— Blatt 1 —

Aufgabe 1

Laden Sie den Quellcode eine modifizierten Version von SMT-RAT von der folgenden URL herunter:

`http://ths.rwth-aachen.de/wp-content/uploads/sites/4/teaching/praktikum_smt_WS1516/ufdemo.tgz`

Entpacken Sie das Archiv und kompilieren Sie zunächst CArL und dann SMT-RAT. Das Ergebnis sollte eine ausführbare Datei mit dem Namen *smtrat* sein.

Die voreingestellte Strategie *UFDemo* beinhaltet lediglich drei Module:

- *CNFerModule*: Konvertiert die Eingabeformel in *CNF*.
- *SATModule*: Implementiert einen *SAT-Solver*.
- *UFDemoModule*: Implementiert einen Theorie-Solver für die Theorie der *Uninterpretierten Funktionen*.

Im Folgenden sollen Sie sich mit dem *UFDemoModule* beschäftigen. In diesem haben sich leider einige Fehler eingeschlichen, die Sie im Team finden, erklären und beheben sollen. Beispieleingaben finden Sie im Ordner *samples/*.

Dokumentieren Sie zu jedem Fehler die folgenden Punkte:

- Ursprünglicher Quellcode
- Konkreter Fehler
- Erklärung, wie Sie den Fehler identifiziert haben
- Erklärung, warum dies zu einem Fehler führt
- Korrigierter Quellcode

Disclaimer

Einige der zu behebenden Fehler, und damit deren Auswirkungen, sind stark vom genutzen System abhängig, beispielsweise der verwendeten Prozessorarchitektur, der Compilerversion, dem Speicherlayout.

Aufgabe 2

Versuchen Sie einige der Beispiele zu lösen. Sie werden feststellen, dass der Solver auf einigen Beispielen abstürzt, beispielsweise mit der Fehlermeldung `"Illegal instruction"` oder `"Segmentation fault"`. Nutzen Sie *gdb* um die Fehler zu lokalisieren und zu beheben.

Starten Sie den Solver durch den Befehl `gdb --args ./smtrat <filename>` und starten Sie ihn in *gdb* durch den Befehl `run`. Einen *Backtrace* erhalten Sie, nachdem das Programm abgestürzt ist, mit dem Befehl `backtrace`.

Aufgabe 3

Auch wenn der Solver nun Beispiele lösen kann, gibt es weitere Problemfelder. Ein häufiges Symptom von Programmierfehlern sind sogenannte *Memory Leaks*. Man spricht von Memory Leaks, wenn für Objekte im Verlauf des Programms Speicher reserviert wird, dieser aber nie wieder

freigegeben wird. Insbesondere auf Beispiele mit längeren Laufzeiten kann dies dazu führen, dass das Programm mehr Speicher verbraucht als benötigt, und so eventuell unnötigerweise abstürzt, da kein Speicher mehr verfügbar ist.

Ein solcher Fehler tritt auch im *UFDemoModule* auf. Nutzen Sie *valgrind*, um die Quelle des Memory Leaks zu lokalisieren. Sie starten das Programm dafür wie folgt: `valgrind --leak-check=full ./smtrat <filename>`

Einige der Fehler aus Aufgabe 2 können auch mit *valgrind* gefunden werden.

Aufgabe 4

Die Methode `isUFInClass()` unterscheidet sich stilistisch massiv vom restlichen Code. In dieser Methode wurden einige *best practices* von modernem C++ nicht beachtet, zu umständliche Lösungen oder sogar obsoleter Code implementiert.

Erstellen Sie eine Liste von Dingen, die Sie an diesem Code anders machen würden. Erklären Sie zu jedem Punkt, warum die gewählte Lösung nicht gut ist.

Implementieren Sie eine alternative Variante der Methode.

Tipps & Tricks

- Zunächst werden Sie (zum Beispiel in *gdb* oder *valgrind*) keine Datei- oder Zeilenangaben sehen. Kompilieren Sie SMT-RAT im Debug-Modus für zusätzliche Informationen. Dafür ändern Sie mittels *ccmake* die Option *DEVELOPER* auf *ON*. Achtung: Nun sind zusätzliche Checks aktiv. Dadurch kann sich die Fehlermeldung ändern.
- Lesen Sie einen *Backtrace* von oben nach unten, also von der Stelle des Absturzes in Richtung Programmstart. Überspringen Sie zunächst Funktionen, die nicht in Ihrem Code implementiert sind – beispielsweise Methoden aus der Standardbibliothek.
- Lesen Sie die Ausgabe von *valgrind* stets von oben nach unten. Nach dem ersten Speicherfehler haben eine oder mehrere Programmvariablen in der Regel ungültige Werte und ziehen damit fast zwangsläufig weitere Fehler nach sich.
- Achtung: *valgrind* ist eigentlich eine Art *Emulator*. Im Gegensatz zu *gdb* ist die Ausführung eines Programms mit *valgrind* um ein vielfaches langsamer.
- Neben Speicherfehlern und Memory Leaks kann *valgrind* auch nach anderen Fehlern suchen. *Valgrind* bietet Tools zur Speicherprüfung, (Cache) Profiling, Heap Profiling sowie Erkennung von Data Races.