Attributierte Klassifikationsbäume zur Testdatenbestimmung

Stefan Lützkendorf
Institut für Terminologie und angew. Wissensforschung GmbH
stefan.luetzkendorf@web.de

Klaus Bothe
Humboldt-Universität zu Berlin
Institut für Informatik
bothe@informatik.hu-berlin.de

Zusammenfassung

Die Klassifikationsbaummethode dient der systematischen Analyse des Eingabedatenbereichs eines Testobjektes. Dabei wird der Eingabedatenbereich unter verschiedenen Aspekten klassifiziert. Testfälle werden dann durch Kombination der Klassen der verschiedenen Aspekte definiert. Den folgenden Schritt – die Bestimmung der konkreten Testdaten – unterstützt die Methode nur insoweit, als dass sie eine textuelle Testfallspezifikation liefert, die *manuell* zu interpretieren und in Testdaten umzusetzen ist. Diese Lücke zwischen der Testfallspezifikation der Klassifikationsbaummethode und der konkreten Testdatenauswahl soll durch die Anreicherung der Klassifikationsbäume mit Attributen geschlossen werden.

1 Die Klassifikationsbaummethode

Die Klassifikationsbaummethode[3] ist eine Methode zur systematischen Bestimmung von Testfällen. Dabei wird der Bereich der Eingabedaten eines Testobjektes unter verschiedenen testrelevanten Aspekten untersucht und in Klassen zerlegt. Die gefundenen Klassen können wiederum unter verschiedenen Aspekten klassifiziert werden. So entsteht ein Baum aus Klassen und Klassifikationen – ein Klassifikationsbaum. Testfälle werden dann mit Hilfe des Klassifikationsbaums gebildet, indem die einzelnen Klassen der verschiedenen Aspekte miteinander kombiniert werden.

Abb. 1 zeigt einen Klassifikationsbaum für ein Beispieltestobjekt sowie eine Kombinationstabelle mit 3 Testfällen. Das Testobjekt ist eine Funktion zur Häufigkeitsbestimmung eines Elements in einer Liste, d. h., eine Funktion mit folgender Signatur:

```
Integer count(List list, Element elem)
```

Die Anwendung dieser Funktion auf das Element `'g'` und die Liste `{'a','c','g','e','g'}` liefert also 2.

Durch den Klassifikationsbaum werden die Testdaten zunächst nach den Länge den Liste, in der gesucht werden soll, getrennt. Es werden leere Listen, einelementige und längere Listen unterschieden. Bei einelementigen Listen wird das Ergebnis durch die Frage entschieden, ob das eine Element in der Liste das gesuchte ist oder nicht. Bei Listen, die länger als ein Element sind, sollen zwei verschieden Aspekte berücksichtigt werden: zum einen, ob bzw. wie häufig das gesuchte Element in der Liste auftaucht, und zum anderen, wie die Elemente der Liste geordnet sind.

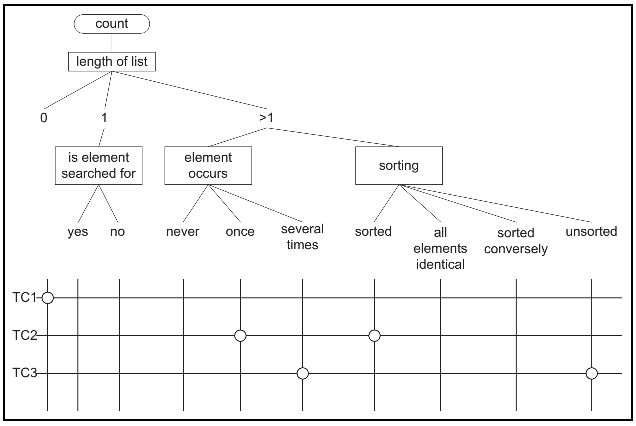

Abb. 1: Beispiel

Bestimmung der Testdaten Nachdem der Klassifikationsbaum konstruiert wurde, können mit Hilfe der Kombinationstabelle Testfälle bestimmt werden. So soll z. B. im Testfall 2 (TC2) in sortierten Listen, die länger als ein Element sind, gesucht werden und das gesuchte Element soll genau einmal enthalten sein.

Die Klassifikationsbaummethode liefert dann für jeden der bestimmten Testfälle eine Testfallspezifikation. D. h., eine textuelle Beschreibung der die Testdaten eines Testfalls bestimmenden Klassen und Klassifikationen:

```
length of list: > 1
  element occurs: once
  sorting: sorted
```

Aus einer solchen Testfallspezifikation kann man dann Testdaten ableiten, die den beschriebenen Bedingungen entsprechen. Z. B. könnte für Testfall 2 die Wahl wie folgt aussehen:

```
elem: 'g'
list: {'a','b',g','x'}
```

2 Das Problem

Die konkreten Testdaten werden *von Hand* ausgewählt (Abb. 2) und in einem separaten Dokument abgelegt.

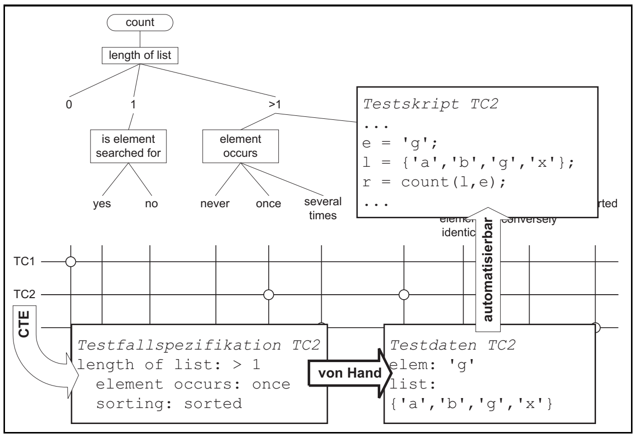

Abb. 2: Das Problem – Die Lücke

Damit verbunden sind zwei Probleme:

a) Aufgrund der Existenz zweier unabhängiger Dokumente - Klassifikationsbäume und Testdatensammlung - entsteht eine semantische Lücke zwischen der logischen Beschreibung von Testfällen und der Auswahl konkreter Testdaten, wodurch insbesondere die konsistente Wartung erschwert wird.

b) Während die Testdaten natürlicherweise direkt als Eingabe für eine Testautomatisierung herangezogen werden, spielen Klassifikationsbäume als Eingabe eines Testsystems keine direkte Rolle.

Beide Probleme sollen durch die im nächsten Abschnitt beschriebene Attributierung von Klassifikationsbäumen auf möglichst allgemeine Art gelöst werden.

Die vom Tester zu bestimmenden Testdaten werden in unserem Ansatz nicht auf eine bestimmte Form beschränkt: Denkbar sind z. B. Programmcode zur Testausführung, Daten in Tabellen- oder XML-Form, die von einem Testrahmen zur Testdurchführung verwendet werden, oder auch Skripte eines Fernsteuerungstools für den Test graphischer Oberflächen.

Grundlage für die Entwicklung eines Tools, das die im folgenden zu beschreibenden attributierten Klassifikationsbäume auswerten kann, war die Existenz eines Werkzeuges, das die Erstellung von Klassifikationsbäumen unterstützt. Hierbei konnte auf den *Classification Tree Editor (CTE)* zurückgegriffen werden [4, 5].

3 Attributierte Klassifikationsbäume

Am Anfang der Testdatenbestimmung steht die Überlegung, welche Daten zur Durchführung des Tests benötigt werden. Für das Beispieltestobjekt – die `count`-Funktion – lautet die Antwort: es wird zum einen die Liste in der gesucht werden soll benötigt und zum anderen das Element dessen Vorkommen zu zählen ist.

Attribute an den Testfällen Im Anfangsschritt auf dem Weg der Zusammenführung von Klassifikationsbaum und Testdatenauswahl könnte man die erforderlichen Testdaten für jeden Testfall einzeln bestimmen und mit den Testfällen verknüpfen. Abb. 3 zeigt für die drei Beispieltestfälle diese Attribute, die die Testdaten charakterisieren.

Damit ist man gegenüber der bisherigen Art der Testdatenbestimmung noch keinen Schritt weiter, da für die Bestimmung immer noch nicht der Baum mit seiner Struktur und seiner Semantik Verwendung findet. Obwohl der Klassifikationsbaum die Testdaten – z.T. sogar eindeutig – bestimmt, wird diese Information nur mittelbar verwendet. Außerdem wird die Kombination der Klassen nicht für die Kombination der Testdaten wiederverwendet. Was aber an dem Beispiel erkennbar ist, ist dass zur Durchführung des Tests an jedem Testfall eine bestimmte Menge von Attributen zur Verfügung stehen muss; im Beispiel eben die Attribute elem und list, die die Parameter der Funktion darstellen.

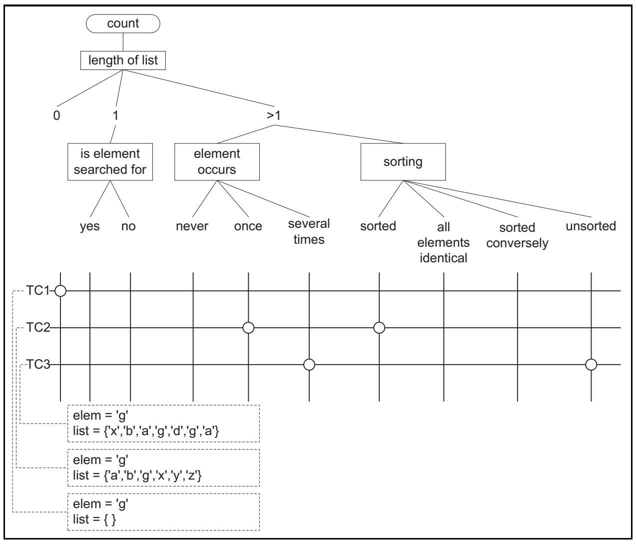

Abb. 3: Attribute an den Testfällen

Attribute am Klassifikationsbaum Um einen Schritt weiterzukommen, kann man versuchen, die Informationen, die für einen Testfall erforderlich sind, aus den Klassen abzuleiten, die den Testfall bilden, bzw. durch deren Kombination der Testfall gebildet wurde.

Die Klasse '0' des Beispiels bestimmt z. B. die Testdaten derart, dass für list die leere Liste gewählt werden muss und für elem ein beliebiges Element aus dem Eingabedatenbereich gewählt werden kann. Bringt man dieses Wissen in Form von Attributen an der Klasse an, steht an jedem Testfall, der mit Hilfe dieser Klasse gebildet wird, diese Information zur Verfügung. Dies ist bei der Klasse '0' noch nicht besonders wirkungsvoll, da sie nicht mit anderen Klassen kombiniert werden kann und daher nur in einem Testfall wirklich sinnvoll Verwendung finden wird. In komplexeren Beispielen weiter unten aber wird der Sinn dieser Attribute augenscheinlich.

Ähnliches wie für die Klasse '0' gilt für die Klassen 'yes' und 'no'. In Abb. 4 sind folglich diese Klassen mit entsprechenden Attributen versehen. Die Attribute an Testfall 1 konnten entfernt werden, da sie durch die ihn konstituierende Klasse '0' zur Verfügung gestellt werden.

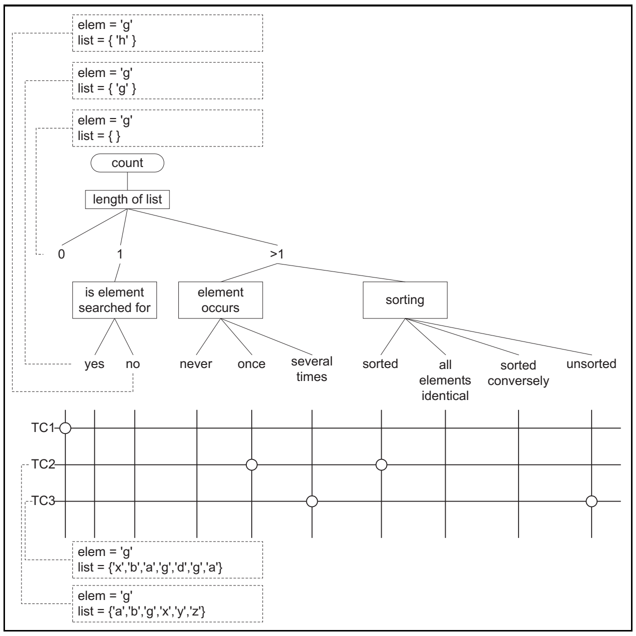

Abb. 4: Attribute an den Blättern

Vererbung In Abb. 4 kann noch eine weitere Beobachtung gemacht werden: es gibt Attributdefinitionen, die sich bei mehreren oder auch bei allen Klassen gleichen. Im Beispiel taucht die Attributdefinition elem = 'g' bei allen Testfällen auf. Das hier stets nach dem gleichen Element gesucht wird, ist aus methodischen Gründen sehr sinnvoll, da die Aspekte, unter denen die Eingabedaten klassifiziert wurden, im wesentlichen nur Aspekte der Liste, in der gesucht werden soll, sind. Um die Wiederholung von gleichen Attributdefinitionen zu vermeiden, bietet sich bei einem Baum die Verwendung von Vererbung an. Man kann so die Attributdefinition für elem = 'g' an die Wurzel des Baums verschieben, so dass sie in allen Testfällen sichtbar ist, ganz gleich durch welche Klassen sie definiert werden. Abb. 5 zeigt den veränderten Baum.

Ausdrücke in Attributen Die bisher verwendeten Attribute waren stets Konstanten. Um die Testdaten für Testfälle mit mehrelementigen Listen auch aus At-

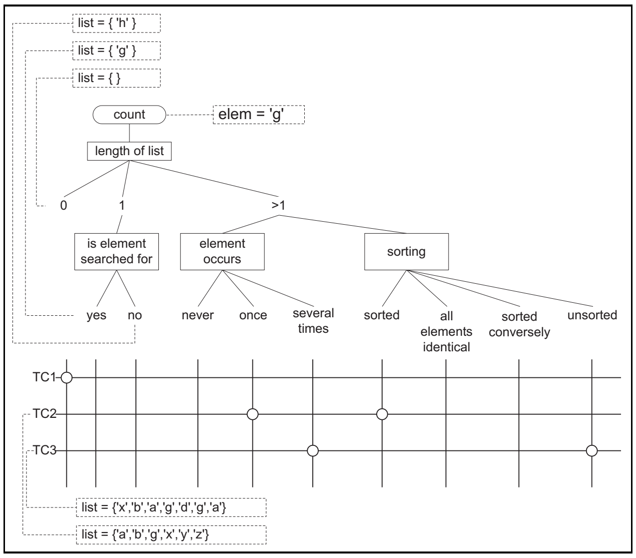

Abb. 5: Attribute an den beliebigen Knoten, Vererbung

tributen des Klassifikationsbaums ableiten zu können, benötigt man ein mächtigeres Mittel.

Für dieses Problem sollen Ausdrücke zur Verfügung gestellt werden, mit deren Hilfe Attributwerte aus den Werten anderer Attribute berechnet werden können.

Zu unserem Beispiel: Hier soll zu den Aspekten *Häufigkeit des zu zählenden Elements* und *Ordnung der Listenelemente* eine Liste konstruiert werden, die den Klassen beider Aspekte genügt. Der Lösung in Abb. 6 liegt folgende Überlegung zugrunde: Die zu konstruierende Liste enthält eine Teilliste die die gesuchten Elemente enthält (hits) und zwei weitere Teillisten (pre und post) die als Präfix bzw. Postfix der zu konstruierenden Liste derart ausgewählt werden, dass sie in Verbindung mit der ersten Liste die Anforderungen aus dem Sortierungs-Aspekt erfüllen. Mit dem Ausdruck list = pre + hits + post kann dann aus den Teillisten eine Liste konstruiert werden, die unter beiden Aspekten den Anforderungen genügt.

Betrachtet man nun wiederum den Testfall 2, stehen zur Bestimmung seiner Testdaten – der Parameter elem und list – u.a. folgende Attribute zur Verfügung:

hits = {'g'} wird direkt durch die Klasse 'once' zur Verfügung gestellt.

occ = 1 dito (Verwendung wird unten erläutert).

pre = {'a','b'} wird direkt durch die Klasse 'sorted' zur Verfügung gestellt.

post = {'x','y','z'} dito.

list = pre + hits + post wird indirekt von der Klasse 'sorted' zur Verfügung gestellt, die diese Attributdefinition von der Klassifikation 'sorted' einerbt.

elem = 'g' wird von beiden Klassen 'once' und 'sorted', die die Definition vom Wurzelknoten erben, bereitgestellt.

Mit Hilfe dieser Attributdefinitionen kann der Wert des Attributs list berechnet werden, unter der Vor-

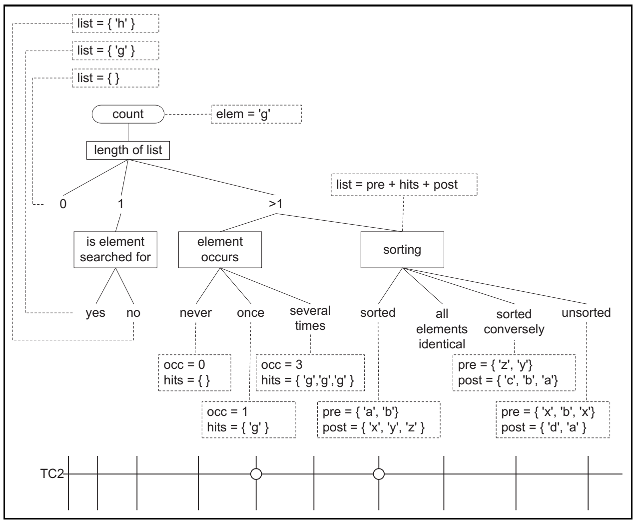

Abb. 6: Ausdrücke in Attribute

aussetzung, dass der Plus-Operator die Verkettung von Listen realisiert. Der sich ergebende Wert {'a','b','g','x','y','z'} entspricht allen Anforderungen der Testfallspezifikation (s.o.): die Liste ist länger als 1, ist sortiert und das zu zählende Element ist einmal enthalten. Das Attribut elem steht als Konstante zur Verfügung, so dass alle erforderlichen Testdaten für Testfall 2 vorhanden sind.

Zusicherungen, Überschreibung Um das Beispiel zu vervollständigen, sind noch die Attribute für die Klasse 'all elements identical' zu definieren. Hier stößt man zuerst auf ein Problem, dass in der Praxis öfter auftritt: Klassen verschiedener Aspekte, die nicht voneinander unabhängig sind. Im Beispiel schließen sich die Klassen 'all elements identical' und 'once' gegenseitig aus. Wenn die Liste das zu suchende Element genau einmal enthält und keine anderen Elemente, kann sie nicht länger als eins sein.

Um solche ungültigen Kombinationen zu verhindern, werden von dem Test-Tool zur Auswertung der attributierten Klassifikationsbäume spezielle Attribute unterstützt: Zusicherungen. Zusicherungen sind normale Attribute, wie sie bisher auch Verwendung fanden, d.h., sie können konstante Werte oder Ausdrücke enthalten. Sie unterscheiden sich von den anderen Attributen ausschließlich durch ein Namensprefix, das per Konvention assert lautet. Der Wert dieser Attribute muss zu wahr evaluieren. Andernfalls gibt das Tool eine Warnung zum entsprechenden Testfall aus.

Nachdem ausgeschlossen wurde, dass 'all elements identical' und 'once' kombiniert werden können, kann die Definition für das list-Attribut erstellt werden. In diesem Beispiel wird dazu ein weiteres Hilfsattribut I eingeführt, das in Abhängigkeit davon, ob eine Kombination mit 'several times' oder mit 'never' vorliegt (getestet über den Wert des Hilfsattributes occ), den Wert des gesuchten Elementes annimmt oder einen beliebigen anderen. Mit Hilfe dieses Hilfsattributes wird der Wert von list bestimmt.

Hier wird also im Sinne der Spezialisierung die Definition des Attributs list in der Klasse 'all elements identical' überschrieben, die von der der Klassifikation 'sorting' ererbt wurde.

Das nun vollständige Beispiel ist in Abb. 7 zu sehen.

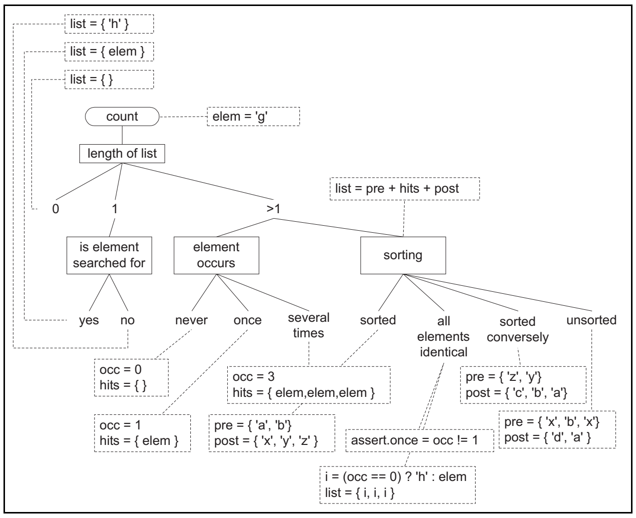

Abb. 7: Vollständiges Beispiel

Ausgabedaten Bisher sind die Attribute zur Bestimmung der Testdaten verwendet worden. Der Klassifikationsbaum beschreibt diese, und die Attributierung macht es möglich, eine Auswahl bestimmter Testdaten durchzuführen. Ein konkreter Satz Testdaten bestimmt aber zusammen mit der Spezifikation auch die zu erwartenden Ausgabedaten. Daher ist es naheliegend, auch diese zusammen mit den Testdaten im Klassifikationsbaum zu verwalten. Abb. 8 zeigt das Beispiel ergänzt um result-Attribute. Damit wird es möglich, nicht nur die Testausführung mit den Testdaten zu versorgen, sondern auch einem Vergleicher die Auswertung des Tests zu erlauben.

Zusammenfassung Nachdem das Prinzip der attributierten Klassifikationsbäume am Beispiel erläutert wurde, soll nun eine etwas formalere Zusammenfassung erfolgen.

Zur Bestimmung der Testdaten für einen Testfall stehen folgende Attribute zur Verfügung:

1. Alle Attribute aller Klassen, die zur Definition des Testfalls kombiniert wurden, einschließlich aller Attribute, die die Klassen von ihren Vorgängerknoten erben.
   Gleichnamige Attributdefinitionen untergeordneter Knoten überschreiben die Definitionen von übergeordneten Knoten.

2. Alle Attribute der Testfallgruppe oder -sequenz, in der der Testfall enthalten ist.
   Diese Attribute überschreiben gleichnamige Attribute der Klassen.

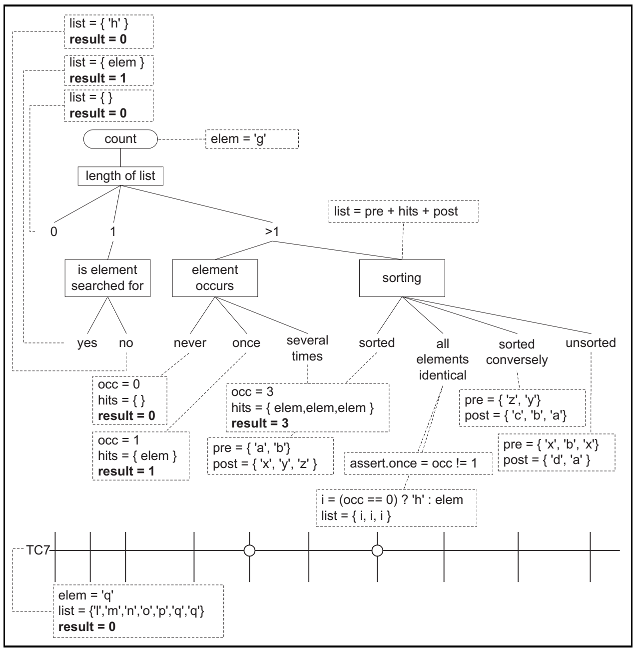

Abb. 8: Ausgabedaten

3. Alle Attribute des Testfalls selbst.
   Diese können wiederum Attributdefinitionen entsprechend 1. oder 2. überschreiben.

Zusätzlich sind folgende Punkte zu beachten:

4. Die Attributdefinitionen aller Klassen, die in einem Testfall kombiniert werden (können), müssen widerspruchsfrei sein.

5. Die Attributdefinitionen müssen zyklenfrei sein.

Die fünf genannten Punkte sollen mit Hilfe von Abb. 9 kurz erläutert werden. Für die Testfälle gelten u. a. folgende Attributdefinitionen:

TC1: a = 1, b = 1 Die Definition a = 1 *erbt* 'class one' von der 'classification 1', b = 1 stellt die Testfallgruppe (TCG) zur Verfügung.

TC2: a = 2, b = 1 Die Definition a = 2 von 'class three' *überschreibt* die Definition von a = 1 der 'classification 1'.

TC3: a = 3, b = 1 Die Definition von a = 3 am Testfall *überschreibt* die in 'class two' geerbte Definition a = 1.

TC4: a = 4, b = 2 Die Definition b = 2 des Testfalls *überschreibt* die von b = 1 der Testsequenz.

TC5: Der Testfall ist ungültig, da sich die Definitionen a = 1 und a = 5 widersprechen.

TC6: Der Testfall ist ungültig, da die Definitionen z = 2 * y und y = z / 2 zirkulär sind.

4 Implementation eines Prototypen

Im folgenden soll kurz ein Überblick über die Struktur und Funktionsweise des Tools gegeben werden, mit dessen Hilfe die Auswertung attributierter Klassifikationsbäume möglich ist.

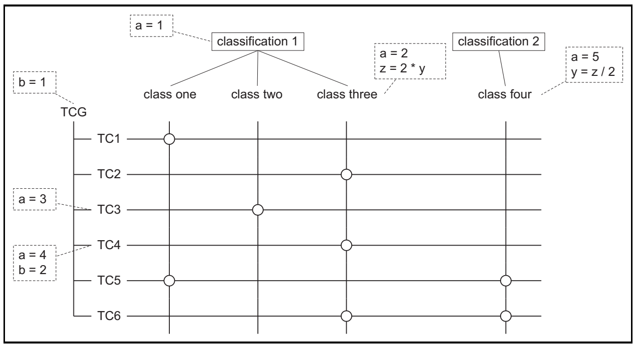

Abb. 9: Beispiel zu den Regeln

Abb. 10 zeigt die verschiedenen Komponenten. Den Ausgangspunkt stellt der *Classification Tree Editor (CTE)* dar, mit dem die graphische Erstellung der Klassifikationsbäume und die Festlegung von Testfällen in einer Kombinationstabelle möglich sind. Außerdem erlaubt der CTE die Anbringung von Attributen an den Knoten des Baumes und den Testfällen. Damit stellt er eine wesentliche Voraussetzung für die praktische Anwendung der oben beschriebenen attributierten Klassifikationsbäume dar. Ausgabe dieser Komponente ist ein attributierter Klassifikationsbaum (AKB) in Form einer wohl strukturierten Textdatei.

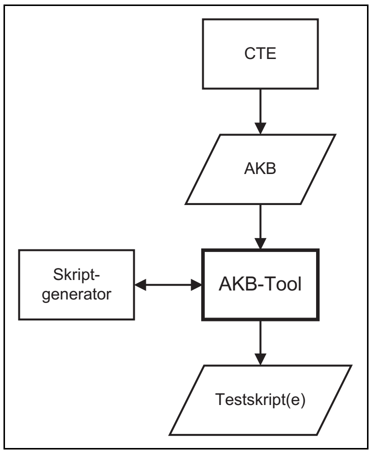

Abb. 10: Architektur des Prototypen

Diese Textdatei stellt die Eingabe des eigentlichen *AKB-Tools* dar. Dies rekonstruiert den Klassifikationsbaum aus der Eingabe und evaluiert die Attribute in der beschriebenen Form. Die so gewonnenen Informationen über einen Testfall werden mit Hilfe eines *Skriptgenerators* in eine konkrete Form verarbeitbarer Testdaten (*Testskript(e)*) überführt. Da, wie bereits erläutert, die konkrete Form der Testdaten nicht fixierbar ist, ist der Skriptgenerator ein externer Modul, der vom Anwender des Tools bereitgestellt bzw. ausgewählt werden muss. Ein Skriptgenerator-Modul muss ein sehr einfaches Interface implementieren, um mit den AKB-Tool kooperieren zu können.

Sowohl AKB-Tool als auch die Skriptgeneratoren sind im Prototyp in Perl implementiert. Daher sind die Ausdrücke in den Attributdefinitionen im Prototyp Perl-Ausdrücke, was in den Beispielen in diesem Paper aus Lesbarkeitsgründen vereinfacht wurde. Jenseits der Lesbarkeit ermöglicht dies aber die Formulierung sehr mächtiger Ausdrücke einschließlich der Verwendung von Funktionen und Sub-Routinen zur Berechnung von Testdaten.

Zur durchgängigen Realisierung einer Klassifikationsbaum-getriebenen Testautomatisierung ist das AKB-Tool in eine umfangreichere Umgebung zur automatischen Testausführung und -auswertung eingebunden worden, worauf hier aber nicht weiter eingegangen werden soll.

5 Erfahrungen einer Fallstudie

Ausgangspunkt zur Entwicklung eines Tools zur Unterstützung attributierter Klassifikationsbäume waren die Testaktivitäten im Rahmen des XCTL-Projektes an der Humboldt-Universität. Dabei handelt es sich um ein Projekt zur Pflege und Entwicklung eines Anwendungsprogramms zur Steuerung einer Röntgen-Topographie-Anlage [1, 2].

In diesem Projekt wurde anfangs Klassifikationsbäume entwickelt und die dazugehörigen Testdaten manuell erzeugt und separat gespeichert und verwaltet. Schrittweise sich ändernde Testvoraussetzungen machten Überarbeitungen der Klassifikationsbäume notwendig und demzufolge auch die Überarbeitung der Testdaten. Hierbei traten, mit der Anzahl der Überarbeitungen wachsend, Inkonsitenzen und Fehler auf. Die Häufigkeit der Überarbeitungen der Klassifikationsbäume war im konkreten Projekt wahrscheinlich höher als üblich, da es sich um ein Reverse-Engineering-Projekt handelte, in dem die Spezifikation erst schrittweise gewonnen wurde.

Nach der Entwicklung der attributierten Klassifikationsbäume und des AKB-Tools, wurden eine Reihe von Bäumen durchgängig attributiert. In der folgenden Tabelle wird der Umfang der Bäume und der Grad der Attributierung kurz für drei Testpakete (Bäume) aufgeführt.

	m_init	m_layer	m_rpl
Anz. Klassifikationen	53	55	18
Anz. Klassen	163	154	59
Anz. Knoten insg.	117	210	78
Attr. an Klassifikationen	42	20	49
Attr. an Klassen	140	162	77
Attr. insgesamt	188	196	132
Anz. Testsequenzen		41	
Anz. Testfälle/-schritte	35	503	20

Aufgrund des doch beträchtlichen Umfangs der Bäume muss hier auf eine beispielhafte Abbildung verzichtet werden. Eingesehen werden können sie aber in[6].

6 Zusammenfassung

Zusammenfassend lässt sich sagen folgendes sagen:

- Mit attributierten Klassifikationsbäumen lassen sich gut größere Testfallmengen und konsistente Testdaten erzeugen.
- Die Wartbarkeit einer Testsuite wird deutlich verbessert.
- Der zusätzliche Aufwand bei Erstellung und Wartung der attributierten Klassifikationsbäume reduziert den Aufwand zur manuellen Erstellung der Testdaten und ist daher vertretbar.
- Der CTE erlaubt eine Attributierung der Klassifikationsbäume so dass eine Basis der Tool-Unterstützung gegeben ist. Hier sind aber noch Verbesserungen denkbar, die die Pflege der attributierten Klassifikationsbäume ergonomischer und übersichtlicher machen würden.

Literatur

[1] Bothe, Klaus: *Reverse Engineering: the Challenge of Large-Scale Real-World Educational Projects. CSEE&T, 14th Conference on Software Engineering Education and Training*, Charlotte, USA, Febr. 2001.

[2] Bothe, Klaus und Ulrich Sacklowski: *Praxisnähe durch Reverse Engineering-Projekte: Erfahrungen und Verallgemeinerungen. 7. Workshop SEUH*, Zürich, CH, Febr. 2001.

[3] Grochtmann, Matthias und Klaus Grimm: *Classification Trees for Partition Testing.* Software Testing, Verification and Reliability, S. 63–82, 1993.

[4] Grochtmann, Matthias und Joachim Wegener: *Test Case Design using Classification Trees and the Classification-Tree Editor CTE.* Proceedings of the 8th International Software Quality Week, Mai 1995.

[5] Lehmann, E. und J. Wegener: *Test Case Design by means of the CTE XL.* Proceedings of the 8th european Conference on Software Testing, Analysis and Review (EuroSTAR 2000), Dezember 2000.

[6] Lützkendorf, Stefan: *Softwaretest in Reverse Engineering-Prozessen.* Diplomarbeit, Humboldt-Universität, Dezember 2001.