

Compressed Abstract Syntax Trees as Mobile Code

Christian H. Stork

Vivek Haldar

University of California, Irvine

Outline of the talk

- # Part I: Making the case for using abstract syntax trees as a mobile code distribution format
- # Part II: Toplevel overview of how we compress abstract syntax trees
- # Part III: Future work, work in progress, wild proposals

Part I: Abstract Syntax Trees as Mobile Code

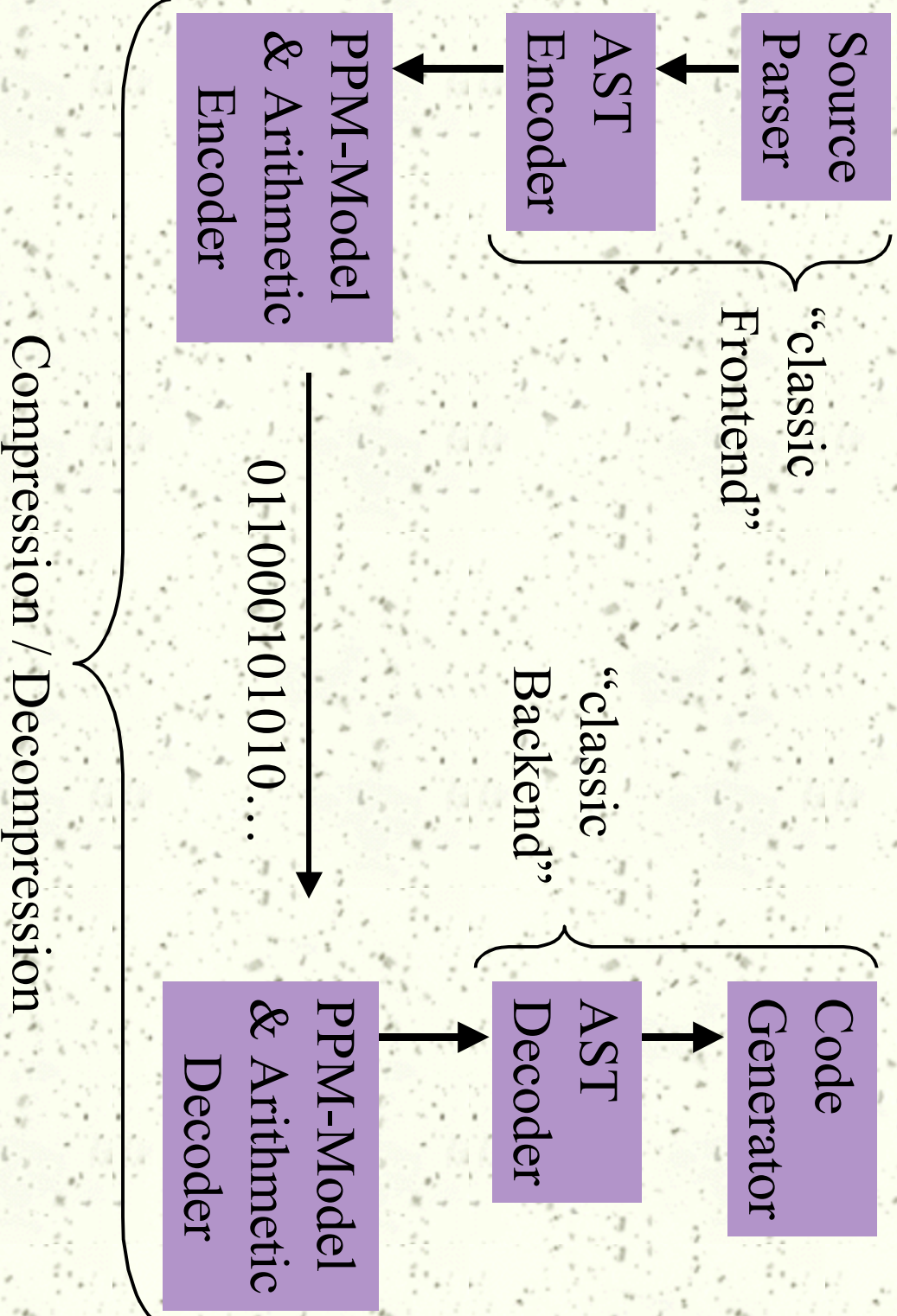
Mobile Code

Mobile code is the platform

Desirable properties:

- ▣ Platform independent
- ▣ Safe to execute
- ▣ Compact
- ▣ Short start up time
- ▣ Tamper proof encoding
- ▣ Amenable for optimization

Overview



Compression / Decompression

Mobile Code Infrastructure

- # Code producers and consumers
- # Producers distribute mobile code as compressed abstract syntax trees
- # Consumers decompress and compile them for specific target
- # Compression allowed to be computation intensive - one time effort at producer's end

Why Abstract Syntax Trees?

- # High level portable representation
- # Compresses well
- # Easy to optimize for specific targets
- # Easy to reason about language level properties (type safety etc.)
- # Protection from low-level (bytecode) attacks

Safety of abstract syntax trees

- # **Goal: safety by construction**
- # **Guarantees provided:**
 - ▣ Adherence to abstract grammar of language
 - ▣ Lexical scoping enforced as part of encoding

Genericity

- # AST framework parameterized by abstract grammar of language
- # Works with new language simply by specifying its grammar
 - # Currently: Java and Oberon

Part II: Compressing Abstract Syntax Trees

Why compress?

Compactness is increasingly an issue:

- # Limited bandwidth networks (wireless)
- # Storage requirements (e.g. embedded systems)
- # Processors much faster than disks/network - so compression gives net gain in performance

Compression Overview

- # *Parsing*: get AST from source
- # *Serialize*: get stream of symbols from AST
- # *Modeling*: use context and *abstract grammar* to build predictive statistical model
- # *Coding*: use arithmetic coding with model

Status and Results

- # Compressor/decompressor Prototype in Python
- # Completely generic - can be used with any abstract grammar
- # Have implemented the Java abstract grammar
 - Works with single Java source files as well as entire packages.
- # Main comparison with Pugh's results for Java bytecode compression

Results : Classes

Name	Class file	Gzip	Bzip2	Pugh	CAST	CAST/ Pugh
ErrorMessage	305	256	270	209	105	50%
CompilerMember	1192	637	641	396	230	58%
BatchParser	4939	2037	2130	1226	1069	87%
Main	11363	5482	5607	3452	3295	95%
SourceMember	13809	5805	5705	3601	2988	83%
SourceClass	32157	13663	13157	8863	7849	89%

Classes from Sun's javac package. All sizes in bytes.
Compared with Pugh's Java bytecode compressor

Results: Archives

Package	Jar file	Gzip	Bzip2	Pugh	CAST	CAST/ Pugh
Javac	36787	32615	30403	18021	14070	78%
Jess	232041	133146	97852	48331	31083	64%

Compressed collections of classes. All sizes in bytes.

- **Compressed ASTs are 5-50% smaller than Pugh.**
- **3-8 times smaller than uncompressed class and JAR files**

Part III: Future Work

Future Work

- # **Compression**
 - ▣ Explore other PPM models for getting probabilities
- # **Well formedness - safety by construction**
 - ▣ Statically encode semantic constraints
- # **Annotations with the AST**
 - ▣ hard to compute, but easy to verify
 - ▣ Aid optimization at target site

Well-formedness

- # *Statically enforce semantic properties as part of encoding*
 - ▣ Illegal programs cannot be expressed
- # Goal: reduce, and ultimately eliminate costly verification phase
- # E.g. for now, we do lexical scoping
 - ▣ In the future: type safety...
- # Well formedness better compression

Transporting Annotations

- # Shift computing burden from consumer to producer
- # Transport hard to compute, easy to verify annotations
 - ▣ Results of escape analysis
 - ▣ PCC proofs...

Conclusion

- # Abstract syntax trees viable as a mobile code format
- # Can be highly compressed
 - Java archives by factor of 3-8
 - 5-50% better than Java bytecode specific compression by Pugh
- # Working on well-formedness and transporting annotations