А. Ю. Шевчук

Компилятор языка Etherbox2

Научный руководитель: к.т.н. Ю. В. Шевчук

Аннотация. Статья посвящена разработке компилятора языка Etherbox2, разработанного в ИПС им. А. К. Айламазяна РАН в рамках проекта «Сенсор» в качестве языка управления сенсорной сетью. Рассматриваются особенности целевой машины и языка Etherbox2, а также средства реализации компилятора.

Ключевые слова и фразы: Компилятор, сенсорная сеть, Etherbox32, Etherbox2.

1. Введение

Данная работа посвящена разработке компилятора Си-подобного языка Etherbox2, используемого для управления сенсорной сетью, разработанной в ИПС РАН в рамках проекта «Сенсор» [1]. Чтобы составить представление о контексте задачи, рассмотрим структуру сенсорной сети.

В упрощенном представлении, сенсорная сеть состоит из управляющей станции и подключенных к ней узлов сенсорной сети. В каждом из таких узлов работает устройство Etherbox32 [2], выполняющее некоторую программу, которая и определяет поведение узла. Как правило, выполняемая узлом программа осуществляет своевременный опрос подключенных к нему датчиков и оповещает управляющую станцию о результатах измерений.

В задачи управляющей станции входит сбор и анализ информации, полученной с узлов, а также изменение поведения сенсорной сети. Если требуется изменить поведение некоторого узла, управляющая станция передает ему на выполнение новую программу.

Такая программа называется „Proglet“[1] и является результатом компиляции программы, написанной на языке Etherbox2, в байткод[2]

[1]По аналогии с Piglet — Пятачок, персонаж сказки А.А.Милна, „очень маленькое существо.“

[2]Код низкого уровня, исполняемый виртуальной машиной. Трансляция в байткод занимает промежуточное положение между интерпретацией и компиляцией в машинный код.

виртуальной машины, работающей на устройстве Etherbox32. Последовательно этот процесс выглядит следующим образом:

(1) Оператор управляющей станции пишет программу (Proglet) на языке Etherbox2.
(2) Программа компилируется в байткод виртуальной машины Etherbox32vm.
(3) Управляющая станция передает получившийся байткод на выполнение на узел сенсорной сети.
(4) Узел выполняет программу, время от времени передавая ответы на управляющую станцию.

Компилятор языка Etherbox2 в байткод виртуальной машины Etherbox32vm, работающий на этапе (2), и является предметом данной работы.

2. Целевая машина

Как уже упоминалось выше, целевой машиной компилятора является виртуальная машина Etherbox32vm [3]. Это стековая машина с одним стеком, реализованная в виде интерпретатора, написанного на языке Си.

Proglet представлен в памяти устройства Etherbox32 в виде последовательности (цепочки) команд. В общем случае, команда представляет собой несколько последовательно расположенных байт, первый из которых содержит код самой команды, а остальные — аргументы команды. Однако многие команды не имеют аргументов (или берут их со стека); такие команды представляются всего одним байтом.

Строение каждой команды описано в интерпретаторе как структура. Например, команда i2c описана следующим образом:

```
struct ebox_cmd_i2c {
  unsigned char cmd_code; /* 15 */
  unsigned char count;  /* count of bytes in data[] */
  unsigned char ocount;  /* bytes really transferred */
  unsigned char data[0];
}
```

На рис. 1 приведен пример фрагмента цепочки, состоящего из четырех команд. Как можно заметить, команды состоят из разного количества байт. Два последних байта, принадлежащих к команде i2c, не входят непосредственно в ее структуру, однако доступ к ним может быть осуществлен через указатель data.

Рис. 1. Пример фрагмента цепочки

Результат работы команды помещается либо на вычислительный стек, либо в „хвост“ самой команды, как, например, в случае с командой i2c. В последнем случае, доступ к результату работы команды осуществляется с помощью специального механизма якорей, который будет описан ниже (п. 3.2).

3. Язык Etherbox2

Язык Etherbox2 был разработан как модификация языка Си, причем большое внимание уделялось тому, чтобы минимизировать отличия языка от оригинала. Такой выбор обусловлен двумя факторами: во-первых, многие средства языка Си хорошо соответствуют специфике задачи, а во-вторых, такой подход избавляет оператора управляющей станции от необходимости изучать новый язык — ему придется изучить лишь немногочисленные особенности диалекта, которые мы сейчас и рассмотрим.

3.1. Измененная семантика вызова функций

Синтаксис вызова функций в языке Etherbox2 перегружен. Условно можно говорить о существовании трех различных видов функций, использующих общий синтаксис.

3.1.1. *Пользовательские функции*

Способ вызова пользовательских функций тот же, что и в оригинальном Си. Аргументы компилируются и кладутся на стек, затем управление передается вызванной функции. Потом функция возвращает управление в прежнюю точку, возможно оставив результат на стеке.

3.1.2. *Команды Etherbox32vm*

Вызов команд Etherbox32vm предназначен для записи в цепочку некоторых команд виртуальной машины. Аргументы (если они есть) по мере необходимости вычисляются и записываются в цепочку в соответствии со структурой команды (см. п. 2). Для удобства чтения было принято имена вызываемых команд начинать с подчерка.

Например:

```
_mspause();
_i2c(1,2,3,4,5);
```

3.1.3. *Псевдо-функции*

Псевдо-функции — это своего рода макросы, значение которых вычисляется во время компиляции, а определения передаются компилятору как входные данные. Такие функции всегда возвращают список констант, а значит могут быть подставлены в программу в качестве инициализатора массива или в качестве аргумента другой функции.

Например:

```
int a[] = { PSEUDO() };
func( PSEUDO() );
```

3.2. Якоря (anchors)

Как уже упоминалось выше (см. п. 2), строение каждой команды может быть представлено в виде структуры. Такое представление может быть удобно не только для интерпретации команд: если определения этих структур видны в самой программе, синтаксис доступа к полям структуры можно использовать для доступа к аргументам и результатам работы сложных команд.

Для того чтобы связать переменную типа „структура“ с командой в памяти, в язык был введен механизм якорей. Этот механизм состоит из двух конструкций.

Сначала якорь устанавливается на начало некоторой команды:

```
x:: _i2c (ARGS());
```

Якоря могут быть установлены только перед вызовами команд (см. п. 3.1.2), и никакими другими конструкциям языка. Отличие якорей от обычных меток состоит в том, что якорь всегда будет помещен непосредственно на начало команды, даже если вызов команды был скомпилирован так, что ей предшествует некоторый код. Якоря

содержатся в собственном пространстве имен с глобальной видимостью.

Далее декларируется переменная, связанная с этим якорем:

```
anchored struct ebox_cmd_i2c x;
```

Класс памяти *anchored* означает, что адрес структуры x будет соответствовать адресу одноименного якоря. Такие переменные находятся в общем пространстве имен и имеют вложенную видимость. Теперь к аргументам команды i2c можно обращаться как к полям структуры x:

```
int a = x.ocount;
```

Конструкции объявления якоря и привязки к нему переменной могут располагаться в программе в любом порядке.

4. Структура компилятора

Компилятор является расширением модуля управления устройствами Etherbox32, который написан на языке программирования Perl. Поэтому компилятор написан на том же языке. Кроме того, Perl является вполне подходящим средством реализации, предоставляя, в частности, удобные средства для работы с текстом.

Компилятор состоит из следующих относительно независимых частей [7].

4.1. Препроцессор

Препроцессор языка Etherbox2 несколько отличается от препроцессора языка Си:

- Изменена семантика директивы include, в качестве аргумента которой передается не имя непосредственно подключаемого файла, а имя Perl-модуля, который содержит функцию генерации заголовка, подставляемого в исходный файл.
- Формат возвращаемого препроцессором текста позаимствован у GNU C Preprocessor [5], за тем только исключением, что директивы контроля строк могут иметь только флаги 1 и 2, обозначающие начало и конец подключаемого файла.

4.2. Синтаксический анализатор (parser)

Синтаксический анализ программы производится при помощи компилятора компиляторов [4] Parse::RecDescent [6], генерирующего синтаксические анализаторы, работающие по принципу рекурсивного спуска [8].

Грамматика языка Си принята почти без изменений: добавлены только одна новая конструкция и один класс памяти (см. п. 3.2).

Результатом синтаксического анализа программы является внутреннее представление программы, так называемое дерево разбора (parse-tree) [9].

На рис. 2 представлен пример дерева разбора для выражения `x[5]`= y + z*3 .

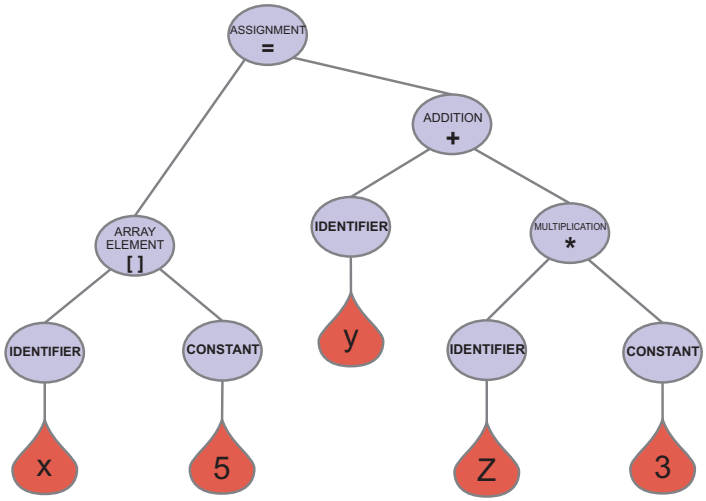

Рис. 2. Дерево разбора выражения x[5] = y + z*3

4.3. Генератор кода

Генератор кода объединен с семантическим анализатором и представляет из себя рекурсивный обработчик тегов дерева разбора. Рекурсивная функция совершает обход дерева в глубину, транслируя

каждую конструкцию языка в некоторую последовательность команд виртуальной машины.

Одной из существенных особенностей компилятора является то, что автоматические переменные хранятся на вычислительном стеке виртуальной машины. Это влечет за собой и другие особенности:

- Для работы с такими переменными были введены дополнительные команды взаимодействия со стеком, позволяющие дублировать (команда dupn) и изменять (команда pushn) значения, находящиеся на некоторой глубине стека.
- При возврате из функций необходимо не только вернуть управление в точку вызова функции, но и очистить стек от значений, помещенных туда в процессе ее выполнения. Для этого была введена специальная команда smartret виртуальной машины.
- Чтобы иметь актуальную информацию о местонахождении переменных на стеке, приходится отслеживать все изменения, происходящие со стеком. Зная количество элементов на стеке в момент создания переменной и в настоящий момент, всегда можно вычислить, на какой глубине стека находится интересующее нас значение.
- Возрастает также и роль системы регуляции стека: при выходе из каждого блока программы нужно синхронизировать количество элементов на стеке с тем, каким оно было до вхождения в блок; в противном случае нарушается адресация автоматических переменных, и происходит утечка памяти, вследствие чего возможно переполнение стека.

Статические же переменные хранятся в цепочке в области аргумента команды data, которая помещается компилятором в самое начало цепочки.

4.4. Ассемблер

Ассемблер производит заключительный этап компиляции: трансляцию последовательности команд языка ассемблера в байткод виртуальной машины. Трансляция производится в несколько проходов: если метка используется в коде программы до того места, где она объявлена, то адрес такой метки сразу определить невозможно. В таком случае приходится совершить очередной проход по коду, в результате

которого адрес метки станет известен. Если вся таблица имен оказывается заполненной, необходимость в очередном проходе отпадает, и ассемблирование завершается.

5. Заключение

По итогам данной работы достигнуты следующие результаты:

- На базе языка Си разработан язык Etherbox2, хорошо отвечающий задачам программирования устройств Etherbox32.
- Разработаны принципы компиляции программ, написанных на языке Etherbox2, подобран инструментарий разработки компилятора.
- В виртуальную машину Etherbox32vm добавлены команды, позволяющие более эффективно компилировать некоторые конструкции языка Etherbox2.
- Реализован компилятор языка Etherbox2.

Чтобы составить некоторое представление об объемах задачи, приведем примерную статистику количества строк, занимаемых различными частями компилятора:

- Синтаксический анализатор (грамматика языка, возвращаемый правилами грамматики код) — 700 строк.
- Ассемблер — 600 строк.
- Генератор кода, семантический анализатор — 1800 строк.
- Препроцессор — 100 строк.

Однако многие конструкции языка реализованы лишь для частных случаев, а значит, в ходе дальнейшей разработки следует ожидать увеличения объема кода.

В данный момент компилятор находится на стадии отладки на прикладных задачах. Введение в эксплуатацию планируется в ближайшее время.

Список литературы

[1] Проект «Сенсор», `http://wiki.botik.ru/Sensor/`.
[2] Спецификация Etherbox32, `http://wiki.botik.ru/Sensor/EtherBox32`.
[3] Команды Etherbox32vm, `http://wiki.botik.ru/Sensor/EtherBox32vm`.
[4] Compiler-Compiler, `http://en.wikipedia.org/wiki/Compiler-compiler`.
[5] The C Preprocessor, `http://gcc.gnu.org/onlinedocs/cpp/index.html`.
[6] Parse::Recdescent, `http://search.cpan.org/dist/Parse-RecDescent/`.

[7] А.Ахо, Р.Сети, Д.Ульман Компиляторы. Принципы, технологии, инструменты, 2003. — 22–44 с.

[8] А.Ахо, Р.Сети, Д.Ульман Компиляторы. Принципы, технологии, инструменты, 2003. — 188–200 с.

[9] А.Ахо, Р.Сети, Д.Ульман Компиляторы. Принципы, технологии, инструменты, 2003. — 47–48 с.

УГП, 5М51

A. Y. Shevchuk. *Etherbox2 language compiler* // Proceedings of Junior research and development conference of Ailamazyan Pereslavl university. — Pereslavl, 2010. — p. 91–99. (*in Russian*).

Abstract. The paper covers development of the compiler of Etherbox2 language, invented in PSI RAS within "Sensor" project. The language is intended for control over the sensor network. Special attention is paid to the features of Etherbox2 language and to the design decisions.

Key Words and Phrases: Compiler, sensor network, Etherbox32, Etherbox2.