CPSC-354 Report

Annika Brown
Chapman University

November 24, 2025

Abstract

Contents

1 Introduction

2 Week by Week

2.1 Week 1

2.1.1 MU Puzzle

It is impossible to solve the MU puzzle. The only way to change the amount of I's is to double them with rule 2, or subtract 3 with rule 3. In order to get rid of all the I's you would have to have them in groups of 3. A power of x is divisible by 3 only if x is a multiple of 3, because the prime factorization of a number must include the prime number 3 for the number to be divisible by 3. So, you would have to get an original group divisible by 3, which is impossible.

2.2 Week 2

2.2.1 Rewriting

1. A={}
Empty ARS
Terminating: Yes
Confluent: Yes
Has Unique Normal Forms: Yes

2. A={a} and R={}

$$\begin{tikzcd} a \end{tikzcd}$$

Terminating: Yes
Confluent: Yes
Has Unique Normal Forms: Yes

3. A={a} and R={(a,a)}

$$\begin{tikzcd} a \arrow[loop above] \end{tikzcd}$$

Terminating: No
Confluent: Yes
Has Unique Normal Forms: No

4. A={a,b,c} and R={(a,b),(a,c)}

$$\begin{tikzcd} & b \\ a \arrow[ur] \arrow[dr] & \\ & c \end{tikzcd}$$

Terminating: Yes
Confluent: No
Has Unique Normal Forms: No

5. A={a,b} and R={(a,a),(a,b)}

$$\begin{tikzcd}
a \arrow[loop above] \arrow[r] & b
\end{tikzcd}$$

Terminating: No
Confluent: Yes
Has Unique Normal Forms: Yes

6. A={a,b,c} and R={(a,b),(b,b),(a,c)}

$$\begin{tikzcd}
& b \arrow[loop above] \\
a \arrow[ur] \arrow[dr] & \\
& c
\end{tikzcd}$$

Terminating: No
Confluent: No
Has Unique Normal Forms: No

7. A={a,b,c} and R={(a,b),(b,b),(a,c),(c,c)}

$$\begin{tikzcd}
& b \arrow[loop above] \\
a \arrow[ur] \arrow[dr] & \\
& c \arrow[loop above]
\end{tikzcd}$$

Terminating: No
Confluent: No
Has Unique Normal Forms: No

confluent	terminating	has unique normal forms	example
True	True	True	a
True	True	False	IMPOSSIBLE
True	False	True	$\begin{tikzcd} a \arrow[loop above] \arrow[r] & b \end{tikzcd}$
True	False	False	$\begin{tikzcd} a \arrow[loop above] \end{tikzcd}$
False	True	True	IMPOSSIBLE
False	True	False	$\begin{tikzcd} & b \\ a \arrow[ru] \arrow[rd] & \\ & c \end{tikzcd}$
False	False	True	IMPOSSIBLE
False	False	False	$\begin{tikzcd} & b \arrow[loop above] \\ a \arrow[ru] \arrow[rd] & \\ & c \arrow[loop above] \end{tikzcd}$

2.3 Week 3

2.3.1 Homework

Exercise 5

ab → ba
ba → ab
aa →
b →

abba → aba → aa →
bababa → ababa → aaba → aaa → a

The ARS isn't terminating because it has the loop ab → ba, ba → ab.

In the above example, abba → , bababa → a, which makes them non-equivalent. There are infinitely many non-equivalent strings, because all strings with an even number of a's are only equivalent to each other, and all strings with an odd number of a's are only equivalent to each other.

There are 2 equivalence classes, all strings with an odd number of a's that have normal form a, and all strings with an even number of a's that have normal form of an empty string.

By getting rid of the first 2 rules, the ARS is terminating, with the same equivalence classes. There is no longer a loop, and you can still get rid of the same letters.

How would you have to modify the ARS so that there is only one equivalence class?

Exercise 5b

ab $\rightarrow$ ba
ba $\rightarrow$ ab
aa $\rightarrow$ a
b $\rightarrow$

abba $\rightarrow$ aba $\rightarrow$ aa $\rightarrow$ a
bababa $\rightarrow$ ababa $\rightarrow$ aaba $\rightarrow$ aaa $\rightarrow$ aa $\rightarrow$ a

The ARS isn't terminating because it has the loop ab $\rightarrow$ ba, ba $\rightarrow$ ab.

bbb and aba are not equivalent. There are infinitely many non-equivalent strings, any string that has all b's is not equivalent to any string that has at least one a in it.

There are 2 equivalence classes, all strings that have no a's have the normal form of an empty string. All strings that have at least one a have the normal form a.

By getting rid of the first 2 rules, the ARS is terminating, with the same equivalence classes. There is no longer a loop, and you can still get rid of the same letters.

What would happen if aa $\rightarrow$ a was changed to aaa $\rightarrow$ a?

2.4 Week 4

2.4.1 Homework

4.1

```
while b != 0:
    temp = b
    b = a mod b
    a = temp
return a
```

This algorithm always terminates under the condition that $a, b \in \mathbb{N}$.

We can define this ARS as: $A = (a, b)|a, b \in \mathbb{N}$, with transition, (a, b) $\rightarrow$ (b, a mod b).

A measure function is: $\phi(a, b) = b$.

Since a $\rightarrow$ b every iteration, if b is decreasing, so is a. In the transition, b $\rightarrow$ a mod b, a mod b will always be less than b, so, $\phi(a, b) > \phi($b, a mod b$)$. Since this algorithm has a measure function, it terminates.

4.2

```
function merge_sort(arr, left, right):
    if left >= right:
        return
    mid = (left + right) / 2
    merge_sort(arr, left, mid)
    merge_sort(arr, mid+1, right)
    merge(arr, left, mid, right)
```

Prove that $\phi(left, right) = right - left + 1$ is a measure function.

For recursive calls, $left \leq mid < right$.

For the left side recursive call:
merge sort(arr, left, mid) $=$ mid - left $+$ 1
right $>$ mid, so, right - left $+$ 1 $>$ mid - left $+$ 1

$\phi(left, right) > \phi(left, mid)$
This call terminates

For the right side recursive call:
merge sort(arr, mid + 1, right) = right - (mid + 1) + 1
$mid \geq left$, so, mid + 1 > left
right - left + 1 > right - (mid + 1) + 1
$\phi(left, right) > \phi(mid + 1, right)$
This call terminates

Since both calls terminate, $\phi(left, right) = right - left + 1$ is a measure function for merge sort.

2.5 Week 5

2.5.1 Homework

(λf.λx.f(f(x))) (λf.λx.(f(f(f x))))

alpha rule:
(λg.λy.g(g(y))) (λf.λx.(f(f(f x))))

beta rule:
(λy.(λf.λx.(f(f(f x)))) ((λf.λx.(f(f(f x))))(y)))

beta rule:
(λy.(λf.λx.(f(f(f x)))) (λx.(y(y(y x)))))

alpha rule:
(λy.(λf.λz.(f(f(f z)))) (λx.(y(y(y x)))))

beta rule:
(λy.(λz.((λx.(y(y(y x))))((λx.(y(y(y x))))((λx.(y(y(y x))))z)))))

beta rule:
(λy.(λz.((λx.(y(y(y x))))((λx.(y(y(y x))))(y(y(y z)))))))

beta rule:
(λy.(λz.((λx.(y(y(y x))))(y(y(y (y(y(y z)))))))))

beta rule:
(λy.(λz.(y(y(y(y(y(y(y(y(y z))))))))))))

2.6 Week 6

2.6.1 Homework

let rec fact = λn. if n=0 then 1 else n * fact (n-1) in fact 3

-> <def of let rec>
let fact = (fix (λfact. λn. if n=0 then 1 else n * fact (n-1))) in fact 3

-> <def of let>
(λfact. fact 3) (fix (λfact. λn. if n=0 then 1 else n * fact (n-1)))

-> <beta rule: substitute fix F>
(fix (λfact. λn. if n=0 then 1 else n * fact (n-1))) 3

-> <def of fix>
((λfact. λn. if n=0 then 1 else n * fact (n-1)) (fix (λfact. λn. if n=0 then 1 else n * fact (n-1)))) 3

-> <beta rule: substitute fix F>
(λn. if n=0 then 1 else n * (fix (λfact. λn. if n=0 then 1 else n * fact (n-1))) (n-1)) 3

-> <beta rule: substitute 3>
if 3=0 then 1 else 3 * (fix (λfact. λn. if n=0 then 1 else n * fact (n-1))) (3-1)

-> <def of if>
3 * (fix (λfact. λn. if n=0 then 1 else n * fact (n-1))) 2

-> <def of fix>
3 * ((λfact. λn. if n=0 then 1 else n * fact (n-1)) (fix (λfact. λn. if n=0 then 1 else n * fact (n-1)))) 2

-> <beta rule>
3 * (λn. if n=0 then 1 else n * (fix (λfact. λn. if n=0 then 1 else n * fact (n-1))) (n-1)) 2

-> <beta rule>
3 * (if 2=0 then 1 else 2 * (fix (λfact. λn. if n=0 then 1 else n * fact (n-1))) (2-1))

-> <def of if>
3 * (2 * (fix (λfact. λn. if n=0 then 1 else n * fact (n-1))) 1)

-> <def of fix>
3 * (2 * ((λfact. λn. if n=0 then 1 else n * fact (n-1)) (fix (λfact. λn. if n=0 then 1 else n * fact (n-1)))) 1)

-> <beta rule>
3 * (2 * (λn. if n=0 then 1 else n * (fix (λfact. λn. if n=0 then 1 else n * fact (n-1)))(n-1)) 1)

-> <beta rule>
3 * (2 * (if 1=0 then 1 else 1 * (fix (λfact. λn. if n=0 then 1 else n * fact (n-1)))(1-1)))

-> <def of if>
3 * (2 * (1 * (fix (λfact. λn. if n=0 then 1 else n * fact (n-1))) 0))

-> <def of fix>
3 * (2 * (1 * ((λfact. λn. if n=0 then 1 else n * fact (n-1)) (fix (λfact. λn. if n=0 then 1 else n * fact (n-1))))
0))

-> <beta rule>
3 * (2 * (1 * (λn. if n=0 then 1 else n * (fix (λfact. λn. if n=0 then 1 else n * fact (n-1))) (n-1)) 0))

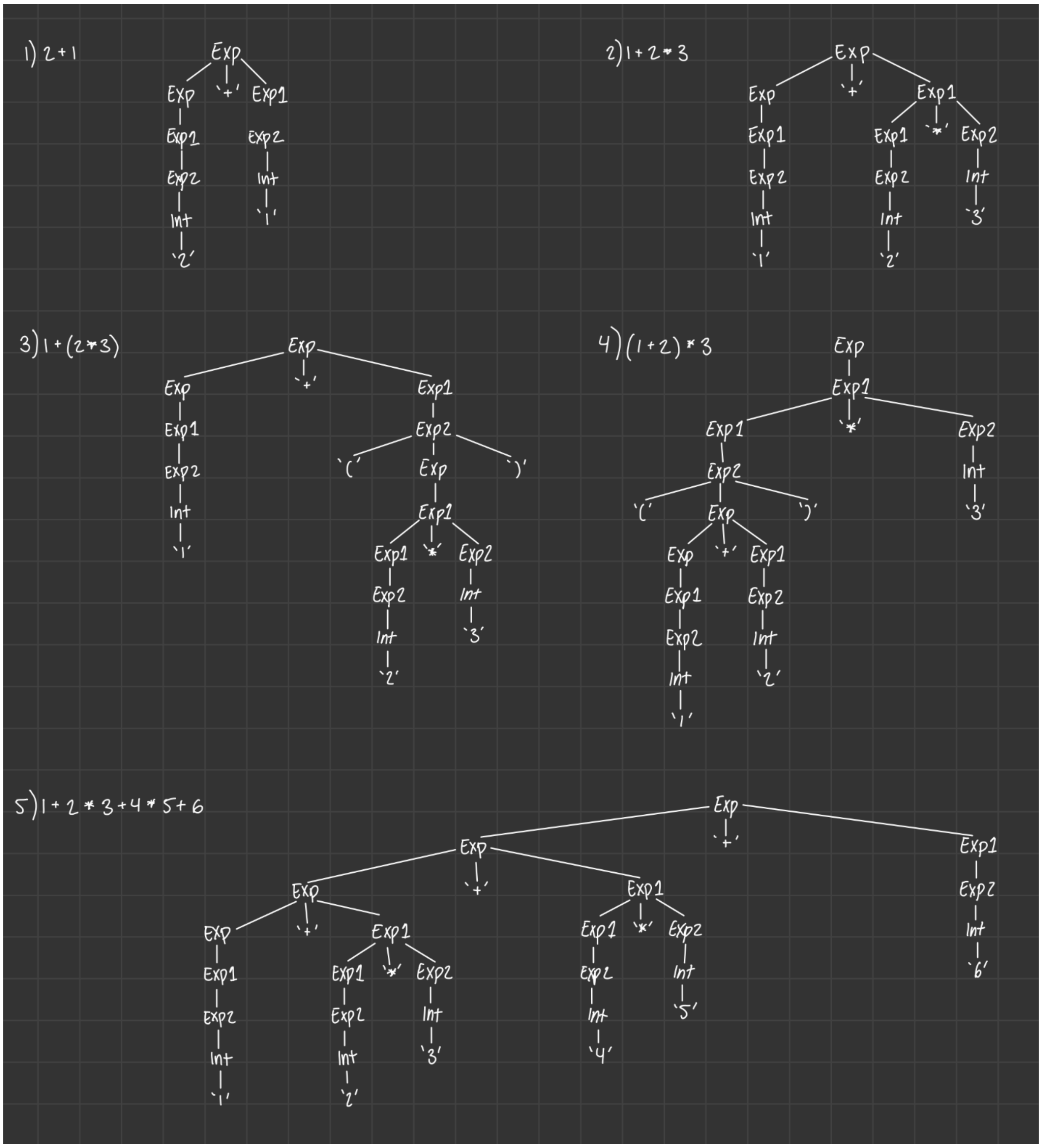
-> <beta rule>
3 * (2 * (1 * (if 0=0 then 1 else 0 * (fix (λfact. λn. if n=0 then 1 else n * fact (n-1))) (0-1))))

-> <def of if>
3 * (2 * (1 * (1)))

-> <multiplication>
6

2.7 Week 7

2.7.1 Homework



2.8 Week 8

2.8.1 Homework

```
Level 5: a + (b + 0) + (c + 0) = a + b + c
rw[add_zero]
rw[add_zero]
rfl

Level 6: a + (b + 0) + (c + 0) = a + b + c
rw[add_zero c]
rw[add_zero]
rfl

Level 7: For all natural numbers a, we have succ(a) = a + 1
rw[one_eq_succ_zero]
rw[add_succ]
rw[add_zero]
rfl

Level 8: 2 + 2 = 4
rw[four_eq_succ_three]
rw[three_eq_succ_two]
rw[two_eq_succ_one]
rw[one_eq_succ_zero]
rw[add_succ]
rw[add_succ]
rw[add_zero]
rfl

Natural Language Proof of Level 5:
a + (b + 0) + (c + 0) = a + b + c
a + b + c = a + b + c               by algorithm 1: addition (m + 0 = 0)
True by reflexivity
```

2.9 Week 9

2.9.1 Homework

```
ADDITION WORLD LEVEL 5

WITH INDUCTION: a + b + c = a + c + b

induction b
rw[add_zero]
rw[add_zero]
rfl
rw[add_succ]
rw[succ_add]
rw[add_succ]
rw[n_ih]
rfl
```

```
a + b + c = a + c + b

Induction on b:
Basis:   a + 0 + c = a + c + 0
   a + c = a + c + 0       def of +
   a + c = a + c        def of +
   True by reflexivity

Inductive Hypothesis: a + n + c = a + c + n

Inductive Step: a + n + c = a + c + n
       (a + n) + c = a + c + n     def of +
       (a + n + c) =  a + c + n     def of +
       (a + n + c) =  a + (c + n) def of +
       (a + n + c) =  (a + c + n) def of +
       (a + c + n) = (a + c + n) inductive hypothesis
       True by reflexivity

WITHOUT INDUCTION: a + b + c = a + c + b

rw[add_assoc]
rw[add_comm b]
rw[add_assoc]
rfl

(a + b) + c = (a + c) + b
a + (b + c) = (a + c) + b associativity
a + (c + b) = (a + c) + b commutativity
(a + c) + b = (a + c) + b associativity
True by reflexivity
```

2.10 Week 10

2.10.1 Homework

PARTY INVITES

LEVEL 6: example (C D S: Prop) (h : C D → S) : C → D → S := by
exact $\lambda cd \mapsto h{<}c, d{>}$

LEVEL 7: example (C D S: Prop) (h : C → D → S) : C D → S := by
exact $\lambda c \mapsto hc.leftc.right$

LEVEL 8: example (C D S : Prop) (h : (S → C) (S → D)) : S → C D := by
exact $\lambda s \mapsto {<}h.lefts, h.rights{>}$

LEVEL 9: example (R S : Prop) : R → (S → R) (¬S → R) := by
exact $\lambda r \mapsto {<}\lambda_{,} \to r, \lambda_{,} \to r{>}$

Discord Question: Why do we need to use a placeholder when there is no evidence?

2.11 Week 11

2.11.1 Homework

```
FALSIFICATION

LEVEL 9: example (A P : Prop) (h : P → ¬A) : ¬(P  A) := by
exact  ⟨p, a⟩ => (h p) a

LEVEL 10: example (A P : Prop) (h: ¬(P  A)) : P → ¬A := by
exact  p a => h ⟨p, a⟩

LEVEL 11: example (A : Prop)(h : ¬¬¬A) : ¬A := by
exact  a => h ( na => na a)

LEVEL 12: example (B C : Prop) (h : ¬(B → C)) : ¬¬B := by
exact  nb => h ( b => False.elim (nb b))
```

2.12 Week 12

2.12.1 Homework

```
hanoi 5 0 2
 hanoi 4 0 1
  hanoi 3 0 2
   hanoi 2 0 1
    hanoi 1 0 2 = move 0 2
    move  0 1
    hanoi 1 2 1 = move 2 1
   move 0 2
   hanoi 2 1 2
    hanoi 1 1 0 = move 1 0
    move  1 2
    hanoi 1 0 2 = move 0 2
        move 0 1
        hanoi 3 2 1
            hanoi 2 2 0
                hanoi 1 2 1 = move 2 1
                move 2 0
                hanoi 1 1 0 = move 1 0
            move 2 1
            hanoi 2 0 1
                hanoi 1 0 2 = move 0 2
                move 0 1
                hanoi 1 2 1 = move 2 1
    move 0 2
    hanoi 4 1 2
        hanoi 3 1 0
            hanoi 2 1 2
                hanoi 1 1 0 = move 1 0
                move 1 2
                hanoi 1 0 2 = move 0 2
            move 1 0
            hanoi 2 2 0
                hanoi 1 2 1 = move 2 1
                move 2 0
                hanoi 1 1 0 = move 1 0
        move 1 2
        hanoi 3 0 2
            hanoi 2 0 1
                hanoi 1 0 2 = move 0 2
                move 0 1
                hanoi 1 2 1 = move 2 1
            move 0 2
            hanoi 2 1 2
                hanoi 1 1 0 = move 1 0
                move 1 2
                hanoi 1 0 2 = move 0 2
```

2.13 Week 13

2.13.1 Homework

```
2. I added ((ab)c)
    It should reduce to a b c
    It reduced to (ab c)

   I added (\x. (\y. x)) y
   It became (\Var1.y), which is correct

   I added (\x. x x) (\x. x x)
   It resulted in: recursion error: maximum recursion depth exceeded

3. Avoiding capture substitution works by adding new variables like Var1 since in some cases there are t
   I added (\x. (\y. x)) y, and it became (\Var1.y), so it avoided capture substitution.
   It is implemented by having it generate a new variable name if the expression starts with a lambda ar

4. I don't always get the expected result.
    I added ((ab)c), and it should reduce to a b c, but it reduced to (ab c), which is correct, but not
    Not all computations reduce to normal form.
    I added (\x. x x) (\x. x x)
    It resulted in: recursion error: maximum recursion depth exceeded

5. The smallest expression I can find that does not reduce to normal form is (\x. x x) (\x. x x)

7.
((\m.\n. m n) (\f.\x. f (f x))) (\f.\x. f (f (f x)))
((\Var1. (\f.\x. f (f x)) Var1) ) (\f.\x. f (f (f x)))
((\Var1. (\Var2.\x. Var2 (Var2 x)) Var1) ) (\f.\x. f (f (f x)))
((\Var2.\Var3. Var2 (Var2 Var3)) (\f.\x. f (f (f x))) )
((\Var2.\Var4. Var2 (Var2 Var4)) (\f.\x. f (f (f x))) )
(\Var5.((\f.(\x.(f (f (f x))))) ((\f.(\x.(f (f (f x))))) Var5)))
```

```
8.
12: eval (((\m.(\n.(m n))) (\f.(\x.(f (f x))))) (\f.(\x.(f x))))
39: eval ((\m.(\n.(m n))) (\f.(\x.(f (f x)))))
 39: eval (\m.(\n.(m n)))
 44: sub ( (\n.(m n)), m, (\f.(\x.(f (f x)))) )
  82: sub ( (m Var1), m, (\f.(\x.(f (f x)))) )
   82: sub ( (m n), n, Var1 )
    85: sub ( m, n, Var1 )
    85: sub ( n, n, Var1 )
   85: sub ( m, m, (\f.(\x.(f (f x)))) )
   85: sub ( Var1, m, (\f.(\x.(f (f x)))) )
 45: eval (\Var1.((\f.(\x.(f (f x)))) Var1))
44: sub ( ((\f.(\x.(f (f x)))) Var1), Var1, (\f.(\x.(f x))) )
85: sub ( (\f.(\x.(f (f x)))),  Var1, (\f.(\x.(f x))) )
 82: sub ( (\Var3.(Var2 (Var2 Var3))), Var1, (\f.(\x.(f x))) )
 82: sub ( (\x.(f (f x))), f, Var2 )
  82: sub ( (f (f x)), x, Var4 )
   85: sub ( f, x, Var4 )
   85: sub ( (f x), x, Var4 )
    85: sub ( f, x, Var4 )
85: sub ( x, x, Var4 )
    82: sub ( (f (f Var4)), f, Var2 )
     85: sub ( f, f, Var2 )
     85: sub ( (f Var4), f, Var2 )
      85: sub ( f, f, Var2 )
      85: sub ( Var4, f, Var2 )
85: sub ( Var1,  Var1, (\f.(\x.(f x))) )
 45: eval ((\Var2.(\Var5.(Var2 (Var2 Var5)))) (\f.(\x.(f x))))
  39: eval (\Var2.(\Var5.(Var2 (Var2 Var5))))
  44: sub ( (\Var5.(Var2 (Var2 Var5))), Var2, (\f.(\x.(f x))) )
  45: eval (\Var6.((\f.(\x.(f x))) ((\f.(\x.(f x))) Var6)))
```

Discord Question: Are there any situations where the debugger can't or shouldn't be used?

3 Essay

4 Evidence of Participation

5 Conclusion

References

[BLA] Author, Title, Publisher, Year.