Ein Präprozessor-Repository für das Reverse-Engineering

Volker Riediger
Institut für Softwaretechnik
Universität Koblenz-Landau
riediger@uni-koblenz.de

1 Einführung

Der Einsatz von Präprozessoren in Programmiersprachen stellt ein bedeutendes Problem in der Softwaretechnik dar. Durch textuelle Transformationen, durch die Inklusion externer Quelltexte und durch tief verschachtelte komplexe Bedingungen wird das Begreifen von Zusammenhängen und die Inspektion bestehender Systeme erschwert. Detailliertes Verständnis bestehender Software ist jedoch die Voraussetzung nahezu aller Aktivitäten im Software-Lebenszyklus: Implementation, Test, Wartung, Pflege und Weiterentwicklung von Softwaresystemen erfordern die Analyse von Quelltexten in Gegenwart von Präprozessor-Anweisungen. Die Präprozessor-Problematik wird damit zu einer wesentlichen Fragestellung im Reverse Engineering und beim Programmverstehen.

Der Fokus im präprozessor-bezogenen Reverse Engineering lag bisher lediglich auf dem C-Präprozessor. Andere weit verbreitete Programmiersprachen blieben unberücksichtigt. In [3] wurden die *Präprozessoren der Sprachen C/C++, COBOL und PL/I* vergleichend untersucht und hinsichtlich ihrer Fähigkeiten zur Manipulation des Präprozessor-Input in einen gemeinsamen Rahmen gestellt.

Mit einem sprachunabhängigen *Repository-Schema* wurde ein Datenmodell für die Repräsentation von Präprozessor-Fakten entwickelt. Dieses Schema erlaubt eine von der konkreten Programmiersprache und deren speziellen Verarbeitungsregeln unabhängige Repräsentation von Präprozessor-Aktionen in *Fold-Graphen*.

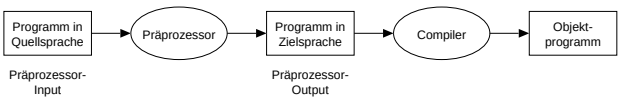

Abbildung 1: Präprozessor-Input und -Output in der Compiler-Kette

Aufbauend auf diesem Repository-Schema werden *graphbasierte Algorithmen und Werkzeuge* realisiert, die im GUPRO-Toolset [1] die Kluft zwischen Präprozessor-Input und Präprozessor-Output überbrücken. Der von der GUPRO-Umgebung unabhängige *Fold-Graph-Viewer* ermöglicht die interaktive Analyse, Visualisierung und Erkundung von Programmen in Gegenwart von Präprozessor-Anweisungen.

2 Präprozessor-Repository

Bei der Verarbeitung einer Quelldatei durch den im GUPRO-C-Frontend eingesetzten C-Präprozessor *GCPP* werden zusätzlich zum normalen Präprozessor-Output Informationen für die spätere Visualisierung und Auswertung in einer Datenstruktur, dem so genannten Fold-Graphen, abgelegt. Diese Graphen sind Instanzen der Graphklasse `FoldGraph` [2, 3], die das Repository-Schema für präprozessorbezogene Analysen und Visualisierungen bildet.

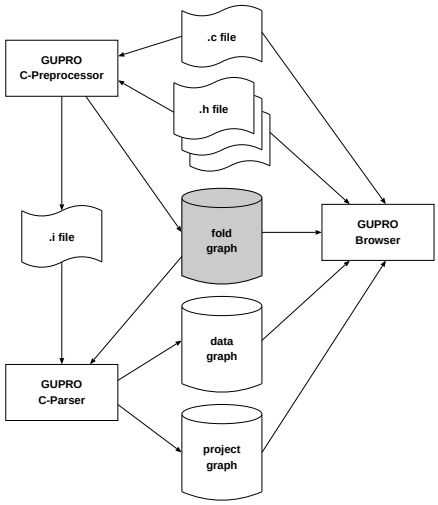

Abbildung 2: Fold-Graphen im GUPRO-Kontext

Die Rolle der Fold-Graphen im GUPRO-Kontext wird durch die Abbildung 2 verdeutlicht. Fold-Graphen werden vom GUPRO-C-Parser *GCPA* zusammen mit dem Präprozessor-Output verarbeitet. Auch der Quelltext-Browser der grafischen Benutzungsschnittstelle von GUPRO greift auf die Folding-Informationen zurück und verbindet den Daten-Graphen zur Analyse mit dem Fold-Graphen zur Visualisierung. Die GUI-Anwendung *Fold-Graph-Viewer* (Abb. 3) kann aber auch außerhalb des GUPRO-Systems als Visualisierungs- und Analyseschnittstelle für Fold-Graphen verwendet werden.

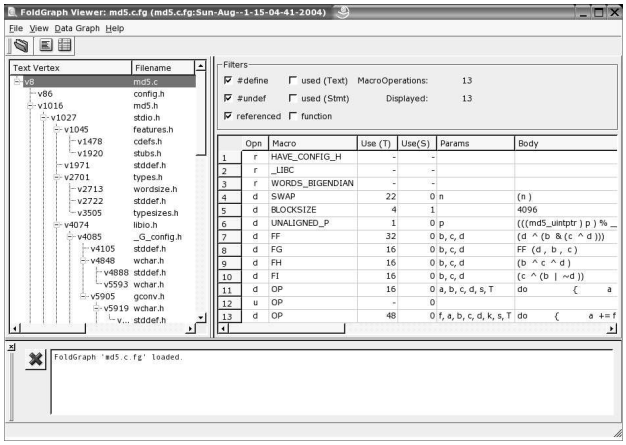

Abbildung 3: Fold-Graph-Viewer

3 Anwendungsbeispiel: Automatische Klassifikation von Makro-Aufrufen

Durch Fold-Graphen wird genügend Information zur Verfügung gestellt, um alle Aktionen des Präprozessors vom Präprozessor-Output auf den Präprozessor-Input abbilden zu können. Ein Beispiel zur Nutzung von Fold-Graphen ist die automatische Klassifikation von Makro-Aufrufen. Dabei wird jeder Makro-Expansion eine syntaktische Kategorie des abstrakten Syntaxbaums zugeordnet. Zu den syntaktischen Kategorien der Sprace C gehören z.B. Constant, FunctionCall, Expression, Declaration und SimpleType.

Eine Klassifikation von Makro-Aufrufen erleichtert das Verstehen des Zwecks der Makro-Definitionen und kann auch zur Beurteilung der Software-Qualität genutzt werden. So können zm Beispiel Fragen, wie die in einem Programm definierten Makros genutzt werden, oder ob und wie die Nutzung ein und desselben Makros sich innerhalb eines Projekts ändert, durch Vergleich der syntaktischen Kategorien beantwortet werden.

Zur Klassifikation werden über die Quelltext-Koordinaten der Makro-Aufrufe im Präprozessor-Input in Koordinaten des Präprozessor-Output transformiert. Die Aufrufe können dabei in beliebig tief verschachtelten Makros liegen. Die Ausgabe-Koordinaten dienen dann als Suchkriterium im abstrakten Systaxbaum. Die von der C-Syntax unabhängige Verarbeitungsweise des C-Präprozessors kann dabei zu drei unterschiedlichen Ergebnissen führen:

1. Der Koordinatenbereich passt exakt zu einem Teilbaum (Match-Typ *exactMatch*)
2. Der Koordinatenbereich liegt vollständig innerhalb eines Teilbaums (Match-Typ *smallestEnclosing*)
3. Es werden mehrere Teilbäume mit überlappenden Koordinaten gefunden werden, wobei die größte Überlappung gewertet wird (Match-Typ *maxOverlap*)

Beispiele für diese drei Klassifizierungen finden sich in Abbildung 4. Die Makro-Expansion `+4` für den Aufruf `PLUS(4)` in Zeile 3 ist ein *exactMatch* der Kategorie *Expression*, der Aufruf `PLUS(2)` des gleichen Makros

```
1:  #define PLUS(x) + x
2:  #define BAD 5; d = 6
3:  a = PLUS(4);
4:  b = 1 PLUS(2) PLUS(3);
5:  c = BAD;
```

Abbildung 4: Präprozessor-Input

```
3:  a = + 4;
4:  b = 1 + 2 + 3;
5:  c = 5; d = 6 ;
```

Abbildung 5: Präprozessor-Output

in Zeile 4 kann dem Ausdruck `1+2` als *smallestEnclosing* (ebenfalls Kategorie *Expression*) zugeordnet werden. Der Makro-Aufruf `BAD` in Zeile 5 erzeugt dagegen unterschiedliche nicht ineinander geschachtelte Teilbäume und wird daher durch *maxOverlap* der Zuweisung `d=6` als *ExpressionStatement* klassifiziert. Die dem Makro-Aufruf entsprechenden Teile des Präprozessor-Output sind in Abb. 5 unterstrichen dargestellt.

Bei der Analyse der GNU Core Utilities, einer der grundlegenden C-Bibliotheken des Linux-Systems, konnten 40% der 19.791 Makro-Aufrufe exakt zugeordnet werden, weitere 47% lagen innerhalb eines Teilbaums, und die restlichen 13% konnten durch maximale Überlappung auf GNU-C-spezifische Erweiterungen der Programmiersprache C zurückgeführt werden. Die Code-Größe der Core Utilities umfasst 67.484 Zeilen C-Quelltext in 92 C- und 239 H-Dateien, die vom Präprozessor in 218.304 nichtleere Zeilen Präprozessor-Output übersetzt werden.

4 Zusammenfassung

Fold-Graphen mit den darauf aufbauenden Algorithmen können als erste Technologie zur feingranularen Analyse größerer Software-Systeme mit Präprozessor eingesetzt werden. Dabei kann die Transformation des Präprozessors auf beliebigen Ebenen untersucht werden. Der Bezug zum Original-Quelltext geht dabei nicht verloren. Die Ergebnisse sind auf viele Präprozessorsprachen anwendbar.

Literatur

[1] J. Ebert, R. Gimnich, H.H. Stasch, and A. Winter, editors. *GUPRO — Generische Umgebung zum Programmverstehen*. Koblenzer Schriften zur Informatik. Fölbach, Koblenz, 1998.

[2] B. Kullbach and V. Riediger. Folding: An approach to enable program understanding of preprocessed languages. In *[4]*, pages 3–12, October 2001.

[3] V. Riediger. *Die Präprozessor-Problematik im Reverse-Engineering und beim Programmverstehen*. Logos Verlag, Berlin, 2005.

[4] *8th Working Conference on Reverse Engineering (Stuttgart)*. IEEE Computer Society, Los Alamitos, California, USA, 2001.