\mathcal{T}_2}$ is a $\mathcal{T}_2$-interpretation, it holds that $\mathcal{I}_{\mathcal{T}_2} \vDash A_{\mathcal{T}_2}$. Therefore, since φ is valid in $\mathcal{T}_2$, this means for an arbitrary interpretation $\mathcal{I}$,

$$\mathcal{I} \vDash A_{\mathcal{T}_2} \implies \mathcal{I} \vDash \varphi$$

Now, $\mathcal{I}_{\mathcal{T}_1} \vDash A_{\mathcal{T}_1}$ can be rewritten as

$$\mathcal{I}_{\mathcal{T}_1} \vDash \bigwedge_{\psi \in A_{\mathcal{T}_1}} \psi$$

Furthermore, since $A_{\mathcal{T}_2} \subseteq A_{\mathcal{T}_1}$ one can split the this into

$$\mathcal{I}_{\mathcal{T}_1} \vDash \left(\bigwedge_{\psi \in A_{\mathcal{T}_2}} \psi \right) \wedge \left(\bigwedge_{\psi \in A_{\mathcal{T}_1} \setminus A_{\mathcal{T}_2}} \psi \right)$$

Moreover, given the tautology $(A \wedge B) \rightarrow A$ one obtains

$$\mathcal{I}_{\mathcal{T}_1} \vDash \left(\bigwedge_{\psi \in A_{\mathcal{T}_2}} \psi \right)$$

So clearly $\mathcal{I}_{\mathcal{T}_1} \vDash A_{\mathcal{T}_2}$ and thus given the observation above it follows that $\mathcal{I}_{\mathcal{T}_1} \vDash \varphi$.

Problem 2.3

For each formula below, decide whether it is satisfiable or not by relying on the decision procedure of the theory or arrays. If the formula is satisfiable, give an interpretation that satisfies the formula.

1. $read(write(A,i,e),j) = e \wedge i \neq j \wedge read(A,j) \neq e$, where i, e, j, are integer-valued constants and A is an array constant.
2. $read(write(write(A,i,e),j,f),k) = read(A,j) \wedge i = k \wedge read(A,k) \neq g \wedge read(A,j) \neq g$, where i, e, j, f, k, g, are integer-valued constants and A is an array constant.

1. $read(write(A,i,e),j) = e \wedge i \neq j \wedge read(A,j) \neq e$

   (a) `read-over-write 1`:
   $read(write(A,i,e),j) = e \mapsto i = j \wedge e = e$ and $read(A,j) \mapsto f_A(j)$ resulting in
   $$i = j \wedge e = e \wedge i \neq j \wedge f_A(j) \neq e$$
   Moving on with congruence closure over $\mathcal{T}_E$.

$[e]$	$[i]$	$[j]$	$[f_A(j)]$
$[e]$	$[i,j]$	$[f_A(j)]$	

   Now iRj and $i \neq j$, resulting in `unsatisfiable`.

   (b) `read-over-write 2`:
   $read(write(A,i,e),j) = e \mapsto i \neq j \wedge read(A,j) = e$ and $read(A,j) \mapsto f_A(j)$ resulting in
   $$f_A(j) = e \wedge i \neq j \wedge f_A(j) \neq e \wedge i \neq j$$
   Moving on with congruence closure over $\mathcal{T}_E$.

$[e]$	$[i]$	$[j]$	$[f_A(j)]$
	$[i]$	$[j]$	$[f_A(j),e]$

   Now $f_A(j)Re$ and $f_A(j) \neq e$, resulting in `unsatisfiable`.

   Hence, one obtains `unsatisfiable`.

2. $read(write(write(A,i,e),j,f),k) = read(A,j) \wedge i = k \wedge read(A,k) \neq g \wedge read(A,j) \neq g$

   (a) `read-over-write 1`:
   $read(write(write(A,i,e),j,f),k) = read(A,j) \mapsto j = k \wedge f = read(A,j)$ and $read(A,j) \mapsto f_A(j)$, $read(A,k) \mapsto f_A(k)$ resulting in
   $$j = k \wedge f = f_A(j) \wedge i = k \wedge f_A(k) \neq g \wedge f_A(j) \neq g$$

Moving on with congruence closure over $\mathcal{T}_E$.

$[e]$	$[i]$	$[j]$	$[f]$	$[g]$	$[k]$	$[f_A(j)]$	$[f_A(k)]$
$[e]$	$[i,k]$	$[j,k]$	$[f, f_A(j)]$	$[g]$	$[f_A(k)]$		
$[e]$	$[i,k]$	$[j,k]$	$[f, f_A(j), f_A(k)]$	$[g]$			
$[e]$	$[i,j,k]$		$[f, f_A(j), f_A(k)]$	$[g]$			

The rules are applied exhaustively. Hence return `satisfiable`.

According to the relation R one possible interpretation $\mathcal{I}$ satisfying the formula is $e^{\mathcal{I}} := 1$, $g^{\mathcal{I}} := 2$, $i^{\mathcal{I}} = j^{\mathcal{I}} = k^{\mathcal{I}} := 0$ and $A^{\mathcal{I}}[0] := 0$, where $0, 1$ and 2 are some distinct elements in the domain of the same sort (i.e. they are integer).

Problem 2.4

Consider the formula:

$$g(f(a-2)) = a+2 \wedge g(f(b)) = b-2 \wedge b+1 = a-1$$

where a, b are constants interpreted over integers, f, g are function symbols, and $+, -, 1, 2$ are interpreted in the standard way over the integers.

1. Use the Nelson-Oppen decision procedure to determine the satisfiability of the above formula. Use the decision procedures for the theory of uninterpreted functions and use simple mathematical reasoning for deriving new equalities among the constants in the theory of linear integer arithmetic. If the formula is satisfiable, give an interpretation that satisfies the formula.
2. Encode the above formula as an input to the Z3 SMTsolver and evaluate Z3 on your encoding, using http://rise4fun.com/Z3. Interpret the result of Z3. Provide the electronic version of your Z3 encoding together with your solution.

1. • Breaking apart in unit clauses and identifying the theories

$$g(f(a-2)) = a+2,$$
$$g(f(b)) = b-2,$$
$$b+1 = a-1$$

where the red coded elements, represent parts of the integer arithmetic theory elements. Note, to be thorough b was chosen as well.

- As a next step "separate" theories.

$$\begin{array}{lcll} g(f(a-2))=a+2 & \mapsto & g(f(c_0))=c_1 & \wedge \quad c_0=a-2 \wedge c_1=a+2 \\ g(f(b))=b-2 & \mapsto & g(f(c_2))=c_3 & \wedge \quad c_2=b \wedge c_3=b-2 \\ b+1=a-1 & \mapsto & & \wedge \quad b+1=a-1 \end{array}$$

were red is integer arithmetic and blue is the theory of uninterpreted functions.

lin. Arith.	*unint. Func.*	*Equ.*
$c_0 = a-2$	$g(f(c_0)) = c_1$	
$c_1 = a+2$	$g(f(c_2)) = c_3$	
$c_2 = b$		
$c_3 = b-2$		
$b+1 = a-1$		

- Starting by reasoning in *lin. Arith.*. From $b+1=a-1$, one obtains $b=a-2$ and $a=b+2$. Now given, $c_2=b$ and $c_0=a-2$, the equality $c_0=c_2$ is derived. Hence,

lin. Arith.	*unint. Func.*	*Equ.*
$c_0 = a-2$	$g(f(c_0)) = c_1$	$c_0 = c_2$
$c_1 = a+2$	$g(f(c_2)) = c_3$	
$c_2 = b$		
$c_3 = b-2$		
$b+1 = a-1$		

- With this new equality congruence closure, i.e. reasoning in *unint. Func.* can be applied.

$[c_0]$	$[c_1]$	$[c_2]$	$[c_3]$	$[g(f(c_0))]$	$[g(f(c_2))]$
$[c_0,$	$c_2]$	$[c_1,$	$g(f(c_0))]$	$[c_3,$	$g(f(c_2))]$
$[c_0,$	$c_2]$	$[c_1,$	$g(f(c_0)),$	$c_3,$	$g(f(c_2))]$

with the last step being the application of function congruence.

Hence, the equality $c_1 = c_3$ is obtained.

lin. Arith.	*unint. Func.*	*Equ.*
$c_0 = a - 2$	$g(f(c_0)) = c_1$	$c_0 = c_2$
$c_1 = a + 2$	$g(f(c_2)) = c_3$	$c_1 = c_3$
$c_2 = b$		
$c_3 = b - 2$		
$b + 1 = a - 1$		

- Moving back into reasoning in *lin. Arith.*. Since $c_1 = c_3$ it follows that $a + 2 = b - 2$, contradicting the equation $a + 1 = a - 1$, i.e. $a + 4 = b$ and $b = a - 2$ leads to $a + 4 = a - 2$.

All derived equalities are derived by the original set of unit clauses, which induce a contradiction. Hence, unsatisfiability is obtained.

2. `https://rise4fun.com/Z3/6tHqd`

```
(declare-sort U)

(declare-const a Int)
(declare-const b Int)

(declare-fun f (Int) U)
(declare-fun g (U) Int)

(assert (= (g (f (- a 2))) (+ a 2)))
(assert (= (g (f b)) (- b 2)))
(assert (= (+ b 1) (- a 1)))

(check-sat)
```

(Note: `U` was chosen as the output / input of f / g is not specified.)

With output `unsat`, meaning that the formula is not satisfiable. The command `(get-model)` returns an error, as there is no model to be returned.

Alternatively one can also use the following encoding

```
(declare-sort U)

(declare-const a Int)
```

```
(declare-const b Int)

(declare-fun f (Int) U)
(declare-fun g (U) Int)

(assert (and (= (g (f (- a 2))) (+ a 2))
         (and (= (g (f b)) (- b 2)) (= (+ b 1) (- a 1)))))

(check-sat)
```

Problem 2.5

Consider the formula:

$$read(write(A, a, f(b)), c{+}1) = f(d) \wedge f(b) \neq f(d{-}1) \wedge (b{+}1 = d \vee c = a{-}1)$$

where A is an array constant with integer elements, a, b, c, d are constants interpreted over integers, f is a function symbol, and $+, 1$ are interpreted in the standard way over the integers.

1. Use the Nelson-Oppen decision procedure to determine the satisfiability of the above formula. Use the decision procedures for the theory of uninterpreted functions and the theory of arrays, and use simple mathematical reasoning for deriving new equalities among the constants in the theory of linear integer arithmetic. If the formula is satisfiable, give an interpretation that satisfies the formula.
2. Encode the above formula as an input to the Z3 SMTsolver and evaluate Z3 on your encoding, using http://rise4fun.com/Z3. Interpret the result of Z3. Provide the electronic version of your Z3 encoding together with your solution.

Firstly, I assume (hope, as otherwise the following would not be correct) that $-$ is also to be interpreted in the standard way over the integers. Secondly, for the sake of being concise, let w stand for $write$ and r stand for $read$. That is, $read(write(A, a, f(b)), c{+}1) \mapsto r(w(A, a, f(b)), c{+}1)$. (This is really just a measure to be able to write the formulas into a single line. I sincerely hope that this is ok.)

1. Since the formula contains elements of the theory of arrays, one is confronted with a non-convex problem. Hence, due to the clause $(b{+}1 =$

$d \vee c = a - 1$) splitting is required. To be sure, consider

$$
\begin{array}{lcl}
read(write(A, a, f(b)), c+1) = f(d) & \mapsto & p_1 \\
f(b) \neq f(d-1) & \mapsto & p_2 \\
b + 1 = d & \mapsto & p_3 \\
c = a - 1 & \mapsto & p_4
\end{array}
$$

By splitting on p_4 and by unit propagation, one obtains that the formula $p_1 \wedge p_2 \wedge (p_3 \vee p_4)$ is satisfiable in both cases. Hence, further evaluation is required. Starting with p_4 being true and p_3 being false, i.e. $\mathcal{I}(p_1) = true, \mathcal{I}(p_2) = true, \mathcal{I}(p_3) = false$ and $\mathcal{I}(p_4) = true$. Hence, the equality $c = a - 1$ will be passed to the decision proceeder.

- Breaking apart the formula $read(write(A, a, f(b)), c+1) = f(d) \wedge f(b) \neq f(d-1) \wedge c = a-1$ in unit clauses and "separating" theories.

$$
\begin{array}{lcl}
r(w(A, a, f(b)), c+1) = f(d) & \mapsto & \\
c_0 = a \wedge c_1 = b \wedge c_2 = d \wedge c_3 = c+1 & \wedge & c_5 = f(c_1) \wedge c_6 = f(c_2) \quad \wedge \quad c_6 = r(w(A, c_0, c_5), c_3) \\
 & & \\
f(b) \neq f(d-1) & \mapsto & \\
c_1 = b \wedge c_4 = d - 1 & \wedge & c_5 = f(c_1) \wedge c_7 = f(c_4) \\
 & & \\
c = a - 1 & \mapsto & \\
c = a - 1 & &
\end{array}
$$

were red is integer arithmetic, blue is the theory of uninterpreted functions and orange is the theory of arrays.

lin. Arith.	*unint. Func.*	*Arr.*	*Equ.*
$c_0 = a$	$c_5 = f(c_1)$	$c_6 = r(w(A, c_0, c_5), c_3)$	
$c_1 = b$	$c_6 = f(c_2)$		
$c_2 = d$	$c_7 = f(c_4)$		
$c_3 = c + 1$			
$c_4 = d - 1$			
$c = a - 1$			

- Starting with reasoning in *lin. Arith.*. By $c_0 = a$, $c = a - 1$ and $c_3 = c + 1$ one obtains $c_3 = c_0$. Hence,

lin. Arith.	*unint. Func.*	*Arr.*	*Equ.*
$c_0 = a$	$c_5 = f(c_1)$	$c_6 = r(w(A, c_0, c_5), c_3)$	$c_3 = c_0$
$c_1 = b$	$c_6 = f(c_2)$		
$c_2 = d$	$c_7 = f(c_4)$		
$c_3 = c + 1$			
$c_4 = d - 1$			
$c = a - 1$			

- Now, continue reasoning in *Arr.*.
  - `read-over-write 1`:
    $c_6 = r(w(A, c_0, c_5), c_3) \mapsto c_0 = c_3 \wedge c_5 = c_6$ resulting in

    $$c_0 = c_3 \wedge c_0 = c_3 \wedge c_5 = c_6$$

    Applying congruence closure

$[c_0]$	$[c_3]$	$[c_5]$	$[c_6]$
$[c_0,$	$c_3]$	$[c_5,$	$c_6]$

    Hence, a new shared equality $c_5 = c_6$ is obtained.

lin. Arith.	*unint. Func.*	*Arr.*	*Equ.*
$c_0 = a$	$c_5 = f(c_1)$	$c_6 = r(w(A, c_0, c_5), c_3)$	$c_3 = c_0$
$c_1 = b$	$c_6 = f(c_2)$		$c_5 = c_6$
$c_2 = d$	$c_7 = f(c_4)$		
$c_3 = c + 1$			
$c_4 = d - 1$			
$c = a - 1$			

- Now engaging in reasoning over *unint. Func.*. That is, one employs congruence closure over the formula $c_3 = c_0 \wedge c_5 = c_6 \wedge c_5 = f(c_1) \wedge c_6 = f(c_2) \wedge c_7 = f(c_4)$.

$[c_0]$	$[c_1]$	$[c_2]$	$[c_3]$	$[c_4]$	$[c_5]$	$[c_6]$	$[c_7]$	$[f(c_1)]$	$[f(c_2)]$	$[f(c_4)]$
$[c_0,$	$c_3]$	$[c_1]$	$[c_2]$	$[c_4]$	$[c_5,$	$c_6,$	$f(c_1),$	$f(c_2)]$	$[c_7$	$f(c_4)]$

No further equivalence classes can be joined. Hence, satisfiability is obtained. However, in this process no new equivalences are derived.

- Lastly, it is easy to see that the equivalence $c_5 = c_6$ can not be used for further inferences in *lin. Arith.*. Therefore, all rules are applied exhaustively and satisfiability is returned.

lin. Arith.	*unint. Func.*	*Arr.*	*Equ.*
$c_0 = a$	$c_5 = f(c_1)$	$c_6 = r(w(A, c_0, c_5), c_3)$	$c_3 = c_0$
$c_1 = b$	$c_6 = f(c_2)$		$c_5 = c_6$
$c_2 = d$	$c_7 = f(c_4)$		
$c_3 = c + 1$			
$c_4 = d - 1$			
$c = a - 1$			

Finally, since the formula is satisfiable, a satisfying interpretation $\mathcal{I}$ can be given as witness.

$a^{\mathcal{I}} := 1$	$b^{\mathcal{I}} := 1$	$c^{\mathcal{I}} := 0$	$d^{\mathcal{I}} := 1$
$A^{\mathcal{I}}[0] := 0$	$A^{\mathcal{I}}[1] := 0$	$f^{\mathcal{I}}(0) := 0$	$f^{\mathcal{I}}(1) := 1$

with the rest being arbitrary.

2. `https://rise4fun.com/Z3/lfqzQ`

```
(declare-sort U)

(declare-const a Int)
(declare-const b Int)
(declare-const c Int)
(declare-const d Int)

(declare-const ar (Array Int U))

(declare-fun f (Int) U)

(assert (= (select (store ar a (f b)) (+ c 1)) (f d)))
(assert (not (= (f b) (f (- d 1)))))
(assert (or (= (+ b 1) d) (= c (- a 1))))

(check-sat)
(get-model)
```

(Note: `U` was chosen as the output of f is not specified.)

With output `sat`, meaning that the formula is satisfiable. The command `(get-model)` returns

```
(model
       ;; universe for U:
       ;; U!val!1 U!val!0
       ;; -----------
       ;; definitions for universe elements:
                 (declare-fun U!val!1 () U)
                 (declare-fun U!val!0 () U)
       ;; cardinality constraint:
                 (forall ((x U)) (or (= x U!val!1) (= x U!val!0)))
       ;; -----------
       (define-fun c () Int 0)
       (define-fun a () Int 1)
       (define-fun b () Int 38)
       (define-fun d () Int 0)
       (define-fun f ((x!0 Int)) U
                 (ite (= x!0 38) U!val!0
                 (ite (= x!0 0) U!val!0
                 (ite (= x!0 (- 1)) U!val!1 U!val!0)))) )
```

Firstly, the universe of the general sort is defined. Here this particular sort contains the values 1 and 0. That is, to be more specific, `(declare-fun U!val!1 () U)` and `(declare-fun U!val!0 () U)` specify two constants with the name `U!val!1` and `U!val!0` respectively. Then the universe of the sort `U`, is limited to those values due to `(forall ((x U)) (or (= x U!val!1) (= x U!val!0)))`.

```
(model
       ;; universe for U:
       ;; U!val!1 U!val!0
       ;; -----------

       ;; definitions for universe elements:
                 (declare-fun U!val!1 () U)
                 (declare-fun U!val!0 () U)
       ;; cardinality constraint:
                 (forall ((x U)) (or (= x U!val!1) (= x U!val!0)))
```

Following this several integer constants are defined. That is, `c` is set to the value 0, `a` to 1, `b` to 38 and `d` to 0.

```
(define-fun c () Int 0)
(define-fun a () Int 1)
(define-fun b () Int 38)
(define-fun d () Int 0)
```

Lastly, the function `f` is a function from integer values to the sort domain of the sort U, such that `f(38)` evaluates to the element `U!val!0` of the sort `U`, `f(0)` evaluates to the element `U!val!0` of the sort `U`, `f(-1)` evaluates to the element `U!val!1` of the sort `U` and `f(x)` in all other cases evaluates to the element `U!val!0` of the sort `U`.

```
(define-fun f ((x!0 Int)) U
      (ite (= x!0 38) U!val!0
      (ite (= x!0 0) U!val!0
      (ite (= x!0 (- 1)) U!val!1 U!val!0)))) )
```

Alternatively one can also use the following encoding

```
(declare-sort U)

(declare-const a Int)
(declare-const b Int)
(declare-const c Int)
(declare-const d Int)

(declare-const ar (Array Int U))

(declare-fun f (Int) U)

(assert (and (= (select (store ar a (f b)) (+ c 1)) (f d))
         (and (not (= (f b) (f (- d 1))))
         (or (= (+ b 1) d) (= c (- a 1))))))
(check-sat)
(get-model)
```