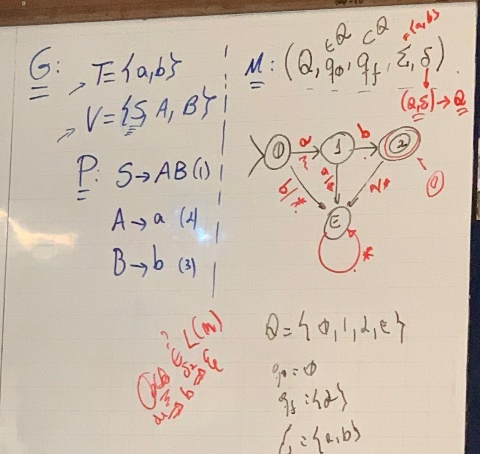
Aula dia 06/02/2022

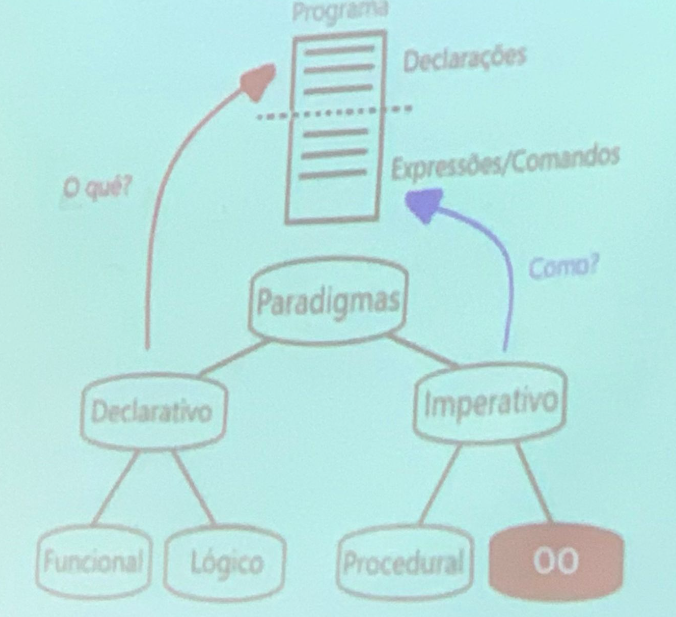
Linguagens Formais – Gramáticas e “Máquinas”

* Bebemos da mesma área que os cursos de linguagens, pois do mesmo jeito é estudado símbolos, palavras, frases ... – (linguagens formais)
* Linguagens formais 🡪 Gramática (gera palavra) ou Máquina 🡪 quidrupla (reconhece a palavra)
* Linguagens artificias 🡪 tem símbolos, porém não é de programação (SQL, HTML)
* Teoria de Linguagens:
  + Cada linha do código JAVA é um “programa”
  + Linguagens formais 🡪 conjunto de palavras sobre um alfabeto (sequência finita de símbolos do alfabeto justapostos)
  + Uso do Sigma \* conta com TUDO e a palavra vazia, já o Sigma + não conta com o vazio
  + Gramática é uma quádrupla ordenada G = {V,T,P,S}
    - V 🡪 símbolos variáveis ou não-terminais
    - T🡪símbolos terminais disjuntos de V
    - P🡪regras de produção  
      S🡪variável inicial
  + ORIGEM 🡪 DESTINO (esquerda gera a direita) – origem não pode ser vazio , já o destino pode.
  + Seta simples (produção)
  + Seta dupla (derivação) 🡪 substituição de uma subpalavra do esquerdo pelo direito
  + Linguagem gerada 🡪 A linguagem gerada pela gramática G é composta por todas as palavras de símbolos terminais deriváveis a partir do S
  + Linguagem equivalentes 🡪 Duas gramáticas geram a mesma palavra
* Sistema de estados finitos (olhar anotações de arquitetura para mais detalhes):
  + É um modelo matemático de sistema com entradas e saídas discretas
  + Pode assumir um n finito e pré-definido de estados
  + Exemplo: Elevador ( ngn importa a história )
* Exemplo de Gramática e Máquina:

08/02/2023 – Linguagem de Programação e Histórico

* OBS: Para o seminário lembrar da formação da palavra

Introdução as LPS:

* Linguagem de Programação:
  + Linguagem formal 🡪 conjunto de palavras sobre um alfabeto
  + Qualquer notação para descrição de algoritmos e estruturas de dados
  + Objetivo 🡪 expressar QUALQUER programa de computador, isso significa que elas são **universais** e tem que estar **implementada (Requisitos obrigatórios)**
  + Requisitos desejáveis 🡪 Ser natural e Implementação eficiente
  + HTML 🡪 simbiose com código, porém não é considerada uma LP
  + Sintaxe e Semântica:
    - Sintaxe 🡪 forma
    - Semântica 🡪 significado
    - EX: 3 + ‘A’, está correta pela sintaxe, entretanto não faz sentido assim estando incorreto quando analisamos seu significado
  + Linguagens funcionais não tem comandos
  + Paradigmas:
    - Conjunto de entidades reunidas por afinidade sob determinadas regras de uso dos recursos disponíveis
    - Conjunto modelo dos conceitos individuais das LPS agrupados segundo afinidade dos projetos e estilos de programação das LPS: declarativo(funcional e logico) e imperativo (procedural e OO)
    - Define o possível e o impossível
    - Se mudamos os paradigmas temos que adaptar e pensar na solução de acordo com ele, exemplo de uma sala sem cadeiras e mesas, teremos que adaptar a aula
  + Histórico:
    - Tentativa de desvincular de máquina e sim a aplicação, uso do compilador
    - Em 1950 - 1960 a computação era mais para operações matemáticas, além de ser imperativa com sinais
    - Em 1960 perceberam que existem mais aplicações do que apenas para matemática, ex COBOL e LISP
    - Em 1967 começa a ter a noção de OO
    - Em 1974 C é feito

Aula dia 13/02/2023

Valores (1/3):

* Definição de valor: **Valor** é algo que que possa operar 🡪 operações próprias dependendo do tipo de valor
  + Computação tenta espelhar e trabalha com isso desde a linguagem de máquina
  + Qualquer entidade que exista durante uma computação, ou seja/ qualquer cosa que pode ser avaliada, operada, armazenada pode ser considerado um valor
  + Para estudar valores devemos agrupá-los em tipos
  + **Tipos**: especificação da classe de valores que podem ser associados á variável, bem como das operações que podem ser usados para criar, acessar e modificar esses valores
  + LPS em geral proveem um conjunto de **tipos primitivos** e mecanismos para estruturas **tipos compostos.** Algumas oferecem a opção de tipos recursivos.
  + Os tipos de dados que compõem um programa devem ser o mais próximo do enunciado do problema.
  + Composição de valores: Junção de valores (struct em c , e classe em JAVA)
  + **Tipos primitivos:**
    - Valores atômicos
    - Indicam a área de aplicação da LP
    - LPS costumam dar nomes diferentes a seus tipos primitivos
    - Mais comuns 🡪 logico, inteiro, real e caractere
  + A representação subjacente fica invisível ao programador 🡪 maior legibilidade e portabilidade dos programas, excluso apenas em casos extremos

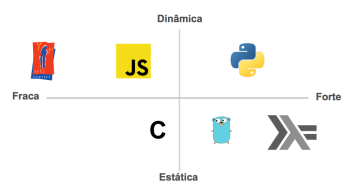
Aula dia 15/02

Valores (2/3)

* + **Tipos compostos:**
    - Compostos a partir de tipos simples, mas pode-se ser composta por tipos compostos também
    - Grande variedade: tuplas, registros, arquivos, arranjos, strings , listas , árvores.
    - Mecanismos básicos de estruturação:
    - A) Produto cartesiano:
      * Conjunto de n tuplas ordenadas
      * Tuplas, Registros, Estruturas e classe
    - B) União disjunta:
      * Mede se os valores são exclusivos – XOR
      * Uma escolha sera feita dentre estruturas alternativas
      * Existe uma característica discriminante
      * EX: Estado Civil , if solteiro nada , if casado pede data de casamento , if divorciado pede data de casamento e divorcio
      * Tags(discriminante) são valores e podem gerear inseguranças , acessar um campo mesmo que ele não exista 🡪 erro de execução
      * Em C é um struct com union , em pascal é um registro com case
    - C) Mapeamento:
      * É uma função de um conjunto finito de valores S em valores de um tipo T
      * Arranjos
      * O conjunto do índice deve ser discreto
      * Abstrações de funções representam outro tipo de mapeamento
    - D) Sequência:
      * Consiste em um número de atributos de ocorrências de itens de dados de um determinado tipo T
      * Deixa em aberto o número de ocorrências de um componente
      * Strings e arquivos sequenciais
    - F) Recursão
      * Um tipo de dados recursivo T pode ter componentes que pertençam ao próprio tipo T
      * Permite definir agregados cujo tamanho pode crescer arbitrariamente e cuja estrutura pode ter complexidade arbitrativa
      * A cardinalidade de um tipo recursivo é infinita
    - E) Conjunto potência
      * É um tipo de variável cujo valor pode ser qualquer subconjunto de um conjunto de Telementos, chamado de base
      * Conjunto de conjunto

Aula dia 27/02/2023

Valores (3/3) – Princípio da completeza de tipos e expressões

* Sistema de tipos:
  + Tipagem estática e dinâmica:
    - Para evitar operações sem sentido, uma implementação de lp deve realizar uma verificação de tipos sobre os operandos
    - **Estática:** Toda variável e parâmetro possui um tipo fixo determinado pelo programador 🡪 verificação em tempo de compilação
    - **Obs:** Linguagem fracamente tipada confia na experiencia do programador e não enche o saco
    - **Dinâmica:** Somente os valores possuem um tipo fixo , ou seja, variáveis e parâmetros podem assumir valores de tipos diferentes durante a execução 🡪 verificação em tempo de execução
    - **Diferenças:** 
      * A tipagem dinâmica torna mais lenta a execução de um programa, além disso é mais flexível
      * A tipagem estática é mais segura
  + Equivalência de tipos:
    - **Equivalência estrutural**: Duas variáveis têm tipos compatíveis se possuem a mesma estrutura
    - **Equivalência de nomes:** Duas variáveis têm tipos compatíveis se possuem o mesmo nome de tipo, definido pelo usuário ou primitivo, ou se aparecem na mesma declaração
    - **Diferenças:**
      * Equivalência de nomes se aproxima mais ao conceito de tipos abstratos de dados, é mais fácil de implementar e na teoria impede dois programas de se comunicas através de arquivos
* Princípio da Completeza de tipo:
  + Principio 🡪 Diferente de Dogma , ou seja , não é obrigatório e sim uma referencia
  + “Nenhuma operação deve ser arbitrariamente restringida sobre os tipos de valores envolvidos”
  + Justificativa: restrições tendem a reduzir o poder de uma expressão de uma LP
  + Polimorfismo 🡪 Não precisa ficar reescrevendo código.
* Expressões:
  + Representamos valores por meio de expressões
  + Fragmentos de um programa que tem significado computacional e fornece um valor
  + São frases de programa que posem ser avaliados a fim de fornecer um valor
    - A) Literal:
      * Forma mais simples de uma expressão
      * Representa um valor fixo
      * Ex : 123 , 1.2 , ‘b’
    - B) Agregado:
      * Expressão que constrói um valor composto a partir de valores de seus componentes
      * Normalmente tem ()
      * Usa expressões
      * EX (a\*2.0,2/2)
    - C) Chamada de função:
      * Calcula um resultado através da aplicação
      * Operador está dentro
      * EX: f(x)
    - D) Expressão condicional:
      * Possui varias subexpressoes, dentre quais somente uma é escolhida para ser avaliada
      * Nem toda LP é assim
      * EX: (a>b)?a:c
    - E) Acesso a variável e constantes:
      * Produz os valores das variáveis denotadas pelos respectivos identificadores

01/03/2023 e 06/03

Armazenamento - Variáveis e tempo de vida

* Variáveis:
  + Objeto que contêm valor
  + Modelam objetos do mundo real que possuem um estado
  + Tem tempo de vida curto, ao contrário de arquivos
  + Diferente das variáveis matemáticas, pois elas são atualizáveis ex x= x+1
  + Modelo abstrato de armazenamento:
    - Uma memória é uma coleção de células
    - Cada célula tem status corrente: alocada ou livre
    - Cada célula tem um conteúdo corrente: valor armazenável ou indefinido
  + **Compostas**:
    - Uma variável de tipo composto consiste em variáveis, o conteúdo desses pode ser inspecionado e atualizado seletivamente
    - Atualização seletiva = atualização de um compenetre de variável
    - Atualização total = atualização total da variável
  + **Arranjo**:
    - Estático = tempo de compilação
    - Dinâmico = no momento da criação de variável
    - Flexível = quando são feitas atribuições a variável
  + **Armazenáveis**:
    - Valores que podem ser armazenados em células simples e que não podem ser atualizados seletivamente 🡪 valores primitivos, conjuntos e apontadores
  + **Tempo** **de vida**:
    - É o intervalo entre a criação e a destruição de uma variável
    - Relaciona-se á economia de memória 🡪 uma variável só precisa ocupar memória se estiver vivo
  + **Locais e globais:**
    - Local 🡪 declarado dentro de um bloco para uso somente dentro do bloco, pode ter a local estática que se comporta como global
    - Global 🡪 Pode ser referencia dentro de um bloco , mas não foi declarada localmente
  + **Heap:**
    - Podem ser criadas e destruídas a qualquer momento
    - São criadas por comandos
    - São anônimas
    - Acessadas por meio de apontadores
    - Alocador 🡪 cria variável de heap
    - Liberador 🡪 destrói variável de heap
  + **Persistentes:**
    - Transiente 🡪 tempo de vida esta limitado a ativação do programa que a criou
    - Persistente🡪 tempo de vida transcende uma ativação de um programa particular
  + **Referencias penduradas (dangling):**
    - É um ponteiro que aponta para uma área de memória que foi deslocada
    - **EX:**
      * Quando uma referência a uma variável local é atribuída com tempo de vida maior
      * Quando uma LP possui um liberador, assim transformando a variável em referencias penduradas
  + **Comandos:**
    - São frases de programas que serão executados a fim de **atualizar variáveis**
    - Caracterizam as LPs imperativas
    - Tipos fundamentais de comandos 🡪 tem que estar completo:
      * Saltos 🡪 IF,THEN,ELSE e SKIP
      * Atribuições 🡪 V:=E , onde V é acesso a variável e passará o valor de E – dereferenciação é o processo inverso
      * Chamadas de procedimento 🡪 Aplica uma abstração de procedimentos a alguns argumementos
      * Comandos colaterais 🡪 Comandos executado sem ordem específica
      * Comandos condicionais 🡪 Possui subcomandos , dentre os quais somente um é escolhido para ser executado
      * Comandos iterativos🡪 Um comando iterativo (loop) tem um subcomando que será executado repetidamente

**Aula dia 08/03 – Amarrações e Vinculações - “Binding”**

* Ambiente:
  + Conjunto de amarrações que foram estabelecidas – todo programa tem
  + Pode-se dizer que comandos são executados a fim de atualizar o ambiente, já que são formadas por variáveis também.
  + Especificação de alguns dos atributos de uma entidade
* Amarrações:
  + Uma declaração produz uma associação ou amarração entre o identificador e a entidade que irá denotar
  + **Denotáveis:**
    - Entidades que podem ser denotadas por identificadores, ou seja , aquilo que pode ganhar nome
    - EX: Valores primitivos e strings , Referencias a variveias
    - Int x = 5; 🡪 Amarração
* Escopo x Bloco:
  + Escopo 🡪 Trecho do programa onde uma declaração é efetiva , ou seja, um identificador é conhecido e pode ser usado
  + Bloco 🡪 Qualquer sentença de programa que delimita o escopo, ex BEGIN e END ou {}
  + Tipos de estrutura de bloco:
    - A) Monolítica 🡪 O programa todo é um bloco
    - B) Nivelada 🡪 O programa é particionado em blocos distintos
    - C) Aninhada 🡪 Cada bloco pode estar aninhado dentro de outro bloco
* Escopo e visibilidade:
  + O que é amarrado uma vez é possível usar várias vezes (batismo)
  + Ocorrência de amarração: ocorrência de um identificador no ponto em que é declarado
  + Ocorrência de aplicação: ocorrência de identificador que denota uma entidade
  + Amarração estática: O corpo da função é avaliado no ambiente de definição da função, pode-se determinar em tempo de compilação a qual ocorrência de corresponde uma ada ocorrência de aplicação de um identificador
  + Amarração dinâmica:

13/03/2023 – Amarrações

* Declarações:
  + É uma frase de programa utilizada para elaborar amarrações
  + Podem ser implícita e explicita
  + Objetivos:
    - Escolha da representação de armazenamento – no caso da declaração fornecer tipo de dado
    - Gerenciamento da memória
    - Definição de operações que podem ser realizadas
    - Permitir verificação estática de tipos
* Definição:
  + Uma declaração simples cujo único efeito é produzir amarrações
  + EX: Em pascal, uma definição de constante amarra um identificador a um valor determinado em tempo de compilação
* Tipos de declarações:
  + De tipo: Serve somente para amarrar um identificador a um tipo existente
  + De novo tipo: Cria um tipo distinto, apropriado para equivalência de tipos de nomes.
  + De variável: Serve somente para amarrar um identificador a uma variável. (int x = 2;)
  + Colaterais: Combinação de amarrações já produzidas, nenhuma das subdeclarações podem ser usar um identificador declarado em outra subdeclaração.
  + Sequencial : Podem ser escritas D1;D2 , permite que as amarrações produzidas por D1 seja usadas em D2
  + Recursivas =: Utiliza amarrações que ela mesma produz
* Blocos
  + Comandos de blocos:
    - Um comando contendo uma declaração que produz amarrações que serão usados somente para executar o programa
    - EX : let pi = 3.14 in write(pi)
    - Nesse caso , pi não existe antes e nem depois

17/04/2023 – Abstração (1/2)

* Precisamos abstrair duas coisas: **Expressões** e **Comandos** e **Entidades**
* Programação de grande porte se beneficia muito da abstração
* É um processo de identificar as qualidades ou propriedades importantes do fenômeno que está sendo modelado
* Em programação, a abstração faz referência à distinção entre O QUE uma parte do programa faz e COMO ela é implementada
* **Definição É uma entidade que incorpora(personifica) uma computação**
* Tipos:
  + De função:
    - Incorpora uma **expressão** para produzir um resultado
    - Visão do usuário da função: uma chamada da função irá mapear os resultados a um resultado.
    - Visão do programador: uma chamada da função irá avaliar o corpo da função, tendo os parâmetros formais amarrados aos argumentos correspondentes
    - O identificador é amarrado a uma determinada abstração de função, que é uma propriedade de retornar um resultado sempre que for chamada com argumentos apropriados.
  + De procedimentos:
    - Incorpora um **comando** a ser executado irá atualizar o valor da variável.
    - Visão do usuário: Uma chamada de procedimento irá atualizar uma variável, de uma maneira que será influenciada pelos argumentos
    - Visão do programador: Uma chamada de procedimento irá executar o corpo do procedimento, tendo os parâmetros formais amarrados aos argumentos correspondentes 🡪 se passarmos dois parâmetros, podemos ter um argumento como resultado do comando.
  + De seletor:
    - Abstração sobre um acesso a variável
    - Possui um corpo que é um acesso a variável, uma chamada de seletor é um acesso que retorna uma referência a variável
* **Princípio da abstração:**
  + É possível construir abstrações sobre qualquer classe sintática ou frase de programa, as sentenças dessa classe especifiquem algum tipo de computação
* **Parâmetros:**
  + **Formal:**
    - Um identificador utilizado dentro de uma função para denotar um argumento
  + **Real:**
    - Uma expressão que produz um argumento
  + **Argumento:**
    - Um valor que pode ser passado para uma abstração. Cada LP possui o seu próprio conjunto de valores que podem ser passados como argumentos;
  + **Mecanismo de passagem de parâmetros:** 
    - Método de associação entre paramentos reais e formais. Há vários tipos:
      * Cópia: permite que valores sejam copiados para dentro ou para fora de uma abstração quando chamada
      * Apelido:

**19/04/2023**

* **Mecanismos de definição:** 
  + Permite que um parâmetro formal X é amarrado diretamente ao argumento
  + Parâmetro constante 🡪 o argumento é um valor X é amarrado ao valor do argumento durante a ativação da abstração chamada
  + Parâmetro variável 🡪 O argumento é uma referência a uma variável
  + Parâmetro procedimental 🡪 argumento é uma abstração de procedimento
  + Parâmetro funcional 🡪 argumento é uma abstração de uma função
  + A passagem de subprogramas como parâmetros tem a vantagem de permitir que um mesmo subprograma possa executar funções diferentes 🡪 EX: aula com música, aula com filme ...
* Mecanismos de definição (1) x cópia (2):
  + 1 possui uma semântica mais simples
  + 1 é mais eficiente
  + Uma desvantagem dos parâmetros variáveis é a possibilidade de utilização de apelidos 🡪 mais difícil de entender
* O princípio da Correspondência:
  + Correspondência percebida entre alguns mecanismos de passagem de parâmetros e declarações, ou seja, tudo que podemos declarar podemos passar por parâmetro
  + Um projetista de LP pode optar por eliminar todas as diferenças entre declarações e mecanismos de passagem por parâmetro
  + **Definição**: Para cada forma de declaração, existe um mecanismo de passagem de parâmetro e vice-versa
* Ordem de avaliação:
  + - Quando uma abstração é chamada, em que momento cada parâmetro vai ser avaliado?
    - Avaliação prévia 🡪 No ponto da chamada. O parâmetro real é avaliado uma única vez e o seu valor substitui cada ocorrência do parâmetro formal
    - Avaliação de ordem normal 🡪 No momento em que o argumento realmente é utilizado. O parâmetro formal é substituído pelo real
    - Avaliação preguiçosa 🡪 O argumento será avaliado na primeira vez que for usado e unicamente nessa hora
* Função strict x nonstrict:
  + Strict(rigorosa): So pode ser avaliada se TODOS os seus argumentos puderem ser avaliados
  + NonStrict(não rigorosa): se função puder ser avaliada mesmo se algum argumento não for.
* Propriedade de Church-Rosser:
  + Se expressão puder ser avaliada, então ela pode ser avaliada através da utilização consistente da avaliação de ordem normal. Se uma avaliação pode ser realizada em ordens diferentes, então todas as ordens de avaliação produzirão o mesmo resultado
  + Qualquer LP que permita efeitos colaterais não possui a propriedade de Church-Rosser
  + Algol-60 permite que o programador faça uma escolha entre a avaliação.

24/04/2023 – Encapsulamento

* Sinônimo de modularização?
  + Sim
  + Chave: abstração
* Encapsulamento:
  + Sistemas de grande porte são constituídos em **módulos -- >** qualquer unidade de programa que possua nome e que possa ser implementada como uma entidade independente
  + Módulos bem projetados tem um único objetivo e possui uma interface pequena com outros módulos 🡪 é reutilizável
  + Componentes exportáveis = são componentes visíveis externamente ao modulo.
  + Os componentes escondidos = são usados somente para auxiliar a implementação de componentes exportáveis
* Pacotes:
  + Pacotes simples:
    - Um pacote é especificado através de uma lista de informações declarativas. Em geral, essas informações podem ser denotáveis da LP, tais como tipos, constantes, variáveis, procedimentos, funções e outros pacotes
    - Um pacote pode ser visto como um conjunto encapsulado de amarrações
  + Tipos abstratos:
    - Um tipo abstrato é um tipo definido através de um grupo de operações
    - Dificuldades e limitações:
      * Pode possuir valores que não correspondem a qualquer valor do tipo desejado
      * Comparações com resultados incorretos
      * Os valores de um tipo podem ser confundidas com valores de outros tipos , salvo se for declarado como novo tipo
    - A representação é escondida o modulo exporta somente o próprio tipo abstrato com suas operações
    - O que é importante é que somente propriedades desejadas dos valores são observáveis, usando as operações associadas ao tipo abstrato
    - Uma representação sempre pode ser alterada, sem haver a necessidade de realizar alguma alteração externamente ao módulo
    - Definir um tipo abstrato exige mais esforço

26/04/2023

* Objetos e Classes:
  + Objeto simples 🡪 o termo objeto é utilizado para uma variável escondida em um modulo, sendo que esse modulo contém operações exportáveis sob esta variável. A vantagem disso é que alterações não provocam alterações externas ao modulo. Tem tempo de vida que é definido da mesma forma que variáveis locais.
  + Diferença de classe x tipos abstratos:
    - TA 🡪 operações possuem uma operação a mais – Na chamada de operação a um TA todos os argumentos estão explícitos
    - Classe 🡪 Instancias diferentes definem procedimentos distintos sendo que cada um deles acessa um objeto de dados distinto. Procedimentos e funções têm parâmetros implícitos

08/05/2023 – Sistemas de tipos

* Um sistema de tipo é uma coleção de regras para atribuição de tipos a várias partes de um programa
* Um verificador de tipos implementa um sistema de tipos 🡪 é baseado nas informações sobre as construções sintáticas da linguagem
* Monomorfismo:
  + Se toda constante variável, constante, resultado de função é parâmetro formal onde deve ser declarado como um tipo específico
  + Vantagem: verificação de tipos é simples e direta
  + Desvantagem: dificulta a criação de programas reutilizáveis
  + Nenhuma linguagem é totalmente monomórfica
* Sobrecarga:
  + Capacidade de amarrar a um único identificador ou operador várias entidades(amarrações) simultaneamente
  + Tipos:
    - Independente de contexto: requer que S1 e S2 sejam distintos. Para identificar uma função a ser chamada, basta saber o tipo de parâmetro real
    - Dependente de contexto: requer que S1 e S2 sejam distintos **ou** que T1 e T2 sejam distintos.
    - OBS: exemplo da laranja 🡪 3 / 2 – pode ser 1 ou pode ser 1.5
* Polimorfismo:
  + Relaciona-se a abstrações que operam de maneira uniforme sobre valores de diferentes tipos
  + ML foi a primeira lp a fornecer um sistema de tipos realmente polimórfico
* Tipos parametrizados:
  + É um tipo que tem outros tipos como parâmetros
* Politipos:
  + Contém uma ou mais variáveis do tipo.
  + Deriva uma família completa de tipos

10/05/2023 – Sistemas de tipos

* Inferência de tipos:
  + O tipo de uma entidade declarado é inferido, ao invés de ser explicitamente determinado



* + De tipo polimórfico 🡪 – Nem sempre é possível que uma inferência de tipo produza um monotipo. Em outras palavras, às vezes será produzido um politipo
  + Desvantagem: um pequeno erro de programação pode confundir o compilador, que irá produzir mensagens de erro obscuras, ou inferir um tipo diferente daquele desejado pelo programador
* Coerção:
  + Coerção é um mapeamento implícito de valores de um tipo para valores de um tipo diferente. Ela é executada automaticamente, sempre que necessário
  + LPs modernas tendem a minimizar ou eliminar a coerção, pois ela não funciona bem com sobrecarga e polimorfismo. Por exemplo, no caso de Ada, o mapeamento é feito explicitamente Float(x)
* Subtipos e Heranças
  + Relacionada a capacidade de permitir que subtipos herdem operações de seus supertipos
  + Também conhecido por polimorfismo de inclusão
  + Pascal reconhece uma forma restrita de subtipo, ao permitir a definição de intervalos de qualquer tipo primitivo discreto T
  + Cada LP reconhece alguns subconjuntos de tipos como subtipos, mas não subconjuntos arbitrários. Por exemplo, nenhuma LP permite declarar uma variável que varie somente sobre os números inteiros primos

15/05/2023 – Sequenciadores

* Um sequenciador é uma construção que altera o fluxo de controle normal (uma única entrada e uma única saída)
* EX : saltos , escapes, execeoes
  + Saltos:
    - Comando simples , sequenciais , condicionais e iterativos possuem uma única entrada e uma única saída
    - Jump 🡪 transferência explicita de controle de um ponto do programa p o outro
  + Escapes:
    - Termina a execução de um comando. Transferência direta de controle.
  + Exceções:
    - Condições excepcionais que devem ser tratado