# Apostila de Automação Industrial

João Paulo Cerquinho Cajueiro

11 de março de 2015

# Sumário

# Capítulo 1

# Sistemas de automação

A palavra automação vem do latim automatus – mover por si mesmo. Logo a automação de uma tarefa consiste em fazer esta tarefa ser realizada sem trabalho humano. Isto pode ser por diversos motivos: seja por que é uma tarefa perigosa e portanto queremos aumentar a segurança das pessoas, como num processo que envolva alta temperatura, por exemplo; seja para fazer a tarefa de forma mais rápida, seja para melhorar a qualidade do produto final ou seja porque simplesmente o custo do trabalho humano é muito elevado. Logo, podemos definir automação da seguinte forma:

> Automação é a substituição do trabalho humano para melhorar segurança, qualidade, produção e custos.

Neste contexto, automação industrial é nada mais que a automação de um sistema industrial, ou de um sistema de manufatura. Embora manufatura venha de fazer com as mãos, a revolução industrial mudou seu conceito, passando a significar a fabricação de praticamente qualquer produto. Do ponto de vista econômico, a manufatura é a transformação de materiais (matéria prima) em itens de maior valor (produto). Isto é conseguido por uma determinada sequência de processos químicos e físicos. De forma mais sucinta:

> Manufatura é a transformação de matéria prima em produtos pela aplicação de um ou mais processos.

Logo, a automação industrial consiste em fazer os processos necessários para a manufatura com o mínimo de esforço ou interferência humana, visando melhor segurança, qualidade, produção e custo. Note que por esforço humano, queremos dizer tanto esforço físico quanto mental, logo uma máquina bastante complexa mas que funcione a manivela, não se classificaria como automação; da mesma forma uma máquina que não exija esforço físico mas requer atenção constante também não é automatizada (seria apenas mecanizada).

## 1.1 História

É interessante pegar alguns pontos chaves na história da automação. Embora várias máquinas mecânicas de diversas graus de complexidade já existissem, como por exemplo relógios mecânicos desde o século VIII na china e desde o século XIII na Europa e os fantásticos robôs de Pierre Jaquet-Droz, do século XVIII, considera-se que a revolução industrial iniciou com a invenção do tear mecânico por Cartwright em 1785, que realiza um movimento relativamente complexo de forma automática a partir de uma roda d'água.

Um outro grande avanço ocorreu por volta de 1788, com a invenção do mecanismo de regulagem de fluxo de vapor de James Watt, o que permitia então controlar a potência de caldeiras e outras máquinas a vapor, controlando uma energia

muito maior que uma roda d'água. Isto foi um grande impulso para a mecanização, mas a verdadeira automatização ainda ficava muito restrita devido a dificuldade de realizar processos complexos de forma automática. Ou seja, retirava-se grande parte do esforço físico do homem, mas ainda era necessário muito esforço mental.

Em 1820 Babbage começou a desenvolver a sua máquina diferencial, que hoje chamaríamos de uma calculadora mecânica. Ela evoluiu até o conceito da máquina analítica, descrita em 1837, que é considerada o primeiro projeto de computador, embora apenas partes dela tenham sido efetivamente construídas.

Em 1880, Herman Hollerith criou um novo método baseado na utilização de cartões perfurados, para automatizar algumas tarefas de tabulação do censo dos EUA que antes duravam 10 anos. Com o método, o processo era concluído em 6.

Ao longo da primeira metade do século XX foram utilizados muitos sistemas eletromecânicos para o controle de processos industriais. Eram os chamados circuitos chaveados, que utilizavam relês para o controle lógico e para o comando de motores.

Em 1936, Alan Turing descreveu um *computador universal* em seu artigo "On Computable Numbers, with an Application to the Entscheidungsproblem", o que hoje é conhecido como uma Máquina de Turing. Suas idéias foram desenvolvidas em 1944, com a construção do Colossus, considerado como o primeiro computador, embora tivesse a função específica de quebrar o código criptográfico alemão na segunda guerra. Ele consistia de um circuito com 1600-2400 válvulas com capacidade de processamento de 25k caracteres/s. A titulo de comparação, os computadores de casa de hoje em dia atingem 10 bilhões de cálculos por segundo.

O avanço da eletrônica fez que que a capacidade de processamento dos computadores aumentasse de forma exponencial. Atualmente o mais rápido computador do mundo é o chinês Tianhe-2, que faz 33,86 quatrilhões de cálculos por segundo, consumindo 24MW.

Na década de 60 a General Motors fez uma especificação de um CLP – Controlador Lógico Programável, que é um computador voltado para o controle de processos industriais. Em 1968 foi criado o primeiro.

Hoje em dia toda automação está relacionada a sistemas computadorizados, seja em CLPs, CNC, robôs industriais, automação dos sistemas de apoio a produção, entre outros.

## 1.2 Classificação

Hoje em dia se definem, grosso modo, 3 tipos de automação, tal qual mostra a figura 1.1: fixa, flexível e programável.

A automação fixa é aplicada à produção de um único produto (ou com mínimas variações), em grandes quantidades: refinaria de petróleo, parafuso, tampas de

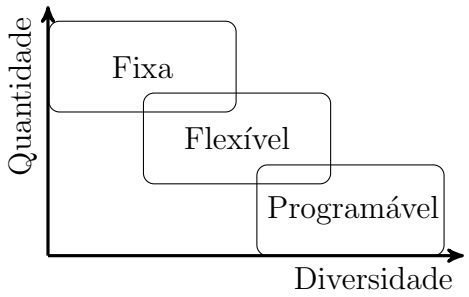



Figura 1.1: Tipos de automação industrial quanto a quantidade e diversidade de produtos.

garrafa, clips, biscoito, cerveja, etc. Ela utiliza equipamentos específicos para aquela tarefa, que portanto tem alto custo mas grande produtividade.

A automação flexível é aplicada à produção de produtos parecidos, em que pequenas modificações permitem a alteração do produto, como por exemplo mudança de um perfil a ser prensado ou extrudado ou a mudança das quantidades do mesmo conjunto de matérias primas (mudança de receita). Tipicamente é feita a chamada fabricação em lotes, onde entre um lote e outro se alteram as peças e/ou as sequências a serem seguidas de forma automática para ter o menor tempo parado possível. Exemplos são livros, circuitos integrados, potes de plástico, máquinas de café.

A automação programável é para produção de produtos diferentes mas cujo volume de produção não justifica um processo único. Ela usa máquinas de propósito geral, tais como robôs, ferramentas de controle numérico e impressoras 3d, onde a definição do processo é quase toda feita por *software*, de modo que o custo do maquinário é diluído em diversos produtos.

A tendência é cada vez mais ter a automação flexível e programável aumentando a capacidade de produção, de modo que a flexível vai ocupando nichos da fixa e a programável da flexível. Apesar disso, em vários casos é difícil imaginar alguns produtos deixando de utilizar a automação fixa.

## 1.3 Pirâmide de automação

A automação em larga escala de uma grande indústria ou de um conjunto de indústrias é mais complexa que a manufatura: envolve problemas de abastecimento, armazenagem, análise de mercado, exigências ámbientais, entre várias outras coisas. Uma forma de se separar os diferentes problemas da automação é através da chamada Pirâmide de Automação, mostrada na figura 1.2.

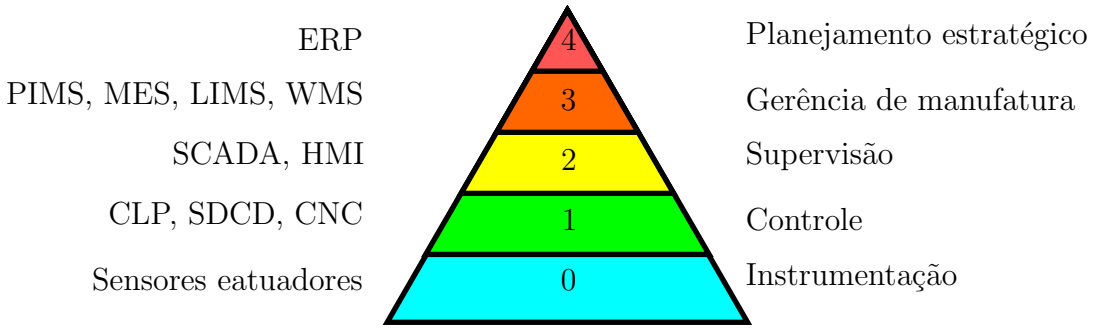



Figura 1.2: Pirâmide de automação.

Note que esta não é a única representação da pirâmide: umas começam pelo 1, outras tem apenas 4 camadas, e assim por diante, logo mais importante que o número de cada camada é o que tais camadas significam.

**Nível 0 :Instrumentação** Camada onde se encontram instrumentos, sensores, motores, máquinas, etc. Consistem nos equipamentos do chamado "*chão de fábrica*".

**Nível 1: Controladores** Controle automático da planta – onde se localizam os Controladores Lógico-Programáveis (CLP), os Sistemas Digitais de Controle Distribuído (SDCD), os Controles Numéricos Computadorizados (CNC) e/ou computadores de controle.

**Nível 2: Supervisão** Supervisão e controle do processo através de Interfaces Homem-Má quina (IHMs) ou SCADA (*Supervisory Control And Data Acquisition*).

**Nível 3: Gerenciamento da Manufatura** Gestão dos recursos da planta e controle da produção. Sistemas PIMS (*Process Information Management System*) e MES (*Manufacturing Execution Systems*).

**Nível 4: Gerenciamento da Empresa** Gestão dos recursos e produção da empresa como um todo. ERP – *Enterprise Resources Planning*.

Este texto faz um estudo da automação industrial de modo *top-down*: começando do nível 3 até o nível 0. O nível 4 é mais importante para um estudo de engenharia de processo e portanto não será abordado.

# Capítulo 2

# Gerenciamento da manufatura: PIMS e MES

PIMS – *Process Information Management System* e MES – *Manufacturing Execution System* são sistemas da camada 3 da pirâmide de automação, responsáveis pelo armazenamento e tratamento de dados do nível 2, concentrando os dados de diversos processos separados em um único ponto. São os chamados *middleware*, pois ficam a meio caminho entre os sistemas de gerenciamento de empresa e os supervisórios, às vezes combinando funções de um ou de outro.

De forma geral, no terceiro nível da pirâmide a preocupação é em consolidar os dados brutos do processo (*data*), para com eles gerar informações (*information*) e conhecimento (*knowledge*) sobre o processo, aumentando o valor destes valores, como mostra a figura 2.1. Os dados são obtidos ou do controlador ou do supervisório de um determinado processo. A relação entre dados ou a variação destes dados no tempo geram informação sobre a planta. A relação entre informações ou a variação de informações no tempo geram conhecimento.

Um exemplo desta relação é mostrado na figura 2.2. Nesta figura, a partir dos dados de temperatura e vazão de um fluido é gerada a informação do calor removido em determinado trocador de calor. A comparação dos calores removidos de diversos trocadores gera o conhecimento de qual trocador é mais eficiente.

## 2.1 PIMS

Para a finalidade de gerar informação e conhecimento, o ponto de partida é obter os dados brutos. Esta é a tarefa principal do PIMS: adquirir, armazenar e apresentar diversos dados de uma planta. O PIMS foi criado e ainda é principalmente usado para processos contínuos, tais como uma refinaria ou siderúrgica, e portanto tem um enfoque muito grande em variáveis analógicas e na relação delas com o tempo.

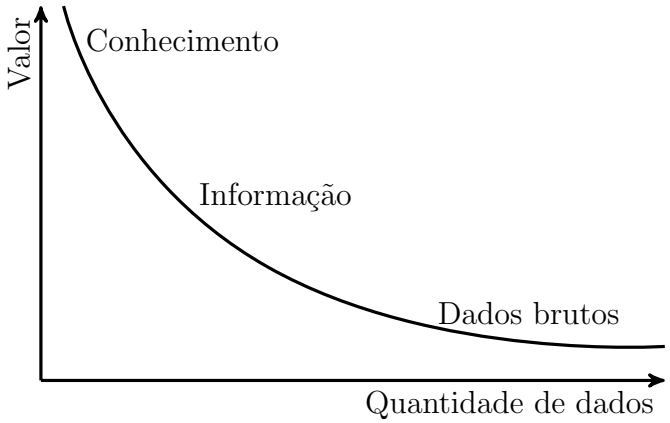



Figura 2.1: Relação entre dados, informações e conhecimentos.

Do ponto de vista do PIMS podemos esquematizar os sistemas de uma fábrica como mostra a figura 2.3, onde se vê que o PIMS tem 4 partes principais: um

**historiador de processos** que se comunica com vários sistemas do nível 1 (CLP, CNC) ou 2 (supervisório) ou ainda de outros sistemas nível 3, tais como um LIMS – *Laboratory Information Management System* ou MES para obter dados brutos dos diversos processos e acumula-os num

**banco de dados temporal** , que armazena os dados indexando-os pelo tempo. Uma

**interface gráfica** faz a recuperação e visualização destes dados, que ainda podem ir para

**aplicações clientes** com variadas funções, desde análise dos dados a interface com outros sistemas.

### 2.1.1 Historiador de Processos

O historiador do processo é necessário pelas seguintes funcionalidades:

**Registro Histórico** para análise de incidentes, controle de qualidade, métricas de performance, entre outros.

**Adequação a normas** como por exemplo para controle ambiental.

**Monitoração de equipamentos** para ontrole de vida útil e apoio à manutenção.

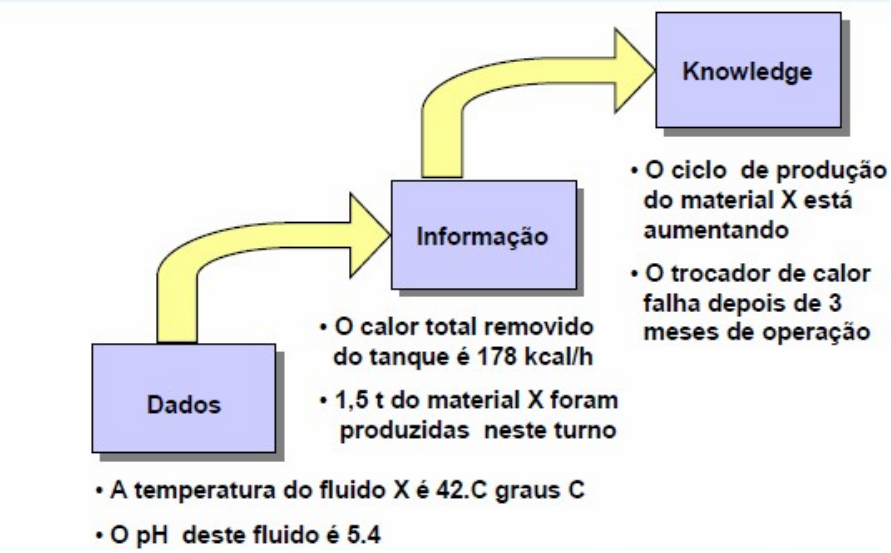



Figura 2.2: Exemplo da transformação de dados para informação e de informação para conhecimento.

**Análise de processo** , facilitando a visualização de dados e detecção de correlações.

Para tanto o historiador se comunica com sistemas do nível 2 (SCADA) e 1 (CLP) para armazenar os dados do processo. Estes dados são principalmente os valores das variáveis do processo, sejam discretos ou contínuos, mas também abarcam outras informações, tais como a ocorrência de alarmes, a marcação de que operador está presente, o período que um equipamento está ligado, entre outros. Cada uma desta isformações é identificada por um marcador único - a chamada *tag*, ao qual também está associado o endereço lógico de onde se obtém tal informação e o tempo em que tal dado foi gerado (*time stamp*). Em alguns casos se associa também uma métrica da qualidade do dado, referente a confiabilidade daquele dado, tal como se o instrumento de medida está calibrado ou não.

Estas informações podem ser obtidas tanto de sistemas SCADA ou de CLPs. Algumas vantagens de pegar informação dos sistemas nível 2 são que:

- o SCADA já converteu os dados para unidades de engenharia enquanto que em alguns CLPs os dados estão em valor bruto (de 0 a 4095);
- muitas variáveis são definidas apenas no sistema SCADA, não existindo nos CLPs, tais como o motivo de alarmes ou qual operador está monitorando a operação;

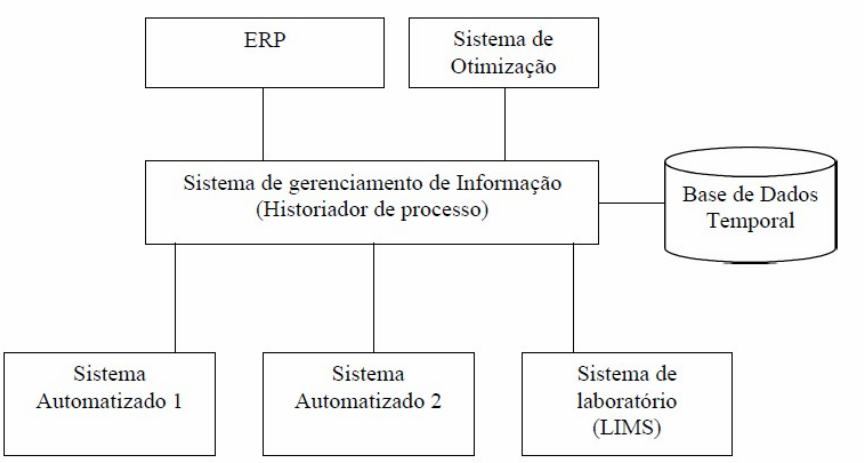



Figura 2.3: Sistema PIMS – REFAZER.

- interface com os sistemas SCADA costuma ser padrão, o que facilita a comunicação e interoperação.

Vantagens de obter as informações do CLP são:

- busca dos eventos com menor atraso temporal;
- pode-se coletar os dados em um ponto único, se todas as redes de CLPs estiverem interligadas;
- CLPs são mais confiáveis e apresentam menor suscetibilidade a falhas que os sistemas SCADA.

### 2.1.2 Banco de Dados Temporal

A maioria das análises realizadas nos dados de um sistema PIMS são em função do tempo, logo é comum ele usar um banco de dados que indexa a informação pelo tempo. Basicamente é uma tabela, relacionando *time stamp*, *tag*, tipo de dado (analógico, booleano, texto), valor e qualidade (se houver).

Um problema do uso deste tipo de banco de dados, ao invés dos chamados bancos de dados relacionais, que são bem mais comuns, é que a busca por informação pode ter uma baixa performance quando a quantidade de dados aumenta muito. Esta é a principal razão para que estes sistemas façam uma compressão de dados, que basicamente se resume a não armazenar dados que não tragam muita informação nova. A figura 2.4 mostra a idéia por trás da compressão de dados.

Algoritmos comuns para a compressão de dados no PIMS são *banda morta*, onde os dados são apenas armazenados se variarem mais do que um mínimo especificado;

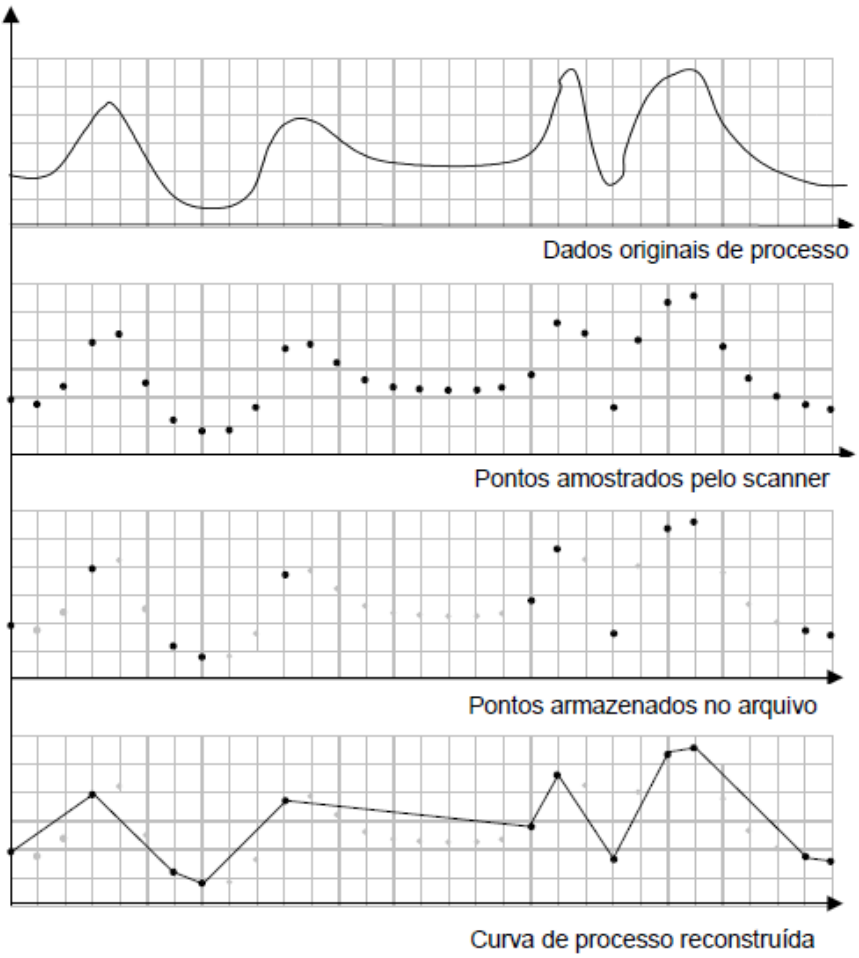



Figura 2.4: Processo de amostragem, compressão e reconstrução dos dados.

o *SDCA – Swinging Doors Compression Algorithm*, onde para cada valor recebido é definida uma reta entre ele e o último valor armazenado, descartando valores que possam ser definidos por esta reta e mais um erro; e o *boxcar/backslope*, que usa a banda morta e mais uma reta definida pelo último valor armazenado.

### 2.1.3 Interface Gráfica

A interface gráfica de um sistema PIMS é, em muitos aspectos, muito parecida com a de um sistema SCADA, contendo representações pictóricas do processo (sinóticos) com os valores de várias variáveis e gráficos de tendência. Tais elementos são melhor vistos no contexto de um sistema SCADA.

## 2.2 MES

Assim como o PIMS, um sistema MES também adquire dados do processo com o objetivo de apresentar uma visão geral do processo. Porém o MES tem uma função mais ativa que o PIMS, que é focado mais no armazenamento dos dados. Outra diferença importante é que os MES são usados principalmente em processos discretos, muitas vezes em sistemas de automação flexível ou programada, e tem que lidar com o sequenciamento dos processos, o que ocorre menos em sistemas contínuos.

Os sistemas MES agregam diversas funções de sistemas anteriores mais simples e específicos e são definidos por terem 11 funções:

**Gerenciamento das definições de produto.** Tudo o que é necessário para a fabricação do produto. Isto inclue armazenamento, lista de materiais e insumos, set-points do processo e receitas.

Por receita, entenda-se uma variação do processo para, na mesma máquina, produzir produtos diferentes ou variações dele. Por exemplo, num processo de pintura a receita diria quais as tintas usadas, em que sequência, em que volume e em que velocidade de aplicação. É muito importante na automação flexível e programável.

**Gerenciamento de insumos.** Permite preparar e executar ordens de produção com garantia de disponibilidade dos insumos.

**Agendamento de produção.** Permite determinar a ordem que a produção será feita, para alcançar os requerimentos de produção definidos pela ERP (camada 4 da pirâmide) utilizando otimamente os recursos.

Tipicamente inclui ferramentas de simulação, que permitem comparar diversas opções de ordens e estimar efeitos quando de mudanças imprevistas na sequência de produção (tipicamente por conta de uma parada não programada).

**Envio de ordens de produção.** Em função do agendamento feito, o MES cuida de enviar as ordens de produção para os diversos postos da planta.

**Acompanhamento da execução de ordens de produção.** Comunicação com sistemas níveis 1 e 2 para garantir a execução das ordens. Inclui também o registro de paradas.

O registro de paradas é feito automaticamente quando o equipamento para, seja por alguma condição espúria detectada no nível 1 ou por ação do operador no nível 2. Em ambos os casos fica registrado uma parada em aberto, que

só finaliza quando o operador complementar certas informações que auxiliam no diagnóstico da parada e o equipamento voltar a funcionar.

**Coleção dos dados de produção.** Equivalente ao historiador de processo do PIMS.

**Análise da performance da produção.** Cálculo dos chamados índices de produção – *KPI, Key Performance Indicators*. É a geração de informação útil a partir dos dados da produção.

Como exmplo de um KPI, temos o OEE – *Overall Equipment Effectiveness* – que aponta a efetividade de um determinado equipamento ou célula de produção. Este índice é dado por:

$$\text{OEE} = \text{disponibilidade} \times \text{performance} \times \text{qualidade}, \tag{2.1}$$

onde por disponibilidade entende-se a razão entre o quanto de tempo o equipamento funcionou e o quanto de tempo ele deveria ter funcionado, ou seja, descontando-se as paradas não programadas; performance é a razão entre a produção do equipamento enquanto funcionava e a capacidade de produção de que o equipamento é capaz, ou seja, descontando a ociosidade do equipamento; e qualidade é o valor do que o equipamento produziu em relação ao valor se não tivesse havido nenhum descarte ou geração de produtos de menor valor.

**Rastreamento da produção.** Permite levantar que produto ou lote foi feito quando e em qual equipamento. Útil para melhoria da produção e imprescindível para remédios e produtos alimentícios.

**Armazenamento dos logs de produção.** Hoje em dia tais logs são inseridos pelo operador no próprio sistema supervisório e realcionados às variáveis de produção.

**Interface de auditoria.** Permite a análise dos diversos dados e informações armazenados e o cruzamento destes dados com outras bases de dados.

## 2.3 Redes de Comunicação: Introdução e noções básicas

# Capítulo 3

# SCADA – Supervisory Control and Data Acquisition

## 3.1 Arquitetura de sistemas SCADA e interfaceamento com níveis de automação

## 3.2 Funcionalidades principais de sistemas SCADA

# Capítulo 4

# Controladores Lógico-Programáveis (CLP)

# Capítulo 5

# Programação - arduino

O arduino é uma plataforma de microcontrolador simplificada. Arduino é: uma placa com um microcontrolador Atmel, uma linguagem de programação e um ambiente de desenvolvimento para esta linguagem.

A placa Arduino tem um conector USB para se ligar ao computador. Isto serve tanto para programar o microcontrolador quanto para comunicação entre os 2. Existem várias versões do arduino. Neste curso serão usados arduinos UNO ou outras placas compatíveis com o original e arduinos nano, que são feitos para ser encaixados em protoboards.

O arduino UNO tem 4 barras de pinos fêmeas para conexão com outros dispositivos: uma com tensões de alimentação (POWER), um com 6 entradas analógicas (ANALOG IN) e 2 com um total de 14 entradas e saídas digitais (DIGITAL). O arduino NANO tem todas estas entradas e saídas em pinos barras de pinos macho.

A linguagem de programação arduino é basicamente a linguagem C++ para microcontroladores, mas com algumas funções e definições que facilitam sua implementação.

A principal diferença da linguagem arduino é que não existe a função main(), mas sim as as funções (ou rotinas) setup() e loop(). A função setup() é executado apenas uma vez no momento que o arduino é ligado (ou resetado) e depois o código dentro da função loop() é executado repetidamente. Com isto o esqueleto de um programa arduino fica:

```
void setup() {
  //codigo a ser executado no inicio
}

void loop() {
  //codigo a ser executado repetidamente
}
```

## 5.1 Piscando um led.

Vamos passar logo a um exemplo para analisar um programa arduino. As placas de arduino tem já nela um led ligado ao pino 13, identificado por um L, logo podemos fazer um programa que faça este led piscar.

```
void setup() {
  //codigo a ser executado no inicio
  pinMode(13,OUTPUT); //define o pino 13 como uma saida
}

void loop() {
  //codigo a ser executado repetidamente
  digitalWrite(13,LOW);   //apaga o led
  delay(500);             //espera meio segundo
  digitalWrite(13,HIGH); //acende o led
  delay(500);             //espera meio segundo
}
```

A chamada pinMode(13, OUTPUT); serve para definir que o pino 13 será uma saída. Obviamente isto só precisa ser feito no início do programa, logo está dentro de setup(). Se quiséssemos ter uma entrada digital, usaríamos a mesma função, mas trocando OUTPUT por INPUT: pinMode(pino,INPUT);.

A função que define o valor de um pino digital é a digitalWrite(pino, valor). Ela é chamada duas vezes dentro de loop(), uma para apagar o led (gravando LOW) e outra para acendê-lo (gravando HIGH). Se quiséssemos saber o valor de um pino digital que tivesse sido definido como entrada, a função seria digitalRead(pino)\ lstinline , que retornaria o valor digital naquele pino.

Um detalhe é que o microcontrolador do arduino funciona numa velocidade de 8 ou 16 MHz, logo se colocássemos apenas as duas chamadas à função digitalWrite não veríamos o led piscar, mas teríamos a impressão que ele está aceso com metade da intensidade. Para vermos o led piscar é necessário colocar um atraso, que é justamente obtido pela função delay(tempo), que gera um tempo morto de quantos milisegundos se colocar.

## 5.2 Sinais analógicos

Em contraste com os sinais digitais, os sinais analógicos são aqueles que podem assumir qualquer valor de tensão. No contexto do arduino, vamos por enquanto assumir que os sinais analógicos estão entre 0 V e 5 V.

Para valores analógicos, usamos as funções analogWrite(pino,valor) e analogRead(pino), que, ao contrário das equivalentes digitais, são restritas a alguns pinos específicos.

As entradas analógicas são identificadas pelos pinos ANALOG IN (A0 a A5 no arduino) e são ligadas a um conversor analógico/digital (A/D) do microcontrolador, que transforma estes sinais numa palavra binária de 10 bits. Como $2^{10} = 1024$, isto significa que a função analogRead retorna um valor entre 0 (para uma entrada de 0 V) e 1023 (para uma entrada de 5 V).

O arduino não tem um conversor D/A, logo a função analogWrite não gera um sinal analógico verdadeiro no pino. O que esta função faz é gerar um sinal modulado por largura de pulso - PWM (*Pulse Width Modulation*).

O sinal PWM é um trem de pulsos digital, com frequência da ordem de 500 Hz (no caso do arduino) cuja razão entre o tempo em alto e o perído (conhecida como *duty cycle*) pode ser alterada pelo parâmetro passado, como mostra a figura **??**. Se um sinal PWM é enviado a um pino com um led, ele piscará 500 vezes por segundo, o que é muito rápido para o olho humano, de modo que na prática o que se vê quando se varia o duty cycle de um sinal PWM que aciona um led é uma variação de sua intensidade. Logo um sinal PWM funciona, para muitas aplicações, como um sinal analógico.

Novamente, não são todos os pinos do arduino que conseguem gerar este sinal PWM, logo a função analogWrite\lstinline está restrita aos pinos 3, 5, 6, 9, 10 e 11. Um outro detalhe que vale a pena ressaltar é que enquanto a função analogRead gera um valor entre 0 1023, a analogWrite recebe como parâmetro um valor entre 0 e 255 apenas.

Uma função útil do arduino para lidar com este tipo de situação é a função map(valor, minEnt, maxEnt, minOut, maxOut), que faz uma transformação linear de valor de acordo com a seguinte equação:

$$(\text{valor} - \text{minIn}) \times \frac{\text{maxOut} - \text{minOut}}{\text{maxIn} - \text{minIn}} + \text{minOut}$$

A partir desta função, um código que leia o valor gerado pelo potenciômetro (em A0) e controle a intensidade do led no pino 5 poderia ser simplesmente:

```
analogWrite(5,map(analogRead(A0),0,1023,0,255));
```

Note que o valor lido pelo analogRead não precisa ser usado apenas na função analogWrite mas pode ser usado para outra finalidade, como por exemplo alterar um atraso.

## 5.3 Controle: for e if

Até aqui foram feitos programas puramente sequenciais, porém em vários momentos é interessante realizar operações repetidas vezes ou realizar algumas tarefas apenas em situações específicas. Para estes casos existem comandos como o **for**

e o **if**. O comando for serve para tarefas repetidas. Por exemplo, se quiséssemos inicializar os pinos de 2 a 10 como saídas com valor LOW, poderíamos usar o seguinte código:

```
for(int i = 2; i<11; i++){
        pinMode(i,OUTPUT);
        digitalWrite(i,LOW);
}
```

O que este código faz é definir uma variável local i com valor inicial 2 depois ele checa se i é menor que 11, se for ele executa os comandos que estão entre as chaves "{" e "}", incrementa a variável i (i++) e checa novamente. Quando $i < 11$ for falso, ele sai do laço.

O comando if executa um determinado código apenas se determinada condição for verdadeira. Por exemplo, para acender um led apenas se um sinal analógico for maior que a metade da escala (512 = 1024/2) pode-se escrever:

```
if(analogRead(A0) >= 512){
        digitalWrite(13,HIGH);
}
```

Note que este código apenas fará alguma coisa se a condição for verdadeira. Para fazer uma coisa OU outra usa-se o comando **else** após o **if**:

```
if(analogRead(A0) >= 512){
        digitalWrite(13,HIGH);
} else {
        digitalWrite(13,LOW);
}
```

## 5.4 Comunicação com o computador

Como já dito, a conexão USB do arduino serve também para a comunicação do mesmo com o computador. Do lado do computador o arduino aparece como uma porta serial e o próprio programa contém um terminal serial pelo qual é possível se comunicar com o arduino. Do lado do arduino, os pinos 0 e 1 são os pinos transmissor e receptor ligados ao USB .

A programação é feita através do objeto Serial. Este objeto tem vários comandos, porém para nós os que interessam neste momento são:

**Serial.begin(baud)** Inicializa a comunicação serial na velocidade (baud rate) indicada.

**Serial.read()** Retorna o valor de um byte recebido ou -1 caso não tenha sido recebido nenhum byte.

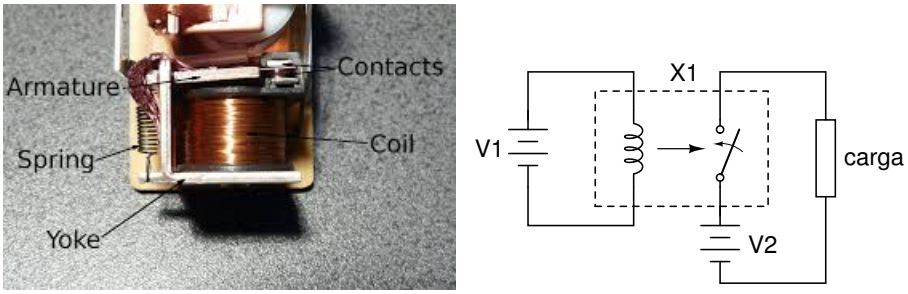
**Serial.print(dado) e Serial.println(dado)** Se dado for um char ou uma string (texto), envia dado. Se dado for um número, envia este número como uma string. No caso de println, é acrescentada uma quebra de linha após dado.

**Serial.available()** Retorna o número de bytes recebidos que ainda não foram lidos.

## 5.5 De relês a CLPs

Antes do desenvolvimento de CLPs, a lógica de operação de sistemas industriais era feita através de dispositivos eletromecânicos chamados relês, que são que chave controladas eletricamente. Ainda hoje há várias aplicações que utilizam relês, principalmente quando se trabalha com altas tensões e correntes e em baixa velocidade. Um elemento muito usado na indústria é o contator, que nada mais é do que um relê trifásico.

Um relê eletromecânico é um chave que se mantém em uma determinada posição (fechada ou aberta) pela ação de uma mola e que muda de estado (abre ou fecha) pela ação de um eletroímã. Ou seja, ao se magnetizar este eletroímã o relê abre (fecha) e volta a fechar (abrir) sem esta magnetização. Eletricamente pode-se visualizar um relê apenas como um indutor e uma chave, e muitas vezes é desta forma que ele é representado em um diagrama esquemático (vide exemplo na Figura 5.1(b)).




(a) Foto de um relê (b) Diagrama esquemático

Figura 5.1: Foto de um relê e representação esquemática.

O estado de repouso define o tipo básico do relê: normalmente aberto ou normalmente fechado. Também é comum relês que tenham 2 ou mais chaves acionadas por uma mesma bobina, inclusive podendo ser uma normalmente aberta e outra normalmente fechada, tal como o relê da figura 5.1(a).

A parte de acionamento de um relê (a bobina) é eletricamente isolada do contato. Esta característica é interessante para separar o circuito de controle do circuito de acionamento, o que permite o acionamento de cargas de alta tensão e alta corrente a partir de um circuito de baixa tensão. Um exemplo de aplicação de relê é mostrado na figura 5.2, onde 2 relês c1 e c2 são usados para o acionamento de um motor trifásico em comandado pelas botoeiras b0, b1 e b2.

Em geral é mais fácil analisar um circuito do ponto de vista lógico separando a parte de acionamento da parte de controle, então é comum que o símbolo de um relê seja separado em duas partes: a bobina (o eletroímã) e o contato (a chave), interligados pelo mesmo nome. Isto é exemplificado na figura 5.3, que redesenha

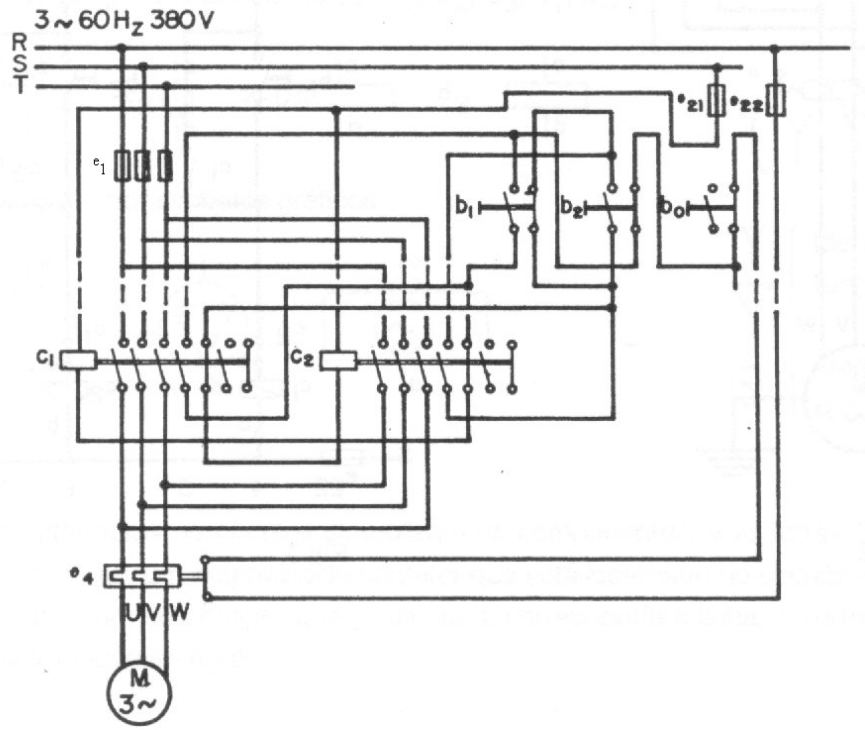



Figura 5.2: Circuito de acionamento de um motor trifásico por relê.

o mesmo circuito da figura 5.2 separando a parte de controle da de acionamento, tornando o circuito bem menos convoluto. A ligação entre os diversos contatos e bobinas dos relês permite a realização de diversas funções interessantes para o controle de circuito. Tais circuitos ficaram conhecidos por *circuitos chaveados*.

### 5.5.1 Diagrama Ladder

Os diagramas ladder são muito utilizados para representar circuitos com relês enfantizando a lógica da ligação. Neste tipo de diagrama as bobinas tem o símbolo $-\bigcirc-$, os contatos normalmente abertos são simbolizados por $\dashv\vdash$ e os normalmente fechados por $\dashv/\vdash$.

A Figura 5.4 mostra o mesmo circuito de controle da Figura 5.3, só que agora descrito em ladder. Com os circuitos conectados desta maneira, o diagrama fica parecendo uma escada; daí o nome[1].

Logo de cara nota-se que não existem neste diagrama as tensões V1 e V2. Isto se dá pois neste diagrama considera-se que as tensões estão entre as duas barras

[1]em inglês *ladder* significa escada.

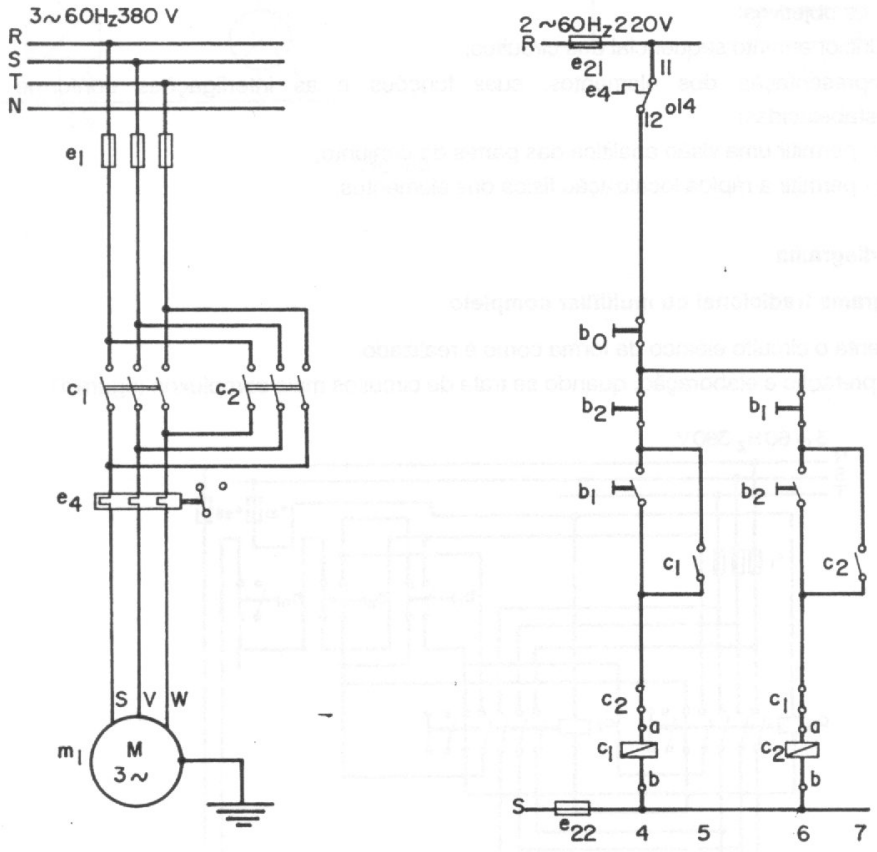



Figura 5.3: Mesmo circuito da figura 5.2, separando a parte de controle da de acionamento.

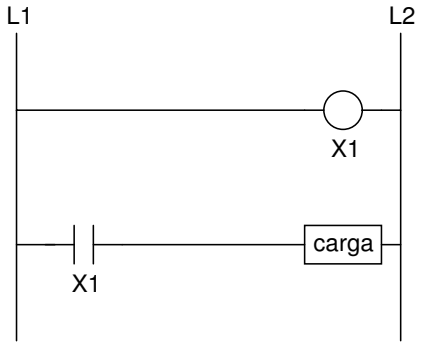
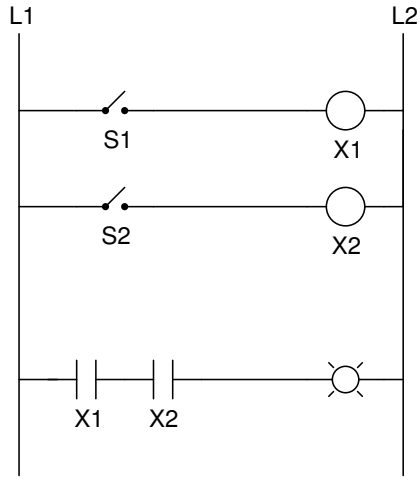



Figura 5.4: Exemplo de circuito de controle em ladder.

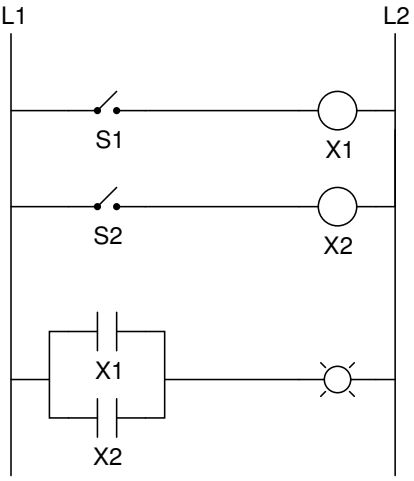
L1 e L2 e abstrai-se a fonte de tensão. Isto lembra, de certo modo, uma ligação real, com os elementos conectados entre os cabos de fase e o neutro.

Com este tipo de diagrama, fica fácil analisar a lógica por trás dos circuitos, o que fez com que este diagrama ainda seja usado, só que agora como uma linguagem gráfica de programação de CLPs.

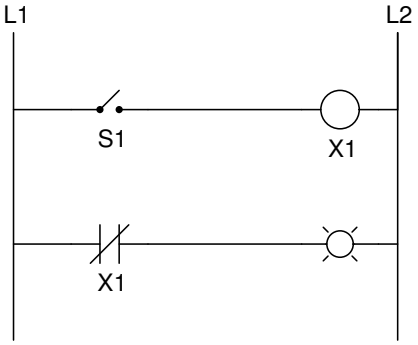
Como exemplo, a figura 5.5 mostra diagramas ladder que acendem uma lâmpada apenas quando 2 relês são acionados, se qualquer um dos relês for acionado ou quando nenhum dos relês é acionado.




(a) Aciona apenas se ambos relês estiverem acionados.




(b) Aciona se um ou outro relê estiver acionado.




(c) Aciona se o relê não estiver acionado.

Figura 5.5: Exemplos de circuitos lógicos chaveados em ladder.

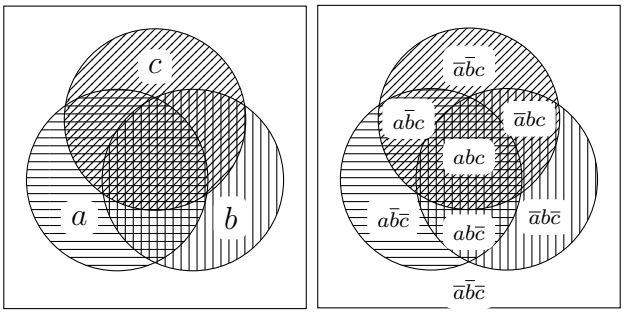
O engenheiro Claude Shannon descobriu em 1934 a relação entre os circuitos chaveados e a álgebra de Boole, que é um mapeamento da lógica em uma álgebra. Uma ligação em série realiza um AND lógico (uma multiplicação booleana), uma ligação paralela realiza um OR lógico (uma soma na álgebra de Boole) e o uso de um contato normalmente fechado realiza a inversão lógica, ou o NOT (representado por uma barra sobre a variável). A álgebra de Boole mostra que a

combinação destas três operações, e portatnto a combinação destes três circuitos, permite realizar qualquer condição lógica para o acionamento do que quer que seja. Porém a aplicação direta dos postulados e teoremas desta álgebra é muitas vezes não intuitiva e portanto complicada e passível a erros.

## 5.6 Mapas de Karnaugh

O chamado mapa de Karnaugh foi desenvolvido pelo matemático e físico Maurice Karnaugh em 1953, enquanto trabalhava no grupo de pesquisas da empresa Bell. Este método é uma poderosa ferramenta para circuitos lógicos.

Como as equações da álgebra de Boole tratam, também, de probabilidade, elas podem ser visualizadas através de um diagrama de Venn. Isto é exemplificado na figura 5.6, que apresenta um diagrama de Venn de 3 variáveis com todas as regiões demarcadas.




(a) Regiões das variáveis (b) Sub-regiões dos ANDs.

Figura 5.6: Diagrama Venn com 3 variáveis.

Utilizando os diagramas, é fácil obter a equação simplificada da função. Por exemplo, considere-se a função $f_1 = a\overline{b}\overline{c} + a\overline{b}c + \overline{a}\overline{b}c$. Desenhando esta função num diagrama de Venn (figura 5.7), fica óbvio que podemos simplificá-la para $f_1 = (a + c)\overline{b}$.

O problema aparece quando acrescentamos mais 1 variável. Como fazer um diagrama definindo todas as 16 possibilidades? Uma solução para isto é desenhar as regiões como retângulos e não como círculos, assim como foi feito na figura 5.8, lado esquerdo. Uma melhora é ainda aplicada no diagrama do lado direito desta figura, onde a indicação de que regiões correspondem a que variáveis é feita não pelo padrão da área, mas sim indicada no lado externo.

O mapa de Karnaugh já é este diagrama de Venn modificado, onde o resultado da função booleana mapeada é marcado em cada região (casa). Cada casa em um mapa de Karnaugh corresponde a uma linha na tabela verdade, que é um AND

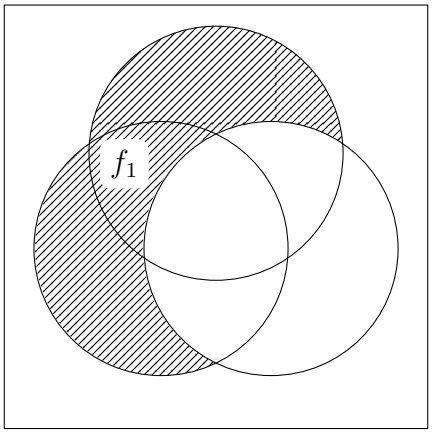



Figura 5.7: Diagrama Venn definindo a região dada por $f_1 = a\bar{b}\bar{c} + a\bar{b}c + \bar{a}\bar{b}c = (a + c)\bar{b}$.

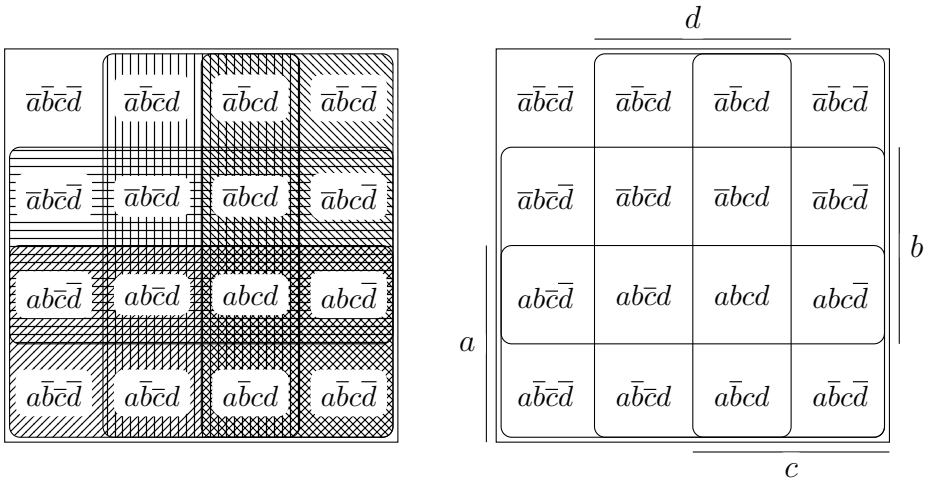



Figura 5.8: Diagrama Venn de 4 variáveis desenhado com regiões quadradas.

de todas as variáveis envolvidas – vamos chamar isto de mintermo. A figura 5.9 mostra os mintermos correspondentes a cada uma das regiões.

Note na figura 5.9 que os vizinhos de cada casa em um mapa de Karnaugh são tais que apenas muda uma variável de cada vez. Por exemplo, da casa 5 para a 1 (acima) só muda o $b$, da 5 para a 7 (direita) só muda o $c$, da 5 para a 4 (esquerda) só muda o $d$ e da 5 para a 13 (abaixo) só muda o $a$.

Isto que foi mostrado para a casa 5 é válido para todas casas, inclusive para as bordas e quinas, pois podemos considerar que o mapa dá a volta em si mesmo. Deste modo considera-se a casa 6 como vizinha da 4 e só muda a variável $c$, a casa 10 vizinha da 2 e só muda a variável $a$ e assim por diante.

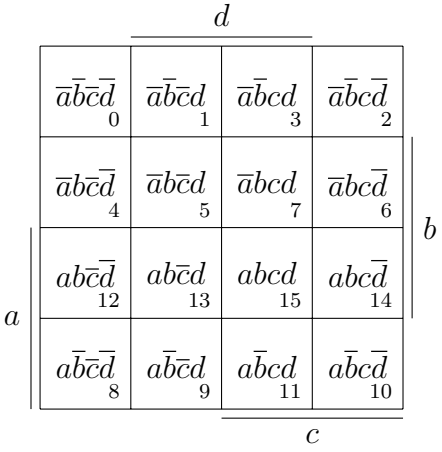



Figura 5.9: Mapa de Karnaugh com os mintermos correspondentes a cada casa.

Desta característica do mapa de Karnaugh vem sua principal utilidade. Por exemplo, considere a função $f_2 = \Sigma_m(3, 7, 12, 13)$ – o somatório dos mintermos das casas 3, 7, 12 e 13 – e seu respectivo mapa de karnaugh na figura 5.10.

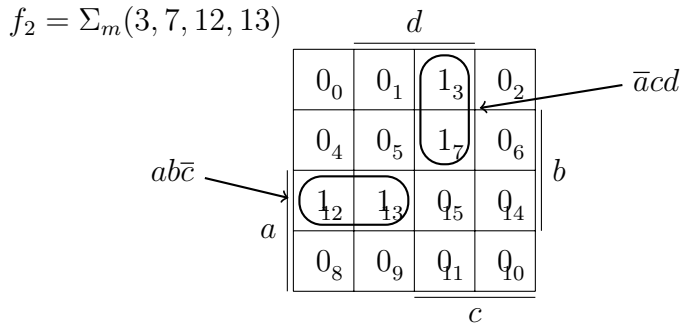



Figura 5.10: Função $f_2$ e simplificação por agrupamento de casas vizinhas.

Analisando a função $f_2$ por álgebra de Boole, vemos que podemos simplificá-la através da aplicação do teorema que diz que $a\overline{b} + ab = a$ e, observando no mapa de Karnaugh, os termos que são unidos e simplificados são justamente os vizinhos.

$$f_2 = \underbrace{\overline{a}\overline{b}cd + \overline{a}bcd}_{\overline{a}cd} + \underbrace{ab\overline{c}\overline{d} + ab\overline{c}d}_{ab\overline{c}}$$

Ou seja, o agrupamento de 2 casas vizinhas corresponde à simplificação de uma variável. Basta ver no próprio mapa quais são as variáveis que não mudam dentro do agrupamento.

Para simplificar 2 ou mais variáveis basta aplicar o teorema repetidas vezes. Simplifiquemos a função $f_3$ (vide figura 5.11), por exemplo. Basta agruparmos a função de duas em duas casas e 2 grupos vizinhos de duas casas viram um único grupo de 4 casas, retirando mais uma variável da função.

$$\begin{aligned} f_3 &= \underbrace{a\bar{b}\bar{c}\bar{d} + a\bar{b}\bar{c}d} + \underbrace{ab\bar{c}\bar{d} + ab\bar{c}d} \\ &= \underbrace{a\bar{b}\bar{c} + ab\bar{c}} \\ f_3 &= a\bar{c} \end{aligned}$$

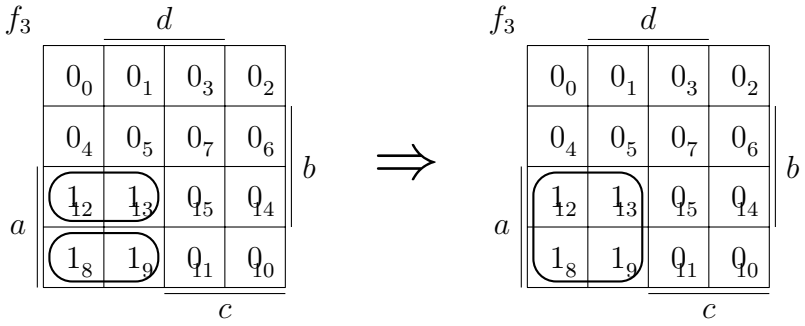
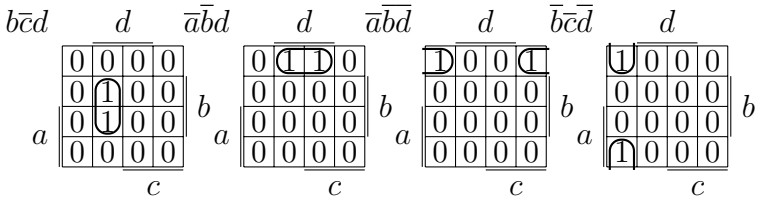


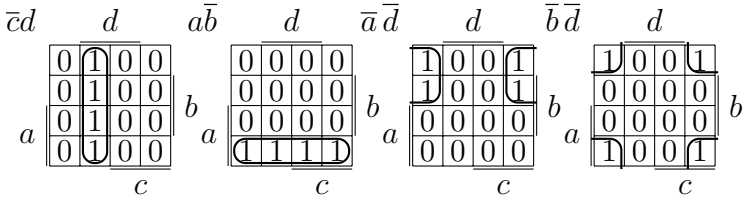

Figura 5.11: Agrupamento das casas da função $f_3$.

Este mesmo procedimento pode ser mostrado para agrupamentos de 8 casas (simplificando então 3 variáveis) ou 16 casas (simplificando 4 variáveis. A figura 5.12[2] mostra algumas possibilidades de agrupamentos de 2, 4 e 8 casas, junto com o produto respectivo. Num mapa de Karnaugh de 4 variáveis um agrupamento de 16 casas seria todo o mapa e corresponderia a função 1.

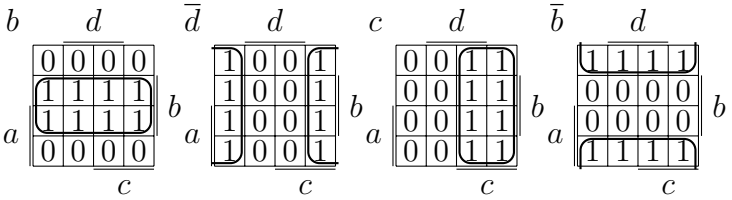
[2]Exemplos retirados do artigo original de Karnaugh: "*The map method for synthesis of combinational logic circuits*", de 1953.




(a) agrupamentos de 2 casas




(b) Agrupamentos de 4 casas

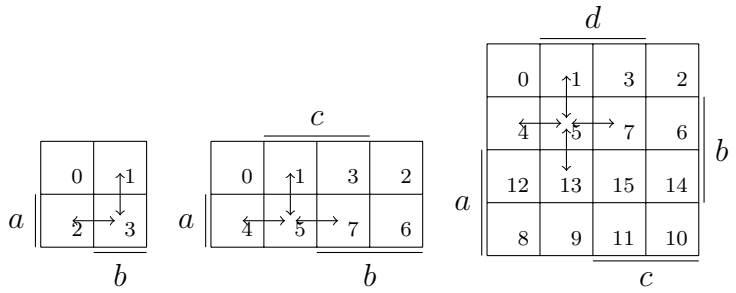



(c) Agrupamentos de 8 casas

Figura 5.12: Exemplos de mapas de karnaugh com os correspondentes produtos algébricos.

### 5.6.1 Mapas de $n$ variáveis

É fácil fazer um mapa de Karnaugh com um número menor de variáveis (i.e.: $n < 4$). Para tanto basta simplesmente sair dividindo o mapa. Deve-se apenas lembrar que uma casa deve ter $n$ vizinhas, já que a simplificação de uma variável corresponde a unir uma casa com a vizinha. Isto é mostrado na figura 5.13 para mapas de 2, 3 e 4 variáveis.




(a) Mapa de 2 variáveis (b) Mapa de 3 variáveis (c) Mapa de 4 variáveis

Figura 5.13: Mapas de Karnaugh de $n = 2, 3, 4$ variáveis, mostrando que cada casa tem $n$ vizinhas.

Mas como aplicar este princípio para funções com mais de 4 variáveis? É impossível fazer um mapa no plano onde cada uma das regiões tem 5 (ou mais) vizinhos. Uma maneira (não muito prática) é trabalhar com um mapa tridimensional como exemplifica a figura 5.14 que mostra um mapa de Karnaugh de 6 variáveis, note que cada casa tem 6 vizinhos: 4 no plano (como no mapa de 4 variáveis) e 2 verticais.

Na prática, um mapa de 5 variáveis é desenhado como 2 de 4 variáveis, sendo um com uma variável (em geral a mais significativa) sendo 0 e o outro com a mesma variável sendo 1. Usa-se este mesmo princípio para mapas de 6 ou mais variáveis, como pode ser visto na figura 5.15.

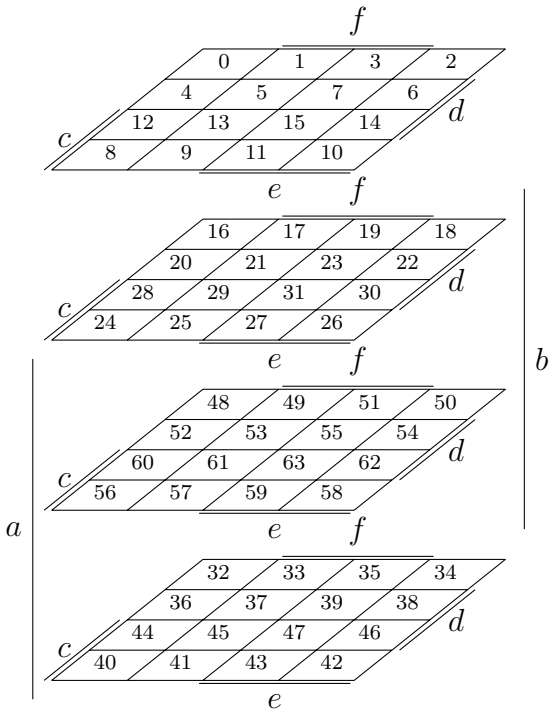
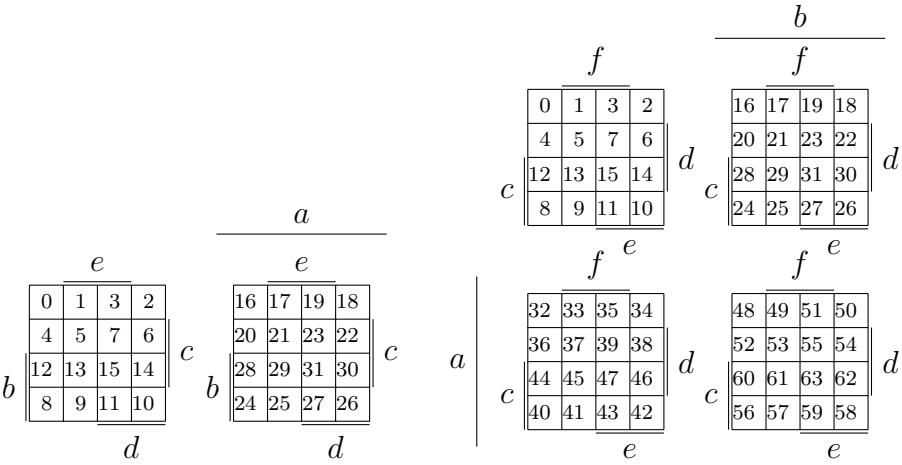



Figura 5.14: Mapa de Karnaugh tridimensional de 6 variáveis.




(a) 5 variáveis

(b) 6 variáveis

Figura 5.15: Mapas de Karnaugh de 5 e 6 variáveis.

## 5.7 Equações e circuitos não-completamente especificados.

É bastante comum a situação de que determinadas entradas de um circuito lógico nunca ocorram. Como exemplo imagine-se uma esteira carregando uma caixa de um lado para o outro, com sensores de fim de curso em ambas extremidades: $s_e$ do lado esquerdo e $s_d$ do lado direito. No funcionamento normal do sistema estes dois sinais nunca serão acionados ao mesmo tempo; nesta situação não importa qual é o resultado do circuito para $s_e = s_d = 1$, já que esta situação nunca vai existir. Diz-se então que este circuito é não-completamente especificado.

A tabela 5.1 mostra o exemplo de um sinal imaginário $z$ determinado em função de $a$, $s_e$ e $s_d$. Neste exemplo, sempre que $s_e = s_d = 1$ a saída $z$ é não-especificada, ou seja, $z$ *não-importa* nestas situações. Neste texto utilizaremos a notação '$\times$' para identificar as situações que um sinal não importa. Outras notações comumente usadas são '$*$', '-' ou '$d$'.

Tabela 5.1: Exemplo de uma função não completamente especificada.

| $a$ | $s_e$ | $s_d$ | $z$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | $\times$ |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | $\times$ |

Pode-se descrever uma função lógica não-completamente especificada na forma soma de mintermos utilizando a notação $d(\cdots)$, que vem do inglês *don't care*. Desta forma o sinal z pode ser descrito por:

$$z = \Sigma_m(2, 4, 6) + d(3, 7) \tag{5.1}$$

Um $\times$ na saída pode ser implementado como um 1 ou um 0, e não se sabe a princípio qual destes dois valores gerará uma solução mais minimizada, logo para obter o menor circuito possível o engenheiro deveria, a princípio, obter as equações considerando que cada $\times$ pode ser 1 ou 0 e checar qual é o menor circuito final. Obviamente para problemas com muitos $\times$'s isto se torna impraticável, pois seria necessário minimizar $2^k$ funções, onde $k$ é o número de $\times$'s presentes.

O mapa de Karnaugh facilita bastante a implementação de circuitos não-completamente especificados, pois podemos considerar se determinado $\times$ é 1 ou 0 visualmente, na hora da implementação.

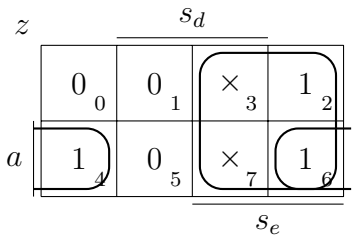



Figura 5.16: Mapa de Karnaugh da função $z$.

## 5.8 Linguagem Ladder – Temporizadores

## 5.9 Linguagem Ladder – Contadores

## 5.10 Linguagem Ladder – Aplicações

# Capítulo 6

# Linguagem Grafcet

O Grafcet pode ser definido em 2 níveis de abstração: o 1 e o 2. O nível 1 serve como uma ferramenta de desenvolvimento, para analisar a partir do problema como um todo a sequência de etapas, as ações a serem realizadas em cada etapa e as condições de transição de uma etapa para outra. No nível 1 as ações e transições são descritas de forma ampla, para permitirem uma melhor visualização do processo pelo desenvolvedor. O resultado final é uma descrição dos requisitos daquelas etapas e transições e não serve como uma linguagem para programar um sistema.

Já o Grafcet nível 2 é propriamente uma linguagem de programação, com diferentes implementações. Uma delas é o SFC - *Sequential Function Chart*, descrita no padrão IEC1131-3 para programação de CLPs. No SFC, cada ação corresponde a uma atuação nas saídas ou variáveis internas do CLP e cada transição corresponde a um valor binário obtido no CLP seja de entradas, seja de comparações de valores analógicos ou de tempo.

Desta forma o principal uso do grafcet é num projeto top-down, onde a partir do problema se desenvolve a sequência a ser seguida no grafcet nível 1, a partir deste se definem todas as entradas, saídas e condições necessárias para o controle automático daquela sequência e então programa-se o sistema usando grafcet nível 2.

Tomemos como exemplo uma tarefa relativamente simples: cozinhar um ovo.

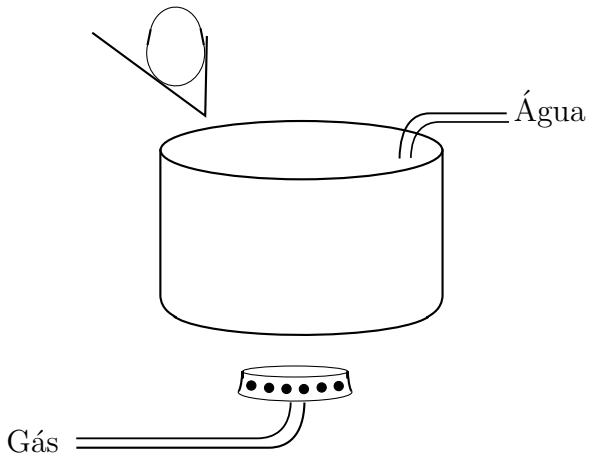



Figura 6.1: Sistema para cozinhar ovo sem sensores ou atuadores.

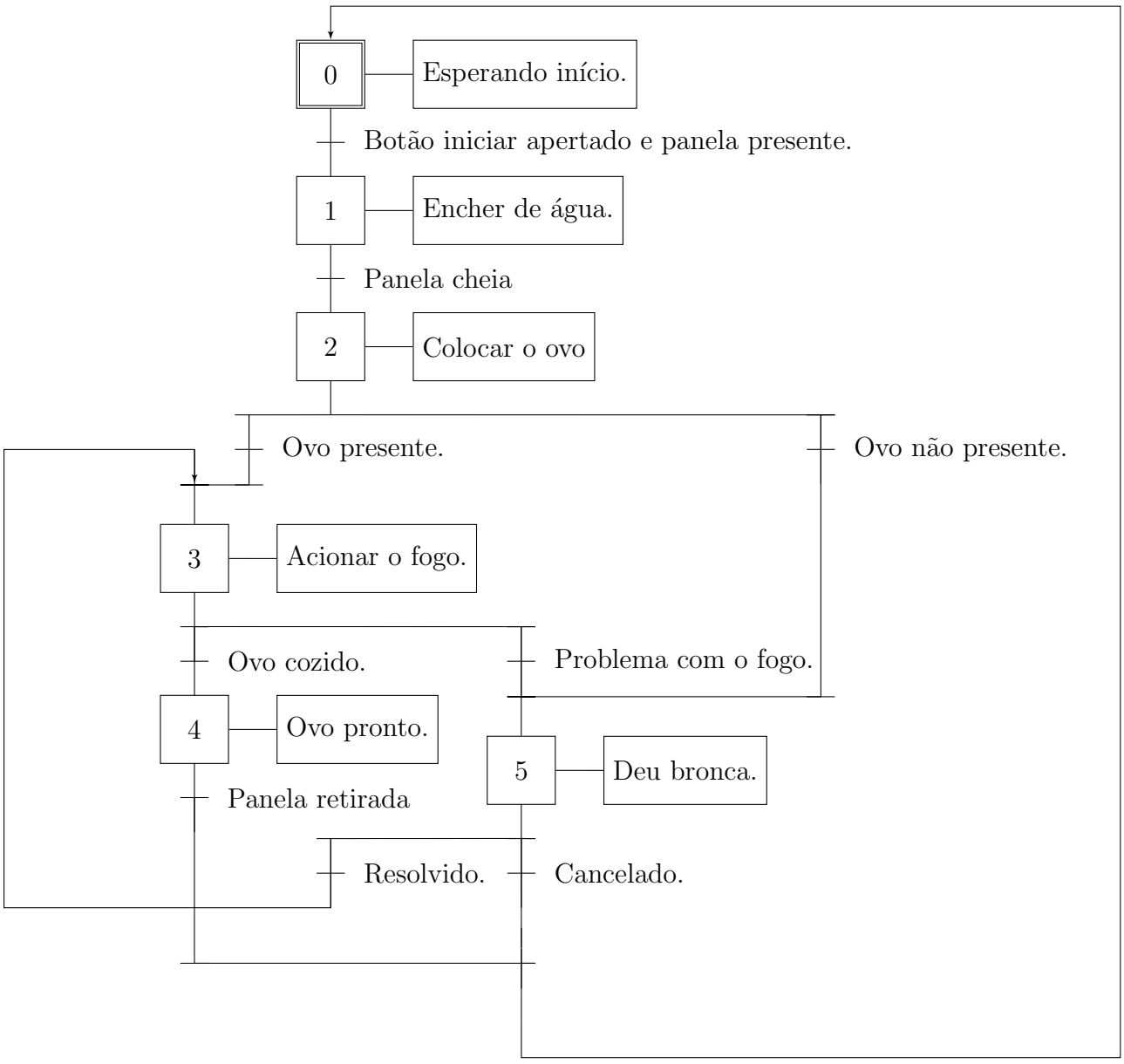



Figura 6.2: Diagrama grafcet nível 1 para cozinhar um ovo.

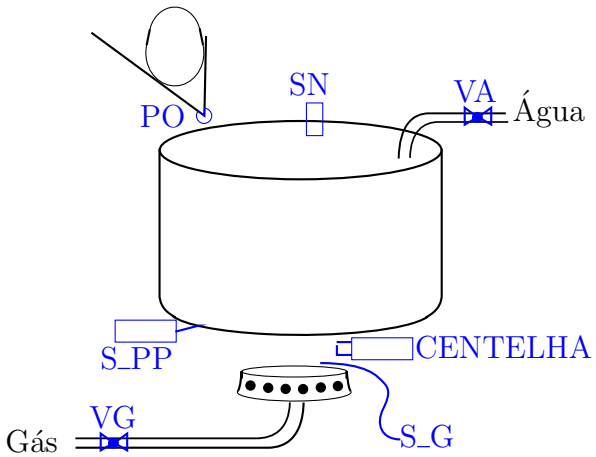
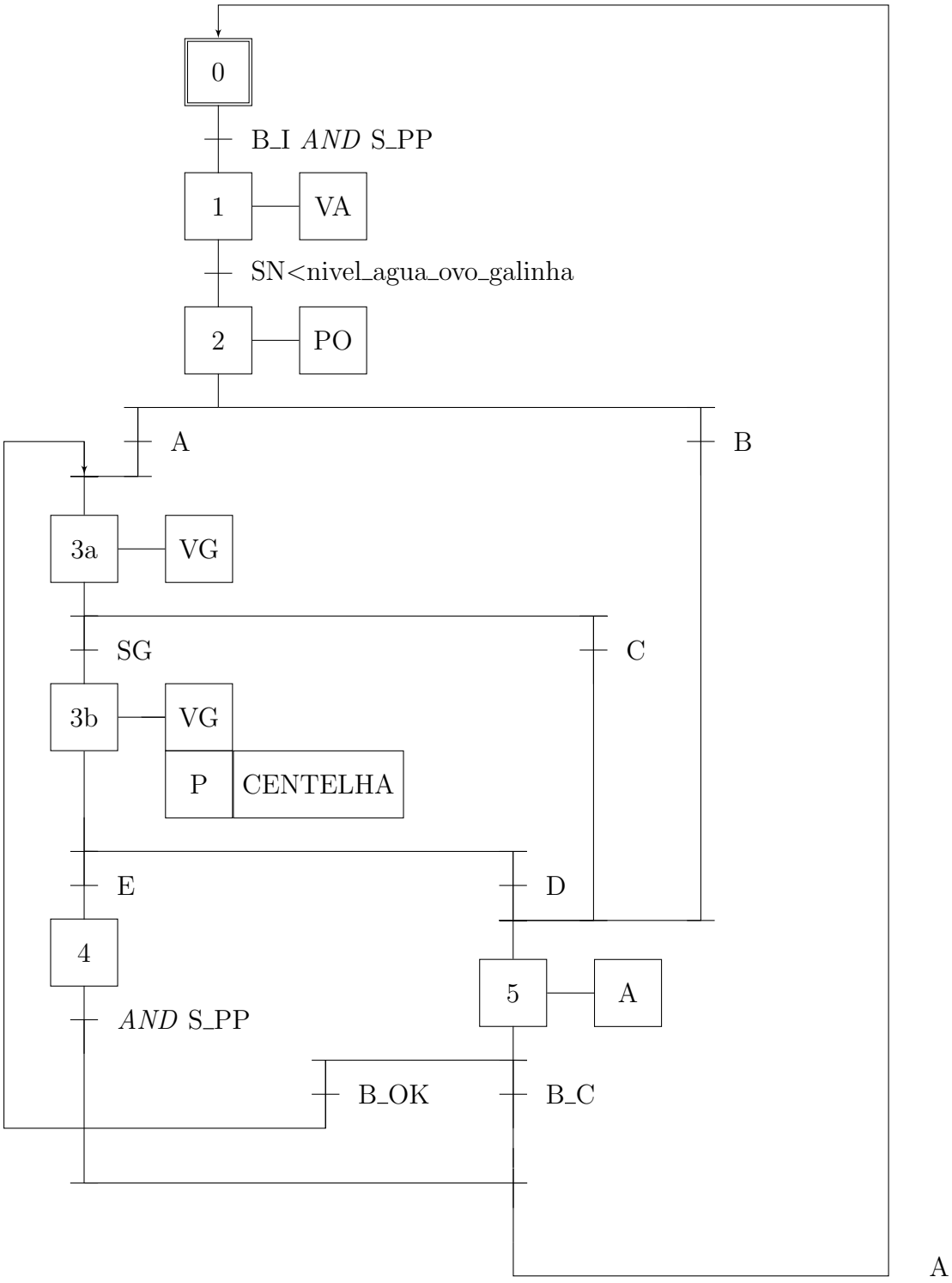



Figura 6.3: Sistema para cozinhar ovo com sensores e atuadores.




= S_N < (nivel_ovo_galinha - volume_ovo_galinha)
B = *NOT* A *AND* 2.T>T#10s
C = *NOT* SG *AND* 3a.T>T#4s
D = SG *AND* 3b.T>T#4s
E = *NOT* SG *AND* 3b.T>tempo_ovo_galinha



Figura 6.4: Diagrama grafcet nível 2 para cozinhar um ovo.

## 6.1 Linguagem Grafcet – Aplicações Parte 1

## 6.2 Linguagem Grafcet – Aplicações Parte 2

# Capítulo 7

# Instrumentação Industrial

## 7.1 Medição de grandezas mecânicas. Características de instrumentos.

## 7.2 Transmissão de dados, aterramento e blindagem em instrumentação.