ASCII-Art Description Language

Jen-Chieh Huang, Yuan Lin, Jie-Gang Kuang,
Xiuhan Hu, Zixuan Gong
{jh3478, yl2324, xh2234, jk3735, zg2203} @columbia.edu

Why dont we start ?

What's ASCII-art ?

[Wiki]
ASCII-art is a graphic design technique that uses computers for presentation and consists of pictures pieced together from the ASCII characters.

Outline

- Hello, Adele
- The Adele Programming Language
- Adele Compiler: an Anatomy
- The Runtime Environment
- Quality Assurance
- Project & Process Management

Hello, Adele

- ASCII-art has been an interesting element in the online community for a long time.
  - Simple facial expression (^__^)
  - Complex & interactive graphical representation
- Handmade ASCII-art is exhausting.
  - Hours of work to adjust the positions of the components.
- Adele is simple and intuitive for creating ASCII artwork.
  - Easy to write, intuitive to use, portable outcome
  - web-ready target code
  - interactive functionalities

Quick Facts about Adele

- Adele is
  - a general purpose language focusing on ASCII-art processing.
    - an **imperative** language, starting the program from the main function.
    - an **object-friendly** language. User-defined types are supported.
    - a **Turing-complete** language
  - portable and web-ready
    - generating **web-ready** executables
    - It’s also **portable**. The target code is **JavaScript**.
  - written in Java using ANTLR4 & StringTemplate4

Hello, World !

The Adele Programming Language I

- Program:
    - Function definition
    - group declaration
    - array/var declaration

- Function
    - return type
    - parameter list
    - body (statements)

```
int test(int a, int b)
   return a + b;
end
```

The Adele Programming Language II

- “`if`” statement

```
if (a > b)
   a = b;
end
```

- “`while`” statement

```
while (a > b)
   a = a + b;
end
```

The Adele Programming Language III

- array

```
int a[2][2];
```

- group (structure)
  - declare
  - instantiate

```
group A
   int a;
end
...
group A instance;
```

The Adele Programming Language IV

- @ operator

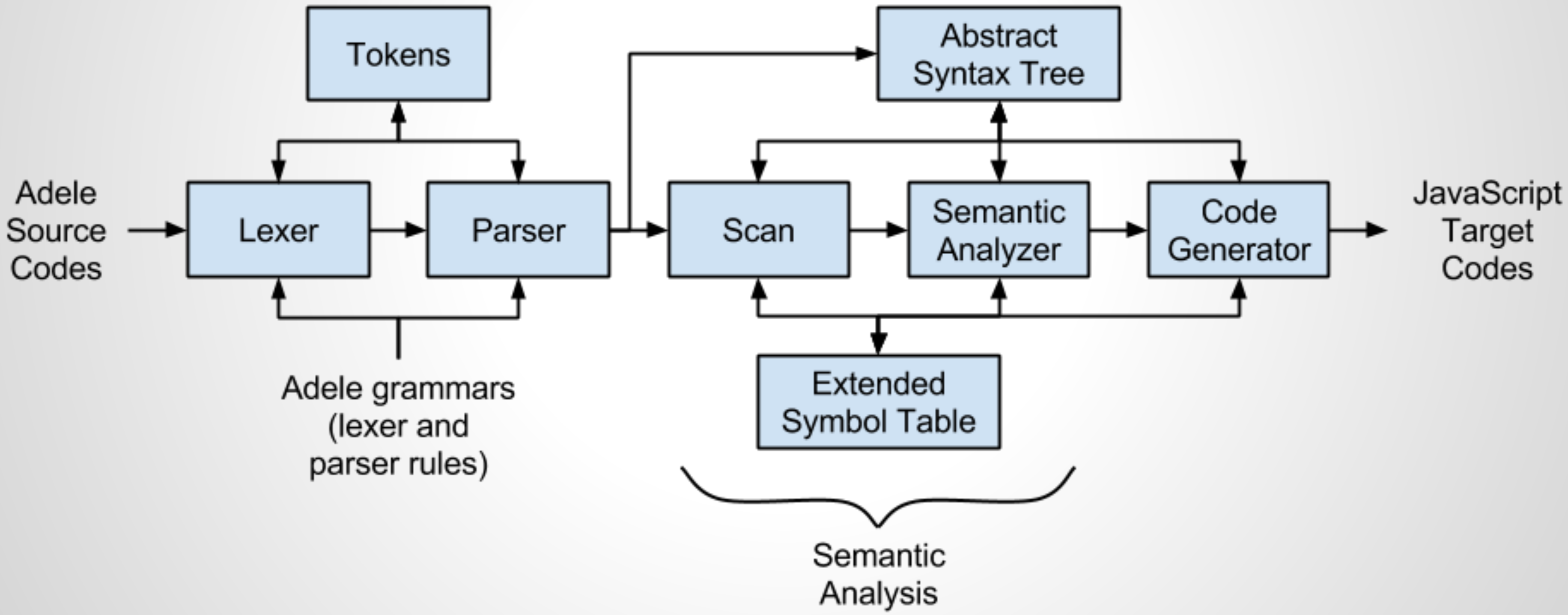
```
graph a = str2graph(“hello adele”);
a @ (1, 1)
```

- // operator

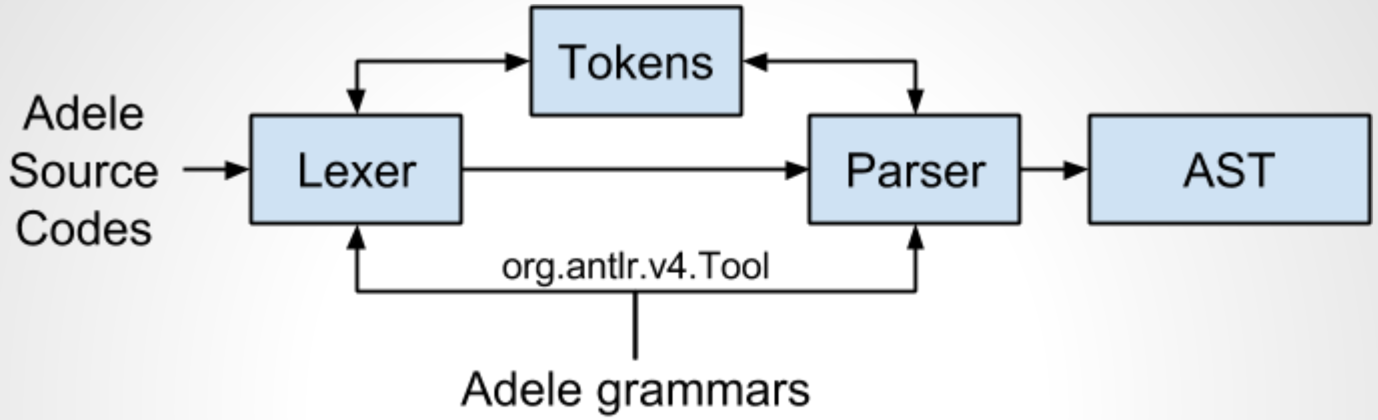
```
graph a = str2graph(“hello adele”);
graph b = str2graph(“!”);
b // a @ (0, 11);   # -> “hello adele!”
```

More Than Fun!

Architecture of Adele Compiler

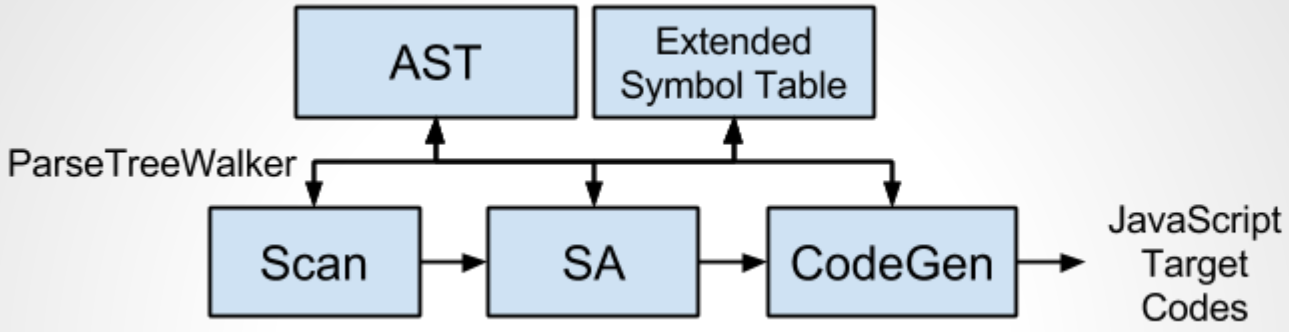


Lexer & Parser



- Grammars in ANTLR4
    - adelelex.g4
    - adele.g4
- Generated by ANTLR4 tool
    - adeleLexer
    - adeleParser
- Integrated flow to generate AST

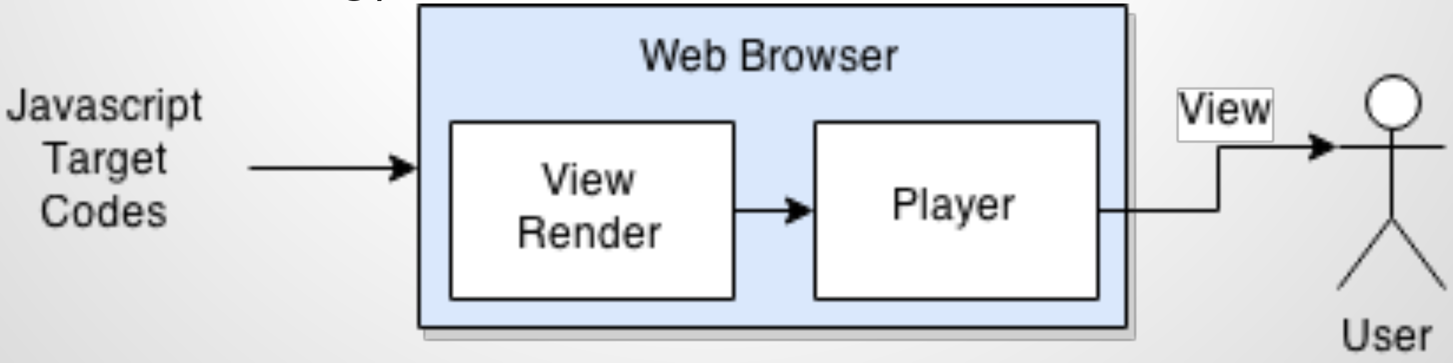
Semantics Analysis & Code Generation



- AST traversal is easy with ANTLR4
  - ParseTreeWalker
- Semantics analysis - 2 passes
  - ScanPhase
  - DefPhase
  - Self-defined extended symbol table
- Code generation - 1 pass
  - TransPhase

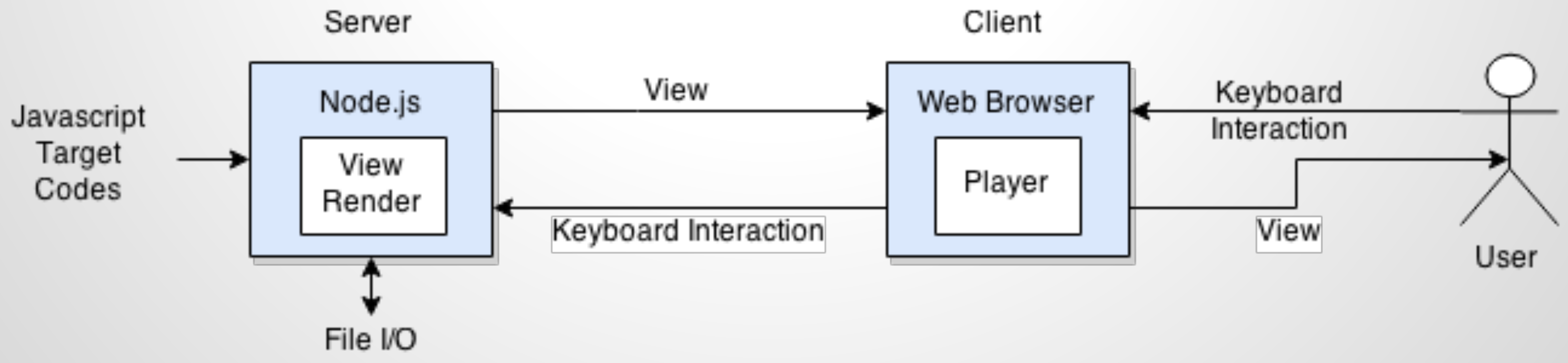
Run-time Environment (1)

- Version 1
    - JavaScript can run on any modern browser
    - So browser act as our target program interpreter:
        - It renders the drawings of ASCII-art first
        - Then plays the art according to the timeline (**`sleep`** function marks intervals between drawing)



Run-time Environment (2)

- Version 2：server-client structure
  - A `Node.js` server program is generated
  - The server communicates with the web browser client via websocket
  - The server draws ASCII-art according to target program, user input and file I/O in realtime.



Development Environment

- Adele is developed under a Unix-based environment, specifically Ubuntu and Mac OS X
- We mainly wrote codes in Java with ANTLR and StringTemplate as toolkits.
  - ANTLR 4
  - StringTemplate 4
- We use **`make (Makefile)`** and **`shell`** scripts to create the pipeline for creating compiler, compiling source code of Adele and testing.

Game Time!

Quality Assurance & Automated Testing (1)

- Static Tests
  - Does the compiler give correct syntactic and semantic error messages?
- Runtime Tests
  - Is the target program equivalent to the source program?

Quality Assurance & Automated Testing (2)

- Static Tests

```
void main()
    int a = 1;
    b = a;          # err
    a = “string”; # err
end
```

Quality Assurance & Automated Testing (2)

- Static Tests

```
void main()
    int a = 1;
    b = a;           # err
    a = “string”; # err
end
------------------------
pass static test
```

Quality Assurance & Automated Testing (2)

- Static Tests

```
void main()
    int a = 1;    # err
    b = a;
    a = “string”; # err
end
```

Quality Assurance & Automated Testing (2)

- Static Tests

```
1 void main()
2     int a = 1;    # err
3     b = a;
4     a = “string”; # err
5 end
```

```
-------------------------
error detected: 3 4, expected: 2 4
-------------------------
[ERROR] line 3: No such variable: b
[ERROR] line 4: Incompatible types: int:string
```

Quality Assurance & Automated Testing (3)

- Runtime Tests

```
int add(int a, int b)
      return a+b;
end
-----------------------
testIntMath … {
    …
    test.ok(target.add(1,2) == 3, "math: int add");
    …
}
-----------------------
✔ testIntMath
```

Quality Assurance & Automated Testing (4)

- Test Plan
    - Tutorial, LRM, grammar rules, features…
    - Aspects covered:
        - array
        - special constructs (e.g. overlay, attach)
        - declaration
        - expression (e.g. assign, function call)
        - group
        - syntax
        - function (e.g. scope, parameter)
        - arithmetic operation
        - flow control

Build Process & Integrated Auto Testing & Style Checking

Project Management (1)

- A hybrid process (waterfall X agile)
  - Predefined goals
  - Weekly short meeting/Quick response development
  - Prototype first. Running changes welcomed.
- Milestones
  - Phase Zero
    - Project definition
  - Hello world
    - Basic grammar ready/codegen
    - Simple runtime environment
  - Quicksort
    - Grammar refined/major codegen/basic testing
    - Simple UI
  - Pacman
    - Grammar/codegen/autotesting done
    - User interaction.

Project Management (2)

- Dynamic team organization
    - Task force-based
        - We constantly learn thing in different domain.
    - Separate testing members and developer members.
        - You don’t test the code you write after commit.
    - Quick response
        - Instant messages
        - Handler first
- { Single expert, all developers } model
    - Experts focusing on researches of the topic, and teach the others.
    - Everyone is developer.

Process Management (1)

- Software version control
  - Hosted on Github
  - For major changes, a development branch will be used, and merged back to master later.
- Software auto testing
  - Integrated in the build process
    - All commits have to be compilable and pass all test cases
  - Developer has to write his own test cases

Process Management (2)

- Software coding style auto verification
  - Integrate “***CheckStyle***” tool to report style inconsistency.
  - The style is derived from Google Java style except
    - Indent level changed to 4
    - Pass JavaDoc check

Questions ?

Runtime Error Handling (*)

- Collect the source code information and use when runtime error happens.
  - Collect function definition and the source line number in DefPhase
  - Embedded into the target code as an (partial) symbol table.
    - hash table
      - function name as key
      - line number as value
- When runtime error happens,
  - Catch all exceptions in the main function.
  - Parse the exception stack using the information collected in the earlier phase. (*)