Resumen

1/1/22 19:27

diseño de programación

- Definición: conjunto de símbolos y de reglas para combinarlos.
- Construcción: Independiente de la arquitectura física.
  - 1 instrucción en diseño sólo nivel → Máquina no diseño máquina.
  - Utilizamos ficticiamente recursables, abstractas.

Gramática → especificación simbólica del diseño

Def. (Σ, N, S, P)

Σ → símbolos no terminales

N → símbolos terminales

P → producciones o reglas gramaticales

S → símbolo inicial. no terminal

Traductores

- Definición: traduce un texto en diseño específico y produce su equivalente en diseño máquina.

diseño fuente (código fuente) → traductor → diseño objeto (código máquina)

- Tipos:
  - Compiladores: traduce a diseño máquina completo. Necesita enlazador: completa programas siguiendo las instrucciones máquina y genera un exe para la máquina real.

  - Intérpretes: Ej. traduce y ejecuta una a una las instrucciones detectadas. No generan ningún programa objeto.

Intérpretes → Carga y ejecuta.

Especif. en diseño máq. → Carga + interpreta. → Datos → Resultados

Esquema de traducción

Programa fuente de CO_2

↓

Analizador sintáctico

Obtiene P_{ST} función de G.

Usa acciones semánticas $C(P_{ST}, C_S)$

↓

Analizador Semántico

$P \to \{Tipo_1, Tipo_2, Tipo_3, T_4\}$

Verificación semántica + tablas de símbolos.

↓

Generador de código

$T(Tipo_1) \to Tipo_2$

↓

Programa objeto de CO_2

Gramática

- Conceptos previos
  - Alfabeto: conjunto finito de símbolos.

    Binario = {0, 1}
  - Cadena: secuencia finita de símbolos usando un alfabeto.

    110, 0110, 1, 00001
  - Cadena terminal: elementos del alfabeto para formar cadenas.

    0, 1
  - Cadena no terminales: variables auxiliares que representan conjuntos de cadenas, no son del alfabeto.

    N = {N0, N1, N2, N3, N4}

Ejemplo: $P: \{\Sigma \to \Sigma 0X \mid (\Sigma)\}$ d

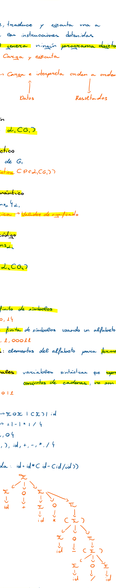
$0 \to \{1 - 1 + 1 / \}$

$V_n = \{\Sigma, 0\}$

$V_T = \{(,), d, +, -, *, /\}$

Modo de entrada: d+d*(d-(d/d))

Conexión?



Vemos las reglas, Árbol sintáctico que relata el texto

Fases en la construcción de un traductor:

Analizador léxico → Analizador sintáctico → Analizador semántico → Generador de código intermedio → Optimizador de código intermedio → Generador de código → Optimizador de código

Fase de análisis — Fase de síntesis

- Análisis léxico: agrupa secuencias, realiza reconocimiento de tokens, elimina comentarios superfluos y comentarios.

Ejemplo:	Token	Grupo	Lexema
	IF	Cond	if
	ID2I	Cond	i, i, i, i
	OP_COMP	Operador	<, >, <=
	NUM2	dato regular	pi, dato1, data...
	UNSIGN	Ch numérico	0, 10, 0.345

  - Expresiones regulares:
    - ? → d. = más veces
    - + → L = 0 veces
    - * → 0 = más veces
    - [ ] → Una de los caracteres de dentro
    - ( ) → Una de las secuencias de dentro

  - Tokens léxicos: cuando el autómata no pueda avanzar.

  Se usan expresiones regulares para describir patrones.

  Ejemplo: ID2I,UY → ([a-z,A-Z,0-9,_]+)*

- Análisis Sintáctico: analizar las secuencias de tokens y comprobar que son correctas sintácticamente.

  Usa derivación:
  - Si es correcta → no
  - Errores sintácticos → S no es correcta, hay errores sintácticos.

  Gramática: Clase de metodo sea G = (V_n, V_t, P, S) sea clase de cuerpo es:

  $P: \{A \to \alpha / A \in V_n, \alpha \in (V_n \cup V_t)^*\}$

  → cada no terminal es izquierda a la regla.

  Gramática: indica que es válido para una misma manera.

- Análisis semántico: analiza el significado de construcciones sintácticas, está ligado a la estructura sintáctica.

  Ejemplo: sel.seguro → ID2I,UY OP_RDO equivalente PVC

  variable ; ;

  Data compatibles:
  - Operación de diferencia
  - operación normal el tipo del resultado.

- Generación de código: se genera un archivo en diseño objeto (C máquina). + significado que el fuente a veces se hace un lenguaje de código intermedio

- Optimización de código: mejorar el código con algunos comportamientos.

Memoria

- Direcciones → Memoria → Registro → Ramas
- Tipos:
  - Dinámicas → variables locales y parámetros de función.
  - Dinámica controlada por el programa (Heap) → variables más dinámicas.

Ciclo de vida de un programa

hello.cpp → hello.i → hello.s → hello.o → hello/hello.exe

1. Preprocesado
2. Compilación
3. Ensamblado
4. Enlazado

- Compilación: proceso cada archivo de código fuente para generar cada archivo objeto.
  - Necesita:
    - Conocer código objeto y entrada externa
    - Asignar direcciones a los símbolos simbólicos y resolver referencias
    - Generar el mapeo de símbolos e información de depuración

- Enlazado: agrupa los archivos objetos y resuelve referencias a una única ejecutable.

  - Tareas:
    - Resolución de símbolos externos.
    - Agrupar secciones similares en regiones.
    - Relocalización de módulos y transformación de referencias.

  - Tipos de enlazado:
    - Módulos: cómo interno o en ejecución.
    - Tipos de módulo cada distinto difícil en módulo
      - Estático → estabilidad global
      - Dinámico → binario
      - Sin enlazado → estabilidad de bloque

    indica si el mismo nombre en dos ámbitos se referencia al mismo objeto o a dos distintos → Visibilidad de los identificadores

- Carga en memoria principal y ejecución

  Mantenimiento → en carga (Estático) / en ejecución (Dinámico)

  ↓

  SO: C(Run)

  + MMU (Módulo)

  - Diferencia entre archivo objeto y ejecutable:
    - Si el ejecutable, la cabecera (cara al punto de inicio del mismo, la primera información en carga)
    - Habrá en esta información de relocalización y se ha realizado en carga.

  - Formatos de archivo objeto y ejecutable
    - a.out: original de sist. Unix Seconds → text: código / data: inicializadas / bss: datos no inicializados
      - ELF: formato de depuración simbólica. Usado por Windows.
    - COFF: información de depuración simbólica. Usado por Windows.
    - XCOFF: similar a COFF: sin efectos estructuradas. Usado por Unix multiuser.

Bibliotecas

- Definición: colección de objetos relacionados entre sí.

  Permiten modularidad y reutilización del código.
- Tipos: Estáticas (.a) → siguen en el enlazado / Dinámicas (.so) → siguen en ejecución

  Estáticas: pasos:
  - Construimos el código fuente
  - Generamos el objeto
  - Archivamos el objeto (.a ocamos de biblioteca)
  - Enlazamos.

  Inconvenientes:
  - Dependencia memoria.
  - Si los actualizamos, debemos compilar los programas que los usan
  - Reducen ejecutables grandes.

  Dinámicas: se siguen en los procesos que los usan en ejecución

  El archivo se diferencia en que:
  - Contiene info de relocalización
  - . . . un índice de símbolos
  - Si la cabecera incluye info de punto de entrada.

Automatización de Compilación y Enlazado

Makefile: compila

Mejora la calidad del mantenimiento fiable y permite control de versiones.

Formato:
- makefile (C Microsoft, nmake)
- IDE: Entornos de Desarrollo Integrados