Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

Автоматическая трансляция проекта Dypgen с языка OCaml на язык F#

Курсовая работа студента 345 группы
Баранова Эдуарда Сергеевича

Научный руководитель ……. Я. А. Кириленко

Санкт-Петербург
2010

Оглавление

Введение

YaccConstructor[1] - генератор синтаксических анализаторов для произвольных контекстно-свободных грамматик. Основной областью применения данного инструмента является реинжиниринг программного обеспечения. Проект разрабатывается на кафедре системного программирования СПБГУ на языке F#.

F#[3] — это мультипарадигмальный язык программирования, разработанный в подразделении Microsoft Research и предназначенный для исполнения на платформе Microsoft .NET. Он сочетает в себе выразительность функциональных языков с возможностями и объектной моделью .NET. Это диалект языка ML, считается наследником OCaml. F# изначально разрабатывал Don Syme в Microsoft Research, а сейчас работа над языком продолжается в Microsoft Developer Division. Распространяется, как часть Visual Studio 2010, но возможна интеграция в более ранние версии.

Одной из главных особенностей построения анализаторов для решения задач реинжиниринга является частая необходимость работы с устаревшими языками, которые имеют большое количество плохо специфицированных диалектов, что затрудняет задание грамматики языка[4]. Это накладывает дополнительные требования на инструмент, в том числе необходима поддержка неоднозначных контекстно-свободных грамматик, так как они являются наиболее распространённым способом задания языков.

Этому требованию удовлетворяет GLR-алгоритм. Он обрабатывает все возможные варианты и создает множество деревьев разбора (лес). Стоит отметить, что по производительности такой анализатор, являясь некоторой "надстройкой" над LR-анализатором, незначительно ему уступает. На сегодняшний день в соотношении производительность/класс разбираемых языков GLR-алгоритм выглядит наиболее предпочтительно.

Рассматривалось два пути получения GLR-генератора в инструменте YaccConstructor. Первый — написать свой новый генератор. Второй — использовать уже готовую реализацию, возможно на другом языке программирования. К сожалению, не существует GLR-генератора под платформу .NET. Если рассматривать продукты под другие платформы, то самым современным является Dypgen.

Dypgen[2] — это GLR-генератор для Objective Caml. Основные достоинства Dypgen:

- Может обрабатывать неоднозначные грамматики и выдаёт список деревьев разбора.

• Для однозначных грамматик синтаксический анализ позволяет более естественно определить грамматику. Возможно выделить определение, подходящее для документации, прямо из исходного файла анализатора.

• Неоднозначности можно убрать, введя приоритеты и отношения между ними, что позволяет выразить грамматику естественно.

• Грамматики являются саморасширяемыми, т. е. действие может добавить новое правило в данную грамматику. Более того, модификация может быть локальной. Новая грамматика действительна только для ограниченной части проанализированных входных данных.

• Также его можно использовать как лексический генератор...

•Это «живой» проект, его разработка продолжается.

Кроме того, на выбор Dypgen также повлияло и то, что он написан на языке OCaml, поскольку F# считается наследником OCaml и простые программы можно перекомпилировать на F# без изменений. Поэтому было решено попытаться перекомпилировать Dypgen под .NET, внося незначительные исправления в коде, а не переписывать с нуля. К сожалению, колличество требуемых исправлений не оказалось настолько незначительным, чтобы вносить все изменения вручную. Поэтому было решено попытаться реализовать автоматическую трансляцию проекта в F#.

1.Цели работы

В рамках данной работы решалась задача изучения возможности трансляции кода с языка OCaml в F#, а также реализация автоматического транслятора для проекта Dypgen.

2.Реализация

Чтобы не писать транслятор полностью, использовалась программа Camlp4[5] - Pre-Processor-Pretty-Printer для Ocaml. Эта программа содержит расширения языка(revised syntax — альтернативный синтаксис для OCaml, quotation system), а также парсеры кода (преобразуют код в Abstract Syntax Tree (AST)) и принтеры (из AST в код). При запуске программы Camlp4 выбираются объектные (``.cmo'') или библиотечные (``.cma'') файлы — парсер и принтер, которые устанавливают как код будет преобразован в AST и как напечатан из полученного дерева. Но у него есть большой минус - каждый раз Camlp4 обрабатывает только один файл и обработка проходит в один проход. Кроме того программа не предоставляет возможности изменять само AST.

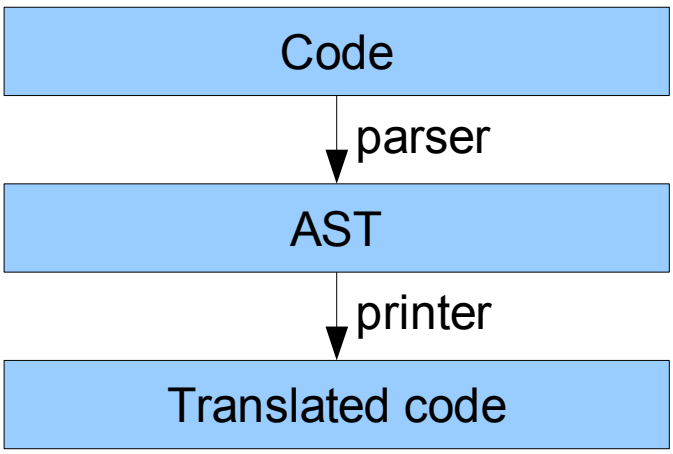

Рисунок 1: Схема работы Camlp4.

Несмотря на это было решено реализовать новый принтер для Camlp4, который будет печатать код для компилятора F#.

Использовался OCaml version 3.11.2 из под среды Cygwin[6], компилятор F# version 1.9.7.8. Работа проводилась над Dypgen version 20100901-1.

Изначально был взят уже готовый принтер для OCaml, который впоследствии модифицировался. Основной принцип его работы - обход в глубину AST. При попадании в вершину разбирается конструкция, находящаяся в ней, и печатается соответствующий код. Принтеры в Camlp4 пишутся с использованием альтернативного(revised) синтаксиса.

Далее рассмотрим возникающие проблемы и методы их решения:

Двумерный синтаксис

Часть работы была посвящена устранению ошибок с сообщением "Incomplete structured construct at or before this point in pattern". Источник этой ошибки - использование в F#, в отличии от OCaml, двумерного синтаксиса. Двумерный синтаксис предназначен для создания более структурированного и читаемого кода. Конструкции одного уровня вложенности должны начинаться с одного столбца(по крайней мере нижняя должна быть не левее), а также текущая конструкция должна быть правее той, в которой она содержится.

Так как изначально использовался принтер для OCaml, в котором нет такого требования, то правила двумерного синтаксиса нарушались. Модуль Format(стандартный модуль языка OCaml) — библиотека обладающая широкими возможностями для форматирования текста с помощью "pretty-printing boxes". Отступы строк устанавливаются в соответствии со структурой box-ов. Было произведено внесение необходимых исправлений в структуру box-ов, которое позволило добиться соответствия требованиям двумерного синтаксиса.

Open vs Include

В языке OCaml есть две команды ”open” и ”include”. Обе позволяют обращаться к элементам подключенного модуля по короткому имени (name вместо module_name.name). Их различие в том, что вторая команда добавляет сигнатуру подключенного модуля в сигнатуру текущего модуля. Например:

```
module A =
struct
  let a = 5;
end;
```

```
module B =
struct
  include A
  let b = 7;
end;
```

```
module C =
struct
  include B
end;
```

Сигнатура модуля C будет:

module C : sig val a : int val b : int end

Тогда можно обращаться внутри модуля C и к функции a, и к функции b

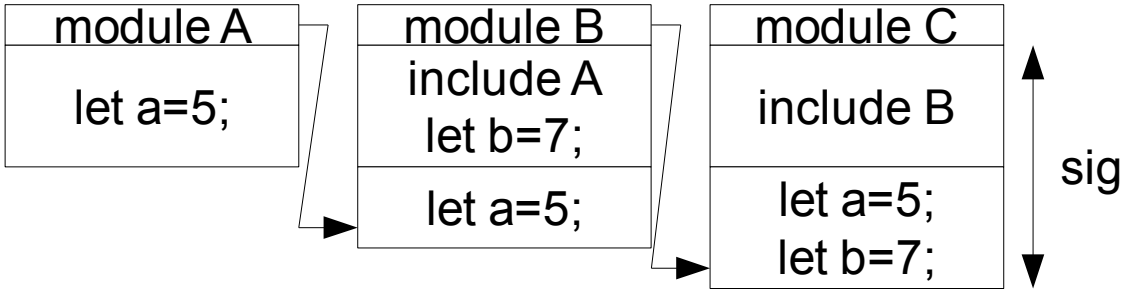

Рисунок 2: Схема взаимодействия модулей при использовании "include". Все функции подключенных модулей добавляются в сигнатуру

А если

module A =

struct

let a = 5;

end;

module B =

struct

open A

let b=7;

end;

module C =

struct

include B

end;

То сигнатура модуля C будет:

module C : sig val b : int end

и обращение из *C* к функции a - выдаст ошибку компиляции.

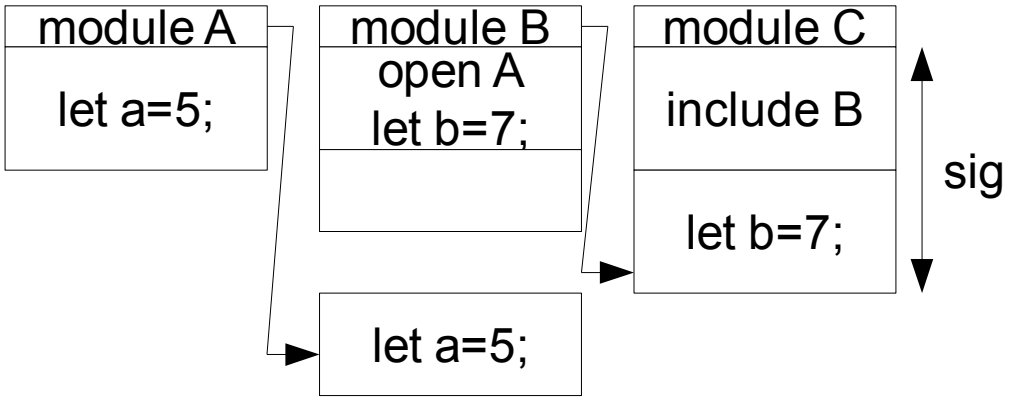

Рисунок 3: Схема взаимодействия модулей при использовании "open". Модуль подключенный с помощью "open" не добавляет свои функции в сигнатуру

Но *include* ещё не реализован в F#. Было решено каждую команду *include* заменять последовательностью команд *open*, так как обращения к функциям подключенных модулей происходят только внутри модуля, который их добавил. Пример выше преобразовался бы в:

```
module A =
struct
  let a = 5;
end;
```

```
module B =
struct
  open A
  let b=7;
end;
```

```
module C =
struct
  open A
  open B
end;
```

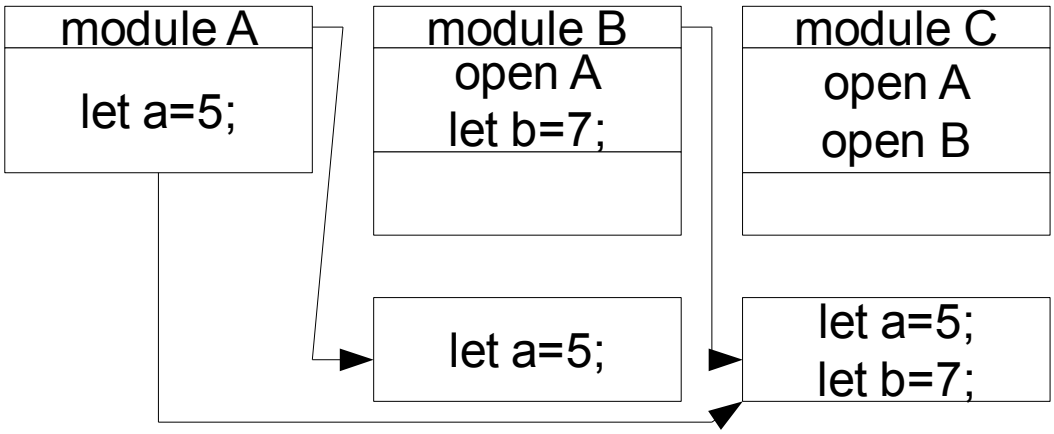

Рисунок 4: Схема, получающаяся при замене "include" на последовательность "open". Ничего не добавляется в сигнатуру, но все функции доступны

Для этого вводится хэш-таблица, которая запоминает всю вложенность модулей. При встрече любой из двух команд транслятор смотрит в таблицу и добавляет всю необходимую цепочку *open*. Кроме того, при встрече *include* добавляются новые элементы в таблицу .

Временные файлы

В предыдущей проблеме впервые сталкиваемся с недостатком Camlp4. Программа каждый раз обрабатывает один файл, в то время как нам требуется запоминать вложенность модулей по всему проекту. В связи с тем, что проект содержит несколько файлов, необходимо передавать информацию, накопленную в хэш-таблице, в следующие. Наиболее простой показалась реализация передачи информации с сохранением в файл/загрузкой из файла. Для этого использовались временные файлы и модуль Marshal(ещё одна из стандартных библиотек OCaml), который позволяет кодировать

структуры данных в последовательность байтов и декодировать обратно. Все накопленные знания о связях модулей записывается и при запуске следующего исходника таблица загружается и её можно использовать и дополнять. Подобное решение будет использоваться и для другой собранной информации, которая будет нужна в дальнейшем.

Аргументы со значением по умолчанию

Одной из возможностей OCaml является возможность использования аргументов функций, у которых задано значение по умолчанию. При последующих вызовах этой функции эти аргументы опускаются, или перед ними надо использовать специальные метки(labels).

К сожалению, это не допускается в F#. Было решено аргументы со значением по умолчанию превратить в обычные, а отдельно запоминать для функций список значений по умолчанию(передача в следующий исходник через временный файл (см. выше)). После чего подставлять значение в вызов функции, если не стоит метка. А если метка есть, то она убирается (labels также не поддерживается в F#).

Функтор Make

Ещё одной возможностью OCaml, не реализованной в F#, являются функторы. Написание трансляции всевозможных функторов - достаточно сложная задача. Но при детальном рассмотрении проекта Dypgen было замечено, что используются только функторы Set.Make и Map.Make. И транслировать при ограничении, что в коде встречаются только такие функторы, возможно.

Результатом работы функторов Set.Make и Map.Make являются модули. В этих модулях содержится набор функций для работы с множеством и словарем соответственно, элементы которых имеют заданный тип(он должен быть строго упорядоченным).

Но зато в .NET платформе есть Set и Map, реализованные как generic классы, и необходимый набор функций для работы с ними реализован для произвольного типа, у которого задано сравнение. Поэтому результат выражения становится просто новым типом вместо модуля.

К сожалению, в текущей реализации не сохраняется определение упорядоченности элементов, содержащихся в Set или Map.

Заключение

В результате работы получены следующие результаты:

- Изучен альтернативный (revised) синтаксис языка OCaml.
- Изучены возможности Camlp4.
- Изучена методика построения трансляторов.
- Изучена возможность трансляции кода с языка OCaml на F#.
- Создан транслятор, преобразующий структуры OCaml в структуры F#.
- Преобразованный код успешно проходит первый этап компиляции (синтаксический анализ).

В дальнейшем необходимо продолжить работу над транслятором по следующим направлением

•Добавление правильного преобразования функторов Make.

•Продолжение работы над транслятором до полной компиляции кода или до момента, когда продолжение работы над транслятором будет сложнее ручного исправления.

•Проверка работоспособности Dypgen под платформой .NET и внедрение в проект YaccConstructor

Список литературы

[1] Recursive-ascent parser generator for F#// google code YaccConstructor URL:http://code.google.com/p/recursive-ascent/

[2] Dypgen// Сайт разработчиков Dypgen URL:http://dypgen.free.fr/

[3] F#// Microsoft F# Developer Center URL:http://msdn.microsoft.com/fsharp/

[4] *Mark G.J. van den Brand, Alex Sellink, Chris Verhoef* Current Parsing Techniques in Software Renovation Considered Harmful.//IWPC ’98: Proceedings of the 6th International Workshop on Program Comprehension. - IEEE Computer Society, Washington,1998.

[5] Camlp4// Camlp4 - Reference Manual URL:http://caml.inria.fr/pub/docs/manual-camlp4/

[6] Cygwin//Сайт разработчиков Cygwin URL:www.cygwin.com/