Automatische Testvektorgenerierung in der modellbasierten Softwareentwicklung

Dr. Thomas Hermes, Axel Schultze, Oliver Predelli

IAV GmbH
Nordhoffstr. 5
38518 Gifhorn
{dr.thomas.hermes, axel.schultze, oliver.predelli}@iav.de

Abstract: Bei der IAV GmbH wurde ein modellbasierter Testfallgenerator entwickelt, mit dem es möglich ist, aus verifizierten MATLAB/Simulink-Modellen automatisch Whitebox-Testfälle in Form von Testvektoren bzw. Folgen solcher Testvektoren (Testsequenzen) zu erzeugen. Mit Hilfe dieser Testsequenzen ist es dann möglich, die Übereinstimmung des Codes in einem zeitdiskreten Zielsystem mit dem zugrundeliegenden Modell zu überprüfen. Die Testsequenzen werden auf der Basis vorgegebener Äquivalenzklassen für die einzelnen im Modell verwendeten Blocktypen erzeugt, die durch den Testvektorgenerator in Testsequenzen für das gesamte zu testende System konvertiert werden. Darüberhinaus werden redundante Testsequenzen zusammengefasst, und die Sollwerte werden unabhängig von den zur Modellierung oder Code-Generierung verwendeten Tools berechnet.

1 Testen – Softwarequalität vs. Testaufwand

Ziel von Software-Entwicklungsprozessen ist es, Software mit möglichst guter Qualität in möglichst kurzer Zeit und damit auch mit möglichst geringem Aufwand zu erzeugen. Wesentliche Bestandteile eines solchen Prozesses sind dabei verschiedene Arten von Softwaretests, bieten sie doch die einzige Möglichkeit, die Qualität des erzeugten Codes direkt zu überprüfen. Für die Entwicklung von eingebetteten Steuergeräten im Automobilbereich können solche Tests je nach Testebene und -art einige hundert bis einige zehntausend einzelne Testfälle umfassen [Sch03]. Die Tests werden im allgemeinen bereits mit Hilfe entsprechender Tools automatisiert durchgeführt; vor der Testdurchführung wird jedoch auch heute noch meistens "manuell" in Testplänen bzw. -skripten festgelegt, welche Tests durchzuführen sind und wie diese im Detail abzulaufen haben.

Darüberhinaus ist es nicht ratsam, die Ermittlung der notwendigen bzw. sinnvollen Tests ausschließlich der persönlichen Erfahrung des Testers zu überlassen. Die Testfälle sollten stattdessen auf der Grundlage geeigneter Teststandards definiert werden, z.B. anhand von Überdeckungskriterien. Dies wird auch in einschlägigen Normen für sicherheitskritische Software in verschiedenen Ausprägungen gefordert [DIN01], [RTC92], [ISO]. Bei der modellbasierten Entwicklung ist es natürlich wünschenswert, auch modellbasierte Abdeckungskriterien zu verwenden.

Eine Automatisierung ist für die Teile des Testprozesses angebracht, die auf bereits vorhandenen, computerlesbaren Daten (z.B. Modellen) beruhen, relativ aufwändig sind, häufig wiederholt werden, und gleichzeitig nach festen Regeln, d.h. deterministisch und reproduzierbar, ablaufen bzw. ablaufen sollten. Dies ist bei der Definition von Whitebox-Testfällen aus Datenflussmodellen der Fall, wie sie z.B. mit Hilfe von MATLAB/Simulink oder ASCET erstellt werden können und vor allem bei Unit- und Modultests verwendet werden. Wichtig ist dabei, dass das Modell bereits verifiziert wurde, z.B. in Reviews, durch Simulationen oder durch Methoden der "formalen Verifikation", da die automatisch aus dem Modell generierten Testvektoren natürlich nicht in der Lage sind, Fehler innerhalb desselben Modells aufzudecken.

2 Kriterien

Aufgrund unserer Erfahrungen in der Entwicklung und im Testen von Automobil-steuergeräten wurden folgende Kriterien für ein solches Tool festgelegt:

Reproduzierbarkeit: Führt man den Testvektorgenerator mit identischem Modell und identischen Einstellungen zweimal aus, sollen identische Ergebnisse erzeugt werden.

Nachvollziehbarkeit: Für jede erzeugten Testsequenz sollte angegeben werden, aufgrund welcher Regeln dieser erzeugt wurde; andererseits sollte für jede der Regeln, die der Testvektorgenerierung zugrundeliegen, angegeben werden, welche Testsequenzen auf dieser Regel beruhen.

Testabdeckung: Die erzeugten Testvektoren sollten vordefinierte Abdeckungskriterien erfüllen, d.h. es muss Festlegungen geben, wann ein Test "vollständig" ist. Der Testvektorgenerator sollte gezielt versuchen, diese Abdeckungskriterien zu erfüllen und dem Benutzer eine Rückmeldung liefern, falls dies in einzelnen Fällen nicht gelingen sollte.

Unabhängigkeit: Die Testvektorgenerierung soll parallel zur Codegenerierung und –ausführung erfolgen. Um wirklich unabhängige Tests zu gewährleisten und falsch positive Testergebnisse zu vermeiden, soll bei der Testvektorgenerierung keins der zur Codegenerierung verwendeten Tools (z.B. Codegenerator, Compiler) verwendet werden.

Erweiterbarkeit: Der Testvektorgenerator muss in der Lage sein, an neue, bisher nicht berücksichtigte Blocksets oder benutzerdefinierte Blöcke (z.B. S-Functions in Simulink) angepasst zu werden.

Leider konnten diese Kriterien mit den auf dem Markt befindlichen Tools nicht erfüllt werden, so dass wir uns zu einer Eigenentwicklung entschlossen.

3 Funktionsweise des Testvektorgenerators

Im Gegensatz zu einem menschlichen Tester kann ein Testvektorgenerator den Sinn oder Zweck eines Modells nicht erfassen; ein solches Tool kann daher keine Testabläufe erzeugen, mit denen Anforderungen an das Modell abgeprüft werden können. Stattdessen ist es aber möglich, nach bestimmten Schemata die im Modell vorhandenen Strukturen auszuwerten, so dass Testvektoren erzeugt werden, mit denen das Vorhandensein bzw. Fehlen dieser Strukturen im Zielsystem aufgezeigt werden kann. Neben der Funktion der einzelnen Strukturen kann damit auch die korrekte Umsetzung von Konstanten, Grenzwerten o.ä. geprüft werden.

Der von uns entwickelte Testvektorgenerator geht dabei in folgenden Schritten vor:

Definition von Abdeckungskriterien für jeden Block des Modells: Für jeden implementierten Blocktyp existiert ein Algorithmus, der anhand der Blockeigenschaften geeignete Testvektoren bzw. -sequenzen für diesen Block in Form von Äquivalenzklassen für die Blockeingänge definiert. [Lin05] beschreibt einen Ansatz zur Definition solcher Abdeckungskriterien; die bei uns bereits seit Jahren verwendeten Kriterien beruhen auf praktischen Erfahrungen bei der manuellen Testdefinition und stellen einen Spezialfall solcher Kriterien dar. Sie können ggf. nach Bedarf konfiguriert und erweitert werden, z.B. auch auf blockübergreifende Kriterien.

Rückverfolgung der ermittelten Testvektoren bzw. –sequenzen zu den Systemeingängen: Die in Form von Eingangsintervallen bzw. Sequenzen solcher Intervalle gespeicherten Testvektoren jedes Blocks werden jeweils zu den Eingängen des zu testenden Systems zurückverfolgt. Die Rückverfolgung geschieht dabei durch einen entsprechenden Algorithmus, der für jeden implementierten Blocktyp vorgegeben ist. Er versucht jeweils, aus dem vorgegebenen Ausgangswertebereich für einen Block passende Eingangsintervalle zu ermitteln, und diese rekursiv weiter zurückzuverfolgen. Die Eingangsintervalle müssen dabei die Eigenschaft haben, dass jede Wertekombination daraus einen Ausgangswert aus dem Ausgangsintervall ergibt. Sofern die Rekursion nicht vor dem Erreichen aller relevanten Eingangssignale des Systems abgebrochen wird (s. unten), ist das Ergebnis ein Wertebereich für jedes Eingangssignal des Systems. Wird die Rekursion abgebrochen, können je nach Blocktyp evtl. andere Kombinationen von Eingangswertebereichen ausprobiert werden, da die Zuordnung der Eingangswertebereiche zu den Ausgangswertebereichen i.allg. nicht eindeutig ist.

Besitzt ein Signal eine Verzweigung, kann es passieren, dass einem Blockausgang bei der Rückverfolgung zwei Wertebereiche zugeordnet werden; in diesem Fall wird die Schnittmenge beider Wertebereiche gebildet und weiter zurückverfolgt. Ist die Schnittmenge leer, wird die Rekursion an dieser Stelle abgebrochen. Dasselbe passiert, wenn ein Ausgangsintervall vorgegeben wird, dessen Werte mit dem jeweiligen Block nicht realisiert werden können.

Bei Blöcken mit inneren Zuständen reicht es nicht aus, einen einzigen Wertebereich pro Eingang zurückzuverfolgen; stattdessen müssen hier auch Wertebereiche für die vorhergehenden Berechnungsschritte zurückverfolgt werden. Die Rückverfolgung einer solchen Sequenz von Wertebereichen erfolgt analog zur Rückverfolgung einzelner Wertebereiche, jedoch unter Berücksichtigung der Zeit als zusätzlicher Dimension.

Optimierung/Zusammenfassung der Testfälle: Da sich die ermittelten Testsequenzen für verschiedene Blöcke überlappen können, können sie durch einen Optimierungs-algorithmus zusammengefasst werden, um so die Anzahl der Testfälle zu verringern.

Auswahl konkreter Werte und Sollwertberechnung: Aus den Eingangswertebereichen jeder Testsequenz werden schließlich konkrete Eingangswerte-Sequenzen ausgewählt, die zusammen mit den anschließend berechneten Soll-Ausgangswerten als Testsequenzen in einer XML-Datei abgespeichert werden.

Um das beschriebene Vorgehen zur Testvektorgenerierung auch auf Blocktypen anwenden zu können, die in unserem Testvektorgenerator bisher nicht implementiert sind, besteht die Möglichkeit, das Verhalten solcher Blöcke mit Hilfe einer einfachen Scriptsprache zu beschreiben und sie so in den Testvektorgenerator zu integrieren. Dadurch kann das Verhalten des Testvektorgenerators auch für bereits implementierte Blocktypen verändert oder erweitert werden.

4 Fazit

Nicht alle Problematiken, die bei der Entwicklung eines Testvektorgenerators auftreten, konnten hier dargestellt werden. So muss z.B. sichergestellt werden, dass die Ausgangswerte eines zu testenden Blocks an den Systemausgängen mit hinreichender Genauigkeit und Auflösung ankommen. Auch muss ermittelt werden, wieviele Berechnungsschritte durchlaufen werden müssen, bevor dies geschieht. Diese Problematik lässt sich durch eine relativ einfache Erweiterung der oben beschriebenen Algorithmen lösen, auf die hier nicht näher eingegangen werden kann. Auch die Details zu den Algorithmen für einzelne Blocktypen, z.B. die Anwendung der hier geschilderten Verfahren auf Zustandsdiagramme (Stateflow), würde den Rahmen dieses Beitrags sprengen.

Die eingangs aufgestellten Kriterien konnten durch den beschriebenen Testvektorgenerator erfüllt werden: Die Reproduzierbarkeit der Ergebnisse ist gewährleistet, da die Testvektoren auf der Basis fester Regeln in Form vorgegebener Algorithmen ohne die Verwendung von Zufallsgeneratoren erzeugt werden. Die Nachvollziehbarkeit ist gegeben, da bei der Erzeugung und Zusammenfassung von Testfvektoren genau verfolgt wird, durch welche Regeln für welchen Blöcke des Modells jeder Testvektor erzeugt wurde; diese Informationen werden ebenfalls in der erzeugten XML-Datei gespeichert. Die gewünschte Testabdeckung (bezogen auf das Modell) kann durch geeignete Regeln für jeden Blocktyp optimiert werden, und die Unabhängigkeit von anderen Tools ist dadurch sichergestellt, dass es sich um eine komplette Neuentwicklung handelt, die ohne Verwendung dieser Tools entwickelt wurde

und auch ohne solche Tools lauffähig ist. Die Erweiterbarkeit für neue Blocktypen ist schließlich durch die Möglichkeit gegeben, die Eigenschaften solcher Blocktypen durch Skripte zu beschreiben und sie so in die Testvektorgenerierung zu integrieren.

Aufgrund des verwendeten Algorithmus ist für jeden zu ermittelnden Testfall eine Rechenzeit zu erwarten, die exponentiell mit der Zahl der bei der Rückverfolgung durchlaufenen Blöcke wächst, sofern diese keine eindeutige Umkehrfunktion haben. Der Testvektorgenerator versucht daher, durch eine Analyse der Modelleigenschaften die Anzahl der bei solchen Blöcken durchzuprobierenden Fälle möglichst klein zu halten.

Nach Tests mit kleineren Testmodellen wird der Testvektorgenerator zur Zeit erstmals in einem realen Projekt eingesetzt. Erste Ergebnisse sind vielversprechend, abschließende Aussagen über die Performance des Tools können jedoch noch nicht gemacht werden.

Der Testvektorgenerator kann nichts, was ein guter Testingenieur prinzipiell nicht auch könnte. Er kann auch keine grundlegenden Probleme des Softwaretests lösen, z.B. aufgrund schlecht testbarer Modelle. Er kann dem Testingenieur aber einen relativ langweiligen, aufwändigen und fehlerträchtigen Teil seiner Arbeit abnehmen und ihn so in die Lage versetzen, auch komplexere Systeme trotz begrenzter Zeit ausreichend zu testen; die erzeugten Testvektoren sind dabei umfassender, fehlerfreier und reproduzierbarer, als ein menschlicher Tester dies in der gleichen Zeit erreichen könnte.

Literaturverzeichnis

[DIN01] DIN EN 65108-7:2001 "Funktionale Sicherheit elektrischer / elektronischer / programmierbarer elektronischer Systeme"

[Her05] Hermes, T.; Schultze, A.; Predelli, O.: "Software Quality is not a Coincidence: A Model-Based Test Case Generator", SAE World Congress 2005, SAE, Paper #2005-01-1664

[ISO] ISO/IEC 15504 "Software Process Improvement and Capability Determination"

[Lin05] Linder, P.: "Constraintbasierte Testdatenanalyse für eingebettete Steuerungssoftware", Tagung "Simulations- und Testmethoden für Software in Fahrzeugsystemen", Berlin, 01.-02.03.2005

[RTC92] RTCA/Do-178B, "Software considerations in airborne systems and equipment certification", RTCA Inc., 1993

[Sch03] Schultze, A.: "Requirement Engineering – Effektives Verbesserungspotential bei der Entwicklung von Steuergerätesoftware", ATZ/MTZ Extra Automotive Electronics, 1/2003