Introduction

- Normal parsing $LL(k)$
- Rules executed in order given
- Re-evaluation of predicates
- Efficiency (lower priority)

What	Notation	Example
rules	lowercase	r, a, a_2
tokens	uppercase	A, B, C
anything	greek	α, β

- LA = lookahead vector of depth k.
- LA[i] = i-th lookahead token
- token vectors AB,AA separated by commas

Generalized Lookahead: Parsing pred-$LL(k)$ languages

Terence Parr
Parr Research Corp
San Fransisco, CA
parrt@parr-research.com

Russell W. Quong
Purdue University
W. Lafayette, IN 47906
quong@ecn.purdue.edu

- Introduction
- Examples
- Generalizations
- Efficiency Hacks
- Guarded Predicates
- Conclusion

```
void r() {
  if ( LA in (AA,BB) ) {
    a_1();
    a_2();
  } else if ( LA in BB ) {
    b();
  } else if ( LA[1] in C ) {
    c();
  }
}
```

```
void a_1 () {
  if ( LA in AA ) {
    consume A;
  } else if ( LA in BA ) {
    consume AB;
  }
} /* a_1 */
```

Normal Parsing (no predicates)

Now consider some examples.

```
  r -> a_1 a_2
     | b
     | c
a_1 -> A
     | BA
a_2 -> A
  b -> BB
  c -> C
```

When testing for an alternative, `alt_X`

```
alt_X = predX $alpha$
```

```
if ( LA() is valid && predX() ) {
  predict alternative_X;
}
```

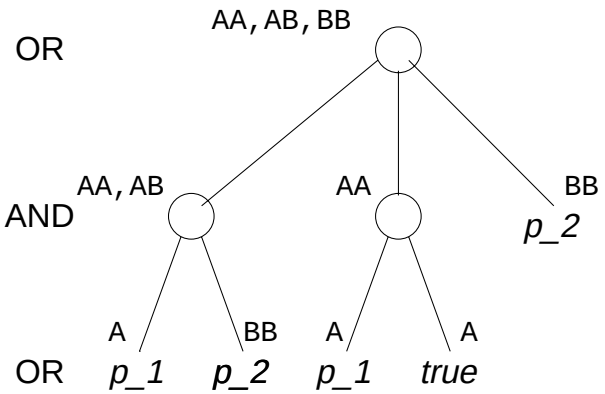
Pred-$LL(k)$ Lookahead

- Evaluate a predicate $\iff$ context is correct.
- Must test if predicate context is correct first.
- Predicate may often uses lookahead
  Say, a symbol table lookup: pred = `isType(LA[2]);`.

```
r -> pred1 a
   | (A | B) C
```

```
void r () {
  if ( LA in FIRST(a) && pred_1() ) {
    a();
  } else if ( LA in (AC,BC) ) {
    consume( [AB]C );
  }
} /* r */
```

```
r -> a b
   | a C
   | b C
a -> pred_1 (A | AA)
b -> pred_2 BB
```



Predicate-context tree

- AND-OR tree, alternating by level
- describes predicting an alternative/rule.
- *evaluating* a node x = evaluating subtree at x
- each node has a *context* = all valid lookahead for subtree at x.
- evaluate a node iff context is correct
- 3 kinds of nodes:
  - AND - true if all children eval true
  - OR - true if any child evals true
  - predicate - just evaluate

Explicit predicate-context guard

E.g. evaluate a `pred_XYZ()`, if LA[2] is X,Y, or Z.

Might try

```
r -> << LA[2] in (X,Y,Z) && pred_XYZ >>? (A|B) (A|B|C|X|Y|Z)
```

```
void r () {
  if ( LA in ([AB][ABCXYZ]) &&
     ( LA[2] in [ABC] && pred_XYZ)
     ) {
     consume two tokens;
  }
} /* r */
```

Wrong! Incorrectly, fails to parse when lookahead is AA.

Reason: manual context makes $<<$ contex && predicate $>>$ fail.

Moral: Cannot use normal && in a predicate for context.

```
void r () {
  if ( (LA in (AA,AB) && pred_1()) &&
     ( LA[2] in (B) && pred_2()) ) {
     a();
     b();
  } else if ( (LA in (AA,AC) && pred_1()) ) {
     a();
     consume C;
  } else if ( (LA in (BB) && pred_2()) ) {
     b();
     consume C;
  }
}
```

Here, LA[2] = second lookahead token.

2) Use a guarded predicate: << context =¿ predicate >>?

Ugh. Yet another ANTLR feature.

Guard = restricts allowable context.

```
void r () {
  if ( LA in ([AB][ABCXYZ]) &&
    ( !  LA[2] in [ABC] || pred_XYZ() )
    ) {
    consume two tokens;
  }
} /* r */
```

2 Solutions:

1) Manually split an alternative.

Ugh. Can be inconvenient or difficult to do.

E.g.

```
r ->  <<pred_XYZ >>?  (A|B) (X|Y|Z)
   |                  (A|B) (A|B|C)
```