UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Well typed embedded program cannot burn

Nanikafuako Augusto Lusende André

Dissertação para obtenção do Grau de Mestre
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Simão Patricio Melo de Sousa

Covilhã, Outubro de 2018

Dedicatória

Dedico este trabalho aos meus pais, pelo incentivo dado na busca dos meus sonhos. A minha esposa e a minha filha.

Agradecimentos

A Jeová, por estar sempre comigo nesta caminhada e tornar esse sonho possível. À minha família, especialmente aos meus pais Sandra Lusende André e Makumbundo Alice Lusende André, pelos incentivos dado que ajudou bastante na conclusão de mais uma etapa importante da minha vida. A minha esposa Mariete Manuela S. Lusende André e minha Filha Alisandra Mirian S. Lusende André pelo amor, compreensão incentivo e ajuda. Ao meu amigo e conselheiro Dr. Valente Ribeiro Muhongo. Ao amigo e colega João Santos Reis, por me ajudar nas questões relacionadas à programação. Ao meu orientador, Simão Patricio Melo de Sousa pela confiança e pela atenção dedicada na orientação deste trabalho. A todos os professores, colegas e amigos que contribuíram para a conclusão deste trabalho.

Resumo

Este trabalho apresenta uma farramenta voltada para o ensino de programação e circuítos electronicos usando o arduino. A ferramenta consiste em um compilador que através de uma linguagem de programação textual e simplificada que é a linguagem ardlight, permitir a escrita de programas que depois de compilados pelo ardlight, possam ser enviados e executados como programa do arduíno. Originalmente, os programas eram executados de uma forma em que o programador tinha de pensar em todos os aspetos a ver com o circuíto a ser construído. Com a extensão proposta, a ideia é que o programa compilado pelo ardlight possa fazer de forma automática o que os construtores de circuítos electrónicos em arduíno fazem, acrescendo ainda os cálculos necessários para a proteção do respectivo circuíto. Para que o ardlight funcione, o programador deve escrever no ficheiro de entrada, informações referentes aos componentes que constituem o circuíto a ser construído. Assim, através dos comandos programados com a liguagem ocaml no ardlight é possível criar um ficheiro de saída com todas instruções necessárias e os cálculos para a construção do circuíto. Destaca-se ainda a inclusão da plataforma Arduino, por ser uma opção mais acessível devido as suas características de hardware e software livre, e também pelo fato de que boa parte das instituições de ensino podem adotar o ensino de robótica, pois o kit do arduíno tem baixo custo. Por fim, no intuito de validar o funcionamento do compilador do ardlight, foram feitos testes e observou-se que o ardlight foi capaz de executar todos os comandos da linguagem, apesar de apresentar algumas limitações.

Palavras-chave

Linguagens de programação. Ensino de programação. circuítos electronicos educacional. Arduíno UNO.

Abstract

This work presents a tool aimed at teaching programming and electronic circuits using arduino. The tool consists of a compiler that, through a simplified textual programming language, allows the writing of programs that after being compiled by ardlight, can be sent and executed as an arduino program. Originally, the programs were run in a way that the programmer had to think of all the aspects to do with the circuit to be built. With the proposed extension, the idea is that the program compiled by the ardlight can automatically do what the arduino electronic circuit builders do, while adding the calculations necessary to protect the respective circuit. In order for the ardlight to work, the programmer must write in the input file information about the components that make up the circuit to be built. Thus, through the commands programmed with the ocaml language in the ardlight it is possible to create an output file with all necessary instructions and calculations for the construction of the circuit. It is also worth noting the inclusion of the Arduino platform, as it is a more accessible option due to its hardware and free software features, and also because many educational institutions can adopt robotics teaching, since the arduino kit has low cost. Finally, in order to validate the operation of the ardlight compiler, tests were made and it was observed that the ardlight was able to execute all the commands of the language, although it presented some limitations.

Keywords

Programming languages. Programming teaching. educational electronic circuits. Arduíno UNO

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

UBI	Universidade da Beira Interior
GPS	Global Positioning System
LCD	Liquid Crystal Display
CEA	Commissariat de L´ Energic Atomique
IBM	International Business Machines
IDE	Integrated Development Environment
LED	Light Emitting Diode
CAM	Categórica Máquina abstrata
FAM	Functional Abstract Machine
TIC	Tecnologias de Informação e Comunicação
DSC	Departamento de Sistemas e Computação
XML	eXtensible Markup Language
CSV	Comma-separated values

Capítulo 1

Introdução

Estamos diante de uma cultura tecnológica cada vez mais influente no processo de ensino e aprendizagem, em que presenciamos o nascimento de áreas do conhecimento decorrentes das inovações tecnológicas. E perante a este facto urge a necessidade de desde cedo implementar matérias e tecnologias básicas aos estudantes para que possa despertar a paixão pela tecnologia. E um dos importantes efeitos das tecnologias no âmbito da cultura estudantil é a mutação pela qual ela tem feito nas várias áreas do conhecimento em que ela atua. E segundo Levy afirma que "devemos construir novos modelos do espaço dos conhecimentos" [Pie99]. E face a essas mudanças, o autor ressalta que o educador "é incentivado a tornar-se um animador da inteligência coletiva de seus grupos de alunos em vez de fornecer de forma direta os conhecimentos".
Perante este contexto, a robótica, a programação e a lógica no contexto educativo são considerada ferramentas poderosas nos processos de aprendizagem, pois exercita e cria a curiosidade, a imaginação e a intuição, elementos centrais que favorecem experiências estimuladoras da decisão e da responsabilidade [GOM10]. A robótica a programação e a lógica no contexto educativo também proporcionam melhora do raciocínio lógico e das habilidades manuais, através das atividades de programação e montagem dos robôs e dos circuitos eletrónicos. Entretanto, a sua inserção no processo de ensino-aprendizagem encontra muitas dificuldades financeiras, pois muitas instituições não têm condições de suportar com o alto custo dos kits proprietários, usados para o desenvolvimento dos projetos. Assim, uma alternativa para a adoção da robótica da programação e da lógica no contexto educativo é a utilização de plataformas de hardware e de software open source, ou seja, plataformas de código aberto. Dentre estas plataformas, o Arduíno ganha destaque por ser uma plataforma de fácil utilização. De acordo com McRoberts com relação ao Arduíno, pessoas que não são da área técnica podem, rapidamente, aprender o básico e criar seus próprios projetos em um intervalo de tempo relativamente curto" [Mic11].

1.1 Contexto do Problema

A dinâmica das tecnologias, aumenta em grande escala, e a procura do que implementar nas primeiras aulas de introdução as tecnologias tem suscitado bastante problema no seio dos professores, pois a complexidade das matérias ligadas as tecnologias parecem muito difícil muitas das vezes para os seus educados, e isso refreia os professores e ficam como que de mãos atadas sem muita das vezes saber o que fazer. Por outro lado ao analisar e fazer uma breve comparação de forma superficial, veremos que há interesse cada vez mais nos alunos em saber como funcionam as coisas no ponto de vista tecnológico. Como por exemplo: As portas automáticas das escolas, os semáforos, sensores de presença etc. Não descartando a complexidade das matérias pois elas em muitos casos envolvem cálculos, fórmulas físicas e certa lógica, mas não discordando essas dificuldades os conhecimentos de Algoritmia ajudaria no aprendizado do educando. E ao observar a literatura reconhecida no meio académico para o estudo do algoritmo encontramos F. Ximenes que define algoritmo como uma sequência ordenada e finita de passos, visando à solução de um dado problema [Xim93]. Para reforçar a definição de F. Ximenes, encontramos

as formulações de Boratti e Oliveira a definir algoritmo como “uma sequência finita e lógica de instruções executáveis, especificadas em uma determinada linguagem, que mostram como resolver alguns problemas” [eO04]. Nesta senda implementar um compilador com sistema de tipo que visam sistemas embutido juntamente com a placa de Arduíno levaria o ensino da robótica da programação e da lógica no contexto educativo teoria á pratica.
Tendo enconta o contexto do problema, introduzinos aqui a teoria de Robin Milner ”Well typed embedded programs cannot go wrong” ou seja “Programas bem tipados não podem dar errado” de (1978) [Mil78]. Agarrados a esse teoria, construimos o compilador do ardlight, este que se aprópria ao lema acima citado e o adapta ”Well typed embedded programs cannot burn”.

1.2 Objectivos e Questões de estudo

Para responder as preocupações levantadas no principio deste trabalho, os objetivos delineados para este trabalho são:

- Investigar a viabilidade de um software auxiliar no desempenho dos estudantes de base na aprendizagem de circuitos integrados standard usando a placa arduíno.
- Desenhar e implementar uma linguagem de programação associada ao seu computador, para o desenho de programas embutidos e respetivos circuitos.

O software escolhido será um compilador. Gennari afirma que compilar “é traduzir um programa escrito em linguagem de programação para a única linguagem que o computador pode entender, que é a binária, também chamada de Linguagem de Máquina”. O autor ainda define o compilador como “o programa que realiza essa tradução”[GEN99]. Cabe lembrar que a lógica desenvolvida pelo aluno não é entendida pelo computador, pois está escrito em linguagem humana. É necessário traduzir essa lógica para uma linguagem que a máquina possa entender e, a partir desse novo código gerado, testar o raciocínio e retornar os resultados. O compilador escolhido para essa pesquisa é o Ardlight desenvolvido pelo mestrando Nanikafuako Augusto Lusende André e coordenado pelo Professor Dr. Simão Patricio Melo de Sousa, da Universidade da Beira Interior, localizada em Covilhã - Portugal.
A questão de estudo desta pesquisa é: Como o software (compiladores) contribuirá para o aprendizado dos alunos de base na área de circuitos eletrónicos ?

1.3 Justificativa

O ensino de Tecnologias de Informação e Comunicações nas escolas de base é importante, pois fará que os estudante entrem em contacto com o mundo das tecnologias desde cedo. A importância desta pesquisa visa contribuir nos objetivos das Tecnologias de Informação e Comunicações no ensino das tecnologias nas escolas de base. Possibilitar que os estudantes aprendam desde cedo conceitos relacionados a circuitos eletrónicos standard, e de uma maneira simples, com a ajuda do software (compiladores) Ardlight possam implementar na pratica tais conhecimento ou seja preocupar-se com a progração e o ardlight gerar o circuíto correto. É importante salientar que a evolução das Tecnologias de Informação e Comunicações é muito rápida, então o estudo dela desde cedo, vai ajudar os estudantes, não só a tomarem conhecimento que ela existe mas também a se interessarem por ela.

1.4 Pressupostos Teórico-Metodológicos

1.4.1 Metodologia Exploratória

Esta pesquisa foi analisada num contexto de Experimento de Ensino, de cariz exploratório, valendo-se de dados quantitativos apesar de ser de natureza, predominantemente, qualitativa. Gil define pesquisa exploratória como A que tem como principal finalidade desenvolver, esclarecer e modificar conceitos e ideias, com vistas a formulação de problemas mais precisos ou hipóteses pesquisáveis para estudos posteriores [GIL99].

1.4.2 Metodologia Iterativa

A metodologia de desenvolvimento do compilador foi iterativa, consistindo de uma série de protótipos compiladores (escritos em Ocaml). Este tipo de abordagem combina as vantagens da prototipagem rápida com a eventual eficiência. Segundo Abdulaziz Ghuloum (2006) A metodologia iterativa é baseada nas sugestões, que propõe uma cessão de compiladores em funcionamento, cada um traduzindo um subconjunto cada vez mais rico do objetivo língua. Isto explora a característica hierárquica do Esquema, em que um pequeno número Um conjunto de formulários fundamentais descreve a sintaxe de cada programa válido do Esquema.

1.4.3 Teste

A estratégia de testes empregada foi ”teste de integração contínua”; onde, a cada iteração do software, a funcionalidade adicionada pelo novo código é verificada. Este teste estratégico é particularmente produtiva quando usada com a metodologia nitidamente iterativa descrito aqui. O protótipo inicial é muito pequeno, e é muito fácil raciocinar sobre o cálculo e depurar manualmente, em cada iteração, é muito claro o que é novo.

1.5 Organização dos Capítulos

Esta dissertação está organizada em IV capítulos. O capítulo I - Introdução: Retrata o cenário em que estamos incerido, trata dos objectivos desta investigação e sua justificativa e dos pressupostos teoricos e metodológico que conduziram esta investigação; O capítulo II - Revisão da Literatura: Apresenta a revisão da literatura que deu suporte a essa dissertação e a relação existente entre esses com a solução propósta; O capítulo III - Ardlight, Linguagem e Compilação: Mostra a implementação da resolução do problema proposto; E por ultímo o capítulo VI - Conclusão e Trabalhos Futuros: Para alêm de concluir este trabalho, mostra a visão futurista a implementar no ardlight.

Capítulo 2

Revisão da Literatura

2.1 Estado da Arte

Neste capítulo, mostraremos alguns trabalhos já feitos em direção ao que pretendemos fazer, e espelharemos a constituição da linguagen Ocaml utilizada para a construção do compilador Ardlight, para contribuir ainda mais no que tange ao ensino das tecnologias desde o ensino de base.

Webportugol

No artigo Auxiliando A Aprendizagem De Algoritmos Com A Ferramenta Webportugol, os autores, Hostins H. e Raabe A., apresentam as fases de desenvolvimento e como fazer a utilização da ferramenta WEBPORTUGOL que busca auxiliar na construção da lógica de programação nas disciplinas iniciais de cursos da área de Computação. O WEBPORTUGOL foi desenvolvido em um applet Java e pode ser utilizado através de um navegador Internet. Foi incluído um recurso de verificação dos algoritmos baseado em valores de entrada e saída pré-definidos. Essa ferramenta é muito forte na introdução aos educandos nas áreas de programação, pois a linguagem de programação usada é muito parecida a usada dia a dia pelos alunos. Atualmente muito usados nos paises como Angola e Brasil [HOS07].

O Logo

No estudo O Logo Como Ferramenta Auxiliar No Desenvolvimento Do Raciocínio Lógico – Um Estudo De Caso, Martins, S. e Correia, L., apresentam um experimento realizado com aluno empregando a Linguagem LOGO como metodologia de apoio ao desenvolvimento do raciocínio lógico para programação em uma proposta de integração de conteúdos das disciplinas de Matemática e Algoritmos na forma de um projeto de contexto interdisciplinar. Concluiu-se ao término do estudo que muitos alunos não estavam totalmente conscientes dos vários conceitos computacionais explorados durante a realização dos exercícios, deixando a desejar no que tange ao aprimoramento e recuperação de suas deficiências, porém, uma das grandes características da linguagem LOGO é possibilitar ao aluno estar mais ativo no processo ensino-aprendizagem imprimindo nesse processo o seu ritmo de absorção de conteúdo e transformação deste em conhecimento [MAR03].

RoboMind FURB

O RoboMind FURB é um plugin também desenvolvido no DSC da FURB para ser utilizado no RoboMind. O RoboMind é um ambiente de programação de robôs open source criado por Arvid Halma, da Universidade de Amsterdam. Ele foi inspirado na linguagem Logo e permite o aprendizado dos conceitos básicos de programação de computadores [BEN08].

FURBOT

FURBOT, uma biblioteca de apoio ao ensino de lógica de programação; o plugin RoboMind FURB, desenvolvido para ser utilizado no ambiente RoboMind; O FURBBOT foi criado em 2008, no âmbito do Departamento de Sistemas e Computação (DSC) da Universidade Regional de Blumenau (FURB). O projeto consiste em um ambiente de apoio ao ensino de lógica de programação com forte apelo na área de jogos. A ideia é que o aluno desenvolva algoritmos de controle de

personagens de tal forma a criar uma atmosfera facilitadora do aprendizado [VAH08]. Para utilizar o FURBOT basta criar um projeto Java em um ambiente de programação como o Eclipse ou o Netbeans, por exemplo. Em seguida, deve-se adicionar ao projeto a biblioteca do FURBOT e outras três bibliotecas auxiliares, necessárias para o seu funcionamento. Além disso, é preciso copiar para o projeto o arquivo eXtensible Markup Language (XML) referente ao exercício a ser solucionado. A ideia é que este XML seja criado pelo professor e contenha as configurações do cenário de acordo com os objetivos do exercício. Feito isto, o aluno deve criar uma classe com o mesmo nome do arquivo XML referente à atividade, sendo que esta classe deve estender a classe br.furb.furbot.Furbot.

O Cubetto

O Cubetto é um brinquedo que visa ensinar lógica de programação para crianças. Ele foi projetado por Matteo Loglio, que hoje é co-fundador e diretor de design da Primo, empresa que produz o brinquedo. O conceito é inspirado na obra de Seymour Papert. Pode-se dizer que o brinquedo é uma simplificação do ambiente Logo tradicional, criado por Papert, composto pela linguagem Logo e um tartaruga gráfica que responde aos comandos do usuário. No entanto, as instruções do Cubetto foram desenvolvidas de forma a evitar qualquer tipo de texto ou linguagem numérica [PRI15].

Os Artigos citados mostram, através de várias analises, em diversas áreas do conhecimento e níveis de ensino, que os problemas encontrados, tanto pelo educador quanto pelo educando, no ensino e aprendizagem das tecnologias. Assim como nesse estudo os autores encontram nas Tecnologias de Informação e Comunicações o recurso necessário para ajudar a mudar esse cenário, mesmo sendo utilizado apenas como apoio no processo, minimizando dessa forma a dificuldade no ensino e aprendizagem. É inegável que os processos que envolvem o ensinar e o aprender as tecnologias devem ser questionados. Neste caso questionamos: Até que ponto o uso de um software (compilador) poderia facilitar a aplicação do ensino de circuitos eletrónicos standard nas escola de base?

As farramentas acima citadas todas elas focam-se no desenvolvimento dos alunos de base usando as TIC´s. O ardlight entra na mesma linha de pensamento, juntamente com as farramentas já existente vai proporcionar aos educadores mais uma opção no desenvpolvimento de circuíto standard de uma forma simples, como consequência o aumento do enteresse dos educandos por essas áreas e também o contacto desde cedo em matérias que no medelo de ensíno actual têm sido muito tarde.

2.2 Ocaml

O OCaml é uma linguagem de programação de propósito geral, com ênfase na expressividade e segurança. Desenvolvido há mais de 20 anos na Inria por um grupo de pesquisadores, ele possui um sistema de tipo avançado que ajuda a detetar seus erros. Ele é usado em ambientes em que um único erro pode custar milhões e acelerar problemas, é suportado por uma comunidade ativa e possui um conjunto avançado de bibliotecas e ferramentas de desenvolvimento. Apesar de todo o seu poder, o OCaml também é bastante simples, o que é uma das razões pelas quais ele é frequentemente usado como uma linguagem de ensino [XL12].

Forças

A linguagem OCaml oferece: Um poderoso sistema de tipos, equipado com polimorfismo paramétrico e inferência de tipos. Por exemplo, o tipo de uma coleção pode ser parametrizado pelo tipo dos seus elementos. Isso permite definir algumas operações em uma coleção, inde-

pendentemente do tipo de seus elementos: classificar uma matriz é um exemplo. Além disso, a inferência de tipos permite definir tais operações sem ter que fornecer explicitamente o tipo de seus parâmetros e resultados. De dados algébricos definíveis pelo usuário e correspondência de padrões. Novos tipos de dados algébricos podem ser definidos como combinações de registos e somas. As funções que operam sobre essas estruturas de dados podem, então, ser definidas por correspondência de padrões, uma forma generalizada da conhecida instrução switch, que oferece uma maneira limpa e elegante de examinar e nomear dados simultaneamente. Gerenciamento automático de memória, graças a um coletor de lixo rápido, discreto e incremental. Compilação separada de aplicativos independentes. Compiladores portáteis de bytecode permitem a criação de aplicativos independentes a partir dos programas Caml Light ou OCaml. Uma interface de função externa permite que o código OCaml interaja com o código C quando necessário. O uso interativo do OCaml também é suportado por meio de um loop "read-evaluate-print" [XL12]. Além disso, o OCaml apresenta: Um sofisticado sistema de módulos, que permite organizar os módulos de forma hierárquica e parametrizar um módulo em vários outros módulos. Uma camada expressiva orientada a objetos, com várias classes de herança, paramétricas e virtuais. Compiladores de código nativo eficientes. Além de seu compilador bytecode, o OCaml oferece um compilador que produz código de máquina eficiente para muitas arquiteturas.

Uma linguagem de programação amplamente utilizada.

A linguagem OCaml foi inicialmente usada para desenvolver aplicações que envolvem computação simbólica: provadores automáticos de teoremas, compiladores e intérpretes, analisadores de programas, etc. Agora ela é usada para desenvolver software em muitas outras áreas de aplicação. Isso é ilustrado por nossa lista de histórias de sucesso selecionadas. A linguagem OCaml é amplamente usada para ensinar programação. Também é usado em projetos acadêmicos na Europa, Ásia e nas Américas. Várias grandes corporações desenvolvem projetos industriais significativos no OCaml: entre eles estão a Dassault Systèmes, a Microsoft, a IBM e o CEA (Comissariado da Energia Atómica). O Consórcio OCaml oferece aos parceiros industriais e acadêmicos uma estrutura formal para participar do desenvolvimento, manutenção e evolução do OCaml. Os parceiros também desfrutam de condições de licenciamento muito liberais em relação à distribuição de software do OCaml [XL12].

2.3 Visão geral da linguagem e das ferramentas do OCaml

2.3.1 Segurança

O OCaml é uma linguagem segura. Programas são verificados pelo compilador antes de poderem ser executados. Isso exclui muitos erros de programação, como, por exemplo, confundir um inteiro e um ponteiro ou acessar um campo inexistente em um registo. Mais precisamente, isso protege a integridade dos dados manipulados por um programa OCaml. Embora o OCaml seja estaticamente verificado, ele não exige que os tipos de parâmetros de função, variáveis locais, etc. sejam explicitamente declarados, ao contrário, digamos, C ou Java. Muitas das informações de tipo necessárias são inferidas automaticamente pelo compilador.

2.3.2 Funções

O OCaml é uma linguagem de programação funcional: não há restrições quanto à definição e uso de funções. Em outras palavras, funções são valores comuns: uma função pode ser passada como um argumento para uma função ou retornada por uma função.

2.3.3 Características imperativas

O OCaml oferece uma gama completa de recursos imperativos. Em particular, variáveis, arrays e componentes de registo podem ser declarados como modificáveis. Várias variedades de loops estão disponíveis.

Recuperação de erro

O OCaml apresenta um mecanismo geral de exceção para sinalizar e manipular erros e situações excepcionais.

2.3.4 Tipos de dados

O OCaml oferece vários tipos de dados integrados, incluindo:

tipos básicos: inteiros, números de ponto flutuante, booleanos, caracteres, strings.

tipos de dados mais sofisticados: tuplos, arrays, listas, conjuntos, tabelas de hash, filas, pilhas, fluxos de dados.

Além desses poderosos tipos internos, o OCaml oferece poderosos meios de definir novos tipos de dados: registos, enumerações e tipos de soma. Tipos de soma são uma generalização simples e poderosa de enumerações. Eles permitem reunir valores heterogéneos em um tipo comum, graças ao uso de tags conhecidas como construtores de dados. Os tipos definidos dentro de um módulo podem ser tornados visíveis para os clientes do módulo de forma concreta ou abstrata, como o programador deseja. Quando um tipo é exportado de maneira concreta, sua definição completa permanece conhecida: portanto, os clientes dos módulos podem construir ou examinar valores desse tipo. Por outro lado, quando um tipo é exportado de maneira abstrata, apenas seu nome é conhecido fora do módulo. Torna-se então impossível, de fora, criar ou inspecionar valores desse tipo. Esse mecanismo fornece um controle refinado sobre o encapsulamento de dados, que é obrigatório para programação em grande escala [XL12].

2.3.5 Aptidão para computação simbólica

OCaml apresenta correspondência de padrões, uma generalização do construtor de análise de caso tradicional. A correspondência de padrões oferece uma maneira limpa e elegante de examinar e nomear dados simultaneamente. O compilador OCaml aproveita esse recurso para executar várias verificações: são detetadas e reportadas as coordenadas supérfluas e ausentes, o que geralmente permite identificar erros sutis. Quando nenhum erro é sinalizado, pode-se confiar que nenhum caso foi negligenciado.

Correspondência de padrões traz conforto e segurança inigualáveis para programas que manipulam dados simbólicos.

2.3.6 Polimorfismo

O OCaml apresenta um sistema de tipos polimórficos: alguns tipos indeterminados podem ser representados por variáveis, que podem ser instanciadas posteriormente à vontade. Assim, por exemplo, uma única função de classificação pode ser aplicada a listas de inteiros, listas de pares inteiros ou listas de registos, sem exigir qualquer duplicação de código.

2.3.7 Programação em larga escala

Um programa OCaml é feito de unidades de compilação que são tratadas separadamente pelo compilador. Este esquema é totalmente compatível com o uso de ferramentas tradicionais de gerenciamento de projetos, como o utilitário “make”. O sistema de módulos é poderoso e seguro: cada interação entre módulos é tipicamente verificada. No OCaml, um módulo pode conter submódulos, o que permite organizar os módulos hierarquicamente, e é possível parametrizar um módulo em vários outros módulos, ou, em outras palavras, definir funções de módulos para módulos.

2.3.8 Programação orientada a objetos

O OCaml permite escrever programas em um estilo orientado a objetos. De acordo com a filosofia da linguagem, a camada orientada a objeto obedece ao paradigma de “tipagem forte”: assim, é impossível enviar uma mensagem para algum objeto que não possa respondê-la. Essa segurança não tem um custo expressivo: graças a características como herança múltipla e classes paramétricas, os padrões de projeto mais complexos podem ser expressos de maneira natural.
Gerenciamento automático de memória e coleta de lixo incremental.
O OCaml oferece gerenciamento automático de memória: a alocação e a desalocação de estruturas de dados é implícita e tratada pelo compilador: não existe um operador “novo”, “malloc”, “excluir” ou “livre”. Isso torna os programas muito mais seguros: a corrupção de memória não pode ocorrer. Além disso, o gerenciador de memória é incremental: ele é executado em paralelo com o aplicativo, para que a coleta de lixo não cause atrasos visíveis.

2.3.9 Regime de avaliação

A estratégia de avaliação do OCaml é rigorosa. Um regime de avaliação preguiçoso pode ser simulado, quando necessário, por meio de suspensões explícitas. Assim, é possível, por exemplo, manipular fluxos de dados potencialmente infinitos. Compilador eficiente, código compilado eficiente O OCaml oferece dois compiladores em lote: um compilador de código de bytes e um compilador de código nativo. Ambos suportam compilação separada. Caml Light oferece apenas um compilador bytecode. Os compiladores bytecode geram executáveis pequenos e portáveis. Além disso, esses compiladores são extremamente rápidos. O compilador de código nativo produz um código de máquina mais eficiente, cujo desempenho atende aos mais altos padrões de compiladores modernos [XL12].

2.3.10 Interatividade

O OCaml fornece um loop interativo de ”leitura e impressão de provas”, que é conveniente tanto para aprendizado quanto para testes rápidos e depuração de programas. Na verdade, não é necessário criar um arquivo ou inserir instruções de exibição nos programas, pois os resultados são automaticamente impressos pelo sistema interativo.

2.3.11 Depurando instalações

Vários métodos diferentes estão disponíveis para depurar programas OCaml: o sistema interativo oferece um método elementar, porém rápido e simples, para testar (pequenas) funções: Primeiro, digita várias entradas diretamente no sistema interativo e verifica se os resultados

são os esperados. Para casos mais complexos, o sistema interativo fornece um meio barato de seguir a computação, usando o chamado mecanismo de rastreamento de chamadas de função. Por último, o OCaml replay debugge é uma ferramenta extremamente poderosa para acompanhar a computação. Ele permite parar o programa a qualquer momento para examinar o valor de variáveis e a pilha de funções de chamada, e até mesmo voltar ao passado para retomar a execução em um determinado ponto de interesse [XL12].

2.3.12 Análise

O OCaml oferece ferramentas de geração de analisadores análogas a “lex” e “yacc”. Além disso, oferece um tipo embutido de fluxos de dados que facilita a definição de parsers de descida recursiva LL (1).

2.3.13 Portabilidade

O OCaml é executado em uma ampla variedade de plataformas. As plataformas oficialmente suportadas são descritas aqui e outras plataformas são suportadas pela comunidade. Por exemplo, os aplicativos OCaml estão disponíveis na loja de aplicativos da Apple; A compilação para o iOS é descrita aqui . Também é possível compilar para javascript com js-of-ocaml, permitindo a criação de aplicativos avançados do lado do cliente diretamente com o OCaml. Um exemplo notável disso é o Try OCaml, o resultado da compilação do próprio compilador OCaml com js-of-ocaml. Finalmente, o ocamljava permite a compilação direta do OCaml para bytecode Java, além de fornecer outros mecanismos para interoperar com o Java.

2.4 História do Ocaml

”Caml” foi originalmente um acrônimo para Categórico Abstract Machine Language. Foi um trocadilho no CAM, na Categórica Máquina Abstrata, e na ML, a família de linguagens de programação à qual o Caml pertence. O nome Caml permaneceu durante a evolução da linguagem, mesmo que a presente implementação não tenha relação com a CAM. Caml foi projetado e implementado pela equipe de Inria Formel , liderada por Gérard Huet. Seu desenvolvimento continuou dentro da equipe Cristal e seu atual sucessor, Gallium [XL12].

2.4.1 A origem

A equipe da Formel se interessou pela linguagem ML em 1980-81. ML foi a meta-linguagem da versão de Edimburgo do assistente de prova do LCF, ambos concebidos por Robin Milner. Foi implementado por um tipo de intérprete escrito em Lisp por Mike Gordon, Robin Milner e Christopher Wadsworth. O próprio LCF foi escrito parcialmente em ML e parcialmente em Lisp. Para poder usar o assistente de prova LCF nos vários sistemas em uso na Formel naquela época (Multics, Berkeley Unix na Vax, Symbolics), Gérard Huet decidiu tornar a implementação da ML compatível com vários compiladores Lisp (MacLisp, FranzLisp, LeLisp, ZetaLisp). Este trabalho envolveu Guy Cousineau e Larry Paulson. O desempenho da implementação do ML foi aprimorado pela adição de um compilador.
Guy Cousineau também adicionou tipos de dados algébricos e correspondência de padrões, seguindo idéias de Robin Milner, que ele, por sua vez, pegou emprestado da Hope, uma linguagem

de programação projetada por Rod Burstall e Dave McQueen. Em algum momento, essa implementação foi chamada Le-ML, um nome que não sobreviveu. Foi usado por Larry Paulson para desenvolver o Cambridge LCF e por Mike Gordon para a primeira versão do HOL, conforme recordado na curta história de HOL de Gordon.
Por volta de 1984, três eventos nos motivaram a nos envolver ainda mais no desenvolvimento do ML:

- Em Edimburgo, Luca Cardelli desenvolveu uma implementação muito mais rápida de ML usando sua Functional Abstract Machine (FAM). Ele também projetou sua própria versão da linguagem, conhecida na época como ML de Cardelli.

- Robin Milner achou que era um bom momento propor uma nova definição de ML para evitar divergências entre várias implementações. Ele definiu o núcleo da linguagem ML Standard, que foi posteriormente complementada por um sistema de módulos projetado por Dave McQueen.

- Ao mesmo tempo, Pierre-Louis Curien desenvolveu um cálculo de combinadores categóricos, bem como uma correspondência entre lambda-calculus e categorical combinators, que, como notado por Guy Cousineau, poderia ser visto como uma técnica de compilação para ML. Na verdade, foi bastante próximo da técnica de implementação original do Edinburgh ML, mas pode ser descrito, comprovado e otimizado de maneira simples. Isso levou à definição da máquina abstrata categórica (CAM).

Isso levou Guy Cousineau a desenvolver uma nova implementação do ML, com base no CAM. No entanto, a linguagem que acabamos implementando não era o padrão ML, mas Caml. Por quê? Nossa principal razão para desenvolver o Caml foi usá-lo para o desenvolvimento de software dentro da Formel. Na verdade, ele foi usado para desenvolver o sistema Coq, que, seguindo a tese de Thierry Coquand em 1985, tornou-se o principal objetivo da equipe. Relutamos em adotar um padrão que nos impedisse mais tarde de adaptar a linguagem às nossas necessidades de programação. Em particular, Philippe Le Chenadec e Michel Mauny desenvolveram ferramentas de manipulação de sintaxe que pareciam úteis e foram incorporadas ao Caml. A sincronização com a equipe do Standard ML antes de adotar as modificações de linguagem que nos pareciam úteis teria introduzido muito atraso em nosso trabalho. Além disso, nossa filosofia estava em conflito com a de uma linguagem "padrão", que não deveria evoluir muito rapidamente. Incorporamos em Caml a maioria das melhorias trazidas pela Standard ML sobre Edinburgh ML [XL12].

2.4.2 A primeira implementação

A primeira implementação de Caml apareceu em 1987 e foi desenvolvida até 1992. Foi criada principalmente por Ascander Suarez. Depois que Ascander saiu em 1988, Pierre Weis e Michel Mauny continuaram com o desenvolvimento e manutenção do sistema. Esta implementação compilou o Caml para o LLM3, a máquina virtual do sistema Le-Lisp.
Guy Cousineau lembra modestamente: "Devo admitir que, quando o desenvolvimento do Caml começou, minha experiência com a implementação da linguagem de programação foi muito limitada. Baseando-se na máquina abstrata LLM3 e no sistema de alocação de memória e coleta de lixo Le-Lisp, economizamos muito trabalho, mas não conseguimos obter alta eficiência. O modelo CAM levou à rápida construção do fechamento e ao bom compartilhamento do ambiente, mas foi deficiente no acesso ao ambiente e tornou as otimizações difíceis. Também introduziu potencialmente vazamentos de memória, uma vez que valores inúteis foram mantidos dentro de

fechamentos. Além disso, não havia percebido que era mais importante ter bom desempenho em programas não funcionais do que em programas muito funcionais. Acima de tudo, eu havia negligenciado a importância da portabilidade e abertura. Apesar dessas insuficiências, pelas quais sou inicialmente responsável, Ascander, Pierre e Michel fizeram um bom trabalho" [XL12].

2.4.3 Caml Light

Em 1990 e 1991, Xavier Leroy projetou uma implementação completamente nova do Caml, baseada em um intérprete de bytecode escrito em C. Damien Doligez forneceu um excelente sistema de gerenciamento de memória. Essa nova implementação, conhecida como Caml Light, foi altamente portátil e facilmente executada em pequenas máquinas desktop, como Macs e PCs. Substituiu a antiga implementação de Caml e ajudou muito a promover o uso de Caml na educação e nas equipes de pesquisa. Seu suporte para fluxos de dados e suas facilidades de análise, devido a Michel Mauny, foi emitido a partir de um esforço contínuo da equipe Formel para promover ferramentas de manipulação de sintaxe [XL12].

2.4.4 Caml Special Light

Em 1995, Xavier Leroy lançou o Caml Special Light, que melhorou sobre a Caml Light de várias maneiras. Primeiro, um compilador de código nativo otimizado foi adicionado ao compilador bytecode. Este compilador de código nativo igualou ou excedeu os desempenhos dos melhores compiladores existentes para linguagens funcionais, e permitiu que o Caml fosse mais competitivo em relação ao desempenho com linguagens de programação imperativas mais comuns, como o C ++. Em segundo lugar, a Caml Special Light ofereceu um sistema modular de alto nível, projetado por Xavier Leroy e inspirado no sistema de módulos do Standard ML. Este sistema de módulos oferece poderosos recursos de abstração e parametrização para programação em grandes [XL12].

2.4.5 Objetive Caml

Sistemas de tipos e inferência de tipos para programação orientada a objetos tem sido uma área quente de pesquisa desde o início dos anos 90. Didier Rémy, mais tarde acompanhado por Jérôme Vouillon, projetou um sistema de tipo elegante e altamente expressivo para objetos e classes. Este projeto foi integrado e implementado no Caml Special Light, levando à linguagem e implementação do Objective Caml, lançado em 1996 e renomeado para OCaml em 2011. O Objective Caml foi a primeira linguagem a combinar o poder da programação orientada a objetos com o ML- digitação estática de estilo e inferência de tipo. Ele suporta muitos idiomas avançados de programação OO (classes tipo paramétricas, métodos binários, especialização mytype) de maneira estaticamente segura, enquanto esses idiomas causam insanidade ou exigem verificações em tempo de execução em outras linguagens OO, como C ++ e Java.

Em 2000, Jacques Garrigue ampliou o Objective Caml com vários novos recursos, que ele vinha experimentando há alguns anos no dialeto da Objetiva Caml. Entre essas características estavam métodos polimórficos, argumentos de função rotulados e opcionais e variantes polimórficas [XL12].

2.4.6 A ascensão do OCaml

Desde o final da década de 1990, o OCaml vem ganhando popularidade e atraiu uma base de usuários significativa. Além dos impressionantes programas desenvolvidos no OCaml, a comunidade de usuários também contribuiu com muitas bibliotecas, frameworks e ferramentas de alta qualidade em áreas que variam de interfaces gráficas de usuário e vinculação de bancos de dados a programação de rede e Web, interoperabilidade entre linguagens e análise estática de programas. Em paralelo, a equipe central de desenvolvimento do OCaml mantém ativamente o sistema básico, melhorando a qualidade da implementação e transportando-a para as arquiteturas e sistemas mais recentes. Como desenvolvedor líder do OCaml, presidente do Caml Consortium e proprietário do OCaml.org, Xavier é considerado um benevolente ditador vitalício (BDFL) da linguagem OCaml [XL12].

2.4.7 Alguns parentes próximos

Além dessas versões mainstream do Caml, deve-se mencionar muitos compiladores relacionados. Michel Mauny e Daniel de Rauglaudre projetaram o Chamau, que oferece recursos exclusivos de manipulação de sintaxe que agora são oferecidos no pré-processador Camlp4 para o OCaml.
Manuel Serrano e Pierre Weis criaram o BIGLOO. Régis Cridlig fez Camlot. Jean Goubault-Larrecq escreveu HimML, que apresenta operações implícitas de hash-consing e eficientes em conjuntos e mapas. Emmanuel Chailloux publicou a CeML. Na equipe do Pará, Francis Dupont implementou Caml para máquinas paralelas, enquanto Luc Maranget criou o Gaml, um compilador para uma linguagem de programação preguiçosa e funcional [XL12].

2.4.8 Citação final

Em 1996, Guy Cousineau escreveu: “Certamente, a história de Caml poderia ter sido mais linear. No entanto, por meio de tentativa e erro, surgiu na França uma capacidade de produzir implementações funcionais de linguagem de programação de alto desempenho, portáteis e flexíveis” [XL12].

2.5 Arduíno

O Arduíno foi criado em 2005 na Itália pelo professor Massimo Banzi e o pesquisador David Guartielles. Banzi procurava um meio barato e que tornasse fácil o trabalho com tecnologia por seus alunos de design de iteração. Em conjunto, eles resolveram criar um micro controlador que pudesse ser utilizado nos projetos dos estudantes de arte e design. Guartielles desenhou a placa e um aluno de Massimo, David Mellis, programou o software para executa-la. A facilidade no uso e o seu baixo custo, fez com que a primeira tiragem da plataforma fosse rapidamente vendida. Hoje já são cerca de 300 mil unidades vendidas pelos diversos distribuidores espalhados pelo mundo [EVA13].
Atualmente, existem vários modelos de placas Arduíno disponíveis no mercado. Todos os modelos consistem em uma pequena placa composta de um microprocessador Atmel AVR, um cristal ou oscilador (relógio simples que envia pulsos de tempo em uma frequência especificada, para permitir sua operação na velocidade correta) e um regulador linear de 5 volts [Mic11]. Diversos dispositivos externos, como motores, relés, sensores luminosos, diodos a laser e alto-falantes podem ser conectados ao Arduino [MON13]. Para adicionar mais funcionalidades ao Arduino,

pode-se ainda utilizar placas especializadas conhecidas como shields. As shields são placas de circuito contendo outros dispositivos como, por exemplo, Global Positioning System (GPS), Liquid Crystal Display (LCD), módulos de Ethernet, entre outros [Mic11]. Para o desenvolvimento deste trabalho foi utilizado o Arduino uno,conforme mostrado na figura 2.1.

Figura 2.1: Placa do Arduino Uno. Fonte:https://www.robocore.net

O Arduíno UNO é uma placa micro-controladora de código aberto amplamente utilizada, baseada no micro-controlador ATmega328P e desenvolvida pela Arduíno.cc. A placa é equipada com conjuntos de pinos de entrada / saída (E / S) digitais e analógicos que podem ter interface com várias placas de expansão (blindagens) e outros circuitos. A placa possui 14 pinos digitais e 6 pinos analógicos. É programável com o IDE do Arduíno (Integrated Development Environment) através de um cabo USB tipo B. Ele pode ser alimentado por um cabo USB ou por uma bateria externa de 9 volts, embora aceite tensões entre 7 e 20 volts. O design de referência de hardware é distribuído sob uma licença Creative Commons Attribution Share-Alike 2.5 e está disponível no site do Arduíno. Arquivos de layout e produção para algumas versões do hardware também estão disponíveis. ”Uno” significa um em italiano e foi escolhido para marcar o lançamento do Arduíno Software (IDE) 1.0. A placa Uno e a versão 1.0 do Arduíno Software (IDE) foram as versões de referência do Arduíno, agora evoluindo para versões mais recentes. A placa Uno é a primeira de uma série de placas USB Arduíno e o modelo de referência para a plataforma Arduíno. Os programas, no mundo Arduíno, são conhecidos como sketches e a programação dos mesmos é feita utilizando Wiring, uma linguagem de programação baseada em C/C++ [BAN16]. Todo sketch deve conter as funções void setup() e void loop(), conforme mostrado na figura 2.2.

```
Void setup (){
  pinMode ( 13, OUTPUT);
}
void loop () {
digitalWrite ( 13 , HIGH);
  delay (1000);
digitalWrite ( 13 , LOW);
  delay (1000);
}
```

Figura 2.2: sketch Blink do Arduino Uno

Exemplo do Arduíno

A finalidade do Arduino num sistema é facilitar a prototipagem, a implementação ou emulação do controle de sistemas interativos, a nível doméstico, comercial ou móvel. Com o arduino é possível enviar ou receber informações de basicamente qualquer sistema eletrônico, como identificar a aproximação de uma pessoa e variar a intensidade da luz do ambiente conforme a sua chegada. Ou abrir as janelas de um escritório de acordo com a intensidade da luz do sol e temperatura ambiente. na figura abaixo demostramos um exemplo de um programa feito em arduino, que conciste em acender um led D.10.

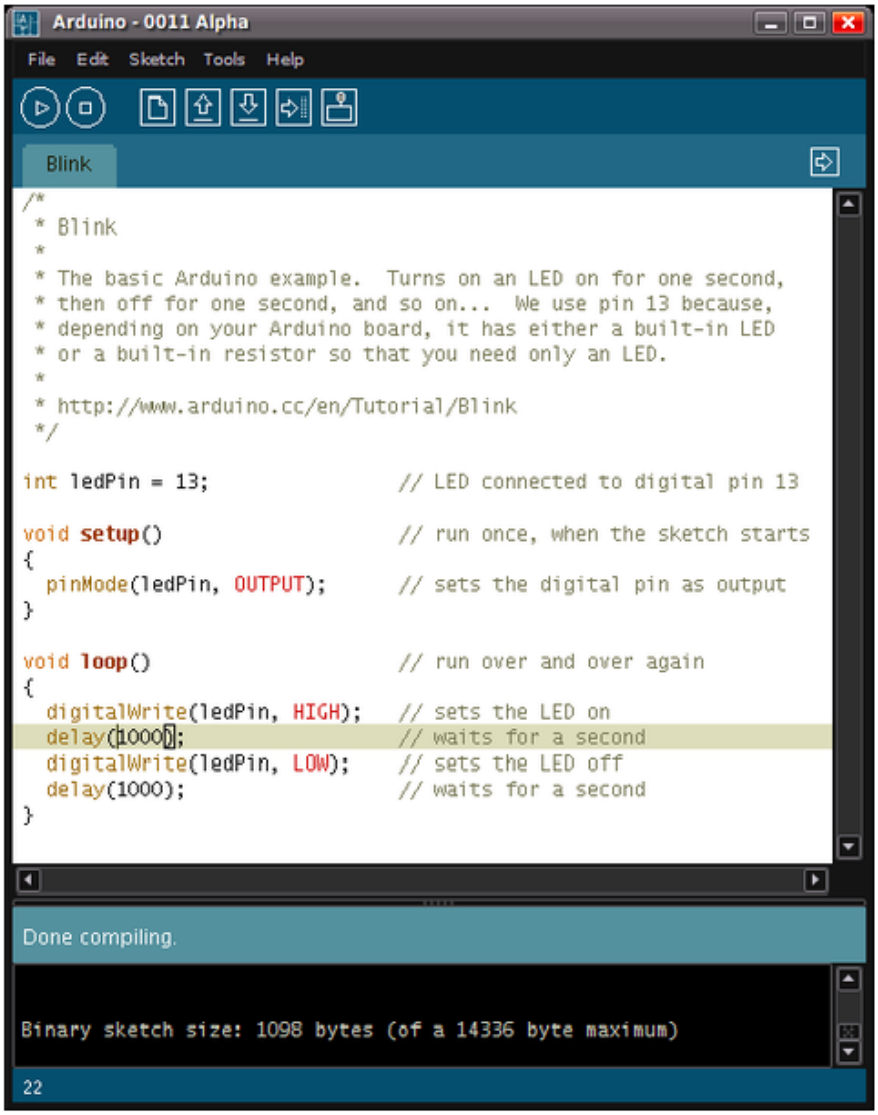

Figura 2.3: Exemplo de um programa feito em Arduino Uno. Fonte: <https://pt.wikipedia.org>

Com esse exemplo, queremos assim dizer que o ardlight (o compilador programado por nós) deve produzir um ficheiro igual a figura acima referida, quando ela tiver como parametro de entrada os mesmos componentes.

Capítulo 3

Ardlight, Linguagem e compilação

Com os objetivos traçados, partimos para a solução do problema antes proposto em construir um compilador tendo como suporte a placa do arduíno.

3.0.1 Descrição da linguagem do ardlight

Uma linguagem de programação é um método padronizado para comunicar instruções para um computador. É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador. Permite que um programador especifique precisamente sobre quais dados um computador vai atuar, como estes dados serão armazenados ou transmitidos e quais ações devem ser tomadas sob várias circunstâncias. Linguagens de programação podem ser usadas para expressar algoritmos com precisão.

O conjunto de palavras (lexemas classificados em tokens), compostos de acordo com essas regras, constituem o código fonte de um software. Esse código fonte é depois traduzido para código de máquina, que é executado pelo microprocessador.

Ao escrever um ficheiro em ardlight (num ficheiro do tipo adl) deve-se sempre começar com a palávra reservada **device** juntamente com uma **chaveta aberta**. Na parte declarativa os nomes dos dispositívos (que representam os típos de dados), usar conforme descritas na báse de dados em csv, começando sempre com a letra maiúscula, seguido do nome a declarar. deve-se terminar sempre com **chaveta fechada**. A figura abaixo mostra um exemplo 3.1.

```
devices {
Led vermelho;
Led azul;
Led amarelo;
MotorDC motor;
Potentiometer botao;
ServoMotor servomotor;
MicromotorDC micromotor;
}
```

Figura 3.1: exemplo da descrição da linguagem do Ardlight

A partir deste ficheiro acima referido, o Ardlight processa a informãçao contida nela, e identifica o dispositivo em questão atravéz do nome. Se o dispositívo em questão, tráta-se de um dispositivo digital ou analógico. Após a sua identificação o Ardlight, atribui uma porta de acordo ao seu tipo (analógico ou digital). Faz a leitura dos dados contidos na base de dado em csv e obtem as outras caracteristicas do dispositivo. Com esses dados o Ardlight constroi uma estrutura de dados em C capaz de ser lida pelo arduino. como mostra a figura abaixo 3.13.

É de salientar que o ardlight compila apenas a parte declarativa do ficheiro de entrada. E o código em C é reproduzido tal como está no outro lado da saída.

```
int vermelho = 6;
int azul = 7;
int amarelo = 8;
int motor = 9;
int botao = 0;
int servomotor = 10;
int micromotor = 11;
void setup() {
pinMode ( vermelho, OUTPUT);
 pinMode ( azul, OUTPUT);
 pinMode ( amarelo, OUTPUT);
 pinMode ( motor, OUTPUT);
 pinMode ( botao, INPUT);
 pinMode ( servomotor, OUTPUT);
 pinMode ( micromotor, OUTPUT);
 }
```

Figura 3.2: Ficheiro de Saída do Ardlight

3.0.2 Lexer

Como primeira etapa do nosso compilador partimos para a construção de Lexer, onde se realizará a análise léxica que é o processo de analisar as entradas de linhas de caracteres (assim como trabalha o código fonte de um programa de computador) e produzir uma serie de símbolos chamado ”símbolos léxicos” ou (lexical tokens em inglês), ou somente ”símbolos” (tokens), que podem ser manipulados facilmente por um parser que é um leitor de saída.
A Análise Léxica é a forma usada em compiladores para verificar determinado alfabeto. Quando analisamos uma palavra, o lexer pode através da análise léxica definir se existe ou não algum carácter desta palavra que não faça parte do nosso alfabeto, ou um alfabeto inventado por nós. Logo após a analise léxica, virá a análise sintática. A figura abaixo mostra como esta construido o nosso lexer 3.3.
O analisador léxico apresentado na fig funciona de duas maneiras: Na primeira etapa lê a entrada de caracteres, um de cada vez, mudando o estado em que os caracteres se encontram. Quando o analisador encontra um carácter que ele não identifica como correto, ele o chama de ”estado morto” então, ele volta à última análise que foi aceita e assim tem o tipo e comprimento do léxico válido.
Um léxico, entretanto, é uma única lista de caracteres conhecidas de ser um tipo correto. Para construir um símbolo, o analisador léxico necessita de um segundo estado.
Segundo estado da análise Nesta etapa são repassados os caracteres do léxico para produzir um valor. O tipo do léxico combinado com seu valor é o que adequadamente constitui um símbolo, que pode ser dado a um parser. (Alguns símbolos tais como parênteses não têm valores, e então a função da análise não pode retornar nada).

3.0.3 Parser

Como primeira etapa do nosso compilador partimos para a construção de um Parser que tem como finalidade processar e analisar sequencialmente as entrada dos símbolos (lida de um arquivo de computador ou do teclado, por exemplo) para determinar a estrutura da gramática, segundo uma determinada gramática formal. E Essa análise tambem faz parte do compilador ardlight, junto com a análise lexical e análise semântica.
A análise sintática contída no ardlight transforma um texto na sua entrada em uma estrutura de dados, em geral uma árvore sintática, o que é conveniente para oprocessamento futuro. E armazena a hierarquia implícita desta entrada. Através da análise lexical obteremos um grupo

```
   open Parser
   open Devices
   open Util
   exception Eof

let incr_linenum lexbuf =
let pos = lexbuf.Lexing.lex_curr_p in
lexbuf.Lexing.lex_curr_p <- { pos with
Lexing.pos_lnum = pos.Lexing.pos_lnum + 1;
Lexing.pos_bol = pos.Lexing.pos_cnum;
}
}
    rule lexer  = parse
     | [' ' '\t' '\r' ]          { lexer lexbuf }
     | '\n'                      { incr_linenum lexbuf ; lexer lexbuf }
     | ['A'-'Z' 'a'-'z'] ['A'-'Z' 'a'-'z' '0'-'9' '_'] * as id
                                 {  try
                                   Hashtbl.find Devices.dispositivos id
                                   with Not_found ->   Tident id}
     | "{"                       { Tlb}
     | "}"                       { remaining lexbuf ; Trb}
     | ";"                       { Tpv}
     | ","                       { Tc}
     | _ as c                    {
               let pos = lexbuf.Lexing.lex_curr_p in
               let ln = pos.Lexing.pos_lnum in
               let cl = pos.Lexing.pos_cnum in
                 prerr_string ("Lexical Error line "^
                                 (string_of_int ln)^
                                 " column "^
                                 (string_of_int cl)^
                                 " with "^String.make 1 c^"\n");
                                 lexer lexbuf}

and remaining = parse
| eof                             { () }
| _ as c                          { Buffer.add_char  fich_c c  ;  remaining lexbuf}
```

Figura 3.3: lexer do Ardlight

de tokens, para que o analisador sintático use um conjunto de regras para construir uma árvore sintática da estrutura.

Em termos práticos, por exemplo, podemos também usar a árvore sintática para decompor um texto em unidades estruturais para serem organizadas dentro de um bloco. A figura abaixo mostra como esta construido o nosso parser 3.4.

```
%{
  open Ast
  open Util
%}
%token <string> Tident
%token <string> Tdev
%token Tlb Trb Tpv Tc
%token Tdevices
%start main
%type <Ast.asa> main
%%


 main :  Tdevices Tlb d=decls; Trb  {to_single_ident (List.rev d)};

 decls : /*empty*/ {[]}
 |dc=decls; dl=decl; Tpv  {dl::dc};

 decl :
 di=Tdev; id=idlist; {di,id};


idlist :
vl=separated_nonempty_list(Tc, Tident); {vl};
```

Figura 3.4: parser do Ardlight

O analisador se trata de um software que realiza a função de carregar os dados de entrada e constrói com uma estrutura de dados com eles. Essa estrutura de dados pode se tratar de uma árvore de análise, árvore abstrata de sintaxe ou outras estruturas que dão ideia de hierarquia, para que resulte em uma representação estrutural da entrada que foi feita a análise. A análise pode proceder vários outros passos que são executados antes da própria análise, ou estes passos podem ser executados em um único passo, onde eles estarão combinados. Muitas vezes o processo realizado pelo analisador sintático é procedido pelo processo de análise lexical, já que esta análise gera como resultado uma tabela dos tokens dos dados de entrada analisados. Os analisadores podem ser programados manualmente, ou podem ser gerados automaticamente por um gerador de analisador.
A entrada de dados que é analisada pelo analisador é normalmente um código de uma linguagem de programação, mas podem ser também textos em linguagem natural, e nesse caso não é construída uma árvore de análise, mas só são extraídas algumas partes do texto. As funções dos analisadores variam desde analisar comandos simples de um código a programas muito complexos. Uma forma importante de realizar a análise é usando expressões regulares, onde uma expressão regular define uma linguagem regular e um mecanismo de expressão regular gerando automaticamente um analisador para a linguagem. Em alguns casos as expressões regulares são usadas antes da própria análise sintática, como etapa da análise lexical cuja saída será utilizada pelo analisador sintático.
Os analisadores sintáticos variam de acordo com as entradas que eles recebem, ou seja, da linguagem de programação que ele irá analisar. Para linguagens de dados o analisador é utilizado para facilitar a leitura de um programa, já para linguagens de programação o analisador faz parte de um compilador, que analisa o código para criar uma forma de representação interna. As linguagens de programação possuem uma gramática determinística, ou seja, que não possui ambiguidade, com isso é implementado o analisador sintático referente a está gramática. A análise para o compilador pode ser feita em uma ou múltiplas passagens pelo código.

3.0.4 Árvore sintática abstrata

Após a analise léxica e o processamento e analise da sequencia de tokens construimos a árvore sintática. Uma árvore de análise sintática, ou simplesmente árvore sintática, é uma estrutura de dados em árvore, que representa a estrutura sintática de uma cadeia de acordo com alguma gramática formal. Em uma árvore sintática, os nós internos são rotulados por símbolos não-terminais da gramática, enquanto os nós folha são rotuladas por símbolos terminais da gramática. Um programa que produz tais árvores é denominado um analisador sintático. Árvores sintáticas podem ser geradas para sentenças em linguagem natural como também durante o processamento de linguagens formais, tais como as linguagens de programação.É importante notar que a árvore sintática e a árvore sintática abstrata são estruturas de dados diferentes apesar de ambas serem relacionadas a construção de compiladores. A figura abaixo mostra como esta construido a árvore sintática 3.5.

```
type ident = string
type device = string
(*type decl = device * (ident list)*)
type decl = device * ident
type asa = decl  list
```

Figura 3.5: Árvore Sintatica do cabeçalho do programa Ardlight

3.0.5 Main e Util

O main e o util são Conjuntos de comandos agrupados em um bloco, que recebe um nome e através deste pode ser chamado. Para permitir o reaproveitamento de código já construído;

- Para evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa;
- Para permitir a alteração de um trecho de código de uma forma mais rápida.
- Para que os blocos do programa não fiquem grandes demais e, por conseqüência, mais difíceis de entender;
- Para facilitar a leitura do programa-fonte de uma forma mais fácil;
- Para separar o programa em partes(blocos) que possam ser logicamente compreendidos de forma isolada. (ver em anexo A1)

3.0.6 Devices e Dispositivos

Na secção devices serviu para construir o código que é capaz de fazer a leitura do ficheiro de uma base de dados do tipo csv, onde fica armazenado todos os dispositivos a utilizar no nosso compilador juntamente com as suas características. tambem faz o cálculo de cada dispositivo utilizado no ficheiro de entrada, a fim de recomendar se necessário, medidas para a proteção do mesmo dispositivo. visto que na placa arduino cada dispositivo precisa de uma porta tanto logica com analogica, então na secção dispositivos, criou-se uma estrutura de dado capaz de distinguir o tipo do dispositivo (analógico ou digital) e atribuí-lo a respectiva porta. A partir de um array de 19 posições foi então possivel fazer a distruição dos dispositivos cada um de acordo ao seu tipo, onde as primeiras 6 (seis) primeiras portas são atribuídas a dispositivos lógicos e o resto das portas a dispositivos analógicos. (ver em anexo B2 e B3)

3.0.7 CSV

Os arquivos CSV (Comma-separated values), são arquivos de texto, que faz uma ordenação de bytes ou um formato de terminador de linha, separando valores com vírgulas. Ele é comumente usado em softwares. A medida que que precisar-mos novos conponentes para a criação de circuitos standard, a nossa base de dado vai crescendo. A figura abaixo mostra como o csv do ardlight está definido 3.1.

Tabela 3.1: Csv do Ardlight.

Nome	Tipo	Intensidade	Tensão	Potência	Resistencia	Tipo de Dado	Ground
Led	Digital	25	1.8	0	220	OUTPUT	1
MotorDC	Digital	70	6	0	0	OUTPUT	1
ServoMotor	Digital	500	4.8	0	0	OUTPUT	1
MicromorDC	Digital	70	3.5	0	220	OUTPUT	1
Potentimeter	Analogic	0	0	0	0	INPUT	0
PushButton	Analogic	0	0	0	0	INPUT	0

3.0.8 Makefile

O objetivo de Makefile é definir regras de compilação para projetos de software. Tais regras são definidas em arquivo chamado Makefile. O programa make interpreta o conteúdo do Makefile e executa as regras lá definidas. Alguns Sistemas Operacionais trazem programas similares ao make, tais como gmake, nmake, tmake, etc. O programa make pode variar de um sistema a outro pois não faz parte de nenhuma normalização.
O texto contido em um Makefile é usado para a compilação, ligação(linking), montagem de arquivos de projeto entre outras tarefas como limpeza de arquivos temporários, execução de comandos, etc. Vantagens do uso do Makefile:

- Evita a compilação de arquivos desnecessários. Por exemplo, se seu programa utiliza 120 bibliotecas e você altera apenas uma, o make descobre (comparando as datas de alteração dos arquivos fontes com as dos arquivos anteriormente compilados) qual arquivo foi alterado e compila apenas a biblioteca necessária.
- Automatiza tarefas rotineiras como limpeza de vários arquivos criados temporariamente na compilação.
- Pode ser usado como linguagem geral de script embora seja mais usado para compilação.(ver em anexo C4)

3.1 Resultados Obtidos

Para validar o funcionamento Ardlight, quanto às instruções disponibilizadas pelo do compilador, foi necessário testa-lo com os mesmos parametros do exemplo acima referido no capítulo II, sobre o piscar de um led. Sendo assim, a figura abaixo mostra como esse ficheiro deve estar definido 3.6.

```
devices{
Led vermelho;
}
void loop() {
  digitalWrite(vermelho, HIGH);
       delay (2000);
  digitalWrite(vermelho,  LOW);
}
```

Figura 3.6: Ficheiro de entrada do teste do Ardlight

Com báse na entrada acima referida tivemos o mesmo resultado que o exemplo do capítulo II. ver na fígura abaixa 3.7.

```
int vermelho = 6;
void setup() {
pinMode ( vermelho, OUTPUT);
 }
void loop() {
  digitalWrite(vermelho, HIGH);
       delay (2000);
  digitalWrite(vermelho,  LOW);
}
```

Figura 3.7: Ficheiro de saída do teste do Ardlight

Uma das coisas importantes a fazer quando trabalhamos com a eletrônica é ter certeza de que estamos a proteger nossos componentes eletrônicos de forma segura. A maneira mais comum de fazer isso é adicionar um resistor em série. Podemos calcular o valor do resistor necessário para uma operação segura usando a lei de Ohm. Além de construir a estrutura de dados capaz de ser lida pelo arduino, o Ardlight cálcula de forma automática qual será a voltagem que o dispositivo precisa para o seu bom funcionamento e proteção. Tambem calcula a resistencia do dispositivo para a posterior dizer que resistencia usar para proteger o dispositivo de queimar. A potência necessária para se saber se a placa do arduino é capaz de suportar ou se será necessario usar uma fonte externa ou um relê. Como mostra a figura abaixo 3.8.

```
vermelho
a voltagem é igual a 2.700000 volt(s)
a resistência é igual a 0.108000 Ohm(s)
a corrente é igual a 0.012273 mA(s)
a potência é igual a 0.045000 Watt(s)
  Usar um resistor de 220   Ohm
```

Figura 3.8: Ficheiro de cálculos do ficheiro de entrada do Ardlight

Outro exemplo (Semáforo)

Exemplo de um semáforo feito em arduino. Como mostra a figura abaixo 3.12. Outros exemplos em anexo(ver anexo D1 e D2)

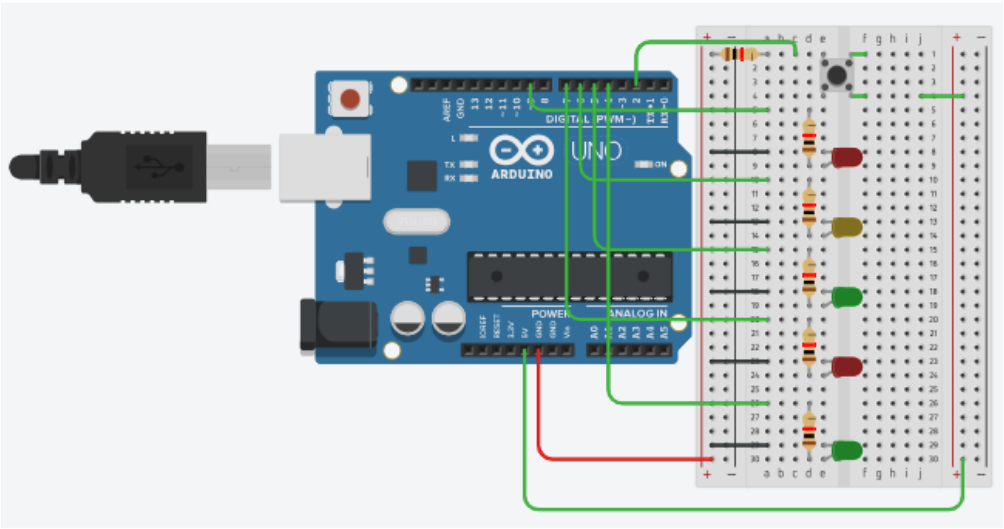

Figura 3.9: Circuíto do Semáforo

```
int vermelho      = 9;
int amarelo       = 6;
int verde         = 5;
int vermelhoPiao  = 7;
int verdePiao     = 4;
int botao         = 2;

void setup(){
  pinMode(vermelho    , OUTPUT);
  pinMode(amarelo     , OUTPUT);
  pinMode(verde       , OUTPUT);
  pinMode(vermelhoPiao, OUTPUT);
  pinMode(verdePiao   , OUTPUT);
  pinMode(botao       ,  INPUT);
}

void loop(){
int estadoBotao   = 0;
   digitalWrite(vermelhoPiao, HIGH);
   digitalWrite(verde       , HIGH);
estadoBotao = digitalRead (botao);
  if  ( estadoBotao == HIGH) {
      delay (2000);
         digitalWrite(verde       ,  LOW);
         digitalWrite(amarelo     , HIGH);
      delay ( 1500);
 digitalWrite(amarelo      ,    LOW);
 digitalWrite(vermelho     ,   HIGH);
      delay (1000);
 digitalWrite(vermelhoPiao ,    LOW);
 digitalWrite(verdePiao    ,   HIGH);
      delay ( 5000);
 digitalWrite(verdePiao    ,    LOW);
for( int contador = 0; contador <= 3; contador++ ) {
 digitalWrite(vermelhoPiao     ,   HIGH);
      delay (500);
digitalWrite(vermelhoPiao     ,   LOW);
      delay (500);    }
digitalWrite(vermelho    ,   LOW); }
}
```

Figura 3.10: Exemplo de Semáforo feito em Arduino

O mesmo exemplo (Semáforo) feito em Ardligth como mostra a figura abaixo 3.12.

```
devices {
Led vermelho;
Led verde;
Led amarelo;
Led vermelhoPiao;
Led verdePiao;
Potentiometer botao;
}
void loop(){
int estadoBotao   = 0;
   digitalWrite(vermelhoPiao, HIGH);
   digitalWrite(verde        , HIGH);
estadoBotao = digitalRead (botao);
 if  ( estadoBotao == HIGH) {
         delay (2000);
         digitalWrite(verde        ,  LOW);
         digitalWrite(amarelo      , HIGH);
         delay ( 1500);
 digitalWrite(amarelo      ,    LOW);
 digitalWrite(vermelho     ,   HIGH);
        delay (1000);
digitalWrite(vermelhoPiao ,    LOW);
digitalWrite(verdePiao    ,   HIGH);
       delay ( 5000);
digitalWrite(verdePiao    ,    LOW);
for( int contador = 0; contador <= 3; contador++ ) {
digitalWrite(vermelhoPiao     ,   HIGH);
         delay (500);
digitalWrite(vermelhoPiao     ,   LOW);
         delay (500);    }
digitalWrite(vermelho     ,   LOW); }
}
```

Figura 3.11: Ficheiro de entrada do Exemplo de Semáforo feito em Ardlight

Ficheiro de saída do Exemplo de Semáforo feito em Ardlight como mostra a figura abaixo 3.12.

```
int vermelho = 6;
int verde = 7;
int amarelo = 8;
int vermelhoPiao = 9;
int verdePiao = 10;
int botao = 0;
void setup() {
pinMode ( vermelho, OUTPUT);
 pinMode ( verde, OUTPUT);
 pinMode ( amarelo, OUTPUT);
 pinMode ( vermelhoPiao, OUTPUT);
 pinMode ( verdePiao, OUTPUT);
 pinMode ( botao, INPUT);
 }
void loop(){
int estadoBotao   = 0;
   digitalWrite(vermelhoPiao, HIGH);
   digitalWrite(verde        , HIGH);
estadoBotao = digitalRead (botao);
 if  ( estadoBotao == HIGH) {
         delay (2000);
         digitalWrite(verde        ,  LOW);
         digitalWrite(amarelo      , HIGH);
         delay ( 1500);
 digitalWrite(amarelo      ,    LOW);
 digitalWrite(vermelho     ,   HIGH);
        delay (1000);
digitalWrite(vermelhoPiao ,    LOW);
digitalWrite(verdePiao    ,   HIGH);
       delay ( 5000);
digitalWrite(verdePiao    ,    LOW);
for( int contador = 0; contador <= 3; contador++ ) {
digitalWrite(vermelhoPiao     ,   HIGH);
         delay (500);
digitalWrite(vermelhoPiao     ,   LOW);
         delay (500);    }
digitalWrite(vermelho    ,   LOW); }
}
```

Figura 3.12: Ficheiro de saída do Exemplo de Semáforo feito em Ardlight

Ficheiro de cálculos do semáforo do Ardlight como mostra a figura abaixo 3.13.

```
vermelho
a voltagem é igual a 2.700000 volt(s)
a resistência é igual a 0.108000 Ohm(s)
a corrente é igual a 0.012273 mA(s)
a potência é igual a 0.045000 Watt(s)
  Usar um resistor de 220   Ohm
verde
 a voltagem é igual a 2.700000 volt(s)
a resistência é igual a 0.108000 Ohm(s)
a corrente é igual a 0.012273 mA(s)
a potência é igual a 0.045000 Watt(s)
  Usar um resistor de 220   Ohm
amarelo
 a voltagem é igual a 2.700000 volt(s)
a resistência é igual a 0.108000 Ohm(s)
a corrente é igual a 0.012273 mA(s)
a potência é igual a 0.045000 Watt(s)
  Usar um resistor de 220   Ohm
vermelhoPiao
 a voltagem é igual a 2.700000 volt(s)
a resistência é igual a 0.108000 Ohm(s)
a corrente é igual a 0.012273 mA(s)
a potência é igual a 0.045000 Watt(s)
  Usar um resistor de 220   Ohm
verdePiao
 a voltagem é igual a 2.700000 volt(s)
a resistência é igual a 0.108000 Ohm(s)
a corrente é igual a 0.012273 mA(s)
a potência é igual a 0.045000 Watt(s)
  Usar um resistor de 220   Ohm
botao
 a voltagem é igual a 6.000000 volt(s)
a resistência é igual a inf Ohm(s)
a corrente é igual a inf mA(s)
a potência é igual a 0.000000 Watt(s)
  Usar um resistor de 220   Ohm
```

Figura 3.13: Ficheiro de cálculos do semáforo do Ardlight

Como mostram os exemplos apresentados, alcançamos com perfeição as mesmas saídas que as programandas de forma manual pelos programadores do arduino, com a vantagem de saber quais potências, registor, relê usar afim de proteger o nosso circuíto.

Capítulo 4

Conclusão e Trabalhos Futuros

4.1 Conclusão

Este trabalho apresentou o desenvolvimento de uma extensão, o ardlight, que teve como objetivo adicionar suporte ao Arduino, como plataforma de programação de circuítos através desta extenção. Neste sentido, foram feitas as adaptações necessárias para a ardlight possibilitar a contrução de um circuíto electrónico do tipo Arduino, bem como, a geração de código para a linguagem do Arduino a partir da linguagem ardlight. Também foi implementada os calculos necessários para a proteçõao dos circuitos electrónicos com base na placa Arduino UNO, sendo que o ardlight possui todos os recursos necessários para executar os componentes standard disponíveis na linguagem. No entanto, se os programas elaborados contiverem ações que dependam de componentes que não estiverem presentes na báse de dados em csv do ardlight, estes não serão executados. Com relação aos experimentos realizados, no que diz respeito à geração, implantação e execução dos circuítos electronicos no arduino, concluiu-se que os objetivos do trabalho foram alcançados. Embora não consiga demonstrar o desenho do circuito em forma de imagem, o compilador foi capaz de executar todos os comandos disponíveis na linguagem ardlight.

4.2 Trabalhos Futuros

Como trabalho futuro pretendemos continuar a aprimorar o ardlight, implementado mais funcionalidade. Por exemplo a implementação de uma farramenta para gerar imagem do circuíto electrónico atraves do ardlight, para facilitar a sua implementação.

Bibliografia

[BAN16] Massimo et al. BANZI. Arduino mega 2560 e genuino mega 2560. 2016. Available from: https://www.arduino.cc/en/Main/arduinoBoardMega2560. 14

[BEN08] Fabiane B. V. et al BENITTI. Bem vindo ao robolab: robótica educativa na furb. 2008. Available from: http://robolab.inf.furb.br/index.php?option=com_content&view=article&id=2&Itemid=8. 5

[eO04] Boratti e Oliveira. Introdução à programação – algoritmos. *Visual Books*, II:6, 2004. 2

[EVA13] Jordan; NOBLE Joshua EVANS, Martin; HOCHENBAUM. Arduino em ação. *Novatec*, page 25, 2013. 13

[GEN99] M.C. GENNARI. Dicionário de informática. *Editora Saraiva*, II:79, 1999. 2

[GIL99] A. C. GIL. Métodos e técnicas de pesquisa social. *Atlas*, page 69, 1999. 3

[GOM10] Cristiane G. et al GOMES. A robótica como facilitadora do processo de ensino-aprendizagem de matemática no ensino fundamental. *UNESP*, 2010. Available from: http://books.scielo.org/id/bpkng/pdf/pirola-9788579830815-11.pdf. 1

[HOS07] RAABE A. HOSTINS, H. Auxiliando a aprendizagem de algoritmos com a ferramenta webportugol,. *XIV Workshop sobre Educação em Computação (WEI 2007),*, 2007. 5

[MAR03] CORREIA L. H. A. MARTINS, S. W. O logo como ferramenta auxiliar no desenvolvimento do raciocíinio l´ogico: Um estudo de caso. *In Proceedings of International Conference on Engineering and Computer Education - ICECE.*, 2003. 5

[Mic11] MCROBERTS Michael. Arduino básico. *Novatec*, I:20–24, 2011. 1, 13, 14

[Mil78] Robin Milner. A theory of type polymorphism in programming. *J. Comput.Syst. Sci.s*, page 348–375, 1978. 2

[MON13] Simon MONK. Programação com arduino: começando com sketches. *Bookman.*, page 1, 2013. 13

[Pie99] Levy Pierre. Cibercultura. page 158, 1999. Available from: https://mundonativodigital.files.wordpress.com/2016/03/cibercultura-pierre-levy.pdf. 1

[PRI15] PRIMO. Cubetto playset manual. 2015. Available from: http://www.primotoys.com/docs/docs/cubetto-manual.htm. 6

[VAH08] Adilson et al VAHLDICK. Projeto furbot. *Blumenau*, 2008. Available from: http://www.inf.furb.br/poo/furbot/introducao.html. 6

[Xim93] F. Ximenes. Microsoft press dicionário de informática inglês-português português-inglês. 1993. 1

[XL12] Damien Doligez Xavier Leroy. Ocaml. *INRIA*, IV, 2012. Available from: https://caml.inria.fr/pub/docs/manual-ocaml/index.html. 6, 7, 8, 9, 10, 11, 12, 13

Apêndice A

Programação do Ardlight

A.1 Main e Util do Ardlight

Estas imagens abaixos apresentam o main e o util do Ardlight que previamente no corpo do do livro foi explicado.

```
open Printf
open Ast
open Lexing
open Util
open Dispositivo
let init_linenum lexbuf  filename =
  let pos = lexbuf.lex_curr_p in
  lexbuf.lex_curr_p <- { pos with
  Lexing.pos_fname = filename ;
  Lexing.pos_lnum = 1;
  Lexing.pos_bol = pos.pos_cnum; }
let print_position lexbuf =
  let pos = lexbuf.lex_curr_p in
  print_string "linha "; print_int pos.pos_lnum; print_newline ();
  print_string "coluna "; print_int ( pos.pos_cnum - pos.pos_bol )
let processar_asa asa =
    let rec processar a = match a with
      [] -> []
    | (d,id)::li ->
        let dispositivo = Dispositivo.device d, id in
        dispositivo::(processar li)
    in let dispositivos = processar asa in
let s = List.fold_left (fun acc (d, id) -> acc ^
Printf.sprintf "int %s = %d;\n" id d.porta) "" dispositivos in
    let s2 = List.fold_left (fun acc (d, id) -> acc ^
    Printf.sprintf "pinMode ( %s, %s);\n " id d.tipodado)
    (s ^ "void setup() {\n") dispositivos in
    s2 ^ "}"
let loop filename =
let inx = open_in filename in
    let lexbuf = Lexing.from_channel inx in
    let () = init_linenum lexbuf filename in
    try
    let dev = Parser.main Lexer.lexer lexbuf in
    let header =  (processar_asa dev) in
    let () = save  (filename^".c") header in
    (*** olhar para a AST, gerar o cabeçalho C adequado e
       gerar desenho do circuito. Esvrever tudo em dois ficheiros ****)
    close_in inx
    (* clpose_out fich.c, fich.desenho *)
    with _ ->
      let pos = lexbuf.Lexing.lex_curr_p in
      prerr_string (sprintf "Parser Error line %d  column %d.\n" pos.pos_lnum pos.pos_cnum)

(*let _ = Metapost.emit "circuitos" (Picture.scale (Num.bp 3.)
circuitos)*)
let _ = loop Sys.argv.(1)
```

Figura A.1: Main do Ardlight

```
let fich_c = Buffer.create 10

let save file header =
   let channel = open_out file in
   let () = output_string channel header in
   let () = Buffer.output_buffer channel fich_c in
          close_out channel


let string_of_decl (d,id) = (d^" : "^id)


let processar dev =
    String.concat "\n" (List.map string_of_decl dev)

 (*função processar
*)

let to_single_ident_aux d lid =
    List.map (fun el -> (d,el)) lid

let rec to_single_ident asa =
match asa with
  [] -> []
| (d,lid)::resto -> (to_single_ident_aux d lid)
                  @ to_single_ident resto
```

Figura A.2: Util do Ardlight

Apêndice B

Programação do Ardlight

B.1 Devices e Dispositivos do Ardlight

Estas imagens abaixos apresentam o Devices e o Dispositivos do Ardlight que previamente no corpo do do livro foi explicado.

```
open Parser
open Num
 let dispositivos = Hashtbl.create 53
  let _ =
    List.iter (fun (kwd, tok) -> Hashtbl.add dispositivos kwd tok)
[
"Led", Tdev "Led";"MotorDC", Tdev "MotorDC";
"ServoMotor", Tdev "ServoMotor";
"MicromotorDC", Tdev "MicromotorDC";
"Potentiometer",Tdev "Potentiometer";
"devices", Tdevices
]
type tipo = Analogico | Digital
type circuito = {
  nome : string;tipo : tipo;intensidade : int ;qt : float ;
  potencia : float ;
  resistencia : int ;tipodedado : string;graund    : bool; }
let ler_GND s = if s = "0" then false else true
       let ler_Tipo s =
        match s with
        | "Digital" -> Digital
        | "Analogic" -> Analogico
        | _ -> failwith "Tipo enexistente"
        let ler_int s =
        try
         ( int_of_string s)
        with
        Failure _ -> failwith " Valor não númerico "
        let ler_Float s =
        try
         ( float_of_string s)
        with
        Failure _ -> failwith " Valor não número "
let ler_Linha linha =
        { nome = List.nth linha 0;
        tipo = ler_Tipo (List.nth linha 1);
        intensidade = ler_int (List.nth linha 2);
        qt = ler_Float (List.nth linha 3) ;
        potencia = ler_Float (List.nth linha 4);
        resistencia = ler_int (List.nth linha 5);
        tipodedado = List.nth linha 6;
        graund = ler_GND  (List.nth linha 7);  }
let ler_fich nome =
       let csv = Csv.load ~separator:';' nome in
       List.map ler_Linha csv
let caracteristicas = Hashtbl.create 53
let _ = let csv = ler_fich "Base de dados devices.csv" in
List.iter (fun (d) ->
Hashtbl.add caracteristicas d.nome (d.tipo,d.intensidade,d.qt,
d.potencia,
d.resistencia,d.tipodedado,d.graund)) csv
let circuitos : (string, circuito) Hashtbl.t = Hashtbl.create 53
let convert nome (tipo,intensidade,qt,potencia,resistencia,tipodedado
,graund) =
  { nome;tipo;intensidade;qt;potencia;resistencia;tipodedado;graund }
```

Figura B.1: Device do Ardlight

```
let relatorio c  (*amp*) =
let() = if c.qt > c.potencia
    then  Printf.printf "energia fornecida:%f volt (s)\n
   é inferior do que a energia pretendida %f volt(s)\n"
   (c.qt -. c.potencia) c.qt  in
    let voltagem = c.potencia -. c.qt in
    let resistencia = voltagem /. ( float_of_int c.intensidade)in
    let intensidade = voltagem /. (float_of_int c.resistencia )in
let amper=(c.qt *.(((float_of_int c.intensidade)*.0.001)/.1.))in
(
 let () = Printf.printf "a voltagem é igual a %f volt(s)\n" voltagem in
 let () = Printf.printf "a resistência é igual a %f Ohm(s) \n"
 resistencia in
 let () = Printf.printf "a corrente é igual a %f mA(s)\n"
 intensidade in
 let() = Printf.printf "a potência é igual a %f Watt(s) \n" amper in
if  ( c.intensidade < 100 )
    then Printf.printf "  Usar um resistor de 220   Ohm \n " else
if  ( c.intensidade > 100 && c.intensidade <= 130 )
then Printf.printf "  Usar um rele de 5VDC 1A e uma Fonte externa \n"
else if (c.intensidade > 130 && c.intensidade <=170 )
then Printf.printf "  Usar um rele de 5VDC 2A e uma Fonte externa \n"
else if  (c.intensidade > 170 )
then Printf.printf "  Usar um rele de 5VDC 10A e uma Fonte externa \n"
else ()
)
```

Figura B.2: Device do Ardlight

```
let portas = Array.make 19 false;;
let dispositivos = Array.make 17 "";;
type device = {
    porta : int;
    nome : string;
    tipodado : string;
}
 let device nome =
let d = Hashtbl.find Devices.dispositivos nome in
let (tipo, intensidade,tensao,potencia,resistencia,tipodado,graund) =

Hashtbl.find Devices.caracteristicas nome in
    let () = Devices.relatorio (Devices.convert nome
    (tipo, intensidade,tensao,potencia,resistencia,tipodado,graund)) in
    let porta = ref (-1) in
    (match tipo with
    Devices.Analogico ->
      (let stop = ref false in
      for l = 0 to 6 do
        if (!stop = false && portas.(l) = false) then
          (stop := true;
          portas.(l) <- true;
          porta := l)
      done)
    | Devices.Digital ->
      (let stop = ref false in
      for l = 6 to 17 do
          if (!stop = false && portas.(l) = false) then
            (stop := true;
            portas.(l) <- true;
            porta := l)
      done););
    { porta = !porta; nome = nome; tipodado = tipodado }
```

Figura B.3: dispositivo do Ardlight

Apêndice C

Programação do Ardlight

C.1 Makefile

Esta imagen abaixo apresentam o makefile do Ardlight que previamente no corpo do livro foi explicado.

```
RESULT = ardlight
SOURCES = ast.ml util.ml parser.mly devices.ml lexer.mll
dispositivo.ml main.ml
OCAMLYACC= menhir
PACKS = csv
OCAMLMAKEFILE = OCamlMakefile
include $(OCAMLMAKEFILE)
```

Figura C.1: makefile do Ardlight

Apêndice D

Outros Exemplos

D.1 Circuítos feitos no Ardlight Led intermitente

Estas imagens abaixo apresentam exemplos de comparação de códigos do arduino aos do Ardlight.

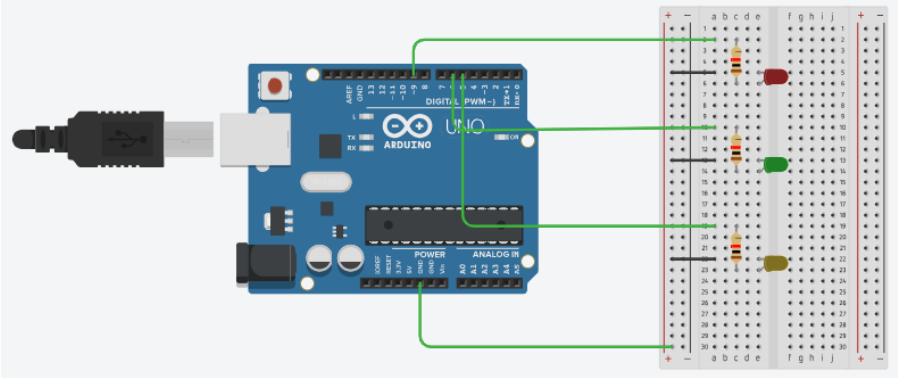

Figura D.1: Circuíto do Led Intermitente

```
int vermelho = 9;
int verde    = 6;
int amarelo  = 5;

void setup(){
  pinMode(vermelho, OUTPUT);
  pinMode(verde   , OUTPUT);
  pinMode(amarelo , OUTPUT);
}
void loop(){
  int brilho;
for( brilho = 0 ; brilho <= 255; brilho = brilho + 5) {
      analogWrite(vermelho , brilho);
      analogWrite(verde    , brilho);
      analogWrite(amarelo  , brilho);
    delay (30);}
for( brilho = 255 ; brilho >= 0; brilho = brilho - 5) {
      analogWrite(vermelho , brilho);
      analogWrite(verde    , brilho);
      analogWrite(amarelo  , brilho);
 }    delay (30);
}
```

Figura D.2: exemplo feito em arduino led intermitente

```
devices {
Led vermelho;
Led verde;
Led amarelo;
}
void loop(){
int brilho;
for ( brilho = 0 ; brilho <=255; brilho = brilho + 5){
       analogWrite (vermelho   , brilho);
       analogWrite (verde        , brilho);
       analogWrite (amarelo     , brilho);
delay (30);
}
for ( brilho = 255 ; brilho <=0; brilho = brilho - 5){
       analogWrite (vermelho   , brilho);
       analogWrite (verde        , brilho);
       analogWrite (amarelo     , brilho);
delay (30);
}
}
```

Figura D.3: exemplo do ficheiro de entrada ardlight led intermitente

```
int vermelho = 6;
int verde = 7;
int amarelo = 8;
void setup() {
pinMode ( vermelho, OUTPUT);
 pinMode ( verde, OUTPUT);
 pinMode ( amarelo, OUTPUT);
 }
void loop()
{
int brilho;
for ( brilho = 0 ; brilho <=255; brilho = brilho + 5){
       analogWrite (vermelho   , brilho);
       analogWrite (verde        , brilho);
       analogWrite (amarelo     , brilho);
delay (30);
}
for ( brilho = 255 ; brilho <=0; brilho = brilho - 5){
       analogWrite (vermelho   , brilho);
       analogWrite (verde        , brilho);
       analogWrite (amarelo     , brilho);
delay (30);
}
}
```

Figura D.4: exemplo do ficheiro de saída ardlight led intermitente

```
vermelho
a voltagem é igual a 2.700000 volt(s)
a resistência é igual a 0.108000 Ohm(s)
a corrente é igual a 0.012273 mA(s)
a potência é igual a 0.045000 Watt(s)
  Usar um resistor de 220   Ohm
verde
 a voltagem é igual a 2.700000 volt(s)
a resistência é igual a 0.108000 Ohm(s)
a corrente é igual a 0.012273 mA(s)
a potência é igual a 0.045000 Watt(s)
  Usar um resistor de 220   Ohm
amarelo
 a voltagem é igual a 2.700000 volt(s)
a resistência é igual a 0.108000 Ohm(s)
a corrente é igual a 0.012273 mA(s)
a potência é igual a 0.045000 Watt(s)
  Usar um resistor de 220   Ohm
```

Figura D.5: ficheiro de cálculos do ardlight para proteger o circuíto led intermitente

D.2 Circuítos feitos no Ardlight Micromotor DC 3V

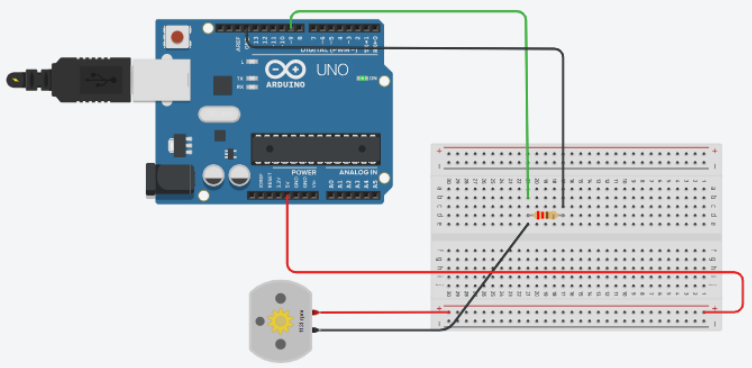

Figura D.6: Circuíto do Micromotor DC 3V

```
devices
 {
MicromotorDC motor;
}
void loop (){
digitalWrite (motor , HIGH);
delay (1000);
digitalWrite (motor , LOW);
}
```

Figura D.7: codígo de entrada do ardlight do circuíto do Micromotor DC 3V

```
int motor = 6;
void setup() {
pinMode ( motor, OUTPUT);
 }
void loop (){
digitalWrite (motor , HIGH);
delay (1000);
digitalWrite (motor , LOW);
}
```

Figura D.8: codígo gerado pelo ardlight do circuíto do Micromotor DC 3V

```
motor
a voltagem é igual a 1.500000 volt(s)
a resistência é igual a 0.021429 Ohm(s)
a corrente é igual a 0.006818 mA(s)
a potência é igual a 0.245000 Watt(s)
  Usar um resistor de 220   Ohm
```

Figura D.9: ficheiro de cálculos do ardlight para proteger o circuíto Micromotor DC 3V

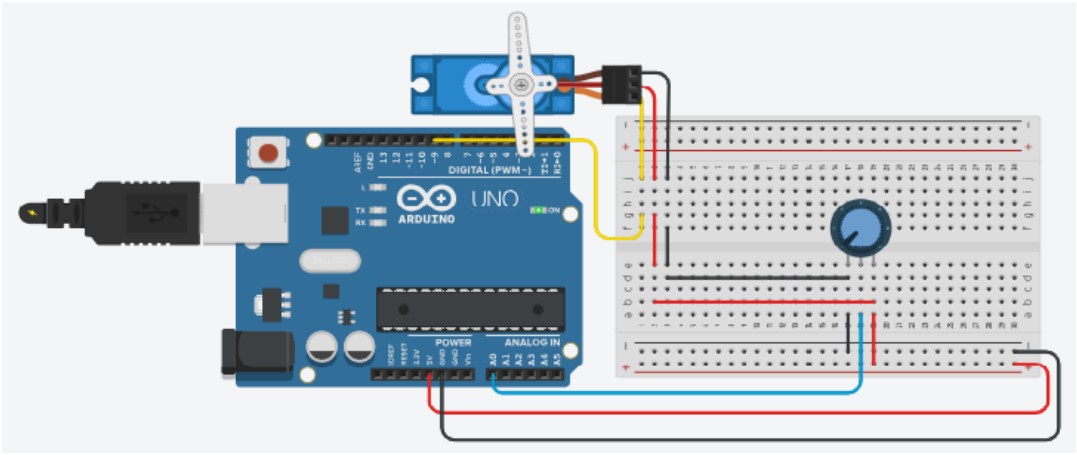

Figura D.10: Circuíto do Servo Motor

```
devices
 {
ServoMotor servomotor;
}
void loop(){
void servo0graus();
void servo90graus();
void servo180graus();

  for(char i=0;i<100;i++) servo0graus();
  for(char i=0;i<100;i++) servo90graus();
  for(char i=0;i<100;i++) servo180graus();
  for(char i=0;i<100;i++) servo90graus();     }

void servo0graus()                {
   digitalWrite(servo1, HIGH);
   delayMicroseconds(600);
   digitalWrite(servo1, LOW);
   for(int i=0;i<32;i++)delayMicroseconds(600); }
void servo90graus()  {
   digitalWrite(servo1, HIGH);
   delayMicroseconds(1500);
   digitalWrite(servo1, LOW);
   for(int i=0;i<12;i++)delayMicroseconds(1500);   }
void servo180graus()    {
   digitalWrite(servo1, HIGH);
   delayMicroseconds(2400);
   digitalWrite(servo1, LOW);
   for(int i=0;i<7;i++)delayMicroseconds(2400);     }
```

Figura D.11: codígo de entrada do ardlight do circuíto do Servo Motor

```
int servomotor = 6;
void setup() {
pinMode ( servomotor, OUTPUT);
 }
void loop(){
void servo0graus();
void servo90graus();
void servo180graus();

  for(char i=0;i<100;i++) servo0graus();
  for(char i=0;i<100;i++) servo90graus();
  for(char i=0;i<100;i++) servo180graus();
  for(char i=0;i<100;i++) servo90graus();    }

void servo0graus()              {
   digitalWrite(servo1, HIGH);
   delayMicroseconds(600);
   digitalWrite(servo1, LOW);
   for(int i=0;i<32;i++)delayMicroseconds(600); }
void servo90graus()  {
   digitalWrite(servo1, HIGH);
   delayMicroseconds(1500);
   digitalWrite(servo1, LOW);
   for(int i=0;i<12;i++)delayMicroseconds(1500);  }
void servo180graus()   {
   digitalWrite(servo1, HIGH);
   delayMicroseconds(2400);
   digitalWrite(servo1, LOW);
   for(int i=0;i<7;i++)delayMicroseconds(2400);    }
```

Figura D.12: codígo gerado pelo ardlight do circuíto do Servo Motor

```
servomotor
a voltagem é igual a 0.200000 volt(s)
a resistência é igual a 0.000400 Ohm(s)
a corrente é igual a inf mA(s)
a potência é igual a 2.400000 Watt(s)
  Usar um rele de 5VDC 10A e uma Fonte externa
```

Figura D.13: ficheiro de cálculos do ardlight para proteger o circuíto Servo Motor