Summer School of Science S3++ 2015

AUTOMATING BIOLOGY

An end to end approach

Team- Sofia Gomez, Đesika Kolarić, Sergi Macia, Mislav Unger
Project Leaders- Vishal Gupta, Ismael Gomez
3.8.2015.

ABSTRACT

According to recent publications [1, 2] 70% of the published scientific papers cannot be reproduced. Consequently, reproducibility has turned into one of the major problems of today's science. Ambiguity, human error and lack of information in the protocols are some of the main problematic factors. In this project, we develop two libraries written in C language that allows the user to describe laboratory protocols in an unambiguous form and send them as orders to a hypothetical robot. We analyze the differences between the two results with a focus on the level of abstraction used.

INTRODUCTION

Reproducibility is a major problem in science. Up to 70% of published papers cannot be reproduced in other labs due to many reasons, but mainly ambiguity, human error and lack of information. Protocols written in natural languages are not precise enough to capture all the important details to ensure that the steps in the protocol are implemented exactly the same way each time. Specifically, imprecise measurements (e.g. reagent quantities, temperature, incubation times, etc.) and ambiguous expressions (e.g. to mix gently, to flicker, overnight, etc.) while describing protocols add to the problem of reproducibility. Automation is applied in biology frequently to do high throughput research. Automation offers the advantage of using robots to move fluid volumes thereby helping to avoid reproducibility issues.

Previous works[3] have shown that transcribing protocols from natural language (English etc.) into formal languages (i.e. programming computer languages) provides for precise instructions that can be executed later in the wet-lab on conventional lab equipment, pipetting robots or microfluidic devices. While robots are designed to perform protocol actions done by scientists using the conventional equipment found in a lab, microfluidics integrate all these processes in a single device.

To attempt to solve the problem of reproducibility we aimed at developing a library of functions which represent the main actions (i.e. lab operations like heating, cooling, mixing, spinning) performed in biological experiments. In this regard, we learnt basic

[1] Prinz, F., Schlange, T. & Asadullah, K. *Nature Rev. Drug Discov.* 10, 712 (2011).
[2] Begley, C. G. & Ellis, et al L. M. *Nature* **483**, 531–533 (2012)
[3] V. Ananthanarayanan and W. Thies, 2010, Tali Herzka (2015)

programming skills in C language to develop the required library in order to express protocols.

We hypothesize that given a well defined library, the ambiguity in protocol descriptions can be removed. We make a noteworthy assumption that a limited set of functions can describe all possible lab protocols. We also assumed that our functions could eventually be implemented by hardware platforms.

We tested our language by using its functions to transcribe a protocol of bacterial transformation and validated the language by asking external users to compare the natural language protocol to the transcribed protocol for its expressivity and ambiguity. This part is further explained in detail in the results section.

MATERIALS AND METHODS

For the development of the library for the description of biological protocols we have used C programming language, with Eclipse IDE as programming environment. All the development has been performed over a Windows 7 operating system, using GCC as a compiler (through MinGW).

We used Bacterial Transformation protocol as a starting point. We analyzed procedures and parameters performed in the protocol, in order to identify the essential actions necessary for the description. After this analysis we defined a set of functions and structures capable of formal protocol description.

We used structured programming as a model paradigm. We capture the information about the materials, equipment and coordinates in structures and we capture the information about actions using functions.

We attacked this problem using two different approaches: in the first approach, we adopted a very specific approach, in which structures describe the materials and equipment needed in the protocol (i.e., chemicals, cells, recipients and machines), and the functions describe the actions performed over such materials and equipment; in the second approach, functions and structures are defined to perform the protocol in a more abstract way, focusing the description on the materials without specifying the labware.

The example protocol is also used for testing our libraries (Standard heat-shock transformation of chemically competent bacteria). A detailed description of the protocol is described below.

Protocol-

1. Take competent cells out of -80°C and thaw on ice (approximately 20-30 min).

2. Take agar plates (containing the appropriate antibiotic) out of 4°C to warm up to room temperature or place in 37°C incubator.

3. Mix 1 to 5µl of DNA (usually 10pg to 100ng) into 20-50µL of competent cells in a microcentrifuge or falcon tube. GENTLY mix by flicking the bottom of the tube with your finger a few times.

4. Place the competent cell/DNA mixture on ice for 20-30min.

5. Heat shock each transformation tube by placing the bottom 1/2 to 2/3 of the tube into a 42°C water bath for 30-60 seconds

6. Put the tubes back on ice for 2 min.

7. Add 250-500µl LB or SOC media (without antibiotic) and grow in 37°C shaking incubator for 45min.

Note: This outgrowth step allows the bacteria time to generate the antibiotic resistance proteins encoded on the plasmid backbone so that they will be able to grow once plated on the antibiotic containing agar plate.

8. Plate some or all of the transformation onto a 10cm LB agar plate containing the appropriate antibiotic.

9. Incubate plates at 37°C overnight.

Figure 1: Example Protocol: Standard heat-shock transformation of chemically competent bacteria.

RESULTS

Here we show the set of structures and functions of each solution (Figure 2 and 3).

```
typedef struct chemical_
{
        char Type[SIZE];
        float Concentration;
        char Hazardousness[SIZE];
} chemical;

typedef struct lab_object_
{
        char Type[SIZE];
        char Material[SIZE];
        float Dimensions;
        chemical Chemical[SIZE];
        /*float Coordinate_x;
        float Coordinate_y;*/
}lab_object;

/*
typedef enum {
        incubator,
        centrifuge,
        PCR,
        microscope
} machine_type;*/

typedef struct lab_equip_
{
        int coor_x;
        int coor_y;
        float settings;
        float return_val;

} lab_equipment;
```

Figure 2: A set of structures from the first team

```
typedef struct material //Chemicals, cells, DNA, agar...
{
       float quantity;
       float mixture;
       float coox;
       float cooy;
       float cooz;

} material;

typedef struct equipment
{
       float power;
       float coox;
       float cooy;
       float cooz;

} equipment;

typedef struct containers
{
       float coox;
       float cooy;
       float cooz;

} containers;
```

Figure 3: A set of structures from the second team

Apart from structures, both teams (Mislav and Sofia, Sergi and Desika) also created a set of functions.

The first solution used the following functions and structures:

STRUCTURES
-Chemicals (defining the type, the concentration and the hazardousness)

-Lab objects (defining the type, the material of which the object is made, its dimensions, and location with the coordinates (x,y))

- Lab equipment (defining the location (x,y))

FUNCTIONS

- **Activate** (target, time, power, temperature) - function to define the desired lab equipment settings.

- **Set chemical** (type, concentration, hazardousness) - function to define the fields of the structure "chemical".

- Set coordinates (x, y) - function to define the position of each machine in the lab by a 2D coordinate system.

- **Move** (lab_object, source, target, depth) - moves the desired lab object from the lab equipment where it was to where is going to be and defines the depth in case you want to dig in in ice.

- **Pipette** (volume, source, target) - pipettes a certain amount of liquid from the source container to the target one.

- **Wait** (time) - blocks the robot for a certain time while an assay is being performed in a certain equipment.

- **Check** (lab_equipment, variable) - asks the lab equipment which has been the result of the assay performed and stores it in a variable so you can stablish conditions depending on the output.

A more general approach was used for the second solution. It used the following functions and structures:

STRUCTURES

-**Material (**defining the quantity of the chemical, the mixture as a new variable if we mix two different chemicals and the location with coordinates (x,y,z))

-Equipment (defines the power which is needed for the equipment to handle the samples and its location with coordinates (x,y,z))

- Containers (defines the location with coordinates (x,y,z))

FUNCTIONS

- Set coordinates (x, y, z) - function to define the position of each machine in the lab by a 3D coordinate system.

- **Move** (source, target, time) - moves the desired lab object from the starting position to the final one. Time is taken into account.

- **Heat** (source, target, temperature, time) - increases or decreases the temperature of a specific container in the desired equipment. Time is taken into account.

- **Shake** (source, target, power, time) - shakes a specific container in the desired equipment and at a certain power. Time is taken into account.

- **Pipette** (source, target, volume, time) - Pipettes a certain amount of liquid from the source container to the target one. Time is taken into account.

The languages were then applied to the same protocol (bacterial transformation). The protocol written in English was shown in the methods part (Figure 1), and its translations into formal languages are shown in ANNEX 1 and ANNEX 2.

DISCUSSION

The analysis of the problem has driven us to two different solutions, both of which have their own advantages and disadvantages. We analyze and compare both solutions here.

The main difference between the two solutions is in the description of the actions done by lab equipment. The first solution offers an explicit description of each of the steps the robot has to follow: setting the values and moving to that location to place the sample inside. The second solution instead uses a single function, for example centrifuging or heating which involves all the actions that solution one describes. Therefore, the second solution has a higher level of abstraction.

The advantage of the first approach for setting the locations is that there is no need to define one more coordinate for object which will not be moved along all three axis, but the second approach is more precise and we can easily avoid errors using it.

Another difference between them is the management of time. In the first solution, time is not measured globally during the whole protocol, using the special wait function to

represent the time between actions. In the second solution, a global variable that counts the time for every action is defined, , so the overall duration of the protocol is measured.

In order to represent the physical placement of the objects in the lab (whenever this is needed) two different strategies are used. For the first solution, we chose to use a 2D grid to set the locations, adding an argument in the function move to define the depth at which you want to place the object. In the second solution, this issue is solved by using a 3D grid.

The rest of the functions are similar in both languages and they take into account the same aspects. Although the teams both solutions were designed in different levels of abstraction, it can be set from the given results that the final languages don't differ greatly.

CONCLUSIONS

In the present work we have created a new language as our first step towards automating biology. We developed two libraries in C language for formal description of biological protocols. We tested these libraries by transcribing a protocol and validating the output to the natural language protocol description. Compared to recent work in the same field like Biocoder[3] or Autoprotocol[4] our libraries need further improvement to be able to describe other protocols. This should be done by analyzing more protocols and refining the functions. Our future work would include automated implementation and actual use of our language in the laboratories.

[4] Autoprotocols.org

ANNEX 1

Below, solution 1 code.

Main.c

```
 * Main.c

#include "protocol.h"
#include <stdio.h>

int main()
{


chemical competent_cells;

competent_cells = set_chemical("competent_cells", 0, "NO");

lab_equipment freezer;

lab_equipment ice;

lab_object tube_1;

tube_1 = set_labObject("tube_1", "plastic", 1.5, competent_cells);

move(tube_1, freezer, ice, 0);


chemical agar;

agar = set_chemical("agar", 2, "NO");

lab_object agar_plate;

agar_plate = set_labObject("agar_plate", "plastic", 10, agar);

lab_equipment fridge;

lab_equipment incubator;

activate(incubator, 30, 0.1, 37);
```

```
chemical agar;
agar = set_chemical("agar", 2, "NO");
lab_object agar_plate;
agar_plate = set_labObject("agar_plate", "plastic", 10, agar);
lab_equipment fridge;
lab_equipment incubator;
activate(incubator, 30, 0.1, 37);
move(agar_plate, fridge, incubator, 0);

chemical plasmid_DNA;
plasmid_DNA = set_chemical("Plasmid_DNA", 2, "NO");
chemical mix_1;
mix_1 = set_chemical("DNA+cells", 2, "NO");
lab_object tube_2;
tube_2 = set_labObject("Tube_2", "plastic", 1.5, plasmid_DNA);
lab_object tube_3;
tube_3 = set_labObject("Tube_3", "plastic", 1.5, mix_1);
pipette(3, tube_1, tube_3);
pipette(30, tube_2, tube_3);
```

```
lab_equipment mixer;

move(tube_3, ice, mixer, 0);

move(tube_3, mixer, ice, 0);


activate(incubator, 45, 0.1, 42);//time in seconds

move(tube_3, ice, incubator, 0);

move(tube_3, incubator, ice, 0);

wait(2*60);

chemical LB;

lab_object tube_4;

pipette(200, tube_4, tube_3);

activate(incubator, 45*60, 0.1, 37);//

move(tube_3, ice, incubator, 0);
wait(45*60);

pipette(50, tube_3, agar_plate);


}
```

protocol.c

```c
 * protocol.c

#include <stdio.h>
#include "protocol.h"

#include <string.h>

void activate(lab_equipment target, float time, float power, float temperature)
{


}

void move(lab_object labobject, lab_equipment source, lab_equipment target, float depth)
{
    printf("The lab object has been moved.\n");
};
void pipette(float volume, lab_object source, lab_object target)
{
    printf("The volume has been pipetted\n");
};

void wait(float time)
{
    printf("I'm waiting for %f minutes.\n", time);
};

float check(lab_equipment target, char variable)
{
    printf ("The result has been stored in variable %c. \n", variable);

    return 0;
};
chemical set_chemical(char type[SIZE], float concentration, char hazardousness[SIZE])
{
    chemical a;


    strcpy (a.Type, type);
    strcpy (a.Hazardousness,hazardousness);
    a.Concentration=concentration;

    return a;
};

lab_object set_labObject(char type [SIZE],
                         char material [SIZE],
                         float dimensions,
                         chemical Chemical)
{
    lab_object b;
    strcpy (b.Type, type);
    strcpy (b.Material, material);
    //strcpy (b.Chemical, chemical);
    b.Dimensions=dimensions;
    /*b.Coordinate_x=x;
    b.Coordinate_y=y;*/
    return b;
};
```

protocol.h

```
/*
 * protocol.h
 */

#ifndef PROTOCOL_H_
#define PROTOCOL_H_
#include <string.h>
#define SIZE 100


typedef struct chemical_
{
    char Type[SIZE];
    float Concentration;
    char Hazardousness[SIZE];
} chemical;

typedef struct lab_object_
{
    char Type[SIZE];
    char Material[SIZE];
    float Dimensions;
    chemical Chemical[SIZE];

}lab_object;



typedef struct lab_equip_
{
    int coor_x;
    int coor_y;
    float settings;
    float return_val;

} lab_equipment;

void activate(lab_equipment target, float time, float power, float temperature);
void move(lab_object labobject, lab_equipment source, lab_equipment target, float depth);
void pipette(float volume, lab_object source, lab_object target);
void wait(float time);
float check(lab_equipment target, char variable);
chemical set_chemical(char type [SIZE], float concentration, char hazardousness [SIZE]);
lab_object set_labObject(char type[SIZE], char material[SIZE], float dimension, chemical Chemical);




#endif /* PROTOCOL_H_ */
```

ANNEX 2

Below, solution 2 code.

main.c

```
 * main.c
 */
# include <stdio.h>
# include "LabProtocol.h"

int main()
{
    //Declaration of the variables

    equipment centrifuge, incubator, vortex, to;
    containers tube1, tube2, tube3, tube4, plate, use, use1;
    material DNA, agar, cc, chemicals, dnacc, LB;
    float pow, quantity, temp, time; //Quantity is given in microliters, temperature in °C and time in s


    //Definition of positions
    //Equipment

    set_coo(&centrifuge, &incubator, &vortex, &tube1, &tube2, &tube3, &tube4, &plate, &DNA, &agar, &cc, &LB);

    move (&cc, &incubator, 5);

    heat (&tube1, &incubator, 0, 1500);

    heat (&tube1, &incubator, 37, 120);

    pipette (&DNA, &tube2, &tube3, 5, 10);

    pipette (&cc, &tube1, &tube3, 50, 10);

    dnacc.mixture= 5 + 50;

    shake (&tube3, &vortex, 1, 7);

    heat (&tube3, &incubator, 0, 1500);

    heat (&tube3, &incubator, 42, 2700);

    heat (&tube3, &incubator, 0, 120);

    pipette (&LB, &tube4, &tube3, 500, 10);

    dnacc.mixture=55 + 500;

    move (&tube3, &incubator, 5);

    heat (&tube3, &incubator, 37, 2700);

    pipette (&dnacc.mixture, &tube3, &plate, 50, 10);

    move (&plate, &incubator, 5);

    heat (&tube3, &incubator, 37, 28800);

    printf("\n Goodbye.");

    return 0;
}
```

LabProtocol.h

```c
/*
 * LabProtocol.h
 */

#ifndef LABPROTOCOL_H_
#define LABPROTOCOL_H_


#endif /* LABPROTOCOL_H_ */

typedef struct material //Chemicals, cells, DNA, agar...
{
    float quantity;
    float mixture;
    float coox;
    float cooy;
    float cooz;

} material;

typedef struct equipment
{
    float power;
    float coox;
    float cooy;
    float cooz;

} equipment;

typedef struct containers
{
    float coox;
    float cooy;
    float cooz;

} containers;


//Functions
int set_coo (equipment *centrifuge,equipment *incubator, equipment *vortex, containers *tube1,containers *tube2,
int move (containers *use, equipment *to, float time);
int heat (containers *use, equipment *to, float temp, float time);
int shake (containers *use, equipment *to, float pow, float time);
int pipette (material *chemicals, containers *use, containers *use1, float quantity, float time);
```

LabProtocol.c

```c
 * LabProtocol.c

#include "LabProtocol.h"

int set_coo (equipment *centrifuge,equipment *incubator, equipment *vortex, containers *tube1,containers
{
    centrifuge->coox= 25;
    centrifuge->cooy= 25;
    centrifuge->cooz= 25;

    incubator->coox = 48;
    incubator->cooy = 30;
    incubator->cooz = 37;

    vortex->coox = 69;
    vortex->cooy = 69;
    vortex->cooz = 69;

    tube1->coox = 19;
    tube1->cooy = 30;
    tube1->cooz = 59;

    tube2->coox = 19;
    tube2->cooy = 20;
    tube2->cooz = 59;

    tube3->coox = 19;
    DNA->coox = 96;
    DNA->cooy = 96;
    DNA->cooz = 96;

    agar->coox = 80;
    agar->cooy = 86;
    agar->cooz = 86;

    cc->coox = 34;
    cc->cooy = 34;
    cc->cooz = 46;

    LB->coox = 96;
    LB->cooy = 96;
    LB->cooz = 96;

    return 1;
}


int move (containers *use, equipment *to, float time)
{
    use->coox=to->coox;
    use->cooy=to->cooy;
    use->cooz=to->cooz;


    printf("Moved!\n");
    printf("%f sec\n", time);
    return 1;
}
{
    use->coox=to->coox;
    use->cooy=to->cooy;
    use->cooz=to->cooz;
    to->power=pow;

    printf("Mixed!\n");
    printf("%f sec\n", time);
    return 1;
}

int pipette ( material *chemicals, containers *use, containers *use1, float quantity, float time)
{
    use->coox=use1->coox;
    use->cooy=use1->cooy;
    use->cooz=use1->cooz;

    chemicals->quantity=quantity;

    //The liquid is supposed to be sucked.

    printf("Pipetted!\n");
    printf("%f sec\n", time);
    return 1;
}
```

Output

```
<terminated> LabProtocol.exe [C/C++ Application] C:\Users\Učenik\Desktop\sergi_macia\S3\LabProtocol\Debug\LabProtocol.exe (03. 08. 2015. 18:11)
Moved!
5.000000 sec
Heated!
1500.000000 sec
Heated!
120.000000 sec
Pipetted!
10.000000 sec
Pipetted!
10.000000 sec
Mixed!
7.000000 sec
Heated!
1500.000000 sec
Heated!
2700.000000 sec
Heated!
120.000000 sec
Pipetted!
10.000000 sec
Moved!
5.000000 sec
Heated!
2700.000000 sec
Pipetted!
10.000000 sec
Moved!
5.000000 sec
Heated!
28800.000000 sec

 Goodbye.
```