Методи за транслация

7. Съставни оператори

Съставните оператори се явяват или са явният механизъм за управление на последователността на изпълнение на операции. Те могат да се разглеждат и като универсални. Мета операции или над операции във езика за програмиране. Съществуват 2 основни конструкции на съставните оператори – разклонение и цикъл. Техните варианти и интерпретации лесно се превеждат в термините на последователност от прости оператори, етикети и условен оператор за преход. Следователно може да се счита, че единственият съставен оператор е условния оператор за преход, разглеждан като конструкция – разклонение което може да има разнообразни синтактични проявления във езиците за програмиране. (оператора if) Обикновено дефинираме съставния оператор като оператор чиито аргументи могат да бъдат (съставни) оператори. По този начин се получава **ниво на вложеност** на операторите в една програма. ОТ тази гледна точка под оператор на програма разбираме всеки оператор от 0-лево ниво на вложеност. (фиг1). Очевидно ако в транслатора се обособи универсално рекурсивно операция с име транслация на оператор то обработката на съставни оператори от ниво 0 няма да се различава от обработката на прости оператори от ниво 0. Този поглед върху съставните оператори ни позволява да разглеждаме всяка програма като последователност от декларации и оператори от ниво 0. В съвременните езици за програмиране имаме възможност да обособяваме такава последователност в така наречения съставен оператор (блок). Ако такъв блок е именуван то той се причислява към подпрограмите. Важното е, че чрез това понятие блок не формално се дефинира област на действие на имената в езика за програмиране. Това правило гласи: всяко име е валидно (познаваемо, разпознаваемо) само в рамките на блока където е декларирано. Тоест всяка програма неявно се счита за (именуван) блок. Следователно всеки допустим тип програмен обект трябва да бъде съотнесен със това правило за да могат да се получат сумарните условия за реализация на операцията позоваване (към или на програмен обект).

8. Област на действие на имената

Различаваме 2 вида област на действие.

А) статична

Когато отношението вложеност на 2 блока се определя от взаимното им разположение в текста на програмата. (фиг2).

Б) динамична област на действие

Когато това отношение на вложеност се определя от гледна точка на разположението на блоковете по време на изпълнение съгласно правилото за копиране. Примерно подпрограмата А активира подпрограмата Б (тоест Б е вложено в А).

Очевидно за всеки тип програмен обект се определя едно от тези 2 правила тоест това се оказва във описанието на езика или пък езика за програмиране предлага конструкции за посочване на предпочитано правило както по време на транслация така и по време на изпълнение.

II. Техники за реализация

1. Разпознаване на правилна програма

По същество всяка програма е краен низ от знаци. Програмата е правилна ако този низ е изводим от съответната формална граматика. Могат да се превеждат само правилни програми. Извод: Първата задача в процеса на транслация е да се провери дали низът е породен от дадена формална граматика. Един низ от знаци (програма) е породен от дадена граматика ако съществува дърво за което е в сила:

А) Коренът му е аксиомата на граматиката

Б) Листата му са знаците (под низове) на низа

В) Междинните възли съответстват на понятие (т.е. на нетерминални знаци) от граматиката.

(фиг3)

Това дърво наричаме синтактично дърво а процесът на построяването му граматичен разбор на програма. Съществуват 2 класически начина за строеж на синтактично дърво:

А) низходящ (TOP-DOWN) – когато дървото се строи от корена към листата.

Б) възходящ (UPSTAIRS) – когато дървото се строи от листата към корена.

Аргументи на алгоритъма изграждащ синтактичното дърво са програмата и граматиката. Представянето на програмата е ясно – това е низ от знаци в които знаците за делене на програмата на редове са заменени с разделители или пък са описани в граматиката като разделители. С низа от знаци, който именуваме PRG свързваме указател от пойнтер с име Pnt, който сочи текуш знак от PRG и операция Inc, която предвижва указателя Pnt на следващ знак (фиг4). Предполагаме, че PRG не е празен низ, а Pnt сочи първият му знак.

За да фиксираме представяне на граматиката приемаме, че тя е от тип LR(1) т.е. регулярна граматика (означава, че се чете по един знак от ляво на дясно) и налагаме следните съглашения:

А) На всеки нетерминален знак (на всяко понятие) съпоставяме по единствен начин цяло число заградено във скоби-човки. Като на аксиомата на граматиката съпоставяме низа <0> (фиг5)

Б) Азбука на граматиката наричаме съвкупността от терминалните и символи и символите определени от А)

В) Във всяка мета лингвистична формула на граматиката заменяме понятията със съответстващите им символи от дефинираната в Б) азбука. Това означава че всяка мета лингвистична формула става низ от символи.

Г) Сортираме получените низове във възходящ ред. Тоест мета лингвистичната формула (фиг6)

Д) Всяка мета лингвистична формула съдържаща повече от 1 алтернатива (знака или) заместваме със съответния брой формули без алтернативи като спазваме последователността на алтернативите. (фиг 6 пак)

Е) Полученото множество от формули записваме във таблица с в колони като понятието отляно на знака по дефиниция е записваме в 1-вата или лявата колона, а остатъкът от формулата без

знака по дефиниция е – в дясната (втората колона) Тази таблица именуваме с името GRM т.е. граматика.

<0> <0> <0>	<1> <2> <3>	P_0_1 P_0_2 P_0_3

Низходящ анализ – отличителна черта на низходящия анализ е целенасочеността. На всяка стъпка разпознавателя т.е. алгоритъма на разпознавателя формира цел и тя е да намери тая последователност от правила за писане в GRM чрез която може да бъде породена част от низа на програмата съответстващ на някаква синтактична категория (понятие от граматиката). Тази цел се постига чрез целенасочено изпробване на възможности (алтернативи). Тази идея може да бъде реализирана по много начини. Една възможна реализация е следната:

А) Реализираме логическа функция (предикат) която сравнява текущия знак от PRG със своя входен аргумент. Именуваме предиката с името EQU и налагаме условието, че при открито съвпадение се изпълнява Inc и се връща стойност истина.

Б) На всеки ред от таблицата GRM съпоставяме предикат, която връща стойност истина (true) ако разпознае низ сочен от Pnt който е породен от правилото записано в дясната страна на този ред. Очевидно е, че алгоритъмът на този предикат е линеен и се явява последователност от активирания (или извикване) на сродни на него предикати и функцията EQU. Ако резултатът е лъжа (false) е необходимо Pnt да бъде възстановен със стойността си при активирането на този предикат. Правилото за образуване на имена на такива предикати е

P__<номер на понятие>_<номер на алтернатива>

В) На всяко понятие дефинирано чрез повече от една алтернатива т.е. разположено на няколко последователно реда в GRM се съпоставя предикат с име P_<номер на понятие>. Алгоритъмът му е прост – последователно активиране на предикатите реализиращи алтернативите му докато не се получи резултат (true) истина или не се изчерпат алтернативите. Мерки за възстановяване на Pnt не са необходими.

Г) Ако изпълнението на предиката P_0 върне стойност true то низът PRG е правилна програма според граматиката GRM.

Този начин на реализация на низходящия граматичен разбор притежава следните достойнства:

А) Разпознавателят (програмата разпознавател) се променя лесно ако към граматиката се добавят или отстраняват правила. Това означава, че имаме възможност да развиваме езика.

Б) Синтактичното дърво не се строи във явен вид, но е възможно във всеки един от тези предикати да се добавят операции за изграждането му като използването на тези операции е опционално.

В) Чрез донастройване на така определените предикати се преминава от разпознаване на програма през интерпретация на програма за да се достигне до изграждане на компилатор.

Възходящ анализ (възходящ граматичен разбор) – най-късият ляв подниз на PRG който е непосредствено преводим към синтактична категория се нарича основа. Същността на възходящия анализ се заключава в търсенето на основа и замяната и със съответстващия и нетерминален знак. Това скъсява PRG от ляво. Това се извършва докато не се получи

нетерминалният знак на аксиомата или се окаже, че няма основа. Очевидно ефективността на този разпознавател зависи от ефективността на алгоритъма за търсене на основа. Една примерна проста реализация на възходящ разбор се заключава в следното: изгражда се рекурсивен предикат чиито глобални аргументи са PRG и GRM. Алгоритъмът е следният:

А) Търси се последователност определен от дясната колона на GRM който съвпада с началото на PRG.

Б) Ако не бъде открито съвпадение се връща стойност false лъжа

В) Ако се открие съвпадение то се замества със знака от лявото поле на съответния ред в GRM и предиката се активира отново рекурсивно

В1) Ако рекурсивното активиране върне стойност истина то се връща като резултат

В2) Ако рекурсивното активиране върне лъжа се възстановява състоянието от преди рекурсивното активиране и се продължава търсенето на съвпадение с низ от следващия ред на таблицата GRM

Г) Ако активирането на този предикат върне истина то PRG е правилна програма

След прилагането на някои програмистки техники и хитринки – този вид граматичен разбор може да бъде по-бърз сравнение със низходящия разбор. Проблема тук е, че за да се реализира преходът разпознаване-интерпретиране-компилация е необходимо да се изградят същите предикати както при низходящия разбор и да се натоварят със същите функции с изключение на дейности разпознаване. По-нататъка в лекциите и упражненията ще използваме низходящия граматичен разбор.

2. Етапи на транслация

Логически процеса на транслация се състои от 2 етапа – анализ на входната програма и синтез на изходната (обектна) програма. От своя страна етапът на анализ съдържа 3 части наречени съответно лексикален, синтактичен и семантичен анализ. Целта на лексикалния анализ е откриване на лексеми в входния низ. По същество този анализатор разпознава всеки позниз заграден от 2 разделителя като лексема от някакъв тип – например име, литерал… Целта на синтактичния анализатор е да открие синтактични структури, изрази, оператори, подпрограми итн… като се базира на работата на лексикалния анализатор. Целта на семантичния анализатор е да предаде смисъл (семантика) на разпознатите синтактични единици. По същество това представлява изграждане на съответни вътрешни за транслатора структури от данни в които се описват характеристиките на програмните елементи. Такива структури от данни са синтактичното дърво и таблицата с имена. В алгоритъмът му на работа трябва да бъдат закодирани и онези семантични особености на входния език, които не намират отражение в синтаксиса му. По същество семантичния анализатор е съвкупност от семантични анализатори съответващи на синтактичните единици. Като функциите му неявно се приписват на синтактичния и лексикалния анализатори. Примери за такива функции са: кодиране на лексеми и поддържане на таблица с имена, синтактично дърво итн… Може да се каже, че семантичния анализатор има 2 основни функции:

А) Поддържане на вътрешната за транслатора структура от данни

Б) Свързване или асоцииране на синтактичните единици с техните характеристики

Тъй като ще реализираме развитие на низходящ граматичен разбор, ще предполагаме, че със всеки предикат от (II – 1) са свързани (т.е се използват от него) 3 процедури. Съответно за лексикален, синтактичен и семантичен анализ като водеща е процедурата за синтактичен анализ. Това означава, че тя активира лексикалния анализ и в зависимост от резултатите използва семантичния анализатор.

Етапът на синтез съдържа 2 части: оптимизация и генериране на код. Оптимизацията означава преструктуриране на входната програма с някаква цел. Тоест оптимизатора е вид конвертор. Оптимизацията не е задължителна в процеса на транслация. Генерирането на код произвежда изходната програма като се базира върху структурите на данни създадени от етапа на анализ. Етапа на разделянето на времето по работата на анализ и синтез не е задължително. Ако входния език позволява тези етапи могат да се съвместят. От гледна точка на лекционния курс и входния език от упражненията, ще предполагаме, че със всеки предикат от реализацията на разпознавателя е свързано (използва се) още една процедура за генериране на код. Разбира се тази процедура ще бъде сложна тъй като в нея ще трябва да бъдат кодирани и лексиката и - синтаксиса и семантиката на изходния език. (фиг 7).

3. Лексикален анализ

Програма извършваща лексикален анализ наричаме лексикален анализатор или скенер. Основна задача на всеки скенер е да разпознава лексикалните единици във входния език. Ще обсъдим дейността на скенера в аспект низходящ граматичен разбор. За нашите нужни реализацията на скенер означава:

А) Определяне в мета езика множество L от видови понятия на родовото понятие лексема.

Б) Реализация на предикатите съответстващи на тези понятие вкл. и за понятието лексема. Като за всеки предикат се предвижда 1 входен и 1 изходен параметър.

В) Входния параметър винаги съдържа понятието или терминалния символ който трябва да се търси в PRG.

Г) Ако търсеното понятие не е открито в PRG то изходния параметър на предиката има неопределена стойност, а предиката връща лъжа

Д) Ако търсеното понятие е открито в PRG, то изходния параметър съдържа разпознатия подниз и неговото вътрешно за транслатора представяне. Например ако е открито число се връща и неговия двоичен еквивалент. Ако е разпозната ключова дума се връща и нейния код в транслатора итн… Това означава, че всеки предикат се натоварва и със семантични функции т.е. изграждане на вътрешно представяне на някои или на всички лексикални единици. Разбира се конвертирането към вътрешно представяне може да бъде разположено като семантична функция в предиката съответстващ на родовото понятие, но това би била несвойствена за него функция чието поддържане е трудно във условията на развиващ се (променящ се) входен език.

Е) Алгоритъмът за търсене бе описан за случая когато се търси понятие чиято мета лингвистична формула съдържа друго понятие.

Ж) Ако се търси низ от терминални знаци алгоритъма за търсене е ясен – търси се съвпадение от позицията на Pnt надясно и при откриване на съвпадение Pnt се премества непосредствено след откритият подниз като сочения от него знак не трябва да бъде разделител.

Извод: Операцията за придвижване на Pnt трябва да разпознава разделителите и коментарите и да ги игнорира.

Извод: Ключът към реализирането на ефективни и надеждни анализатор и скенер съгласно описаната формална процедура за изграждането им се крие в доброто описание на входния език. От тази гледна точка термини които затрудняват изучаването на езика могат да се окажат проблемни от гледна точка на реализацията на транслатора.

4. Синтактичен анализ

Целта на синтактичния анализ е да разпознае синтактичните единици различни от лексеми и да изгради ако е необходимо даннова структура (например синтактично дърво), която ще се изпълни със съдържание от последващия семантичен анализ. Ние обсъдихме алгоритъма на синтактичния анализатор по същество във въпрос номер 2 във II. Тук ще обсъдим само няколко препоръки от гледна точка на:

А) избрания низходящ синтактичен разбор

Б) целта която преследваме като реализация

За да се постигне тази цел е необходимо:

А) В реализацията да се въведат няколко променливи флагове указващи режима на работа на транслатора (разпознаване анализ и синтез). Съвкупността от стойностите на тези флагове наричаме режим на работа на транслатора. Очевидно е, че режима може да бъде усложнен чрез детайлизиране на тези флагове. Например генериране на код, строеж на синтактично дърво итн… Но за това се изисква добро предварително проектиране и дисциплина на реализацията. Очевидно алгоритъма на всеки предикат трябва да бъде съобразен със режима в който се използва.

Б) За да не се изгражда синтактичното дърво в явен вид е достатъчно да се поддържа системен стек FIFO-стек в който всеки предикат анализиращ алтернатива записва номера на мета лингвистичната формула (т.е. името на предиката) чието анализиране е породило неговият успешен край на работата (авто документиране на работата на транслатора).

В) Със всеки предикат се свързват една или няколко семантични процедури (в зависимост от възможните режими на работа на транслатора) и се описва взаимовръзката между тях.

Г) Тей като независимо от режима множеството от асоциации (въпр номер 3 от I) трябва да се поддържа е необходимо да се обособи модул (или част от програмата) от базови семантични процедури. Тяхното предназначение е да реализират операциите позоваване на име и достъп до позиция като използват и поддържат множеството от асоциации. Обикновено това множество от асоциации като вътрешни данни се представя чрез така наречената таблица на имената. По същество това е стек чиито елементи са асоциации, но при търсене на асоциация той се разглежда като таблица чийто редове се номерират от върха към дъното. Тези особености на реализацията на таблицата с имената са породени от статичната област на действие на имената във входния език за който ще правим транслатор на упражненията.

Реализацията на тези 4 пункта ще разглеждаме като реализация на синтактичен анализатор. Очевидно е че неговата реализация предполага проектиране на семантичен анализатор. Освен това трябва да имаме предвид, че:

А) Синтактичния анализатор използва семантичния анализатор. Т.е. извършен синтактичен анализ означава и извършен семантичен анализ.

Б) След края на синтактичния анализ всички имена на програмни обекти са заменени с вътрешни за транслатора имена, а именно – номер на ред от таблицата с имената.

5. Семантичен анализ

По същество семантичния анализ изпълва със съдържание системните структури от данни на транслатора. Тъй като обсъдихме функционалността на базовите семантични процедури, то за останалите можем да кажем, че функционалността им се заключава във генериране на позиция в оперативната памет на програмен елемент. Позицията е памет съдържаща описание на характеристиките на програмен елемент (и неговата текуща стойност). От тази гледна точка позицията с е състои от дескриптор и код. Добре е дескрипторите да описват всички характеристики на програмните елементи независимо от това, че:

А) Някой от характеристиките могат да бъдат наследени от други програмни елементи или пък да са функция от другите характеристики на елемента.

Б) Някой от характеристиките могат да не бъдат използвани в етапа на синтез.

Този подход ще позволи да се реализират множество варианти на работа на транслатора в етапа синтез. Съвкупността от всичките даннови структури описващи една програма трябва да се разглеждат като позиция на тази програма. Тази гледна точка решава проблема със асоциациите на подпрограмите. Разбира се в този случай операцията търсене на асоциация би се усложнила в случай на статична област на действие на имената. Този проблем има елегантно решение в следните няколко пункта:

А) Поддържа се единна таблица с имена в която имената се записват в реда на срещането си в програмата. Съгласно приетото правило за четене.

Б) На всяка подпрограма съответства единствена асоциация в тази таблица.

В) Кода на позицията на всяка подпрограма е множество (списък) от кодове предопределени от възможните режими на синтез. Например за интерпретация той трябва да съдържа текста или сорса на самата подпрограма, а при компилация текста на преведената или изходната подпрограма.

Извод: реализацията на семантичен анализатор предполага извършено проектиране на етапа синтез на програмата.

6. Интерпретиране на програмата

По същество интерпретацията е емулация на входния език в термините на реализационния език. От гледна точка на избрания в лекциите подход това означава на всеки предикат да се съпостави интерпретационна процедура или метод който се използва в режим интерпретация непосредствено преди изход от предиката с стойност истина. В алгоритмите на тази интерпретационни процедури са известни от семантиката на входния език. Остава единствено за реализацията на тези алгоритми да бъдат подбрани онези най-прости изразни средства на реализационния език, които:

А) имат директен аналог в изходния език

Б) известен е (възможен е) преводът им в термините на изходния език

Очевидно най-добре е да се спазват семантиката на изходния език като се пише в термините на реализационния език. Това е правилния подход, защото:

А) Ще се проявят онези вътрешни обекти и операции, които ще трябва да се закодират в последствие в термините на изходния език.

Б) Преминаването от интерпретация към генериране на код става рутинна (проста, елементарна) процедура.

Извод: Реализацията на интерпретатор изисква много добро владеене на семантиката на реализационния език от гледна точка на изходния език. В частност необходима е емулация на механизма за управление на паметта на изходния език като реализацията на интерпретатора трябва да започне с реализацията на този механизъм.

7. Генериране на изходна програма

По същество генерирането на код е най-неангажиращата дейност в процеса на създаване на транслатор. Тя се заключава в:

А) Прост превод на алгоритмите на интерпретационните процедури в термините на изходния език.

Б) Оформяне на този новополучен текст в съответствие със изискванията на изходния език.

От гледна точка на следвания в лекциите подход условието на подточка Б) би трябвало да отпадне. Разбира се при превода (от реализационен в изходен) трябва да се отчете, че част от текста на интерпретационните процедури е излишен. Например текстове описващи механизмите за управление на данните, за управление на последователността на действие и за управление на паметта на изходния език. От гледна точка на генерирането на код тези текстове (механизми) единствено гарантират еквивалентността на входната и изходната програма.

Извод: най-тежък като работа е интерпретаторът тъй като при неговата реализация програмистът трябва да ползва едновременно 3 семантики – на входния, на реализационния и на изходния езици, и 2 синтаксиса: на изходния и на реализационния език.

8. Оптимизация

Под оптимизация на програма се разбира нейната обработка със цел да бъде подобрено някое нейно качество без да се влошава съществено други нейни качества. Например, трудно е да направим програмата по-бърза без да увеличим използвания обем оперативна памет. Най-често под оптимизация се разбират тези 2 критерия: скорост и използван обем оперативна памет. Различават се 2 групи оптимизация: машинно-независима и машинно-зависима.

Машинно независима оптимизация – тази оптимизация се извършва върху вътрешното представяне на входната програма. Тя е съвкупност от методи целящи:

А) Изчисляване на изрази формирани от литерали

Б) Елиминиране на повтарящи се изчисления

В) Отстраняване на безсмислени оператори или операции

Г) Минимизиране на броя на използваните променливи.

Д) Минимизиране на операциите позоваване на име и достъп до паметта (позиция)

Е) Минимизиране на броя на формалните параметри на подпрограмата

Ж) Прилагане на правилото за копиране за еднократно активирани подпрограми

З) Прилагане на правилото за копиране за онези подпрограми чиито код е по-малък от кода на алгоритъма за предаване на параметри

И) Изнасяне извън цикъл на операции чиито аргументи не зависят от параметъра на цикъла.

Й) Замяна на операции в цикъл с по-бързи

К) Линеаризация на многомерни масиви в цикъл

Л) Обединяване, сливане и разчленяване на цикли

М) И други..

Машинно-зависима оптимизация – тя се поддържа (извършва) на ниво вътрешно представяне на изходната програма. Зависи от конкретната изходна виртуална машина (езика) и в случая на компилатори цели използването на най-бързите команди и рационално използване на различните видове памет: бързи регистри, бърза свръх оперативна памет, оперативна памет, външна памет. Основните алгоритми които се използват при този вид оптимизация са:

А) Оптимизация на броя на използваните регистри.

Б) Минимизация на броя на прехвърлянията на данни между регистри и оперативна памет

В) Ефективното използване на индексните регистри

Обикновено оптимизаторът се реализира като отделен модул, тъй като алгоритмите му са тежки и сложни и не необходимо да бъдат използвани при всяка транслация.

9. Схеми за реализация на транслатор

Всеки реализатор на транслатор използва 3 езика: входен, реализационен и изходен. Тази триада от езици схематично се изобразява чрез така наречените Т диаграми (фиг 8). Видно е, че съвпадението на входния и реализационен език е невъзможно (в общия случай). А съвпадение на входния и изходния език е безсмислено. Извода е, че наредената двойка (реализационен език, изходен език) определя основните схеми за реализация на транслатора. Обикновено терминът транслатор се употребява когато:

А) Или реализационния език и изходния език съвпадат (фиг 9)

Б) Или за реализационния език има транслатор до изходния език (фиг 10)

Казаното се илюстрира чрез следните 2 Т диаграми (фиг 9 и 10)

Втората диаграма (фиг 10) показва, че и реализационния език може да се счита за машинно-ориентиран. Обикновено език от високо ниво за който имаме транслатор се казва че той е реализиран език или език за програмиране и в тоя смисъл използваме термина машинно-зависим.

Обикновено нереализираните езици се наричат алгоритмични езици.

Ако изходния език е от високо ниво, но е реализиран се получава схемата (фиг 11). Тази последна схема означава, че и входния език се нарича машинно-ориентиран (или реализиран език) ако чрез някоя от тези схеми за транслация се окаже че за него има изпълнител. Следователно терминът транслатор се употребява само когато реализационният изходният езици са машинно-ориентирани (т.е. реализирани езици) и съвпадат. Когато реализационният и изходният езици са машинно-ориентирани, но не съвпадат се употребява терминът cross-

translator. Това означава, че изходната програма ще работи на процесор които е различен от този на който работи транслатора. Очевидно реализацията на транслатор в термините на машинно-ориентиран реализационен език е тежка работа. По-добре е ако реализационния език е език от високо ниво. В този случай е възможно реализационния език да бъде под множество на входния език. Това поражда идеята за нарастване на изразителната сила на реализационния език със собствени сили (boot-stripping).

Съществуват различни схеми за реализация на транслатор с този механизъм като целта е минимизация на времето и усилията на реализация на транслатор.