LR(1) Parser Generator Hyacc

Introduction

Xin Chen, Ph.D.

University of Hawai'i

Tutorial at the International Symposium on Code Generation and Optimization (CGO 2012), San Jose, CA

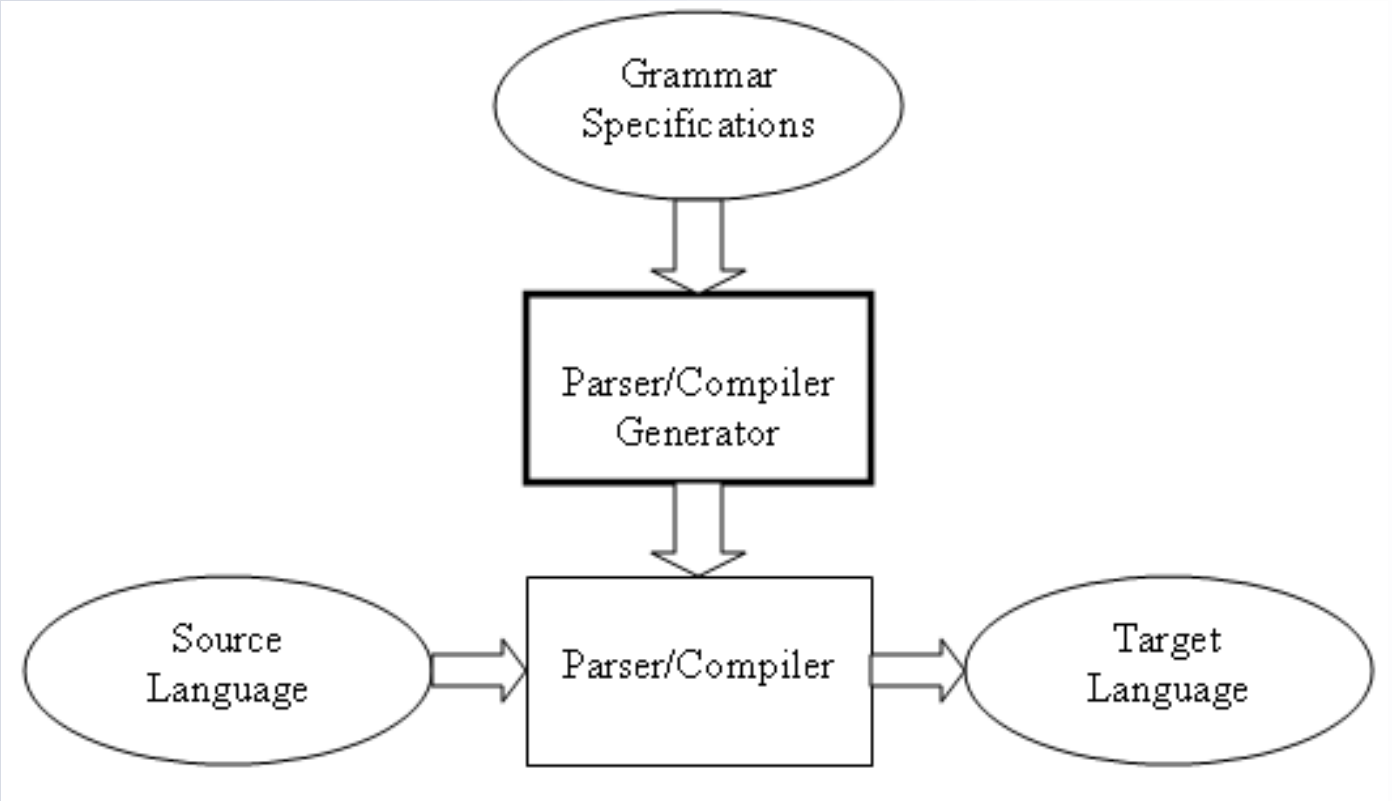
March 31, 2012

Outline

- Introduction
- Problem and Motivation
- Literature Review
- Approach and Results
- Related Work
- Contributions and Limitations
- Summary

I. Introduction

- Parser generator: A software that generates a parser automatically



I. Introduction

- What is a LR(1) parser generator
  - LR algorithm
    - Left to right scanning
    - Right-most derivation
    - LR(1): use 1 lookahead symbol
    - E.g., Hyacc (2008), Menhir (2004), …
  - LALR Algorithm
    - E.g., Yacc (1975), Bison (1989), Byacc (1990) …
  - SLR
  - LL
    - E.g., ANTLR (1993)
  - GLR
    - Handles ambiguity, e.g., for natural language grammars

II. Problem & Motivation

- Problems of compiler/parser generation algorithms
  - Non-LR(1) parser generation algorithms – frequently used
    - LL, SLR, LALR, GLR …
  - LR(1) parser generation algorithm – NOT frequently used
    - Advantages:
      - More recognition power for CFG:
        Superset of LL, SLR, LALR, and equivalent to LR(k):
        LR(1) $\supset$ LALR(1) $\supset$ SLR, LR(1) = LR(k) $\supset$ LL(k)
      - No "mysterious reduce/reduce conflict" of LALR
      - No left recursion problem of LL
      - Same parsing machine size for SLR and LALR grammars
    - Disadvantages:
      - Expensive in performance?
      - Generated parsing machine is too big?

II. Problem & Motivation

- Motivation
  - Reduced space LR(1) algorithms exist
    - Better performance
    - Not systematically studied
    - Implementation is rare
  - We want to unravel the potential power of LR(1) parsing

III. Literature review: Parsing Theory & Practice

- (1952) left-to-right sequential parsing
- (1956) Fortran compiler
- (1957) Chomsky CFG
- (1963) BNF and Algol60
- (1965) Knuth canonical LR algorithm
- (1968) LL algorithm
- (1969) SLR, LALR
- (1975) Yacc
- (1977) Pager LR algorithms
- (1981) Spector LR algorithm
- (1985) Tomita GLR algorithm
- (1989) Bison
- (1992) GLR implementation
- (1993) ANTLR
- (2002) Tribble MLR algorithm
- Recent research:
  - Non-LR(1) grammars
  - Models: NN, ML, …
  - Parallel computation
  - Quantum, DNA computation

LR(1) Parser Generation theory

- Original size
  - The canonical LR(1) algorithm (Knuth 1965)
- May or May not be reduced-space
  - The partitioning algorithm (Korenjak 1969)
- Reduced-space
  - The practical general method (Pager 1977)
  - The lane-tracing algorithm (Pager 1977)
  - The splitting algorithm (Spector 1981,1988)

Excerpts from the comp.compiler newsgroup

- People keep looking for LR(1) parser generators
  - After my previous posting (of LR(1) parser generator GDT_PC) I received lots of requests for paper about GDT_PC … (1992)
  - I am looking for the source for an LR(1) compiler, preferably written in C, and with open source licensing. Google cannot help … (2001)
  - If anyone of you have any idea of a LR(1) parser generator preferably those which have input grammar format as in yacc??? (2003)
  - Do you know of any good C/C++ LR(1) parser generator out there? I've scanned the net but couldn't find any… (2003)

Summary of literature review

- LR(1) has advantage over Non-LR(1) algorithms
- LR(1) performance is a problem
- Reduced space LR(1) algorithms exist, however
  - No systematic study
  - Implementations are rare
- There is a need for LR(1) parser generation

IV. Approach & Results

- 1) Implement parser generator Hyacc
- 2) Study details of LR(1) algorithms
- 3) Study Performance
- 4) Extend to partial LR(k)

1) Parser Generator Hyacc

- Features
  - LR(0)/LALR(1)/LR(1)/partial LR(k) parser generator
  - Compatible with Yacc and Bison
  - Works together with Lex
  - ANSI C compliant: good portability
  - Over 17,000 LOC
  - Many more …

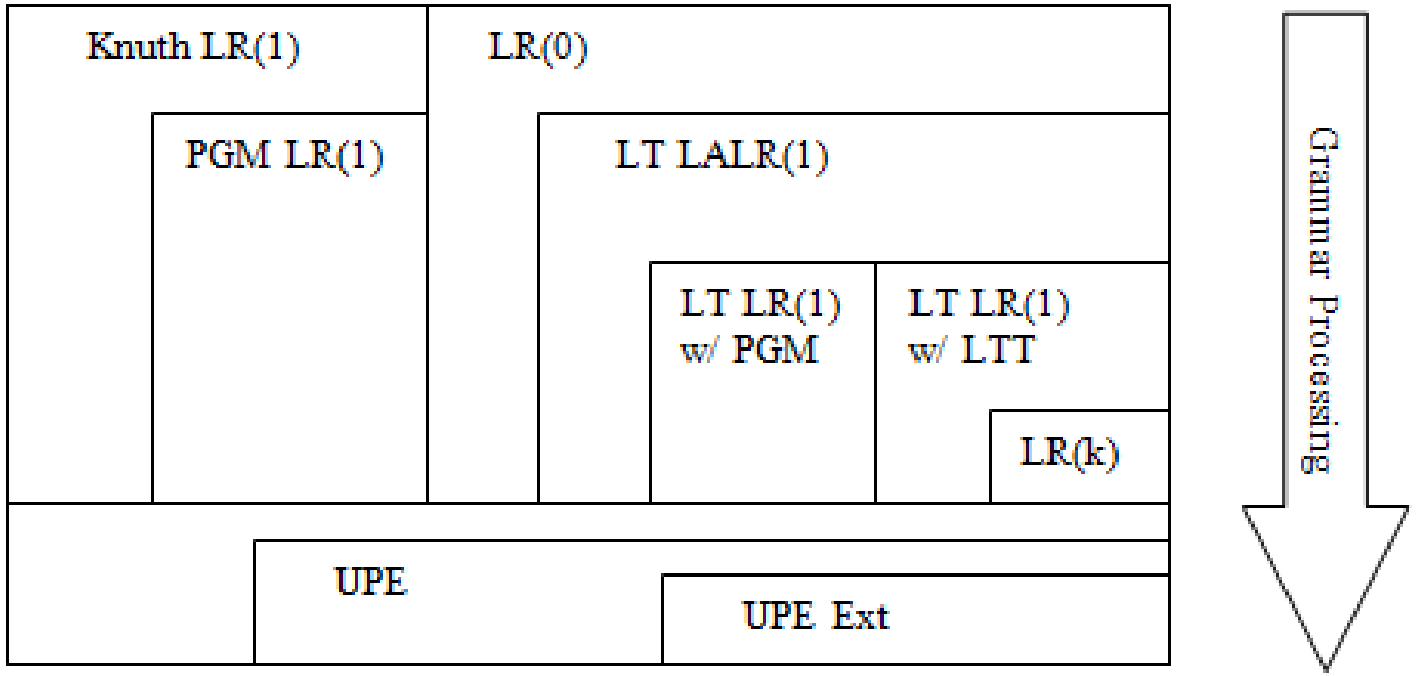
Hyacc is open sourced

- Released to sourceforge.net on GPL license
  - Version 0.9: January 2008
  - Version 0.95: April 2009
  - Version 0.97: January 2011

Algorithms in Hyacc

- LR(0)
- LALR(1)
  - Based on Lane-tracing Phase 1 (LT LALR(1))
- LR(1)
  - Knuth canonical algorithm (Knuth LR(1))
  - Practical general method (PGM LR(1))
  - Unit production elimination algorithm (UPE)
  - Extension to the UPE algorithm (UPE Ext)
  - Lane-tracing algorithm (LT LR(1))
- LR(k)
  - Edge-pushing algorithm (EP)

Relationship of algorithms in Hyacc – look from input grammar



PGM: Practical General Method; LT: Lane-Tracing; LTT: Lane-Tracing Table;
UPE: Unit Production Elimination; UPE Ext: UPE Extension

Performance

- Performance of Hyacc v.s. Menhir and MSTA
  - Implementation language: Hyacc (C), MSTA (C++), Menhir (Caml)
  - Same testing environment
  - Performance of these three are on the same level

Testing Grammar	C++ grammar		C grammar	
Algorithm	Knuth LR(1)	PGM LR(1)	Knuth LR(1)	PGM LR(1)
Menhir (2004)	1.971 s	1.484 s	1.640 s	0.557 s
MSTA (1999)	5.319 s	-	0.918 s	-
Hyacc (2008)	3.529 s	1.779 s	1.047 s	0.420 s

2) Measuring and comparing parser generation algorithms

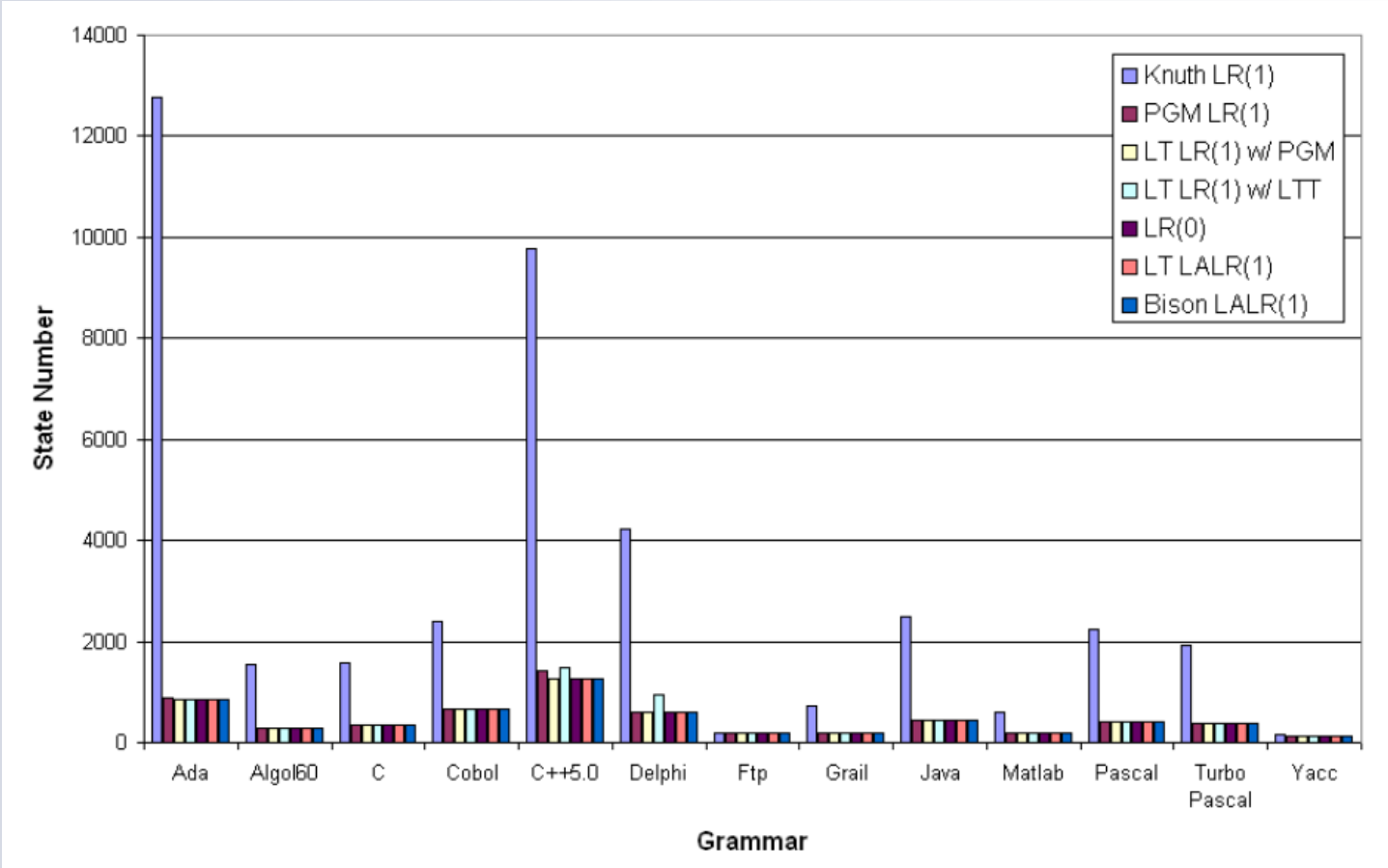
- Test cases
  - 17 simple grammars
  - 13 real language grammars: Ada, Algol60, C, Cobol, C++ 5.0, Delphi, Ftp, Grail, Java 1.1, Matlab, Pascal, Turbo pascal, yacc

2) Measuring and comparing parser generation algorithms

- Test cases
  - 17 simple grammars
  - 13 real language grammars: Ada, Algol60, C, Cobol, C++ 5.0, Delphi, Ftp, Grail, Java 1.1, Matlab, Pascal, Turbo pascal, yacc
- Algorithms to compare:
  - 4 LR(1) algorithms (Knuth, PGM, LT w/ PGM, LT w/ LTT)
  - 2 LALR(1) algorithms (LT LALR(1), Bison LALR(1))
  - LR(0) algorithms

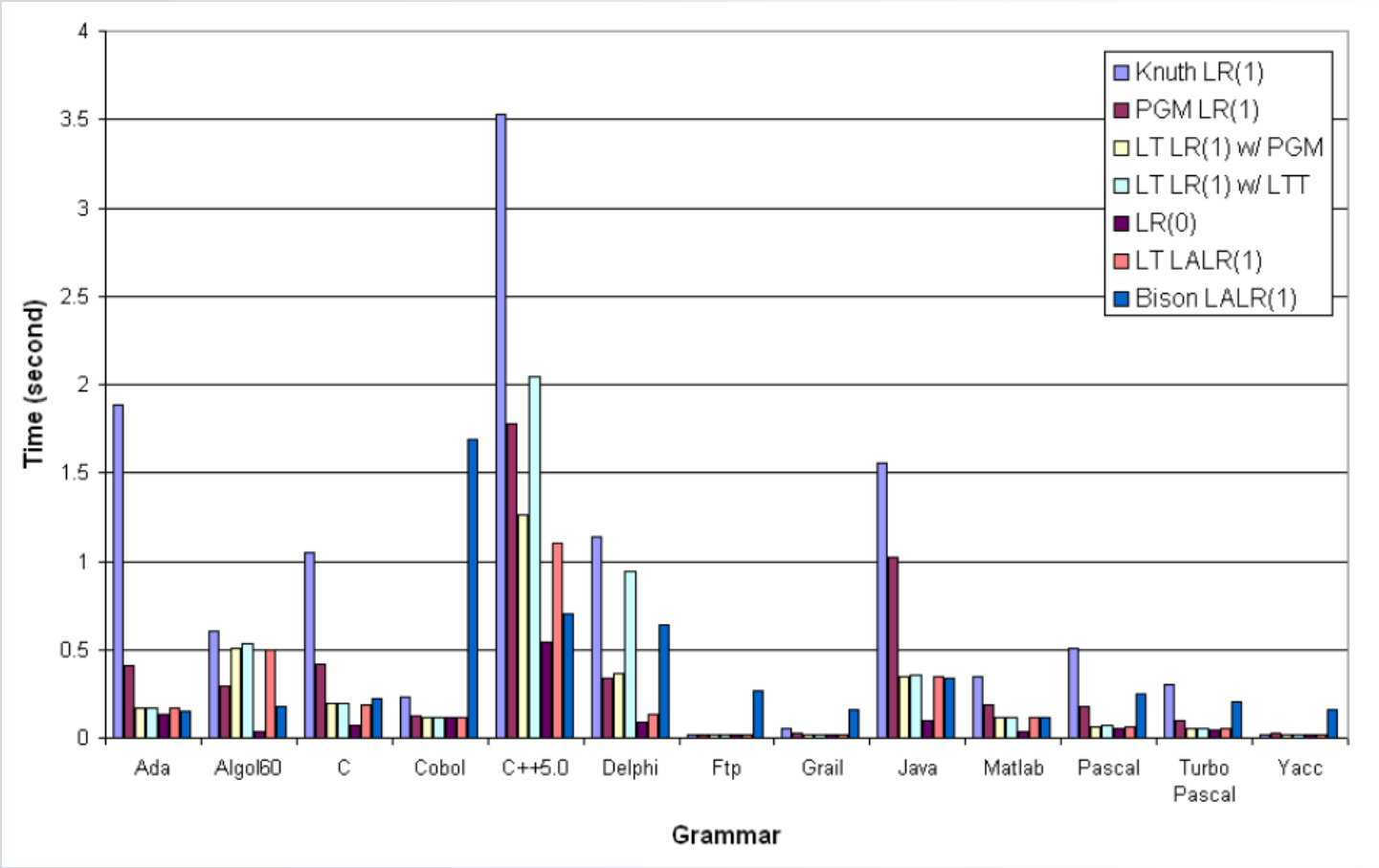
Comparing algorithms

- Parsing machine size (number of states) comparison



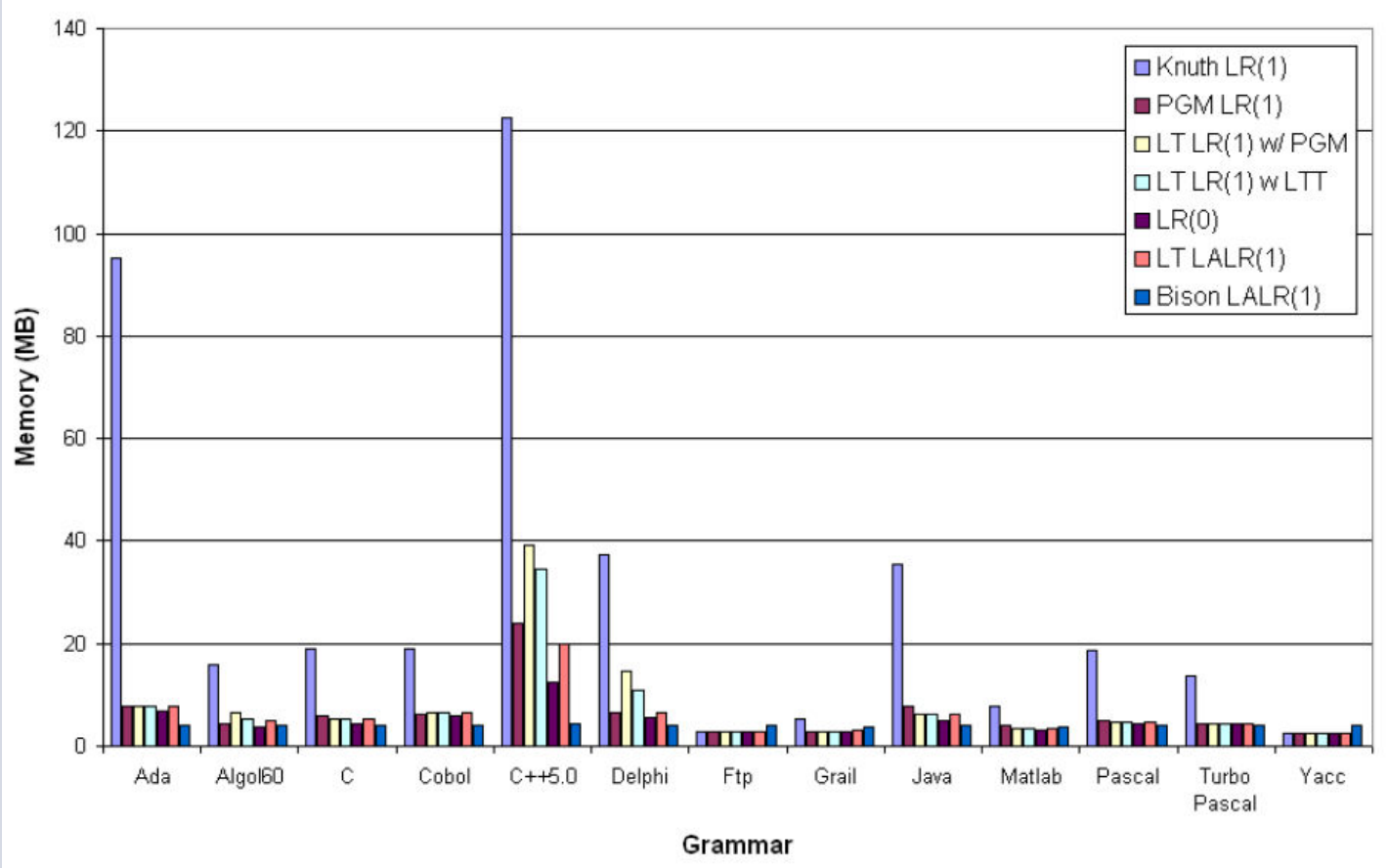
Comparing algorithms

- Running time comparison



Comparing algorithms

- Memory usage comparison



Summary on measuring LR(1) algorithms

- Reduced-space LR(1) algorithms (PGM, LT) v.s. Knuth LR(1)
  - Reduced-space LR(1) has much better performance
- Reduced-space LR(1) v.s. LALR(1)
  - Close in Performance for LALR(1) grammars
- PGM v.s. LT
  - PGM: easier to understand and implement
  - LT: preferred for extension to LR(k)

V. Related work

- Implementations of Practical General Method
  - LR: ANSI standard Fortran 66 (Lawrence Liverpool Lab, 1981)
  - LRSYS: Pascal (Lawrence Liverpool Lab, 1985)
  - LALR: MACRO-11, on a RSX-11 machine (1988)
  - GDT_PC: (Sweden, 1988)
  - Menhir: Objective Caml (France, 2004)
  - The Python Parsing module: Python (2007)

V. Related work

- Implementations of Practical General Method
  - LR: ANSI standard Fortran 66 (Lawrence Liverpool Lab, 1981)
  - LRSYS: Pascal (Lawrence Liverpool Lab, 1985)
  - LALR: MACRO-11, on a RSX-11 machine (1988)
  - GDT_PC: (Sweden, 1988)
  - Menhir: Objective Caml (France, 2004)
  - The Python Parsing module: Python (2007)
- Implementations of Lane-tracing Algorithm
  - In Assembly for OS 360/65 (1977)

V. Related work

- Implementations of Practical General Method
  - LR: ANSI standard Fortran 66 (Lawrence Liverpool Lab, 1981)
  - LRSYS: Pascal (Lawrence Liverpool Lab, 1985)
  - LALR: MACRO-11, on a RSX-11 machine (1988)
  - GDT_PC: (Sweden, 1988)
  - Menhir: Objective Caml (France, 2004)
  - The Python Parsing module: Python (2007)
- Implementations of Lane-tracing Algorithm
  - In Assembly for OS 360/65 (1977)
- Implementations of Spector's splitting Algorithm
  - Muskox (1994)

V. Related work

- Implementations of Practical General Method
  - LR: ANSI standard Fortran 66 (Lawrence Liverpool Lab, 1981)
  - LRSYS: Pascal (Lawrence Liverpool Lab, 1985)
  - LALR: MACRO-11, on a RSX-11 machine (1988)
  - GDT_PC: (Sweden, 1988)
  - Menhir: Objective Caml (France, 2004)
  - The Python Parsing module: Python (2007)
- Implementations of Lane-tracing Algorithm
  - In Assembly for OS 360/65 (1977)
- Implementations of Spector's splitting Algorithm
  - Muskox (1994)
- Other LR implementations
  - MSTA (possibly splitting LALR parsing machine)
  - Dr. Parse, Yacc++ (Commercial, details unknown)

V. Related work

- Advantages of Hyacc
  - Algorithm coverage
  - Efficiency
  - Portability
  - Usability
  - Availability
  - open source

VI. Contributions and Limitations

- Contribution
  - Implemented an efficient, practical parser generator: Hyacc
  - Evaluated different LR(1) algorithms
  - Showed LR(1) parser generation is practical
  - Potential influence on the industry
    - LR(1) to replace LALR(1)

VI. Contributions and Limitations

- Limitations
  - Hyacc feature needs enrichment
    E.g. Several more directives in Yacc still need implementation:
    %union, %type, %nonassoc
  - Parse engine available in C only
  - LR(k) still incomplete

VII. Summary

- Motivation
  - LR(1) has advantage over non-LR(1) algorithms
  - LR(1) problem: performance
  - Reduced space LR(1) algorithms exist but need study
  - Industry need

VII. Summary

- Motivation
  - LR(1) has advantage over non-LR(1) algorithms
  - LR(1) problem: performance
  - Reduced space LR(1) algorithms exist but need study
  - Industry need
- Current work
  - Measured and evaluated algorithms
  - Released LR(0)/LALR(1)/LR(1)/partial LR(k) parser generator Hyacc

VII. Summary

- Motivation
  - LR(1) has advantage over non-LR(1) algorithms
  - LR(1) problem: performance
  - Reduced space LR(1) algorithms exist but need study
  - Industry need
- Current work
  - Measured and evaluated algorithms
  - Released LR(0)/LALR(1)/LR(1)/partial LR(k) parser generator Hyacc
- Hyacc Future work
  - More features
  - Support more languages in parse engine
  - LR(k)

Questions?

Double Rainbow Over Kauai - Hawaii