

Concurrency in Arduino

Development of a programming language

Christoffer Trebbien Jønsson, Daniel Runge Petersen, Gustav
Svante Grønkjær Graversen, Jamie Lee Smith Hammer, Lars
Emanuel Hansen, Sebastian Aaholm

Computer Science, cs-22-dat-4-03, 2022-05

Semester Project



Copyright © Aalborg University 2021

Here you can write something about which tools and software you have used for typesetting the document, running simulations and creating figures. If you do not know what to write, either leave this page blank or have a look at the colophon in some of your books.



Electronics and IT
Aalborg University
<http://www.aau.dk>

AALBORG UNIVERSITY

STUDENT REPORT

Title:

Project Title

Abstract:

Here is the abstract

Theme:

Scientific Theme

Project Period:

Fall Semester 2010

Project Group:

XXX

Participant(s):

Author 1

Author 2

Author 3

Supervisor(s):

Supervisor 1

Supervisor 2

Copies: 1

Page Numbers: 31

Date of Completion:

April 4, 2022

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.



Elektronik og IT
Aalborg Universitet
<http://www.aau.dk>

AALBORG UNIVERSITET

STUDENTERRAPPORT

Titel:

Rapportens titel

Abstract:

Her er resuméet

Tema:

Semestertema

Projektperiode:

Efterårssemestret 2010

Projektgruppe:

XXX

Deltager(e):

Forfatter 1

Forfatter 2

Forfatter 3

Vejleder(e):

Vejleder 1

Vejleder 2

Oplagstal: 1

Sidetal: 31

Afleveringsdato:

4. april 2022


Rapportens indhold er frit tilgængeligt, men offentliggørelse (med kildeangivelse) må kun ske efter aftale med forfatterne.

Contents

Todo list	vii
Preface	xi
1 Introduction	1
1.1 Problem analysis content	1
2 Problem analysis	3
2.1 Arduino	4
2.2 Concurrency	5
2.3 What is the problem	6
2.4 Language evaluation criteria	6
2.5 Concurrency on an Arduino	9
2.6 Problem statement	11
3 Language design	15
3.1 Parser generator	15
3.2 Language grammar	19
3.3 Minted ANTLR test	19
3.4 Parser generator v.2 arc	25
3.5 Inspiration to our language	27
Bibliography	29
A Reading instructions	31
A.1 Review priority	31

Todo list

■ A little more detail on what we define as our target audience - hobbyists.	5
■ Write into definition of our target audience	5
■ I don't agree entirely on this definition, see page 60.	5
■ Maybe include references or examples?	6
■ Try to polish your argument	6
■ Early conclusion and problem statement. Perhaps too early? If not, it is a bit superficial	6
■ Giovanni: ??	6
■ Real-time systems has never been mentioned	6
■ This section is more appropriate to the chapter on Language design	6
■ Description or subsections?	7
■ Maybe briefly mention the criteria in full. Not detailed or copied, but mentions.	7
■ What does it mean to reduce data types. Perhaps explain it as generalizing data types with an example.	8
■ Abstraction should be explained properly as from the perspective of the type system.	8
■ Introduce the different programming languages	9
■ Explain what the advantages and disadvantages of protothreads is relative to the our problem	11
■ Introduce the different operative systems	11
■ Is writing about the OS Erika relevant?	11
■ unfinished section. We want only one problem statement, and preferably some references to previous choices	11
■ the way you put it sounds like: "we don't feel like writing code, then we use a parser generator". Maybe there are some deeper reasons why it's best to employ a parser generator in contrast to implementing your own ad hoc parser	15
■ Perhaps mention that the biggest advantage of auto-generating a scanner and parser is productivity?	15
■ Grammar in table is incomplete	16
■ this part has left recursion (see comment for details).	16
■ Incomplete argumentation. Comments welcome but not necessary	19

 Everything breaks when I add this in math mode below. So I am going to use comments.	20
---	----

Preface

Here is the preface. You should put your signatures at the end of the preface.

Aalborg University, April 4, 2022

Author 1

<username1@XX.aau.dk>

Author 2

<username2@XX.aau.dk>

Author 3

<username3@XX.aau.dk>

Chapter 1

Introduction

This is the introduction, write here if your text is relevant for it!

1.1 Problem analysis content

Arduino What it is, who uses it, and the challenges of the standard Arduino programming model in regards to concurrency/RTS.

Approaches Operating systems, scheduling, and differences between the options.

Scanner/Parser Generation of code. Or should this be after design? Otherwise, discussion of strengths and weaknesses.

Compilation Transpilation and compilation.

Problem statement Yes

HOWEVER, the content should be considered from the perspective of a programming language, rather than the perspective of computation.

Chapter 2

Problem analysis

Input your files here

Arduino har ikke concurrency. Folk er dårlige til at skrive Arduino kode, specielt concurrent-lignende-kode.

1. Arduino
2. Arduino bruger hobbyister
3. hobbyister problemer med at skrive god kode til Arduino
4. concurrency
5. concurrency på Arduino architectur single core
6. concurrency problemer for hobbyister at skrive det godt på Arduino
7. hvordan kan vi løse concurrency og dårlig programmering igennem et programmerings sprog.
8. Hvordan får man concurrency? -> OS/scheduling
9. hvilke muligheder for scheduling findes på Arduino? Hvilke OS.
10. (hvilken os eller library vil vi bruge til concurrency)
11. hvad er fordele og ulemper ved de forskellige concurrency mekanismer i forhold til sprog design?
12. hvordan måler vi kvaliteten af vores sprog, og hvilke parametre er relevante i vores situation.

har potentiale:

- (compiler transpiler)
- vil vi lave et survey/spørgeskema/test? Nok ikke.

- Hvilke værktøjer skal vi bruge?
- komplet concurrency vs subset af concurrency
- generelt begrænsning på hvad sproget skal bruges til.
- Should we use a parser generator, and which ones are available?

2.1 Arduino

Arduino is an open-source platform that enables users to easily and simply create small projects. Including projects such as an LED lighting up when a button is pushed, or building a thermostat to display the current ambient temperature. It is classified as a microcontroller board, which is a small computer that enables users to read some input and turn it into an output, and between this apply some logic. Arduino can be split into two categories, the physical and software:

The physical part of the Arduino is the board. There exist many types of boards that have different components and specs, but the most common board is the Arduino UNO. The Arduino UNO SET provides a USB cable to connect to a user's PC, a power input, some digital and analog pins, the microcontroller itself, the ATmega328, and various other components [1].

The software that Arduino makes use of is its IDE, which enables users to write code, also called sketches, which can be uploaded on the board and then be run. The code for Arduino is, in some ways, a simplified version of C++, which has integrated methods specifically made for Arduino.

The board is made to be inexpensive and simple to use, so it is possible for people with little to no experience working with electronics to pick up and start making projects. As in the past, it was necessary to have in-depth knowledge about microcontrollers and the like, to be able to create such projects. The Arduino IDE is also cross-platform, enabling users of Linux, Windows, and Mac OS to run the Arduino software [2].

2.1.1 Arduino Language

The Arduino language is a smaller version of C++. It holds most of the known semantics made from C++, but with some differences, which will be covered in this section [2]. A program in Arduino is called a sketch, Arduino sketch normally follows a basic structure, that consists of implementing the procedures "setup()" and "loop". "setup()" is called once at the beginning of the execution of the program; it is meant to initialize the data structure and pins [3]. After the setup procedure, the loop() function begins. The loop() function is a loop that runs while the Arduino is on, and implements the Arduino behavior [4].

2.1.2 Who uses it

Arduino is used by a wide range of people, such as students, hobbyists, programmers, and professionals [2]. Many different groups of people use the Arduino for different purposes. Some are focused on the learning aspects of the Arduino, others are more interested in easily designing concepts for a product. Students can use the board to learn the basics of electronics, hobbyists use it to build projects, and professionals use it to design product concepts [5].

For this report, the focus is on hobbyists who use the Arduino to create projects and want to learn the basics of electronics. The main concern regarding this group of users is that they spend a limited amount of time working with Arduino, as the time they spend with it will be in their leisure time. Moreover, hobbyists might have limited coding proficiency or be complete beginners. We characterize the users we want to focus on as hobbyists who want to do as much as they can with their limited time and limited coding proficiency.

A little more detail on what we define as our target audience - hobbyists.

2.1.3 What do hobbyists use the Arduino for

Hobbyists use Arduino for many different things, a Google search will show a vast amount of project ideas to people showing off their projects and maybe guiding people through to make their version. Such as this list of 10 uses for the Arduino [6]. The list includes projects that range from a simple thermostat to display the current temperature, to an automatic gardening device to water plants.

This shows just some of the projects that hobbyists could be interested in making, but for some projects, it might be useful for hobbyists to use concurrency.

Write into definition of our target audience

2.2 Concurrency

Concurrency is the ability for tasks to be scheduled in a way that makes things work at the correct time. For example, if one task is running it will be paused if a more pertinent task appears, The system will then start that task. The scheduler is usually what takes care of these choices and prioritization [7].

Concurrency is used for things like coordinating the execution of processes to maximize throughput. Concurrency is not to be confused with parallelism, which requires multiple cores because you have to do several things at the same time. Concurrency, on the other hand, does not need to be at the same time, just in the right order at the correct times, so it is possible to do with a single core. Since the Arduino has only a single core, this project deals with concurrency without parallelism.

It is possible to achieve concurrency on Arduino, but not without scheduling of some kind. Scheduling is to schedule different tasks to run on the same CPU, fooling users to think that they all run at the same time, while in fact it is running different tasks very fast and switching between them. This scheduling is not built into Arduino so the project will have to decide on how to achieve it. Part of the

I don't agree entirely on this definition, see page 60.

problem is choosing how to do this, while also keeping the end-user in mind. A hobbyist typically tries to achieve concurrency through the standard functions in Arduino, but many of these functions are not ideal for concurrency.

Maybe include references or examples?

Several tutorials on how to implement different techniques to achieve some sort of concurrency in Arduino exist. These techniques include `Milis()` and `Interrupt()`. `Milis()` uses the `milis` function to tell the time, while not pausing to achieve the ability to wait and not do unnecessary tasks at all times. The `interrupt()` function, on the other hand, cancels the process if something happens, and this can be used to change between different tasks.

Try to polish your argument

2.2.1 Concurrency for hobbyist

Concurrency is not achieved effortlessly with Arduino, as an additional layer of programming complexity is necessary. This is, of course, a problem for the hobbyist who is not the most experienced programmer, since it makes the problem of getting the Arduino to do what they want even harder. It also increases the time the hobbyist spends implementing a solution that seems relatively intuitive. To help the hobbyist achieve correct and functional concurrency easily and simply, this is what the project will try to do..

Early conclusion and problem statement. Perhaps too early? If not, it is a bit superficial

2.3 What is the problem

Giovanni: ??

Real-time systems has never been mentioned

The problem we will try to solve in this project is an extension for hobbyists working on Arduino, concurrency and real-time systems. The problem is based on hobbyists having trouble implementing concurrency simply and effectively without getting into the nitty-gritty details, that a hobbyist does not want to deal with. It is also based on many tutorials, which have implementations containing bad practices.

This has led us to the problem we want to try and solve, hobbyists have a hard time using concurrency easily on Arduino. We will try to solve this by creating a programming language for hobbyists to easily and correctly implement concurrency on their Arduino devices. Therefore our problem statement is:

How can you create a programming language that would allow a hobby-level user of Arduino to use concurrency easily and correctly concerning the best Arduino concurrency practices?

2.4 Language evaluation criteria

This section is more appropriate to the chapter on Language design

There is no common consensus on objectively evaluating a language. The measurement of compilation speed, execution speed, and file size speak to the efficiency of the *implementation* of the language, not the *design* of the language. Popularity could be measured, but would vary greatly with time, and contain several skews from bias, as well as popularity not necessarily meaning it is better, which would have to be considered as well.

As such, a set of prioritized language evaluation criteria has been determined based on the criteria presented in [8]. An objective evaluation of the language based on these criteria is not practically possible [8], however, the criteria may work well when making decisions during language design, implementation, and evaluation. The most relevant concerns in regard to the choice of relevant criteria are presented in this section.

The primary concern is with the users of the programming language; writers of Arduino programs, in this case, the Arduino hobbyist, as described earlier. The programming of an Arduino project may be secondary to hobbyists, who prioritize the hardware aspects of their project.

The secondary concern is the concurrency issue that follows from the primary concern - making an Arduino behave concurrently is not a trivial programming task. There are several variations on the theme of simulating concurrency, but each with different issues, and none solve the issue in a general way [9].

The tertiary and last concern follows from the secondary issue - determining how to use concurrency to solve the problem in an Arduino. Concurrency problems can be subtle and complex, so understanding that you are even dealing with a concurrency issue may not be immediately apparent, and thus a simple solution may be hard to spot.

2.4.1 Priority of criteria characteristics

Concepts of programming languages lists four criteria: readability, writability, reliability, and cost. These criteria are each affected by several characteristics, with varying influence and importance [8].

Description or subsections?

Readability

Writability

Reliability

Cost

Readability

Writability

Reliability

Cost

Maybe briefly mention the criteria in full. Not detailed or copied, but mentions.

With these considerations in mind, we have prioritized and selected the characteristics for each criterion as we expect them to matter in this context.

The primary concern deals primarily with a subset of considerations related to the cost criterion. Specifically, the cost associated with learning and understanding the programming language is important. This suggests that general simplicity, in

the form of few but expressive language constructs as well as clear, consistent combination and application of the constructs is important.

Expressivity and syntax design are also important characteristics, as seen from the secondary and tertiary concerns. The language should express new constructs that are not available in the Arduino language in a concise manner. An aim of the syntax could be to describe concurrent programs simply.

The number of data types is probably a somewhat important characteristic. As long as the expressivity of the language is not greatly affected, the number of data types is reduced compared to the Arduino language is likely to have a positive effect on overall simplicity and writability. On the other hand, good support for abstraction, that is, user-defined types, may enable advanced users to have the freedom of many primitive types, without significantly impacting the overall simplicity of the language.

It is important to consider the language paradigm as well, as this has a large effect on the type checking mechanisms commonly used, not to mention the syntax design. It is a determinant factor in the description of mutability versus immutability, state versus statelessness, aliasing, and pointer management.

By default, Arduino does not use exceptions and exception handling as per the C++ language. The common solution for Arduino code writers is to write code that handles the possible exceptions that may occur without the language construct. For the sake of footprint, this will also be the preferred solution for this project.

Characteristics	Very important	Important	Somewhat important	Not important
Simplicity		X		
Orthogonality		X		
Data types			X	
Syntax design		X		
Support for abstraction			X	
Expressivity		X		
Type checking		X		
Exception handling				X
Restricted aliasing	X			

Table 2.1: Summary of priorities. Characteristics not mentioned are of low or no priority.

It is worth noting that the Arduino programming language has already addressed several of the above concerns when compared to C++ for the platform. Examples of this can be seen in the introduction of new constants and a reduction in some language capabilities, such as exceptions and try-catch blocks from C++. The Arduino IDE is another point to consider in favor of cost concerns for the Arduino platform.

What does it mean to reduce data types. Perhaps explain it as generalizing data types with an example.

Abstraction should be explained properly as from the perspective of the type system.

2.5 Concurrency on an Arduino

In this section, some of the more relevant ways of achieving concurrency on an Arduino will be explored. Emphasis will be put on the immediate advantages and disadvantages related to each of the different ways of concurrency, and how well they might fit as a solution to the problem. The first discussion will be about the prospect of using programming libraries to achieve concurrency on an Arduino. The second discussion is going to be about an alternative solution, namely implementing an operating system (OS) to handle the concurrency.

2.5.1 Achieving Concurrency with Programming Libraries

Protothreads, Eventually, ARTe (and maybe TaskManagerIO)

Introduce the different programming languages

Protothreads

Protothreads is a library for the C language, which has been packaged to a library for Arduino. The point of this library is to give programmers a simpler way to write programs for an event-driven system in memory-constrained environments such as an Arduino UNO.

Protothreads provides a blocking context on top of an event-driven system, without the overhead of per-thread stacks. The purpose of protothreads is to implement sequential flow of control without complex state machines or full multi-threading. [10]

Protothreads are lightweight, stackless threads that provide a mechanism for concurrent programming, which is designed for memory-constrained systems, such as a smaller embedded system or wireless sensor nodes. Protothreads can be used with or without any underlying operating system to provide blocking for event handlers' [11]. It uses a cooperative concurrency form, which means it is up to the user to synchronize the program to run concurrently. This means that the program is event-driven and before the program can continue to another task, it needs to "complete" the current task before moving on to the next one. When working with Protothreading, it is nice to know what the overhead is on two bytes. This means that there is no hidden memory cost during the execution of the program.

When using local variables inside a Protothread, the local variables are not preserved and therefore can not be allocated onto the memory, therefore the programmer would have to use global variables instead, if they want to store something inside a variable.

Lastly, it is important that the code inside a Protothread needs to be "fast", meaning that the programmer can not use any blocking function such as the `delay()` function because this would block the other functions to run. [11]

In relation to what protothreads are, there has been taken an example of a simple and small program to show what Protothreading can be used to which can be seen at Figure ?? [12].

```

1  #include "protothreads.h"
2
3  const int buttonPin = 12;    // the number of the pushbutton pin
4  const int ledPin = 8;       // the number of the LED pin
5  int buttonState = 0;        // variable for reading the pushbutton status
6
7
8  pt ptBlink;
9  int blinkThread(struct pt* pt) {
10     PT_BEGIN(pt);
11
12     for(;;) {
13         digitalWrite(ledPin, HIGH);    // turn the LED on (HIGH is the voltage level)
14         PT_SLEEP(pt, 500);
15         digitalWrite(ledPin, LOW);     // turn the LED off by making the voltage LOW
16         PT_SLEEP(pt, 500);
17     }
18     PT_END(pt);
19 }
20
21 pt ptButton;
22 int buttonThread(struct pt* pt) {
23     PT_BEGIN(pt);
24
25     for(;;) {
26         int sensorValue = digitalRead(buttonPin);
27         Serial.println(sensorValue);
28         PT_SLEEP(pt, 1);
29         PT_YIELD(pt);
30     }
31     PT_END(pt);
32 }
33
34 void setup() {
35     PT_INIT(&ptBlink);
36     PT_INIT(&ptButton);
37     Serial.begin(9600);
38     pinMode(ledPin, OUTPUT);
39     pinMode(buttonPin, INPUT);
40 }
41
42 void loop() {
43     PT_SCHEDULE(blinkThread(&ptBlink));
44     PT_SCHEDULE(buttonThread(&ptButton));
45 }

```

Listing 2.1: A small program on how a Protothreads can be implemented

Eventually C++ Library

Eventually is an Arduino Event-based Programming Library. Where the goal is to make a more event-oriented environment for the Arduino programming language.

To give a better understanding of how the Eventually C++ library is working, in this section there has been taken an example from a GitHub page [13] where there can be seen, a code example on how to use the Eventually library. Since it is a C++ library it would work on any Arduino board, which includes the one the group has acquired.

Explain what the advantages and disadvantages of protothreads is relative to the our problem

ARTE

Arduino Real-Time extension (ARTE) is being developed by Real-Time System Laboratory. We tried to create an example similar to the former examples, however, it was simply not possible. While the scope of this project only focuses on the most popular Arduino UNO, and they still remain to support other devices than the Arduino DUO boards, no more time will be allocated to this option. However, once they release official support for the Arduino UNO, it is a viable option to look into. As for now, it is simply not ready for us to work with.

2.5.2 Achieving Concurrency with an Operating System

(Erika OS), FreeRTOS, Simba OS, TaskManagerIO

Introduce the different operative systems

Is writing about the OS Erika relevant?

FreeRTOS

Free Real-Time Operating System (abbreviated to FreeRTOS) is an operating system specifically designed for microcontrollers and microcomputers, such as the Arduino. It has been developed in partnership with the leading chip companies in the world, over more than 18 years, and with a special emphasis on reliability, accessibility and ease of use [14]. This leads itself well to our project, while we are targeting hobbyists. FreeRTOS utilises preemptive scheduling [15], which means that it implements a scheduler to be responsible for deciding which tasks to do in which order.

2.5.3 Summary

2.6 Problem statement

The problem statement goes here. Temporary problem statement (Probably a full language, rather than an extension):

To create an extension on the Arduino programming language to support concurrency in real-time systems. The language should aid hobbyists in writing idiomatic Arduino programs.

unfinished section. We want only one problem statement, and preferably some references to previous choices

```

1  #include <Eventually.h>
2
3  #define LIGHT_PIN 8
4  #define BUTTON_PIN 12
5  bool pinState = true;
6
7
8  EvtManager mgr;
9  bool blinker(){
10     mgr.resetContext();
11     mgr.addListener(new EvtTimeListener(1000, true, (EvtAction)blink_pin)); //Event
12     ↪ for LED til blink
13     mgr.addListener(new EvtPinListener(BUTTON_PIN, (EvtAction)digital_read)); //Event
14     ↪ for button input
15 }
16
17 void blink_pin(){
18     if (pinState == true){
19         digitalWrite(LIGHT_PIN, HIGH); // turn the LED on (HIGH is the voltage level)
20     }
21     else{
22         digitalWrite(LIGHT_PIN, LOW); // turn the LED off by making the voltage LOW
23     }
24     pinState = !pinState;
25 }
26
27 void digital_read(){
28     int sensorVal = digitalRead(BUTTON_PIN);
29     Serial.println(sensorVal);
30     delay(1);
31 }
32
33 void setup() {
34     Serial.begin(9600);
35     pinMode(LIGHT_PIN, OUTPUT);
36     pinMode(BUTTON_PIN, INPUT);
37
38     blinker();
39 }
40
41 USE_EVENTUALLY_LOOP(mgr)

```

Listing 2.2: A small program on how a Eventually can be implemented

2.6.1 Second attempt

To create a programming language for Arduino, which leverages some concurrency mechanism, to make it simpler or easier for hobbyists to write idiomatic Arduino programs with concurrency.

To create a programming language for Arduino, which leverages some concurrency mechanism, to make it simpler or easier for hobbyists to write idiomatic Arduino programs with concurrency.

```

1  #include <Arduino_FreeRTOS.h>
2
3  void TaskBlink( void *pvParameters );
4  void TaskAnalogRead( void *pvParameters );
5
6  void setup() {
7      Serial.begin(9600);
8
9      while (!Serial) {
10         ;
11     }
12
13     xTaskCreate(
14         TaskBlink
15         , "Blink" // A name just for humans
16         , 128 // This stack size can be checked & adjusted by reading the Stack
17           ↪ Highwater
18         , NULL
19         , 2 // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0
20           ↪ being the lowest.
21         , NULL );
22
23     xTaskCreate(
24         TaskAnalogRead
25         , "AnalogRead"
26         , 128 // Stack size
27         , NULL
28         , 1 // Priority
29         , NULL );
30 }
31
32 void loop()
33 {
34 }
35
36 void TaskBlink(void *pvParameters) // This is a task.
37 {
38     (void) pvParameters;
39
40     pinMode(LED_BUILTIN, OUTPUT);
41
42     for (;;) // A Task shall never return or exit.
43     {
44         digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage
45           ↪ level)
46         vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
47         digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
48         vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
49     }
50 }
51
52 void TaskAnalogRead(void *pvParameters) // This is a task.
53 {
54     (void) pvParameters;
55     for (;;)
56     {
57         int sensorValue = digitalRead(12); // read the input on analog pin 0:
58         Serial.println(sensorValue); // print out the value you read:
59         vTaskDelay(1); // one tick delay (15ms) in between reads for stability
60     }
61 }

```

Listing 2.3: A small example of a possible implementation of Free RTOS.

Chapter 3

Language design

3.1 Parser generator

Instead of writing code by hand, several tools automatically generate code for scanning and parsing based on a grammar. Therefore, a discussion of *JavaCC*, *ANTLR4*, and *CUP* precedes the formal language description. The tools mentioned are only a few of those available and have been chosen because the group has experience with them through coursework. The main reason a parser generator has been chosen is because of the gains in productivity that it makes possible. It also makes it possible to create a functioning compiler yet still have language design as a larger focus.

Each tool is evaluated using a reduced fragment of the Bims grammar from table 3.1 [16]. For each tool, the grammar was adapted, and the tool was run with the rewritten grammar as input. The process of rewriting, inputting, and running the tool was compared, and a tool was selected.

the way you put it sounds like: “we don’t feel like writing code, then we use a parser generator”. Maybe there are some deeper reasons why it’s best to employ a parser generator in contrast to implementing your own ad hoc parser

Perhaps mention that the biggest advantage of auto-generating a scanner and parser is productivity?

$n \in \mathbf{Num}$ – Numerals
 $x \in \mathbf{Var}$ – Variables
 $a \in \mathbf{Aexp}$ – Arithmetic expressions
 $b \in \mathbf{Bexp}$ – Boolean expressions
 $S \in \mathbf{Stm}$ – Statements

$S \rightarrow x := a \mid \text{skip} \mid S_1; S_2$
 $b \rightarrow a_1 = a_2 \mid a_1 < a_2 \mid \neg b_1 \mid b_1 \wedge b_2 \mid (b_1)$
 $a \rightarrow n \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2 \mid (a_1)$

Table 3.1: Sample Bims syntax [16]

3.1.1 JavaCC

JavaCC stands for Java Compiler Compiler and is an open-source parser generator and lexical analyzer developed by Oracle, written in the Java language. It is one of the most popular parser generators for Java[17], and it is very well documented.

JavaCC takes as input a Context-Free Grammar (CFG) in Extended Backus-Naur Form (EBNF) to generate a scanner and a parser based on the input. The parser generated is a Left-to-right, Leftmost derivation with 1 token lookahead (LL(1)) parser by default. However, the parser can be extended to k lookahead for parts of the grammar if necessary [17].

Setting up JavaCC was easy as it is well documented, and requires Java to be installed. Having set this up, we were able to begin writing our grammar. After setting up, it became clear that writing grammars in JavaCC were simple but less intuitive. In JavaCC, the grammar has to be written as code, whereas the syntax of other tools resembles the way it is seen in books, as shown in Table 3.1. An example of this can be seen in listing 3.1

```

1  PARSER_BEGIN(Example)
2
3  public class Example {
4      public static void main(String args[]) throws ParseException {
5          Example parser = new Example(System.in);
6          parser.Input();
7      }
8  }
9
10 PARSER_END(Example)
11
12 void Input() : {}
13 {
14     MatchedBraces() ("\n" | "\r")* <EOF>
15 }
16
17 void MatchedBraces() : {}
18 {
19     "{" [ MatchedBraces() ] "}"
20 }
```

Listing 3.1: An example of the JavaCC syntax

Rewriting Bims

Because the generated parser is LL(1), the input grammar must be non-left-recursive. Bims, however, is left-recursive and has to be rewritten. This can be seen in Table 3.2.

Grammar in table is incomplete

this part has left recursion (see comment for details).

$$\begin{aligned}
S &\rightarrow x := aS' \mid \text{skip } S' \mid \text{while } S' \\
S' &\rightarrow ;SS' \mid \epsilon \\
b &\rightarrow a_1 = a_2 \mid a_1 < a_2 \mid \neg b_1 \mid b_1 \wedge b_2 \mid (b_1) \\
a &\rightarrow n \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2 \mid (a_1)
\end{aligned}$$

Table 3.2: Rewrite of Bims without left recursion

3.1.2 ANTLR4

ANTLR (short for ANother Tool for Language Recognition) is a powerful parser generator that can be used to process, execute, or even translate structured text or binary files. It is widely used around the world by both Twitter and Oracle for parsing queries. [18]

One of the major advantages of ANTLR is their high level of support in almost any popular IDE, making it easy to work with regardless of the preferred development environment of the programmer. The ability to generate the lexer, parser and concrete syntax tree from a single grammar file is also very appealing. The grammar is very similar to that of EBNF and utilises a custom parsing technology called Adaptive LL(*) or ALL(*). This differs from the LL(*) used in ANTLR3, by analysing the grammar dynamically at runtime rather than statically, before the parser executes [19].

ANTLR is very easy to work with. One simply installs the plugin in their IDE of choice and read through the simple-to-understand documentation found on the official GitHub page of ANTLR [20]. Understanding the grammar rules and how to set it up in its respective grammar4 (.g4 file extension) is very manageable thanks to this. After the grammar file has been written it is possible to run the ANTLR tool on it, which then automatically generate a number of files for us, such as a parser, a lexer and a set of tokens. Then the generated code can be compiled against the ANTLR runtime. This also provides the developer with both a visual representation of the created parse tree or a tree in a LISP-like text form, along with a list of tokens found - all of these options are accessible through optional flags during compilation. The fourth major version of ANTLR introduces a lot of new capabilities compared to that of ANTLR3, which helps to reduce the learning curve by making the development of grammars and language applications easier - in part thanks to the powerful extensions and tools included in their plugins.

3.1.3 CUP

"CUP stands for Construction of Useful Parsers and is a Look-Ahead Left-to-right, Rightmost derivation (LALR) parser generator for Java" [21]. CUP implements a standard LALR(1) parser generation. Documentation for CUP recommends using the scanner generator JFlex, as a Lexical Analyzer Generator. Figure 3.1 is an

example of how CUP was used.

```

1 /* Simple +/-/* expression language; parser evaluates constant expressions on the fly*/
2 import java_cup.runtime.*;
3
4 parser code {
5     parser p = new parser(new Scanner(new FileReader(fileName)));
6     Object result = p.parse().value;
7 :}
8
9 //parser code {
10 //     // Connect this parser to a scanner!
11 //     scanner s;
12 //     Parser(scanner s){ this.s=s; }
13 //:}
14
15 /* define how to connect to the scanner! */
16 init with { : p.init(); :};
17 scan with { : return p.next_token(); :};
18
19 terminal PLUS;
20 terminal Integer NUMBER;
21
22 non terminal exp;
23
24
25 exp ::= exp PLUS NUMBER | NUMBER;
26

```

Figure 3.1: An example of CUP

In this we created a simple grammar that could add two integers together; this was done to try using CUP and to understand how everything worked.

JFlex

JFlex is an abbreviation for Java Flex. Flex is an abbreviation for Fast Lexical Analyzer Generator. As the name suggests, it is a tool for generating fast lexers. It makes adjustments to the size and speed of the generated lexer possible.

In figure 3.2 there is an example of how JFlex looked, in conjunction with our previously mentioned CUP file.

```

23 LineTerminator = \r|\n|\r\n
24 WhiteSpace     = {LineTerminator} | [ \t\f]
25
26 DecIntegerLiteral = 0 | [1-9][0-9]*
27
28 %%
29
30 /* literals */
31 {DecIntegerLiteral}          { return symbol(sym.NUMBER); }
32 \"                          { string.setLength(0); yybegin(STRING); }
33
34 /* whitespace */
35 {WhiteSpace}                 { /* ignore */ }
36
37 /* operators */
38 \+                          { return symbol(sym.PLUS); }

```

Figure 3.2: An example of JFlex

3.1.4 Experiences with CUP

The general experience with CUP was good, except for connecting the parser and the lexical analyzer. At first, CUP was easy and workable, and setting up a simple grammar and creating the parser and symbol tree for it went well. But when there were specific issues it was difficult to find solutions, as CUP is not a very popular parser generator, which meant there was a minimal amount of information to be found.

3.1.5 Result

CUP will not be used in the project, as it proved difficult, combined with a lack of documentation, to connect the parser generator with the lexical analyzer generator, whereas other tools, such as JavaCC and ANTLR, automatically creates the lexical analyzer.

Incomplete argumentation. Comments welcome but not necessary

Based on our experiences with the different compiler compilers we have chosen to work with ANTLR. We chose ANTLR primarily because of how powerful it is and the great extensions it provides. These factors in particular could help us to efficiently develop our language and compiler. It does commit us to a LL parser which is not as expressive as an LR parser but we do not believe this will be a problem for our language since it will be relatively compact.

3.2 Language grammar

Which arithmetic and other operations should be available in the language?

3.3 Minted ANTLR test

```
1 grammar Expr;
2 prog:      (expr NEWLINE)* ;
3 expr:      expr ('*' | '/') expr
4           | expr ('+' | '-') expr
5           | INT
6           | '(' expr ')'
7           ;
8 NEWLINE : [\r\n]+ ;
9 INT     : [0-9]+ ;
```

$Start \rightarrow \text{Setup Pipeline}$
 $PLS \rightarrow PL\ PLS \mid \epsilon$
 $PL \rightarrow Def$
 $stmts \rightarrow stmt\ ;(\text{newline})\ stmts \mid \epsilon$
 $stmt \rightarrow expr \mid ass \mid dcl$
 $dcl \rightarrow \text{type funcID (Params)\{stmts\}}$
 $ass \rightarrow \text{type varID} = expr$
 $params \rightarrow param, params \mid param$
 $param \rightarrow \text{type varID}$
 $expr \rightarrow \text{for params in varID}$

Everything breaks when I add this in math mode below. So I am going to use comments.


```

    Start → Setup
    Setup → Setup { SetupDeclaration }
    SetupDeclaration → Declaration; SetupDeclaration
                     | FunctionCall; SetupDeclaration
                     | ε
    Statements → ExpressionStatement; Statements
               | ReturnStatement; Statements
               | IfStatement; Statements
               | ForLoop; Statements
               | WhileLoop; Statements
               | SwitchStatements; Statements
               | ε
    ExpressionStatement → Expression
    ReturnStatement → 'return' '(' Expression ')'
    IfStatement → 'if' '(' LogicalExpression ')' '{' Statements '}' ElseStatement
    ElseStatement → 'else' '(' LogicalExpression ')' '{' Statements '}'
    ForLoop → 'for' '(' VariableDeclaration 'in' Identifier ')' '{' Statements '}'
    WhileLoop → 'while' '(' LogicalExpression ')' '{' Statements '}'
    SwitchStatements → 'when' '(' Identifier ')' '{' SwitchCase+ ElseStatement '}'
    SwitchCase → Expression '=>' '{' Statements '}'
    ExpressionList → Expression (,Expression)*
    Expression → ArithmeticExpression
               | RelationalExpression
               | LogicalExpression
               | ConditionalExpression
    ArithmeticExpression → AtomArithmeticExpression(* | /)ArithmeticExpression
                       | AtomArithmeticExpression(+ | -)ArithmeticExpression
    AtomArithmeticExpression → NUMBER
                           | '(' ArithmeticExpression ')'
                           | IDENTIFIER
    RelationExpression → ArithmeticExpression(< | > | ==)ArithmeticExpression
    LogicalExpression → RelationalExpression(or | and)RelationalExpression
                    | not RelationalExpression
    Declaration → FunctionDeclaration
                | VariableDeclaration
    FunctionDeclaration → TypeOf IDENTIFIER '(' VariableDeclarationList ')' '{' Statements '}'
    TypeOf → TYPE (TYPEOPERATOR)*
    VariableDeclarationList → VariableDeclaration (,VariableDeclaration)*
    VariableDeclaration → TypeOf IDENTIFIER '=' Expression
    FunctionCall → IDENTIFIER '(' ExpressionList ')'

```



```

NUMBER → ('-')?('0'..'9')+('.'('0'..'9')+)?;
DIGIT → [0..9]*
ALPHA → 'a'...'z' | 'A'...'Z' | '_'
IDENTIFIER → ALPHA(DIGIT | ALPHA)*
TYPEOPERATOR → ('[ ]');
TYPE → (num | text | bool);
WS → [ \t \n \r]+ -> skip;
COMMENTS → '//' .*? '\n' -> skip;
LINECOMMENT → '/' .*? '*' -> skip;
KEYWORDS → return, while, it, for, in, when,
           void?, Arduino{}, setup, true, false, else, define
OPERATORS → '+' | '-' | '*' | '/'
LOGICALOPERATION → 'and' | 'or' | 'not' | (xor)
RELATIONOPERATORS → '>' | '<' | '==' | '!=' | '<=' | '>='
BOOL → 'true' | 'false'

```


3.4 Parser generator v.2 arc

Parser grammar

```

    start → setup declarations
    declarations → declaration declarations
    setup → setup '{' setupDeclaration '{'
           | ε
    setupDeclaration → declaration ';' setupDeclaration
           | functionCall ';' setupDeclaration
           | ε
    statements | returnStatement statements
           | ifStatement statements
           | forLoop statements
           | whileLoop statements
           | whenStatement statements
           | variableDeclaration statements
           | assignmentStatement
           | ε
    assignmentStatement → IDENTIFIER ('[' NUMBER ''])? '=' ('[' (expression (',' expression)*) ? ']' | expression)
    returnStatement → 'return' expression
    ifStatement → 'if' '(' logicalExpression ')' ' ' statements ' ' else
    else → 'else' '{' statements '}'
    forLoop → 'for' '(' parameterDeclaration 'in' IDENTIFIER ')' '{' statements '}'
    whileLoop → 'while' '(' expression ')' '{' statements '}'
    whenStatement → 'when' '(' IDENTIFIER ')' '{' whenCase+ else '}'
    whenCase → expression '=>' '{' statements '}'
    expression → (NUMBER | IDENTIFIER | BOOL)
           | (functionCall | IDENTIFIER '[' NUMBER ']')
           | '(' expression ')'
           | 'not' expression
           | expression (MULTI | DIVI) expression
           | expression (PLUS | MINUS) expression
           | expression RELATIONEQOPERATORS expression
           | expression RELATIONOPERATORS expression
           | expression 'and' expression
           | expression 'or' expression;
    functionCall → IDENTIFIER '(' expression (',' expression)* ')'
    declaration → functionDeclaration
           | variableDeclaration
    functionDeclaration → typeOf IDENTIFIER '(' (parameterDeclaration (',' parameterDeclaration)*) ? ')' '{'
    variableDeclaration → typeOf IDENTIFIER '=' ('[' (expression (',' expression)*) ? ']' | expression) ';'
    typeOf → TYPE_TYPEOPERATORparameterDeclaration

```

Lexical grammar

$$\begin{aligned}
WS &\rightarrow [\backslash t \backslash n \backslash r]^+ \rightarrow \text{skip} \\
NUMBER &\rightarrow '-'? \text{DIGIT}^+ ('.' \text{DIGIT}^+)? \\
\text{fragment} \text{DIGIT} &\rightarrow [0-9] \\
BOOL &\rightarrow \text{'true'} \mid \text{'false'} \\
\text{TYPE_TYPE} \text{OPERATOR} &\rightarrow \text{TYPE} (\text{TYPE} \text{OPERATOR})^* \\
\text{fragment} \text{TYPE} \text{OPERATOR} &\rightarrow \text{'['} \text{' } \\
\text{fragment} \text{TYPE} &\rightarrow \text{'num'} \\
&\mid \text{'text'} \\
&\mid \text{'bool'} \\
&\mid \text{'char'} \\
COMMENTS &\rightarrow \text{'//'} .*? '\n' \rightarrow \text{skip} \\
\text{LINE} \text{COMMENTS} &\rightarrow \text{'/*'} .*? '*/' \rightarrow \text{skip} \\
RETURN &\rightarrow \text{'return'} \\
WHILE &\rightarrow \text{'while'} \\
IT &\rightarrow \text{'it'} \\
FOR &\rightarrow \text{'for'} \\
IN &\rightarrow \text{'in'} \\
WHEN &\rightarrow \text{'when'} \\
VOID &\rightarrow \text{'void'} \\
ARDUINO &\rightarrow \text{'arduino'} \\
SETUP &\rightarrow \text{'setup'} \\
ELSE &\rightarrow \text{'else'} \\
DEFINE &\rightarrow \text{'define'} \\
MULTI &\rightarrow \text{'*'} \\
DIVI &\rightarrow \text{'/' } \\
PLUS &\rightarrow \text{'+' } \\
MINUS &\rightarrow \text{'-' } \\
\text{RELATIONEQ} \text{OPERATIONS} &\rightarrow \text{'==' } \\
&\mid \text{'!=' } \\
\text{RELATION} \text{OPERATORS} &\rightarrow \text{'<' } \\
&\mid \text{'>' } \\
&\mid \text{'<=' } \\
&\mid \text{'>=' } \\
AND &\rightarrow \text{'and'} \\
OR &\rightarrow \text{'or'} \\
NOT &\rightarrow \text{'not'} \\
XOR &\rightarrow \text{'xor'} \\
IDENTIFIER &\rightarrow \text{ALPHA} (\text{DIGIT} \mid \text{ALPHA})^* \\
ASSIGNMENT &\rightarrow \text{'=' } \\
SEPERATOR &\rightarrow \text{' ,' } \\
\text{fragment} \text{ALPHA} &\rightarrow [\text{a-z}] \mid [\text{A-Z}] \mid \text{'_'}
\end{aligned}$$

3.5 Inspiration to our language

When designing a programming language it is a good idea to look at other programming languages for inspiration. We started by discussing the languages that we all have worked with in former semesters, which is C, JavaScript and C#. It was also important to check C++ as it was the language, our language should compile to. The discussion in the group was which things the programming languages did well, and what our programming language should include. When discussing these languages there became a common understanding, the programming languages discussed were not intuitive enough for a hobbyist and therefore we needed more research on how other languages did this. Researching then began on looking on more modern and intuitive languages so our programming language would be easier to learn and write in. The languages Dart, Kotlin, Rust, Python and Zyg were all looked at to find the inspiration for our language. One thing we all looked after in the languages was if it had a readable syntax, it did not have to be for whole language, it was only specific ways of writing syntax. Therefore you will see inspiration from many programming languages, when describing ours.

3.5.1 Types

The types in the language are num, text and bool. A value of the type num is a correct input as long it is a real number. Bool is like the bool in other languages, as it can only be true or false. The type text is an array of characters made by using quotation marks (") from the start to end of the input. An num array can be made by using square brackets (

) in front of the type num, how ever an num array can only accept real numbers.

3.5.2 Control Structures

There are ... types of control structures if-,else-,when-statement, while- and for loop.

A while loop is made by an expression, which evaluates to true or false, as long as the expression is true the while loop executes the statements within the scope, made by curly brackets ({}).

Bibliography

- [1] *What Is an Arduino? - Learn.Sparkfun.Com.* URL: <https://learn.sparkfun.com/tutorials/what-is-an-arduino/all#introduction> (visited on 03/04/2022).
- [2] *What Is Arduino?* URL: <https://www.arduino.cc/en/Guide/Introduction> (visited on 03/04/2022).
- [3] *Arduino. setup.* <https://www.arduino.cc/reference/en/language/structure/sketch/setup/>. 2022.
- [4] *Arduino. loop.* <https://www.arduino.cc/reference/en/language/structure/sketch/loop/>. 2022.
- [5] *Is Arduino Used in Real Life Products?* The Robotics Back-End. Dec. 9, 2018. URL: <https://roboticsbackend.com/is-arduino-used-in-real-life-products/> (visited on 03/07/2022).
- [6] *Top 10 Arduino Uses of 2021.* All3DP. Jan. 13, 2021. URL: <https://all3dp.com/2/best-arduino-uses/> (visited on 03/07/2022).
- [7] Randal. Bryant and David O'Hallaron. *Computer Systems*. Pearson, 2016.
- [8] Robert W. Sebesta. *Concepts of programming languages*. Pearson, 2016.
- [9] Francesco Restuccia, Marco Pagani, Agostino Mascitti, Michael Barrow, Mauro Marinoni, Alessandro Biondi, Giorgio Buttazzo, and Ryan Kastner. "ARTE: Providing real-time multitasking to Arduino". In: *Journal of Systems and Software* 186 (2022), p. 111185. ISSN: 0164-1212. DOI: <https://doi.org/10.1016/j.jss.2021.111185>. URL: <https://www.sciencedirect.com/science/article/pii/S016412122100265X>.
- [10] Ben Artin. *Protothreads for Arduino*. <https://gitlab.com/airbornemint/arduino-protothreads>. visited 24/03/2022. 2020.
- [11] Adam Dunkel. <http://dunkels.com/adam/pt/index.html>. Accessed: 23/03/2022. 2022.
- [12] *Arduino Protothreads [Tutorial]*. <https://roboticsbackend.com/arduino-protothreads-tutorial/>. visited 23/03/2022. The Robotics Back-End, 2019.
- [13] Jonathan Bartlett. *Eventually*. Feb. 23, 2022. URL: <https://github.com/johnnyb/Eventually> (visited on 03/07/2022).
- [14] *About FreeRTOS Kernel*. <https://www.freertos.org/RTOS.html>. visited 04/04/2022. FreeRTOS.

- [15] *Scheduling Basics*. <https://www.freertos.org/implementation/a00005.html>. visited 04/04/2022. FreeRTOS.
- [16] Hüttel Hans. *Transitions and trees: An introduction to structural operational semantics*. Cambridge University Press, 2011.
- [17] JavaCC Community. *JavaCC Github page*. <https://javacc.github.io/javacc/>. Accessed: 21-03-2022. 2021.
- [18] Parr Terence. *About ANTLR*. 2014. URL: <https://www.antlr.org/about.html>.
- [19] Parr Terence. *The Definitive ANTLR4 Reference*. 2nd ed. Pragmatic Bookshelf, 2014.
- [20] Terence Parr. *ANTLR Documentation*. visited 21/03/2022. URL: <https://github.com/antlr/antlr4/blob/master/doc/index.md>.
- [21] C. Scott Ananian, Frank Flannery, Dan Wang, Andrew W. Appel, and Michael Petter. *Cup 0.11b*. <http://www2.cs.tum.edu/projects/cup/index.php>. 2022.

Appendix A

Reading instructions

These are the reading instructions for the next supervisor meeting.

A.1 Review priority

1. Read and answer all green notes
2. Read 3
3. Ignore sources that are not input properly yet, except if they are marked by a green note.