AUFGABE 7: ABSTRAKTE INTERPRETATION & VERIFIKATION

In dieser Aufgabe beschäftigen Sie sich mit drei Aspekten der statischen Analyse: Laufzeitfehlern, beziehungsweise dem Nachweis ihrer Abwesenheit, numerischen Berechnungen und Rundungsfehlern, sowie funktionaler Korrektheit. Hierzu werden Sie einen einfachen Filter sowie eine Ringpufferimplementierung mittels der Programme Astrée und FRAMA-C analysieren und dabei verschiedene Eigenschaften der beiden Implementierungen beweisen.

1 Aufgabenstellung

Vermerken Sie Ihre Antworten zu den Fragen der einzelnen Aufgaben an den vorgesehenen Stellen in der vorgegebenen `answers.md`*. Bitte erstellen Sie, um die Abgabe durch Mergerequests zu vereinfachen,* ***pro Aufgabe einen eigenen Branch****. Um einen konsistenten Zustand zu gewährleisten, benutzen Sie dazu bitte folgenden Befehl:*
`git fetch git@gitlab.cs.fau.de:ezs/vezs22-vorgabe.git aufgabe7 && git checkout -b aufgabe7 FETCH_HEAD`

Einleitende Bemerkungen

Wie in den Übungsvideos dargestellt, beschäftigen wir uns in diesem Aufgabenblatt (getrennt) mit der Korrektheit eines Bildfilters sowie eines Ringpuffers. Die Arbeitsweise des Bildfilters können sie hierbei anhand des vorgegebenen Beispielbilds (`crosswalk.ppm`) und der Ausgabe des in `gaussian.c` beschriebenen Programms selbst beobachten und überprüfen. Es bietet sich hierfür an, die Ausgabe in eine Datei umzuleiten (`./gaussian crosswalk.ppm >filtered.ppm`).

Sowohl Ein- als auch Ausgabe erfolgen hierbei im PPM-Format P3 (*portable pixmap format*, `https://en.wikipedia.org/wiki/Netpbm#PPM_example`), allerdings unterstützen wir keine Kommentare. Gängige Bildbetrachter wie bspw. *Eye of GNOME* können dieses Format anzeigen (`eog crosswalk.ppm`). Sollten Sie eigene Bilder verwenden wollen, so lassen sich bspw. mit Hilfe von *GIMP* Bilder in das PPM-format umwandeln. Durch die relativ simple Natur des PPM-Formats und der Speicherung als ASCII-Text, lassen sich Bilder zur Fehlersuche auch mit gängigen Textvergleichswerkzeugen (`diff`) vergleichen. Eine optische Betrachtung der Bilder sollte für diese Aufgabe allerdings ausreichen.

Sowohl Astrée als auch FRAMA-C bedienen sich einer graphischen Oberfläche, bitte greifen Sie hier auf eine graphische Arbeitsumgebung wie etwa `remote.cip.cs.fau.de` oder native Xpra-Clienten zurück.

Statische Programmanalyse mittels Astrée

In diesem Aufgabenteil betrachten Sie zunächst einen einfachen Bildfilter. Der Fokus liegt dabei zunächst auf der Abwesenheit von Laufzeitfehlern (Speicherkorrektheit, kein Aufruf von undefiniertem Verhalten, ...), und bewegt sich dann hin zur Betrachtung numerischer Eigenheiten des Filters. Bedenken Sie, dass Sie für die Analyse mit Astrée eine Verbindung in den CIP mit graphischer Umgebung (bspw. Xpra) benötigen.

☞ gaussian.c

☞ /proj/i4ezs/tools/astree_c/bin/a3c

Aufgabe 1 *Einführende Frage*

Was ist der Unterschied zwischen den Astrée-Konzepten der *Known Facts* und der *Assertions*? Wann sollte welches dieser beiden Konzepte zum Einsatz kommen?
Antwort:

Eingabedaten

Hinweis: Bitte beachten Sie unbedingt die Hinweise zur Verwendung von Astrée in den Übungsfolien und nutzen Sie unbedingt die vorgegebene Projektkonfigurationsdatei `./astree-cfg/vezs.dax` als Grundlage ihres Analyseprojektes! Bitte beachten Sie, dass Astrée zwei unterschiedliche Einsatzzwecke besitzt: Einerseits kann das Werkzeug die Einhaltung von Codestilregeln überwachen („Failed coding rule checks“, Alarme hier dürfen Sie in unserer Übungsaufgabe ignorieren), sowie semantische Überprüfungen zur Vermeidung von Laufzeitfehlern (alle anderen Kategorien, diese müssen durch Sie behoben werden).

Aufgabe 2 *Sensor-Stub*

Oft unterliegen Sensor- beziehungsweise Eingabedaten in Regelungssystemen bestimmten natürlichen Einschränkungen (etwa Wertebereiche eines Sensors, Anzahl der Ergebniswerte, usw.). Allerdings sind diese Annahmen oft nicht aus dem Quelltext ersichtlich und die Analyse von Hardwaretreibern schwierig. Stattdessen kommen während der Analyse hier meist sogenannte Stubs zum Einsatz, die die jeweilige Schnittstelle kapseln, und der Analyse gegenüber nur die benötigten Eigenschaften (ohne eigentliche Funktionalität) anzeigen. In unserem Falle betrifft dies das Einlesen der Sensordaten in der Funktion `basic_ppm_parse`, welche eigentlich

Sensordaten aus einer Eingabedatei einliest. Implementieren Sie einen Stub, der das Einlesen kapselt, und der Analyse die folgenden Eigenschaften der Eingabe mitteilt: Die Maximalgröße der zu verarbeitenden Bilder sind durch die Konstanten `MAX_WIDTH` und `MAX_HEIGHT` begrenzt. Ferner ist der maximale Farbwert eines Bildpunktes zwar konfigurierbar, jedoch darf davon ausgegangen werden, dass die Farbkomponenten (rgb, also Rot, Grün und Blau) eines Pixels aufgrund des zugrundeliegenden Datentyps in den Dateien den Wert 255 nicht überschreiten.

Filter

Wie Sie der `main`-Funktion entnehmen können, erfolgt nach dem Einlesen der Bilddaten eine Filterung des Bildes mittels eine Gaussunschärfefilters. Dazu werden zunächst in der Funktion `generate_gauss_kernel` die Parameter der Faltungsmatrix errechnet und mittels der generischen `convolve`-Funktion dann die eigentliche Faltung durchgeführt.

Hinweis: Der Filtercode umfasst zum einen Schleifen, die das gesamte Bild ablaufen, und zum anderen Schleifen, die über die Faltungsmatrix iterieren. Wird das Ablaufen der einzelnen Pixel ausgerollt, steigt der von Astrée verwaltete Zustand enorm an, was wiederum zu extrem langen Analysezeiten führt. Diese Schleifen auszurollen sollte in der Übung nicht nötig sein! Die Faltungsmatrizen hingegen sind klein genug und können bequem auch komplett ausgerollt werden.

Aufgabe 3 *Filterinitialisierung (I)*

Betrachten Sie zunächst die Funktion `generate_gauss_kernel` und führen sie nun eine erste Analyse der Funktion durch. Initial tritt bei der Analyse dieser Funktion eine Vielzahl von Warnungen auf. *Was ist die Ursache für diese Warnungen, wieso scheitert hier die Analyse zunächst? Welches Intervall ergibt sich für die einzelnen Einträge der Filtermatrix?*
Antwort:

Unterstützen Sie nun die Analyse durch semantikerhaltende Codetransformation und geeignete Annotationen, um diese Problemstellen zu beseitigen.

Hinweis: Falls Sie hier vor Problemen stehen, kann es hilfreich sein, Intervallrechnungen händisch nachzuvollziehen, und die Ergebnisse mit den Zwischenergebnis-

sen der Astrée-Analyse zu vergleichen, und immer dann selektiv Zusatzwissen in die Analyse einzubringen, wenn die Berechnungen auseinanderklaffen. Ihre Annotationen sollten dabei zumindest so präzise gestaltet sein, dass die am Funktionsende geforderten Wertebereichseinschränkungen für die Matrixeinträge gelten.

☞ __ASTREE_log_vars

☞ __ASTREE_known_*

Aufgabe 4 *Filterverwendung (I)*

Im nächsten Verarbeitungsschritt erfolgt die Faltung der Eingabedaten mittels der errechneten Filtermatrix in der Funktion `convolve`. Stellen Sie zunächst – wiederrum durch Codetransformation und Annotationen – sicher, dass Astrée hier keine Laufzeitfehler detektiert.

Aufgabe 5 *Filterverwendung (II)*

Um die Wertebereichsbeschränkungen des Aufgabeformates zu garantieren, werden die Farbwerte mittels der Macros `min` und `max` an den Grenzen hart abgeschnitten. Dieser auch als Clipping bezeichnete Vorgang ist wie so oft bei Signalverarbeitungsvorgängen für den vorliegenden Bildfilter nicht erstrebenswert, viel mehr wird eine Gleichverteilung der Werte über den zur Verfügung stehenden Ausgabebereich angestrebt. Der Filter bietet aus diesem Grund einen linearen Skalierungsfaktor `factor`. Fügen Sie geeignete Annotationen ein, um die Clippingfreiheit zu fordern und bestimmen Sie mittels Astrée den maximal zulässigen Skalierungsfaktor.

Aufgabe 6 *Filterinitialisierung (II)*

Tatsächlich ist die in Aufgabe 3 ermittelte Intervallabschätzung der Einträge der Filtermatrix und der hiermit in Aufgabe 5 gewonnene Skalierungsfaktor sehr pessimistisch: Der hier beschriebene Gaußfilter ist ein normierter Filter, welcher ein gewichtetes Mittel umgebender Pixel errechnet. Die Normierung bedeutet, dass die Summe der Gewichtsfaktoren eins ergibt. *Welchen Wertbereich hat jedoch Astrée aktuell von dieser Kenngröße? Welche symbolischen Zusammenhänge sind Astrée hier zwischen relevanten Variablen bekannt?*.

☞ __ASTREE_log_vars

Antwort:

Eigentlich errechnet die Funktion `generate_gauss_kernel` jedoch konstante Matrixwerte. Ersetzen Sie die Filterinitialisierung deshalb durch diese Konstanten (**Hinweis:** bitte halten sie beide Varianten im Code und nutzen sie stattdessen die Präprozessordirektive `USEGAUSSIANCONSTANTS`. Ferner: Die Konstanten lassen sich mit sehr geringem Aufwand bestimmen, sie benötigen keine komplexe Vorgehensweise.). *Welcher Wert ergibt sich nun für den Skalierungsfaktor? Wieso ist dieser immer noch nicht* 1*?*
Antwort:

Hinweis: Tatsächlich ist Astrée in der Lage, komplexe Zusammenhänge zwischen Variablen wie oben geschildert zu inferrieren und damit bessere Schranken zu beweisen. Durch benutzerdefinierte Assertions ist es damit möglich, wie mit einem Modellprüfer (engl. *Model Checking*) einen Großteil funktionaler Eigenschaften zu beweisen, die über reine Intervallarithmetik hinausgehen. Um den Arbeitslast zu reduzieren, muss diese Betrachtung in diesem Semester leider entfallen. Es kann jedoch sehr lehrreich sein, einen Blick auf die beiliegenden Beispiele (Help ⇒ Welcome ⇒ View Examples) zu werfen, um eine Intuition für Möglichkeiten und Grenzen der vollautomatischen Verifikation mittels Abstrakter Interpretation zu gewinnen, auch wenn dies für die Bearbeitung dieser Aufgabe nicht erforderlich ist.

Ringpuffer

Moderne Mikrocontroller stellen häufig eine DMA-Einheit zur Verfügung, die Peripheriedaten *asynchron* zum Prozessor beschaffen kann. Deswegen sollen in dieser Aufgabe die von der Sensorhardware gelieferten Daten in Schüben verarbeitet werden. Hierfür stellen wir Ihnen eine Implementierung eines Ringpuffers bereit.

☞ `bndbuf.c`, `bndbuf.h`

Aufgabe 7 FRAMA-C

Weisen Sie nun mittels FRAMA-C Aspekte der funktionalen Korrektheit der gegebenen Implementierung nach. Betrachten Sie in dieser Teilaufgabe zunächst nur die Funktionen aus dem Namensschema `bnd_*`, und ignorieren Sie zunächst die Funktionen der Gruppe `framac_*`.

Dabei können Sie mittels `make frama-c-gui` die graphische Oberfläche mit den empfohlenen Parametern starten, die Befehlszeilenvariante von FRAMA-C starten Sie mittels `make frama-c`. Wie bereits eingangs erwähnt empfehlen wir aufgrund der besseren Informationsaufbereitung dringend die Nutzung der graphischen Oberfläche.

Beweisen Sie nun die folgenden Eigenschaften:

- Die Füllstandsvariable `count` verlässt nie die Pufferkapazität.
- Die Füllstandsvariable `count` entspricht der Differenz (unter Berücksichtigung der Umlaufsemantik an den Puffergrenzen) zwischen Schreib- und Lesezeiger.
- Falls der Puffer nicht vollständig belegt ist, speichert `bnd_put` das Element tatsächlich im Puffer ab.

Gehen Sie dabei wie folgt vor: Definieren Sie zunächst die durch uns vorgegebenen Prädikate und definieren Sie zunächst die Vor- und Nachbedingungen der Methoden im Modul `src/bndbuf.c`, welche für die Abwesenheit von Laufzeitfehlern erforderlich sind. Erweitern Sie dann die Nachbedingungen sukzessive um die oben geforderten Eigenschaften bzw. die mit ihnen verknüpften Prädikate und axiomatisieren Sie die Funktionen entsprechend.

Wie unterscheidet sich der Verifikationsansatz mittels FRAMA-C von der Verifikation mittels Astrée?

Antwort:

Aufgabe 8 *Einfache Axiomatisierung von Datenstrukturen*

Im Modul `src/bndbuf.c` finden sich die Funktionen `framac_axiom_new_push_pop` und `framac_axiom_new_repeated_push_pop`. Welche Eigenschaft wird durch den erfolgreichen Beweis für die Ringpufferimplementierung jeweils nachgewiesen? Welche Aussage ist stärker und damit aussagekräftiger?
Antwort:

Beweisen Sie nun die Gültigkeit der beiden als Funktionen formulierten Aussagen (d.h. die relevanten `assert`- und `ensures`-Klauseln), indem Sie die beiden Funktionen sowie die Beschreibung der aufgerufenen Ringpuffermethoden erweitern und verfeinern, sowie weitere Informationen über die Schleife zur Verfügung stellen.

☞ `behaviors`
☞ Invarianten

Erweiterte Übung

Aufgabe 9 *Fortgeschrittene Axiomatisierung von Datenstrukturen*

Welcher Zusammenhang besteht zwischen den in Teilaufgabe 8 betrachteten Funktionen und einem allgemeinen Verifikationsprozess für unsere Ringpufferdatenstruktur? Welche übergeordnete Beweisidee liegt der Datenstrukturbeschreibung mittels der Axiome zugrunde?
Die in Teilaufgabe 8 betrachteten Aussagen sind dabei eher grundlegend. Generalisieren Sie die dort getroffenen Aussagen über den Zusammenhang von `bnd_put` und `bnd_get` zunächst informell in Textform. Schreiben Sie anschließend eine eigenen, formale „Testfallformulierung“, die den von Ihnen so beschriebenen allgemeineren Zusammenhang als Beweisproblem in FramaC ausdrückt (d.h. C-Code mit `ensures`, `assert` und so erforderlich `requires`-Annotationen). Ein formaler Beweis dieses neuen Axioms ist aufgrund der potentiell hohen Komplexität **nicht** erforderlich. (Sie können zur Abgabe den entsprechenden Codeteil beispielsweise durch einen `#if 0`-Block vor der Analyse verbergen).

Antwort:

Hinweise

- Bearbeitung: Gruppenarbeit
- Abgabefrist: 01.02.2023
- Fragen bitte an `i4ezs@lists.cs.fau.de`