# Ein Faktenextraktor für C++ mit Unterstützung für Typrekonstruktion

Philipp Haller und Volker Kuttruff und Olaf Seng
FZI Forschungszentrum Informatik
Karlsruhe
{haller, kuttruff, seng}@fzi.de

## 1 Einleitung

Die Qualität der Struktur von Softwaresystemen bestimmt maßgeblich den zur Wartung nötigen Aufwand. Damit die Software wartbar bleibt, müssen strukturelle Schwachstellen [2] immer wieder gesucht und behoben werden. Zur Suche existieren bereits zahlreiche auf Metriken und Heuristiken basierende Ansätze [1][4], die voraussetzen, dass die im Quelltext vorhandene Struktur zunächst mit Hilfe eines Faktenextraktors in eine geeignete Repräsentation überführt wird.

Bei der Faktenextraktion können folgende Schwierigkeiten auftreten. Zunächst kann es immer wieder vorkommen, dass zum Zeitpunkt der Analyse z.B. aus lizenzrechtlichen Gründen nicht der komplette Quelltext zur Verfügung steht, so dass nicht alle Referenzen aufgelöst werden können. Im Falle von C++ existieren eine Vielzahl von Sprachdialekten, die zum Teil nur mangelhaft dokumentiert sind. Somit muss in der Praxis mit fast beliebigen Spracherweiterungen gerechnet werden. Eine weitere Schwierigkeit stellen Präprozessordirektiven dar, mit dem beliebige Textbausteine in den Quelltext kopiert werden können. Ohne eine vollständige Verarbeitung der Präprozesseranweisungen kann es vorkommen, dass der Quelltext nicht einmal eine korrekte Blockschachtelung aufweist.

In diesem Papier wird der Entwurf und die Implementierung eines Faktenextraktors für die Sprache C++ beschrieben, der folgenden Anforderungen genügt: der Faktenextraktor soll robust sein, d.h. er soll mit unvollständigem Quelltext und verschiedenen Sprachdialekten umgehen können. Ein Abbruch der Extraktion ist unter keinen Umständen tolerierbar. Da die Suche nach Strukturproblemen regelmäßig erfolgen soll, sollte ein Analyselauf selbst für Systeme mit über einer Million Zeilen Quelltext über Nacht durchlaufen. Die aufgrund unvollständiger Quellen oder nicht unterstützter Spracherweiterungen verlorengegangene Typinformation soll aus der Verwendung im Quelltext abgeleitet werden.

## 2 Ansatz

Als Grundlage unseres Faktenextraktors dient der frei verfügbare C++-Zerteiler des Eclipse CDT-Projekts. Um Typinformationen zu rekonstruieren verwenden wir ein einfaches Typinferenzsystem.

### 2.1 C/C++ Development Tools (CDT)

Das Eclipse CDT-Projekt[1] ist eine Open-Source Entwicklungsumgebung für C und C++, die als Pluginsammlung für die Eclipse Entwicklerplattform realisiert ist. Ein wesentlicher Bestandteil des CDT-Pakets ist ein auch unabhängig von Eclipse nutzbarer Zerteiler, dessen Ergebnisse u.a. für Quelltextvervollständigungen benutzt werden. Um während der Eingabe unvollständigen Quelltext verarbeiten zu können und um möglichst viele verschiedene C/C++-Dialekte zu unterstützen, wurde der Zerteiler bewusst fehlertolerant gehalten. Diese Fehlertoleranz bezieht sich sowohl auf die syntaktische als auch auf die semantische Analyse, d.h. im Gegensatz zu den meisten anderen Compiler-Frontends bricht die semantische Analyse nicht bei Fehlern wie unauffindbaren Deklarationen ab. Resultat einer solchen Analyse ist ein u.U. unvollständiger attributierter Strukturbaum, wobei auf den Aufbau des Strukturbaums mit Hilfe einer entsprechenden CDT-Zerteiler API Einfluss genommen werden kann.

### 2.2 Typrekonstruktion

Der Umgang mit unvollständigem und/oder fehlerhaftem Quellcode resultiert oftmals in fehlenden Typdeklarationen. Um für Qualitätsanalysen trotzdem möglichst viele Informationen aus dem zu untersuchenden Quellcode herauszuziehen, haben wir einen CDT-Zerteiler basierten Faktenextraktor entwickelt, welcher – soweit möglich – nicht bekannte Typen rekonstruiert. Wird bei einer Variablendeklaration `x` ein Typ referenziert (d.h. der Typname ist bekannt), für den aber keine Deklaration gefunden werden kann, so wird ein leerer Typ mit diesem Namen angelegt. Wird im weiteren Verlauf auf Merkmale des Typs via `x` zugegriffen, so werden diese im angelegten Typ eingeführt. Die angenommenen (Parameter-)Typen der rekonstruierten Attribute und Methoden ergeben sich hierbei aus dem Kontext des Zugriffs und werden sukzessive verfeinert. Das Ableiten der Typinformationen kann hierbei als Datenflussproblem aufgefasst werden. Folgendes Beispiel illustriert dies:

```
01  class K { ... }
02  class L : K { ... }
03
04  void f() {
05     X x; X *y; K k; L l;
06     ...
07     k = x.a;
```

[1] http://www.eclipse.org/cdt

| Projekt | #Zeilen | Zeit | Speicher | #Klassen | #Attribute | #Methoden | #Funktionen | #Aufrufe | #Zugriffe |
|---|---|---|---|---|---|---|---|---|---|
| Fallstudie 1 | 88710 | 02:25:14 | 231MB | 258 | 1443 | 3337 | 33 | 14004 | 16851 |
| OO Writer | 872015 | 03:37:57 | 398MB | 1421 | 8606 | 20662 | 3412 | 135192 | 67177 |

Abbildung 1: Analyseergebnisse

```
08      x.b(l);
09      y->b(k);
10      ...
11  }
```

Sei `X` ein nicht bekannter Typ, `K` und `L` bekannte Typen. Bei der Deklaration von `x` in Zeile 5 wird nun ein zu rekonstruierender Typ mit Namen `X` angelegt. Aus Zeile 7 kann man folgern, dass `X` eine Attribut besitzt, dessen Typ mindestens so speziell wie `K` sein muss, während Zeile 8 besagt, dass `X` eine Methode `b` besitzt, dessen Parametertyp `L` oder genereller sein muss. Letztere Information kann mit Hilfe des Zugriffs aus Zeile 9 verfeinert werden, d.h. der Parametertyp muss den Typ `K` oder genereller haben. Über den Rückgabetyp von `b` kann man in diesem Beispiel keine Aussage machen. Durch den Faktenextraktor wird also ein Typ mit Namen `X` rekonstruiert, welcher ein Attribut `a` vom Typ `K` und eine Methode mit der Signatur `void b(K p1)` hat.

### 2.3 Typinferenz

Das obige Verfahren setzt voraus, dass sowohl der Namen des zu rekonstruierenden Typs als auch die zur Rekonstruktion der Attribut- und Parametertypen benutzten Typen bekannt sind. Ist dies nicht der Fall, zum Beispiel weil einer der zur Rekonstruktion verwendeten Typen wiederum ein noch zu rekonstruierender Typ ist, so kann man mit Hilfe von Typinferenz die konkreten Typen der Merkmale approximieren. Allerdings können die klassischen Typinferenztechniken wegen möglicherweise unvollständigen Quellen nicht eingesetzt werden. Aus diesem Grund haben wir in unserem Faktenextraktor einen Typinferenzer implementiert, der versucht, die durch Zuweisungen induzierten Typconstraints zu lösen.

## 3 Verwandte Arbeiten

Für die Faktenextraktion lässt sich prinzipiell jedes ausreichend offene C++-Compiler-Frontend nutzen, wie zum Beispiel das kommerzielle Frontend der Edison Design Group (EDG) oder das Frontend des GCC-Übersetzers. Problematisch ist hierbei allerdings generell die Fehlertoleranz, d.h. einen attributierten Syntaxbaum erhält man meist nur für den unterstützten C++-Sprachdialekt. Weiterhin existieren kommerzielle Lösungen zur Faktenextraktion wie z.B. Columbus/CAN[2], welche aber aufgrund aufgetretener Performance- und Stabilitätsprobleme nicht weiter verfolgt wurde.

## 4 Evaluation und Diskussion

Die Praxistauglichkeit unseres Faktenextraktors haben wir anhand zweier Fallstudien getestet. Bei der ersten Fallstudie handelt es sich um eine Eigenentwicklung, die die MFC-Bibliothek von Microsoft verwendet. Anhand dieser mittelgroßen Fallstudie wird der Umgang des Faktenextraktors mit Systemen getestet, die mit Microsoft C++-Übersetzern entwickelt wurden. Als zweite, große Fallstudie haben wir OpenOffice Writer untersucht. Um möglichst gute Ergebnisse zu erzielen, haben wir zunächst alle benötigten Include-Verzeichnisse in die Analyse mit einbezogen. Tabelle 1 zeigt die Systemgröße, die Dauer der Extraktion und den Speicherverbrauch sowie die wichtigsten Analyseergebnisse. Beide Analysen wurden auf einem Intel Pentium 4 mit 3 GHz und 1GB Hauptspeicher durchgeführt.

Die Analysezeit ist selbst für die große Fallstudie ausreichend kurz. Der Speicherverbrauch hält sich ebenfalls noch in Grenzen. Dass die Analysedauer für OO Writer nur das ca. 1,5fache der Zeit der erheblich kleineren Fallstudie beträgt, liegt daran, dass hier für fast jede Datei ein Großteil der Schnittstellen der MFC-Bibliothek vom Zerteiler verarbeitet werden muss.

Die stichprobenartige Untersuchung der extrahierten Fakten für einzelne Quelldateien lässt auf eine weitgehend vollständige Analyse schließen.

Um das Verfahren der Typrekonstruktion zu testen wurden in einem Lauf die ”Header”-Dateien der MFC-Bibliothek von der Analyse ausgeschlossen. Wie erwartet wurden hier die von der Fallstudie verwendeten Typen fast vollständig rekonstruiert. Zum Beispiel wurden für die Klasse `CRect` vier Attribute und sieben Methoden (von tatsächlich acht verwendeten) richtig rekonstruiert.

Bezüglich der Korrektheit der Ergebnisse muss erwähnt werden, dass ohne vollständige Quellen und ohne Berücksichtigung eines Makefiles keine zu 100% korrekten Ergebnisse erzielt werden können. Dies kann für Qualitätsanalysen aber in Kauf genommen werden, da alle gefundenen strukturellen Auffälligkeiten zunächst nur Anomalien sind, die manuell als Problem zu bestätigen sind. Eventuelle Ungenauigkeiten bei der Faktenextraktion können zu diesem Zeitpunkt entdeckt werden.

[2] http://www.frontendart.com

## Literatur


[1] O. Ciupke. Automatic Detection of Design Problems in Object-Oriented Reengineering. In *Technology of Object-Oriented Languages and Systems - TOOLS 30*, 1999.

[2] M. Fowler. *Refactoring - Improving the Design of Existing Code*. Addison Wesley, 1999.

[3] P. Haller. Ein Faktenextraktor für C++ mit Unterstützung für Typinferenz. Studienarbeit, FZI an der Universität Karlsruhe, 2005.

[4] R. Marinescu. Detecting Design Flaws via Metrics in Object-Oriented Systems. In *Proceedings of Technology of Technology of Object-Oriented Languages and Systems - Tools 39*, 2001.