Discrete Structures
Computer Assignment

Tony Lopar s1013792
Carlo Jessurun s1013793
Marnix Dessing s1014097

February 2, 2018

1 Language

The language we used is Python. This is the only sensible choice based on the fact the framework given for the assignment is written in Python. And due to the time frame we don't have time to write our own framework. Also Python has a calculus library Sympy (see next section). Since we are allowed to use Simpy, it was the most obvious choice to work with that.

2 Libraries

We use the following libraries:

1. The framework that comes with the assignment.
2. Sympy

2.1 Framework

The framework pretty much explains its self. We had to implement the parsing of the F(n) our self, after the associated homogeneous part was parsed. We had the following approach: tony jij hebt dit gedaan toch?

2.2 Sympy

Homogeneous

Step 1
For implementing the homogeneous part of the algorithm we were able to extract every step into a seperate function. The implementation for solving homogeneous recurrence formulas is as follows:

```
def solve_homogeneous_equation(init_conditions, associated):
    # Step 1: Rewrite in the default form

    # Step 2: Determine characteristic equation
    polynomial = build_polynomial(associated)

    # Step 3: Find roots and multiplicities of characteristic equation
    solutionsWithMultiplicity = solve_polynomial_roots(polynomial)

    # Step 4: Write down general solution
    generalSolution = build_general_solution(solutionsWithMultiplicity)

    # Step 5: Use initial conditions to determine values of the parameters
    alphaSolutions = solve_alphas(generalSolution, init_conditions)
    directFormula = insert_alphas_in_solution(alphaSolutions, generalSolution)

    debug_print("Final solution: S(n)=" + directFormula)
    return directFormula
```

Step 2
In the second step, we use string operations to construct a polynomial. Based on the associated parts that were already parsed for us by the framework.

Step 3
For finding the roots and their multiplicities we use sympy:

```
roots(Eq(parse_expr(polynomial), 0), r)
```

We use sympy roots with the equation and the symbol to solve ('r') for finding a key value list with roots and multiplicities. For passing the right equation type we use Eq and pare_expr to construct a equation that we solve for 0.

Step 4
During the fourth step, we use only some string operations to construct a general solution containing the alphas (a0 ... an) and roots based on their multiplicities.

Step 5
With the fifth step, we construct different equations for all the initial conditions. We replace the ns in the general solution from 4 with the n from an initial condition. Next we create an sympy equation Eq with the expression and its expected result.

```
exrp = parse_expr(generalSolution.replace("n", str(n)))
eq = Eq(exrp, int(init_conditions[n]))
equations.append(eq)
alphaSolutions = solve(equations)
```

Finaly we get a list of all the alpha values, for the corresponding equations created above, by calling solve for the list of equations:

```
alphaSolutions = solve(equations)
```

We now use string operations to replace all the alphas (a0 .. an) with the values in the list *alphaSolutions*.

We now have a direct solution for our recurrence relation.

3 Problems and solutions

Problem 1:

Solving a homogeneous relation for some init values was impossible...