Domsağo
a Spoken Programming Language
as a Step Towards
Personal Assistant Programming

Lior Samuel
Domsaĝo
a Spoken Programming Language
as a Step Towards
Personal Assistant Programming

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Lior Samuel

Submitted to the Senate
of the Technion — Israel Institute of Technology
Sivan 5780 Haifa May 2020
This research was carried out under the supervision of Prof. Yossi Gil, in the Faculty of Computer Science.

**Acknowledgements**

I would like to extend my most sincere gratitude to my parents and brothers, who taught me the values of hard work and perseverance.

I would also like to thank my adventurous friends, who stood by my side through this odyssey.

To Shir, the love of my life, Thank you for always having faith in me.

And finally to my advisor Yossi Gil, who made all of this possible.

The generous financial help of the Technion is gratefully acknowledged.
Contents

Abstract i

List of Figures iv

List of Tables v

List of Programs vii

List of Abbreviations ix

1 Introduction 1
  1.1 Challenges . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
  1.2 Contribution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.3 Why Esperanto? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2 Intelligent Personal Assistants 5
  2.1 Virtual Assistants Today . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
  2.2 Current Challenges . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
    2.2.1 Intents and Actions . . . . . . . . . . . . . . . . . . . . . . . . . 8
    2.2.2 A Learning Assistant vs Teaching an Assistant . . . . . . . . . . 8

3 Related Work 11
  3.1 Parsing of Esperanto . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
  3.2 Programming Languages and Natural Languages . . . . . . . . . . . . . 11
  3.3 Machine Translation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
  3.4 Constructed Programming Languages . . . . . . . . . . . . . . . . . . . 13
  3.5 Conversational Programming . . . . . . . . . . . . . . . . . . . . . . . . 14
  3.6 Vocal Programming . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

4 Domsaĝo 17
  4.1 Notation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
  4.2 Tokenization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
  4.3 Grammar . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
    4.3.1 Commands . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
F Examples

G Computational Power
   G.1 Hoare Logic .................................................. 75
   G.2 Turing Completeness ......................................... 76
   G.3 Announcing Primes .......................................... 86

H Installation Guide & Quick start
   H.1 Installing and Using the Demo .............................. 89
Abstract

Modern improvements in speech analysis, together with chatbot technology, gave rise to Intelligent Personal Assistants. These are programs with conversational interfaces that allow their users to execute simple tasks, query useful information, control smart devices, and more. Unlike a human assistant, however, these assistants cannot learn new things through conversation. Improvements can only be made through special applications or by programming new skills from scratch.

Composing a sequence of premade operations into a single operation is rather standard in other domains of computing: The shell interface of operating systems, from DOS’s batch language to Unix various shells, we find scripts, of sequencing and invoking simpler commands. These scripts make it possible to write simple “programs” without resorting to actual programming in a general purpose programming languages. This work explores the idea of employing the idea of scripting to personal assistants. The challenge here is in adapting the technical restrictions of standard scripts to the world of spoken instructions given in a natural language.

The difficulties of constructing such a language are:

- We want to communicate with our assistants in a natural language. However, natural languages are ambiguous by nature. Each sentence is conveying information not only through words but also through speech stops, tonal changes, and context.

- In contrast, to be used for programming this scripting language must be well structured, unambiguous, and impervious to subtext.

- Programming currently resides in text, while conversation mainly in voice and speech. Although we feel these mediums to be interchangeable, the transition between them is rarely straightforward in many regards.

This research proposes such a scripting language based on Esperanto, leveraging its simplicity for ease of parsing. I also describe a prototype for a similar English-based interpreter.
Prologue

This thesis describes the spoken scripting language called Domsaĝo (dom-sa-jo), meant to enhance personal assistants such as Alexa, Siri, and the Google assistant. The word Domsaĝo is an Esperanto word composed of ‘domo’—meaning house, and ‘saĝo’—meaning sage or wise man. Therefore, Domsaĝo aspires to be the sage who expertly manages the smart home and all of its appliances. The purpose of Domsaĝo is to allow the definition of new assistant skills in a manner as easy as the execution of preexisting ones.

Program 0.1 Example Domsaĝo Program

```
1 radiki nombro signifas
2 asignu du al radiko
3 poste dum radiko fojoj radiko estas pli malgranda ol nombro tiam
4 asignu unu pli radiko al radiko
5 finu
6 poste revenu radiko
7 finu
```

After reading this thesis, I hope you will be able to:

- Understand a few words of Esperanto
- Understand Program 0.1, and the meaning of its output given an input $X$.
- Install Domsaĝo on your machine, define the program above and invoke it with $X = 682$ receiving 27 as a result.
- Understand the abilities and limitations of Domsaĝo.
- Understand the difficulty of implementing Domsaĝo using natural languages.
- Realise the need for a simple interaction mechanism powerful enough to teach the intelligent personal assistant new skills, and through this, understand Domsaĝo’s niche in the intelligent personal assistant market.

There are also several issues which will not be addressed in this work:

- How to edit these verbal assistant routines dictated for Domsaĝo or a similar future language.
• A way to review existing verbal assistant routines or scheduled commands.

• How to achieve an accurate speech-to-text conversion of Esperanto.
# List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Intelligent Personal Assistant</td>
<td>8</td>
</tr>
<tr>
<td>4.1</td>
<td>Comparison Operators</td>
<td>24</td>
</tr>
<tr>
<td>A.1</td>
<td>Esperanto Tenses I</td>
<td>53</td>
</tr>
<tr>
<td>A.2</td>
<td>Esperanto Tenses II</td>
<td>53</td>
</tr>
<tr>
<td>A.3</td>
<td>Esperanto Pronouns</td>
<td>54</td>
</tr>
<tr>
<td>D.1</td>
<td>Domsaño Numbers</td>
<td>59</td>
</tr>
<tr>
<td>D.2</td>
<td>Domsaño Time Units</td>
<td>59</td>
</tr>
<tr>
<td>D.3</td>
<td>Prepositions</td>
<td>60</td>
</tr>
<tr>
<td>D.4</td>
<td>Domsaño Reserved Words</td>
<td>61</td>
</tr>
</tbody>
</table>
## List Of Programs

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>Example Domsago Program</td>
<td>iii</td>
</tr>
<tr>
<td>2.1</td>
<td>Silly Name Generator Example</td>
<td>9</td>
</tr>
<tr>
<td>4.1</td>
<td>Two Sequenced Announcements</td>
<td>31</td>
</tr>
<tr>
<td>4.2</td>
<td>Light State Announcement</td>
<td>31</td>
</tr>
<tr>
<td>4.3</td>
<td>Explosive Announcement</td>
<td>31</td>
</tr>
<tr>
<td>C.1</td>
<td>Excerpt from Talon programming demo</td>
<td>58</td>
</tr>
<tr>
<td>F.1</td>
<td>Nightify Routine</td>
<td>71</td>
</tr>
<tr>
<td>F.2</td>
<td>Alert When Water are Ready</td>
<td>71</td>
</tr>
<tr>
<td>F.3</td>
<td>Pretend To Be Home</td>
<td>72</td>
</tr>
<tr>
<td>F.4</td>
<td>Disciplining a Child</td>
<td>72</td>
</tr>
<tr>
<td>F.5</td>
<td>Morning Routine</td>
<td>73</td>
</tr>
<tr>
<td>F.6</td>
<td>Party Routines</td>
<td>73</td>
</tr>
<tr>
<td>F.7</td>
<td>Fabricate Sunrise</td>
<td>73</td>
</tr>
<tr>
<td>G.1</td>
<td>Constant Function</td>
<td>77</td>
</tr>
<tr>
<td>G.2</td>
<td>Successor Function</td>
<td>78</td>
</tr>
<tr>
<td>G.3</td>
<td>Projection Function</td>
<td>78</td>
</tr>
<tr>
<td>G.4</td>
<td>Composition Operator</td>
<td>79</td>
</tr>
<tr>
<td>G.5</td>
<td>Primitive Recursion Operator</td>
<td>81</td>
</tr>
<tr>
<td>G.6</td>
<td>Minimization Operator</td>
<td>83</td>
</tr>
<tr>
<td>G.1</td>
<td>Prime Announcing Routine</td>
<td>87</td>
</tr>
</tbody>
</table>
List of Abbreviations

AI  Artificial Intelligence
API  Application Programming Interface
EBNF  Extended Backus-Naur Form
GUI  Graphical User Interface
IoT  Internet of Things
IPA  Intelligent Personal Assistant
LALR  Look-Ahead Left-to-Right
NLP  Natural Language Processing
PoS  Part of Speech
StT  Speech to Text
VUI  Voice User Interface
Chapter 1

Introduction

In recent years the IPA market has grown from a global value of 796$ million in 2017 to about 3.6$ billion by 2020\(^1\). An IPA is a management application with which users can communicate through speech. Some of these assistants, however, can only handle a fixed set of commands. Other assistants are capable of chaining commands through a dedicated application, executing them in a specified order, allowing the user to create *shortcuts* for long command sequences, and some companies allow extending their IPAs with new *skills*, which requires programming these skills as dedicated applications.

Currently, assistant software architecture is composed of *intents* and *actions*. An assistant skill identifies key phrases, maps them to concrete intents, and caters to them by performing actions. Intents are essentially program states reflecting the user’s will, which actions are in charge of fulfilling—sometimes transferring the program state to a different intent in the process.

This approach does not scale to more complex applications. After deploying a skill, the user is limited to the narrow set of pre-designed intents, closed to extension or optimization. This approach severely limits the potential of voice communication.

In this work, I try to provide a more helpful assistant, capable of executing basic commands, chaining them together into more complex routines, and passing values between them. Most importantly, all of this functionality is enabled *through speech*. This objective is achieved by using a structured language, which allows the computer to compile sentences into code with but little prior knowledge of actual meaning. This work uses Esperanto as an assistant communication language and creates grammar for a vocal scripting language capable of operating a smart home. I tested this language using a virtual home, as seen in Fig. 1.1, which shows how the lights are turned on and change colors (the default color is white). For an Esperanto speaker, the speech presented in Fig. 1.1 would make perfect sense. Others should consult Chapter 4 for some insights into the structure of Domsaĝo.

This approach benefits those who are unable to create computerized routines textually, such as the visually impaired, or those physically incapable of using a keyboard. Of

\(^1\)https://www.alliedmarketresearch.com/intelligent-virtual-assistant-market
these, programmers suffering from repetitive stress injury have made the most progress in the field. To these crowds, vocal programming opens a range of possible occupations. Moreover, I believe that such an approach is the most natural way of enhancing IPA functionality since most IPA communication is done hands-free.[39]

In the computer world, exists the notion of a **power-user**—a person who studies the operating system he uses, and harnesses its features to enhance his work-flow. The goal of this work is to set the voice users of technology free from the constraints of their applications and turn them into power-users.

### 1.1 Challenges

When using a programming language, one expects deterministic results. That, in turn, forces a one-to-one translation between symbols and meaning. When reading code out loud, however, the verbal counterparts of most symbols are too long to pronounce effectively. Most symbols are thus translated as either half-stops or full-stops, which are not precisely recognizable by most Speech to Text (StT) engines.

However, stops are as much an integral part of language as speech itself. The sentence “I am inspired by cooking, my dog, and my family”—which professes my hobby of cooking, my admiration for my pet, and my appreciation of my kin—is quite different from “I am inspired by cooking my dog and my family.”—which tells mainly of my desire to boil those closest to me in a large cauldron. In the case of Domaço, the interpreter must tell apart function arguments only by spoken words. However when saying **kalkulu** **dek** **ses** **kaj** **sep** for some routine invocation, one cannot tell whether the arguments to the routine are 16 (**dek** **ses**) and 7 (**sep**), or 10 (**dek**), 6 (**ses**), and 7 (**sep**)
without the additional information conveyed by half-stops.

Another challenge of this thesis was grammatical correctness. All Domsaĝo scripts are legal Esperanto utterances. To this end, the interpreter had to deal with the few special-cases and complicated rules of Esperanto. The result is an unambiguous grammar that still sounds natural.

1.2 Contribution

The contributions of this project:

1. The spoken programming language Domsaĝo—the first Esperanto-based programming language, and first assistant scripting language.

2. A demonstration of using LALR compilation instead of Natural Language Processing (NLP) when processing natural language input.

3. An example of harnessing a designed language’s grammatical structure and word structure for compilation.

4. A proof that Domsaĝo is Turing complete.

5. A prototype for an English implementation.

6. The Domsaĝo language specification and the Domsaĝo interpreter are both products of this work.

1.3 Why Esperanto?

To realise the ideal of vocal programming, the language used should follow simple rules and patterns. Agglutinative languages in particular use a simple composition mechanism, appending simple words to each other in order to create more elaborate concepts, while leaving the component words intact. Since components remain recognizable as a part of the whole term, complex words are more easily deductible. This, in turn, reduces the complexity of linguistic analysis required to understand a given word. This mechanism can also be harnessed by the speaker to define new terms, and makes the language easily understandable overall.

Many natural and planned languages are agglutinative and could be viable options for this thesis. A brief list of examples would be:

- Indigenous North-American native languages such as Lakota or Nahuatl.
- Dravidian languages from south India, such as Tamil or Tulu.
- Turkic languages such as Turkish or Uzbek.
- Controlled languages such as Attempto Controlled English.
• The constructed languages Loglan and Lojban.

This research’s chosen language had to be regular, understandable, descriptive, and as unambiguous as possible. The planned language Toki Pona for example consists of under 130 words. [38] This might give an illusion of simplicity, but this brief vocabulary is compensated by long sequences of words and highly contextual interpretations for each sentence.

Loglan is a logical language created as a way to test the Sapir-Whorf hypothesis. [12] The hypothesis states that a person’s spoken language affects his way of thinking. By this reasoning, a logical language would make its speakers more logically inclined. As mentioned before, Robert Heinlein used Loglan as an interface language in his book “The Moon is a Harsh Mistress.” However, Loglan is hard to master by design, and this fact makes it a less appealing candidate. Another downside of Loglan is the existence of its more refined sibling: Lojban.

Lojban was created by the logical language group as an improvement of Loglan, applying various new concepts to ensure uniformity and relative ease of learning. It has a precise word conversion algorithm used to derive Lojban words from existing words. However, Lojban is still alien to a casual observer. Code snippets in Lojban would be hard to read, and any programmer would need to invest considerable effort to master it. Also, since Lojban is not only a planned language but arguably an unnatural language, its use as a spoken programming language would imply little if nothing on the ability of natural languages to be used for programming.

English should have been the most obvious candidate, and it receives a lot of market success for obvious reasons. However, English’s structure is not pure enough for simple parsing. The word “love” by itself has about seventeen different meanings as a noun and seven other meanings as a verb,² which are context-dependent. Controlled versions of English exist, Attempto Controlled English (ACE) being one of them [21], but these languages are declarative and are more fitting for logical reasoning than for programming. Knowing of a brilliant and profoundly different way to parse English without NLP would be cause for a patent, not a thesis.

Esperanto, whose history is elaborated in Appendix A, has some very desirable features. First, the ability to identify its parts of speech unambiguously is in itself of tremendous help to a parser. Another essential quality is its similarity to other European languages, making it easier to learn and roughly understand without learning. That will be evident in the code snippets of this thesis. Also, the Esperanto speaking community is massive, including approximately 2 million speakers—a population as large as Latvia³. So this research is immediately useful to a sizable group.

²According to Merriam-Webster’s dictionary: https://www.merriam-webster.com/dictionary/love
³https://www.worldometers.info/world-population/population-by-country/
Chapter 2

Intelligent Personal Assistants

2.1 Virtual Assistants Today

Let us quickly survey the IPAs available today, including typical use, supported languages, and extendability.

Since the first appearance of Siri in 2011, different software and electronics companies have launched IPA products. This staggering increase is due to advances in Artificial Intelligences (AIs) \[23, 43\] and computing power, \[42, 17\] which make it possible to interpret natural languages. Fig. 2.1 depicts the data processing done by most IPAs.

The most popular IPAs allow the creation of skills for enriching their basic behavior, creating active developer communities \[30\]. These IPAs are subject to constant comparisons by the ever-evolving market \[47\]. Although some assistants are capable of more than just Voice User Interface (VUI), this will be the sole focus of this chapter.

Siri was the first commercial IPA, created by Apple. \[54\] Since its debut in 2011, Siri remained the most popular virtual assistant, holding a 36% market share as late as June 2019. \[57\] At the time of its release, many other applications already made use of auditory output, and Siri was expected to replace them all through a single application. \[27\] Its initial skill set included calendar control and social network updates, and, most notably, its sense of humor. \[13\]

Nowadays, in addition to calendar and smart home manipulation, Siri allows the use of shortcuts. These predefined routines group several skill invocations by Siri in a single command. However, having no branching or memory mechanism, they do not grant the user any new capabilities. Siri’s users cannot extend it with new skills.

Google Assistant allows for basic control over Internet of Things (IoT) products, calendar manipulation, music, and offers real-time translation \[24\]. It was released in 2016 and is included in all android devices. The Google Assistant is extendable via the Google Home app, which houses both public assistant apps and personal
developments of assistant owners. The Google Assistant holds a 36% market share similar to Siri, [57] and fairs better on answer accuracy tests. [19, 34]

Google Assistant understands English, Spanish, Hindi, and other languages (as specified in Table 2.1). New assistant apps can be programmed in JavaScript and uploaded to the Google Cloud. Its users can extend it through “Actions On Google” and the Google Home app.

Alexa was released by Amazon in 2015, and its smart speaker, the Amazon Echo, has reached over 50 million homes by 2020\(^1\). Although Amazon dominates about 70% of the smart speaker market as a platform for Alexa, it is not yet available on smartphones, which might explain its market share as an assistant being as low as 25%. [57] Alexa boasts 37,000 skills that extend its IPA’s intrinsic capabilities for home automation, shopping, music streaming, phone calls, and messaging\(^2\). Programmers can enhance Alexa with skills written in Node.js, Java, Python, C#, or Go.

Cortana is an IPA developed by Microsoft for the operating system Windows 10, and

---

\(^1\)https://marketingland.com/alexa-devices-maintain-70-market-share-in-u-s-according-to-survey-265180

\(^2\)https://www.amazon.com/alexa-skills
devices such as Xbox. It was first released in 2014 and currently holds a market share of about 19%. [57] It saves information in a unique notebook system to better understand its owner, and leverage services such as Bing and Foursquare to gain some contextual understanding, similar to that of the Google Assistant. Although Cortana is technically extendable with assistant skills, Microsoft restricted its skills development kit to large organizations or security groups. Microsoft invests heavily in researching the assistant market and incorporating Cortana as a skill of other IPAs. This allows a user of any assistant to invoke Cortana to leverage its abilities over Microsoft Office 365 suite. [32] Cortana currently supports Chinese, English, French, German, Italian, Portuguese, and Spanish. [16]

Bixby is an IPA developed by Samsung Electronics. [10] It was first released in 2017 and currently understands Chinese, English, French, and Korean. Bixby is available in Samsung mobile devices, Galaxy Home, and Family Hub refrigerators but holds a market share as small as 4.2% of the entire assistant market in 2017. Although Bixby includes predefined routines for various situations, it has no skill mechanism to date.

Braina developed by Brainasoft is available for Windows computers. [11] Braina can receive voice or text commands in several different languages to control the computer. Premium Braina users can define custom commands, but this feature refers to shortcuts of existing commands and provides no extension to Braina’s capabilities.

Mycroft is an open-source voice assistant, developed to address the privacy and security concerns of IPA device users. [44] Mycroft can also be extended through software plugins similar to other assistant skills. Mycroft is officially available in English, albeit there is community support for other languages. Programmers wishing to extend Mycroft can use Python to write new skills. Since Mycroft is both open source and more easily extendable than any other IPA, it is closest to the ideal of an easily adaptable assistant. However, Mycroft can only be programmed in Python, and not in a natural language.

Alice is a Russian speaking assistant developed by Yandex and released in 2017. [51] It might be an early attempt at capitalizing on the Russian market by delivering an assistant speaking its native language. However, with the Google Assistant and Siri already speaking Russian, that effort might prove hopeless. Due to targeting a niche audience, very little research was dedicated to Alice, mostly detailing its high-level process diagram with no regard for specific capabilities. [46] Alice can

---

Table 2.1: Intelligent Personal Assistants compared to Domsaño

<table>
<thead>
<tr>
<th>Name</th>
<th>Developer</th>
<th>Language</th>
<th>Teaching Mechanism</th>
<th>Skill-Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexa</td>
<td>Amazon</td>
<td>English + 6 others</td>
<td>Skills Store</td>
<td>Java, Go, PowerShell, Node.js, C#, Python, Ruby</td>
</tr>
<tr>
<td>Alice</td>
<td>Yandex</td>
<td>Russian</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>AliGenie</td>
<td>Alibaba</td>
<td>Chinese</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Bixby</td>
<td>Samsung</td>
<td>English + 6 others</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Braina</td>
<td>Brainasoft</td>
<td>English</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Cortana</td>
<td>Microsoft</td>
<td>English + 8 others</td>
<td>Azure cloud</td>
<td>.NET, Node.js</td>
</tr>
<tr>
<td>Google Assistant</td>
<td>Google</td>
<td>English + 20 others</td>
<td>Google Play Store</td>
<td>JavaScript</td>
</tr>
<tr>
<td>Mycroft</td>
<td>Mycroft AI</td>
<td>English</td>
<td>Hardware memory</td>
<td>Python</td>
</tr>
<tr>
<td>Siri</td>
<td>Apple</td>
<td>English + 19 others</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Domsaño</td>
<td>This thesis</td>
<td>Esperanto</td>
<td>Speech, text</td>
<td>Esperanto</td>
</tr>
</tbody>
</table>

be hosted as a smartphone app and is installed on Yandex station speakers. Since 2018 Alice is extendable through Yandex.Dialogues, and allows the creation of new skills programmed in Python or Node.js. The creation of these skills involves both programming and uploading the skill to the Yandex cloud.

**AliGenie** was developed by the Alibaba Group and released in 2017. It speaks only Chinese and is installed on Tmall Genie smart speakers. Its primary purpose is to capitalize on the Chinese consumer market through assistant enabled purchases. Indeed, the smart speaker market in China has grown to 10.6 billion devices sold at the start of 2019. It is too early to tell at what point AliGenie will become available in other languages, if at all. It cannot be enhanced or programmed externally. [1]

Some studies have already tried to compare the most popular IPAs [40, 19, 34]. In this thesis, however, we are interested in the learning capabilities of each assistant. Listed in Table 2.1 are the features each IPA, compared to Domsaño to emphasize the goal of this research.

Table 2.1 shows that most IPAs can be operated in English for their day to day functions, with a small subset of them understanding additional languages. When it comes to extending the IPAs, however, the options are not as impressive. *Nearly half* of the above-listed IPAs do not have any teaching mechanism, and those that do, require programming skills in Python or another language. Such requirements make small improvements, customizations, and routines impossible for the average user. The purpose of Domsaño is to offer a simple scripting language for IPA extension. With this language, existing commands can be combined, allowing effortless flexibility in IPA usage. On a more surprising note, Domsaño will also be the first assistant to use a male voice. [27]
2.2 Current Challenges

2.2.1 Intents and Actions

Modern IPAs are based on an “intents and actions” architecture. *Intents* are data entities meant to represent the core of the request the user makes of the assistant, while *actions* are the possible responses to each intent. Assistant skills are developed (for the few IPAs capable of such things) as a set of intents, each mapped to several triggering phrases, along with the action responses to each intent and a set of transitions between them.

This architecture eliminates the difficulty of NLP for the average user by focusing on phrases and speech patterns instead of actual semantic processing of user input. Programmers also leverage this architecture, building new apps as conversations with the assistant.

This paradigm replaces imperative programming with elaborate JSON constructs, hiding the consequences of actions inside sparsely populated files. It makes for large source files that are hard to understand—let alone modify—by a third party. This architecture is not extendable by its very nature (unlike programs built under the Unix philosophy, for example).

2.2.2 A Learning Assistant vs Teaching an Assistant

IPAs may recognize behavioral patterns and suggest action sequences that would facilitate these patterns. For example, repeated instances of romantic music activation right after dimming the lights might indicate a habit to the IPA, suggesting the activation of music in future interactions. A user may also use the Graphical User Interface (GUI) available to some IPAs to manually define such a routine. In this thesis, I will try to provide a better solution through programming the assistant directly.

What about the skills systems of some IPAs? It allows for certain enhancements of the assistants’ capabilities, but only for a person familiar with programming, online app deployment interfaces and other such procedures specific for each assistant. It is not accessible to the average user. As a simple demonstration, the example voice application for beginning developers provided by Google is a silly name generator. This app merely asks the user for his lucky number and favorite color returning them both as a name [53]. This application is comprised of several JSON files totaling over 110 lines (not including parentheses), I posted a snippet of this application in Appendix B. To emphasize the unnecessary complexity, examine Program 2.1 as it implements the same functionality in Domsago.
Program 2.1 Silly Name Generator Example

```
sensencnomi signifas:
1. Anoncu citilo kio estas via
2. bonsxanca nombro malcitilo
3. poste atentu al bonsxanca nombro
4. poste anoncu citilo kio estas via
5. sxatata koloro malcitilo
6. poste atentu al sxatata koloro
7. poste anoncu sxatata koloro pli
8. bonsxanca nombro al sensenca nomo
9. estast sxatata koloro
10. poste anoncu citilo via sensenca nomo
11. poste anoncu sensecsa nomo
12. Finu.
```

To silly-name means:
1. Announce "what is your
2. lucky number?"
3. afterwards listen for lucky number
4. afterwards announce "what is
5. your favorite color?"
6. afterwards listen for favorite color
7. afterwards assign favorite color +
8. afterwards assign favorite color +
9. lucky number to silly name
10. afterwards announce "your silly
11. name is"
12. afterwards announce silly name.
13. End.

Research shows that assistants’ answer accuracy dropped in recent years\(^5\). It seems as if the questions are getting more diverse, but the programmatic effort to identify and match answers to them does not scale to meet demand. There is another way; however: users could educate their assistants and teach them to contend with challenging queries correctly. In Chapter 3, one can find an example of an assistant capable of being taught new concepts [37]. In this thesis, I aim to teach the assistant both new concepts and also complex routines.

---

\(^5\)https://www.perficientdigital.com/insights/our-research/digital-personal-assistants-study
Chapter 3

Related Work

In this chapter, I will present some other projects related to parsing Esperanto or studying its grammatical properties. The projects reviewed here have all either studied Esperanto linguistically or have studied spoken machine interaction without Esperanto.

3.1 Parsing of Esperanto

In her thesis B. C. Aasgaard [15] built a parser that extracts all possible meanings from Esperanto text, producing several derivation trees for each sentence. The parser would break each word to its component word stems and its derivation as a part of speech. In turn, it fed these tagged sequences to a generic syntax parser, which derived the grammatical structures used in the sentence. Aasgaard was mostly successful in her attempt to parse Esperanto. Her only failures were with pronouns and other specific words, which could be easily listed as exceptional cases.

In another research, Bick [8] created a constraint-grammar based parser for Esperanto, scoring an accuracy rate of 99.5%. He based his parser on the most straightforward rules of the Esperanto grammar, but the structure of the language was perfectly for the task. In contrast, Craneburgh [18] tried to leverage data-oriented parsing along with morphology and syntax to identify Esperanto parts of speech with a unified model. Although his results were reasonably accurate on smaller data sets, his Bitpar parser [52] was unable to produce any results for large data sets due to memory allocation failures.

3.2 Programming Languages and Natural Languages

Naur [45] tried to look at programming languages through the lens of the social aspects of mathematics and natural languages, taking their history as evidence of how programming languages must evolve. He argued that in programming languages as in natural languages, a more advanced language should be based on a small number of highly abstract features, being powerful enough to convey any possible meaning. He also mused that more advanced programming languages would use orderly programmatic structures
Figure 3.1 Programming language communities according to the TIOBE index, May 2019

rather than labels and go-to statements. Based on the experience of Otto Jespersen in the delegation for the adoption of an international auxiliary language, Naur theorized that beyond a particular minimum adoption requirement, the choice of language is made based on personal preference rather than on objective value. Accordingly, he predicted that Cobol and Fortran would remain the most popular programming languages, with other languages developing slowly and with little interaction—just like Esperanto has retained some of its defects over the years instead of fully evolving into Ido. Gobbo [22] has reached the same conclusion in his comparison between the search for a universal programming language and a universal auxiliary language.

This prediction is amusing to the modern reader, who knows these programming languages are comparable to stone tools. Moreover, the TIOBE index [58] tracks the communities of over two hundred distinct programming languages and does not even list Cobol and Fortran as part of its top ten, see Fig. 3.1.

3.3 Machine Translation

Some of the first machine translation projects used Esperanto as an intermediate translation language. The notion of an intermediate language provides an easy way to translate several languages into one another. Specifically, given a large group of languages $L$ translatable through the intermediate language $I$, and a new language $L'$, effort must be put only to translate $L' \rightarrow I$ and $I \rightarrow L'$ (2 translation mechanisms) as demonstrated in Fig. 3.2 instead of $L' \rightarrow l_i : \forall L_i \in L$ (which would require $\binom{|L|}{2}$ translation mechanisms).

The first to use Esperanto as an intermediate language was Petr Petrovich Troyanskii, [31] who built a machine capable of translating text from one language into several

---

1http://interlanguages.net/Hist.html
Figure 3.2 Translation through an intermediate language

![Diagram showing translation through intermediate languages](image)

others simultaneously. Another example of such use is the DLT (Distributed Language Translation) project. [64] The project used a somewhat simplified version of Esperanto as an intermediate language and showed a working prototype translating English to French.

### 3.4 Constructed Programming Languages

An effort similar to this work has been made to use the constructed language Lojban. Lojban is a successor of the logical language Loglan, developed by James Cooke Brown to investigate the Sapir-Whorf hypothesis [49, 63]. The logical language group developed Lojban and first published it in 1997. In their paper, Andrews et al. [55, 2] have constructed a reasoning-engine which could be fed information in the Lojban language, reason it in comparison to its known data, and return an answer spoken in Lojban. Kreczmar et al. [36] attempted a Loglan programming language but their efforts were discontinued. An essential difference between this work and the Lojban work is that
Lojban is not designed first and foremost for human communication, but rather for being structurally unambiguous above all else. Similarly, the purpose of Loglan was to examine the effects of such a language on the speaker’s mind predicted by the Sapir-Whorf hypothesis. Therefore, Lojban is less oriented towards human usage and convenience than Esperanto is. This led us to believe that Esperanto, as a programming language, while being sufficiently unambiguous, will be far easier to learn and to master.

3.5 Conversational Programming

Azaria et al. [4] developed a digital assistant called LIA (Learning by Instruction Agent) to implement a concept of conversational programming—meaning users could teach LIA new routines and concepts through dialogue. At any given interaction, a user could define a routine for LIA in natural language, and the assistant software would ask for clarification of any unknown concepts. This way, new concepts could be taught by the user only when necessary, allowing for a much smoother interaction than bottom-up programming. LIA parses its input through a continually evolving context-free grammar, in order to quickly identify different command patterns. It also identifies potential problems with user-defined routines and branches them for special treatment accordingly. These branches are also brought up for the user to define the special treatment required in each branch. [37] By learning new concepts from user explanations, LIA can recognize new concepts without any tagged examples to learn from, an ability called zero-shot learning. [56] Although this project shows promising results in human-computer interaction, I hope in this research to provide an entirely unambiguous and robust method for understanding human input. LIA also has some disadvantages of its own; for example, adding new commands to the grammatical structure of LIA causes parsing errors when future invocations try to elaborate a known command. [37] Still, new capabilities are added continuously to LIA [14], and I see it as a role model to the heights achievable through personal assistant programming.

In another fascinating research, McCarthy [41] challenged the programming languages of the new millennium with the proposal of a programming language based on speech-act theory called Elephant 2000. McCarthy designed the language to receive requests, questions, offers, acceptances of offers, permissions, answers to questions, and assertions of fact. Elephant programs could output these language constructs along with promises and commitments to be upheld by the program. Elephant also has a “direct memory of the past” rather than data structures and detailed instructions. McCarthy is purposefully ambiguous about the nature of the memory and mechanics of the language. He meant it to know as little as possible about every concept it dealt with, manipulating it only through the patterns of obligations given to it and a perfect recollection of the past. His idea was implemented later by Holmes [29] through an interpreter based on a named entity recognition engine and an LALR parser. The implementation is barely explained and gives no examples except for a brief flight reservation program described in the
original paper by McCarthy. Therefore, it is impossible to determine how well the language works and what it is capable of.

3.6 Vocal Programming

Another recently developing field is that of programming using vocal commands instead of keyboards. Vocal programming is developed as a way to prevent repetitive stress injuries for full-time programmers. It was researched as early as 2005 by Begel and Graham [6, 7, 5], and continued later by Gordon [25, 26]. Professionals such as Wengel [62, 61] have been following the progression of this technology [3, 50] in recent years, but are critical of its unintuitive nature.

Since modern vocal programming tools rely heavily on arbitrary keywords in order to simulate keystrokes, their operating language inevitably becomes confusing to the untrained ear, with words like “slap, tark, chook” being common utterances [48]. Users of these tools report that acquiring basic proficiency takes as long as two months.

Talon [28] is a more sophisticated system that leverages both an StT engine and an eye-tracking device. This pairing allows Talon to simulate a keyboard via voice commands and a mouse using the eyes and non-verbal cues for navigability. Although it grants many options to the user, Talon only simulates a computer usage experience using the keyboard and mouse. One could say that although the programmer’s muscles are free from grasping these objects, their mind is needlessly holding onto them. It is my opinion that migration to the world of voice should not be based on simulating legacy computer gear, but that these tools should be cast aside when facing a more natural communication option. In that sense, Talon is no more convenient or less cumbersome than sticking to the mouse and keyboard.

I hope to provide a more straightforward yet just as powerful alternative by using a spoken language with practical grammar instead of unique keywords.
Chapter 4

Domsaĝo

The product of this research is the scripting language, Domsaĝo. It is a programming language designed to automate a smart home by dictating commands vocally. Its characteristics are:

1. Domsaĝo code is naturally verbal. Any Domsaĝo program can be read without loss of meaning and understood by any Esperanto speaker.

2. Domsaĝo is interpreted via traditional parsing techniques. It is compiled with an LALR parser, without the costly usage of modern NLP.

3. Domsaĝo is expressive enough to enable essential appliance control as well as complicated smart-home automation tasks.

4. Domsaĝo is also proven to be Turing complete.

I have already defined the name Domsaĝo in the prologue on page iii as a wise man controlling the house. In this chapter, I will show just how wise it is.

4.1 Notation

Before describing Domsaĝo’s grammar, I must clarify my use of Extended Backus-Naur Form (EBNF):

- Terminal symbols are written with a ‘green fixed − width font enclosed in single quotes.’ These are keywords that are said in Esperanto precisely as written.

- Non-terminals are written as ⟨italics enclosed in triangle brackets⟩. These are derivation symbols for Domsaĝo’s grammar.

- A derivation rule is written as →
Take the following rule as an example:

\[
\text{⟨if command⟩} \rightarrow
\text{‘se’} \text{ ⟨expression⟩} \text{ ‘tiam’} \text{ ⟨sentence⟩} \text{ ‘finu’}
\]

- If a single non-terminal may be derived in many ways, the different productions appear in distinct lines, although short alternatives are just separated by a column (|).

As an example, the following rule adds one production to the former:

\[
\text{⟨if command⟩} \rightarrow
\text{‘se’} \text{ ⟨expression⟩} \text{ ‘tiam’} \text{ ⟨sentence⟩} \text{ ‘finu’}
\text{‘se’} \text{ ⟨expression⟩} \text{ ‘tiam’} \text{ ⟨sentence⟩} \text{ ‘alie’} \text{ ⟨sentence⟩} \text{ ‘finu’}
\]

- A part that may appear once in the production or not at all is [surrounded by red square brackets]

Therefore the following derivation rule is equivalent to the former:

\[
\text{⟨if command⟩} \rightarrow
\text{‘se’} \text{ ⟨expression⟩} \text{ ‘tiam’} \text{ ⟨sentence⟩} \text{ [‘alie’} \text{ ⟨sentence⟩]} \text{ ‘finu’}
\]

- Sometimes the production of a variable must be suffixed by the following terminal. This is denoted by writing ⟨non-terminal⟩‘suffix.’

The following rule is an example of simple fractions in Esperanto, such as duono, triono, etc.

\[
\text{⟨fraction literal⟩} \rightarrow \text{[⟨other digit⟩]} \text{‘ono’}
\]

- Finally, a part may appear zero or more times in the production. In that case, it is {surrounded by red curly braces}.

4.2 Tokenization

As an international language, Esperanto is easy to learn by design: its words are formed in easily identifiable patterns. Domsago leverages this ease of identification in the following ways\(^1\):

\(^1\)The symbol letter refers to any letter of the Latin alphabet, see Appendix A for details of writing and reading Esperanto
• Esperanto **Nouns** are words that end with *o*. Plural nouns end with *oj*. Esperanto nouns can be spoken with an accusative case, which appends an *n* at the end of the word. Domsaĝo currently ignores this annotation.

*Derivation 1.*

\[
\langle \text{noun} \rangle \rightarrow \{\langle \text{letter} \rangle\}^* \text{'o'}[^*] \text{'j'}[^*] \text{'n'}
\]

• **Adjectives** in Esperanto end with *a* and are similarly added *j* if plural. They can also end with an *n* to become accusative along with their noun.

*Derivation 2.*

\[
\langle \text{adjective} \rangle \rightarrow \{\langle \text{letter} \rangle\}^* \text{'a'}[^*] \text{'j'}[^*] \text{'n'}
\]

• **Prepositions** in Esperanto are words such as *inter*\(^2\), *trans*\(^3\) etc\(^4\). Domsaĝo uses them as *separators* between arguments and parameters. Other legal separators in Domsaĝo are the comma and the keyword *kaj*\(^5\) mentioned hereafter. The specific meaning of each word is irrelevant when used as a separator.

*Derivation 3.*

\[
\langle \text{separator} \rangle \rightarrow \text{','} | \text{'kaj'} | \langle \text{preposition} \rangle
\]

• The definition and invocation of routines in Domsaĝo are done using verbs. **Infinitive** verbs are *i*-terminated words and are used for definition. **Imperative** verbs are used for routine invocation and are terminated by *u*. Domsaĝo also uses **present** verbs (terminated by *as*) to inspect the state of appliances, as will be explained in Section 4.4. The tokenizer can easily identify different verb tenses with the same *word stem* and associate between them.

*Derivation 4.*

\[
\langle \text{infinitive verb} \rangle \rightarrow \{\langle \text{letter} \rangle\}^* \text{'i'}
\]

*Derivation 5.*

\[
\langle \text{imperative verb} \rangle \rightarrow \{\langle \text{letter} \rangle\}^* \text{'u'}
\]

*Derivation 6.*

\[
\langle \text{present verb} \rangle \rightarrow \{\langle \text{letter} \rangle\}^* \text{'as'}
\]

• Domsaĝo uses the keyword *gxi*\(^6\) to reference the last calculated value or object,

---

\(^2\)inter: inside  
\(^3\)trans: beyond  
\(^4\)For an exhaustive list of prepositions in Esperanto see Appendix A.  
\(^5\)kaj: And  
\(^6\)gxi: it
allowing for information passing between functions and for referencing the last
value a Domsaĝo routine has provided.

Derivation 7.

\[ \langle i \rangle \rightarrow \langle \text{gxi} \rangle \langle n \rangle \]

The Domsaĝo tokenizer recognizes these rules and classifies each word according to
its Part of Speech (PoS). This classification provides a variety of different token types
to the syntactic analysis phase, comparable to traditional programming languages.

4.3 Grammar

The start symbol for the EBNFs grammar of Domsaĝo is a \textit{speech}; a Domsaĝo speech
derives a \textit{sentence} grammar symbol, or a single \textit{routine definition}.

Derivation 8.

\[ \langle \text{speech} \rangle \rightarrow \langle \text{sentence} \rangle \mid \langle \text{routine definition} \rangle \]

Where a \textit{sentence} is a collection of \textit{serial commands} separated by the keyword \textit{poste}.

Derivation 9.

\[ \langle \text{sentence} \rangle \rightarrow \langle \text{serial command} \rangle \{ \langle \text{poste} \rangle \langle \text{serial command} \rangle \} \]

Each serial command is composed of several \textit{parallel commands}, separated by the
keyword \textit{samtempe}.

Derivation 10.

\[ \langle \text{serial command} \rangle \rightarrow \langle \text{parallel command} \rangle \{ \langle \text{samtempe} \rangle \langle \text{parallel command} \rangle \} \]

4.3.1 Commands

Commands are either \textit{atomic} (i.e., commands that contain no other commands in them)
or \textit{compound} (i.e., commands that engulf other commands).

Derivation 11.

\[ \langle \text{parallel command} \rangle \rightarrow \langle \text{atomic command} \rangle \mid \langle \text{compound command} \rangle \]

Atomic commands include variable \textit{assignments}, routine \textit{invocations}, and \textit{return}
commands from a currently executing routine:

Derivation 12.

\[ \langle \text{atomic command} \rangle \rightarrow \langle \text{assignment} \rangle \mid \langle \text{invocation} \rangle \mid \langle \text{return} \rangle \]

*poste*: afterwards
*samtempe*: simultaneously
The semantics of each atomic command are explained in Section 4.3.1. Functions and routines are explained later in Section 4.3.2, and their definitions explained in Section 4.6.

As for compound commands, Domsaĝo offers the usual conditional and iterative control flow:

*Derivation 13.*

\[
\langle \text{compound command} \rangle \rightarrow
\langle \text{if command} \rangle
\langle \text{while command} \rangle
\langle \text{for command} \rangle
\]

It also includes some special compound commands, explained in Section 4.7.

*Derivation 14.*

\[
\langle \text{compound command} \rangle \rightarrow \ldots
\langle \text{delayed command} \rangle
\langle \text{scheduled command} \rangle
\langle \text{repeating command} \rangle
\langle \text{once command} \rangle
\langle \text{whenever command} \rangle
\]

**Atomic Commands**

*Assignments* in Domsaĝo use dynamic typing, so any data-type described in Section 4.5 is compatible with any identifier.

*Derivation 15.*

\[
\langle \text{assignment} \rangle \rightarrow \text{\texttt{assignu}} \langle \text{expression} \rangle \text{\texttt{al}} \langle \text{assignable} \rangle
\]

*Derivation 16.*

\[
\langle \text{assignable} \rangle \rightarrow \langle \text{name expression} \rangle
\]

The different types of *assignable* will be explained later in der. 43.

*Invocations* of routines can accept parameters.

*Derivation 17.*

\[
\langle \text{invocation} \rangle \rightarrow \langle \text{imperative verb} \rangle \langle \text{arguments} \rangle
\]

A parameter can be any expression, separated from its neighbors as described at der. 3. Some routines require no parameters at all.

*Derivation 18.*

\[
\langle \text{arguments} \rangle \rightarrow [\langle \text{expression} \rangle \{\langle \text{separator} \rangle \langle \text{expression} \rangle\}]
\]
Domsaĝo allows free use of prepositions to help the user express the relation between different method arguments. As an intuitive example, consider the python invocation `randint(1,10)` or Domsaĝo command `hazardu nombro, unu, dek` as opposed to `hazardu nombron inter unu kaj dek` which means `generate a random number between 1 and 10`.

Domsaĝo does not verify these prepositions, and so the final responsibility for speaking sensible commands rests on the shoulders of the user.

Finally, `return` commands in Domsaĝo can pass values outside of routines. However, since routine invocations are commands and not expressions, the values are not passed directly. The return value of a procedure is available through the keyword `gxi` or its accusative case `gxin`.

4.3.2 Functions and Routines

In Domsaĝo, routines are sets of subsequent commands, similar to shortcuts possible for other IPAs, except that unlike shortcuts, Domsaĝo routines receive arguments to their invocation.

The predefined routines of Domsaĝo are:

- `anoncu` outputs a message through all speakers available to Domsaĝo.
- `presu` commands Domsaĝo to present a message on any available screen, like the `print` routine of any language.

\[gxi: it\]
\[se: if\]
\[tiam: then\]
\[alie: otherwise\]
\[dum: while\]
• **sxaltu** is a command to activate a device. The only argument to this routine is the name of the device. For example, to turn on a reading light, one could command `sxaltu la legan lumon` likewise, devices are turned off with the routine `malsxaltu`.

Functions are similar to routines, except that they return a value using the keyword `gxi` as described in der. 19.

Domsaĝo has several predefined functions:

• **atentu** is a command to receive input from the user, as used in Program 2.1. It is similar to concepts such as `input()` in Python or `read` in Bash.

• **hazardu** receives a data type to generate, and possibly a lower and an upper bound. It then generates a random value of said parameters. Currently, it can generate only numbers, time spans, and time points.

### 4.4 Expressions

Domsaĝo supports simple arithmetic operations, comparisons, and boolean operations.

#### Arithmetics

The *factor* symbol is used as the first step in expression compilation to manage associativity. It can be constructed from a literal, a variable, or an expression encapsulated between *maldekstra krampo* and *dekstra krampo*. Negative factors are created via the keyword `malpli`.

**Derivation 22.**

\[
(factor) \rightarrow \begin{align*}
\text{‘malpli’ } & (factor) \\
\text{ verbal number } & \\
\text{name } & \\
\text{‘maldekstra krampo’ } & (expression) \ ‘dekstra krampo’
\end{align*}
\]

A factor can be elevated to a *term* symbol if it is multiplied via `fojoj`, divided via `partoj`, or stands by itself.

**Derivation 23.**

\[
(term) \rightarrow \begin{align*}
(term) \ ‘fojoj’ & (factor) \\
(term) \ ‘partoj’ & (factor) \\
(factor) &
\end{align*}
\]

---

14 *maldekstra krampo*: left parenthesis  
15 *dekstra krampo*: right parenthesis  
16 *malpli*: minus  
17 *fojoj*: times  
18 *partoj*: parts
Subtraction is done with the reserved word `malpli` mentioned before, and addition is done via `pli`\(^{19}\).

**Derivation 24.**

\[
\langle \text{expression} \rangle \rightarrow \\
\langle \text{expression} \rangle \ 'pli' \ (\text{term}) \\
\langle \text{expression} \rangle \ 'malpli' \ (\text{term}) \\
\langle \text{term} \rangle
\]

This rule elevates a term to an *expression* symbol.

**Relations And Comparisons**

Any relation in Domsaço can be applied normally or negated with the keyword `ne`\(^{20}\).

**Derivation 25.**

\[
\langle \text{expression} \rangle \rightarrow :: \\
\langle \text{expression} \rangle \ 'ne' \ (\text{relation}) \ (\text{expression})
\]

To accommodate the vocal paradigm of Domsaço, the familiar operators `<, >, =` were replaced by the keywords: `granda`\(^{21}\), `malgranda`\(^{22}\) and `egala`\(^{23}\). The relations are prefixed by `pli`\(^{24}\), suffixed by `ol`\(^{25}\) or `al`\(^{26}\) and can be joined together by `aux`\(^{27}\).

**Derivation 26.**

\[
\langle \text{relation} \rangle \rightarrow \\
'\text{estas egala al}' \\
'\text{estas pli granda ol}' \\
'\text{estas pli granda aux egala al}' \\
'\text{estas pli malgranda aux egala al}' \\
'\text{estas pli malgranda ol}'
\]

The meaning of these relations are summarized in Table 4.1.

**Boolean Expressions**

`vero`\(^{28}\) and `malvero`\(^{29}\) are the boolean constants of Domsaço. The binary logic operators

\[^{19}\text{pli: more, plus}\]
\[^{20}\text{ne: not}\]
\[^{21}\text{granda: big}\]
\[^{22}\text{malgranda: small}\]
\[^{23}\text{egala: equal}\]
\[^{24}\text{pli: more}\]
\[^{25}\text{ol: than}\]
\[^{26}\text{al: to}\]
\[^{27}\text{aux: or}\]
\[^{28}\text{vero: true}\]
\[^{29}\text{malvero: false}\]
Table 4.1: possible comparison operators in Domsaĝo

are ambaux\(^{30}\), aux mentioned above, and the unary negation operator is ne. \(^{31}\)

Another way to yield boolean values is through state queries, which are done using present verbs. These queries check whether an appliance is performing a particular function, or is in a specific state (for example, turned on).

4.5 Types and Literals

Domsaĝo is a dynamically and weakly typed language, which means that its variables may change types freely at run time. Most types in Domsaĝo are simple enough to be created on-the-fly by speech. As in other languages, these freely created values are called literals. Literals come in four varieties: number and string literals are similar to what we find in other programming languages. The time points and time span literals in Domsaĝo are unique.

4.5.1 Number Literals

Numerical literals can be defined by a sequence of digits:

\(^{30}\)ambaux: both

\(^{31}\)The use of the word kaj would be preferable in this context, but using kaj as a binary logic operator would conflict with its use as a separator.
Derivation 29.

\[
\langle \text{number literal} \rangle \rightarrow \langle \text{digits sequence} \rangle
\]

where there are distinct rules for each number of digits.

Derivation 30.

\[
\langle \text{digits sequence} \rangle \rightarrow
\begin{align*}
'\text{nul}' \\
\langle \text{one digit literal} \rangle \\
\langle \text{two digit literal} \rangle \\
\langle \text{three digit literal} \rangle \\
\cdot
\end{align*}
\]

Like in English, the digits \text{nul}^{32} and \text{unu}^{33} are treated differently from other digits since, for example, they cannot be most significant digits.

Derivation 31.

\[
\langle \text{other digit} \rangle \rightarrow '\text{du'} | '\text{tri'} | '\text{kvar'} | '\text{kvin'} | '\text{ses'} | '\text{sep'} | '\text{ok'} | '\text{naux'}
\]

Single digit numbers are mundane:

Derivation 32.

\[
\langle \text{one digit literal} \rangle \rightarrow '\text{unu'} \mid \langle \text{other digit} \rangle
\]

Two-digit numbers have their most significant digit denoted by the suffix \text{dek}^{34}, and then their least significant digit.

Derivation 33.

\[
\langle \text{two digit literal} \rangle \rightarrow \langle \langle \text{other digit} \rangle '\text{dek'} \mid (\langle \text{one digit literal} \rangle) \rangle
\]

This rule captures \text{dek} (10), \text{deku} (11), \text{dudek} \text{ses} (26), \text{nau} \text{dek} \text{naux} (99), and others while forbidding invalid Esperanto phrases such as \text{unudek} ("one time ten"), and \text{tridek nul} ("thirty and zero").

The rule for three digits is similar, using the suffix \text{cent}^{35} for the most significant digit.

Derivation 34.

\[
\langle \text{three digit literal} \rangle \rightarrow \langle \langle \text{other digit} \rangle '\text{cent'} \mid (\langle \text{two digit literal} \rangle) \rangle
\]

The derivation rules for four, five, and more digits are omitted. An example of deriving a number literal by the above rules is given in Fig. 4.1.

---

\text{\textsuperscript{32}nul: zero}  
\text{\textsuperscript{33}unu: one}  
\text{\textsuperscript{34}dek: ten}  
\text{\textsuperscript{35}cent: hundred}
Numerical literals can also be duono (half), du triono (two thirds) and other fractions.

Derivation 35.

\[
\langle \text{number literal} \rangle \rightarrow : \\
\langle \text{fraction literal} \rangle
\]

The fractions 1/2, 1/3, …, 1/9 are described by:

Derivation 36.

\[
\langle \text{fraction literal} \rangle \rightarrow [\langle \text{other digit} \rangle] \text{ono}'
\]

The derivation rules for more complex fractions are omitted.

### 4.5.2 Time Points

**Time Points** are used to trigger future events or mark special occasions.

Derivation 37.

\[
\langle \text{time point} \rangle \rightarrow \\
\langle \text{round hour} \rangle \\
\langle \text{round hour} \rangle \ 'kaj' \ \langle \text{natural literal} \rangle \ 'minutoj' \\
\langle \text{round hour} \rangle \ 'kaj' \ \langle \text{fraction literal} \rangle
\]

Derivation 38.

\[
\langle \text{round hour} \rangle \rightarrow \ 'la' \ \langle \text{numerator} \rangle \ 'horo'
\]

The numerators mentioned above are adjectives\(^36\) that refer to a number.

Derivation 39.

\[
\langle \text{numerator} \rangle \rightarrow \langle \text{natural literal} \rangle \ 'a'
\]

Domsaĝo uses the 24-hour system; for example, the derivation tree for the time point 18:20 (06:20 PM) is in Fig. 4.2

\(^36\)reminder: any ‘a’ terminated word is an adjective in Esperanto
4.5.3 Time Spans

*Time Spans* in Domsaĝo are used for lengthy actions, such as the time a boiler might need to heat water, or the time it takes to bake a cake.

Derivation 40.

\[ \langle \text{time unit} \rangle \rightarrow ('\text{tago}' \mid '\text{horo}' \mid '\text{minuto}' \mid '\text{sekundo}') [']\]

Derivation 41.

\[ \langle \text{time span} \rangle \rightarrow \]
\[ (\langle \text{natural literal} \rangle \langle \text{time unit} \rangle \{\langle \text{delimiter} \rangle \langle \text{natural literal} \rangle \langle \text{time unit} \rangle\}) \]
\[ (\langle \text{natural literal} \rangle \langle \text{time unit} \rangle 'kaj' \langle \text{fraction literal} \rangle) \]

The time units above are those useful for everyday tasks. Larger time units exist, though omitted.

Fig. 4.3 is an example parse tree for the expression *du tagoj kaj duono*\(^{37}\):
4.5.4 Strings

*Strings* are created traditionally, with the keywords `maldekstra citilo`\(^{38}\) and `dekstra citilo`\(^{39}\) used instead of the usual quotation marks. For the sake of conciseness, I added shorter keywords: `citilo`\(^{40}\) `malcitilo`\(^{41}\).

**Derivation 42.**

\[
\langle \text{string} \rangle \rightarrow '\text{maldekstra citilo}' \langle \langle \text{letter} \rangle \mid .\rangle '\text{dekstra citilo}'
\]

\[
'\text{citilo}' \langle \langle \text{letter} \rangle \mid .\rangle '\text{malcitilo}'
\]

Again, the symbol `letter` refers to any letter in the Latin alphabet.

4.5.5 None

Similar to the *None* of python and the *null* of Java, Domsaľo also possesses an empty data type. It is identified by the literal `Nenio`\(^{42}\), and is used in a similar fashion to other programming languages.

4.6 Identifiers

In Section 4.3.1, I discussed assignments in Domsaľo. Values in Domsaľo can be assigned to any of the following:

**Derivation 43.**

\[
\langle \text{name expression} \rangle \rightarrow \langle \text{name} \rangle \mid \langle \text{field access} \rangle \mid \langle \text{array access} \rangle
\]

**Derivation 44.**

\[
\langle \text{field access} \rangle \rightarrow \langle \text{name expression} \rangle 'de' \langle \text{name expression} \rangle
\]

**Derivation 45.**

\[
\langle \text{array access} \rangle \rightarrow \langle \text{name expression} \rangle 'a' 'de' \langle \text{name expression} \rangle
\]

Variables are known by *names*, where a name is a *noun* prefixed by any number of *adjectives*.

**Derivation 46.**

\[
\langle \text{name} \rangle \rightarrow ['la'] \langle \langle \text{adjective} \rangle \rangle \langle \text{noun} \rangle
\]

\(^{38}\text{maldekstra citilo: right quotation mark}\)

\(^{39}\text{deksstra citilo: left quotation mark}\)

\(^{40}\text{citilo: quotation mark}\)

\(^{41}\text{malcitilo: opposite quotation mark}\)

\(^{42}\text{Nenio: None}\)
These names are identifiers: entities, each referring to some part of the machine memory. More complex identifiers are made via the \texttt{de} keyword.

The other type of identifier is routine names, which are verbs—defined in their infinitive form and invoked in their imperative form. These forms are described in der. 4 and 5

\textit{Derivation 47.}

\[
\langle \text{definition} \rangle \rightarrow \langle \text{infinitive verb} \rangle \langle \text{arguments} \rangle \text{`
\text{signifas}' `sentence' `finu'}
\]

A routine can be defined with zero or more parameters.

\textit{Derivation 48.}

\[
\langle \text{parameters} \rangle \rightarrow [\langle \text{name} \rangle \{\langle \text{separator} \rangle \langle \text{name} \rangle \}]
\]

As discussed regarding der. 18 Domsaĝo does not verify the syntactical correctness of the code in Esperanto, e.g., \texttt{sxalti la kato sur la knabo} is a valid start for a Domsaĝo routine definition.

\subsection{Procedures, Data Types and Citizenship}

In some languages, functions are considered “first-class citizens”, while in other languages, this is not the case. For example, take a look at the following python code:

\begin{verbatim}
1 a=0
2 a=lambda: 0
\end{verbatim}

The first line assigns a numerical value to ‘a’ while the following line binds it to a function. Both cases are equally valid. Such is not the case in Domsaĝo. First, data related identifiers are nouns in Domsaĝo while procedure names are verbs, and no Esperanto word (hence no legal Domsaĝo identifier) can be both at the same time. It would be possible to ensure that Domsaĝo assigns meaning to words per word stem (which would yield the same variable for function definition as for data assignment), but would that be more desirable?

Consider the word ‘index,’ which functions as both a verb and a noun in English. Should we give up on the noun index forever if we want an indexing procedure? Indeed, in my opinion, the action of indexing and the noun index should be able to coexist. For this reason, there are no mechanisms for anonymous functions in Domsaĝo, and there is no valid utterance that could transfer a method from a verb identifier to a noun identifier.

---

\texttt{de}: of
\texttt{sxalti la kato sur la knabo}: to activate the cat on top of the child
\texttt{meaning that functions can be assigned to identifiers just like any data value

30
4.6.2 Appliances

**Appliances** in Domsa£o are a form of memory-mapped IO. I divide them into several categories:

**Switches** are appliances with a boolean state, such as regular light bulbs.

**Knobs** can receive a value within a given range, e.g., dimmers or other power regulators.

**Accumulators** collect data through sensors or some virtual feed. A sensor is either recording, paused, or stopped. Examples in this category are audio recorders, cameras, thermostats and boiler sensors.

**Transmitters** are a general category for appliances which provide an output channel from the smart-home system. This category includes screens, speakers and, radio transmitters.

4.7 A Matter of Time

Domsa£o has several temporal modifiers to commands, which can cause the commands to execute later instead of immediately upon utterance. The simplest form of temporal modification is done with the keyword **je**\(^{46}\).

Derivation 49.

\[
\langle scheduled \, command \rangle \rightarrow \\
\langle command \rangle \, \textquotesingle je \textquotesingle \, \langle time \, point \rangle \\
\langle command \rangle \, \textquotesingle je \textquotesingle \, \langle time \, span \rangle
\]

One can also use the keyword **cxiu**\(^{47}\) to set repeating commands to reoccur after a set time indefinitely. Domsa£o will continue to re-invoke the command at regular intervals until canceled.

Derivation 50.

\[
\langle repeating \, command \rangle \rightarrow \langle command \rangle \, \textquotesingle cxiu \textquotesingle \, \langle time \, span \rangle
\]

Commands that are triggered once and then forgotten are called *once commands*. They are set with the keyword **unufoje**\(^{48}\). The trigger to perform the command should be an expression that Domsa£o evaluates periodically. Once the expression becomes true, the action is triggered.

Derivation 51.

\[
\langle once \, command \rangle \rightarrow \langle command \rangle \, \textquotesingle unufoje \textquotesingle \, \langle expression \rangle
\]

\(^{46}\) je: at

\(^{47}\) cxiu: every

\(^{48}\) unofoje: once
However, one can also use `cxiufoje` to define a `whenever command` to be executed whenever their triggering event reoccurs. That means the action is performed every time the provided expression changes evaluation from false to true.

\[ \text{Derivation 52.} \]

\[
\langle \text{whenever command} \rangle \rightarrow \langle \text{command} \rangle \text{ } 'cxiufoje' \text{ } \langle \text{expression} \rangle
\]

Using timed actions as part of a natural language raises some interesting issues concerning the way sequenced temporal actions should behave. For example, consider the following speech:

**Program 4.1 Two Sequenced Announcements**

1. `Anoncu citilo saluton malcitilo`  
   Announce "hello"
2. `je ses sekundoj`  
   after 6 seconds
3. `poste anoncu citilo gxis malcitilo`  
   afterwards announce "bye"
4. `je kvar sekundoj`  
   after 4 seconds

Should “bye” be said four seconds after “hello” or two seconds before it?

**Program 4.2 Light State Announcement**

1. `Anoncu citilo sxalta malcitilo`  
   Announce “turned on”
2. `cxiufoje la lumo sxaltas`  
   every time the light activates
3. `poste anoncu citilo malsxalta malcitilo.`  
   afterwards announce "closed".

Similarly, should the word “closed” be spoken after every time the light is turned on? Perhaps once right after the speech is parsed, or once after the light is first turned on? The answers to these questions dictate evaluation order, which in turn has significant implications on program behavior.

This question has an explosive effect when reoccurring commands are used. For example:

**Program 4.3 Explosive Announcement**

1. `Anoncu citilo sxalta malcitilo`  
   Announce “turned on”
2. `cxiufoje la lumo sxaltas`  
   every time the light activates
3. `poste anoncu citilo malsxalta malcitilo`  
   afterwards announce "turned off"
4. `cxiufoje la lumo malsxaltas.`  
   every time the light deactivates.

Deciding that a temporal command following a repeated execution command will be evaluated after each execution, means that the number of “turned off” announcements will grow linearly over time.

My choice is that these scheduling instructions apply only to the last serial command before them, and inhibit the serial commands to follow only once. Therefore, the result of Program 4.1 is that “hello” is announced first after six seconds from parsing, and only four seconds later, “bye” is announced. Similarly, for the speech presented in Program 4.2, the word “closed” is announced only once, and only after the first time the

---

49`cxiufoje`: every time
light has been turned on. This decision disarms Program 4.3 since the second repeated announcement is evaluated only once.

Furthermore, what happens when one delayed action is significantly longer than another? If both actions were timed consecutively (as in Program 4.1), then the second action is only evaluated after the first has expired, adding the execution time of the first action to the delay of the second. This delay might sound cumbersome, but it assures the serialized nature of the two statements. However, if two unrelated speeches postponed different actions, they should be able to interrupt each other. Thus a new unrelated action could be invoked even in the middle of another.

### 4.7.1 Temporal Closures

Assume we have a local variable, *sxatata temperaturo*\(^{50}\), and we want to set the desired temperature of a water heater to that value, tomorrow. There are two ways to go about this:

- Wait until tomorrow, then set the value of the water heater’s desired temperature at that time to *sxatata temperaturo*.
- Evaluate *sxatata temperaturo* at the moment of dictation, remember it, and set that value to the water heater the next day.

The problem with the first technique is that during the following day, the value of *sxatata temperaturo* may change; for example, from our favorite shower temperature to our favorite weather temperature. Domsaňo avoids such unexpected results by taking the second approach and creating a *closure*. A closure is a snapshot of the system memory at the moment of the command utterance. Domsaňo references this snapshot later when executing the delayed command. The result is that the delay mechanism postpones to later the command that the programmer thinks about *right now*, which I believe to be the most logical and desired outcome.

The downside of this approach is twofold:

- The naïve implementation of an agnostic closure requires a complete copy of the program state. Since closures are not the primary focus of this thesis, I take this approach to be good enough.

- My agnostic closure prevents the assignment of atomic types by deferred commands since variables are copied into the closure *by value*. Only the fields of appliances and complex types can be changed in this way since they are copied *by reference*.

Even with these downsides in mind, I felt that temporal closures were necessary to make temporal commands truly useful and predictable tools for changing program state over time.

\(^{50}\) *sxatata temperaturo*: favorite temperature
4.8 Computational Power

Being as simple as Domsaño is, one might fear that it is not Turing complete. Although I provide a formal proof in Appendix G, there is also a simple and intuitive explanation. Since a Domsaño program can assign values to variables, as seen in der. 15, is capable of conditional branching as described in der. 20 and is also capable of indefinite-length loops provided by der. 21—it is capable of Turing complete programming.
Chapter 5

An English Prototype

An assistant speaking only Esperanto is not useful enough for the general public. Since there are only several thousands of native Esperanto speakers worldwide, I tried to provided a more general solution. To this end, I propose a prototype assistant with the same capabilities as Domsaĝo, different only by its outward use of English in place of Esperanto. This prototype operates on top of the existing Domsaĝo interpreter.

As a first step, the assistant will preform the StT transformation of recorded audio. This component of the prototype can be provided by many different platforms, such as Google Cloud’s speech Application Programming Interface (API) \(^1\). Since English text is harder to tag than Esperanto, the next step is to scan the English text and annotate it to clue Domsaĝo about parts of speech, providing the same benefit as parsing Esperanto. The prototype then converts these tagged tokens to Esperanto-like text and feeds it to Domsaĝo’s interpreter. The interpreter, in turn, executes the given commands. That concludes the entire process from English speech to machine execution. The result is an approximation of my Holy Grail—an English speaking teachable assistant.

I tested this process with the programs provided throughout this thesis, and a collection of single commands.

5.1 First Approach: Singular Tagging

My first attempt at implementing the prototype used python Natural Language Tool Kit (NLTK [9]) to tag the parts of speech for each word.

I have encountered the following setbacks:

- Unfortunately, NLTK provides only a possible tag per-word, and many words were not tagged correctly. Moreover, some distinctions exist in Esperanto but not in English; imperative and infinitive cases of verbs are both written as the “base form” in English, for example—making them impossible to tell apart. These ambiguities prevented the Domsaĝo parser from understanding the programs.

\(^1\)https://cloud.google.com/speech-to-text
• When using the context of the entire speech for parsing, the prototype achieved an accuracy of only 66.52%; this is mostly due to incorrect tenses in verb recognition.

• Later parts of the pipeline had no means of recovering from parsing mistakes made by NLTK, which meant only the simplest programs could be executed.

5.2 Second Approach: Ambiguous Tags

In this attempt, I embraced the ambiguity of the English language as part of the preprocessor’s challenge. Thus instead of matching each word with a single guessed tag, the tagger attached all possible tags to each word to be sorted out later by a syntactic ambiguity resolver. This method ensured that every word contained the right tag within its tag-set. The ambiguity resolver then iterates the different interpretation combinations for all tokens.

The resolver then translates each possible tagging into a pseudo-Esperanto script and passes it to the Domsaľo interpreter. It then keeps only the interpretations compatible with Domsaľo’s syntax.

I have observed the following issues:

• The ratio of incorrect to correct tags is about 2:1, often a given program would have hundreds of interpretations, only one of which is entirely valid.
• Given the number of possible interpretations for each token, the total number of possible interpretations becomes huge even for short programs. Iterating brute-force over all possible options would require a runtime of $O(2^w)$ where $w$ is the number of words in a given program. For example, even Program 2.1 in itself could be tagged 2,654,208 different ways, and for Program G.1, this number goes beyond $10^{27}$.

Speeding up the search requires more information about each disqualified interpretation. Every failed parsing for a pseudo-Esperanto translation can be pinpointed to a single offending token, which violates Domsaĝo’s grammar. The ambiguity resolver could use the offending token’s location to narrow down the search space during runtime—if provided by the Domsaĝo interpreter.

### 5.3 Third Approach: Part of Speech Pruning

To manage the interpretation search space, the ambiguity resolver constructs a tagging tree in which each node is a single possible interpretation for a word. Each layer of this tagging tree contains all possible tags of the word, and each path from the root to a leaf provides a single possible interpretation for the program text. For example, the initial interpretation tree created for Program G.1 is given in Fig. 5.2. The only correct implementation is marked in green. However, Fig. 5.2 has 53 other paths to disqualify.

Whenever a possible tag combination is disqualified, Domsaĝo’s interpreter returns the specific token failing its parsing process, and the ambiguity resolver prunes the entire sub-tree of that tag. For example, when trying the first interpretation of Program G.1 (marked in blue), Domsaĝo’s interpreter signals that no grammar rule can accommodate the interpretation of “means” as a noun. That invalidates an entire sub-tree of 9 interpretations, leaving us with Fig. 5.3.

By leveraging Domsaĝo’s grammar in this manner, the ambiguity resolver quickly narrows down the number of interpretations.

However, the tree’s construction requires duplicate nodes for each interpretation of each word along its path. This duplication leads us to a $O(2^w)$ memory complexity, where $W$ is the number of words in a given program. Which, in turn, caused the longer examples presented in this thesis to be impossible to build.

### 5.4 Optimizations

In order to reduce the amount of memory required, the ambiguity resolver now constructs its tag-trees with shared nodes at each level, using copy-on-write during the pruning phase to maintain the minimum amount of duplicates. This sharing mechanism dramatically reduces the parse and resolution time of each program. It also reduces the amount of memory required for the interpretation trees to $O(W)$. Thus I have arrived at the process described at Fig. 5.4 and left the following issues for future work:
- Word order does not always match between Esperanto and English. Therefore the new interpreter could not handle negated size relations, possessive apostrophes, and similar deviations from Domšagó’s grammatical structure.

- Even after the syntactic ambiguity is resolved, a certain *semantic ambiguity* remains to be sorted out. Specifically, many of Domšagó’s reserved words are imperative verbs and are used in a way that would suit both interpretations for the token.

**Figure 5.2** Initially Ambiguous Parsing Tree of Constant Function

```
"constantify":infinitive

"numbers":noun

"means":noun

"return":noun  "0":number  "end":imperative

"end":end

"end":noun

"return":imperative  "0":number  "end":imperative

"end":end

"end":noun

"return":keyword(return)  "0":number  "end":keyword(end)

"end":end

"return":keyword(return)

"means":imperative ...

"numbers":imperative ...
```

**Figure 5.3** Pruned Ambiguous Parsing Tree of Constant Function

```
"constantify":infinitive

"numbers":noun

"means":noun

"return":noun  "0":number  "end":imperative

"end":end

"end":noun

"return":imperative  "0":number  "end":imperative

"end":end

"end":noun

"return":keyword(return)  "0":number  "end":keyword(end)

"end":end

"return":keyword(return)

"means":imperative ...

"numbers":imperative ...
```

38
Figure 5.4 English information processing via Domsaño

- English speech
  - speech to text engine
    - ambiguously tagged tokens
    - English text
      - part of speech tagger
        - tagged tokens
          - ambiguity resolver
          - esperantifier
            - Esperanto-like text
              - Domsaño interpreter
                - machine instructions
Chapter 6

Conclusions and Further Work

6.1 Conclusions

The main contribution of this thesis is a practical look at vocal programming in a spoken language. The programming language is unambiguous, Turing complete, and useful for IoT management and personal assistant programming. As personal digital assistants become ubiquitous and advance to the next generation [33], I hold that such abilities should be among their most essential features.

I based this programming language on Esperanto-a constructed language-since its grammar is unambiguous and easily parsable by design. This ease of parsing solved two of the most pressing issues of modern assistants: understanding user speech and figuring the intended result. However, an implementation concerning only Esperanto would still be impractical to most people. For this reason, I implemented a prototype English version that wrestles with the language’s inherent ambiguity. However, only time will tell which approach will best deal with these linguistic shortcomings in the real world.

I also briefly explored the use of temporal commands in this programming language. Though currently not an intrinsic part of most languages, temporal commands provide a fundamental tool when concerned with home automation and day-to-day routines. I feel that the given mechanisms, though crude, are robust enough for most purposes.

Esperanto is not a commonly spoken language. That makes Domsaţo’s chances of success as an actual programming language rather slim. However, I feel that this should not take away from the message of this work. Much like the digital assistant M developed by Facebook, some projects are useful not in themselves, but for the lessons learned in their making. [35]
6.2 Further Work

6.2.1 Managing Conversation Context

Much of the information in human communication is contextual. In any natural language interface, the management and leverage of context for machine understanding would be hugely beneficial; however, it goes against the modern practice of functional programming and stateless software. In this research, although Domṣaṇgo had a state composed of a scheduler and a smart home (all appliances included), I tried to keep its state as minimal as possible. I have introduced the keyword $gx1$ to refer to the last computed value contextually, but have halted my efforts at this point for simplicity’s sake. Despite this, one should expect later attempts to manage a more elaborate context construct, which will allow the interpreter to recognize what different pronouns uttered by the user might reference.

6.2.2 Other Constructed Languages

The logical languages Loglan and Lojban were both mentioned during this research. These languages were, for a long time, considered optimal for human-computer communication. Their high regularity and inherent strictness ensure they would be suitable for computational interpretation and execution even when scaled to long sentences. Like different textual programming languages offer different features that adhere to the programmer community’s varying needs, the logical languages could provide a more expressive vocal programming option.

6.2.3 International Phonetic Interpreter

The main hurdle in discussing Esperanto’s implementation as a spoken programming language was the lack of an StT engine capable of recognizing it in speech. For ease of use and implementation, modern StTs are coupled with their target language of interpretation. However, an StT tool that could convert speech to characters from the international phonetic alphabet would enable the construction of arbitrary languages for computer communication according to the same concepts applied here.

Use of the international phonetic alphabet as a standard tool for speech analysis and generation would also help to resolve ambiguities related to pronunciation, i.e., differentiating the word “read” between past (“I have read this”) and other tenses (“I will read this later”), or the word live which can either be an adjective (“a live show”) or a present verb (“I live here”).

The categorization of speech stops and tonalities is also crucial, such as differentiating a half stop meant to convey a comma from one meant to convey a colon. Although conveniently ignored so far (including in this paper), speech-stops sometimes significantly impact sentence meaning. For example, read the following sentence with and without the Oxford comma: “I love my parents, Kylie Minogue, and Kermit the frog.”
6.2.4 Lingua Computica

This thesis has only dealt with existing constructed languages and the benefits they might have on human-computer vocal interaction, showing that keeping both computer and human in mind results in a simple yet expressive instruction language. I decided to use Esperanto, but it is not a perfect language. Indeed, the language Ido already includes several improvements to its grammar. I suggest that designing a language specifically for this purpose could create a human-computer equivalent of a “lingua franca”—a universal bridge language between humans and machines, clearly understood and useful for both parties.

6.3 Discussion

In programming as in court, “littera est lex”¹, and clarity is necessary for proper function. However, prosaic speech and text are usually ambiguous by nature, so the correct meaning must be “divined” by the recipient. In this thesis, I tried to handle ambiguity in text and speech to effectively bring speech into the realm of programming.

The product described in this work is not complete since it lacks an StT engine to make a fully operational unit. However, this work calls out the need for a scripting language for personal assistant programming, similar to the concept of a terminal shell (such as the C shell or Bourne-again shell). This language allows the user to string together known commands and manipulate their outputs and execution times.

In Domsaĝo, I created programmatic constructs for function definitions, conditional execution, and loops, like all standard programming languages, and added a layer of temporal manipulation to top. I have created a simple parallelization mechanism for atomic commands as well as a delay mechanism for exact time intervals or specific points in time. I have described the keyword je, which functions in a way similar to the Bash program at by delaying subroutine execution to a later time. This timing mechanism enables the user to delegate deferred tasks with ease. Another important feature is the cxin modifier. This modification allows the user to schedule a repeating action at a known interval indefinitely, similar to the crontab command in Bash. One notable difference is that scripting languages allow transfers of data between different commands using variables and the pipe operation (marked with |), whereas Domsaĝo allows only variable use and access to the last command result via the keyword gxi².

Since this thesis dealt with scripting digital assistants and smart homes, it contained many examples for the manipulation of electric appliances, but little explanation about how to extend these abilities. This thesis sought only to simplify the construction of useful routines from existing building blocks. The difficulty of developing an entirely new assistant skill, however, remains as it was.

¹the letter is the law
²gxi: it
I do not anticipate a future need for a pipeline mechanism, but think it might be convenient to provide syntactic sugar for other programmatic structures such as $A() \& \& B()$ used in Bash to execute program $B$ only if $A$ terminated successfully, as well as $A() \mid\mid B()$ used to execute $B$ only if $A$ failed. These structures will grant the user more flexibility in handling failed assistant actions and allow more elaborate scripts with fallback plans. This suggestion is not to say that such mechanisms are impossible now (let us remember that Domsaĝo is Turing complete), but still require the user to maintain proper return values from all commands and scripts.

I have dealt with the challenge of precise grammatical parsing, and shown that its core issues could be overcome in Esperanto using a simple LALR parser. My choice of Esperanto over other constructed or natural languages is not coincidental, as discussed in Section 1.3. Esperanto is neither the most logical constructed language; nor does it have the most straightforward grammar (indeed, many languages based on Esperanto are even more basic). However, being the most well known constructed language, having survived two world wars, and being the only constructed language boasting a crowd of native speakers—Esperanto is a good compromise between simplicity and character. The features that have proven useful for this work were:

- Esperanto is a designed schematic language aimed for ease of learning (rather than sounding more naturalistic). This core virtue is what allows for its easy parsing.

- Unambiguous part of speech recognition by suffix (adjectives, nouns, and verbs) is natural in Esperanto, as explained in Section 4.2.

- Accurate verb-tense identification used to identify function definitions, function invocations, and state queries.

- Word conversion between parts of speech was useful in array indexing and access, as explained in Section 4.6.

- Invariable letter-to-phoneme correspondence is another crucial feature of the language. In English speech, for example, it is impossible to tell “hear” apart from “here” without additional context, and a bow (at the waist) is spelled the same as a bow (used for archery) even though they sound different. This problem does not exist in Esperanto.

- Number parsing is exceedingly hard in Esperanto, where all numbers follow a simple scheme. Consider on the other hand the complex rules needed to parse the number “quatre-vingt douze” (which is ninety-two, but literally means “four-twenties twelve”)

- Time units are also unambiguous in Esperanto, unlike the division of minutes (with the word minute meaning either a time unit or a small amount) and seconds (either a time unit or a place in some order).
Some of Esperanto’s disadvantages were also felt during this work:

- Esperanto’s alphabet includes letters with diacritics (ĉ, ĝ, ĥ, ĵ, š, ŭ), which require acquaintance.

- The default gender for any word in Esperanto is male unless the female suffix ‘-ino’ is used. The absence of a male suffix should not perhaps be considered a significant drawback, but in a planned language such as Esperanto, making the default gender-neutral would be an easy solution to implement.

- Esperanto draws its many consonants from several different languages. This diversity essentially makes it incompatible with any other language. The sound of ĥ, for example, might come naturally to Hebrew speakers, but not to English speakers, and similarly, the Japanese language has no distinction between l and r. This distinction (or lack thereof) makes Esperanto’s phonology less accessible to Japanese native speakers.

I also did not elaborate on any method to edit any repeating command or predefined function. For this purpose, some audible development environment must be created.
Bibliography


[51] Say “Privet” to Alice, Yandex’s Intelligent Assistant. URL: https://yandex.com/company/blog/say-privet-to-alice-yandex-s-intelligent-assistant/.


[53] Silly name generator for Google assistant. URL: https://github.com/actions-on-google/dialogflow-silly-name-maker.


[63] Benjamin Lee Whorf. Linguistics as an exact science. Massachusetts Institute of Tech., 1940.

Keyword Index

al, 24
alie, 22
ambaux, 25
aux, 24

citilo, 29
cxiu, 31
cxiufoje, 32
de, 30
dekstra citilo, 29
dekstra krampo, 23
dum, 22
egala, 24
fojoj, 23
gran, 24
gx, 19
inter, 19
je, 31
kaj, 19

malcitilo, 29
maldekstra citilo, 29
maldekstra krampo, 23
malgranda, 24
malpli, 23
malvero, 24
ne, 24
ol, 24
partoj, 23
pli, 24
poste, 20
samtempe, 20
se, 22
tiam, 22
trans, 19
unufoje, 31
vero, 24
Grammar Rules Index

adjective, 19
arguments, 21
array access, 29
assignable, 21
assignment, 21
atomic command, 20

compound command, 21
definition, 30
digits sequence, 26

expression, 24, 25
factor, 23
field access, 29
fraction literal, 27

if command, 22
imperative verb, 19
infinitive verb, 19
invocation, 21
it, 20

name, 29
name expression, 29
noun, 19
number literal, 26, 27
numerator, 27

once command, 31
one digit literal, 26
other digit, 26

parallel command, 20
parameters, 30
present verb, 19

relation, 24
repeating command, 31
return, 22
round hour, 27

scheduled command, 31
sentence, 20
separator, 19
serial command, 20
speech, 20
string, 29

term, 23
three digit literal, 26
time point, 27
time span, 28
time unit, 28
two digit literal, 26

whenever command, 32
while command, 22
Appendices
Appendix A

Esperanto

Esperanto is a constructed language created by Doctor Ludwik Lejzer Zamenhof (1859 - 1917). It was created as a simple auxiliary language to facilitate communication between different people. It draws its grammatical structure and vocabulary from the natural European languages.

In 1887, Zamenhof published his first book about Esperanto: “International language: Introduction and complete textbook” in Russian. He signed the book under the pseudonym “Doktoro Esperanto” (Dr. Hopeful), which later lent its name to the language itself. Since then, the number of speakers grew around Europe, America, China, and Japan. Nowadays, it is estimated that the language has over 2 million speakers worldwide, about a thousand of which are native speakers.

A.1 Alphabet and Pronunciation

The Esperanto alphabet is based on Latin script and an addition of six letters with diacritics. Each letter denotes only one sound.

The letters of Esperanto are:

a, b, c, ĉ, d, e, f, g, ĝ, h, ĥ, i, j, ĵ, k, l, m, n, o, p, r, s, ŝ, t, u, ŭ, v, z

The above-accented letters: ĉ, ĝ, ĥ, ĵ, ŝ and ŭ are not coded by the ASCII format, and not covered by the standard keyboard. In order to deal with this issue, the international Esperanto-speaking community developed several ways of writing:

H system notation describes these letters by writing the letter h after any letter that needs to become accented except for ŭ which is written as a plain u. This system was proposed by LL Zamenhof and is considered as the official way to write Esperanto.

X system notation uses x instead of h to note accented letters, and also uses it to differentiate ŭ from u. This system causes less ambiguity in the text since x is not a part of the Esperanto alphabet.
Other variants exist, such as using a caret or a greater-sign to denote an accented character. This way, the letter ŝ could be written as either: `s, s^, or s>.

This thesis uses the x system to write all Esperanto text given as Domsaĝo code. Esperanto uses a “one sound per letter” rule, so each letter is pronounced as follows:

- **Vowels** - Esperanto follows the five-vowels system
  
  a like a in father and not like a in fake.
  
  e like e in bend.
  
  i like i in wish or sing, and not like i in might.
  
  o like o in horse, without the short oo accompanied to it in note or boat.
  
  u like oo in boot and never like u in mute or but.

- **Consonants** - consonants that do not appear here follow standard English pronunciation
  
  c is the voiceless alveolar sibilant affricate as the ts in hats
  
  ĉ is the palato-alveolar sibilant affricate as ch in church
  
  g is the voiced velar stop as in go or give, but never as in gem or allege
  
  ĝ is the voiced post-alveolar affricate as in gem or jacket
  
  ĥ is the voiceless velar fricative as the ch in the Scottish word loch
  
  j is the palatal approximant, like the y in you
  
  ĵ is the voiced post alveolar fricative as the s is vision or pleasure
  
  ŝ is the voiceless post alveolar fricative as the sh in she or share
  
  ŭ is the voiced labio-velar approximant as the w in well

- **Diphthongs** - these letter inter-plays are worth mentioning
  
  aj like the i in fire or the y in fly.
  
  aŭ like the ow in owl or cow.
  
  ej like the ey in hey or the ay in way.
  
  eŭ is not used in modern English, but is like the eu in the Greek saying “deus ex machina”.
  
  oj like the oy in boy.
  
  oŭ like the o in dome or chrome.
  
  uj like the ui in ruin.
A.2 Esperanto Grammar

Esperanto has eight different word classes: noun, verb, adjective, adverb, pronouns, prepositions, conjunctions, and determinant. The first five stand out as being large, dynamically built classes - words within these categories are made by adding a specific ending to a word stem. The last three categories are different by being closed sets of pre-built static words. The overlap between these two categories is the adverb class, which includes both a predefined set of words and an ability to extend it via the word stem and ending system.

I will not go far into detail in the scope of this work, but rather provide the information relevant to each word class used in this thesis. Further reading into Esperanto is easy to find across the internet and is always recommended.

- Nouns are created by adding the ending \( o \) to a word stem. This suffix creates a singular nominative noun, which could be turned from singular to plural form by further adding the ending \( j \), and turned from nominative to accusative by the ending \( n \)—which will be explained shortly. The benefit of this rule is the easy recognition of nouns by the regular expression: \([a-z]+o(j?)\).

- Adjectives are created by adding the ending \( a \) to a word stem. This suffix results in a singular nominative adjective, which could be transformed by \( j \) and \( n \) as described for nouns. Adjectives turn to the plural and accusative forms per their described noun. A few examples would be:
  Bela pomo - a beautiful apple.
  Belaj pomoj - beautiful apples.
Therefore adjectives are captured by the regular expression: \([a-z]+a(j?)\).

- Verb endings in Esperanto are determined only by tenses instead of number and gender. All forms of verbs are presented here for wholeness’s sake, but in practice, only the indicative is used most of the time, and in my work the imperative verbs are the most useful. See Tables A.1 and A.2 for details. One can thus identify an imperative verb by the regular expression \([a-z]+u\)

- In Esperanto, unlike English, a special suffix is dedicated to each sentence’s direct object. This suffix is called an accusative case and is simply the letter \( n \). The

<table>
<thead>
<tr>
<th>Form</th>
<th>Past tense</th>
<th>Present tense</th>
<th>Future tense</th>
</tr>
</thead>
<tbody>
<tr>
<td>indicative</td>
<td>is</td>
<td>as</td>
<td>os</td>
</tr>
<tr>
<td>active participle</td>
<td>inta</td>
<td>anta</td>
<td>onta</td>
</tr>
<tr>
<td>passive participle</td>
<td>ita</td>
<td>ata</td>
<td>ota</td>
</tr>
<tr>
<td>active gerund</td>
<td>inte</td>
<td>ante</td>
<td>onte</td>
</tr>
<tr>
<td>passive gerund</td>
<td>ite</td>
<td>ate</td>
<td>ote</td>
</tr>
</tbody>
</table>

Table A.1: Verb time and word endings in Esperanto
<table>
<thead>
<tr>
<th>Form</th>
<th>Word ending</th>
</tr>
</thead>
<tbody>
<tr>
<td>infinitive</td>
<td>i</td>
</tr>
<tr>
<td>imperative</td>
<td>u</td>
</tr>
<tr>
<td>conditional</td>
<td>us</td>
</tr>
</tbody>
</table>

Table A.2: Verb modes and word endings in Esperanto

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Address</th>
<th>Pronoun</th>
</tr>
</thead>
<tbody>
<tr>
<td>singular</td>
<td>1st person</td>
<td>mi</td>
</tr>
<tr>
<td></td>
<td>2nd person</td>
<td>ci</td>
</tr>
<tr>
<td></td>
<td>3rd person</td>
<td>li</td>
</tr>
<tr>
<td>plural</td>
<td>1st person</td>
<td>ni</td>
</tr>
<tr>
<td></td>
<td>2nd person</td>
<td>vi</td>
</tr>
<tr>
<td></td>
<td>3rd person</td>
<td>ili</td>
</tr>
</tbody>
</table>

Table A.3: The pronouns used in Esperanto

Accusative case affects both the noun, which is the direct object, and all related adjectives. In Hebrew, its parallel would be the word “et.” Here are a few examples:

Mi manĝis belan pomon - I ate a beautiful apple.
Mi manĝis belajn pomojn - I ate beautiful apples.

So when taking the accusative case into account, one can identify nouns and adjectives with the following regular expressions:

For nouns: \[a-z]+o(j?)(n?)\]
And for adjectives: \[a-z]+a(j?)(n?)\]

These patterns are the ones used in ders. 1 and 2

- Adverbs are made of a dynamically built group of words ending with \(e\), and several special groups of adverbs.

- Pronouns are a closed set of words that do not follow the noun rule. As such, they can be identified out of a finite list with no need for a regular expression. It might be worth noting that the singular-second person pronoun ci is seldom used in Esperanto and is only listed in Table A.3 for the sake of completeness. In actuality, vi is used for both singular and plural second person.
Appendix B

Example Google Assistant Application

The following is a single file out of the code for a silly name generator - a tutorial application for the Google assistant. In order to understand it, it would be enough to read lines 21,35, and 49. Compare it to Program 2.1

```json
{
  "id": "01c9e176-80f8-4f63-a661-9efb715f847e",
  "name": "make_name",
  "auto": true,
  "contexts": [],
  "responses": [
    {
      "resetContexts": false,
      "action": "make_name",
      "affectedContexts": [],
      "parameters": [
        {
          "id": "77ebeac8-6bf2-4871-af4e-811d7e921d30",
          "required": true,
          "dataType": "@sys.number",
          "name": "number",
          "value": "$number",
          "prompts": [
            {
              "lang": "en",
              "value": "What is your lucky number?"
            }
          ],
          "isList": false
        },
        {
          "id": "eca3de68-2529-4e56-bda3-97a8a702253a",
          "required": true,
          "dataType": "@sys.color",
          "name": "color",
          "value": "$color",
          "prompts": [
            {
              "lang": "en",
              "value": "What is your favorite color?"
            },
            {
              "lang": "en",
              "value": "What color is your favorite?"
            }
          ],
          "isList": false
        }
      ]
    }
  ]
}
```
"messages": [
{
  "type": 0,
  "lang": "en",
  "speech": "Alright, your silly name is $color $number. I hope you like it! See you next time!"
}
],
"defaultResponsePlatforms": {},
"speech": []}
Appendix C

Example Voice Coding with Talon

The code segment appearing in Program C.1 is a partial transcript from a voice coding demonstration video. 1 The tool used for voice coding is Talon, as explained in Chapter 3. Youtube’s automatic subtitle feature generated the video’s transcript, which I trimmed to leave only the relevant part of the first half—it is long enough on its own. I added syntax highlighting according to Talon’s keyword collection. 2 Provided on the right is the resulting Python code, along with extra spaces to align speech to resulting code. The python code is aligned to match the verbal code (approximately line to paragraph). The entire demo transcript contains 996 words, for 49 lines of python code.

---

1 https://www.youtube.com/watch?v=ddFl63dgpaI
2 https://github.com/talonvoice/examples/blob/master/basic_keys.py
Program C.1 Excerpt from Talon programming demo

```python
import atexit
import logging
import os
import threading
import logging
import atexit
CMD_PATH = ('ipc://' + os.path.join(TALON_HOME, '.sys', 'dc_cmd.sock'))
CMD_PATH = ('ipc://' + os.path.join(TALON_HOME, '.sys', 'dc_pub.sock'))

from talon_init import
from .api import
from .dispatch import
from . import
import os
import logging

def _engine_cb(handle, topic, buf, size):
    try:
        buf = bytes(ffi.buffer(buf, size))
        topic = ffi.string(topic).decode('utf8')
        except Exception:
            logging.exception("engine dispatch err on ":”)
            topic = ffi.string(topic).decode('utf8')
            Engine.engines(handle)._on_msg(topic, buf)
```
Appendix D

Reserved Words

Domsago leverages the special word-groups of the Esperanto language to enhance its grammar beyond mere PoS recognition. The easiest word group to recognize is the group of prepositions since it is a finite one. Domsago does not identify most of these words specifically but uses any word of this group as a separator between command arguments, as shown in der. 3. Table D.3 lists all of these prepositions.

Another Domsago feature discussed in Section 4.5.1 is its ability to identify numbers. The reserved words used for this specific context are shown in Table D.1. To create time span and time point literals described in Section 4.5.3, Domsago uses the reserved words mentioned in Table D.2. All other reserved words are detailed in Table D.4.

<table>
<thead>
<tr>
<th>Number</th>
<th>Esperanto Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>nul</td>
</tr>
<tr>
<td>1</td>
<td>unu</td>
</tr>
<tr>
<td>2</td>
<td>du</td>
</tr>
<tr>
<td>3</td>
<td>tri</td>
</tr>
<tr>
<td>4</td>
<td>kvar</td>
</tr>
<tr>
<td>5</td>
<td>kvin</td>
</tr>
<tr>
<td>6</td>
<td>ses</td>
</tr>
<tr>
<td>7</td>
<td>sep</td>
</tr>
<tr>
<td>8</td>
<td>ok</td>
</tr>
<tr>
<td>9</td>
<td>naŭ</td>
</tr>
<tr>
<td>10</td>
<td>dek</td>
</tr>
<tr>
<td>100</td>
<td>cent</td>
</tr>
<tr>
<td>1,000</td>
<td>mil</td>
</tr>
<tr>
<td>fraction suffix</td>
<td>*ono</td>
</tr>
</tbody>
</table>

Table D.1: Words Reserved for Numbers in Domsago

<table>
<thead>
<tr>
<th>Time Unit</th>
<th>Esperanto Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second</td>
<td>sekundo</td>
</tr>
<tr>
<td>Minute</td>
<td>minuto</td>
</tr>
<tr>
<td>Hour</td>
<td>horo</td>
</tr>
<tr>
<td>Day</td>
<td>tago</td>
</tr>
<tr>
<td>Week</td>
<td>semajno</td>
</tr>
<tr>
<td>Month</td>
<td>monato</td>
</tr>
<tr>
<td>Year</td>
<td>jaro</td>
</tr>
</tbody>
</table>

Table D.2: Words Reserved for Time Literals in Domsago
<table>
<thead>
<tr>
<th>Preposition</th>
<th>Translation</th>
<th>Esperanto Example</th>
<th>Example Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al</td>
<td>To</td>
<td>Parolu al mi</td>
<td>Talk to me</td>
</tr>
<tr>
<td>Anstataŭ</td>
<td>Instead of</td>
<td>Štalo anstataŭ ŝtono</td>
<td>Steel instead of stone</td>
</tr>
<tr>
<td>Antaŭ</td>
<td>Before</td>
<td>Iru antaŭ mi</td>
<td>Go before me</td>
</tr>
<tr>
<td>Apud</td>
<td>Next to</td>
<td>Atendu apud la domo</td>
<td>Wait next to the house</td>
</tr>
<tr>
<td>Ĉe</td>
<td>At</td>
<td>Renkontu min ĉe la pordo</td>
<td>Meet me at the door</td>
</tr>
<tr>
<td>Ĉirkaŭ</td>
<td>Around</td>
<td>Vojaĝi ĉirkaŭ la mondo</td>
<td>Travel around the world</td>
</tr>
<tr>
<td>De</td>
<td>Of</td>
<td>La fino de la linio</td>
<td>The end of the line</td>
</tr>
<tr>
<td>Dum</td>
<td>While</td>
<td>Atendu dum mi preparas</td>
<td>Wait while I prepare</td>
</tr>
<tr>
<td>Ekde</td>
<td>Since</td>
<td>Ekde kiam?</td>
<td>Since when?</td>
</tr>
<tr>
<td>Ekster</td>
<td>Outside</td>
<td>Li estas ekster mia domo</td>
<td>He is outside my house</td>
</tr>
<tr>
<td>El</td>
<td>Out of, made of</td>
<td>Martelo el ŝtono</td>
<td>Hammer of stone</td>
</tr>
<tr>
<td>En</td>
<td>In</td>
<td>Mi estas en pozicio</td>
<td>I am in position</td>
</tr>
<tr>
<td>Ĝis</td>
<td>Until</td>
<td>Ĝis la revido</td>
<td>Until we meet again</td>
</tr>
<tr>
<td>Inter</td>
<td>Between</td>
<td>Li marŝas inter la arboj</td>
<td>He walks between the trees</td>
</tr>
<tr>
<td>Kontraŭ</td>
<td>Against</td>
<td>Frato kontraŭ frato</td>
<td>Brother against brother</td>
</tr>
<tr>
<td>Krom</td>
<td>Except</td>
<td>Mi amas ĉiujn krom li</td>
<td>I love everyone except him</td>
</tr>
<tr>
<td>Kun</td>
<td>With</td>
<td>Promenu kun dio</td>
<td>Walk with god</td>
</tr>
<tr>
<td>Laŭ</td>
<td>According to</td>
<td>Laŭ ĉi tio, vi eraras</td>
<td>According to this, you are mistaken</td>
</tr>
<tr>
<td>Malgraŭ</td>
<td>Despite</td>
<td>Li silentis malgraŭ lia kolero</td>
<td>He kept silent despite his anger</td>
</tr>
<tr>
<td>Ol</td>
<td>Than</td>
<td>Ĝi estas pli granda ol ni</td>
<td>This is bigger than us</td>
</tr>
<tr>
<td>Per</td>
<td>With</td>
<td>Tranĉu ĝin per la tranĉilo</td>
<td>Cut it with the knife</td>
</tr>
<tr>
<td>Post</td>
<td>After</td>
<td>Faru ĝin post la tagmanĝo</td>
<td>Do it after lunch</td>
</tr>
<tr>
<td>Pri</td>
<td>About</td>
<td>Libro pri amo</td>
<td>A book about love</td>
</tr>
<tr>
<td>Pro</td>
<td>Because</td>
<td>Li estas laca pro lia infano</td>
<td>He is tired because of his child</td>
</tr>
<tr>
<td>Sen</td>
<td>Without</td>
<td>Mi ne povas vivi sen kafon</td>
<td>I can’t live without coffee</td>
</tr>
<tr>
<td>Sub</td>
<td>Under</td>
<td>Slosilo sub la tapiŝo</td>
<td>A key under the rug</td>
</tr>
<tr>
<td>Super</td>
<td>Over</td>
<td>Li iris super la ponto</td>
<td>He went over the bridge</td>
</tr>
<tr>
<td>Sur</td>
<td>On</td>
<td>Kato sur la arbo</td>
<td>A cat on the tree</td>
</tr>
<tr>
<td>Trans</td>
<td>Across</td>
<td>Iru trans la straton</td>
<td>Go across the street</td>
</tr>
</tbody>
</table>

Table D.3: Esperanto Prepositions
<table>
<thead>
<tr>
<th>Reserved Word(s)</th>
<th>Meaning</th>
<th>Usage Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambaŭ</td>
<td>Both</td>
<td>der. 27</td>
</tr>
<tr>
<td>Asignu al</td>
<td>Assign to</td>
<td>der. 15</td>
</tr>
<tr>
<td>Aŭ</td>
<td>Or</td>
<td>der. 27</td>
</tr>
<tr>
<td>Ĉiu</td>
<td>Every</td>
<td>der. 50</td>
</tr>
<tr>
<td>Ĉufoje</td>
<td>Whenever</td>
<td>der. 52</td>
</tr>
<tr>
<td>De</td>
<td>Of</td>
<td>ders. 44 and 45</td>
</tr>
<tr>
<td>Citilo ... malcitilo</td>
<td>Quote ... unquote</td>
<td>der. 42</td>
</tr>
<tr>
<td>Dum faru finu</td>
<td>During do end</td>
<td>der. 21</td>
</tr>
<tr>
<td>Estas egala al</td>
<td>Is equal to</td>
<td>der. 26</td>
</tr>
<tr>
<td>Estas pli granda ol</td>
<td>Greater than</td>
<td>der. 26</td>
</tr>
<tr>
<td>Estas pli malgranda ol</td>
<td>is Smaller than</td>
<td>der. 26</td>
</tr>
<tr>
<td>Fojoj</td>
<td>Times</td>
<td>der. 23</td>
</tr>
<tr>
<td>Ĝi</td>
<td>It</td>
<td>der. 19</td>
</tr>
<tr>
<td>Je</td>
<td>At</td>
<td>der. 49</td>
</tr>
<tr>
<td>Kaj</td>
<td>And</td>
<td>der. 3</td>
</tr>
<tr>
<td>La</td>
<td>The</td>
<td>der. 46</td>
</tr>
<tr>
<td>Malpli</td>
<td>Less</td>
<td>der. 24</td>
</tr>
<tr>
<td>Malvero</td>
<td>False</td>
<td>der. 27</td>
</tr>
<tr>
<td>Ne</td>
<td>Not</td>
<td>der. 27</td>
</tr>
<tr>
<td>Nenio</td>
<td>None</td>
<td>Section 4.5.5</td>
</tr>
<tr>
<td>Partoj</td>
<td>Parts</td>
<td>der. 23</td>
</tr>
<tr>
<td>Pli</td>
<td>More</td>
<td>der. 24</td>
</tr>
<tr>
<td>Poste</td>
<td>Afterwards</td>
<td>der. 10</td>
</tr>
<tr>
<td>Revemu</td>
<td>Return</td>
<td>der. 19</td>
</tr>
<tr>
<td>Samtempe</td>
<td>Simultaneously</td>
<td>der. 11</td>
</tr>
<tr>
<td>Se tiam alie finu</td>
<td>If then else end</td>
<td>der. 20</td>
</tr>
<tr>
<td>Signifas ... finu</td>
<td>Means ... end</td>
<td>der. 47</td>
</tr>
<tr>
<td>Unufoje</td>
<td>Once</td>
<td>der. 51</td>
</tr>
<tr>
<td>Vero</td>
<td>True</td>
<td>der. 27</td>
</tr>
</tbody>
</table>

Table D.4: Domsạgo Reserved Words and Their Meaning
Appendix E

Grammar Rules

der. 1

\[ \langle \text{noun} \rangle \rightarrow \{ \langle \text{letter} \rangle \} \text{'o'}[\text{'j'}][\text{'n'}] \]

der. 2

\[ \langle \text{adjective} \rangle \rightarrow \{ \langle \text{letter} \rangle \} \text{'a'}[\text{'j'}][\text{'n'}] \]

der. 3

\[ \langle \text{separator} \rangle \rightarrow \text{','} | \text{'kaj'} | \langle \text{preposition} \rangle \]

der. 4

\[ \langle \text{infinitive verb} \rangle \rightarrow \{ \langle \text{letter} \rangle \} \text{'i'} \]

der. 5

\[ \langle \text{imperative verb} \rangle \rightarrow \{ \langle \text{letter} \rangle \} \text{'u'} \]

der. 6

\[ \langle \text{present verb} \rangle \rightarrow \{ \langle \text{letter} \rangle \} \text{'as'} \]

der. 7

\[ \langle \text{it} \rangle \rightarrow \text{gxi}[\text{'n'}] \]
der. 8

\[ \langle \text{speech} \rangle \rightarrow \langle \text{sentence} \rangle \mid \langle \text{routine definition} \rangle \]

der. 9

\[ \langle \text{sentence} \rangle \rightarrow \langle \text{serial command} \rangle \ \{ \langle \text{poste} \rangle \ \langle \text{serial command} \rangle \} \]

der. 10

\[ \langle \text{serial command} \rangle \rightarrow \langle \text{parallel command} \rangle \ \{ \langle \text{samtempe} \rangle \ \langle \text{parallel command} \rangle \} \]

der. 11

\[ \langle \text{parallel command} \rangle \rightarrow \langle \text{atomic command} \rangle \ | \ \langle \text{compound command} \rangle \]

der. 12

\[ \langle \text{atomic command} \rangle \rightarrow \langle \text{assignment} \rangle \ | \ \langle \text{invocation} \rangle \ | \ \langle \text{return} \rangle \]

der. 13

\[ \langle \text{compound command} \rangle \rightarrow \]
\[ \quad \langle \text{if command} \rangle \]
\[ \quad \langle \text{while command} \rangle \]
\[ \quad \langle \text{for command} \rangle \]

der. 14 continues der. 13

\[ \langle \text{compound command} \rangle \rightarrow : \]
\[ \quad \langle \text{delayed command} \rangle \]
\[ \quad \langle \text{scheduled command} \rangle \]
\[ \quad \langle \text{repeating command} \rangle \]
\[ \quad \langle \text{once command} \rangle \]
\[ \quad \langle \text{whenever command} \rangle \]

der. 15

\[ \langle \text{assignment} \rangle \rightarrow \langle \text{asignu} \rangle \ \langle \text{expression} \rangle \ \langle \text{al} \rangle \ \langle \text{assignable} \rangle \]
der. 16

\langle assignable \rangle \rightarrow \langle name \ expression \rangle

der. 17

\langle invocation \rangle \rightarrow \langle imperative \ verb \rangle \ \langle arguments \rangle

der. 18

\langle arguments \rangle \rightarrow \langle(\langle expression \rangle \ {\langle separator \rangle \ \langle expression \rangle})\rangle\rangle

der. 19

\langle return \rangle \rightarrow \text{'revenu'} \ \langle(\langle expression \rangle)\rangle

der. 20

\langle if \ command \rangle \rightarrow \text{'se'} \ \langle expression \rangle \ \text{'tiam'} \ \langle sentence \rangle \ \text{['alie' \ \langle sentence \rangle]} \ \text{'finu'}

der. 21

\langle while \ command \rangle \rightarrow \text{'dum'} \ \langle expression \rangle \ \text{'tiam'} \ \langle sentence \rangle \ \text{'finu'}

der. 22

\langle factor \rangle \rightarrow

\text{'malpli'} \ \langle factor \rangle
\langle verbal \ number \rangle
\langle name \rangle

\text{'maldekstra \ krampo'} \ \langle expression \rangle \ \text{'dekstra \ krampo'}

der. 23

\langle term \rangle \rightarrow

\langle term \rangle \ \text{'fojoj'} \ \langle factor \rangle
\langle term \rangle \ \text{'partoj'} \ \langle factor \rangle
\langle factor \rangle
der. 24

\[
\text{⟨expression⟩ } \rightarrow \\
\text{ ⟨expression⟩ ‘pli’ ⟨term⟩} \\
\text{⟨expression⟩ ‘malpli’ ⟨term⟩} \\
\text{ ⟨term⟩}
\]

der. 25 continues der. 24

\[
\text{⟨expression⟩ } \rightarrow \\
\text{ :} \\
\text{ ⟨expression⟩ [‘ne’] ⟨relation⟩ ⟨expression⟩}
\]

der. 26

\[
\text{⟨relation⟩ } \rightarrow \\
\text{ ‘estas egala al’} \\
\text{ ‘estas pli granda ol’} \\
\text{ ‘estas pli granda aux egala al’} \\
\text{ ‘estas pli malgranda aux egala al’} \\
\text{ ‘estas pli malgranda ol’}
\]

der. 27 continues der. 24

\[
\text{⟨expression⟩ } \rightarrow \\
\text{ :} \\
\text{ ⟨expression⟩ ‘ambaux’ ⟨expression⟩} \\
\text{ ⟨expression⟩ ‘aux’ ⟨expression⟩} \\
\text{ ‘ne’ ⟨expression⟩} \\
\text{ ‘vero’} \\
\text{ ‘malvero’}
\]

der. 28 continues der. 24

\[
\text{⟨expression⟩ } \rightarrow \\
\text{ :} \\
\text{ ⟨name expression⟩ ⟨present verb⟩}
\]

der. 29

\[
\text{⟨number literal⟩ } \rightarrow \text{ ⟨digits sequence⟩}
\]
der. 30

\( \langle \text{digits sequence} \rangle \rightarrow \)

'null'

\( \langle \text{one digit literal} \rangle \)

\( \langle \text{two digit literal} \rangle \)

\( \langle \text{three digit literal} \rangle \)

:

der. 31

\( \langle \text{other digit} \rangle \rightarrow \) 'du' | 'tri' | 'kvar' | 'kvin' | 'ses' | 'sep' | 'ok' | 'naux'

der. 32

\( \langle \text{one digit literal} \rangle \rightarrow \) 'unu' | \( \langle \text{other digit} \rangle \)

der. 33

\( \langle \text{two digit literal} \rangle \rightarrow \) [\( \langle \text{other digit} \rangle \)] 'dek' [\( \langle \text{one digit literal} \rangle \)]

der. 34

\( \langle \text{three digit literal} \rangle \rightarrow \) [\( \langle \text{other digit} \rangle \)] 'cent' [\( \langle \text{two digit literal} \rangle \)]

der. 35 continues der. 29

\( \langle \text{number literal} \rangle \rightarrow \) :

\( \langle \text{fraction literal} \rangle \)

der. 36

\( \langle \text{fraction literal} \rangle \rightarrow \) [\( \langle \text{other digit} \rangle \)] 'ono'

der. 37

\( \langle \text{time point} \rangle \rightarrow \)

\( \langle \text{round hour} \rangle \)

\( \langle \text{round hour} \rangle \) 'kaj' \( \langle \text{natural literal} \rangle \) 'minutoj'

\( \langle \text{round hour} \rangle \) 'kaj' \( \langle \text{fraction literal} \rangle \)
der. 38

\langle round \ hour \rangle \rightarrow \langle la \rangle \langle \text{numerator} \rangle \langle horo \rangle

der. 39

\langle \text{numerator} \rangle \rightarrow \langle \text{natural literal} \rangle \langle a \rangle

der. 40

\langle \text{time unit} \rangle \rightarrow (\langle tago \rangle \mid \langle horo \rangle \mid \langle minuto \rangle \mid \langle sekundo \rangle)[j]

der. 41

\langle \text{time span} \rangle \rightarrow
\langle \text{natural literal} \rangle \langle \text{time unit} \rangle \{(\langle \text{delimiter} \rangle \langle \text{natural literal} \rangle \langle \text{time unit} \rangle)\}
\langle \text{natural literal} \rangle \langle \text{time unit} \rangle \langle kaj \rangle \langle \text{fraction literal} \rangle

der. 42

\langle \text{string} \rangle \rightarrow \langle \text{maldekstra citilo} \rangle \{\langle \text{letter} \rangle \mid \text{.} \} \langle \text{dekstra citilo} \rangle
\langle \text{citilo} \rangle \{\langle \text{letter} \rangle \mid \text{.} \} \langle \text{malcitilo} \rangle

der. 43

\langle \text{name expression} \rangle \rightarrow \langle \text{name} \rangle \mid \langle \text{field access} \rangle \mid \langle \text{array access} \rangle

der. 44

\langle \text{field access} \rangle \rightarrow \langle \text{name expression} \rangle \langle de \rangle \langle \text{name expression} \rangle

der. 45

\langle \text{array access} \rangle \rightarrow \langle \text{name expression} \rangle \langle \text{a} \rangle \langle de \rangle \langle \text{name expression} \rangle

der. 46

\langle \text{name} \rangle \rightarrow [la] \{\langle \text{adjective} \rangle \} \langle \text{noun} \rangle
der. 47

\( \langle \text{definition} \rangle \rightarrow \langle \text{infinitive verb} \rangle \ (\text{arguments}) \ '\text{signifas}' \ (\text{sentence}) \ '\text{finu}' \)

der. 48

\( \langle \text{parameters} \rangle \rightarrow [\langle \text{name} \rangle \ {\langle \text{separator} \rangle \ \langle \text{name} \rangle}] \)

der. 49

\( \langle \text{scheduled command} \rangle \rightarrow \)
\( \langle \text{command} \rangle \ '\text{je}' \ (\text{time point}) \)
\( \langle \text{command} \rangle \ '\text{je}' \ (\text{time span}) \)

der. 50

\( \langle \text{repeating command} \rangle \rightarrow \langle \text{command} \rangle \ '\text{cxiu}' \ (\text{time span}) \)

der. 51

\( \langle \text{once command} \rangle \rightarrow \langle \text{command} \rangle \ '\text{unufoje}' \ (\text{expression}) \)

der. 52

\( \langle \text{whenever command} \rangle \rightarrow \langle \text{command} \rangle \ '\text{cxiufoje}' \ (\text{expression}) \)
Appendix F

Examples

This appendix gives several examples for Domsaĝo routines and short speeches, in order to demonstrate some of the things achievable through the most basic personal assistant programming. After reading this thesis cover-to-cover, one should already understand Program 0.1, and figure that for any $X$ it returns $\lceil \sqrt{X} \rceil$.

The following are some everyday household actions automated via Domsaĝo:

- **Sxaltu la lumojn** - meaning: “activate the lights.” This command tells Domsaĝo to activate all the lights it can access.

  ![Abstract Syntax Tree - Sxaltu la lumojn](image)

- **Sxlosu la antauxan prodon** - meaning: “lock the front door.”

- **Fermu la fenestrajn blokojn** - meaning “close the window blinds.”

The above commands can be grouped into a routine to simplify the process.

**Program F.1 Nightify Routine**

1. Nokti signifas
2. Sxlosu la antauxan prodon
3. poste fermu la fenestrajn blokojn
4. poste sxaltu la lumojn
5. Finu

1. To nightify means:
2. Lock the front door
3. Afterwards close the window blinds
4. Afterwards activate the lights
5. End.

I personally often find myself wondering whether the water is hot enough to take a shower. With Domsaĝo, it is easy to create a noticeable change when an appliance has done its purpose:
Program F.2 Alert When Water are Ready

1 Assign ruĝzo al ĝis koloro de laborlamparo 1 Assign red to the color of the office lamp
2 cxinfoje ĝis nuna temperaturo de kaldrorno 2 whenever the current temperature of the boiler
3 estas pli granda ol kvardek. 3 is greater than forty.

A popular use for smart home systems is to feign a presence in the house during vacation a feat easily achieved with Domsago:

Program F.3 Pretend To Be Home

1 Szajnigi signifas 1 To pretend means:
2 hazardu tempon 2 Randomize time
3 inter la sepa horo kaj la oka horo 3 between 0700 and 0800
4 poste assignu gxin al unua tempo 4 afterwards assign it to first time.
5 poste hazardu tempon 5 afterwards randomize time
6 inter la dudek unua horo 6 between 2100
7 kaj la dudek dua horo 7 and 2200
8 poste assignu gxin al dua tempo. 8 afterwards assign it to second time.
9 poste sxaltu la televizo 9 afterwards activate the television
10 je la unua tempo. 10 at the first time.
11 poste malsxaltu la televizo 11 afterwards turn off the television
12 je la dua tempo. 12 at the second time.
13 Finu. 13 End.
14
15 Szajnigu cxiu tago. 15 Pretend every day.

Below are some additional examples:

Figure F.2 Abstract Syntax Tree - Nightify Routine

Technion - Computer Science Department - M.Sc. Thesis  MSC-2020-43 - 2020
**Program F.4** Disciplining a Child

1. `baziti knabo signifas` to ground a child means
2. `aixiom la pordon de la kambio de la knabo` lock the door of the child's room
3. `poste anoncuv citilo vi estas bazita.` afterwards announce "you are grounded.
4. `Nun pensu pri via malbona konduto malcitilo` Now think about your bad behaviour
5. `finu` end.

**Program F.5** Morning Routine

1. `matenu signifas` To morningify means
2. `raportu la veteron` report the weather
3. `poste raportu la trafikon al Haifa` afterwards report the traffic to Haifa
4. `finu` end
5. `matenu je sesa kaj duono cxiu tago.` morningify at six thirty every day.

**Program F.6** Party Routines

1. `hazardum signifas` to random-light means
2. `hazardu nombro gxis la kvanto de lumoj` randomize number up to the quantity of lights
3. `poste assignu gxin al luma nombro` afterwards assign it to light number
4. `poste hazardo koloron` afterwards randomize color
5. `poste assignu gxin al luma koloro` afterwards assign it to light color
6. `poste hazardu nombro inter sepdek kaj cent` afterwards randomize number between 70 and 100
7. `poste assignu gxin al luma brilo` afterwards assign it to light brightness
8. `poste sxangxu la koloro de luma nombro de lumoj al luma koloro` afterwards change the color of the light numberth light to light color
9. `samtempe sxangxu la brilo de` simultaneously change the brightness of
10. `luma nombro de lumoj al luma brilo` the light numberth light to light brightness
11. `finu` end
12. `festi signifas` to party means
13. `sxangxu la koloroj de lumojn al rugxo` change the color of the lights to red
14. `poste ludo mian amajn kantojn` afterwards play my favourite songs
15. `poste hazardumini cxiu sekundo` afterwards random-light every second
16. `samtempe cxiufoje la antauxan pordon malfermas` simultaneously whenever the front door opens
17. `anoncuv citilo nova gasto alvenis malcitilo` announce "a new guest arrived"
18. `finu` end

**Program F.7** Fabricate Sunrise

1. `sunlevigxi signifas` to sunrise means
2. `sxangxu la koloron de dormlumo al bluo` change the color of the bedroom lamp to blue
3. `poste assignu unu al brilindeko` afterwards assign 1 to brightness-index
4. `poste dum brilindeko estas pli malgranda aux egala al cent tiam` afterwards while brightness-index is less than or equal to 100 then
5. `assignu brilindeko al la brilo de la dormumo je kvin sekundoj` assign brightness-index to the brightness of the bedroom lamp in 5 seconds
6. `poste assignu brilindeko pli unu al brilindeko` afterwards assign brightness-index + 1 to brightness-index
7. `finu` end
8. `sunlevigxi cxiu tago je la sesa horo` sunrise every day at 06:00
9. `finu` end
10. `sunlevigxi cxiu tago je la sesa horo` sunrise every day at 06:00
Appendix G

Computational Power

This appendix deals with the computational power of Domsaĝo. First, we will briefly describe the proving system of Hoare logic, and then we will use it to prove that Domsaĝo is Turing complete by implementing a fundamental class of computational functions: the $\mu$-recursive functions.

G.1 Hoare Logic

Hoare logic is a formal system with a set of logical rules named after computer scientist Tony Hoare. It is used for proving the correctness of computer programs. In Section G.2, I use Hoare logic to prove the capabilities of Domsaĝo formally. During these proofs, I use the following notations:

A Hoare proof is comprised of Hoare triples, such as $\{ P \} C \{ Q \}$, in which $P$ and $Q$ are a precondition and a postcondition, respectively. Both are comprised of assertions made in first-order logic. $C$ is a command or a code segment. The meaning of this triple is that if $P$ holds before the execution of $C$, and if the execution of $C$ terminates, then $Q$ holds right after the termination of $C$.

An extension to this is the Hoare* system, which deals in total correctness. Its triples are noted as $\langle P \rangle C \langle Q \rangle$ which means that if $P$ holds before the execution of $C$ then $C$ necessarily terminates and also $Q$ holds after the termination of $C$.

The statements of a proof using Hoare or Hoare* logic looks like this:

$$(n) \langle P \rangle C \langle Q \rangle (n_1, n_2), (RULE)$$

Where $n$ is the current line number, $n_1, n_2$ are previous statements that contribute to the proof of the current line or lines $C$, and RULE is the axiom or compositional rule that provides the correctness of the current line. The rules in this thesis are used as defined by Francez [20]. In some cases I would like to use the skip axiom without
dedicating a line to a skip command, these instances will be marked with $C=0$.

To remain concise, I will write the conditions for Hoare logic in a Java-like syntax instead of Domsaĝo syntax, with upper case letters like $X$ denote logical variables outside of the verified program, teletype words like `nombro` denote program variables and symbols like square braces and dots function for array and field access, respectively.

**G.2 Turing Completeness**

To prove that a programming language has sufficient computational ability, one must prove that it can compute any function that a Turing machine \[60\] can. As another arbitrary show of ability, I programmed a function in Domujo that iterates over the prime numbers.

To prove Turing equivalence, I will use Domujo to compute a class of functions called $\mu$-recursive functions. The output for each routine will be simulated via different electrical appliances.

1. **Constant Function $C^k_n$**
   \[
   C^k_n(x_1, x_2, \ldots, x_k) = n \quad (G.1)
   \]

2. **Successor Operator $S$**
   \[
   S(x) = x + 1 \quad (G.2)
   \]

3. **Projection Function $P^k_i$**
   \[
   P^k_i(x_1, x_2, \ldots, x_k) = x_i \quad (G.3)
   \]

4. **Composition Operation $\circ$**
   \[
   h \circ (g_1, \ldots, g_m)(x_1, \ldots, x_k) = h(g_1(x_1, \ldots, x_k), \ldots, g_m(x_1, \ldots, x_k)) \quad (G.4)
   \]

5. **Recursion Operator $\rho$**
   \[
   \rho(g, h) = f \text{ where } \quad (G.5)
   \]
   \[
   f(0, x_1, x_2, \ldots, x_k) = g(x_1, x_2, \ldots, x_k) \quad \text{ and } \quad f(y + 1, x_1, x_2, \ldots, x_k) = h(y, f(y, x_1, x_2, \ldots, x_k), x_1, x_2, \ldots, x_k)
   \]

6. **Minimization Operator $\mu$**
   \[
   \mu(f)(x_1, x_2, \ldots, x_k) = z \iff f(z, x_1, x_2, \ldots, x_k) = 0 \text{ and } \quad (G.6)
   \]
   \[
   f(i, x_1, x_2, \ldots, x_k) > 0 \text{ for } i = 0, \ldots, z - 1
   \]
**Definition G.2.1** (Function Implementation). For Domşaĝo procedure *proceduri* with arguments *enigajxoj* and mathematical function $f : \mathbb{N}^m \rightarrow \mathbb{N}$ I say that procedure *proceduri* implements function $f$ if and only if:

$$\forall *enigajxoj = X \in \mathbb{N}^m: \langle *enigajxoj=X \rangle \text{proceduru enigajxoj} \langle gx_i = f(X) \rangle$$

I will mark this relation as I(*proceduro*, $f$)

Since I would like to prove this quality for implementations of the $\mu$-recursive functions, I will use the following rule:

$$\langle *enigajxoj=X \rangle \text{S} \langle \text{esprimo}=Y \rangle \quad \langle \text{esprimo}=Y \rangle \quad \langle *enigajxoj=X \rangle \text{proceduru enigajxoj} \langle gx_i=Y \rangle$$

(IMPL)

With S being a string of Domşaĝo statements, *esprimo* being a Domşaĝo expression, and provided that the body of *proceduri* is: *S poste revenu esprimo* This is a simplified version that does not concern itself with procedures stopping abruptly to return their calculated values. However, it will prove sufficient for my purposes. Its proof is as simple as a concatenation of the assignment rule (ASS) and the invocation rule (INVOC).

**Lemma G.2.2.** The zero function $C^k_0$ as a special case of Eq. (G.1) can be implemented in Domşaĝo

**Proof.** I claim that Eq. (G.1) is implemented by Program G.1, which I will prove using Hoare logic.

**Program G.1** Constant Function

1. **Konstanti nombroj signifas:**
2. **Revenu nul.**
3. **Finu.**

<table>
<thead>
<tr>
<th>Statement</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $\langle \text{nombroj}=X \rangle \emptyset \langle \text{nombroj}=X \rangle$</td>
<td>(SKIP)</td>
</tr>
<tr>
<td>2. $(\text{nombroj} = X) \rightarrow (\text{nombroj} = X \land 0 = Y)$</td>
<td>(ARITH)</td>
</tr>
<tr>
<td>3. $\langle \text{nombroj}=X \rangle \emptyset \langle \text{nombroj}=X \rangle$</td>
<td>(1, 2) (CONS)</td>
</tr>
<tr>
<td>4. $\langle \text{nombroj}=X \rangle \text{konstantu nombroj} \langle gx_i=Y \rangle$</td>
<td>(IMPL)</td>
</tr>
<tr>
<td>5. $(gx_i = Y \rightarrow gx_i = C^k_0(X)$</td>
<td>(DEFN)</td>
</tr>
<tr>
<td>6. $\langle \text{nombroj}=X \rangle \text{konstantu nombroj} \langle gx_i=C^k_0(X) \rangle$</td>
<td>(4, 5) (CONS)</td>
</tr>
</tbody>
</table>

**Lemma G.2.3.** The successor function defined in Eq. (G.2) can be implemented in Domşaĝo.

---

*esprimo:* expression
I will demonstrate using a televido:

\textit{Proof.} I claim that Eq. (G.2) is implemented by Program G.2, which I will prove using Hoare logic.

\textbf{Program G.2 Successor Function}

\begin{tabular}{ll}
1 & \textbf{Posteuli nombro signifas:} \\
2 & revenu nombro pli unu. \\
3 & \textbf{Finu.} \\
1 & To successor a number means: \\
2 & return number+1. \\
3 & \textbf{End.} \\
\end{tabular}

\begin{align*}
(1) \langle nombro=X \rangle & \preceq \langle nombro=X \rangle & (\text{\texttt{SKIP}}) \\
(2) (nombro = X) & \rightarrow (nombro + 1 = Y) & (\text{\texttt{ARITH}}) \\
(3) \langle nombro=X \rangle & \preceq \langle nombro+1=Y \rangle & (1,2)(\text{\texttt{CONS}}) \\
(4) \langle \text{nombroj}=X \rangle & \text{postaulu nombro} \langle gxi=Y \rangle & (\text{\texttt{IMPL}}) \\
(5) (gxi = Y & \rightarrow gxi = S(X)) & (\text{\texttt{DEFN}}) \\
(6) \langle \text{nombroj}=X \rangle & \text{konstantu nombroj} \langle gxi=S(X) \rangle & (4,5)(\text{\texttt{CONS}})
\end{align*}

\textbf{Lemma G.2.4.} The projection function \( P^k \) as a special case of Eq. (G.3) can be implemented in Domsaço.

\textit{Proof.} I claim that Eq. (G.3) is implemented by Program G.3, which I will prove using Hoare logic.

\textbf{Program G.3 Projection Function}

\begin{tabular}{ll}
1 & \textbf{Projekci nombro signifas:} \\
2 & revenu la unua de nombroj. \\
3 & \textbf{Finu.} \\
1 & To project numbers means: \\
2 & return the first of the numbers. \\
3 & \textbf{End.} \\
\end{tabular}

\begin{align*}
(1) \langle \text{nombroj}=X \rangle & \preceq \langle \text{nombroj}=X \rangle & (\text{\texttt{SKIP}}) \\
(2) (\text{nombroj} = X) & \rightarrow (\text{nombroj}[1] = Y) & (\text{\texttt{ARITH}}) \\
(3) \langle \text{nombroj}=X \rangle & \preceq \langle \text{nombroj}[1]=Y \rangle & (1,2)(\text{\texttt{CONS}}) \\
(4) \langle \text{nombroj}=X \rangle & \text{projekcu nombroj} \langle gxi=Y \rangle & (\text{\texttt{IMPL}}) \\
(5) (gxi = Y & \rightarrow gxi = P^1(X)) & (\text{\texttt{ARITH}}) \\
(6) \langle \text{nombroj}=X \rangle & \text{projekcu nombroj} \langle gxi=P^1(X) \rangle & (4,5)(\text{\texttt{CONS}})
\end{align*}

\textbf{Lemma G.2.5.} The composition operator \( \circ \) defined in Eq. (G.4) can be implemented in Domsaço

\textit{Proof.} I claim that Eq. (G.4) is implemented by Program G.4, which I will prove using Hoare logic.

I take the following as given:

\textsuperscript{2}televido: television
(a) The argument rutinoj referred in Program G.4 is an \( m \)-sized array of routine objects, each with a \( k \)-sized array as an argument.

(b) \( g_1, \ldots, g_m : \mathbb{N}^k \to \mathbb{N} \) are functions s.t. \( \forall i_o \in \mathbb{N} : I(ia \ de \ rutinoj,g_{i_o}) \) The argument formiti\(^3\) is a function object, receiving an \( m \)-sized array as an argument.

(c) The argument nombroj from Program G.4 is a \( k \) sized array of natural numbers.

(d) \( h : \mathbb{N}^m \to \mathbb{N} \) is a function s.t. \( I(formiti,h) \)

I use the implementation provided in Program G.4

Program G.4 Composition Operator

1 Formi rutinoj, formito kaj nombroj signifas: To compose routines and numbers means:
2 Enigajxoj estas nova listo 2 Arguments is a new list
3 poste assignu unu al io 3 afterwards assign i to i
4 poste dum io estas pli malgranda aux egala 4 afterwards while i \( \leq \)
5 al la longo de rutinoj faru: 5 the length of routines do:
6 Kalkulo estas la ia de rutinoj 6 calculation is the \( 1 \)th of routines
7 poste kalkulu nombroj 7 afterwards calculate numbers
8 poste assignu enigajxoj pli gxin al enigajxoj 8 afterwards assign arguments + it to arguments
9 poste assignu io pli unu al io. 9 afterwards assign i + 1 to i.
10 Finu. 10 End.
11 Poste formitu enigajxoj 11 Afterwards Sub-compose the arguments
12 poste revenu gxin. 12 afterwards return it.
13 Finu. 13 End.

A loop invariant is required to prove the above segment. I chose to use:

\[ pi(i_o, n) = (i_o + n = m + 1) \land (\text{nombroj} = Z) \land (\text{enigajxoj} = X \sim g_Y(Z) \sim \ldots \sim g_i_{i_o - 1}(Z)) \]

(G.7)

For some external variables \( X,Y \) which symbolize the content of enigajxoj before iteration and the starting index of the loop respectively.

I will start the proof within the loop:

\[ \begin{align*}
1 \langle & \ \text{rutinoj}[i_o]=\text{rutinoj}[i_o] \\
& \ \wedge \text{enigajxoj}=X \\
& \ \wedge i_o=Y \\
& \ \wedge \text{nombroj}=Z \rangle \implies \langle & \ \text{kalkulo}=\text{rutinoj}[i_o] \\
& \ \wedge \text{enigajxoj}=X \\
& \ \wedge i_o=Y \\
& \ \wedge \text{nombroj}=Z \rangle \quad (ASS)
\end{align*} \]

\[ \begin{align*}
2 \langle & \ \text{enigajxoj}=X \\
& \ \wedge i_o=Y \\
& \ \wedge \text{nombroj}=Z \rangle \implies \langle & \ \text{kalkulo}=\text{rutinoj}[i_o] \\
& \ \wedge \text{enigajxoj}=X \\
& \ \wedge i_o=Y \\
& \ \wedge \text{nombroj}=Z \rangle \quad (ARITH)
\end{align*} \]

\[ \begin{align*}
3 \langle & \ \text{kalkulo}=\text{rutinoj}[i_o] \\
& \ \wedge \text{enigajxoj}=X \\
& \ \wedge i_o=Y \\
& \ \wedge \text{nombroj}=Z \rangle \implies \langle & \ \text{I}[\text{kalkulo, televideo}[i_o].\text{kanalo}] \\
& \ \wedge \text{enigajxoj}=X \\
& \ \wedge i_o=Y \\
& \ \wedge \text{nombroj}=Z \rangle \quad (d)(GIVEN)
\end{align*} \]

\[^3\text{formiti is referred to as sub-compose in English}\]
Lemma G.2.6. The operator $\rho$ defined in Eq. (G.5) can be implemented in Domساgio.

Proof. I claim that Eq. (G.5) is implemented by Program G.5, which I will prove using Hoare logic.

Program G.5

Primitive Recursion Operator

```
Rekursi unufunkco, dufunkco, alio
kaj nombroj signifas:
Se alio estas pli granda al nul tiam:
    asignu alio - 1 al alio
    poste rekursu unufunkco, dufunkco, alio kaj nombroj
    poste dufunkcu alio,
    la kanalo de radio kaj nombroj
    poste asignu alio pli unu al alio.
    Alie:
        unufunkcu nombroj
        poste asignu brilo de la ampolo
        al kanalo de la radio
    Finu.
Finu.
```

To recurse first, second, other and numbers means:
If the other is greater than 0 then:
Assign other - 1 to other
afterwards recurse first, second, other, and numbers
afterwards assign other + 1 to other
First-ate numbers
afterwards assign the light’s brightness to the radio’s channel.

For this proof, I am assuming $I(\text{unufunkcu}, \text{ampolo.brilo})$ and $I(\text{dufunkcu}, \text{radio.kanalo})$.

I take the following as given:

- Argument unufunkco contains a procedure with $k$ arguments such that $\text{unufunkci} \rightarrow \text{brilo de ampolo}$ and dufunkco contains a procedure with $k+2$ arguments such that $\text{dufunkci} \rightarrow \text{kanalo de radio}$.
- $g$ and $h$ are computable functions such that $I(\text{unufunkcu}, \text{brilodeampolo})$ and $I(\text{dufunkcu}, \text{kanaloderadio})$.
- The appliances ampolo and radio exist.
Also, since rekursu is a recursive function, we must assume its correctness in order to prove it, in manner of induction. To prove that rekursu terminates, I will use two invariants:

\[ p_i(\text{alia}, \text{nombroj}, n) = \begin{cases} \text{alia} = 0 \\ \text{alia} + 1 = n \\ \text{nombroj} = Z \end{cases} \quad q = \left( \text{radio.kanalo} = f((\text{alia}) \leftarrow Z) \land \text{nombroj} = Z \right) \]

\[ (G.8) \]

With the “frown” operator above used for concatenation of ordered lists. I also use the following induction assumption for conciseness:

\[ \text{ind}(n) = \langle p_i(\text{alia}, \text{nombroj}, n) \rangle \text{rekursu unufunkco, dufunkco, alio kaj nombroj} \langle q \rangle \]

\[ (G.9) \]

This means if, before any invocation of rekursu unufunkco, dufunkco, alio kaj nombroj the parameter alio was not 0, was one less than invariant parameter \( n \), and the routine parameter nombroj was equal to list \( Z \), then after rekursu returns, the radio channel radio.kanalo is equal to invoking function \( f \) on list \( Z \) with alio inserted at its head, and parameter nombroj is still equal \( Z \).
Lemma G.2.7. The minimization operator $\mu$ defined in Eq. (G.6) can be implemented in Domsaño.
For any implementation \textit{minimumigi} I must therefore prove:

\[
\langle \exists n \in \mathbb{N} : f(n) = 0 \rangle \quad \text{minimumigi funkco kaj enigajxoj} \quad \langle \forall n' < n : f(n') > 0 \rangle \quad (G.10)
\]

\[
\{ \neg \exists n \in \mathbb{N} : f(n) = 0 \} \quad \text{minimumigi funkco kaj enigajxoj} \quad \{ \text{false} \} \quad (G.11)
\]

\textbf{Proof.} I claim that Eq. (G.6) is implemented by Program G.6, which I will prove using Hoare logic.

\textbf{Program G.6 Minimization Operator}

\begin{verbatim}
1 Minimumigi funkco kaj nombroj signifas:
2 Brilo de ampulo estas nul
gpose funkc la brilo de ampulo,
3 kaj nombroj
gposte dum la kanalo de radio
4 ne estas egala al nul tian:
5 Assign brilo de ampulo pli unu
6 to light's brightness
7 Assign 0 to light's brightness
8 Afterwards function-ate the
9 light's brightness and numbers.
10 Afterwards while the radio channel
11 is not equal to 0 then:
12 Assign light's brightness +1
13 to light's brightness
14 Afterwards function-ate the
15 light brightness and numbers.
16 Finu.
17 Finu.
\end{verbatim}

To prove the above segment I need a loop invariant. I have chosen to use:

\[
pi(\text{radio.kanalo}, \text{ampulo.brilo}, n) = \left( \begin{array}{l}
\forall x \in \mathbb{N} : f(x, x_1, \ldots, x_k) = 0 \\
\forall x < (\text{ampulo.brilo}) : f(x) > 0 \\
\forall \text{radio.kanalo} = f(\text{ampulo.brilo}) = \text{nombroj}
\end{array} \right)
\]

\quad \langle f(\text{ampulo.brilo} + n) = \text{ampulo.brilo} = 0 \rangle

\quad \langle 0 < x < (\text{ampulo.brilo}) : f(x) > 0 \rangle

\quad \langle \text{radio.kanalo} = f(\text{ampulo.brilo}) = \text{nombroj} \rangle

\quad (G.12)

By definition we can make the following conclusions:

\[
\forall n \neq 0 : pi(\text{radio.kanalo}, \text{ampulo.brilo}, n) \rightarrow \text{radio.kanalo} > 0 \quad (G.13)
\]

\[
pi(\text{radio.kanalo}, \text{ampulo.brilo}, 0) \rightarrow \text{radio.kanalo} = 0 \quad (G.14)
\]

\[
\exists x \in \mathbb{N} : f(x, x_1, \ldots, x_k) = 0 \rightarrow pi(\text{radio.kanalo}, \text{ampulo.brilo}) \quad (G.15)
\]

\[
\forall x \in \mathbb{N} : f(x, x_1, \ldots, x_k) = 0 \rightarrow pi(\text{radio.kanalo}, \text{ampulo.brilo}, \infty) \quad (G.16)
\]

for conciseness, I will also mark the following invariant for the relevant property of \(f\):

\[
\exists(f, \text{nombro}) = \exists n \in \mathbb{N} : f(n) \sim \text{nombro} \quad (G.17)
\]
I will start the proof within the while loop:

\[
\begin{align*}
(1) \langle ampolo.brilo+1=X+1 \land nombroj=Y \rangle L_{7-8} \langle ampolo.brilo=X+1 \land nombroj=Y \rangle \quad (ASS) \\
(2) \langle ampolo.brilo=X \land nombroj=Y \rangle L_{7-8} \langle ampolo.brilo=X+1 \land nombroj=Y \rangle \quad (ARITH) \\
(3) \langle radio.kanalo = f((X) \sim Y) \land nombroj=Y \land \exists f(nombroj) \rangle L_{9-10} \langle radio.kanalo = f((X+1) \sim Y) \land nombroj=Y \land \exists f(nombroj) \rangle \quad (IMPL) \\
(4) \langle radio.kanalo = f((X) \sim Y) \land nombroj=Y \land \exists f(nombroj) \rangle L_{9-10} \langle radio.kanalo = f((X+1) \sim Y) \land nombroj=Y \land \exists f(nombroj) \rangle \quad (ARITH) \\
(5) \langle radio.kanalo = f((X) \sim Y) \land nombroj=Y \land \exists f(nombroj) \rangle L_{7-10} \langle radio.kanalo = f((X+1) \sim Y) \land nombroj=Y \land \exists f(nombroj) \rangle \quad (2,4)(SEQ) \\
(6) \langle \pi((radio.kanalo, ampolo.brilo, n'+1)) \rangle \rightarrow \langle radio.kanalo = f((X) \sim Y) \land nombroj=Y \rangle \quad \langle \pi((radio.kanalo, ampolo.brilo, n')) \rangle \quad (ARITH) \\
(7) \langle radio.kanalo = f((X+1) \sim Y) \land nombroj=Y \rangle \rightarrow \langle radio.kanalo = f((X+1) \sim Y) \land nombroj=Y \rangle \quad (ARITH) \\
(8) \langle \pi((radio.kanalo, ampolo.brilo, n'+1)) \rangle L_{7-10} \langle \pi((radio.kanalo, ampolo.brilo, n')) \rangle \quad (5-7)(CONS) \\
(9) \langle \pi((radio.kanalo, ampolo.brilo, n'+1)) \rangle \rightarrow \langle radio.kanalo>0 \rangle \quad (ARITH) \\
(10) \langle \pi((radio.kanalo, ampolo.brilo, 0)) \rangle \rightarrow \langle radio.kanalo=0 \rangle \quad (ARITH) \\
(11) \langle \exists n : \pi((radio.kanalo, ampolo.brilo, n)) \rangle L_{5-11} \langle \pi((radio.kanalo, ampolo.brilo, 0)) \rangle \quad (8-10)(REP,*) \\
(12) \langle \exists n : \pi((radio.kanalo, ampolo.brilo, n)) \rangle L_{2} \langle \pi((radio.kanalo, ampolo.brilo, n)) \rangle \quad \langle \exists n : \pi((radio.kanalo, ampolo.brilo, n)) \rangle \quad \langle \exists n : \pi((radio.kanalo, ampolo.brilo, n)) \rangle \quad (11,16)(SEQ) \\
(13) \langle radio.kanalo = f((0) \sim Y) \land nombroj=Y \land \exists f(nombroj) \rangle L_{3-4} \langle radio.kanalo = f((0) \sim Y) \land nombroj=Y \land \exists f(nombroj) \rangle \quad (IMPL) \\
(14) \langle \pi((radio.kanalo, ampolo.brilo, 0)) \rangle L_{2} \langle \pi((radio.kanalo, ampolo.brilo, 0)) \rangle \quad \langle \pi((radio.kanalo, ampolo.brilo, 0)) \rangle \quad \langle \pi((radio.kanalo, ampolo.brilo, 0)) \rangle \quad (12,13)(SEQ) \\
(15) \langle radio.kanalo = f((0) \sim Y) \land nombroj=Y \land \exists f(nombroj) \rangle \rightarrow \langle \exists n : \pi((radio.kanalo, ampolo.brilo, n)) \rangle \quad (ARITH) \\
(16) \langle radio.kanalo = f((0) \sim Y) \land nombroj=Y \land \exists f(nombroj) \rangle \rightarrow \langle \exists n : \pi((radio.kanalo, ampolo.brilo, n)) \rangle \quad (14,15)(CONS) \\
(17) \langle radio.kanalo = f((0) \sim Y) \land nombroj=Y \land \exists f(nombroj) \rangle \rightarrow \langle \exists n : \pi((radio.kanalo, ampolo.brilo, 0)) \rangle \quad (11,16)(SEQ) \\
(18) \langle radio.kanalo = f((0) \sim Y) \land nombroj=Y \land \exists f(nombroj) \rangle \rightarrow \langle \exists n : \pi((radio.kanalo, ampolo.brilo, 0)) \rangle \quad (ARITH) \\
\end{align*}
\]
\[
\begin{align*}
\text{(19)} & \quad \{ \mathbb{E}(f, \text{nombroj}) \} L_{2-11} \left( \mathbb{E}(f, \text{nombroj}) \right) \\
& \quad \mathbb{E}(f, \text{nombroj}) = 0 \\
& \quad \wedge \exists n < \text{ampolo brilo} : f(n) > 0 \\
& \quad \text{(17, 18) (CONS)}
\end{align*}
\]

\[
\begin{align*}
\text{(20)} & \quad \{ \text{ampolo brilo} + 1 = x + 1 \} L_{7-8} \left( \text{ampolo brilo} = x + 1 \right) \\
& \quad \text{ampolo brilo} = x + 1 \\
& \quad \wedge \mathbb{E}(f, \text{nombroj}) \\
& \quad \text{radio kanalo} = f((x) \wedge Y) \\
& \quad \text{(ASS)}
\end{align*}
\]

\[
\begin{align*}
\text{(21)} & \quad \{ \text{ampolo brilo} = x \} L_{7-8} \left( \text{ampolo brilo} = x + 1 \right) \\
& \quad \text{ampolo brilo} = x + 1 \\
& \quad \wedge \mathbb{E}(f, \text{nombroj}) \\
& \quad \text{radio kanalo} = f((x) \wedge Y) \\
& \quad \text{(ARITH)}
\end{align*}
\]

\[
\begin{align*}
\text{(22)} & \quad \{ \text{ampolo brilo} = x + 1 \} L_{9-10} \left( \text{ampolo brilo} = x + 1 \right) \\
& \quad \text{ampolo brilo} = x + 1 \\
& \quad \wedge \mathbb{E}(f, \text{nombroj}) \\
& \quad \text{radio kanalo} = f((x) \wedge Y) \\
& \quad \text{(IMPL)}
\end{align*}
\]

\[
\begin{align*}
\text{(23)} & \quad \{ \text{ampolo brilo} = x + 1 \} L_{9-10} \left( \text{ampolo brilo} = x + 1 \right) \\
& \quad \text{ampolo brilo} = x + 1 \\
& \quad \wedge \mathbb{E}(f, \text{nombroj}) \\
& \quad \text{radio kanalo} = f((x + 1) \wedge Y) \\
& \quad \text{(ARITH)}
\end{align*}
\]

\[
\begin{align*}
\text{(24)} & \quad \{ \text{ampolo brilo} = x \} L_{7-10} \left( \text{ampolo brilo} = x + 1 \right) \\
& \quad \text{ampolo brilo} = x + 1 \\
& \quad \wedge \mathbb{E}(f, \text{nombroj}) \\
& \quad \text{radio kanalo} = f((x + 1) \wedge Y) \\
& \quad \text{(21, 23) (SEQ)}
\end{align*}
\]

\[
\begin{align*}
\text{(25)} & \quad \{ \text{ampolo brilo} = x \} \rightarrow \{ \text{ampolo brilo} = x \} \\
& \quad \text{ampolo brilo} = x \\
& \quad \wedge \mathbb{E}(f, \text{nombroj}) \\
& \quad \text{radio kanalo} \neq 0 \\
& \quad \text{(ARITH)}
\end{align*}
\]

\[
\begin{align*}
\text{(26)} & \quad \{ \text{ampolo brilo} = x + 1 \} \rightarrow \{ \text{ampolo brilo} = x + 1 \} \\
& \quad \text{ampolo brilo} = x + 1 \\
& \quad \wedge \mathbb{E}(f, \text{nombroj}) \\
& \quad \text{radio kanalo} \neq 0 \\
& \quad \text{(ARITH)}
\end{align*}
\]

\[
\begin{align*}
\text{(27)} & \quad \{ \text{ampolo brilo} = x \} \rightarrow \{ \text{ampolo brilo} = x \} \\
& \quad \text{ampolo brilo} = x \\
& \quad \wedge \mathbb{E}(f, \text{nombroj}) \\
& \quad \text{radio kanalo} \neq 0 \\
& \quad \text{(24 - 26) (CONS)}
\end{align*}
\]

\[
\begin{align*}
\text{(28)} & \quad \{ \text{ampolo brilo} = x \} \rightarrow \{ \text{ampolo brilo} = x \} \\
& \quad \text{ampolo brilo} = x \\
& \quad \wedge \mathbb{E}(f, \text{nombroj}) \\
& \quad \text{radio kanalo} \neq 0 \\
& \quad \text{(27) (REP)}
\end{align*}
\]

\[
\begin{align*}
\text{(29)} & \quad \{ 0 = 0 \} L_{2} \left( \text{ampolo brilo} = 0 \right) \\
& \quad \text{ampolo brilo} = 0 \\
& \quad \wedge \mathbb{E}(f, \text{nombroj}) \\
& \quad \text{(ASS)}
\end{align*}
\]

\[
\begin{align*}
\text{(30)} & \quad \{ \text{ampolo brilo} = 0 \} L_{3-4} \left( \text{ampolo brilo} = 0 \right) \\
& \quad \text{ampolo brilo} = 0 \\
& \quad \wedge \mathbb{E}(f, \text{nombroj}) \\
& \quad \text{radio kanalo} = f((0) \wedge Y) \\
& \quad \text{(IMPL)}
\end{align*}
\]

\[
\begin{align*}
\text{(31)} & \quad \{ \neg \mathbb{E}(f, \text{nombroj}) \} L_{2-4} \left( \text{radio kanalo} = f((0) \wedge Y) \right) \\
& \quad \text{ampolo brilo} = 0 \\
& \quad \wedge \mathbb{E}(f, \text{nombroj}) \\
& \quad \text{radio kanalo} = f((0) \wedge Y) \\
& \quad \text{(29, 30) (SEQ)}
\end{align*}
\]
Therefore line 19 above proves Eq. (G.10) while line 36 proves Eq. (G.10).

**Theorem G.1.** Domsaĝo is a Turing complete programming language.

**Proof.** According to Turing [59], the class of recursively computable functions is equal to the class of functions computable by a Turing machine. Since I have shown in Lemmas G.2.2 to G.2.7 that Domsaĝo is capable of implementing any \( \mu \)-recursive function, I have proven it Turing complete.

**G.3 Announcing Primes**

The following is a prime number announcing program (and its translation). It consists of three functions: a square function, a predicate for primality, and an iteration over natural numbers announcing the primes to the user. To handle the function outputs, I will use an oven and a light-bulb.
Program G.1 Prime Announcing Routine

1 **Kvadrati nombron signifas:**
2 Assignu nombro fojoj numero al
3 temperatura de forno.
4 Finu.
5
6 **Cxuprimi nombron signifas:**
7 Assignu du al unuo
8 poste kvadratu unuo
9 poste dum la temperatura de forno ne estas
10 pli granda ol numero tiam:
11 Assignu unuo al duo
12 poste dum unuo fojoj duene estas
13 pli granda ol numero tiam:
14 Se unuo fojoj du estas
15 egala al numero tiam:
16 Malsxaltu la ampolon
17 poste revenu.
18 Finu.
19
20 Poste asignu duo pli unu al duo
21 Finu.
22
23 Poste asignu unuo pli unu al unuo
24 poste kvadratu unuo.
25 Finu.
26
27 Assignu du al indekso
28 poste dum vero tiam:
29 Cxuprimu indekso
30 poste se la ampolo sxaltas tiam:
31 Anoncu indekso.
32 Finu.
33 Poste asignu indekso pli unu al indekso
34 Finu.

To square a number means:
2 Assign number * number to
3 the temperature of the oven.
4 End.
5
6 To check primality of a number means:
7 Assign two to first.
8 Afterwards square first.
9 Afterwards while the oven's temperature is not greater than number then:
10 Assign first to second.
11 Afterwards while first * second is not greater than number then:
12 If first * second is equal to number then:
13 Turn off the light.
14 Afterwards return.
15 End.
16
17 Afterwards add 1 to second.
18 End.
19
20 Afterwards add 1 to first.
21 Afterwards square first.
22 End.
23
24 Afterwards activate the light.
25 End.
26
27 Assign two to index.
28 Afterwards while true then
29 Check primality of index.
30 Afterwards if the light is on then
31 Announce index.
32 End.
33
34 Afterwards add 1 to index.
35 End.

The effect of invoking Program G.1 are shown in Fig. G.1.

96
Figure G.1 Demonstration of prime identification routine

(a) A virtual house containing 100 unlit lights

(b) Lights colored by prime routine
Appendix H

Installation Guide & Quick start

This appendix contains instructions on how to obtain and install Domsaĝo, as well as run the provided examples. The only requirements are a python installation of version 3.6, at least. It is important to state that the demonstration uses a GUI, so users must be able to run graphical applications to enjoy the full experience.

In this chapter, we will show the Linux and Windows commands needed for installation side by side. In these instances, the Windows commands will be on the left, preceded by a greater-sign (>). The Linux commands will be on the right, preceded by a dollar sign ($). Commands agreed upon by both operating systems will be written once.

**H.1 Installing and Using the Demo**

The simplest way to install Domsaĝo is from the GitHub repository using pip.

First, we create a Python venv (virtual environment). That will create a separate directory in which to install Domsaĝo and its required packages while maintaining the integrity of the system Python environment.

```
python -m venv domsa_venv
```

If, when creating the venv, a message is displayed about needing to install a system package (e.g., python3-venv), follow the instructions in the message and try the command again.

Next, we enter the venv:

```
> .\domsa_venv\Scripts\activate    $ source domsa_venv/bin/activate
```

Though not always required, it is a good idea to upgrade Python’s system packages. That can prevent future installation error messages.

```
python -m pip install --upgrade pip
python -m pip install --upgrade setuptools wheel
```

Now we can install Domsaĝo through git:

```
python -m pip install git+https://github.com/liordon/domsaĝo.git
```

And that’s it! Domsaĝo is now installed and usable. For example, let us try running the virtual house demo. Run the following commands inside the python interpreter of your new venv:

```python
import domsagxo

domsagxo.eo_demo()
```

At this point, the virtual house should appear before you. This would be a good time to try the Domsaĝo code mentioned in Fig. 1.1, or define the `radiki` routine from Program 0.1 and invoke `radiku sescent okdek du poste anoncu gxin`, getting the reply 27 as promised. I hope that upon rereading it with your newly acquired knowledge of Domsaĝo, you could recognize it for computing $\lceil \sqrt{x} \rceil$ Or you could try affecting the provided devices in a myriad of other manners, to your heart’s content.
התרומות העיקריות של מחקר זה הם:

• תיאור שפת מעטה לעוזרים אישיים המבוססת על שפת האספרנטו, שתפקידה המסירה מתכנת

• פעולה לשימוש בריצוף LALR, באפשרות ניתוח שפה חדש מתכנת לשפה הטבעית

• תיאור והבנה של שפת המעטה כשלמה, ב fgets ולפייה טרירית

• הדרجة של ציון במשתתפים

• הרגשה למשתתפים

• כל התוכן וה.Promise שליהם של עם סכום

במסגרת התחזוקה של חיבור זה שפה מובילה את 

שפת המעטה הבנויה בחיבור זה קרו 

(dom-sa-jo), והיא שפה אימפרטיבית多少钱 זור 

Michelle. פירוש השם מ⇌วลגשל תכנית או "הصحف של הבית". במסגרת בחיבור זה אבקש את החכמים בחיבור זה.

הצעזו המכסים בתוכן על חשבון ב الثقافيים ואיתנים.
The improvements in recent years in the abilities of speech analysis, together with chatbot technology, led to the widespread use of what was initially thought of as gadgets—gadgets, Alexa and others—known as personal assistants. These are programs with a voice interface, enabling users to request simple tasks such as reminders, managing shopping lists, answering questions on the Internet, and much more. These personal assistants try to break the stereotype of the "memory" role, and indeed, most of them are female by default. However, unlike the stereotype, these "noble assistants" are unable to learn new things through an ordinary conversation. They can be taught only through special applications or by programming new 'features'. For some, even the最难学习的 functionalities are not available.

A simple way to bridge the gap between single-command interaction and developing a new skill is to use a script language, similar to how a shell script can be used to concatenate multiple commands and avoid the need to develop an application in the Java language. This script language can be simple enough to be analyzed by a parser like LALR, and still powerful enough to simulate Turing. This language allows the user of the "personal assistant" to also become its trainer or tutor, and the language that is used for teaching—this script language—should fit the voice interface, which is, of course, a programming language.

The obstacles that have prevented the creation of this until now are:

• The multi-meaning of words in natural language—meaning is not always determined by the words themselves, but also by pauses, context and the relationship of the words to each other.
• On the other hand, programming languages are used in the world of programming, which is more rigid, strict and unambiguous.
• Although natural languages are used in the world of spoken language, programming languages are used in the world of writing, and despite our feeling that the transition between these worlds is simple—this is not the case: how is it possible to use special characters (some vowels, various types of brackets and mathematical symbols)?
• How can the ease of distinguishing between types of pauses in speech be achieved?
• Can we be confident that the computer will understand our language?
• And other research that has been published on these issues.

Each study was aimed at a different field of interest—personal assistants or programming speech—as different topics, and the differences that were written were written separately. In this study, I attempt to provide a solution that is not based on the computational power assigned to the device, but at the same time it does not completely abandon the world of spoken dialogue for human beings.

Under the framework of this study, I will not deal with the processing of text to speech—this is a separate problem that is not addressed in this study. The solution presented in this study comes from the field of designed languages: languages that were invented by people for a specific purpose—whether it is a peace treaty or a created for entertainment alone.

These languages are characterized by a simple, clear grammar that simplifies computational processing, in addition to being known by the ear to make their learning and use easier. I believe that this type of language is precisely the way to bridge the obstacles that were described, in addition to being very close to the idea of "script shell".
העבודה זו בוצעה בהנחייתו של פרופסור יוסי גיל, בפקולטה למדעי המחשב.

הכרת תודה מסורה לטכניון על מימון מחקר זה.
דומסגל
שפת תכנות דבורה
לזורכ חכונת עורים ואריזים

והבך על מתカー

לッシュ מיילו חלק של תדרישותkeleton המואר
וגישérer למודיעים בנודי המわかる

ליאור שמואל

הוגש לסנט הטכניון — מכון טכנולוגי לישראל
סיוון תשפ' חיפה מאי 2020
דומגל
שפת תכנות דבורה
לזרך תכנון חיזוק אריזים

לייאור שמואל