Parsing Expression Grammar and Packrat Parsing (Survey)

IPLAS Seminar Oct 27, 2009

Kazuhiro Inaba

This Talk is Based on These Resources

- The Packrat Parsing and PEG Page (by Bryan Ford)
  - http://pdos.csail.mit.edu/~baford/packrat/
    - (was active till early 2008)
- A. Birman & J. D. Ullman, “Parsing Algorithms with Backtrack”, Information and Control (23), 1973
- B. Ford, “Packrat Parsing: Simple, Powerful, Lazy, Linear Time”, ICFP 2002
- B. Ford, “Parsing Expression Grammars: A Recognition-Based Syntactic Foundation”, POPL 2004

Outline

- What is PEG?
  - Introduce the core idea of Parsing Expression Grammars
- Packrat Parsing
  - Parsing Algorithm for the core PEG
- Packrat Parsing Can Support More…
  - Syntactic predicates
- Full PEG
  - This is what is called "PEG" in the literature.
- Theoretical Properties of PEG
- PEG in Practice

What is PEG?

- Yet Another Grammar Formalism
  - Intended for describing grammars of programming languages (not for NL, nor for program analysis)
  - As simple as Context-Free Grammars
  - Linear-time parsable
  - Can express:
    - All deterministic CFLs (LR(k) languages)
    - Some non-CFLs

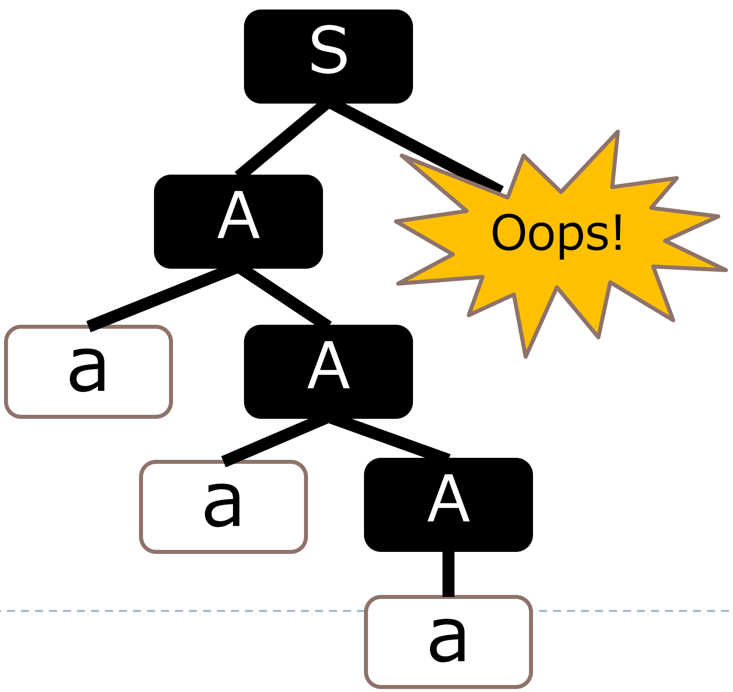
What is PEG? – Comparison to CFG

(Predicate-Free) Parsing Expression Grammar	Context-Free Grammar
A ← B C	A → B C
Concatenation	Concatenation
A ← B / C	A → B | C
Prioritized Choice	Unordered Choice
When both B and C matches, prefer B	When both B and C matches, either will do

Example

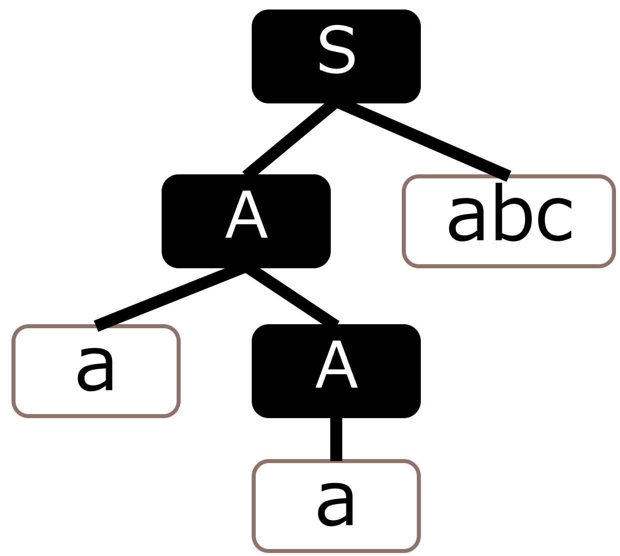
(Predicate-Free) Parsing Expression Grammar

- S ← A a b c
- A ← a A / a
  - S fails on “aaabc”.



Context-Free Grammar

- S → A a b c
- A → a A | a
  - S recognizes “aaabc”



Another Example

(Predicate-Free) Parsing Expression Grammar

- S ← E ;
  / while (E) S
  / if (E) S else S
  / if (E) S
  / …

- ```
  if(x>0)
      if(x<9)
          y=1;
      else
          y=3;
  ```
  unambiguous

Context-Free Grammar

- S → E ;
  | while (E) S
  | if (E) S else S
  | if (E) S
  | …

- ```
  if(x>0)
      if(x<9)
          y=1;
      else
          y=3;
  ```
  ambiguous

Formal Definition

- Predicate-Free PEG G is <N, Σ, S, R>
  - N : Finite Set of Nonterminal Symbols
  - Σ : Finite Set of Terminal Symbols
  - S ∈ N : Start Symbol
  - R ∈ N → rhs : Rules, where
    - rhs ::= ε
| A (∈ N)
| a (∈ Σ)
| rhs / rhs
| rhs rhs
    - Note: A←rhs stands for R(A)=rhs
    - Note: Left-recursion is not allowed

Semantics

- [[e]] :: String → Maybe String where String=Σ*
- [[c]] = λs → case s of (for c ∈ Σ)
  - c : t → Just t
  - _ → Nothing
- [[e1 e2]] = λs → case [[e1]] s of
  - Just t → [[e2]] t
  - Nothing → Nothing
- [[e1 / e2]] = λs → case [[e1]] s of
  - Just t → Just t
  - Nothing → [[e2]] s
- [[ε]] = λs → Just s
- [[A]] = [[R(A)]] (recall: R(A) is the unique rhs of A)

Example (Complete Consumption)

S ← a S b / c

- [[S]] “acb” = Just “”
  - [[aSb]] “acb” = Just “”
    - [[a]] “acb” = Just “cb”
    - [[S]] “cb” = Just “b”
      - [[aSb]] “cb” = Nothing
        - [[a]] “cb” = Nothing
      - [[c]] “cb” = Just “b”
    - [[b]] “b” = Just “”

Example (Failure, Partial Consumption)

S ← a S b / c

- [[S]] “b” = Nothing
  - [[aSb]] “b” = Nothing
    - [[a]] “b” = Nothing
  - [[c]] “b” = Nothing

- [[S]] “cb” = Just “b”
  - [[aSb]] “cb” = Nothing
    - [[a]] “cb” = Nothing
  - [[c]] “cb” = Just “b”

Example (Prioritized Choice)

S ← A a
A ← a A / a

- [[S]] “aa” = Nothing
  - Because [[A]] “aa” = Just “”, not Just “a”

  - [[A]] “aa” = Just “”
    - [[a]] “aa” = Just “a”
    - [[A]] “a” = Just “”
      - …

“Recognition-Based”

- In “generative” grammars such as CFG, each nonterminal defines a language (set of strings) that it generates.
- In “recognition-based” grammars, each norterminal defines a parser (function from string to something) that it recognizes.

Outline

- What is PEG?
  - Introduce the core idea of Parsing Expression Grammars
- Packrat Parsing
  - Parsing Algorithm for the core PEG
- Packrat Parsing Can Support More…
  - Syntactic predicates
- Full PEG
  - This is what is called "PEG" in the literature.
- Theoretical Properties of PEG
- PEG in Practice

Parsing Algorithm for PEG

- Theorem: Predicate-Free PEG can be parsed in linear time wrt the length of the input string.

- Proof
  - By Memoization
    (All arguments and outputs of
      - [[e]] :: String -> Maybe String

    are the suffixes of the input string)

Parsing Algorithm for PEG

- How to Memoize?
  - Tabular Parsing [Birman&Ullman73]
    - Prepare a table of size |G| × |input|, and fill it from right to left.
  - Packrat Parsing [Ford02]
    - Use lazy evaluation.

Parsing PEG (1: Vanilla Semantics)

S ← aS / a

- doParse = parseS :: String -> Maybe String
- parseA s =
  - case s of 'a':t -> Just t
  - _ -> Nothing
- parseS s = alt1 `mplus` alt2 where
  - alt1 = case parseA s of
    - Just t -> case parseS t of
      - Just u -> Just u
      - Nothing -> Nothing
    - Nothing-> Nothing
  - alt2 = parseA s

Parsing PEG (2: Valued)

S ← aS / a

- doParse = parseS :: String -> Maybe (Int, String)
- parseA s =
  - case s of 'a':t -> Just (1, t)
  - _ -> Nothing
- parseS s = alt1 `mplus` alt2 where
  - alt1 = case parseA s of
    - Just (n,t)-> case parseS t of
      - Just (m,u)-> Just (n+m,u)
      - Nothing -> Nothing
    - Nothing -> Nothing
  - alt2 = parseA s

Parsing PEG (3: Packrat Parsing)

S ← aS / a

- type Result = Maybe (Int, Deriv)
- data Deriv = D Result Result

- doParse :: String -> Deriv
- doParse s = d where
  - d = D resultS resultA
  - resultS = parseS d
  - resultA = case s of ‘a’:t -> Just (1,next)
  - _ -> Nothing
  - next = doParse (tail s)
- …

Parsing PEG (3: Packrat Parsing, cnt'd)

S ← aS / a

- type Result = Maybe (Int, Deriv)
- data Deriv = D Result Result

- parseS :: Deriv -> Result
- parseS (D rS0 rA0) = alt1 `mplus` alt2 where
  - alt1 = case rA0 of
    - Just (n, D rS1 rA1) -> case rS1 of
      - Just (m, d) -> Just (n+m, d)
      - Nothing -> Nothing
    - Nothing -> Nothing
  - alt2 = rA0

- alt1 = case parseA s of
  - Just (n,t)-> case parseS t of
    - Just (m,u)-> Just (n+m,u)
    - Nothing -> Nothing
  - Nothing -> Nothing
- alt2 = parseA s

Packrat Parsing Can Do More

- Without sacrificing linear parsing-time, more operators can be added. Especially, “syntactic predicates”:

  - [[&e]] = λs → case [[e]] s of
    - Just _ → Just s
    - Nothing → Nothing

  - [[!e]] = λs → case [[e]] s of
    - Just _ → Nothing
    - Nothing → Just s

Formal Definition of PEG

- PEG G is <N, Σ, S, R∈N→rhs> where
  - rhs ::= ε
    - | A (∈ N)
    - | a (∈ Σ)
    - | rhs / rhs
    - | rhs rhs
    - | &rhs
    - | !rhs
    - | rhs? (eqv. to X where X←rhs/ε)
    - | rhs* (eqv. to X where X←rhs X/ε)
    - | rhs+ (eqv. to X where X←rhs X/rhs)

Example: A Non Context-Free Language

- $\{a^n b^n c^n \mid n>0\}$

  is recognized by

- S ← &X a* Y !a !b !c
- X ← aXb / ab
- Y ← bYc / bc

Example: C-Style Comment

- C-Style Comment

- Comment ← /* ((! */) Any)* */
  - (for readability, meta-symbols are colored)

- Though this is a regular language, it cannot be written this easy in conventional regex.

Outline

- What is PEG?
  - Introduce the core idea of Parsing Expression Grammars
- Packrat Parsing
  - Parsing Algorithm for the core PEG
- Packrat Parsing Can Support More…
  - Syntactic predicates
- Full PEG
  - This is what is called “PEG” in the literature.
- Theoretical Properties of PEG
- PEG in Practice

Theoretical Properties of PEG

- Two Topics
  - Properties of Languages Defined by PEG
  - Relationship between PEG and predicate-free PEG

Language Defined by PEG

- For a parsing expression e
- [Ford04] F(e) = {w∈Σ* | [[e]]w ≠ Nothing }
- [BU73] B(e) = {w∈Σ* | [[e]]w = Just "" }
- [Redziejowski08]
  - R. R. Redziejowski, "Some Aspects of Parsing Expression Grammar", Fundamenta Informaticae(85), 2008
    - Investigation on concatenation [[e1 e2]] of two PEGs
  - S(e) = {w∈Σ* | ∃u. [[e]]wu = Just u }
  - L(e) = {w∈Σ* | ∀u. [[e]]wu = Just u }

Properties of $F(e) = \{w \in \Sigma^* \mid [[e]]w \neq \text{Nothing}\}$

- F(e) is context-sensitive
- Contains all deterministic CFL
- Trivially Closed under Boolean Operations
  - $F(e1) \cap F(e2) = F(\ (\&e1)e2\ )$
  - $F(e1) \cup F(e2) = F(\ e1\ /\ e2\ )$
  - $\sim F(e) = F(\ !e\ )$
- Undecidable Problems
  - "$F(e) = \Phi$"? is undecidable
    - Proof is similar to that of intersection emptiness of context-free languages
  - "$F(e) = \Sigma^*$"? is undecidable
  - "$F(e1)=F(e2)$"? is undecidable

Properties of B(e) = $\{w \in \Sigma^* \mid [[e]]w = \text{Just ``''}\}$

- B(e) is context-sensitive
- Contains all deterministic CFL

- For predicate-free e1, e2
  - B(e1)∩B(e2) = B(e3) for some predicate-free e3
- For predicate-free & well-formed e1,e2 where
  well-formed means that [[e]] s is either Just”” or Nothing
  - B(e1)∪B(e2) = B(e3) for some pf&wf e3
  - ~B(e1) = B(e3) for some predicate-free e3

- Emptiness, Universality, and Equivalence is undecidable

Properties of B(e) = {w∈Σ* | [[e]]w = Just “”}

- Forms AFDL, i.e.,
  - markedUnion(L_1, L_2) = $aL_1 \cup bL_2$
  - markedRep(L1) = $(aL_1)^*$
  - marked inverse GSM (inverse image of a string transducer with explicit endmarker)

- [Chandler69] AFDL is closed under many other operations, such as left-/right- quotients, intersection with regular sets, ⋯
  - W. J. Chandler, “Abstract Famlies of Deterministic Languages”, STOC 1969

Predicate Elimination

- Theorem: G=<N,Σ,S,R> be a PEG such that F(S) does not contain ε. Then there is an equivalent predicate-free PEG.

- Proof (Key Ideas):
  - [[&e]] = [[!!e]]
  - [[!e C]] = [[(e Z / ε) C]] for ε-free C
    - where $Z = (\sigma_1/\cdots/\sigma_n)Z / \varepsilon$, $\{\sigma_1, \cdots, \sigma_n\} = \Sigma$

Predicate Elimination

- Theorem: PEG is strictly more powerful than predicate-free PEG

- Proof:
  - We can show, for predicate-free e,
    - $\forall$w.([[e]] “” = Just “” $\Leftrightarrow$ [[e]] w = Just w)

    by induction on |w| and on the length of derivation

  - Thus we have
    - “”$\in$F(S) $\Leftrightarrow$ F(S)=Σ*

    but this is not the case for general PEG (e.g., S←!a)

Outline

- What is PEG?
  - Introduce the core idea of Parsing Expression Grammars
- Packrat Parsing
  - Parsing Algorithm for the core PEG
- Packrat Parsing Can Support More…
  - Syntactic predicates
- Full PEG
  - This is what is called “PEG” in the literature.
- Theoretical Properties of PEG
- PEG in Practice

PEG in Practice

- Two Topics
  - When is PEG useful?
  - Implementations

When is PEG useful?

- When you want to unify lexer and parser
  - For packrat parsers, it is easy.
  - For LL(1) or LALR(1) parsers, it is not.

```
list<list<string>>
```

  - Error in C++98, because >> is RSHIFT, not two closing angle brackets
  - Ok in Java5 and C++1x, but with strange grammar

```
(* nested (* comment *) *)
```

```
s = “embedded code #{1+2+3} in string”
```

Implementations

- Java:
  - *Rats!* by Robert Grimm, a part of the eXT
  - ANTLR, a well-established parser generat packrat parsing with LL parsing technique
  - LGI, a dynamic PEG-based parser genera
- Python:
  - The pyparsing monadic parsing combinato
  - Packrat parsing support has also been inc
- Haskell:
  - Frisby by John Meacham is a monadic pa support dynamic specification of composa
  - Pappy by Bryan Ford is a simple prototyp
- **C, C++**:
  - The Narwhal compiler suite by Gordon Tis
  - The PEG Template Library for C++0x by
  - The peg/leg parser generator emphasizes
- **C#**: NPEG is a library providing objects to build
- **JavaScript**: OMeta supports PEG-based patte
- **Tcl**: The new grammar::peg module supports gra
- **Smalltalk**: OMeta provides PEG-based patterr
- **Scheme**: Tony Garnock-Jones has written a p
- **Common Lisp**: CL-peg by John Leuner suppo
- **Lua**: Roberto Ierusalimschy has provided the L
- **Ruby** now has the Treetop grammar descriptio

Performance (Rats!)

- R. Grimm, "Better Extensibility through Modular Syntax", PLDI 2006
  - Parser Generator for PEG, used, e.g., for Fortress

System	Algorithm	Modules	Lex	AST	LoC
Rats!	PEG	9	—	—	790
SDF2	GLR	57	—	—	1,680
Elkhound	LALR/GLR	1	1	1	2,370
ANTLR	LL	1	1	—	1,280
JavaCC	LL	1	1	—	1,240

	Recognizer		Parser	
System	T-put	Heap Util.	T-put	Heap Util.
Rats!	518.0	51.5	317.0	58.0
SDF2	136.1	—	21.4	—
Elkhound	141.5	—	139.4	—
ANTLR	538.6	11.5	393.6	28.0
JavaCC	1,114.3	10.6	382.9	63.2

Experiments on Java1.4 grammar, with sources of size 0.7 ~ 70KB

PEG in Fortress Compiler

- Syntactic Predicates are widely used
  - (though I'm not sure whether it is essential, due to my lack of knowledge on Fortress…)

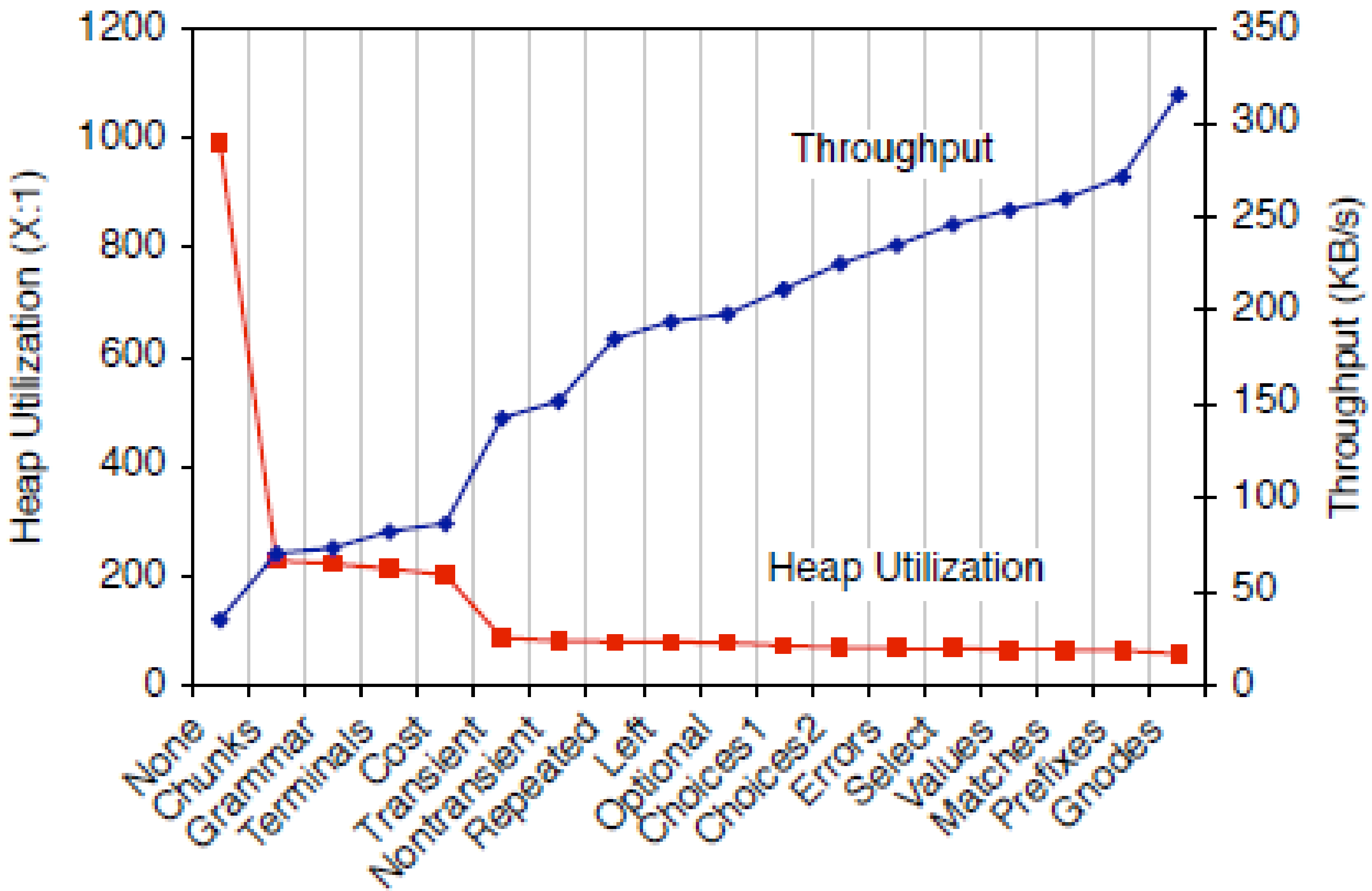
```
/* The operator "|->" should not be in the left-hand sides of map
    expressions  and map/array comprehensions.
 */

String mapstoOp =
     !("|->" w Expr (w mapsto / wr bar / w closecurly / w comma)) "|->" ;

/* The operator "<-" should not be in the left-hand sides of
   generator clause lists. */

String leftarrowOp = !("<-" w Expr (w leftarrow / w comma)) "<-";
```

Optimizations in Rats!



Summary

- Parsing Expression Grammar (PEG) ⋯
  - has prioritized choice e1/e2, rather than unordered choice e1|e2.
  - has syntactic predicates &e and !e, which can be eliminated if we assume ε-freeness.
  - might be useful for unified lexer-parser.
  - can be parsed in O(n) time, by memoizing.