Tolerance in Grammarware

PEM Colloquium
Vadim Zaytsev, SWAT, CWI

CC BY-SA 2012
Grammar-based source code analysis

- A spectrum of approaches w.r.t. tolerance
- We will go from right to left
- Figure borrowed (for extension) from:

Figure 10. A spectrum of approaches for source code analysis.
Precise parsing

Error repair: panic mode

• The simplest method to detect multiple syntax errors

• Provide a list of synchronising tokens (beacon symbols)
  • ;
  • }
  • ...anything obvious and unambiguous

• In case of error, skip everything until the next synchronising token

Error repair: phrase level

- Local correction
- Default options for symbols
- Typically
  - insert ; if it is not present
  - balance the brackets
  - …most heuristics of later blocks of Grammar Hunter
- Sometimes, real error occurs before the detection point

Permissive grammars

- Insertion recovery rules
- Substitution recovery rules
- Choose interpretation with minimum recoveries
- Aimed at error handling
  - error repair
  - error reporting

L. C. L. Kats, M. de Jonge, E. Nilsson-Nyman, E. Visser,
Providing Rapid Feedback in Generated Modular Language Environments, OOPSLA 2009
Global error correction

- Given string \( x \) and grammar \( G \),
  - If \( x \not\in L(G) \),
  - construct string \( y \) such that
    - \( y \in L(G) \)
    - number of changes from \( x \) to \( y \) is minimal
- The closest program is not always the intended one

Hierarchical error repair

• Think of a parser as a state machine
• For every state, there are transitions for allowed tokens
• If an error occurs, no transition for the input token
• A non-error transition is taken
  • based on synchronisation stack
• Compatible at least with LR and LL

Error productions

• Know your enemy
  • Define your enemy with a grammar
• Works well for known kinds of errors
• (Should this be a part of a language?)
  • permissive grammars

Multiple passes

- Lazy iterative skeleton grammar parsing
- Parse first with a skeleton grammar
  - obtain the global structure
- Parse the islands with subgrammars
  - if possible
- Also enables
  - “grammarware as a service” and “parsing in the cloud”

Hopefully: V. Zaytsev, *Islands in the Cloud*, SCAM’12
Parsing in the cloud

Baseline grammar

Adapted grammar

Skeleton grammar

Island subgrammars

Parse tree skeleton

- grammar transformation
- grammar transformation + subgrammar
- delivering islands
- parsing the overall structure
- parsing the islands
- filling the gaps

Hopefully: V. Zaytsev, *Islands in the Cloud*, SCAM’12
Parsing in the cloud

```
compilation-unit:
  using-directives? global-attributes?
  namespace-member-declarations?

using-directives:
  using-directive
  using-directives using-directive

using-directive:
  using-alias-directive
  using-namespace-directive

using-alias-directive:
  "using" id "=" namespace-or-type-name ";"

...```

Hopefully: V. Zaytsev, *Islands in the Cloud*, SCAM'12
Parsing in the cloud

vertical(using-directives);
deyaccify(using-directives);
inline(using-directives);
inline(using-alias-directive);
inline(using-namespace-directive);
massage(using-directive+?, using-directive*);

factor(
  ("using" identifier "=" namespace-or-type-name ";" )
  | ("using" namespace-name ";")
  | ("using" namespace-name ";")
  | (namespace-name ";")
  | (namespace-name) ";" );

...
Parsing in the cloud

compilation-unit:
  ("using" using-directive-insides ";")* ("[" "assembly" ":"
  ga-section-insides "]")* namespace-member-declaration*
namespace-member-main:
  "namespace" qualified-identifier class-base?
  "{" namespace-body-insides "}" ";"?
  "class" identifier class-base?
  "{" class-member-declarations? "}" ";"?
  "struct" identifier struct-interfaces?
  "{" struct-member-declarations? "}" ";"?
  "interface" identifier interface-base?
  "{" interface-member-declarations? "}" ";"?
  "enum" identifier enum-base? "{" enum-body-insides "}" ";"?
  "delegate" type id "(" formal-parameter-list? ")" ";"?

Hopefully: V. Zaytsev, Islands in the Cloud, SCAM'12
layout \( L = [\ \ \t\r\n]* !>> [\ \ \t\r\n] \);

```plaintext
syntax CompilationUnit = ("using" NotSemicolon ";")* 
  ("["] "assembly" ":" NotRightSquareBracket "]")* 
  NamespaceMemberDeclaration* ;
```

```plaintext
syntax NotRightSquareBracket = NRSBChunk+ () >> [\]]];
lexical NRSBChunk = ![\]\ \t\r\n]+ >> [\]\ \t\r\n];
```

---

```
{Parsing in the cloud}
```

```
Hopefully: V. Zaytsev, Islands in the Cloud, SCAM’12
```
Parsing in the cloud

using-directive-insides:

gs-section-insides:

attribute-section-insides:

Adapted grammar

Baseline grammar

grammar transformation

Parse tree skeleton

delivering islands

filling the gaps

Island subgrammars

parsing the overall structure

classifying the islands

Refined parse tree

Hopefully: V. Zaytsev, *Islands in the Cloud*, SCAM’12
Skeleton grammars

- Productions for interesting constructs are reused
- Default productions used for the rest
- Nonterminal mapping is maintained
  - facilitates reasoning about false positives & negatives

S. Klusener, R. Lämmel, *Deriving Tolerant Grammars from a Base-line Grammar*, ICSM 2003
Bridge grammars

• Bridges connect islands
  • can enclose other bridges but never cross

• Reefs add info to nearby islands
  • e.g., indentation and delimiters

• Can be further enhanced with
  • bridge repairer
  • artificial islands

Robust parsing

• Combination of
  • error productions
  • island grammars for multiple languages
  • bridges between islands are parts of islands

• Works well for multilingual parsing
  • e.g., VB + JS + ASP + HTML

N. Synytskyy, J. R. Cordy, T. R. Dean, *Robust Multilingual Parsing Using Island Grammars*, CASCON'03
Island grammars

- Detailed production rules for interesting constructs
- Liberal production rules for the rest
  - ~[\.]+ [\.] → Statement
  - ~[\ \t\n]+ → Water {avoid}
- Minimal set of assumptions about the overall structure
  - (e.g., a program is a list of statements)

Fuzzy parsing

- Floating islands: no [information about] water
- Complete full lexical analysis
- Perform syntactic analysis of selected parts
  - triggered by anchor symbols
- Inspired (and used) by fact extractors

Fuzzy parsing

declare function local:mccabe($w) {
1+count($w//if) +count($w//evaluate/when)
-count($w//evaluate/when[contains(@unparsed,"OTHER")])
+count($w//perform[contains(@unparsed,"TIMES")])
+count($w//perform[contains(@unparsed,"UNTIL")])
+count($w//search/when) +count($w//search/end)};

let $doc := doc("portfolio.xml") return <results>
{for $section in $doc//section, $para in $section/paragraph
let $cc := local:mccabe($para) where $cc>20
return <component>
<section> {data($section/@name)}</section>
<paragraph> {data($para/@label-name)} </paragraph>
<cc>{$cc} </cc>
</component> </results>

Iterative lexical analysis

- Straightforward shortest pattern matching
- \{.*\} \rightarrow Block
- Bottom-up language engineering
- Several levels of matching:
  - from “simple matches” (1) and “short matches” (2)
  - to “good guesses” (7) and “desperation” (8)
- Enables syntactic analysis of irregular code

Hierarchical lexical analysis

- No syntactic constraints
- Works well for conceptual source models
- Even across languages
- Definition example:
  ```plaintext
  [ [type] ] functionName ( [ { formalArg }+ ] )
  [ { type argDecl ; }+ ]
  ```

Lexical analysis

grep " [0-9][0-9]*[A-Z0-9\-] * PRODCODE" *
grep "MOVE *[A-Z0-9\-]*PRODCODE" *
perl -pi -w -e 's/SEN/SWAT/i;' *
if ($current_line =~ /(MOVE|SET|IF|...)/)
  {}

To summarise
To summarise

- Lexical analysis
- Hierarchical lexical analysis
- Iterative lexical analysis
- Fuzzy parsing
- Island grammars
- Robust parsing
- Bridge grammars
- Skeleton grammars
- Parsing in the cloud
- Error productions
- Hierarchical error repair
- Permissive grammars
- Panic mode
- Precise parsing

one scale of tolerance?
Stay tuned!

vadim@grammarware.net