A reference GLL implementation

October 2023

Adrian Johnstone

ROYAL HOLLOWAY UNIVERSITY OF LONDON

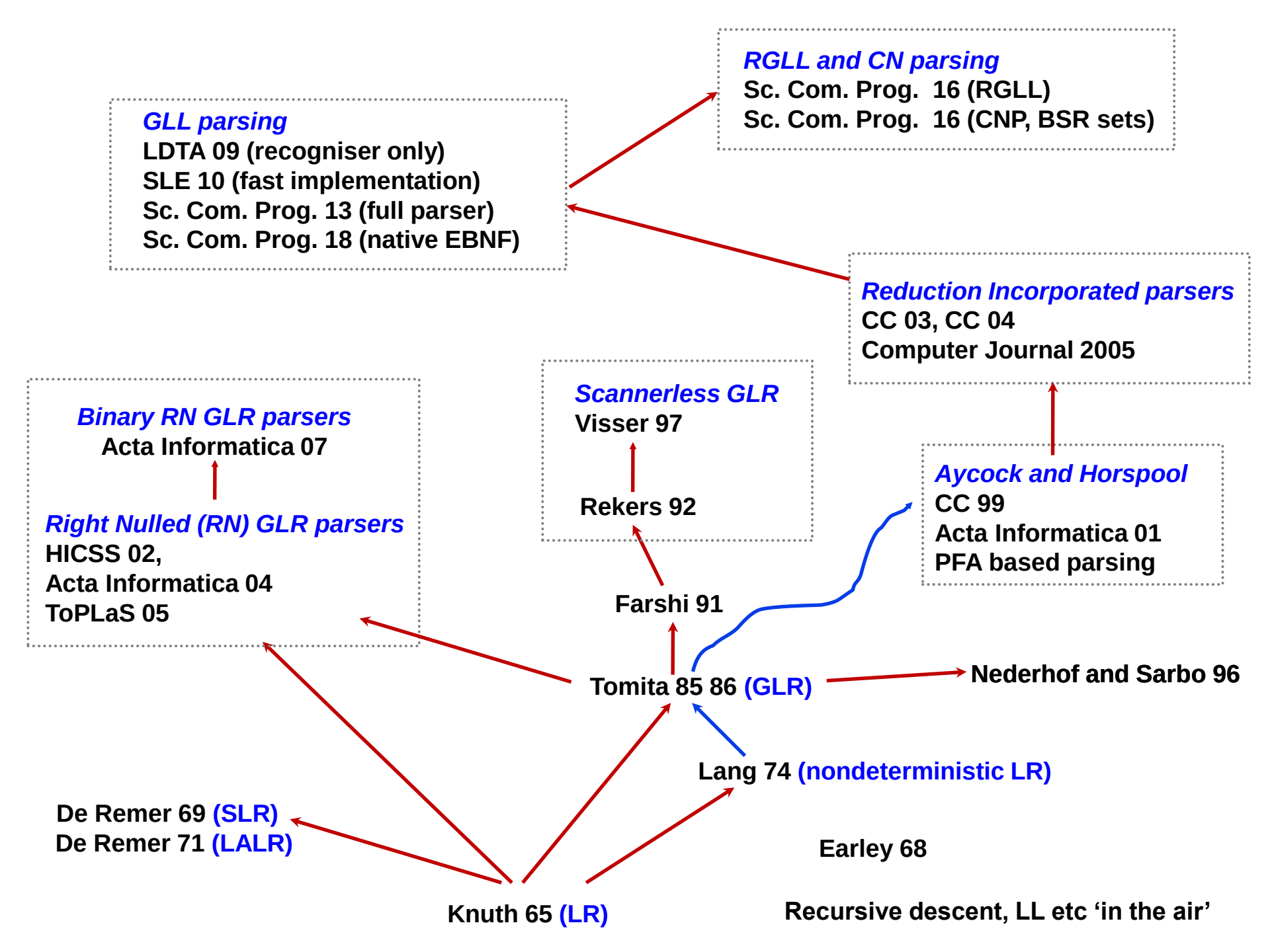
Acknowledgements

The Leverhulme Trust

EPSRC
Engineering and Physical Sciences Research Council

The Leverhulme Trust for project grant RPG-2013-396

EPSRC for project EP/I032509/1


RGLL and CN parsing
Sc. Com. Prog. 16 (RGLL)
Sc. Com. Prog. 16 (CNP, BSR sets)
GLL parsing
LDTA 09 (recogniser only)
SLE 10 (fast implementation)
Sc. Com. Prog. 13 (full parser)
Sc. Com. Prog. 18 (native EBNF)
Reduction Incorporated parsers
CC 03, CC 04
Computer Journal 2005
Binary RN GLR parsers
Acta Informatica 07
Scannerless GLR
Visser 97
Aycock and Horspool
CC 99
Acta Informatica 01
PFA based parsing
Right Nulled (RN) GLR parsers
HICSS 02,
Acta Informatica 04
ToPLaS 05
Rekers 92
Farshi 91
Tomita 85 86 (GLR)
Nederhof and Sarbo 96
Lang 74 (nondeterministic LR)
De Remer 69 (SLR)
De Remer 71 (LALR)
Earley 68
Knuth 65 (LR)
Recursive descent, LL etc 'in the air'

LALR/LL parsing: fine for linear things
Flips over if you drive it too hard

Santa Pod European Final 2010

Backtracking parsers: OK on special surfaces
Breaks down easily

2011 Japanese GP: Jenson Button (McLaren) during race

General parsers: go anywhere
Slow and bulky

Land Rover Defender 90 1999

https://www.youtube.com/watch?app=desktop&v=Cz1BpbsbFkA

Some use cases

GEG – Good Enough for Gnu

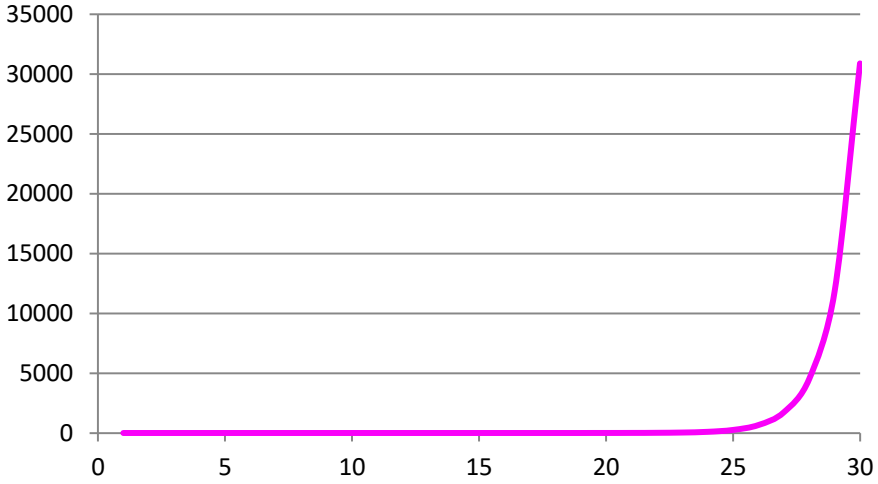
GER – Good Enough for RE matching

GEX – Good Enough for XML

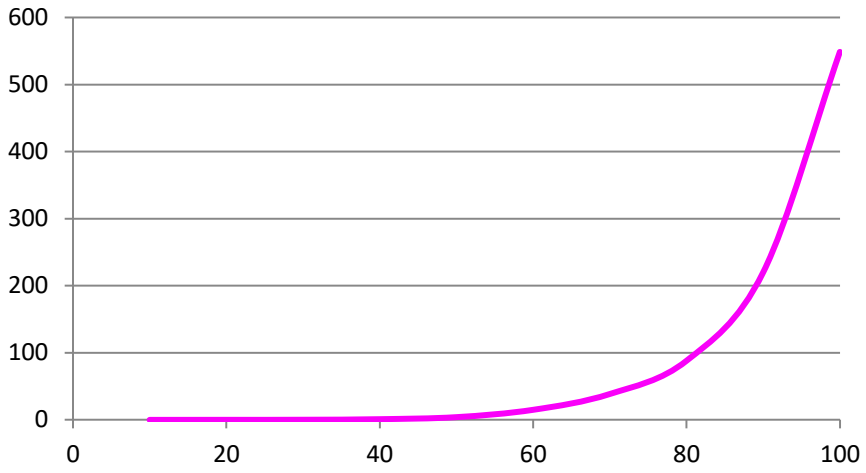
GES – Good Enough for Semantics

GEP – Good Enough for Pathological Grammars

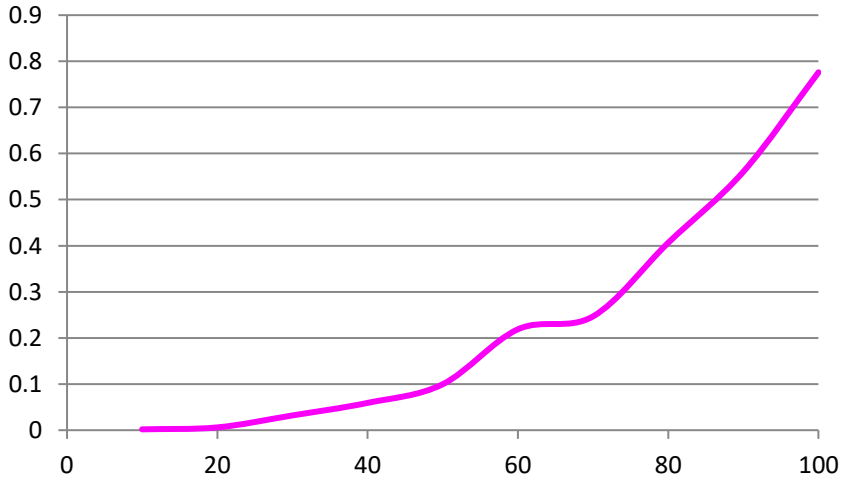
Parse b^n to SPPF with S ::= b | S S | S S S



JSDF

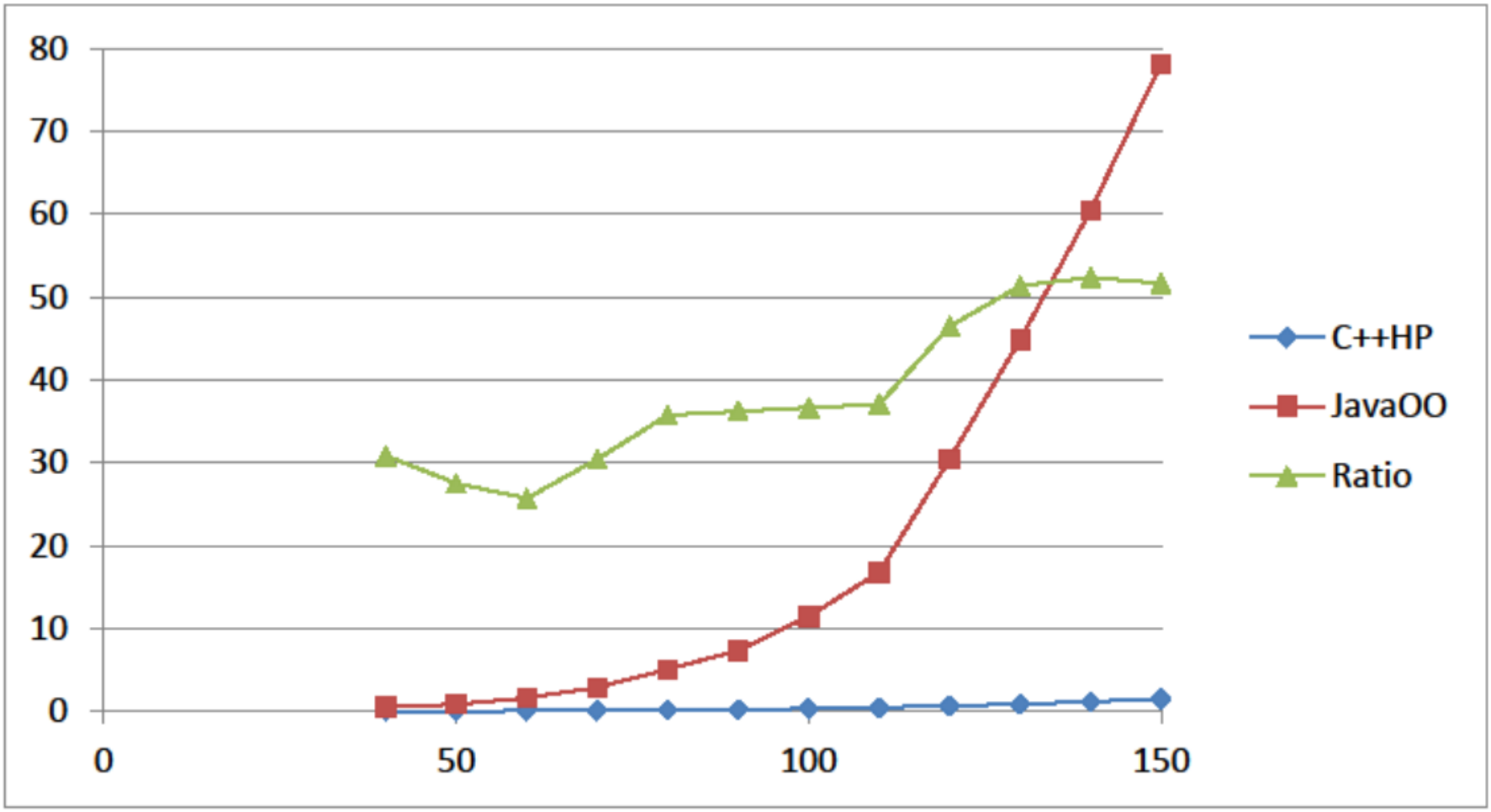


SDF2



Compiled GLL (ANSI C)

Caveat implementor

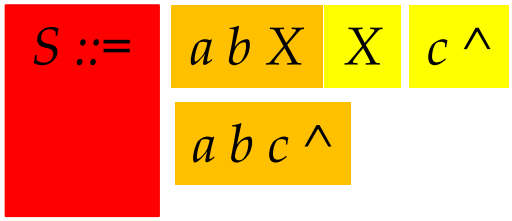


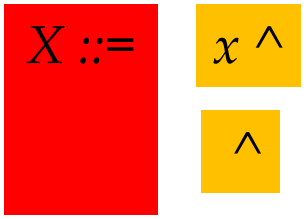
GLL fragments

A grammar in general is a nondeterministic specification of a language

GLL provides a sequentialisation of the general parsing problem. The basic idea is to split the grammar specification up into a set of GLL fragments

S ::= a b X X c | a b c X ::= a | #





A control flow view of GLL

The core idea is that a piece of GLL computation comprises the triggering of a GLL fragment with a configuration comprising a stack *v*, a partially constructed derivation tree *w,* and an input index *i*

A GLL descriptor (*L, v, i, w*) associates a GLL fragment labelled *L* with a configuration (*v, i, w*)

At the outer level, GLL works by processing descriptors which can create more descriptors, effectively scheduling future computation

An instance X' of X causes a jump to the (red) LHS code for X which creates descriptors for the initial fragments of X (orange)

When an executing fragment reaches the end of a production ^, a return (pop) action creates descriptors for the (yellow) fragment following X'

Stack management

A graph structured stack records the call graph of the parse functions

Stacks with common prefixes can be trivially shared

Due to the context free nature of parsing, stacks with the same top element can be merged