

Betreff: Re: [IWSC2013] Stand

Von: Daniel Meyer <Daniel3.Meyer@st.ovgu.de>

Datum: 2/5/13 6:13 PM

An: sansschul@tu-braunschweig.de

hey,

So ich hab hier nochmal die Gedanken beim drübergucken notiert.

Insgesamt isses doch egal welche obfuscation als erstes kommt, da hat ich dann doch umwege gefunden.

Das Umgehen von Inversen is entweder dadurch umgangen, dass die Transformationen sich eh nicht behindern haben, und man sie nacheinander machen konnte.

Oder das man (bei den wo man immer wieder rein muss) werte abgespeichert wurden, um ziemlich genau festzulegen, dass sie schon transformiert wurden.

Hier noch kurz meine notizen von eben beim reüssieren meiner Implementation 😊

Expansion (der erste teil jedenfalls, also eliminierung von Assignment operatoren) und Contraction gleichzeitig

werden in einem Schwung in einer datei gesucht, da die nur eine zeile betreffen können werden in listen gespeichert

Refactoring setzt dann immer beim letzten element (von der Startposition her)

an, transformiert das assignment entspricht seiner listen zughörigkeit, und löscht das element aus der liste

in welche liste ein assignment gehört, kommt auf sein AST Aufbau an, das wird vorher unterschieden

Dann folgt Expansion Teil 2, also das ausbauen von Increment und Decrement ausdrücken

Trennung dadurch, dass ich immer nach den relativ nahsten möglichen AST Nodes gesucht habe, um nicht für jedes einzelnen jede node durchgehen zu müssen

Teil 1 sucht nach Assignment Teil 2 nach Pre- bzw. Postfix Expressions

Hier wird direkt nach der Veränderung die datei neugeladen,

die node wird in einer hashmap gespeichert hier compilation unit und startposition abgefragt werden, ob das schon vorhanden is.

Die startpositions können sich hier nicht ändern, aufgrund des ständigen neu ladens der datei.

Über den schon transformierten ändert sich logischerweise nichts mehr.

Dann folgt conditional refactoring

Hier wird wieder in einem schwung gesucht

hier wird wieder gesucht welches schon shorthand is und welches zu einem shorthand gemacht werden kann.

Beide werden wieder in listen gesetzt, welches als letztes kommt wird als erstes transformiert Änderung daher möglich, da die startposition bei beiden gleich bleibt wenn man von unten kommt

als letztes Loop

ähnlich expansion teil 2.

es wird nach While und ForStatements gesucht, nach einer änderung muss die datei neu geladen werden.

geänderte werden wieder in einer hashmap gespeichert, wobei der key zusammengesetzt is aus den loop initialisierer sozusagen, dem typ der schleife, und der datei

Hier musste das ganze etwas komplexer gespeichert werden, da z.b. Loops in Loops probleme ergeben hätten.

Die speicherungen bei 2. und 4. kümmern sich außerdem dadrum, dass man in keiner endlosschleife landet.

Ansonstne würde z.b. die 1. for loop zur while. Die datei wird neu geladen, und aus der while wird wieder einer for usw.

Hoffe das sollte die Frage klären

Ciao
Daniel

On 05.02.2013 15:59, Sandro Schulze wrote:

Hi Daniel,

danke erst einmal fuer deine Anmerkungen. Mit den falschen Werten im Diagramm war einfach das Problem, dass ich bei den Tabellen immer von der gleichen Reihenfolge der Obfuscations ausgegangen ist. Aber die Tabellen waren ja absteigend nach similarity geordnet. Ist jetzt aber korrigiert 😊

Eine Frage hab ich noch bzgl. der obfuscations: Wie genau wurde die Reihenfolge bei den Kombinationen festgelegt? Einfach per Zufall? Generell spielen die ja keine Rolle, aber ich wollte die Info schon einbauen.

Und dazu noch eine Frage: Wie bist du denn mit inversen Obfuscations, z.B. Extraction/Compression, umgegangen? Es kann ja sein, dass ich an einem Knoten Extraction mache, dann muss ich mir doch merken, dass ich an dem gleichen Knoten keine compression mache oder? Selbst, wenn du den ganzen AST nur genau einmal durchläufst, so wird ja doch an jedem Knoten auf alle Obfuscations getestet, oder?

Ansonstne bich ich schwer am rotieren 😊 Waere cool, wenn du die Fragen heute noch beantwortest

Gruss
Sandro