The Meta-Environment

A Language Workbench for Source code Analysis, Visualization and Transformation

Jurgen Vinju
IPA herfstdagen 2008
Overview

- Goal of The Meta-Environment
- Methods of The Meta-Environment
- Application in Source Code Analysis
- Discussion on quality throughout the talk
Meta

- Analysis & Transformations tools are complex:
  - details of languages are in large quantities
  - details of analysis and transformation are complex
  - requirements of application area

- Meta tools to the rescue
  - factor out the “boilerplate”
  - provide reusable design & implementations
  - provide common jargon

- Engineering trade-offs
Programming environment tools

**Tools**
- Compilers
- Code generators
- Static checkers
- Refactoring
- Outlines
- Cross-linking
- Decompilers
- etc, etc, ...

**Fields**
- Compiler construction
- Model Driven Eng.
- Formal methods
- Reverse engineering
- Re-engineering
- Domain Specific Lang.
- Aspect Oriented Prog.
- Software Maintenance

Goals of The Meta-Environment
Source Code Representations

- Source code juggling tools
  - all instances of meta programming
- Three main levels of representation
  - Tools are all mappings between representations
- The Meta-Environment uses this similarity
  - representations have languages
  - mappings between languages share functionality
Meta Programming Overview

- Meta programming is complex
  - Many different (legacy) languages
  - “Devil in the details” of syntax and (static) semantics
  - Tools are (company/application/domain) specific
- Meta-Environment goal: facilitate meta programming
  - *generic* tools (reusable across languages)
  - *generated* tools (language and task parametric)
Examples of tools

<table>
<thead>
<tr>
<th>Generated tools</th>
<th>Generic tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Parsers</td>
<td>• Syntax highlighters</td>
</tr>
<tr>
<td>• Pretty printers</td>
<td>• Structure editors</td>
</tr>
<tr>
<td>• Tree transformers</td>
<td>• Source browsers</td>
</tr>
<tr>
<td>• Tree fact extractors</td>
<td>• Fact visualizers</td>
</tr>
<tr>
<td>• Fact analyzers</td>
<td>• Debugging interfaces</td>
</tr>
<tr>
<td></td>
<td>• Un-parsers</td>
</tr>
</tbody>
</table>

Goals of The Meta-Environment
Challenges

- Meta programming is not compiler construction
  - Why? Different quality attributes!
- Language and task independent
  - No heuristics (example: `longest match' in scanners)
  - No abstraction (example: keep layout and comments)
- More flexible
  - Modular (example: embeddings, dialects)
  - Reusable (example: several tasks share a parser)
- And so the challenges are
  - Finding the appropriate level of abstraction
  - Getting enough efficiency without heuristics or abstraction
The Meta-Environment

- Since early 90's: generation of IDE components
- Generated and generic meta programming tools
- Today: focus on analysis tools
  - Parsing
  - Analysis
  - Visualization
The Meta-Environment

- A domain specific language for each task
  - SDF: parser generation
  - ASF: tree transformation
  - BOX: formatting
  - RScript: reasoning
  - Prefuse: visualization

- Meta-environment: generic (parameterized) IDE
Some applications

- COBOL analysis and transformation (*CALCE*)
  - goto removal, type recovery, object recovery, dead code detection
  - documentation generation
- Java analysis and transformation
  - code smell detection, dead code detection, model extraction, proof obligation extractors, ...
  - control flow visualization, ...
- C analysis and transformation (*Ideals*)
  - comment convention analysis, aspect removal, aspect weaving, ...
### Reusing The Meta-Environment

<table>
<thead>
<tr>
<th>System</th>
<th>Type</th>
<th>ATerms</th>
<th>SDF</th>
<th>ASF</th>
<th>Meta</th>
<th>ToolBus</th>
<th>BOX</th>
<th>TIDE</th>
<th>ApiGen</th>
<th>ASF+SDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASF+SDF</td>
<td>Trafo</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Stratego</td>
<td>Trafo</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Action Not.</td>
<td>DSL</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chi</td>
<td>Model</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>ELAN</td>
<td>M+P</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu$CRL</td>
<td>Model</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>FERUS</td>
<td>Trafo</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>haRVey</td>
<td>Proving</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Lucent</td>
<td>Trafo</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FirstResult</td>
<td>Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>TLA+</td>
<td>Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIG</td>
<td>Analysis</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>TOM</td>
<td>Trafo</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>CALCE</td>
<td>A+T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Examples of reuse of generic tools and tools generators by other meta programming systems
SDF for parser generation

- Parser generation from grammars
  - Declarative \textit{no side-effects, or heuristics}
  - Existing programming languages
  - Modular \textit{composable}

- Features of SDF:
  - integrated lexical and context-free syntax
  - accepts all context-free languages (not just LR(k) or LALR)
  - reusable modules via renaming and parameterization
  - declarative disambiguations
  - fully informative parse trees
SDF - implementation

- SGLR implements SDF
  - Scannerless parsing (no more meaningless tokens)
  - Generalized (unbounded lookahead without backtracking)
  - Filtering (removal of ambiguous derivations)
  - Fully informative parse forests
  - Makes genericity possible by avoiding heuristics
SDF - Example

Disambiguation by associativity

Reuse of a library module

Disambiguation by longest match

lexical restrictions
Id -/- [A-Za-z0-9\_]

context-free syntax
Id
Exp Exp
“\"” {Id “,”}+ Exp
“(“ Exp “)”

Ambiguous expression
language

Methods of The Meta-Environment
ASF for Tree transformations

- ASF - describing syntax tree transformation
  - Functional and Algebraic Programming

- Features of ASF:
  - Conditional Term Rewriting (match and replace)
  - List matching (matching mod associativity)
  - Concrete syntax (patterns in source language)
  - Traversal functions (automated traversal)
  - HiFi (retain layout and comments)
  - Fully typed (i/o is syntactically correct)
ASF - implementation

- Concrete syntax
  - Using SDF to generate parsers for ASF (ASF+SDF)
- Running ASF
  - Eager (innermost) evaluation
  - Matching and construction of parse forests
  - No (a priori) abstraction
  - Maximal sub-term sharing (equality in O(1))
- Compilation to C
- ASF: *tree manipulation with user-defined syntax*
Step 1: An SDF grammar is used to generate two parsers: one for the input language, one for the ASF module.

Step 2: Both the source code and the ASF module are parsed to obtain parse trees.

Step 3: The ASF rewrite engine transforms the input parse tree to an output parse tree, by applying the parsed ASF module.

Step 4: The target code is extracted from the parse tree, optionally preceded by a phase that adds brackets where needed.
ASF+SDF - Example

context-free syntax

List matching, a shorthand for reaching any element or sublist of a (syntactic) list

"Id+"[0-9] -> {Id ","}+
"Exp"[0-9] -> Exp

equations

[extract]
Id,Id Exp = Id

ASF is an extension of SDF
First define meta variables to be able to write patterns

Using concrete syntax to match a patterns of our expression language

First define meta variables to be able to write patterns
Rscript for analysis

- Explained by Paul Klint on Monday
- Relational Calculus
- Integration with ASF+SDF
- Source Code Locations
- Type system
  - Relations, sets, ...
  - Structural type-equivalence
- Integration with visualizations via type system
Visualizations with Prefuse

- http://www.prefuse.org
- Third-party open-source library for visualization and animation in Java
  - *Table driven*
  - Fast
  - Easy
The Meta Fact Browser

- Plugin architecture for Java/Prefuse-based visualizations of relations
- Automatic matching of applicable visualizations to extracted data
- Links to source code
- Simple plugins so far
- Experimenting with more advanced visualizations and interactions
Analysis to Visualization via Types

- Types drive visualization selection, e.g:
  - rel[str,int] -> bar chart, pie chart, etc:
  - set[MethodName,loc]
  - rel[int,int,rgb] -> scatterplot

- Analysis works towards visualizable types

- Many (non)sensible combinations possible
  - Open experimentation
  - Harms understanding for newbies
Comment Convention Analysis

• Company specific C code and idioms
• Comment idiom for in/output parameters

```c
int functionName(char *arg1, int arg2, MyType **arg3)
{ /* Body of function */
    /* Input(s) : arg1
    *       my first argument
    *    arg2
    *       this is an integer input
    * Output(s) : arg3
    *           should be an output parameter */
    /* Body of function */
}
```

Does the code implement the declared behavior?
• Need a C grammar that can parse the C dialect of the company, (including pre-preprocessor stuff)
• Need a grammar for the company-specific comment convention
• Need to extract model of the declared in/output behavior
• Need to extract model of the control and data flow of the program
• Need an analysis that correlates the two models
• Need a visualization to present the results
C grammar with pre-processor

- ANSI-C grammar from standard
- Extend with Non-C macro uses
- Import parameterized CPP grammar
- Test on large code base
Comment Idiom Definition

lexical syntax

IO-Comment -> LAYOUT {prefer}
"/" {WS "*/"}* Definition+ "*/" -> IO-Comment

Inputs | Outputs | InOuts | Returns | Purpose | Notes -> Definition

"Input(s)" WS ":" WS Params -> Inputs
"Output(s)" WS ":" WS Params -> Outputs
"InOut(s)" WS ":" WS Params -> InOuts
"Returns" WS ":" Description -> Returns
"Purpose" WS "::" Description -> Purpose
"Notes" WS "::" Description -> Notes

Param Line (\[ \] \[\t\] \[\t\] Param Line)* -> Params
Param \{WS "*"\}]* -> Params

[A-Za-z\_\-][A-Za-z0-9\_\-]* -> Param {category("Identifier")}
"<none>" -> Param

WS "*/" \[\t\] \[\t\] \[\t\] Description -> Line
(Word | WS)* "*/" -> Line

[\t \n]+ -> WS
~[\t\n]+ -> Word
[\*] -> Star

int functionName(char *arg1, int arg2, MyType **arg3)
/* Input(s) : arg1
   * my first argument
   * arg2
   * this is an integer input
   * Output(s) : arg3
   * should be an output parameter
   */
{
   /* Body of function */
}
Auto highlighting

```c
int VSCV_unflatten_type_string(
    const char *type_string,
    const int length,
    VSCV_type_description *desc,
    **remaining_type_string)
/* Input(s) : 
*     type_string
*     string containing a type description in Labview
*     format
*     length
*     length of type_string (in bytes)
* Output(s) : 
*     desc
*     parse result of type_string
*     remaining_type_string
*     pointer to first byte not parsed in type_string
*     NOTE: if complete type_string was parsed, this variable
*     points to type_string+length and need not be
*     valid for dereferencing use!
*     NOTE: set parameter to NULL if not interested in
*     this value
* InOut(s) : <none>
* Returns : error code
*          : OK: Function succesful
*          : VSCA_PARAMETER_ERR: Parameter error
*          : VSCA_ENCODING_ERR: Syntax/semantical error in type_string
*          : VSCA_MEMORY_ERR: Memory allocation error
* Purpose : Parse a type_string in Labview format into a structured
*          : type description in internal format
* Notes : This function mimicks the Labview API for type_to_string
*/
{
    char *func_name = "VSCV_unflatten_type_string";
    int r = OK;

    char *curr_ptr = (char *)NULL; /* pointer into type_string */
    char *ret_curr_ptr = (char *)NULL; /* pointer into type_string */
    char *end_ptr = (char *)NULL; /* pointer into type_string */
}
```
Auxiliary extraction functions

Concrete pattern in C with embedded comment

Visitor/Traversal function that searches for all C function declarations
A bunch of tuples that relate function names to parameter names

"string_length" > , < "VSTR_translate_rp_get_subsystem_state" , "string" > , < "VSTR_translate_rp_get_system_status" , "string_length" > , < "VSTR_translate_rp_get_system_status" , "string" > , < "VSTR_translate_rp_set_loadlock_state" , "string_length" > , < "VSTR_translate_rp_release_access" , "string_length" > , < "VSTR_translate_rp_request_access" , "string_length" > , < "VSTR_translate_rq_get_subsystem_state" , "subsystem" > , < "VSTR_translate_rq_get_system_status" , "<none>" > , < "VSTR_translate_rq_get_system_status" , "string" > , < "VSTR_translate_rq_set_loadlock_state" , "string_length" > , < "VSTR_translate_rq_set_loadlock_state" , "string" > , < "VSTR_translate_rq_release_access" , "string_length" > , < "VSTR_translate_rq_release_access" , "gate" > , < "VSTR_translate_rq_function_header" , "string_length" > , < "VSTR_translate_rq_function_header" , "string" > , < "VSTR_translate_rq_request_access" , "gate" > , < "VSTR_initialise" , "<none>" > } > , < ComputedInputs , rel [ str , str ] , { < "handle_communication_timeout" , "EH_handle" > , < "handle_communication_timeout" , "context_p" > , < "VS_rq_get_sim_mode" , "data_mode" > , < "VS_rq_get_sim_mode" , "trace_mode" > , < "VS_rq_get_sim_mode" , "sim_mode" > , < "VSrq_get_sim_mode" , "object_p" > , < "VSSI_simulate_set_loadlock_state" , "loadlock_state" > , < "VSSI_simulate_set_loadlock_state" , "loadlock" > , < "get_lv_server_index" , "id" > , < "VSTR_translate_rq_set_loadlock_state" , "loadlock_state" > , < "VSTR_translate_rq_set_loadlock_state" , "loadlock" > , < "V_SIrq_mc_changed" , "object_p" > , < "VSTR_translate_rq_thxa_trace" , "string_length" > , < "VSGN_printable_byte_string" , "string_length" > , < "VSGN_printable_byte_string" , "string" > , < "VSIT_sim_mode" , "mode" > , < "VSTM_LV_server_response" , "rp_origin" > , < "VSTM_LV_server_response" , "rp_string_length" > , < "VSQU_queue_add_front" , "item_data" > , < "free_type_description_r" , "free_desc" > , < "free_type_description_r" , "desc" > , < "get_or_make_ts_client_index" , "id" > , < "get_or_make_ts_client_index" , "object_p" > , < "VSGN_put_type_desc" , "fp" > , < "VSGN_put_type_desc" , "desc" > , < "VSGN_put_short_string" , "string" > , < "VS_class_rp_set_loadlock_state" , "error_code" > , < "VS_class_rp_set_loadlock_state" , "repl_addr" > , < "VS_class_rp_set_loadlock_state" , "object_p" > , < "VSGN_s32_to_u32" , "si" > , < "queue_extract" , "delete_item" > , < "queue_extract" , "front" > , < "VSTR_translate_rq_enxa_log" , "string_length" > , < "VS_rq_rd_imp_data" , "object_p" > , < "VSTS_release_access" , "rp_addr" > , < "VSTS_release_access" , "object_p" > , < "VSTS_release_access" , "gate" > , < "VSGN_put_longs" , "id" > , < "VSQU_iterator.peek" , "iterator" > , < "VSCN_ReadMsg" , "len" > , < "VSCN_ReadMsg" , "read_fd" > , < "VS_rq_set_sim_mode" , "data_mode" > , < "VS_rq_set_sim_mode" , "trace_mode" > , < "VS_rq_set_sim_mode" , "sim_no"
Extract Actual In/output parameters

- Control and data flow analysis
- Track assignments to pointers
- Inter procedural (!)
- We used CodeSurfer
  - Building a control and data flow model takes time
  - We didn’t have time
- Output format of CodeSurfer transformed to relations
Analyze

- Use relational calculus to compute a report
- Differences and overlappings
  - Plain old union, intersection, difference
- Causality of differences
  - Comments are wrong
  - Code is wrong
  - Comment extraction is wrong
  - Model extraction is wrong
- 6 iterations in fixing extraction in half a day
Rscript Analysis

\[
\text{rel}[\text{str}, \text{str}] \ RealInputComments = \text{rangeX}(\text{InputComments}, \{ "<\text{none}>" \})
\]
...
\[
\text{set}[\text{str}] \ \text{VoidDeclarations} = \text{domain}(\text{rangeR}(\text{DeclarationArity}, \{ 0 \}))
\]
...
\[
\text{set}[\text{str}] \ \text{DeclarationsToBeChecked} = \text{domain}(\text{CommentedOrUncommentedDeclarations} \ \backslash \ \text{VoidDeclarations}
\]

\[
\text{set}[\text{str}] \ \text{surferDeclarationsNotFoundByASF} = \text{ComputedDeclarations} \ \backslash \ \text{DeclarationsToBeChecked}
\]

\[
\text{set}[\text{str}] \ \text{ASFDeclarationsNotFoundBySurfer} = \text{DeclarationsToBeChecked} \ \backslash \ \text{ComputedDeclarations}
\]

\[
\text{rel}[\text{str}, \text{str}] \ \text{inputsCommentedNotComputed} = \text{diff}(\text{RealInputComments}, \text{ComputedInputs})
\]

\[
\text{rel}[\text{str}, \text{str}] \ \text{InventedInputsAreOutputs} = \text{inputsCommentedNotComputed} \ \text{inter} \ \text{ComputedOutputs}
\]
### Applications of The Meta-Environment

**Visualization**

<table>
<thead>
<tr>
<th>str [0]</th>
<th>loc [1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle_communication_socket</td>
<td>area-in-file('terms/asml/VSLV.c',area(448,12,448,39,10840,27))</td>
</tr>
<tr>
<td>set_identification_from_context</td>
<td>area-in-file('terms/asml/VSLV.c',area(395,12,395,43,9809,31))</td>
</tr>
<tr>
<td>VSLV_send_request</td>
<td>area-in-file('terms/asml/VSLV.c',area(291,4,291,21,7600,17))</td>
</tr>
<tr>
<td>VSLV_close_connections</td>
<td>area-in-file('terms/asml/VSLV.c',area(231,4,231,63,6345,22))</td>
</tr>
<tr>
<td>VSLV_open_connections</td>
<td>area-in-file('terms/asml/VSLV.c',area(167,4,167,25,5251,21))</td>
</tr>
<tr>
<td>make_transition</td>
<td>area-in-file('terms/asml/VSSL.c',area(896,11,896,26,26254,15))</td>
</tr>
<tr>
<td>send_error</td>
<td>area-in-file('terms/asml/VSTM.c',area(1609,11,1609,21,33857,10))</td>
</tr>
<tr>
<td>send_reply</td>
<td>area-in-file('terms/asml/VSTM.c',area(1481,11,1481,21,31344,10))</td>
</tr>
<tr>
<td>send_request</td>
<td>area-in-file('terms/asml/VSTM.c',area(1390,11,1390,23,29631,12))</td>
</tr>
<tr>
<td>handle_traffic</td>
<td>area-in-file('terms/asml/VSTM.c',area(1271,11,1271,25,27084,14))</td>
</tr>
<tr>
<td>make_lv_server_index</td>
<td>area-in-file('terms/asml/VSTM.c',area(1164,11,1164,31,24867,20))</td>
</tr>
<tr>
<td>get_lv_server_index</td>
<td>area-in-file('terms/asml/VSTM.c',area(1103,11,1103,30,23829,19))</td>
</tr>
<tr>
<td>get_or_make_ts_client_index</td>
<td>area-in-file('terms/asml/VSTM.c',area(966,11,966,38,21083,27))</td>
</tr>
<tr>
<td>VSTM_get_status</td>
<td>area-in-file('terms/asml/VSTM.c',area(794,4,794,19,17660,15))</td>
</tr>
<tr>
<td>VSTM雒-client_request</td>
<td>area-in-file('terms/asml/VSTM.c',area(754,4,754,26,16808,22))</td>
</tr>
<tr>
<td>VSTM雒-server_disconnect</td>
<td>area-in-file('terms/asml/VSTM.c',area(623,4,623,29,14314,25))</td>
</tr>
<tr>
<td>VSTM雒-server_connect</td>
<td>area-in-file('terms/asml/VSTM.c',area(677,4,577,26,13417,22))</td>
</tr>
<tr>
<td>VSTM雒-timeout</td>
<td>area-in-file('terms/asml/VSTM.c',area(451,4,451,19,10900,15))</td>
</tr>
<tr>
<td>VSTM雒-server_response</td>
<td>area-in-file('terms/asml/VSTM.c',area(327,4,327,27,8408,23))</td>
</tr>
<tr>
<td>VSTM雒-client_request</td>
<td>area-in-file('terms/asml/VSTM.c',area(162,4,162,26,4923,22))</td>
</tr>
<tr>
<td>VSTS_request</td>
<td>area-in-file('terms/asml/VSTS.c',area(989,4,989,16,9666,12))</td>
</tr>
<tr>
<td>VSTS_error</td>
<td>area-in-file('terms/asml/VSTS.c',area(891,4,891,14,17879,10))</td>
</tr>
<tr>
<td>VSTS雒-reply</td>
<td>area-in-file('terms/asml/VSTS.c',area(640,4,640,14,13059,10))</td>
</tr>
<tr>
<td>VSTS雒-get_subsystem_state</td>
<td>area-in-file('terms/asml/VSTS.c',area(530,4,530,28,11149,24))</td>
</tr>
<tr>
<td>VSTS雒-get_system_status</td>
<td>area-in-file('terms/asml/VSTS.c',area(425,4,425,26,9376,22))</td>
</tr>
</tbody>
</table>

- Simple clickable list
- Source code inspection
## Results

<table>
<thead>
<tr>
<th>Basic facts</th>
<th>Comm</th>
<th>Coded</th>
<th>Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>—</td>
<td>20994</td>
<td>—</td>
</tr>
<tr>
<td>Functions</td>
<td>92</td>
<td>173</td>
<td>53%</td>
</tr>
<tr>
<td>void functions</td>
<td>18</td>
<td>21</td>
<td>85%</td>
</tr>
<tr>
<td>difference</td>
<td>74</td>
<td>152</td>
<td>49%</td>
</tr>
<tr>
<td>Input params</td>
<td>124</td>
<td>304</td>
<td>41%</td>
</tr>
<tr>
<td>Output params</td>
<td>95</td>
<td>190</td>
<td>50%</td>
</tr>
<tr>
<td>All params</td>
<td>219</td>
<td>494</td>
<td>45%</td>
</tr>
</tbody>
</table>

Adherence to standard is < 50%

Comments are 68% correct! (which is very good)

### Raw inconsistencies

<table>
<thead>
<tr>
<th>Raw inconsistencies</th>
<th>Number</th>
<th>Ratio (%)</th>
<th>of basic facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not coded inputs</td>
<td>21</td>
<td>17%</td>
<td>comm inputs</td>
</tr>
<tr>
<td>Not comm inputs</td>
<td>5</td>
<td>2%</td>
<td>coded inputs</td>
</tr>
<tr>
<td>Not coded outputs</td>
<td>6</td>
<td>6%</td>
<td>comm outputs</td>
</tr>
<tr>
<td>Not comm outputs</td>
<td>25</td>
<td>12%</td>
<td>coded outputs</td>
</tr>
</tbody>
</table>

### Raw summary

<table>
<thead>
<tr>
<th>Raw summary</th>
<th>Number</th>
<th>Ratio (%)</th>
<th>of basic facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inconsistencies</td>
<td>57</td>
<td>25%</td>
<td>comm params</td>
</tr>
<tr>
<td>Inconsistent params</td>
<td>33</td>
<td>14%</td>
<td>comm params</td>
</tr>
<tr>
<td>Inconsistent functions</td>
<td>24</td>
<td>32%</td>
<td>comm funcs</td>
</tr>
</tbody>
</table>
### Results (cont’d)

#### Refined inconsistencies

<table>
<thead>
<tr>
<th>Refined inconsistencies</th>
<th>Number</th>
<th>Ratio (%)</th>
<th>of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comm inp as outp</td>
<td>3</td>
<td>60%</td>
<td>not com inp</td>
</tr>
<tr>
<td>Code has added inp</td>
<td>2</td>
<td>40%</td>
<td>not com inp</td>
</tr>
<tr>
<td>Comm outp as inp</td>
<td>21</td>
<td>84%</td>
<td>not com outp</td>
</tr>
<tr>
<td>Code has added outp</td>
<td>4</td>
<td>16%</td>
<td>not com outp</td>
</tr>
<tr>
<td>Comm inp as outp</td>
<td>3</td>
<td>50%</td>
<td>not cod outp</td>
</tr>
<tr>
<td>Code has less outp</td>
<td>3</td>
<td>50%</td>
<td>not cod outp</td>
</tr>
<tr>
<td>Com. outp as inp</td>
<td>21</td>
<td>100%</td>
<td>not cod inp</td>
</tr>
<tr>
<td>Code has less inp</td>
<td>0</td>
<td>0%</td>
<td>not cod inp</td>
</tr>
</tbody>
</table>

#### Refined summary

<table>
<thead>
<tr>
<th>Refined summary</th>
<th>Number</th>
<th>Ratio (%)</th>
<th>of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inp versus outp params per function</td>
<td>24</td>
<td>73%</td>
<td>incons. params</td>
</tr>
<tr>
<td>Added or rem params per function</td>
<td>9</td>
<td>27%</td>
<td>incons. params</td>
</tr>
<tr>
<td>Funcs with both issues</td>
<td>3</td>
<td>15%</td>
<td>incons. funcs</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2%</td>
<td>incons. funcs</td>
</tr>
</tbody>
</table>
Summary

Applications of The Meta-Environment

- Engineering process:
  - Parsing: grammar engineering in SDF
  - Extraction: parse tree traversal & program dependency graph
  - Analysis: enrichment and actual analysis
  - Visualization: tables

- Result:
  - Company specific analysis in a week or so
Advertisements

- http://www.meta-environment.org
- Open source
- Developers welcome!
- http://www.eclipse.org/imp
  - Technology transfer to Eclipse
  - Reuse of Eclipse platform
- Look out for "Rascal":
  - combined analysis and transformation DSL