ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
УФИМСКИЙ ГОСУДАРСТВЕННЫЙ АВИАЦИОННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра ТК

курс лекций по дисциплине

Системное программное обеспечение

Тема: Семантический анализ

Преподаватель: к.т.н., доцент Карамзина А.Г.

Тема № 15

Семантический анализ

- Назначение семантического анализа
- Этапы семантического анализа
- Идентификация лексических единиц языков программирования

Назначение семантического анализа

Полный распознаватель для языка программирования можно построить на основе распознавателя К3-языков.

Однако такой распознаватель имеет экспоненциальную зависимость требуемых для выполнения разбора цепочки вычислительных ресурсов от длины входной цепочки.

Компилятор, построенный на основе такого распознавателя, будет неэффективным с точки зрения либо скорости работы, либо объема необходимой памяти.

Поэтому такие компиляторы практически не используются, а все реально существующие компиляторы на этапе разбора входных цепочек проверяют только синтаксические конструкции входного языка, не учитывая его семантику.

С целью повышения эффективности компилятора разбор цепочек входного языка выполняется в два этапа:

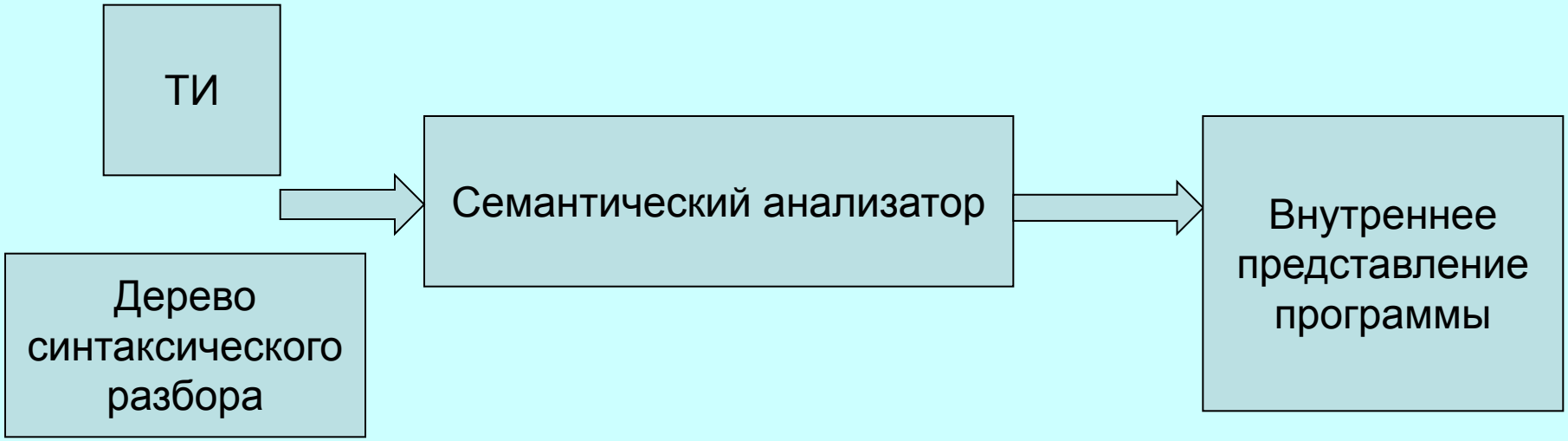
- синтаксический разбор на основе распознавателя одного из известных классов КС-языков;
- семантический анализ входной цепочки.

Назначение семантического анализа

Для проверки семантической правильности входной программы необходимо иметь всю информацию о найденных лексических единицах языка.

Эта информация помещается в таблицу лексем на основе конструкций, найденных синтаксическим распознавателем. Примерами таких конструкций являются блоки описания констант и идентификаторов (*если они предусмотрены семантикой языка*) или операторы, где тот или иной идентификатор встречается впервые (*если описание происходит по факту первого использования*).

Поэтому полный семантический анализ входной программы может быть произведен только после полного завершения ее синтаксического анализа.



Назначение семантического анализа

Семантический анализ обычно выполняется на двух этапах компиляции:

- на этапе синтаксического анализа (*каждый раз по завершении распознавания определенной синтаксической конструкции (как правило, это процедуры, функции и блоки операторов входного языка*) *входного языка выполняется ее семантическая проверка на основе имеющихся в таблице идентификаторов данных*).

- в начале этапа подготовки к генерации кода (*после завершения всей фазы синтаксического анализа, выполняется полный семантический анализ программы на основании данных в таблице идентификаторов (сюда попадает, например, поиск неописанных идентификаторов)*).

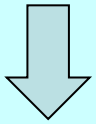
Иногда семантический анализ

В каждом компиляторе обычно присутствуют оба варианта семантического анализатора. Конкретная их реализация зависит от версии компилятора и семантики входного языка.

Этапы семантического анализа

Семантический анализатор выполняет следующие основные действия:

1. ***проверку соблюдения во входной программе семантических соглашений входного языка***



заключается в сопоставлении входных цепочек программы с требованиями семантики входного языка программирования.

Каждый язык программирования имеет четко заданные и специфицированные семантические соглашения, которые не могут быть проверены на этапе синтаксического анализа. Именно их в первую очередь проверяет семантический анализатор.

Этапы семантического анализа

Примерами таких соглашений являются следующие требования:

- *каждая метка, на которую есть ссылка, должна один раз присутствовать в программе;*
- *каждый идентификатор должен быть описан один раз, и ни один идентификатор не может быть описан более одного раза (с учетом блочной структуры описаний);*
- *все операнды в выражениях и операциях должны иметь типы, допустимые для данного выражения или операции;*
- *типы переменных в выражениях должны быть согласованы между собой;*
- *при вызове процедур и функций число и типы фактических параметров должны быть согла*

Конкретный состав требований, которые должен проверять семантический анализатор, жестко связан с семантикой входного языка (*например, некоторые языки допускают не описывать идентификаторы определенных типов*).

Этапы семантического анализа

если не описан один из идентификаторов - ошибка

a := *b* + *c*;

если это числовые переменные и константы, то выполняется оператор сложения

если это строковые переменные и константы, то выполняется оператор конкатенации строк

идентификатор *a* не может быть константой – иначе нарушается семантика оператора присваивания

если одни из идентификаторов числа, а другие – строки, или идентификаторы массивов или структур, то такое сочетание аргументов для операции сложения недопустимо

Этапы семантического анализа

Семантический анализатор выполняет следующие основные действия:

2. Существуют правила преобразования типов, принятые для данного языка. Эти преобразования может сделать разработчик программы – но тогда преобразования типов в явном виде будут присутствовать в тексте входной программы (*в рассмотренном примере это не так*).

В другом случае это делает код, порождаемый компилятором, когда преобразования типов в явном виде в тексте программы не присутствуют, но неявно предусмотрены семантическими соглашениями языка.

Для этого в составе библиотек функций, доступных компилятору, должны быть функции преобразования. Вызовы этих функций как раз и будут встроены в текст результирующей программы для удовлетворения семантических соглашений о преобразованиях типов во входном языке, хотя в тексте программы в явном виде они не присутствуют.

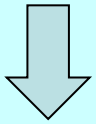
Чтобы это произошло, эти функции должны быть встроены и во внутреннее представление программы в компиляторе. За это также отвечает семантический анализатор.

- присвоение результата переменной *a*.

Этапы семантического анализа

Семантический анализатор выполняет следующие основные действия:

3. ***проверку элементарных семантических (смысловых) норм языков программирования, напрямую не связанных с входным языком***



является сервисной функцией, которую предоставляют большинство современных компиляторов.

Эта функция обеспечивает проверку компилятором некоторых соглашений, применимых к большинству современных языков программирования, выполнение которых связано со смыслом как всей входной программы в целом, так и отдельных ее фрагментов.

Этапы семантического анализа

Примерами таких соглашений являются следующие требования:

- *каждая переменная или константа должна хотя бы один раз использоваться в программе;*
- *каждая переменная должна быть определена до ее первого использования при любом ходе выполнения программы (первому использованию переменной должно всегда предшествовать присвоение ей какого-либо значения);*
- *результат функции должен быть определен при любом ходе ее выполнения;*
- *каждый оператор в исходной программе должен иметь возможность хотя бы один раз выполниться;*
- *операторы условия и выбора должны предусматривать возможность хода выполнения программы*
- *операторы цикла цикла.*

Конкретный состав проверяемых соглашений зависит от семантики языка. Однако в отличие от семантических требований языка, строго проверяемых семантическим анализатором, выполнение данных соглашений не является обязательным.

Идентификация лексических единиц языков программирования

Идентификация переменных, типов, процедур, функций и других лексических единиц языков программирования – это установление однозначного соответствия между данными объектами и их именами в тексте исходной программы.

Идентификация лексических единиц языка чаще всего выполняется на этапе семантического анализа.

Примерный перечень действий компиляторов для идентификации переменных, констант, функций, процедур и других лексических единиц языка:

- имена локальных переменных дополняются именами тех блоков (*функций, процедур*), в которых эти переменные описаны;
- имена внутренних переменных и функций модулей исходной программы дополняются именем самих модулей, причем это касается только внутренних имен и не должно происходить, если переменная или функция доступны извне;
- имена процедур и функций, принадлежащих объектам (*классам*), в объектно-ориентированных языках программирования дополняются наименованием типа объекта (*класса*), которому они принадлежат;
- имена процедур и функций модифицируются в зависимости от типов их формальных аргументов.