Verifiable Composition of Deterministic Grammars

August Schwerdfeger and Eric Van Wyk

Department of Computer Science
and Engineering
University of Minnesota

www.melt.cs.umn.edu
schwerdf@cs.umn.edu
evw@cs.umn.edu

Sample Program, Java + SQL + Tables

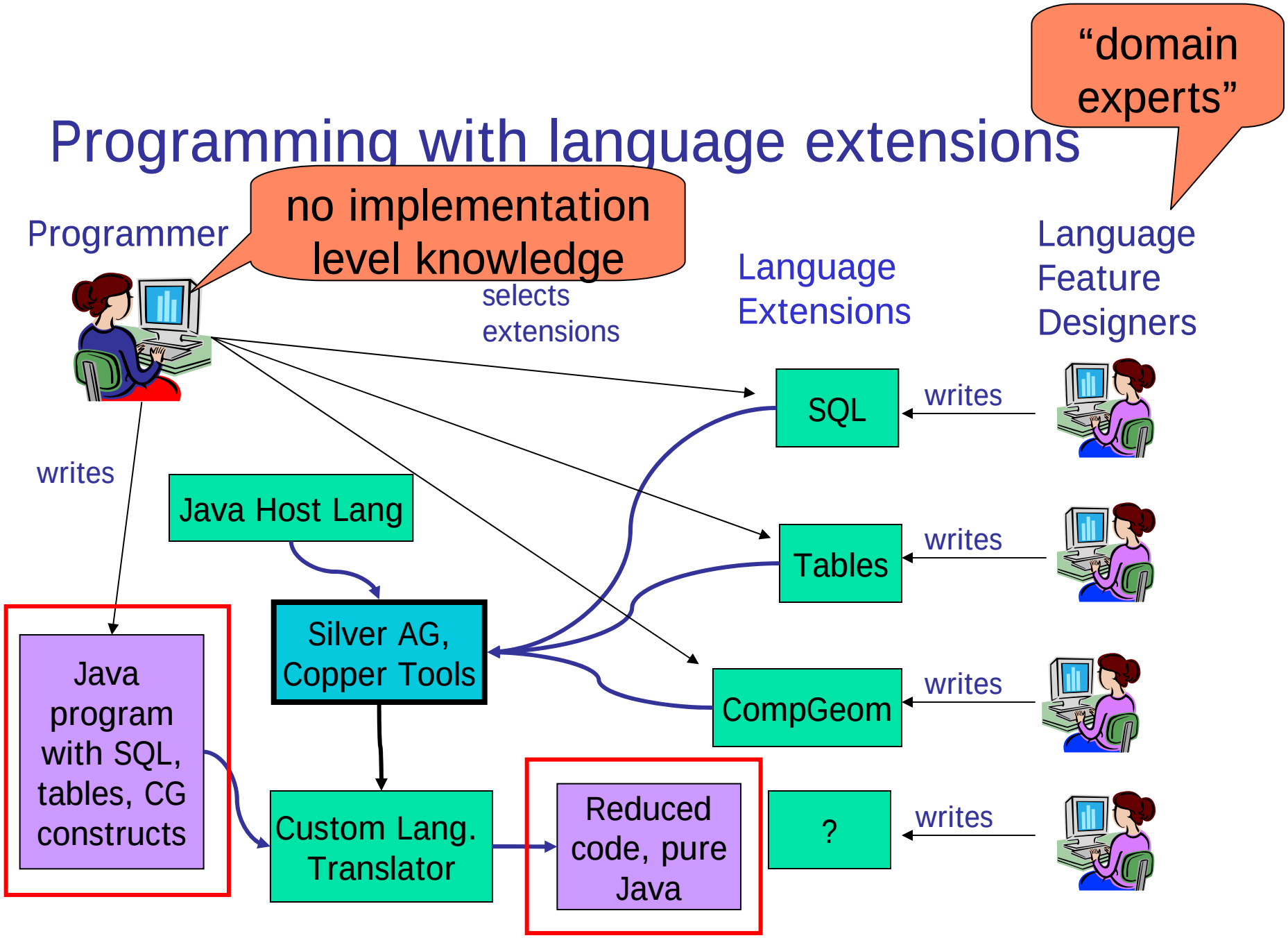
```
class Demo {
  int demo ( ) {
   int SELECT ;
   int T ;
   connection c "jdbc:derby:/home/derby/db/testdb"
      import table person, details ;
   Integer limit = 18 ;
   ResultSet rs = using c query {
         SELECT age, gender, last_name
           FROM person , details
          WHERE person.person_id = details.person_id
            AND details.age > limit } ;
    Integer = rs.getInteger("age");
    String gender = rs.getString("gender")
    boolean b ;
    b = table ( age > 40       : T * ,
                gender == "M" : T F ) ;
  }
}
```

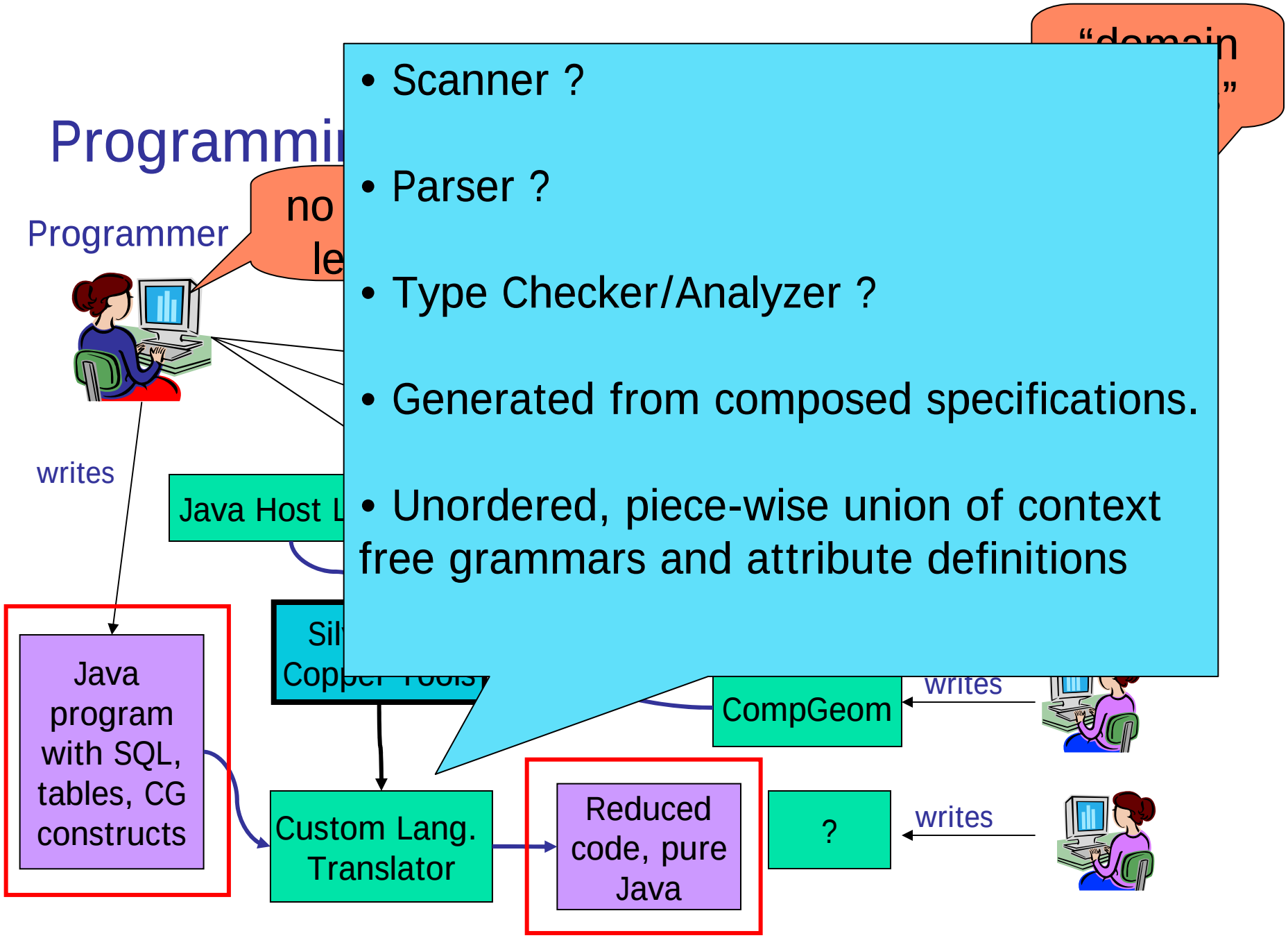
natural syntax

statically detect syntax and type errors in query

Two extensions, from different sources

Programming with language extensions




Programmi
Programmer
writes
Java Host L
Java program with SQL, tables, CG constructs
Custom Lang. Translator
Reduced code, pure Java
CompGeom
writes
?
writes
• Scanner ?
• Parser ?
• Type Checker/Analyzer ?
• Generated from composed specifications.
• Unordered, piece-wise union of context free grammars and attribute definitions

Some challenges.

- ```
  int SELECT ;
  ...
  rs = using c query { SELECT last_name
                       FROM person WHERE … } ;
  ```

- ```
  connection c "jdbc:derby:./derby/db/testdb“
       import table person, details;
  ...
  b = table ( c1 :  T F ,
              c2 :  F * ) ;
  ```

Some challenges.

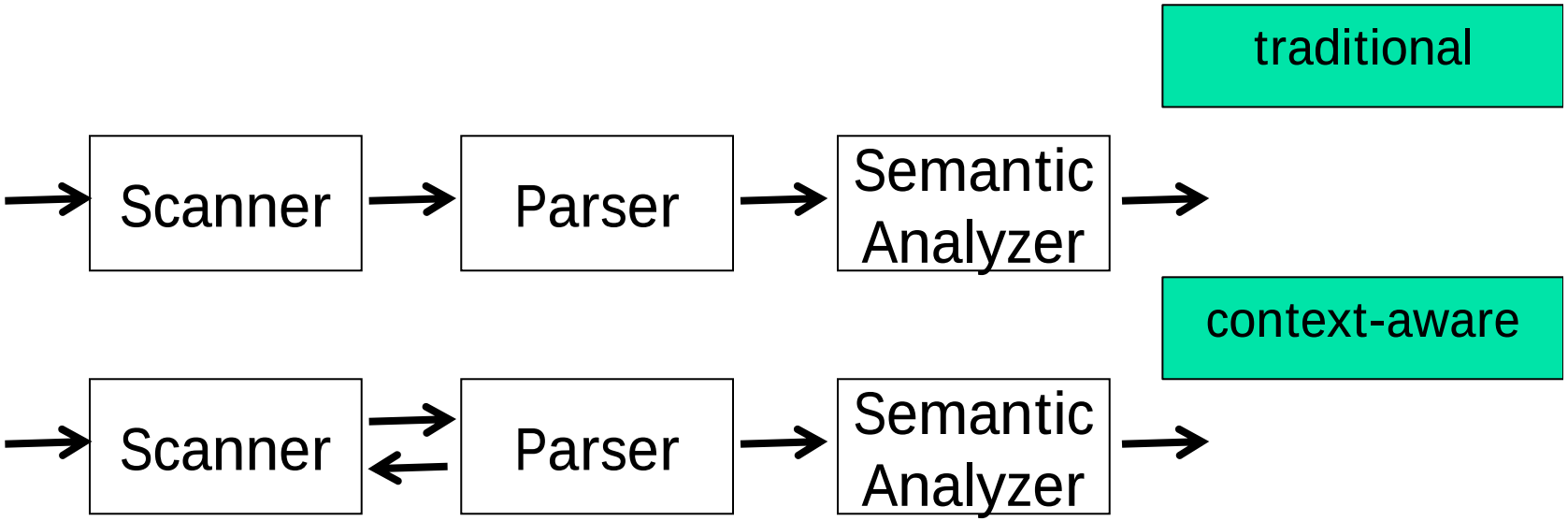
- ```
  x = 3 + y * z ;
  ...
  str =~ /[a-z][a-z0-9]*\.test/
  ```

- ```
  List<List<Integer>> ll ;
  . . .
  x = y >> 4 ;
  ```

- ```
  aspect ... before ... call( o.get*() )
  ...
  x = get*3 ;
  ```

  [from Visser's OOPSLA 06 paper on parsing AspectJ]
  ```

Context Aware Scanning



- Parser passes “valid lookahead terminals” to scanner
  - terminals with shift, reduce, or accept entries in parse table for current LR parse state
- Scanner only returns tokens from the valid look-ahead set.

Context Aware Scanning

- This scanning algorithm subordinates the disambiguation principle of maximal munch to the principle of disambiguation by context.

- It will return a shorter valid match before a longer invalid match.
  - In `List<List<Integer>>`, “>” in valid lookahead, “>>” is not.

- A context aware scanner is essentially an implicitly-moded scanner.
  - Each parser state is a different mode.

- No explicit specification of valid look ahead.
  - Generated from standard grammars and terminal regexs.
- [GPCE 07]

Sample Program, Java + SQL + Tables

```
class Demo {
  int demo ( ) {
    int SELECT ;
    int T ;
    connection c "jdbc:derby:/home/derby/db/testdb"
      import table person, details ;
 Integer limit = 18 ;
   ResultSet rs = using c query {
         SELECT age, gender, last_name
           FROM person , details
          WHERE person.person_id = details.person_id
            AND details.age > limit } ;
    Integer = rs.getInteger("age");
    String gender = rs.getString("gender");
    boolean b ;
    b = table ( age > 40       : T * ,
                gender == "M" : T F ) ;
  }
}
```

We can compose extension grammars to build parser and scanner for this.

Syntactic and Lexical Determinism

- Freedom of conflicts in LR parse table indicates syntactic determinism of the grammar.
- Lexical determinism – scanner will never return more than one terminal symbol.

- A monolithic analysis
  - no conflicts in parse table → syntactic determinism
  - for each parse state p,
    for each pair of distinct terminals t1, t2, both in valid-lookahead(p),
    t1 and t2 regexs must be disjoint languages (no overlap)
    or
    (t1 > t2 or t2 > t1) (disambiguation by lexical precedence)
    → lexical determinism

Modular Analysis

- Programmer combines multiple extensions
  - Will he or she be told of shift-reduce conflict in some state? Or of a lexical ambiguity?
- Can the extension designer check this modularly?

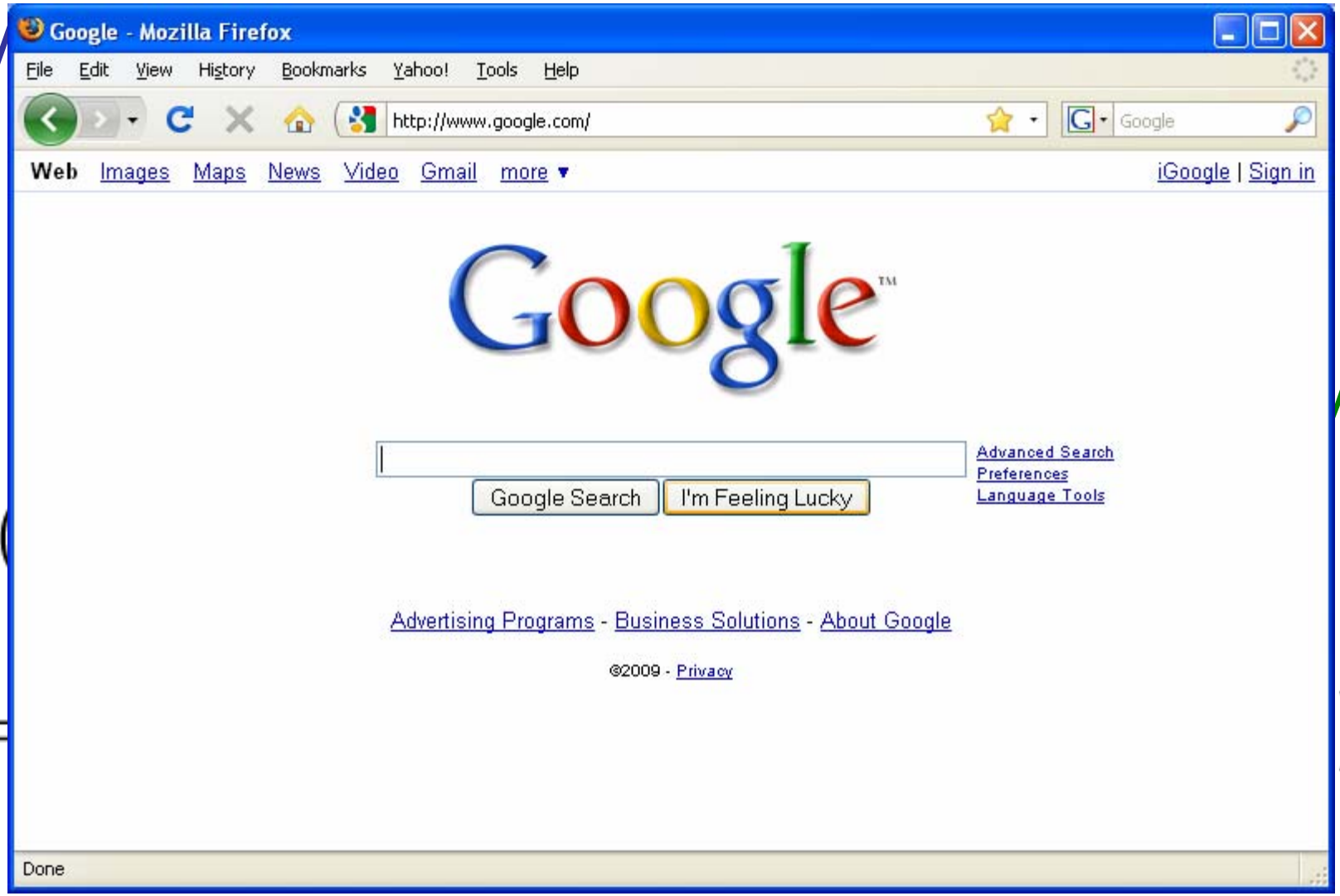
$$(\forall i \in [1,n].isComposable(\Gamma^H, \Gamma_i^E) \wedge conflictFree(\Gamma^H \cup_G \Gamma_i^E)) \Longrightarrow conflictFree(\Gamma^H \cup_G^\star \{\Gamma_1^E, \ldots, \Gamma_n^E\})$$

- [PLDI 09]

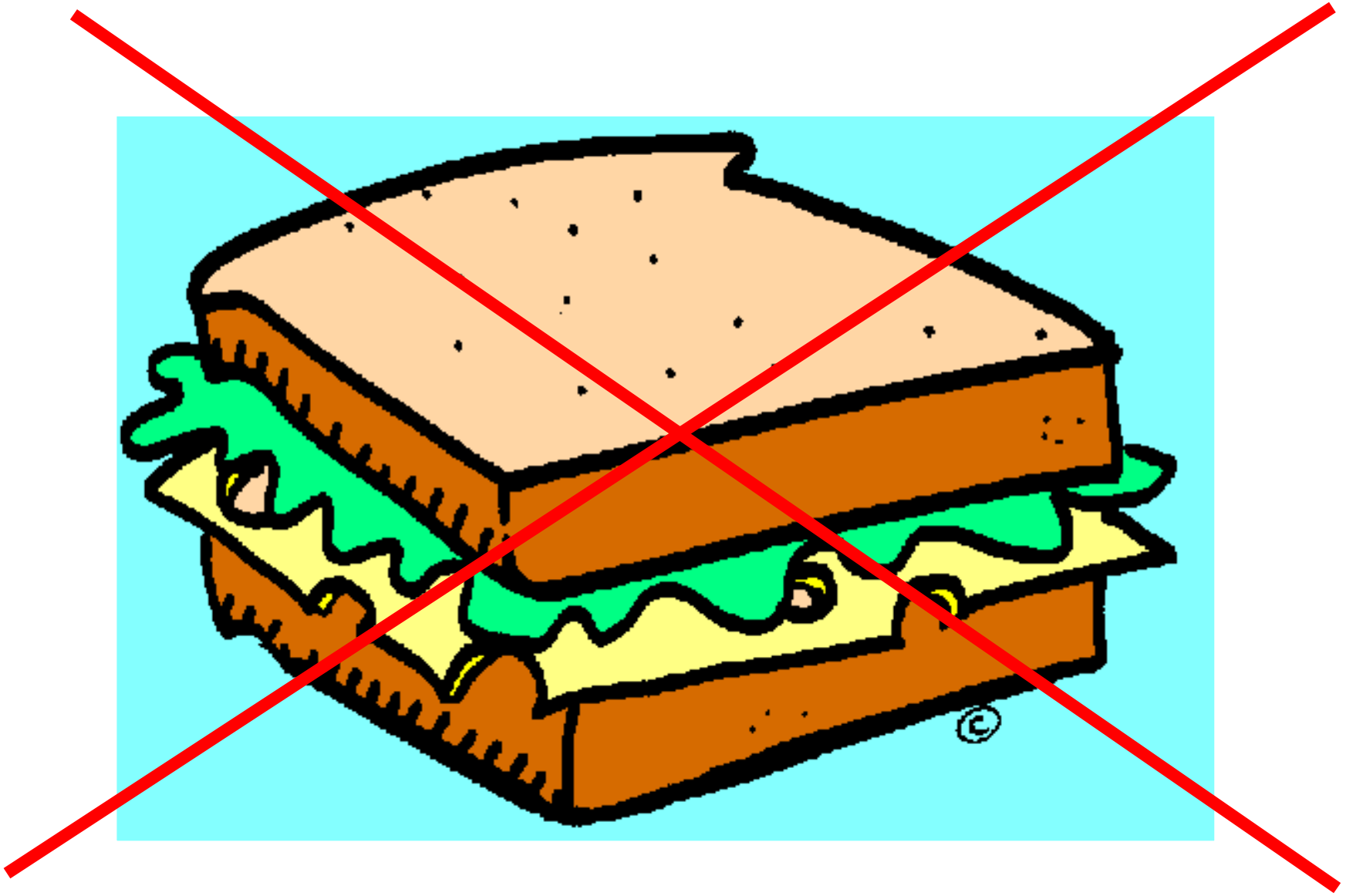
M

-

- y?


Google - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
http://www.google.com/
Google
Web Images Maps News Video Gmail more ▾
iGoogle | Sign in
Google™
Google Search
I'm Feeling Lucky
Advanced Search
Preferences
Language Tools
Advertising Programs - Business Solutions - About Google
©2009 - Privacy
Done

- [PLDI 09]



Goals, Restrictions, Expressiveness

- Extension constructs can contain host language constructs.
  - ext. productions have host NTs on right hand side
  - the tables construct contains host language expressions
- Few restrictions (beyond LALR(1) restrictions) on embedded languages
  - e.g. SQL, embedded regular-expressions
- Restrictions:
  - grammar structure, grammar properties (follow sets), LR DFA
- Expressiveness vs. Safety

Restriction 1 : Grammar structure

- marking token
- ```
  b = table ( age > 40       : T * ,
              gender == "M" : T F ) ;
  ```

- Transition from host state to ext. state only by shifting a marking token.

- *Expr ::= 'table' TRows*

  TRow ::= Expr ':' TFStarList
  ```

Restriction 2: Follow Sets

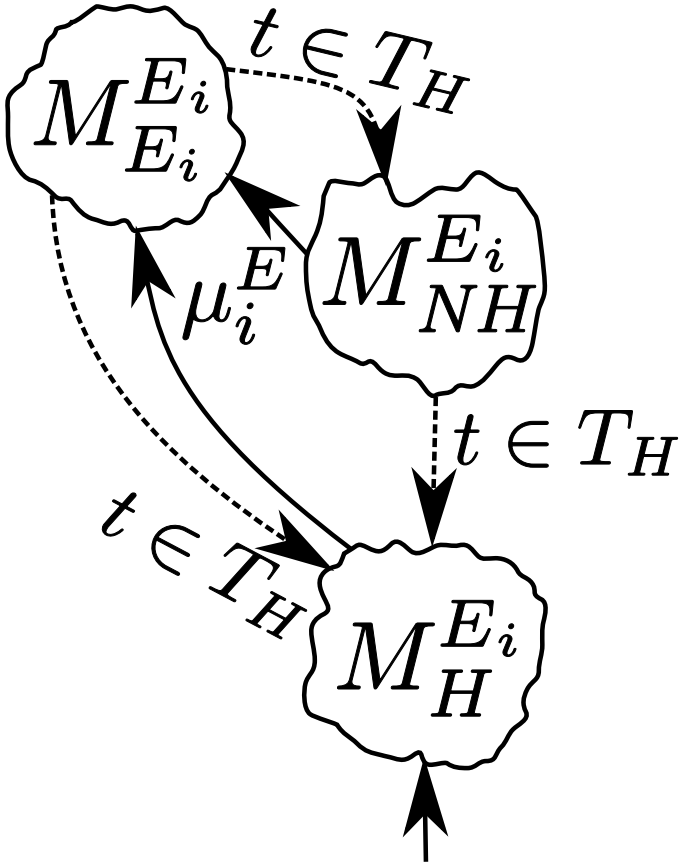
- In the combined host language, single extension grammar ($H + E_i$),
  - no new testament to added the follow sets of host language non terminals.
  - except for marking tokens

Restriction 3: LR DFA

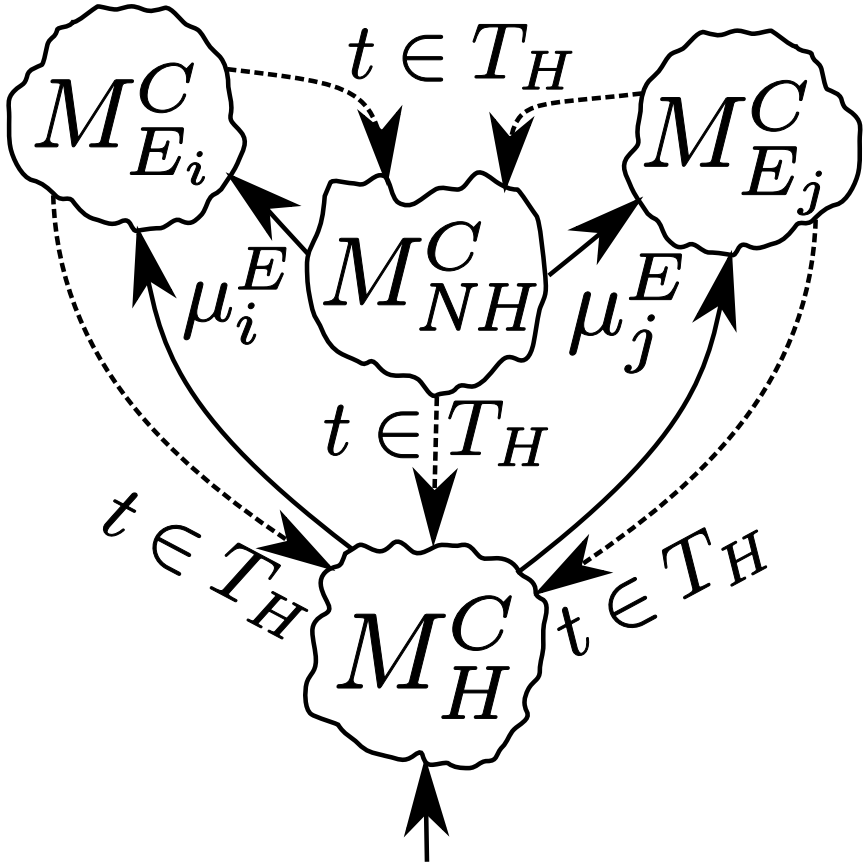
- host language state s in LR-DFA (H + E_i) has no new look-ahead in its items, except marking terminals
- host language states s in LR-DFA (H + E_i)

  and not in LR-DFA (H) are such that their items and look-ahead are the subset of a state in LR-DFA(H)

Partitioning of Host-and-1-ext LR-DFA



Partitioning of Host-and-2-exts LR-DFA



Modular Lexical Determinism Analysis

- Partition of parser DFA ensures that lexical ambiguities are only between
  - terminals in host and a single extension
    - resolved by extension writer
  - marking terminals on two different extensions
    - cannot be resolved by extension writer
    - only programmer who composed the language can resolve them.
    - accomplished by “transparent prefixes”

Experience with the restrictions

- SQL, tables, various other all pass easily
- *Expr ::= ‘table’ TRows*

  TRow ::= Expr ‘:’ TFStarList
- follow set of *Expr* already contained ‘:’
- Extensions are more easily added to syntactically rich languages
  - they have larger follow sets
  - not so easy to add to small toy languages

Restrictions and new infix operators

- Grammars that add new infix operators do not pass the modular analysis.

- However, many extensible language systems allow type-based overloading of existing operators.
- Thus, it is less of a problem.

Restrictions and AspectJ

- Java 1.4 + abc grammar for AspectJ
  - declarative specification for a deterministic scanner and parser
- This fails the modular analysis
  - it adds to follow sets of Java 1.4
  - marking terminals in the wrong place
    - Java: Dcl ::= Modifiers Type *Id* ...
    - AspectJ: Dcl ::= Modifiers *Aspect*
  - Can refactor the host grammar to fix this problem.

Expressiveness vs. Safe composition

- Compare to
  - other parser generators
  - libraries

- The modular compositionality analysis does not require context aware scanning.
- But, context aware scanning makes it practical.

Tool support

- Copper – context-aware parser and scanner generator
  - implements context-aware scanning for a LR parser
  - lexical precedence
  - parser attributes
  - disambiguation functions – when disambiguation by context and lexical precedence is not enough
  - currently integrated into Silver
  - also a stand alone version
  - generated parser and scanner in Java

Related Work

- Traditional LALR(1) parsing tools (Yacc)
  - "brittle" – composition of grammars can introduce shift-reduce and reduce-reduce conflicts.
- Parsing Expression Grammars
  - require an ordering on productions with the same left hand side nonterminal.
- Tattoo
  - Introduced similar notion of parse-state-based context aware scanning.
  - Described only as an optimization. No discussion of increased expressiveness.

Related Work

- Generalized LR
  - parse any CFG
  - Visser's SGLR – Scannerless GLR
    - also uses parser context in recognizing "terminals"
    - parses them in all possible ways and later throws out the ones that don't fit into the parse.
  - not deterministic – for extensible languages some assurance of parser and scanner behavior is desirable
  - trade determinism analysis for larger class of grammars
  - matter of philosophy as to which one prefers

Related Work

- Lexical-based context-aware scanning.
- The two level scanners of Rus, Knaack, and Halverson:
  - One can specify that a regular expression should only match if the token(s) to the left satisfy some criteria.
- Their pattern-matching parser also supports conditional reduction of productions:
  - X ::= a Y b
  - In the sentential form “s r a Y b r” the PMP will replace a Y b with X.
  - This will be allowed only in the right context.
- Scanner and parser are still disjoint.

More information

- www.melt.cs.umn.edu
  - downloads, papers, etc.
- evw@cs.umn.edu
- Thanks to my students - August Schwerdfeger, Jimin Gao, Lijesh Krishnan, Derek Bodin, Yogesh Mali.

  ... Thanks for your attention.
- Thanks to National Science Foundation, IBM, and the McKnight Foundation for funding aspects of this work.