ADP Compiler 0.8 manual

Peter Steffen[1], Marco Rüther, Christian Lang
and Georg Sauthoff

[1]Faculty of Technology, Bielefeld University, 33594 Bielefeld, Germany,
email: psteffen@techfak.uni-bielefeld.de

February 9, 2007

Contents

1 Introduction

This manual gives a short overview of the ADP compiler. Section 2 gives an exemplified introduction to the ADP compiler's compilation process. Section 3 then gives a complete account of the compiler's command line interface.

This manual does not contain any information about the internals of the ADP compiler. See Peter Steffen's PhD thesis (file `diss.pdf` in the compiler distribution) for some internal details of the ADP compiler.

2 Quickstart

Copy the file `ElMamun.lhs` out of the directory `INSTALLDIR/share/adpc/examples`. Then type

```
adpc ElMamun.lhs
```

This generates a number of files, the most important ones the files `Makefile` and `ElMamun.xml`. The file `Makefile` contains the compile targets necessary to compile `ElMamun.lhs` into the desired C target code files. In the next step, simply type

```
make
```

This generates the main target file `ElMamun.c` and for every algebra in the source file (`seller`, `buyer`, `count`) a target code file `ElMamun_`*algebra*`.c`. These target files are then each compiled by the C compiler `gcc` and linked together to the executable binary file `ElMamun`. See Figure 1 for an overview of the compilation process.

After the compilation finished, you can simply start `ElMamun` by typing

```
./ElMamun
```

This gives the following interactive command line:

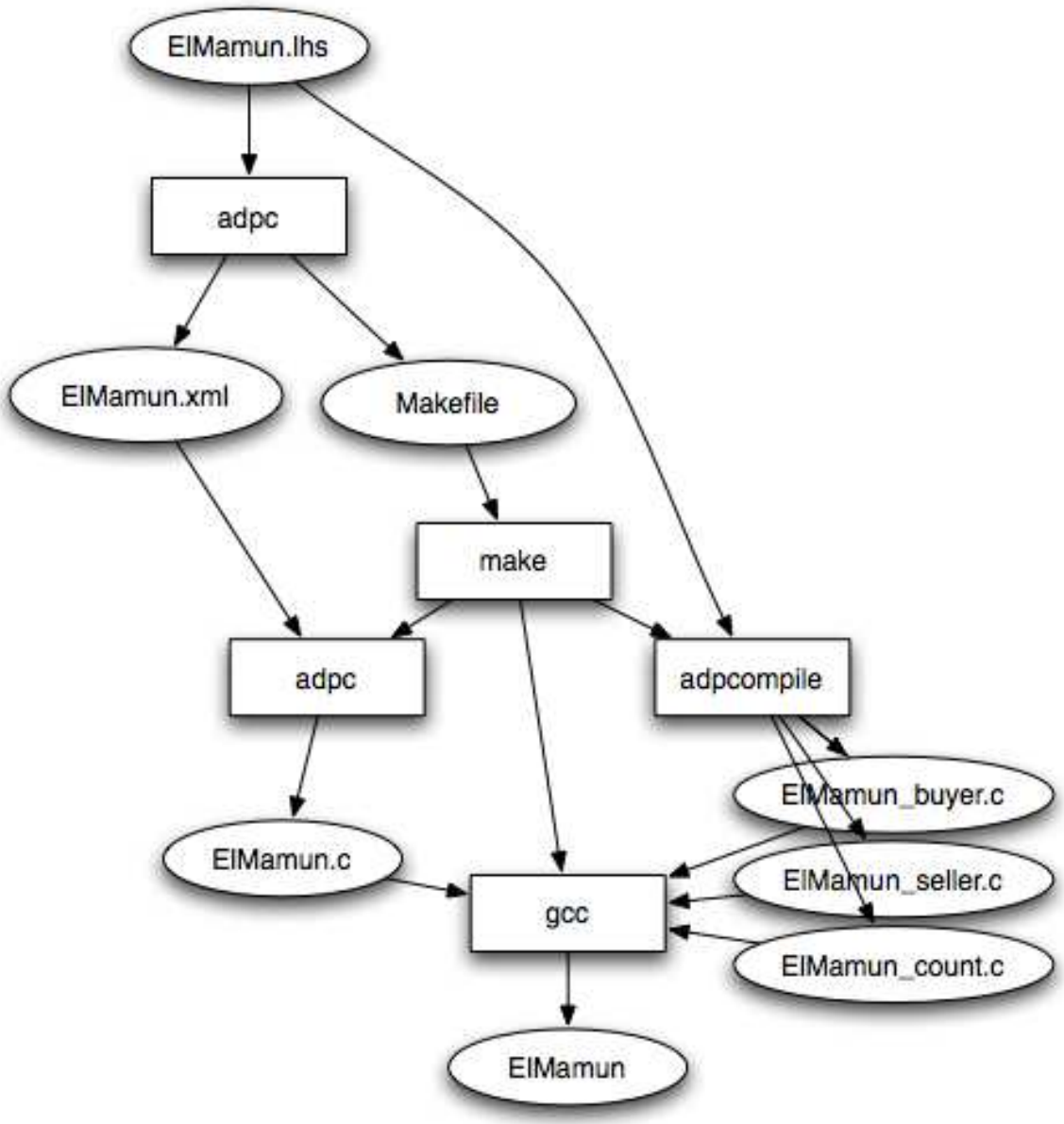

Figure 1: Adpc framework

```
Welcome to ElMamun!

ElMamun>
```

At the command prompt you need to type a formula that shall be evaluated by the ElMamun program. For example, type:

```
ElMamun> 1+2*3*4+5
```

This gives the following result:

```
ElMamun> 1+2*3*4+5

Input: 1+2*3*4+5
Algebra: count, score: 51

======================================================================

Input: 1+2*3*4+5
Algebra: buyer, score: 30
Suboptimal range: [30 - 35]

 Score | Candidate
----------------------------------------------------------------------
    30 | ((1+((2*3)*4))+5)
    30 | ((1+(2*(3*4)))+5)
    33 | (((1+(2*3))*4)+5)
    30 | (1+(((2*3)*4)+5))
    30 | (1+((2*(3*4))+5))
    35 | (1+(2*((3*4)+5)))

======================================================================

Input: 1+2*3*4+5
Algebra: seller, score: 81
Suboptimal range: [76 - 81]

 Score | Candidate
----------------------------------------------------------------------
    81 | (((1+2)*3)*(4+5))
    81 | ((1+2)*(3*(4+5)))

======================================================================
ElMamun>
```

The first result shown is the result for algebra `count`. It states that there are 51 ways to evaluate the formula. The second result is the result for algebra `buyer`. It is a minimizing algebra, and the best score for the formula is 30. The list below shows six results that occur in the suboptimal range of 30 to 35. The last result then shows the results for algebra seller. It is a maximising algebra, and the best score for this input is 81. There exist two candidates that achieve this score.

Repeat this compilation process with the second program in the examples directory, `RNAfold.lhs`.

3 The ADP compiler's command line interface

The ADP compiler consists of two executable programs. The program `adpc` is responsible for creating the main program interface of the target program, the `Makefile`, and several other parts. Apart from the name of the input file, `adpc` has no command line options. The `Makefile` then contains the calls

to the main ADP compiler executable, `adpcompile`. This program does the main work of the ADP compiler and has a lot of command line options. See the `Makefile` for a typical command line call. In a typical application setting, you do not need to call `adpcompile` directly. This is done automatically by the `Makefile`. But there exist cases where it is necessary to call `adpcompile` separately from the `adpc` program flow. In these cases, you need to take care of the `adpcompile` command line options. The following section gives a complete description of the `adpcompile` command line interface.

3.1 OPTIONS

-h Display this information

This option shows the adpcompile command line interface.

-H `option` Display detailed information on <option>

This displays the corresponding section of the adpcompile manual for the given command line option.

-c `file` Compile to imperative target code

Compile the given adp file to C.

-z `file` Optimize combinators in ADP source code

This mode reads the given ADP file, optimizes the next-combinators in it, and generates an optimized copy of the given file.

-l `file` Generate recurrences typeset in LaTeX

This option generates recurrences typeset in LaTeX.

-g `file` Generate ADP template for signature

This option generates a ADP template for the given signature. The template consists of the algebra type, enumeration and count algebras, the definition of the pair-operator and a simple grammar, that generates the term algebra.

3.2 Table design

-tg `file` **-ctg** Derive good table configurations

Option **-tg** generates a `good` table configuration. A configuration is good, if it achieves a polynomial runtime of the given ADP program with the minimal number of DP tables. While option **-tg** stops after the table design, the option **-ctg** can be used together with option **-c** for the imperative target code generation.

-to `file` **-cto** Derive optimal table configurations

Option **-to** generates an optimal table configuration. A configuration is optimal, if it achieves the best possible asymptotic runtime with the minimal number of DP tables. Again, option **-to** starts a stand-alone table design, while **-cto** optimizes the tables during the target code generation.

-tx `file` **-ctx** Derive approx. optimal table configurations

Since the table design is an NP-complete problem, options **-tg** and **-to** are not suitable for larger grammars. Our experience showed that grammars with up to 40 nonterminal symbols can be processed with option **-to**. With more nonterminals, the running time of the optimization can easily go into hours and days. Here the options **-tx** and **-ctx** can be used. These options calculate optimal table configurations with the use of an approximal approach (GRASP). In our experience this approximal results are as good as the results from option **-to**.

-iuc Ignore user annotated table configuration

To support the table design phase, the input program can be annotated with the keywords "tabulated" and "nontabulated". This means, a production annotated with "tabulated" has to be tabulated in every case, and a production annotated with "nontabulated" must not be tabulated. This user annotations work together with large grammars and options **-tg** and **-to**, which would otherwise calculate too long. With option **-iuc** the ADP compiler ignores all user annotations.

-tadd `number` Maximal number of additional tables

Option **-tadd** can be used together with options **-to** and **-tx** to improve the constant factor of an optimal table configuration. This is done by introducing a given number of additional tables. If this improves

constant factors by the factor given with option **-taddc** the additional tables are added to the table configuration.

-taddc `number` Necessary constant factor improvement for add. tables

Option **-taddc** specifies the necessary constant factor improvement for option **-tadd**

-taddn `number` Expected input length for constant factor improvement

Option **-taddn** specifies the expected input length for the constant factor improvement.

3.3 Target code generation

-bt `mode` Generate backtracing code (enumeration algebra needed) mode: s -> single, so -> suboptimal

With option **-bt** the ADP compiler generates backtracing code. The given mode can be either `s` for a single-result-backtrace, or `so` for a suboptimal backtrace.

-W Include window mode

Option **-W** generates program code for a sliding window mode. Beginning with position 1 of the input sequence, the analysis is repeatedly processed on subsequences of the specified size. After each calculation, the results are printed out and the window is moved by the window position increment (**-W**), until the end of the input sequence is reached.

-al `alg` .. `alg` Specify order of algebra usage

Option **-al** specifies the algebras for the compilation.

-alpp `alg` .. `alg` Specify pretty printing algebras

Option **-alpp** specifies the pretty printing algebra that shall be used for the candidate ouput in the backtrace modes.

-cs `alg` Automatically generate signature and enumeration for algebra <alg>

The option **-cs** automatically generates the signature and enumeration algebra from the given algebra. This option is useful for the backtracing modes.

-vl `verbosity level` Specify output verbosity level verbosity level: t -> target, r -> trace, rr -> detailed trace d -> debug

Option **-vl** specifies the verbosity level of the compilation.

-o `filename` Output to <filename>

Option **-o** specifes the output file.

-v Show version

This option shows the adpcompile version number.