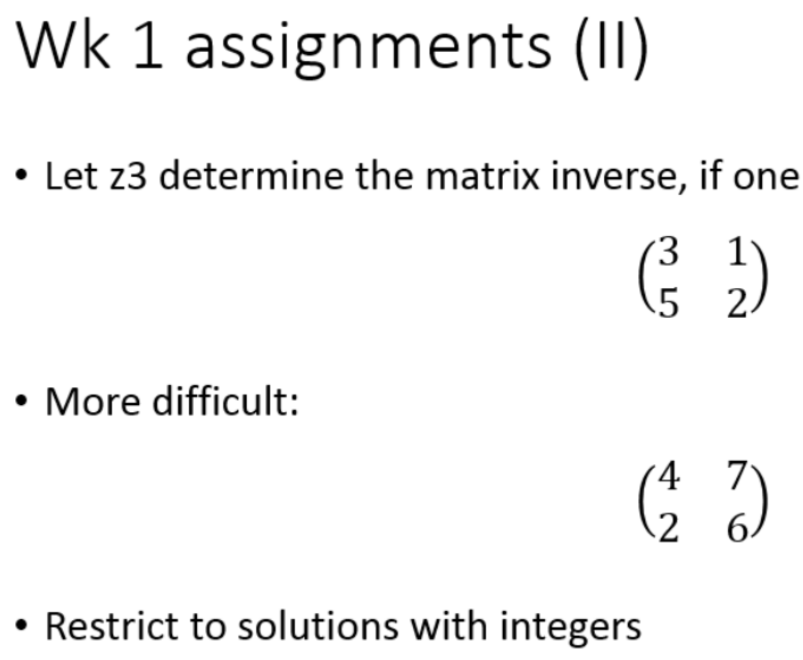
Z3 Assignments, week 1

Group 2



Wk 1 assignments (II)

- Let z3 determine the matrix inverse, if one exists:

$$\begin{pmatrix} 3 & 1 \\ 5 & 2 \end{pmatrix}$$

- More difficult:

$$\begin{pmatrix} 4 & 7 \\ 2 & 6 \end{pmatrix}$$

- Restrict to solutions with integers

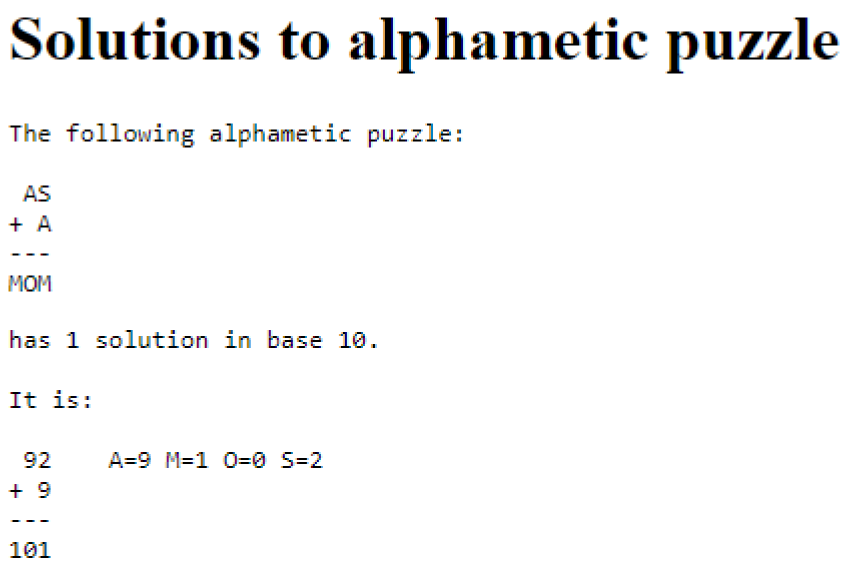
Explanation:

We created a general matrix solver strictly for integers (as seen below).
Conditions of the solver:

- We first find the determinant
- In case of a determinant different than 1, the solver returns UNSAT
- In case of a determinant equal to 1, the solver returns SAT and gives back the model of the solution
- After that, we assign 4 new variables as the inverse variables of the matrix
- Finally, they are multiplied, resulting in the matrix accordingly

```
(declare-const A Int)
(declare-const B Int)
(declare-const C Int)
(declare-const D Int)

(declare-const invA Int)
(declare-const invB Int)
(declare-const invC Int)
(declare-const invD Int)

(declare-const finalA Real)
(declare-const finalB Real)
(declare-const finalC Real)
(declare-const finalD Real)

(declare-const finalAInt Int)
(declare-const finalBInt Int)
(declare-const finalCInt Int)
(declare-const finalDInt Int)
(declare-const N Int)
(declare-const Check Bool)

(declare-const Determinant Int)
(declare-const FinalDeterminant Real)

(assert(= A 3))
(assert(= B 1))
(assert(= C 5))
(assert(= D 2))
(assert(= Check false))

(assert(= Determinant (- (* A D) (* B C))))
(assert(= invA D))
(assert(= invB (- B)))
(assert(= invC (- C)))
(assert(= invD A))

(assert(= (ite (= Determinant 1) false true) Check))
(assert(= FinalDeterminant (/ 1 Determinant)))
```

```
(assert(= finalA (* FinalDeterminant invA )))
(assert(= finalB (* FinalDeterminant invB )))
(assert(= finalC (* FinalDeterminant invC )))
(assert(= finalD (* FinalDeterminant invD )))
(assert (= finalAInt (to_int finalA)))
(assert (= finalBInt (to_int finalB)))
(assert (= finalCInt (to_int finalC)))
(assert (= finalDInt (to_int finalD)))
(check-sat)
(get-model)
```



Solutions to alphametic puzzle

```
The following alphametic puzzle:

 AS
+ A
---
MOM

has 1 solution in base 10.

It is:

 92    A=9 M=1 O=0 S=2
+ 9
---
101
```

Explanation:

We have come up with a solution for this solver by setting the following conditions.
Conditions of the solver:

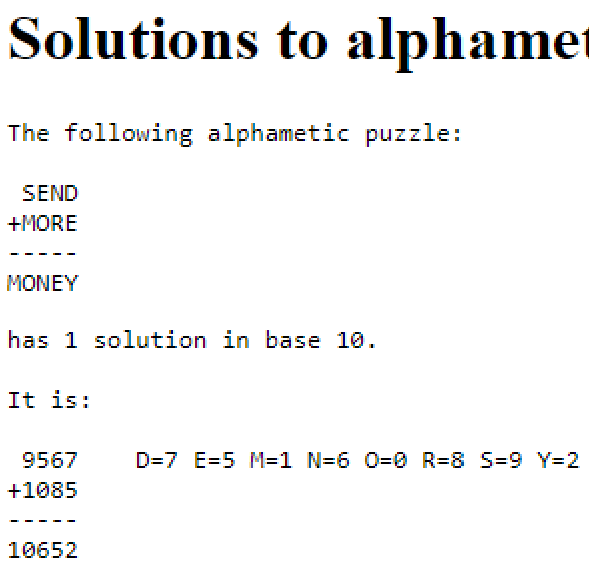
- We make sure each letter has different values.
- We set a constraint for the letters to make sure that the assigned numbers are from 1-9.

- We set constraints for the words to make sure they match the current digits.
- Finally, we assert the result being equal to the sum of the 2 words.

Code:

```
(declare-const A Int)
(declare-const S Int)
(declare-const M Int)
(declare-const O Int)
(declare-const FirstWord Int)
(declare-const SecondWord Int)
(declare-const Result Int)

(assert(and (not(= A S )) (not(= A M)) (not(= A O))))
(assert(and (not(= S A )) (not(= S M)) (not(= S O))))
(assert(and (not(= M S )) (not(= M A)) (not(= M O))))
(assert(and (not(= O S )) (not(= O A)) (not(= O M))))

(assert (and (< A 10) (> A 0)))
(assert (and (< O 10) (>= O 0)))
(assert (and (< S 10) (> S 0)))
(assert (and (< M 10) (> M 0)))


(assert (and(< FirstWord 100) (> FirstWord 9)))
(assert (and(< SecondWord 10) (> SecondWord 0)))
(assert (and(< Result 999) (> Result 99)))

(assert(= FirstWord (+ (* A 10) S)))
(assert(= SecondWord (+ A)))
(assert(= Result (+ (* M 100) (* O 10) M)))
(assert(= Result (+ FirstWord SecondWord)))


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

(Note: Same logic has been used for the second puzzle)

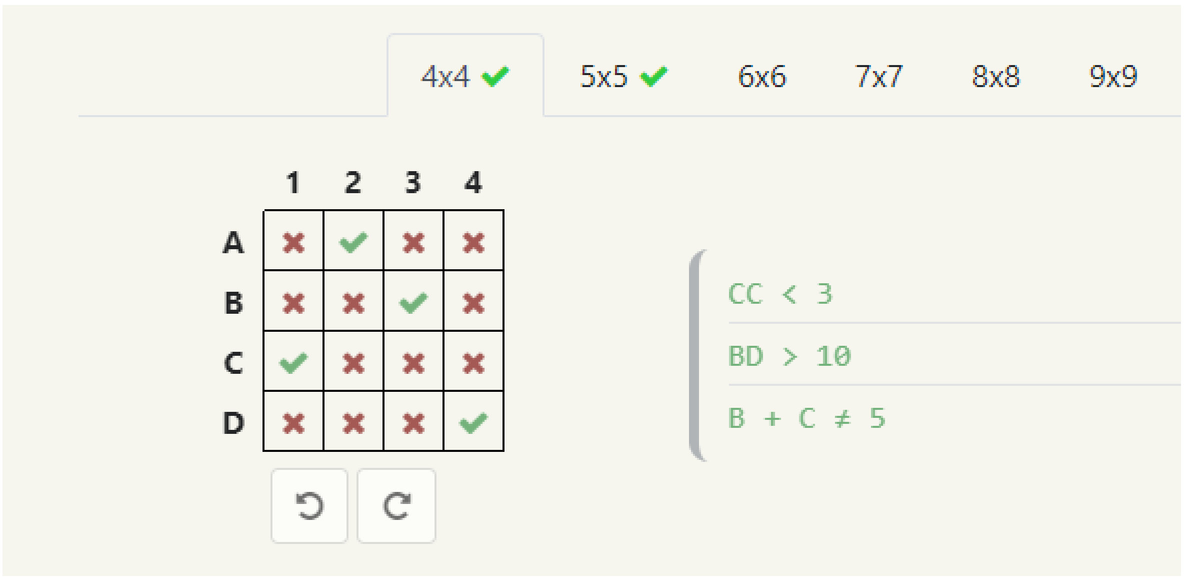


Solutions to alphametic puzzle

```
The following alphametic puzzle:

 SEND
+MORE
-----
MONEY

has 1 solution in base 10.

It is:

 9567    D=7 E=5 M=1 N=6 O=0 R=8 S=9 Y=2
+1085
-----
10652
```

```
(declare-const S Int)
(declare-const E Int)
(declare-const N Int)
(declare-const D Int)
(declare-const M Int)
(declare-const O Int)
(declare-const R Int)
(declare-const Y Int)

(declare-const FirstWord Int)
(declare-const SecondWord Int)
(declare-const Result Int)

(assert(and (not(= S E)) (not(= S N)) (not(= S D)) (not(= S M)) (not(= S
O)) (not(= S R)) (not(= S Y))))
(assert(and (not(= E S)) (not(= E N)) (not(= E D)) (not(= E M)) (not(= E
O)) (not(= E R)) (not(= E Y))))
(assert(and (not(= N E)) (not(= N S)) (not(= N D)) (not(= N M)) (not(= N
O)) (not(= N R)) (not(= N Y))))
(assert(and (not(= D E)) (not(= D N)) (not(= D E)) (not(= D M)) (not(= D
O)) (not(= D R)) (not(= D Y))))
```

```
(assert(and (not(= M E)) (not(= M N)) (not(= M D)) (not(= M S)) (not(= M
O)) (not(= M R)) (not(= M Y))))
(assert(and (not(= O E)) (not(= O N)) (not(= O D)) (not(= O M)) (not(= O
S)) (not(= O R)) (not(= O Y))))
(assert(and (not(= R E)) (not(= R N)) (not(= R D)) (not(= R M)) (not(= R
O)) (not(= R S)) (not(= R Y))))
(assert(and (not(= Y E)) (not(= Y N)) (not(= Y D)) (not(= Y M)) (not(= Y
O)) (not(= Y R)) (not(= Y S))))
(assert (and (< S 10) (> S 0)))
(assert (and (< E 10) (>= E 0)))
(assert (and (< N 10) (>= N 0)))
(assert (and (< D 10) (>= D 0)))
(assert (and (< M 10) (> M 0)))
(assert (and (< O 10) (>= O 0)))
(assert (and (< R 10) (>= R 0)))
(assert (and (< Y 10) (>= Y 0)))



(assert (and(< FirstWord 10000) (> FirstWord 999)))
(assert (and(< SecondWord 10000) (> SecondWord 999)))
(assert (and(< Result 100000) (> Result 9999)))

(assert(= FirstWord (+ (* S 1000) (* E 100) (* N 10) D)))
(assert(= SecondWord (+ (* M 1000) (* O 100) (* R 10) E)))
(assert(= Result (+ (* M 10000) (* O 1000) (* N 100) (* E 10) Y)))
(assert(= Result (+ FirstWord SecondWord)))


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



	1	2	3	4
A	✗	✓	✗	✗
B	✗	✗	✓	✗
C	✓	✗	✗	✗
D	✗	✗	✗	✓

```
CC < 3
BD > 10
B + C ≠ 5
```

Explanation:

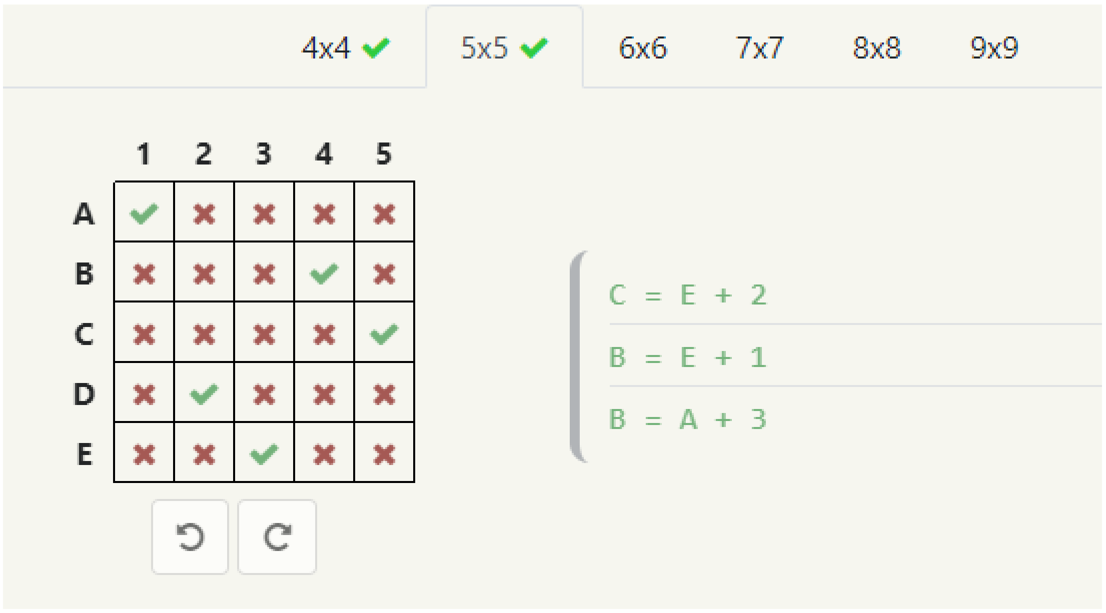
To solve this puzzle we came up with a solution for this solver by setting the following conditions.
Conditions of the solver:

- The letters must be no greater than 4 and no less than 0.
- Make sure the letters are not equal.
- Then we have to check if C * C less than 3.
- And B * D must be greater than 10.
- And B + C must not be equal to 5.

Code 4 x 4:

```
(declare-const A Int)
(declare-const B Int)
(declare-const C Int)
(declare-const D Int)

(assert (and(< A 5) (> A 0)))
(assert (and(< B 5) (> B 0)))
(assert (and(< C 5) (> C 0)))
(assert (and(< D 5) (> D 0)))

(assert(and (not(= A B )) (not(= A C)) (not(= A D))))
(assert(and (not(= B A )) (not(= B C)) (not(= B D))))
(assert(and (not(= C A )) (not(= C B)) (not(= C D))))
(assert(and (not(= D A )) (not(= D B)) (not(= D C))))

(assert (< (* C C) 3))
(assert (> (* B D) 10))
(assert (not (= (+ B C ) 5)))

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

(Note: Same logic has been used for the second puzzle)



	1	2	3	4	5
A	✓	✗	✗	✗	✗
B	✗	✗	✗	✓	✗
C	✗	✗	✗	✗	✓
D	✗	✓	✗	✗	✗
E	✗	✗	✓	✗	✗

C = E + 2

B = E + 1

B = A + 3

Code:

```
 (declare-const A Int)
(declare-const B Int)
(declare-const C Int)
(declare-const D Int)
(declare-const E Int)


(assert (and (> A 0) (<= A 5)))
(assert (and (> B 0) (<= B 5)))
(assert (and (> C 0) (<= C 5)))
(assert (and (> D 0) (<= D 5)))
(assert (and (> E 0) (<= E 5)))


(assert(and (not(= A B )) (not(= A C)) (not(= A D))
(not(= A E))))
(assert(and (not(= B A )) (not(= B C)) (not(= B D))
(not(= B E))))
(assert(and (not(= C A )) (not(= C B)) (not(= C D))
(not(= C E))))
(assert(and (not(= D A )) (not(= D B)) (not(= D C))
(not(= D E))))

(assert (= (+ E 2) C))
(assert (= (+ E 1) B))
(assert (= (+ A 3) B))


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