MontiMatcher: Ähnlichkeitsanalyse-Framework zur Produktlinienextraktion und Evolutionsüberwachung

Bernhard Rumpe, Christoph Schulze, Michael von Wenckstern
Software Engineering, RWTH Aachen Universität, Deutschland
{rumpe, schulze, vonwenckstern}@se-rwth.de

1 Einleitung

Eingebettete Systeme, besonders im Automobilbereich, sind hochgradig konfigurierbar. Dies spiegelt sich im Varianten- und Versionenreichtum der Softwarekomponenten wider. Zur sicheren Integration von neuen Komponentenversionen oder -varianten während der Softwareevolution in das bereits existierende Gesamtsystem, ist es notwendig die Verhaltenskompatibilität zwischen verschiedenen Evolutionsstufen einer Komponente zu analysieren. MontiMatcher, ein Model-Checking-Framework zur Verhaltensanalyse, leitet vollautomatisiert gerichtete Kompatibilitäts- oder Ähnlichkeitsbeziehungen zwischen Simulink-Komponenten her. In einer Softwareproduktlinie (SPL) können diese Verhaltensähnlichkeiten zwischen Softwarekomponenten benutzt werden um diese durch generische Komponenten zu ersetzen, wodurch sich die Wartungskosten und Qualität der SPL verbessern. Da MontiMatcher neben konkreten Simulink-Komponentenimplementierungen auch auf Testfallspezifikationen von Komponenten arbeitet, können generische Komponenten einer SPL bereits in der Designphase identifiziert werden. Dieses Paper gibt einen Überblick über die zugrundeliegende Architektur und Methodik von MontiMatcher sowie die durchgeführten Evaluierungen im Rahmen von Industriekooperationen mit der Daimler AG und FEV GmbH, welche teilweise in [5] vorgestellt wurden.

2 Architektur und Methodik

Abb. 1 stellt die großen Funktionsmodule von MontiMatcher sowie dessen Kontrollfluss als Aktivitätsdiagramm dar. Neben zwei Simulink-Modellen, die deterministische Implementierungen eingebetteter Software als Komponenten- und Konnektorarchitekturen mit unidirektionalen Komponentenportverbindungen zur Informationsübertragung darstellen, akzeptiert das Framework auch zwei deterministische Testspezifikationen nach der Klassifikationsbaum-Methode [4]. Zuerst werden basierend auf den Simulink-Eingabemodellen nach [6] oder den Testeingabespezifikationen die beiden Kontrollflussgraphen (CFGs) ① extrahiert. Anschließend zerlegt ②, ähnlich zum Programmslicing, die beiden CFGs mit mehreren Ausgabeports in mehrere kleinere CFGs mit jeweils einen Ausgabeport. Da verschiedene Varianten/Versionen einer Komponente oftmals strukturelle Gemeinsamkeiten enthalten, wendet ③ Klonerkennungsalgorithmen auf den CFGs an. Somit werden in den nachfolgenden Schritten nur strukturelle Unterschiede auf Verhaltensähnlichkeit überprüft. ④ transformiert jeweils einen CFG zu je einen Eingabe-/Ausgabe-Automaten mit internen Variablen. Der Kompatibilitätsalgorithmus ⑤ überprüft ob der erste Automat abwärtskompatibel (das gleiche Verhalten enthält) zum zweiten Automat ist. Falls dies nicht der Fall ist, dann werden die Transitionen des ersten Automaten solange abstrahiert ⑥ bis dieser kompatibel zum zweiten Automaten ist. Anschließend berechnet ⑦ die CEGAS (Counterexample Guided Abstraction Similarity) Metrik; dies ist der Quotient aus der Anzahl nicht abstrahierter Transitionen bezüglich der Gesamttransitionsanzahl des ersten Automaten. Die CEGAS-Metrik, inspiriert vom CEGAR (Counterexample Guided Abstraction Refinement) Paradigma [2], ist das Ähnlichkeitsergebnis von MontiMatcher. Falls nur eine Überprüfung der Kompatibilitätsbeziehung zwischen beiden Komponenten gewünscht ist, werden die Schritte ⑥ und ⑦ übersprungen.

Der Kompatibilitätsalgorithmus [5] ⑤ besteht aus folgenden Teilfunktionalitäten: In ⓐ wird mittels Bounded-Model-Checking versucht eine Eingabesequenz für die beiden Automaten der zu vergleichenden Komponenten, welche den gleichen Ausgabeport repräsentieren, zu finden, so dass die Ausgabesequenz an einer Stelle verschieden ist. War diese Methodik für

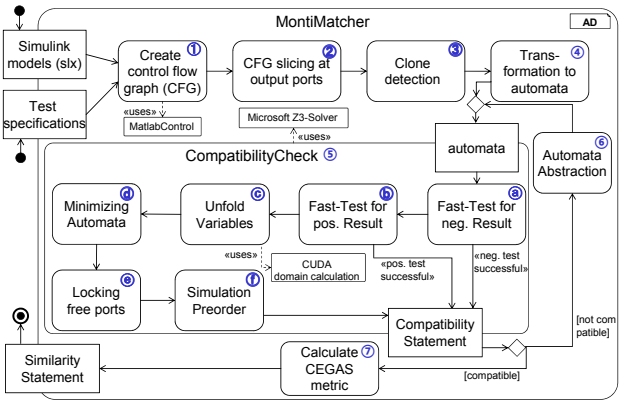

Abbildung 1: Software-Architektur von MontiMatcher

[RSW16] B. Rumpe, C. Schulze, M. von Wenckstern.
MontiMatcher: Ähnlichkeitsanalyse-Framework zur Produktlinienextraktion und Evolutionsüberwachung.
In: Softwaretechnik-Trends, 36(2), 2016.

einen der Ausgabeports erfolgreich, dann gibt ⑤ *nicht kompatibel* zurück. Andernfalls versucht ⓑ mittels affinen Abbildungen die internen Variablen des ersten Automaten auf den zweiten Automaten so abzubilden, dass die Ausgabefunktion sowie die Variablenaktualisierungsfunktionen als auch die Startwerte der internen Variablen identisch sind; ist dies gelungen dann liefert ⑤ *kompatibel* zurück. Waren die Schnelltests ⓐ oder ⓑ erfolglos, entfaltet ⓒ alle möglichen Belegungskombinationen der internen Variablen zu einzelnen Zuständen [1]. Dies funktioniert in der Regel nur für drei bis sieben Variablen, abhängig von den Variablenwertebereichen, denn die Anzahl der Zustände nimmt mit der Variablenanzahl exponentiell zu. Anschließend minimiert ⓓ die Automaten mittels Bisimulationsalgorithmen [3], um die Anzahl zu vergleichender Zustände und Transitionen so gering wie möglich zu halten. Falls der erste Automat mehr Eingangsports als der zweite Automat besitzt, dann fixiert ⓔ diese zusätzlichen Eingangsports mit konstanten Werten. Der Simulationsalgorithmus [3] ⓕ überprüft ob der erste Automat das Verhalten des zweiten Automaten nachbilden kann, indem er alle möglichen Transitionen und Zustände, die vom Startzustand beider Automaten aus erreicht werden können, besucht; dabei überprüft der Algorithmus ob die jeweils in Simulationsrelation stehenden Zustände für identische Eingaben auch gleiche Ausgaben liefern. Die Überprüfung der Ausgabegleichheit sowie die Identifikation von „toten“ Transitionen erfolgt mittels Microsofts Z3-Solver.

3 Evaluierung

Die Kompatibilitätsüberprüfung für Simulink-Modelle wurde anhand der Daimler SPES_XT Demonstratormodellen durchgeführt. Neben den in Tab. 1 auserwählten Beispielen der Evaluation wurde das Tool an einer Reihe unterschiedlich gearteter Simulink-Komponenten erprobt. Laut Zeile 2-3 in Tab. 1 ist die *Limiter*-Komponente mit Verkehrszeichenerkennung kompatibel zu der Version ohne Verkehrszeichenerkennung, wenn der zusätzliche Eingangsport *Sign_b* auf *false* fixiert ist. Da MontiMatcher für jeden Ausgangsport (*Vmax* und *Active_b*) der *Limiter*-Komponente einen eigenen Automaten erzeugt, sind diese Messwerte separat angegeben. Zeile 6 vergleicht die *CruiseControl*-Komponenten der Version 3 und 4 miteinander. Bei allen mit Stern gekennzeichneten Portnamen wurden die Optimierungsschritte ②, ③, ⓐ, ⓑ und ⓓ aktiviert, bei denen ohne Sterne wurden diese zu Vergleichszwecken deaktiviert. Wie dieser kleine Evaluierungsausschnitt zeigt, sind die vorgestellten Optimierungsschritte zur Verhaltenskompatibilitätsüberprüfung größerer Komponenten notwendig.

Port	Zeit [ms]	Ergebnis
Vmax	24624	T[Sign_b = false]/F
Active_b	1761	T/T
Vmax*	644	T[Sign_b = false]/F
Active*	0	T/T
CC(34)	30113	T[FTS_active_b = false]/F
CC(34)*	783	T[FTS_active_b = false]/F

Tabelle 1: Ergebnisse des Kompatibilitätscheckes beim SPES_XT Demonstrator [Ausschnitt].

SW Komp.	Anz. Testfälle	Zeit [ms]	Abgeleitete Ähnlichkeit [%]	Experten-ähnlichk. [%]
Sensor	$5^{\dagger}/3^{\ddagger}$	4'214	30 (0/60)$^{\parallel}$	33
Park Brake	$17^{\dagger}/5^{\ddagger}$	5'283	88 (75/100)$^{\parallel}$	75
Controller	$149^{\S}/165^{\P}$	315'784	97 (92,5/100)$^{\parallel}$	90

Tabelle 2: Ergebnisse der Ähnlichkeitsanalyze ($^{\parallel}$ gerichtete Ähnlichkeit, $^{\dagger}/^{\ddagger}$ Variante A/B, $^{\S}/^{\P}$ Version 1/2) [Ausschnitt].

Tab. 2 stellt einen Teil der Ergebnisse der testbasierten Ähnlichkeitsanalysen bei der FEV GmbH dar. Die Evaluierung zeigte, dass die hergeleiteten Ähnlichkeiten sehr gut mit der Expertenmeinung korrelieren. Für größere Testspezifikationen von komplexeren Komponenten nimmt die Analysezeit proportional zu den Testfällen zu.

4 Zusammenfassung

MontiMatcher erfüllt für eingebettete Softwaresysteme zwei Aufgaben. **Verifikation von Komponentenevolution:** Es kann zum einen zur Verifikation der Verhaltenskompatibilität zwischen verschiedenen Komponentenversionen in der Softwareevolutionsphase eingesetzt werden, um ungewollte Verhaltensänderungen bei Komponentenupgrades oder -refactorings zu entdecken. **Softwareproduktlinienextraktion:** Zum anderen identifiziert MontiMatcher ähnliche Komponenten sowie verwandte Softwaresysteme, um diese zu einer SPL zusammenzufassen.

Die ersten Evaluierungen von MontiMatcher zeigen sein großes Potential auf, allerdings kämpft das Framework wie alle anderen Model-Checking-Tools bei großen Softwaresystemen mit dem Zustandsraumexplosionsproblem [1].

Literatur

[1] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.

[2] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *CAV*, 2000.

[3] R. v. Glabbeek. The Linear Time-Branching Time Spectrum I - The Semantics of Concrete, Sequential Processes. In *Handbook of Process Algebra*, 2001.

[4] M. Grochtmann and K. Grimm. Classification trees for partition testing. *Software Testing, Verification and Reliability*, 3(2), 1993.

[5] B. Rumpe, C. Schulze, M. v. Wenckstern, J. O. Ringert, and P. Manhart. Behavioral Compatibility of Simulink Models for Product Line Maintenance and Evolution. In *SPLC*. ACM, 2015.

[6] C. Zhou and R. Kumar. Semantic Translation of Simulink Diagrams to Input/Output Extended Finite Automata. *Discrete Event Dynamic Systems*, 22(2), 2012.