1 Semantische Evaluation

Das Ziel der semantischen Evaluation ist es, einen der Proxies, die im Rahmen der 1. Stufe der Exploration erzeugt wurden, hinsichtlich der vordefinierten Testfälle zu evaluieren. Da die gesamte Exploration zur Laufzeit des Programms durchgeführt wird, stellt sie hinsichtlich der nicht-funktionalen Anforderungen eine zeitkritische Komponente dar.

Da die Anforderungen an die gesuchte Komponente mit bedacht spezifiziert werden müssen, ist es irrelevant, ob es mehrere Proxies gibt, die den vordefinierten Testfällen standhalten. Vielmehr soll bei der semantischen Evaluation lediglich ein Proxy gefunden werden, dessen Semantik zu positiven Ergebnissen hinsichtlich aller vordefinierten Testfälle führt. Somit wird die semantische Evaluation beendet, sobald ein solcher Proxy gefunden ist.

Bei der Exploration soll letztendlich in einer Bibliothek L zu einem vorgegebenen required Type R ein Proxy gefunden werden. Die Menge dieser Proxies wurde im vorherigen über $cover(R, L)$ beschrieben. Die in dieser Menge befindlichen Proxies können eine unterschiedliche Anzahl von Target-Typen enthalten.

Das in dieser Arbeit beschriebene Konzept basiert auf der Annahme, dass bei der Entwicklung davon ausgegangen wird, dass der gesamte Anwendungsfall - oder Teile davon - , der mit der vordefinierten Struktur und den vordefinierten Tests abgebildet werden soll, schon einmal genauso oder so ähnlich in dem gesamten System implementiert wurde. Aus diesem Grund kann für die semantische Evaluation grundsätzlich davon ausgegangen werden, dass die erfolgreiche Durchführung aller relevanten Tests umso wahrscheinlicher ist je weniger Target-Typen im Proxy verwendet werden.

Somit werden zuerst die Proxies auf ihr semantisches Matching überprüft, in denen lediglich ein Target-Typ verwendet wird. Die Menge der Proxies aus einer Menge von Proxies P mit einer Anzahl A von Target-Typen wird durch folgende Funktion beschrieben:

$$proxiesMitTargets(P, A) := \{P | P.targetCount = A\}$$

Die maximale Anzahl der Target-Typen in einerm Proxy zu einem required Typ R ist gleich der Anzahl der Methoden in P.

$$maxTargets(R) := |methoden(R)|$$

So kann der Algorithmus für die semantische Evaluation der Menge P von Proxies, die für einen required Typ R erzeugt wurden, mit der Menge von Testfällen T wie folgt im Pseudo-Code beschrieben werden. Dabei sei davon auszugehen, dass ein Test aus T mit einem Proxy p über eine Methode `eval(p)` ausgewertet werden kann. Diese Methode gibt bei erfolgreicher Durchführung den Rückgabewert `true` und anderenfalls `false` zurück.

```
function semanticEval(R, P, T){
        for( i = 1; i <= maxTargets(R); i++ ){
                proxy = evalProxiesMitTarget(P,i,T)
                if( proxy != null ){
                        // passenden Proxy gefunden
                        return proxy
                }
        }
        // kein passenden Proxy gefunden
        return null;
}

function evalProxiesMitTarget(P,anzahl,T){
        for( proxy : relevantProxies(P,anzahl) ){
                if( evalProxy(proxy, T) ){
                        // passenden Proxy gefunden
                        return proxy
                }
        }
        // kein passenden Proxy gefunden
        return null
}

function relevantProxies(P,anzahl){
        return proxiesMitTargets(P,anzahl);
}

function evalProxy(proxy, T){
        for( test : T ){
                if( !test.eval(proxy) ){
                        \\ wenn ein Test fehlschlaegt, dann
                           entspricht der
                        \\ Proxy nicht den semantischen
                           Anforderungen
                        return false
                }
        }
        return true
}
```

Die Dauer der Laufzeit der oben genannten Funktionen hängt maßgeblich von der Anzahl der Proxies PA ab. Im schlimmsten Fall müssen alle Proxies hinsichtlich der vordefinierten Tests evaluiert werden.

Die Heuristiken, die in den folgenden Abschnitten vorgestellt werden, sollen im Allgemeinen die Anzahl der zu prüfenden Proxies reduzieren. Sie werden immer innerhalb der Methode `relevantProxies` angewendet. So kann diese

Methode wie in folgendem Listing erweitert werden. Die jeweilige Heuristik wird dann über die Methode `applyHeuristic` beschrieben.

```
function relevantProxies(P,anzahl){
        proxies = proxiesMitTargets(P,anzahl)
        optmizedProxies = applyHeuristic(proxies)
        return optmizedProxies
}
```

1.1 Ratingbasierte Heuristiken

Die folgenden Heuristiken haben zum Ziel, die Reihenfolge, in der die Proxies hinsichtlich der vordefinierten Tests evaluiert werden, so anzupassen, dass ein passender Proxy möglichst früh evaluiert wird. Dabei dient jeweils ein Rating der Proxies als Kriterium für die Festlegung der Reihenfolge.

1.1.1 Qualitatives Rating

Bei dem so genannten *qualitative Rating* eines Proxies handelt es sich um einen numerischen Wert. Um diesen Wert zu ermitteln, wird für jeden Matcher ein Basisrating vergeben. Folgende Funktion beschreibt das Basisrating für das Matching zweier Typen S und T:

$$base(S,T) = \left\{ \begin{array}{l|l} S \Rightarrow_{exact} T & 100 \\ S \Rightarrow_{gen} T & 200 \\ S \Rightarrow_{spec} T & 200 \\ S \Rightarrow_{contained} T & 300 \\ S \Rightarrow_{container} T & 300 \end{array} \right\}$$

Dabei ist zu erwähnen, dass einige der o.g. Matcher über dasselbe Basisrating erfügen. Das liegt daran, dass sie technisch jeweils gemeinsam umgesetzt wurden.[1]

Das qualitative Rating eines Proxies P wird über die Funktion $qualRating(P)$ beschrieben. Dieses ist von dem qualitativen Rating der Methoden-Delegation innerhalb des Proxies P abhängig. Das qualitative Rating einer Methoden-Delegation ist von den Basisratings der Matcher abhängig, über die die Parameter- und Rückgabe-Typen der aufgerufenen Methode und der Delegationsmethoden gematcht werden können. Das qualitative Rating einer

[1] Der *GenTypeMatcher* und der *SpecTypeMatcher* wurden gemeinsam in der Klasse `GenSpecTypeMatcher` umgesetzt. Der *ContentTypeMatcher* und der *ContainerTypeMatcher* wurden gemeinsam in der Klasse `WrappedTypeMatcher` umgesetzt. (siehe angehängter Quellcode)

Methoden-Delegation MD soll über die Funktion $mdRating(MD)$ beschrieben werden.

Für die Definition der beiden Funktionen $qualRating(P)$ und $mdRating(MD)$ gibt es unterschiedliche Möglichkeiten. In dieser Arbeit werden 4 Varianten als Definitionen vorgeschlagen, die in einem späteren Abschnitt untersucht werden.

Für die Vorschläge zur Definition von $qualRating(P)$ sei P ein struktureller Proxy mit n Methoden-Delegation. Darüber hinaus gelten für die Definition von $mdRating(MD)$ für eine Methoden-Delegation MD folgende verkürzte Schreibweisen:

$$
\begin{aligned}
pc &:= MD.call.paramCount \\
cRT &:= MD.call.returnType \\
dRT &:= MD.del.returnType \\
cPT &:= MD.call.paramTypes \\
dPT &:= MD.del.paramTypes \\
pos &:= MD.call.posModi
\end{aligned}
$$

Weiterhin seien die folgenden Funktionen gegeben:

$$
\begin{aligned}
basesMD(MD) = \quad & base(dRT, cRT) \cup base(cPT[0], dPT[pos[0]]) \\
& \cup ... \cup base(cPT[pc], dPT[pos[pc]])
\end{aligned}
$$

$$
sum(v_1, ...v_n) = \sum_{i=1}^{n} v_i
$$

$$
max(v_1, ..., v_n) = v_m | 1 \leq m \leq n \wedge \forall i \in \{1, ..., n\} : v_i \leq v_m
$$

$$
min(v_1, ..., v_n) = v_m | 1 \leq m \leq n \wedge \forall i \in \{1, ..., n\} : v_i \geq v_m
$$

Variante 1: Durchschnitt

$$mdRating(MD) = \frac{sum(basesMD(MD))}{pc + 1}$$

$$qualRating(P) = \frac{sum(mdRating(P.dels[0]), ..., mdRating(P.dels[n-1]))}{n}$$

Variante 2: Maximum

$$mdRating(MD) = max(basesMD(MD))$$

$$qualRating(P) = \frac{max(mdRating(P.dels[0]), ..., mdRating(P.dels[n-1]))}{n}$$

Variante 3: Minimum

$$mdRating(MD) = min(basesMD(MD))$$

$$qualRating(P) = \frac{min(mdRating(P.dels[0]), ..., mdRating(P.dels[n-1]))}{n}$$

Variante 4: Durchschnitt aus Minimum und Maximum

$$mdRating(MD) = \frac{max(basesMD(MD)) + min(basesMD(MD))}{2}$$

$$qualRating(P) = \frac{\begin{array}{l} max(mdRating(P.dels[0]), ..., mdRating(P.dels[n-1])) \\ \quad + min(mdRating(P.dels[0]), ..., mdRating(P.dels[n-1])) \end{array}}{2}$$

Da die Funktion *qualRating* von *mdrating* abhängt und für *mdrating* 4 Variante gegeben sind, ergeben sich für jede gegebene Variante für die Definition von *qualRating* weitere 4 Varianten. Dadurch sind insgesamt 16 Varianten für die Definition von *qualRating* gegeben.

Zur Anwendung der Heuristik muss das qualitative Rating bei der Auswahl der Proxies in der semantischen Evaluation beachtet werden. Die erfolgt innerhalb der Methode `applyHeuristic(proxies)`. Für diese Heuristik sei dazu eine Methode `sort(proxies, rateFunc)` angenommen, die eine Liste zurückgibt, in der die Elemente in der übergebenen Liste `proxies` aufsteigend nach den Werten sortiert, die durch die Applikation der übergebenen

Funktion `rateFunc` auf ein einzelnes Element aus der Liste `proxies` ermittelt werden. Darauf aufbauend wird die Methode `applyHeuristic(proxies)` für diese Heuristik in Pseudo-Code wie folgt definiert:

```
function applyHeuristic(proxies){
        return sort(proxies, qualRating)
}
```

1.1.2 Quantitatives Rating

1.2 Testergebnisbasierte Heuristiken

Die folgenden Heuristiken haben zum Ziel, basierend auf den fehlgeschlagenen Evaluationen von Proxies, darauf schließen, dass auch die Evaluation anderer Proxies fehlschlägt. Dadurch werden diese Proxies bei der weiteren Evaluation herausgefiltert.