Tool Demo: BNF Converter

Markus Forsberg and Aarne Ranta
markus@cs.chalmers.se

Haskell Workshop 2004

Chalmers University of Technology, Sweden

Developing a compiler front-end

- We start with an **Idea** about the language:
  - Language Specification
  - In the mind of the implementor
- We continue by developing a set of **modules**, usually with the help of existing tools.

Lexer

Parser

Abstract Syntax

Documentation

Case analysis (e.g. type checking)

Pretty Printer

Problem: Consistency

- Hard to keep all modules **consistent!**
- Say that we want to extend our language with a new language construct. Then we have to change every module!

Lexer

Parser

Abstract Syntax

Documentation

Case analysis (e.g. type checking)

Pretty Printer

Problem: boring code

- We have to write a lot of **boring** code.

Example: Happy parser generator code

```
...
Stm :: { Stm }
Stm : Labeled_stm { LabelS $1 }
  | Compound_stm { CompS $1 }
  | Expression_stm { ExprS $1 }
  | Selection_stm    { SelS $1 }
  | Iter_stm            { IterS $1 }
  | Jump_stm          { JumpS $1 }
...
```

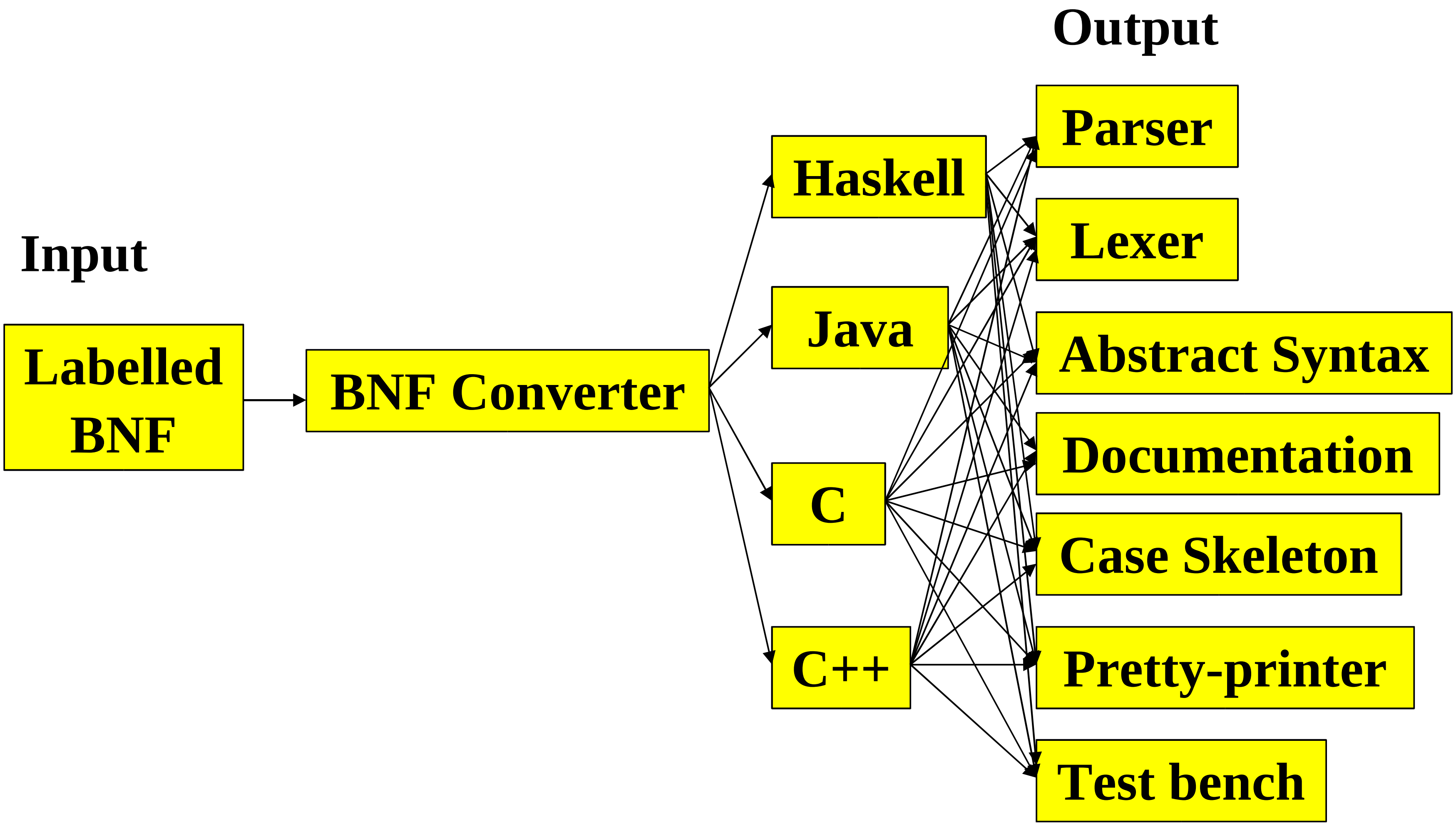
Problem: Language-specific result

- We end up with a compiler front end in a **specific programming language**.
- But, we (may) want to **design** in a **declarative language**
- and as the **final product** use an **imperative language** (e.g. a compiler in C).
- or incorporate our language in a system written in another language
- Then we have to **rewrite everything!** Irgghh...

Solution

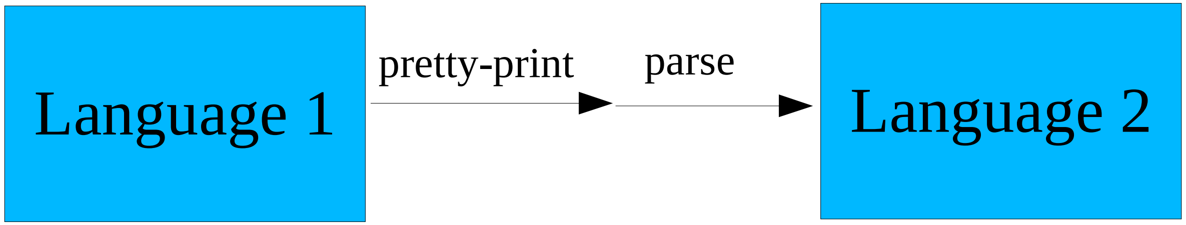
- Use a **single source** to generate all modules.
- Use a **simple formalism** for the single source.
- Use a **declarative style** for the single source. *Describe instead of implement the language*.

System overview: BNF Converter



Data exchange format

Another use of BNF Converter is as a **data exchange format**



The multi-linguality of BNF Converter provides a convenient way of communicating data between different programming languages

BNFC requirement

1. The language’s lexical structure must be describable by a **regular expression**.

2. The language must not only be context-free, but **LALR(1)** parsable (actually, this is requirements from the used tools).

3. The language implementation can be **separated** into a lexer, a parser and whatever more that lurks in the back-end.

Most modern-day programming languages have (at least) a well-defined subset that fullfills these requirements.

Grammar projects

Existing languages developed in BNF Converter:

- C
- Java
- OCL
- Alfa
- External Core in GHC
- ASN.1

New languages developed with BNF Converter:

- Grammatical Framework
- BNF Converter's own source format

BNFC availability

GPL License

Available at BNFC Homepage:
http://www.cs.chalmers.se/~markus/BNFC

Also available as a **Debian Linux package**, in the **testing** distribution.

The People behind BNF Converter (in alphabetical order)

Björn Bringert
Markus Forsberg
Peter Gammie
Patrik Jansson
Antti-Juhani Kaijanaho
Michael Pellauer
Aarne Ranta

Demo: External Core

- A grammar written by Aarne Ranta – approximately 2.5 h work including debugging (GHC 5.02.2).

- Extracted from the abstract syntax and the Happy parser from the GHC source code.

- WC count (source format)

```
  92    474    26792Core.cf
```

instead of

```
  89     243    1324   ExternalCore.lhs
 240    1042    5168  ParserExternalCore.y
 168     906    4667  PprExternalCore.lhs
 497    2191  11159 total
```

where the lexer source and the language document are still missing.