Erzeugung und Anwendungen dynamischer Objektprozessgraphen

Jochen Quante, Rainer Koschke
Arbeitsgruppe Softwaretechnik
Fachbereich 3, Universität Bremen
http://www.informatik.uni-bremen.de/st/
{quante, koschke}@informatik.uni-bremen.de

1 Einführung

Objektprozessgraphen beschreiben den Kontrollfluss eines Programms aus der Sicht eines einzelnen Objekts. Wenn dieses Objekt von zentraler Bedeutung für das untersuchte Programm ist, können diese Graphen eine große Unterstützung für das Verstehen des Programms – sowohl seiner Struktur als auch seiner Abläufe – sein. Neben der dynamischen Extraktion solcher Graphen zeigen wir dies an einem Beispiel.

2 Tracing

Eine *Spur (engl. Trace)* beschreibt die Folge von Operationen, die in einem Programmlauf ausgeführt wird. Eine *Objektspur* enthält den Teil einer Spur, der für ein gegebenes Objekt relevant ist. Eine Objektspur entspricht damit genau einem Pfad im Objektprozessgraphen, und umgekehrt ergibt sich der Objektprozessgraph aus der Menge aller für ein Objekt möglichen Objektspuren.

Der Objektprozessgraph kann somit dynamisch angenähert werden, indem die Objektspuren verschiedener Testläufe zu einem Graphen vereinigt werden. Dies wird allerdings in den meisten Fällen nur zu einem unvollständigen Graphen führen. Im Gegensatz dazu liefert eine entsprechende statische Analyse [1] einen Graphen, der alle möglichen Objektspuren enthält – allerdings auch mehr als das. Aufgrund pessimistischer Annahmen, die beim Vorhandensein von Funktionszeigern, dynamischem Binden und ähnlichem nötig sind, enthält der Graph meist auch Pfade, die nicht ausführbar sind. Beide Verfahren, dynamische und statische Analyse, liefern also nur Näherungen des idealen Objektprozessgraphen.

Im folgenden wird die dynamische Extraktion genauer beschrieben.

3 Dynamische Extraktion

Die dynamische Extraktion erfordert zwei Phasen: In der ersten Phase wird das Programm instrumentiert, um die notwendigen Spur-Informationen generieren zu können. In der zweiten Phase wird dann das so modifizierte Programm ausgeführt, Trace-Daten werden gesammelt, und diese Daten werden schließlich ausgewertet.

3.1 Instrumentierung

Die Instrumentierung führen wir auf dem abstrakten Syntaxbaum durch. Zur Vereinfachung der eigentlichen Instrumentierung wird zunächst der Syntaxbaum normalisiert, indem die verschiedenen Schleifen (while, for, do-while) sowie switch-Anweisungen durch entsprechende if- und goto-Konstruktionen repräsentiert werden. Abb. 1(a) zeigt ein normalisiertes Beispielprogramm. Anschließend werden an den nötigen Stellen Tracing-Anweisungen eingefügt:

- begin/end_life: Gültigkeitsbereiche, Konstruktoren/Destruktoren
- read/write: Objektzugriffe, Methodenaufrufe des Objekts
- branch_true/false: Verzweigungen
- entry/return: Unterprogrammaufrufe
- labels: für Identifikation von Schleifen

Im Gegensatz zu anderen Ansätzen (z.B. [2]) werden also zusätzliche Informationen über den Kontext und das Verhältnis zum Anwendungscode gesammelt. Die entsprechenden Tracing-Anweisungen schreiben neben dem Typ der Operation einen eindeutigen Identifikator der Stelle im Programmtext sowie die Adresse des Objekts, sofern sich die Operation auf ein Objekt bezieht. Schließlich wird aus dem modifizierten Syntaxbaum wieder Code erzeugt.

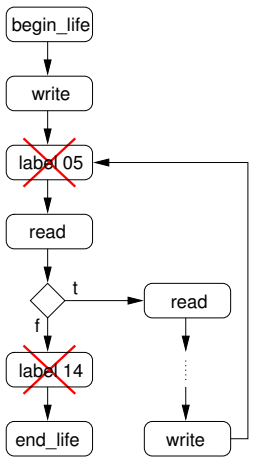
Das so instrumentierte Programm wird dann in der zweiten Phase ausgeführt und erzeugt bei jedem Durchlauf einen Trace. Aus diesen Traces können durch Filterung wiederum Objektspuren extrahiert werden, die dann als Eingabe für die Erzeugung des Objektprozessgraphen dienen.

3.2 Konstruktion des Objektprozessgraphen

Die Erzeugung des Objektprozessgraphen erfolgt wiederum in zwei Schritten. Zunächst wird die Objektspur zeilenweise durchlaufen, und für jeden vorkommenden Identifikator wird ein Knoten erzeugt. Die Knoten werden entsprechend der Reihenfolge in der Objektspur miteinander verbunden. Dabei entstehen Schleifen, wo im Originalprogramm ebenfalls Schleifen vorhanden waren. Für das Beispielprogramm zeigt Abb. 1(b) den entstehenden Graphen.

```
void notset(int * s)
{
  int i;
  i = 0;
  label_05:
  if (i < z)
  {
    s[i] = ~s[i];
    ++i;
    goto label_05;
  }
  else {
  }
  label_14:
}
```

(a) Norm. Beispielprogramm



(b) Objektprozessgraph

Abbildung 1: Beispielprogramm und Objektprozessgraph für eine Instanz von i

Im zweiten Schritt wird der so generierte Roh-Graph vereinfacht, indem Knoten entfernt werden, die für den Objektprozessgraphen irrelevant sind. Dazu gehören Labels sowie für den Kontrollfluß irrelevante Verzweigungen und Teilgraphen. Dieser Schritt wird iteriert, bis keine weiteren Vereinfachungen mehr möglich sind. Das Ergebnis ist der Objektprozessgraph.

4 Fallstudie

Das Verfahren wurde unter anderem an dem Chat-Client ircII getestet. Er besteht aus 50-60 KLOC C-Code, der Kontrollflußgraph hat ca. 25.000 Knoten, der Aufrufgraph ca. 1.700 Knoten. Als relevantes Objekt wurde der Socket betrachtet, da dieser das zentrale Anliegen (Kommunikation) der Anwendung charakterisiert.

In Abb. 2 wird der resultierende Objektprozessgraph für den Socket von ircII dargestellt. Er liefert hilfreiche Informationen für das Verstehen von Aufbau und Funktionsweise dieser Anwendung. Der Ablauf beginnt beim Start-Knoten (A). Zunächst wird der Socket erzeugt und initialisiert. Wir sehen die Aufrufstruktur über verschiedene Funktionen. Anschließend gelangen wir zur Hauptschleife (B). Von hier zweigen verschiedene Pfade ab, die sich mit Serverinteraktionen und Benutzereingaben beschäftigen. Alle diese Pfade führen zum Senden von Informationen an den Server (C). Schließlich wird diese Schleife verlassen, der Socket wird geschlossen und der Lebenszyklus des Sockets endet (D).

Neben diesem groben Überblick über die Abläufe sind aus dieser Darstellung auch Details über die Implementierung zu entnehmen. So ist erkennbar, dass der Aufruf der verschiedenen Benutzerfunktionen über Funktionszeiger erfolgt, da in diesem Fall ein Call-Knoten zu mehreren Funktionen führt (E).

5 Ausblick

Neben dem Programmverstehen kann der Objektprozessgraph Ausgangspunkt für viele weitere Anwendungen sein:

- Protokollerkennung: durch Abstraktion kann das Protokoll der Komponenten, wie es von der Anwendung benutzt wird, hergeleitet werden.
- Merkmalslokalisierung
- Testabdeckung bezüglich einzelner Objekte
- Identifikation unausführbarer Pfade

Literatur

[1] T. Eisenbarth, R. Koschke, and G. Vogel. Static object trace extraction for programs with pointers. *Journal of Systems and Software*, 77(3):263–284, Sep 2005.

[2] J. Larus G. Ammons, R. Bodik. Mining specifications. In *Proc. 29th symp. on Principles of prog. languages*, Portland, Oregon, USA, 2002.

[3] J. Quante and R. Koschke. Object process graphs. In *Proc. 10th European Conf. on Software Maintenance and Reengineering*, Bari, Italy, March 2006.

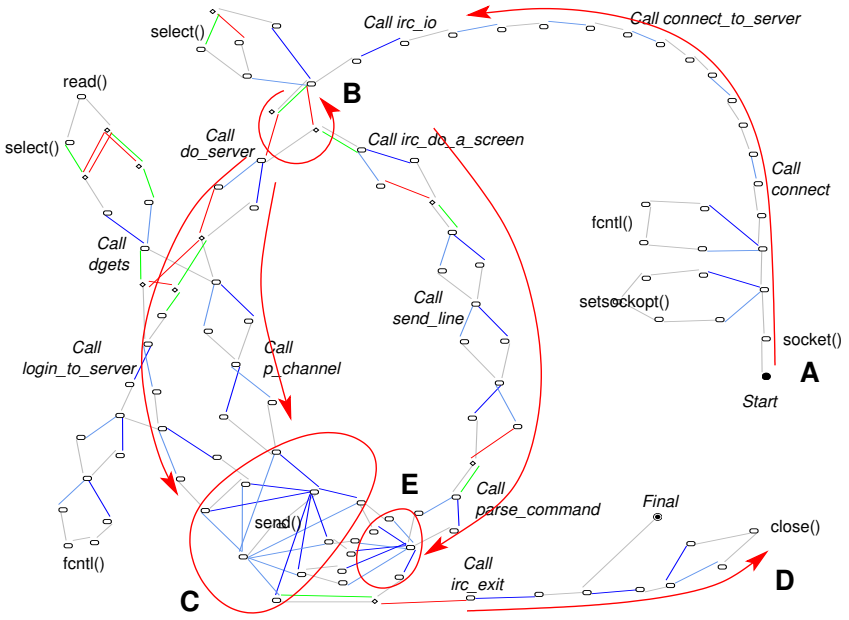


Abbildung 2: Objektprozessgraph für ircII