Introduction

- C/C++ Preprocessor – useful and widely used tool
- The compiler gets the preprocessed code and not the original source code that the programmer sees
  - in many cases the two codes are markedly different
  - problems in program comprehension, analysis and maintenance
- The preprocessor is like a black box
  - the connection between its input and output is well-defined, but in concrete cases it may be hard to see precisely what is going on
Directive usage

- Unix software packages*
  - Directives make up 8.4% of the source code lines
  - Dependencies: each line depends on 0.59 macros in average
  - Extreme case
    - one line in GCC source depends on 41 different macros
    - considering all configurations this increases to 187

Schema for preprocessing

- **Schema**
  - describes the source code from a preprocessing point of view
  - object oriented model of preprocessor related language elements and their relationships

- **Schema instance**
  - an embodiment of the schema in a concrete case
  - a graph that describes the connection between the original and the preprocessed code (the procedure of preprocessing)
Schema instance

\#define B(x) x+A
\#define A 1
B(3) → 3+1
Configurations

- Due to conditional blocks several versions of the code exist in the source simultaneously

Configuration

- code processed during one particular run of the preprocessor
- determined by the context of preprocessing: predefined macros and command line (macros, include paths, etc.)
Schema Instance Types

- Dynamic instance – configuration-dependent
  - follows the normal run of the preprocessor
  - conditionally excluded code is omitted, separate file nodes, a macro call has exactly one definition

- Static instance – configuration-independent
  - follows the code structure
  - all conditional blocks are included, macros may have more definitions (from different configurations)

- The schema describes both types of instances
Dynamic and static instance

main1.c:
--------------------------------------
#include "config.h"
...
define MACRO
...
#include "support.h"
...

support.h:
--------------------------------------
#include "config.h"

config.h:
--------------------------------------
#ifdef MACRO...

Dynamic

Static
Columbus Schema for C/C++ Preprocessing
Implementation

- Command line tool **CANPP**
- Builds Schema Instances
  - dynamic instances
  - (static instances will be available soon)
  - outputs: PPML (XML), GXL, binary
  - API
- Works also as a usual preprocessor
  - can be incorporated into existing build processes
  - supports MSVC and GCC dialects
- Part of the Columbus framework
Possible applications

- Schema is for general use
  - may aid the work of researchers
    - e.g. processing XML instead of implementing a preprocessor or simply using the API
- Dynamic – particularly for programmers, e.g.:
  - visualization
- Static – overall program comprehension, e.g.:
  - include hierarchy
  - macro-related metrics
  - revealing conditions for a given line to be compiled
Summary

- Our technique fills the gap between the original and the preprocessed code
  - Solves problems with comprehension and analyzer tools
- Schema, schema instances
  - Dynamic – helps in concrete cases, configurations
  - Static – overall program comprehension
- Part of the Columbus framework
  - XML, GXL