Program Generation and Modification Using Multiple Domains

Ira D. Baxter

DMS Software Reengineering Toolkit

- *Automated* source code analysis and *modification*
  - *Leverage transformation machinery needed to build DMS vision*
- Enables wide variety of SE tasks to be automated
  - *Commercial applications*
    - Source Formatters, Hyperlinked Source Browsers
    - Documentation extraction
    - Metrics
    - Preprocessor conditional simplification
    - Test Coverage and Profiling tools
    - Clone Detection and removal
    - XML DTD compilation
    - DSL code generation: Factory Automation
    - Migrations (JOVIAL to C)
  - *Research applications*
    - Aspect-weaving (U. Alabama Birmingham)
    - Large-scale C++ component restructuring (SD/Boeing)   **OOPSLA DEMO**
    - Code generation/quality checking for spacecraft (NASA/JPL)

DMS Software Reengineering Toolkit = Generalized Compiler

- Underlying Hypergraph representation: trees, graphs, …
- Parsing/Prettyprinting
  - UNICODE lexer with binary conversions, lexical format/comment capture
  - GLR (context-free) parser with automatic tree builder
  - "Text Box" building language; reproduces comments!
- Analysis
  - Multipass attribute grammars
  - Generalized symbol table support: inheritance, overloading, …
- Transformation
  - Complete AST interface => procedural transforms (& analyzers)
  - Conditional Source to Source transforms w/ associative/commutative laws
- Predefined Domains
  - Specification, Technology, and Legacy languages

    Spectrum, .MDL   XML, SQL, IDL   C/C++, C#, Java, COBOL, *many more*…

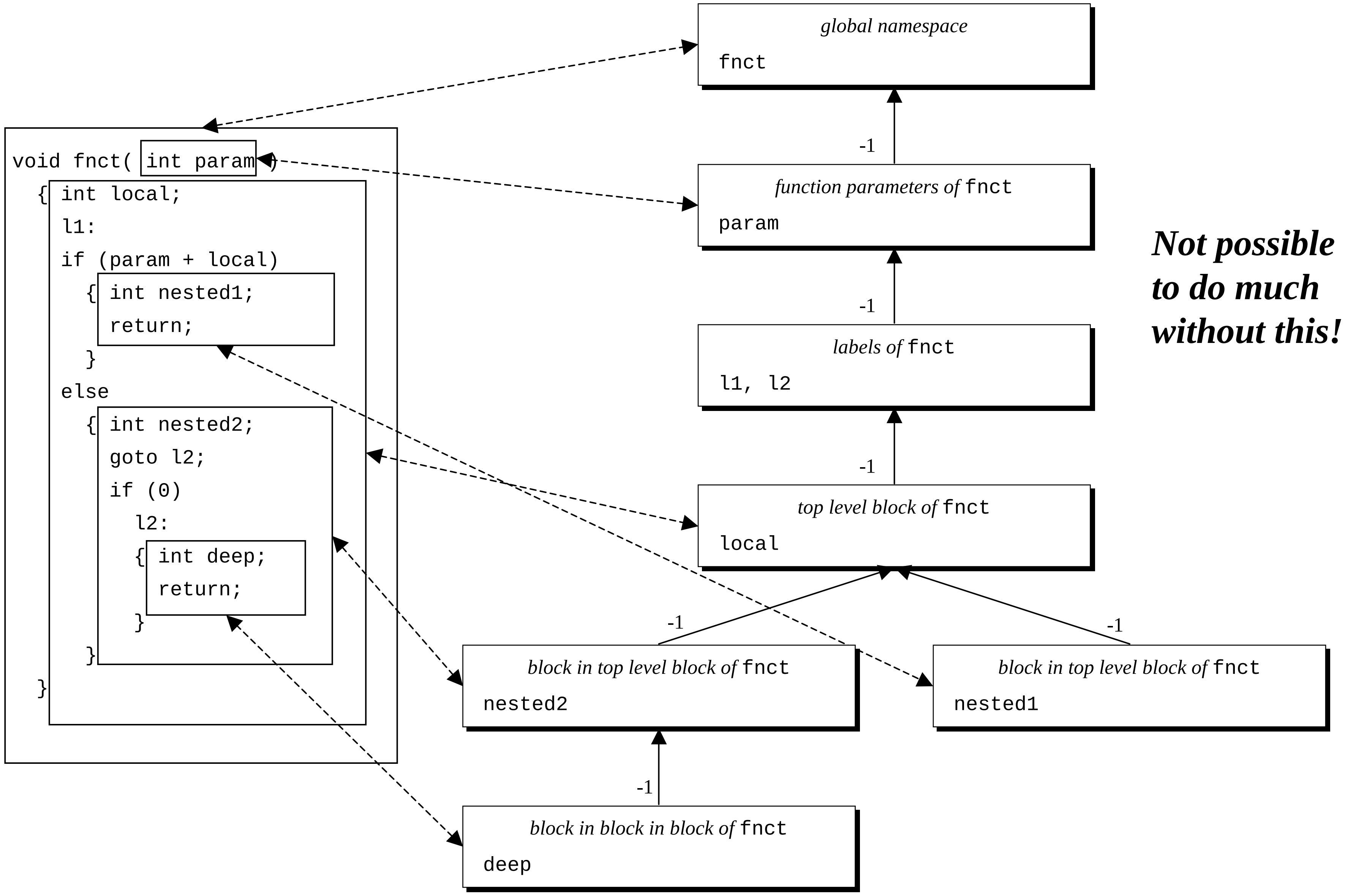
Knowledge Capture in Managable Chunks:

DMS Domain Parts

• Syntax	
– External Form	Lexical Specification and Grammar Rules
– Parser	GLR parser + custom code (preprocessor…)
– AntiParser	Formatting Rules + custom lexeme generators
• Semantics	
– Analyzers	Typically includes Name/Type resolution
– Optimizations	(Source to Source) transforms in domain
– Refinements	(Source to Source) transforms between domains

Draco "domain" paradigm… James Neighbors, 1978

DMS C++ Symbol Table for a function


```
void fnct( int param )
  { int local;
    l1:
    if (param + local)
      { int nested1;
        return;
      }
    else
      { int nested2;
        goto l2;
        if (0)
          l2:
          { int deep;
            return;
          }
      }
  }
```

global namespace
fnct
-1
function parameters of fnct
param
-1
labels of fnct
l1, l2
-1
top level block of fnct
local
-1
-1
block in top level block of fnct
nested2
block in top level block of fnct
nested1
-1
block in block in block of fnct
deep

Not possible to do much without this!

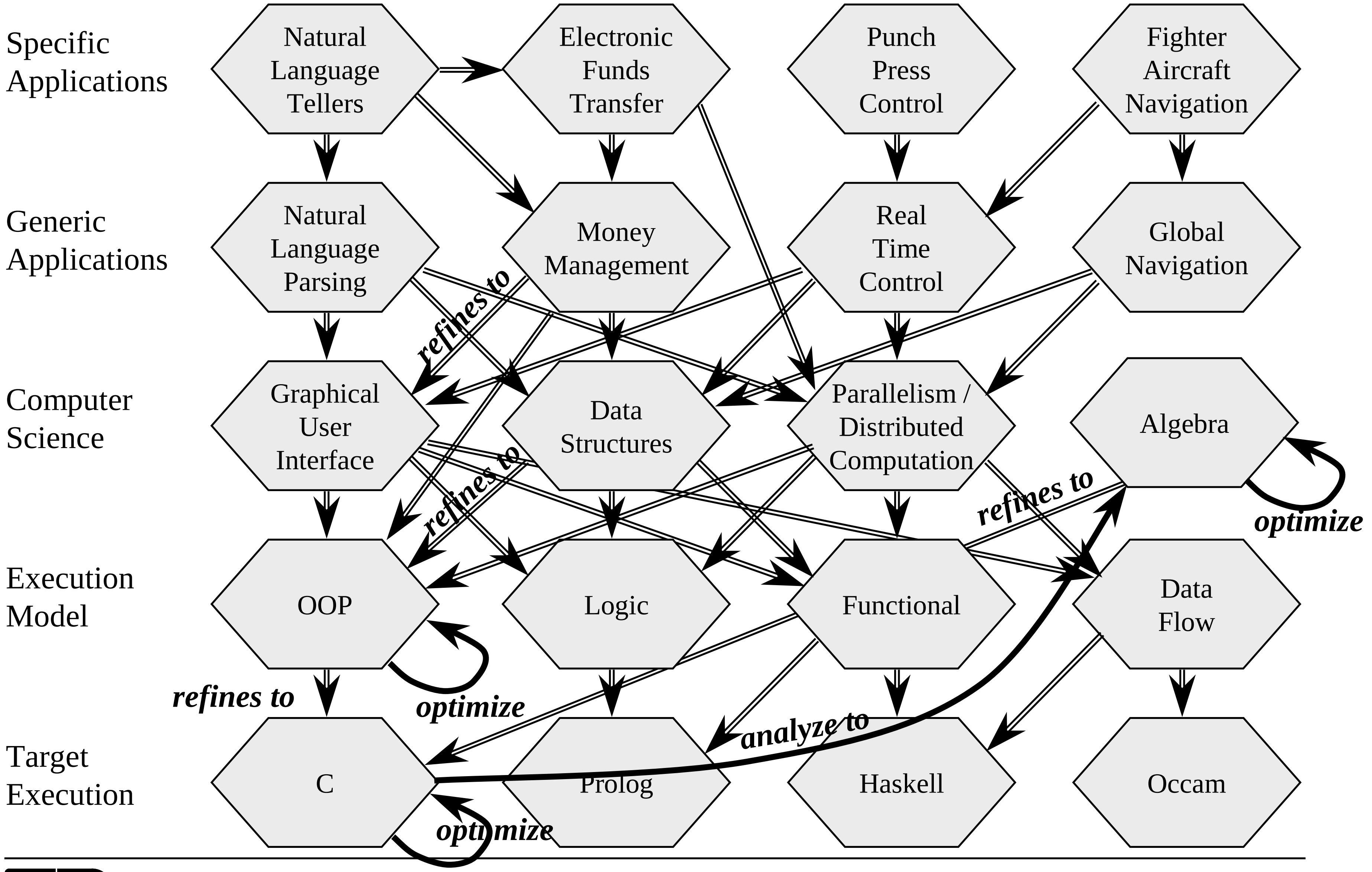
How to represent multiple domains?

- Union grammar?   … for C and C#
  - Combine rules of two grammars

    program_element$_C$ **= function_declaration**
    expression$_C$ **= identifier '[' exp ']'**
    program_element$_{C\#}$ **= class_declaration**
    expression$_{C\#}$ **= identifier '[' exp ']'**

  - Simplifies problem of writing transformation rules
    - Only one "syntax"
  - Makes separating semantics more difficult
    - What if **`a[i]`** has different index origin?
- Separate grammars!

A Domain Interconnection Network


Specific Applications
Generic Applications
Computer Science
Execution Model
Target Execution
Natural Language Tellers
Electronic Funds Transfer
Punch Press Control
Fighter Aircraft Navigation
Natural Language Parsing
Money Management
Real Time Control
Global Navigation
Graphical User Interface
Data Structures
Parallelism / Distributed Computation
Algebra
OOP
Logic
Functional
Data Flow
C
Prolog
Haskell
Occam
refines to
refines to
refines to
refines to
optimize
optimize
optimize
analyze to