Extensible C++ Parsing

Mike Kucera
Jason Montojo
IBM Eclipse CDT Team

 September 26, 2007

Motivation

- We want to properly support other dialects of C++
- C/C++ Language Extensions for Embedded Processors
  - ISO/IEC DTR 18037
- Vendor-specific C++ Extensions
  - GNU g++
  - Intel C++
  - Microsoft Visual C++
  - XL C++
- C++0x (extension of ISO C++)

Requirements

- Flexible
  - Support multiple dialects of C++
- Accurate
  - Useful error reporting
  - Refactoring

Requirements & LPG

- Flexible
  - ✓ LPG facilitates creation of parsers for related languages
- Accurate
  - ✗ LPG's backtracking LR algorithm cannot handle ambiguities in ISO C++ grammar

Ambiguities in C++

- Identifiers and type names are syntactically the same
- Lexers in CDT are not aware of types
- For example: `x * y;`
  - Binary multiplication of `x` and `y`?
  - Declaration of pointer of type `x` called `y`?
  - Cannot disambiguate unless we know what `x` is

Ambiguities in C++ (cont.)

- Two solutions:
  - Parse every alternative
    - Store each possibility within the AST
    - Analyze the tree afterwards to choose between conflicting alternatives
    - Currently used by CDT's GNU C/C++ parsers
  - Collect type information during the parse
    - Resolve ambiguities as they are encountered
    - Backtrack as necessary if we follow the wrong alternative
    - Currently implemented by next generation modular C99 parser (which will be the basis of modular C++ parser)

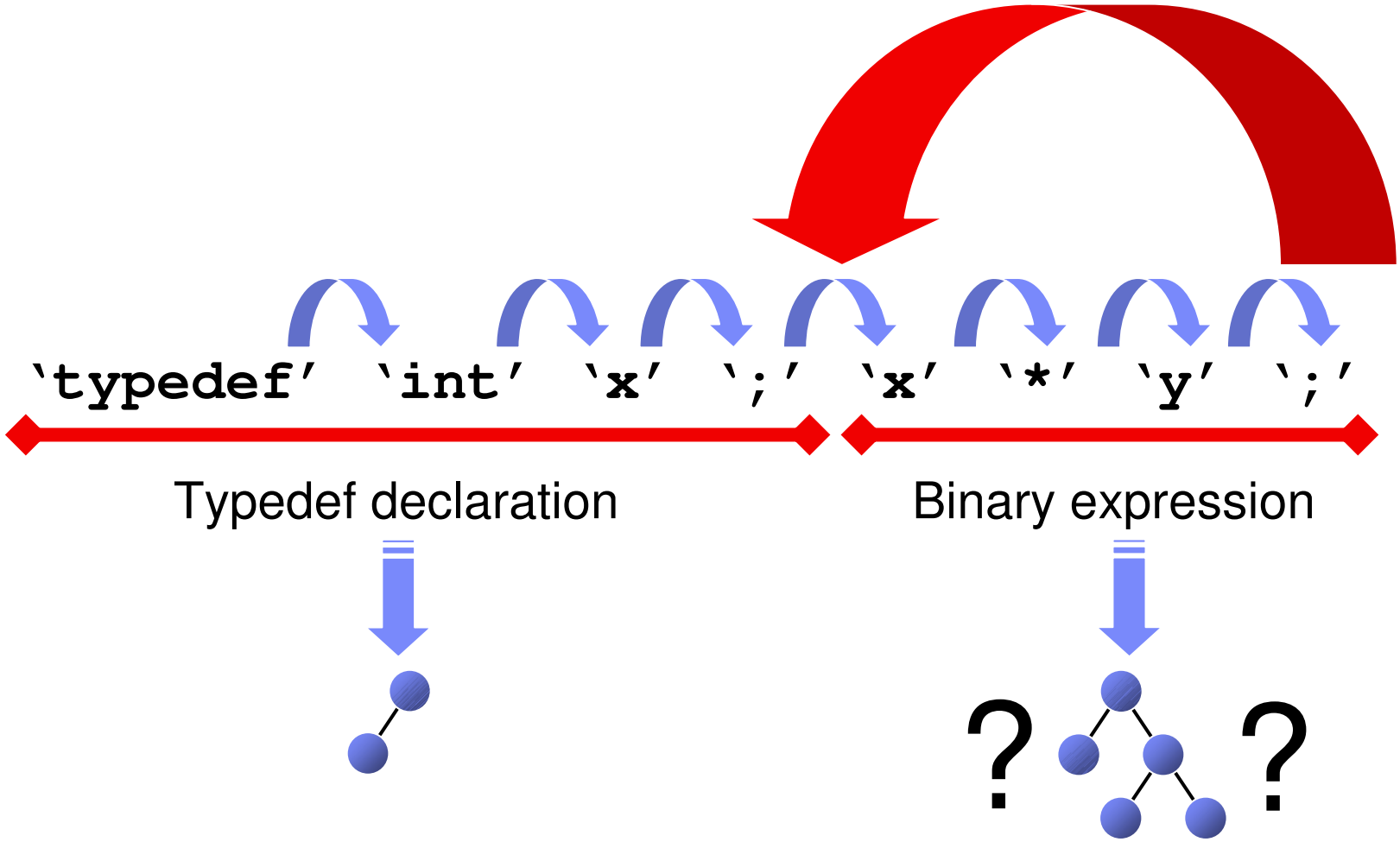
Solution #1: Parse Every Alternative

- Benefits:
  - Accurate – we can choose to keep the alternative that results in fewer errors
- Drawbacks:
  - AST contains more branches
  - Need to select among each set of alternatives in order to get a proper AST
    - Need to traverse AST to locate ambiguities

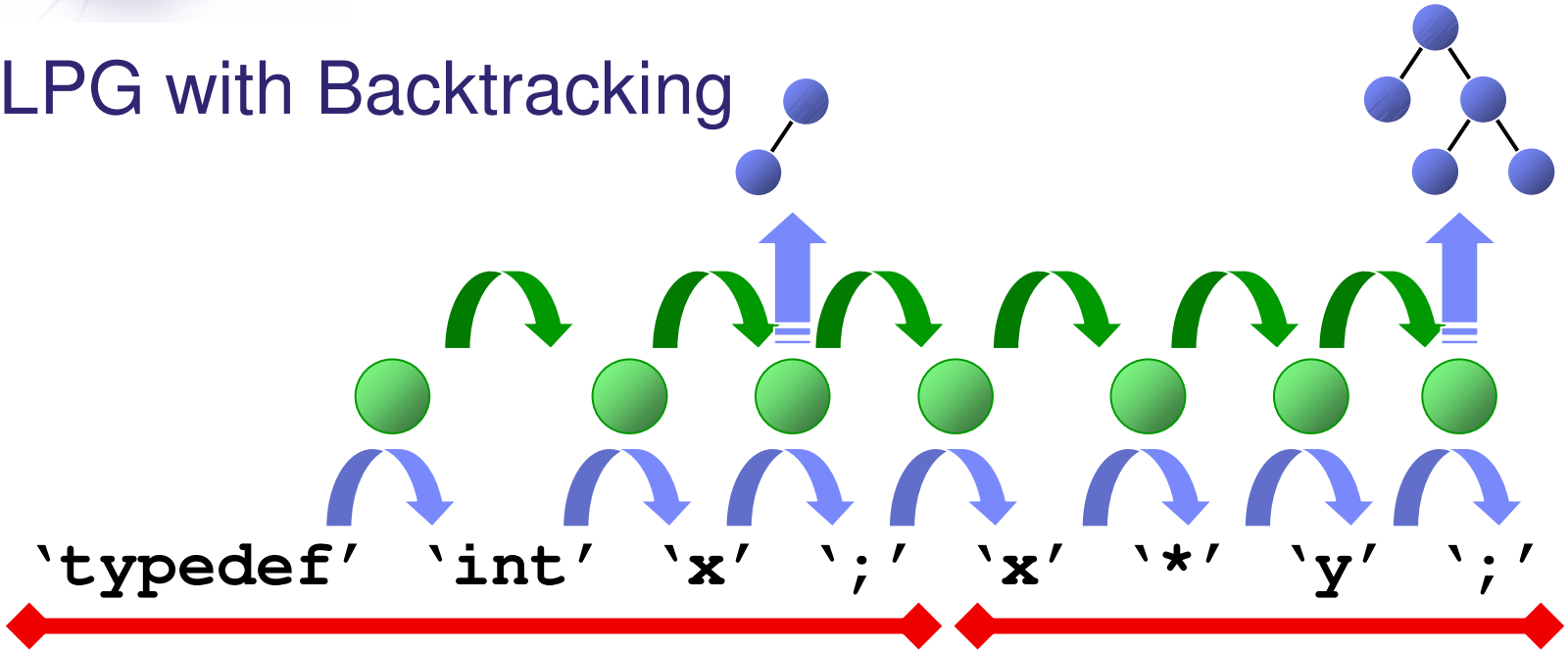
Solution #2: Collect Type Information

- Benefits:
  - See things from a compiler perspective
    - More accurate view of what errors exist in the code
  - Disambiguate during parsing
    - Saves time and space
- Drawbacks:
  - See things from a compiler perspective
    - Reported errors might sound confusing
  - Resulting parse may not be accurate if code is not correct enough

Parsing with LPG



LPG with Backtracking



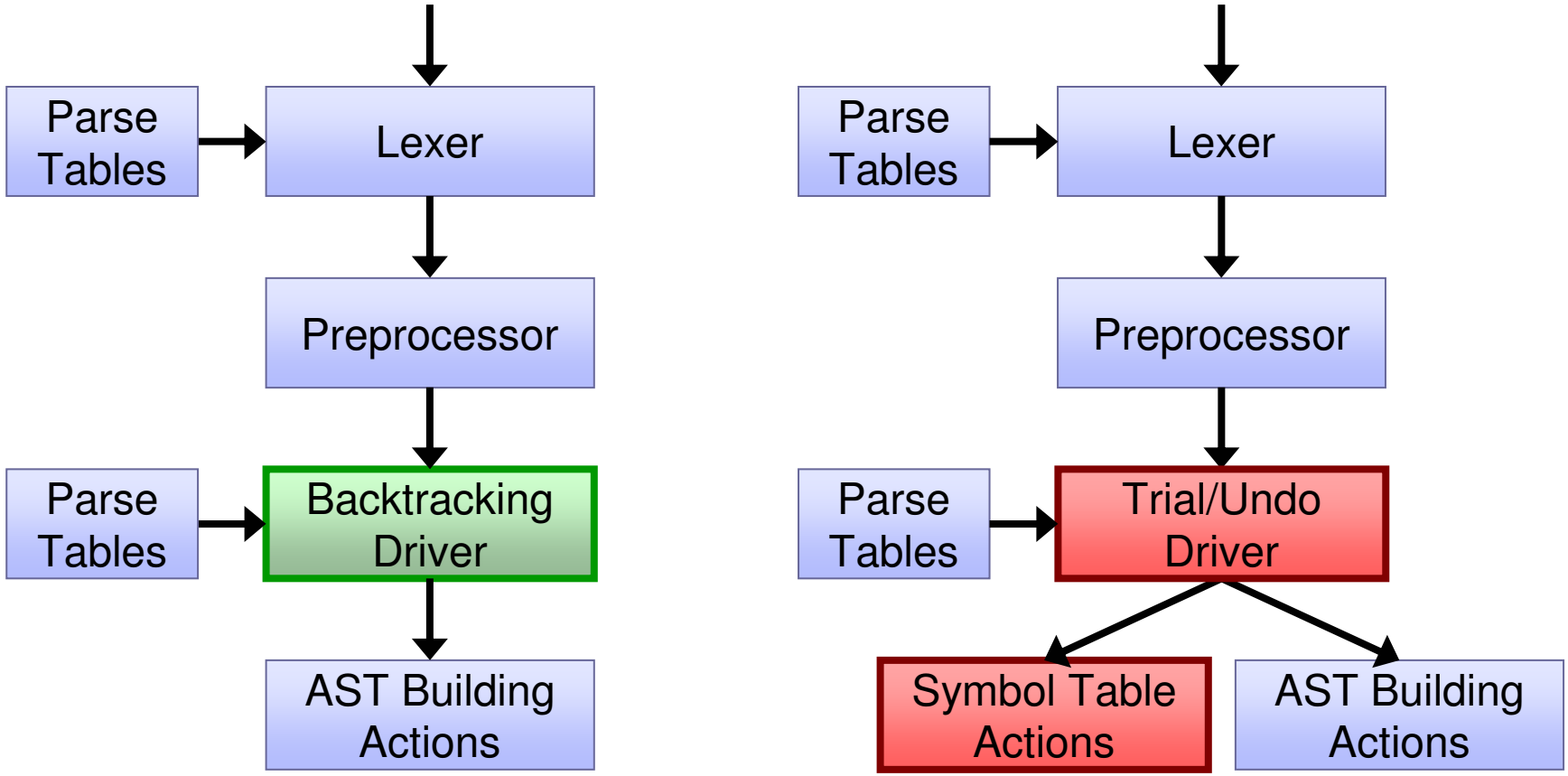
Problems with LPG

- Enabling backtracking means actions are not executed until the parse is complete
  - Performance issue since **all** actions are saved (more than 1 per token) until parsing is finished
  - We cannot use actions to maintain a symbol table during the parse
  - Without a symbol table, we cannot parse C++ accurately because of ambiguities in the grammar

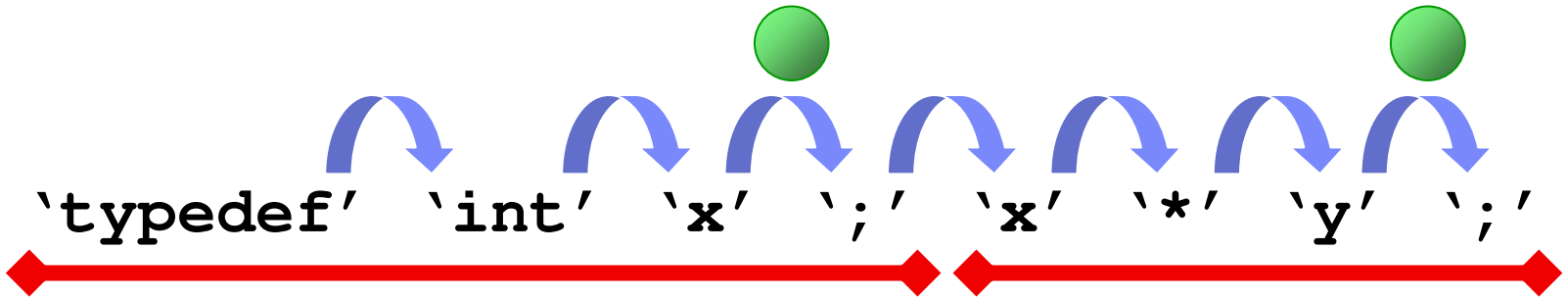
Solution: Specialized Actions

- Implemented a new LPG parser driver to support A Backtracking LR Algorithm for Parsing Ambiguous Context-Dependent Languages, A. Thurston & J. Cody
  - Trial Action
    - Adding entries to symbol table
    - Decide whether to backtrack or keep going
  - Undo Action
    - Removing entries from symbol table
  - Final Action
    - Building the AST

Architecture



Parsing: Trial Phase



Typedef declaration

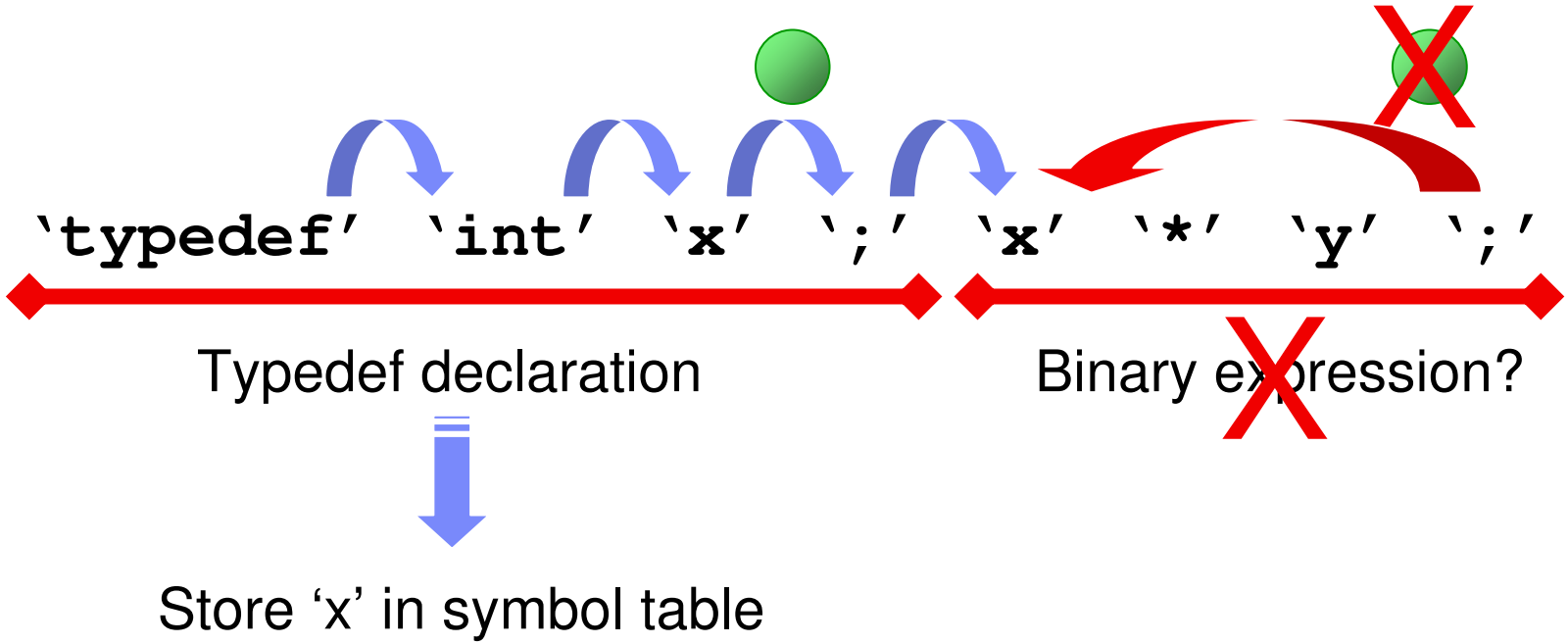
Store ‘x’ in symbol table

Binary expression?

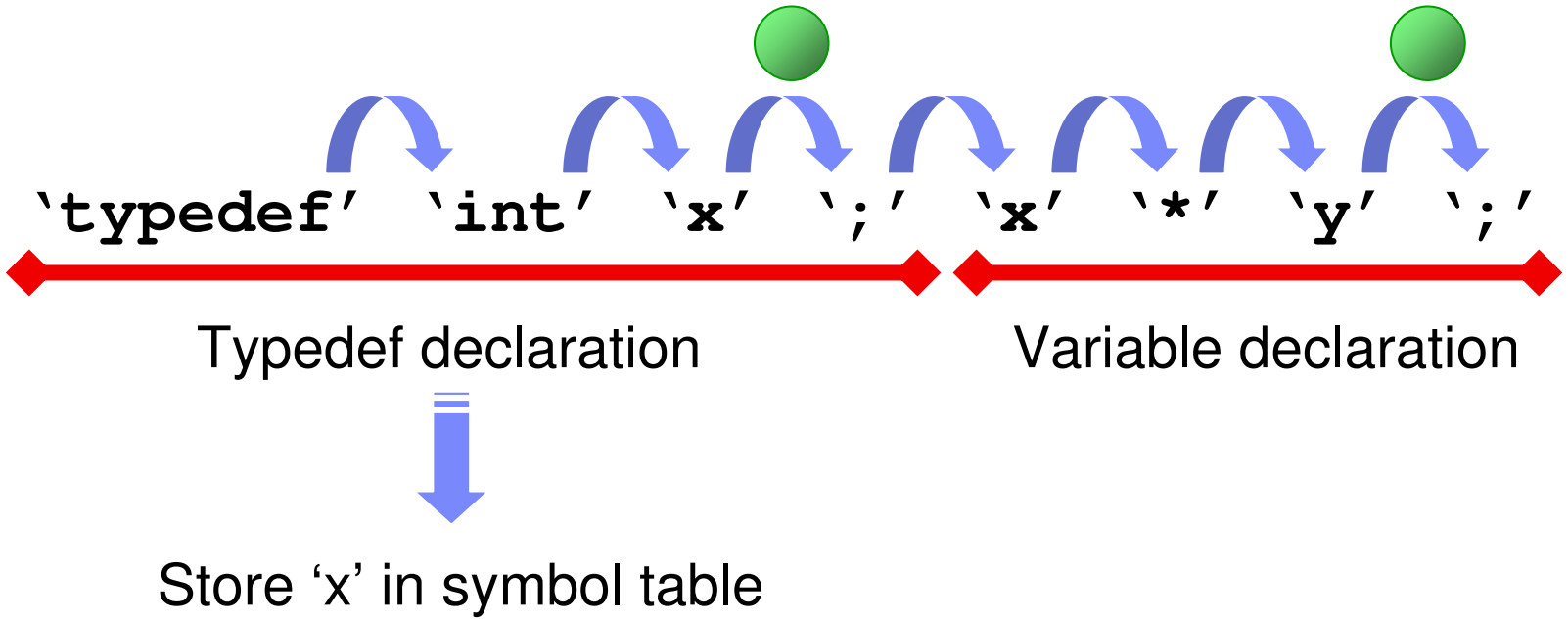
Is ‘x’ a type?

Yes, so this isn’t
a binary expression.
We should backtrack.

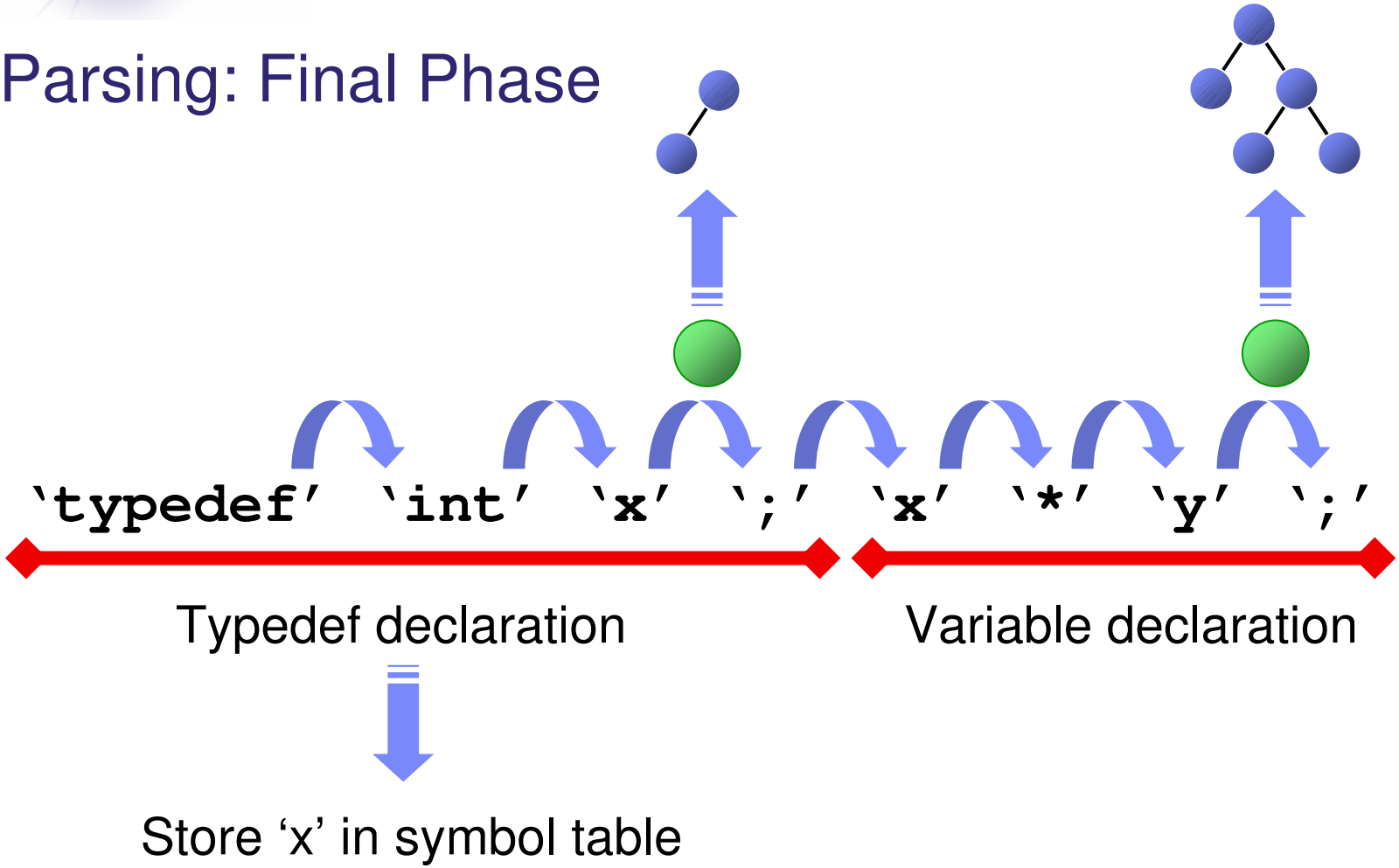
Parsing: Undo Phase



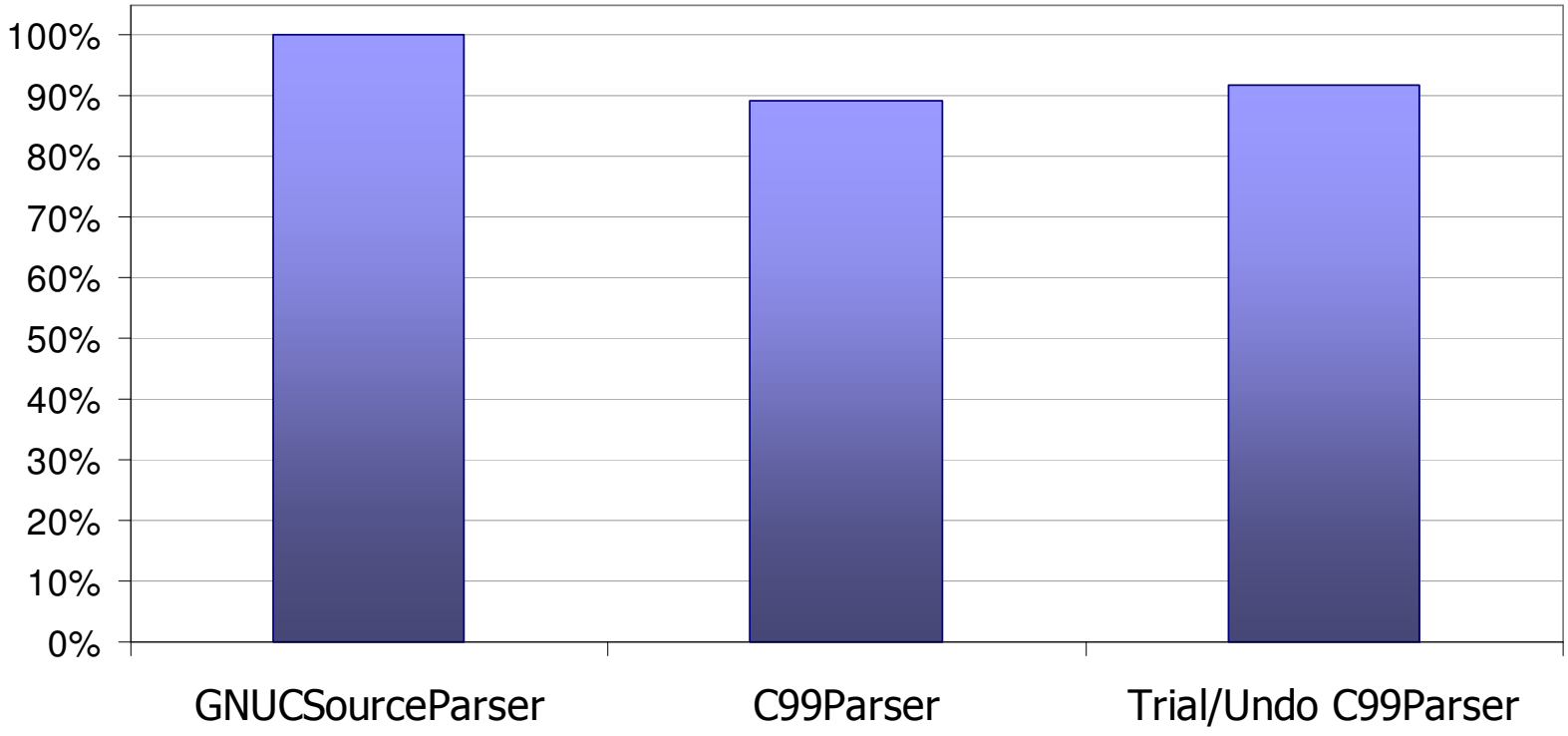
Parsing: Trial Phase #2



Parsing: Final Phase



Relative Parser Throughput



Questions?