Generic Software Transformations

Jan Heering
Department of Software Engineering
CWI
Amsterdam
Jan.Heering@cwi.nl
www.cwi.nl/~jan

Joint work with Ralf Lämmel

October 24, 2004
Generic transformations

- transformation schemes
- capture common principles underlying transformations across
  - different languages
  - different constructs in the same language
- can be instantiated to actual transformations
- basic arguments
  - language: GPL, DSL, modeling language
  - language concept: construct, cross-cutting concept
Example

Instantiations of generic extract

- **extract**[C, variable]
  common subexpression elimination in C

- **extract**[EBNF, non-terminal]
  elimination of common parts of right-hand sides of EBNF syntax rules

- **extract**[C, function]
  function folding in C

- **extract**[Java, method]
  method folding in Java

- **extract**[C++, template]
  folding class definitions into template instances in C++
Even more genericity … (I)

Disregard purpose of transformation

- time/space optimization vs. structure improvement/refactoring
- generic notion of code smell
- basic arguments
  - language: GPL, DSL, modeling language (as before)
  - criterion: time/space use, static (many)
Even more genericity … (II)

Abstract from system

- systems very useful, but also cause of system-centered fragmentation of the field
- ASF+SDF vs. DMS vs. Stratego vs. TXL vs. …
- increasingly hard to develop system-generic view
Outlook/problems

• genericity often intuitively clear, but hard to express formally
• common ground
  – software transformation \(\approx\) theorem proving
  – \(\text{STS} \approx\) theorem prover
  – (extended) equational logic/universal algebra/term rewriting
• what about the conditions?
  separate logical conditions from control/strategy
• how/what to parameterize?