Programming Language Independent
Abstract Syntax Trees

Nordic Workshop on Programming Theory 2003

Karl Trygve Kalleberg
<karltk@ii.uib.no>

Magne Haveraaen
<magne@ii.uib.no>

Department of Informatics
University of Bergen, Norway
Overview

• Motivation
  – Better understand common/emerging
    • Programming concepts
    • Language properties
  – Language tool interoperability
    • Transformation
    • Refactoring

• Approach
  – Analysis + synthesis ➞ formalism
A = 42 + 23;

```
A = 42 + 23;
```
Programming languages vs programming concepts

(\lambda x. \text{Display } x) \text{ “Hello world”}
Language evolution

• New concepts expressed through
  – Idioms
    • language-specific recipes
  – Patterns
    • recipes common to a class of languages

• New concepts require
  – Adaptation to existing semantics
  – Possibly new syntax
Evolution: if+goto

label:
... code ...
if(cond) goto label;

while(cond) {
... code ...
}

if(!cond) goto label1;
... code ...
goto label2
label1:
... code ...
label2:

if(cond) {
... code ...
} else {
... code ...
}
Evolution: assertions

- Assertions in Java 1.4

```java
if (!cond)
    throw new AssertionError(stringExpr);

assert cond : stringExpr;
```
Evolution: contracts

- Pre/post conditions and class-invariants in Eiffel

```eiffel
func(arg: TYPE) is
    require
        boolean-expression
    ...
    do
        body
    ...
    ensure
        boolean-expression
    ...
end
```
Use case: CodeBoost

Refine to obtain accurate information

Perform transformation

Emit code

Source code

AST

CST

AST

CST

Source code
Accurate information

- `for(exp1; exp2; exp3) body;`
  - Undecideable termination
  - Sugar for `exp1; while(exp2){body; exp3;}`
- `for x in [1,2,3]: body`
  - Will always terminate
- `map(\lambda x. body, [1,2,3])`
Structuring and reuse

- Many concepts
  - Classes, inheritance, aspects, coordination, patterns, best practices
- Fewer constructs
  - package, class, template
- Experimentation easier at the abstract level
  - Not compounded by syntax
  - Allows domain-specific structuring
Conclusion

• Build more abstract AST using accurate information
  – Support language evolution in tandem with concept evolution
  – Increase convenience for tools
  – Simplify reasoning for programmer

• Not a new language!
  – Represented in contemporary languages