UNIVERSITÄT DUISBURG ESSEN

paluno
The Ruhr Institute for Software Technology

Symbolisches Model Checking variabler Verhaltensspezifikationen

Masterarbeit im Studiengang 'Angewandte Informatik – Systems Engineering'
Fachbereich Wirtschaftswissenschaften, Arbeitsgruppe Software Systems Engineering
Betreuer: Dr. Kim Lauenroth, Prof. Dr. Klaus Pohl

Tim Jonischkat, MSc

** 14. November 1986 in Herne*
wohnhaft in Essen-Rüttenscheid

10/2005 - 06/2010.
Studium der Angewandten Informatik - Systems Engineering (BSc/MSc) an der Universität Duisburg-Essen

11/2007 - 07/2010.
Software-Ingenieur bei der Capgemini sd&m AG in einem Individualsoftware-Großprojekt für die Schenker AG

seit 08/2010.
Wissenschaftlicher Mitarbeiter am paluno - The Ruhr Institute for Software Technology an der Universität Duisburg-Essen

Abstract.

Die Nutzung von Software beschränkt sich aufgrund der steigenden Durchdringung des wirtschaftlichen und sozialen Lebens mit Technologie nicht nur auf Informationssysteme im Unternehmen, sie findet auch zunehmenden Gebrauch in Automobilen, in der Unterhaltungselektronik und anderen Gegenständen des alltäglichen Gebrauchs.

Der Markt erfordert zudem eine zunehmende Individualisierbarkeit und schnelle Innovationszyklen dieser Produkte, denen durch die Softwareproduktlinienentwicklung begegnet wird.

Die vorliegende Arbeit stellt dazu einen Beitrag zur Qualitätssicherung früher Softwareartefakte in der Domänenentwicklung einer Softwareproduktlinie dar. Der beschriebene Ansatz konzentriert sich dabei auf das Model Checking formaler Verhaltensspezifikationen als Ergebnis der Designphase gegen formale Anforderungsspezifikationen aus der Anforderungsanalyse. Er dient damit der zeitnahen Aufdeckung von Designfehlern in Basisartefakten einer Produktlinie mit Methoden der Inter-Modell-Konsistenzprüfung.

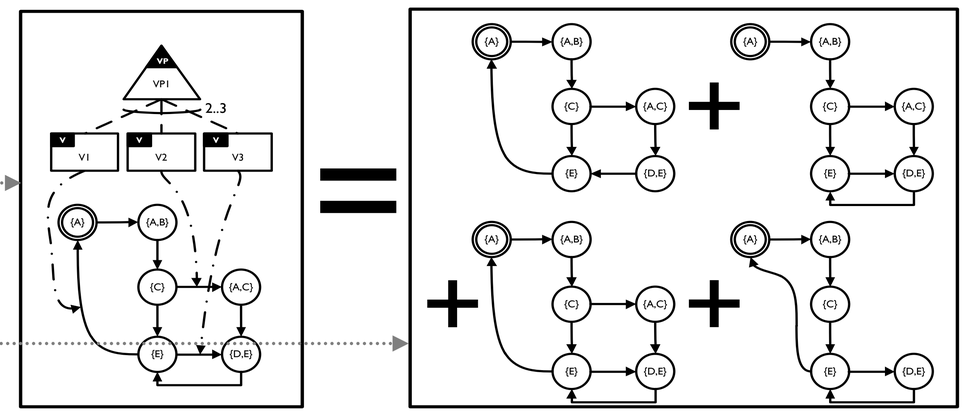
In dieser Arbeit wird das symbolische Model Checking für den Einsatz zur Konsistenzprüfung variabler Verhaltensspezifikationen und variablen CTL-Spezifikationen erweitert.

[McMillan92]
Kenneth Lauchlin McMillan. 1992. Symbolic Model Checking: An Approach to the State Explosion Problem. Ph.D. Dissertation. Carnegie Mellon Univ., Pittsburgh, PA, USA.

[Toehning09]
Kim Lauenroth, Klaus Pohl, Simon Toehning. Model Checking of Domain Artifacts in Product Line Engineering. International Conference on Automated Software Engineering, 2009

I. Motivation

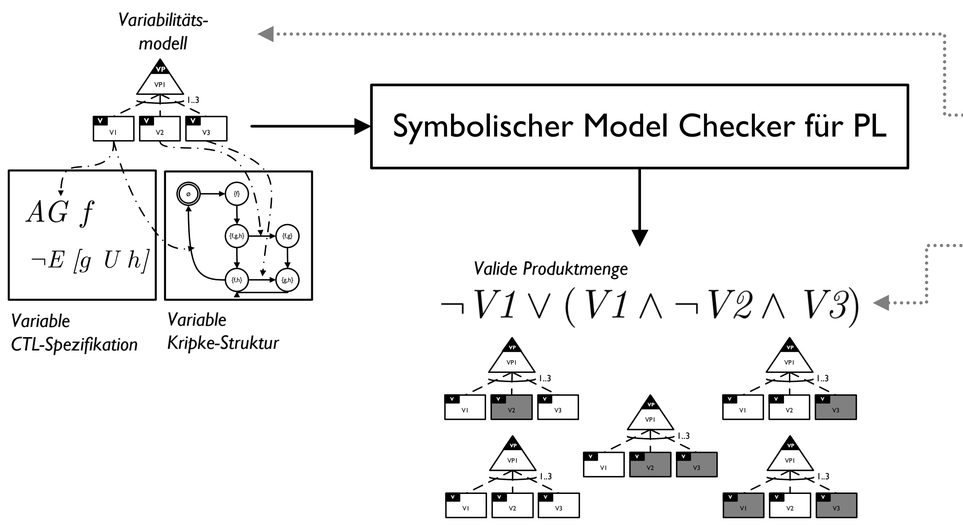
- Die Entwicklung von Softwareproduktlinien (SPLs) ermöglicht gleichartige, individualisierbare Produkte in optimierter Zeit und zu optimierten Kosten.
- Die Produktlinienartefakte bilden dazu die Variabilität der Produktlinie explizit ab, einzelne Produkte können durch Variabilitätsbindung abgeleitet werden.
- Die Verifikation von Produktlinienspezifikationen (PLS) kann mit bestehenden Methoden der Einzelsystementwicklung nicht durchgeführt werden, da eine PLS mit *n* Varianten bis zu 2^n mögliche Produkte abbildet - die Verifikation würde exponentiell skalieren.
- Existierende Ansätze zum Model Checking von PLS arbeiten auf einer expliziten Repräsentation des Zustandsraumes einer Verhaltensspezifikation - die Skalierbarkeit ist durch den hohen Speicherbedarf begrenzt.

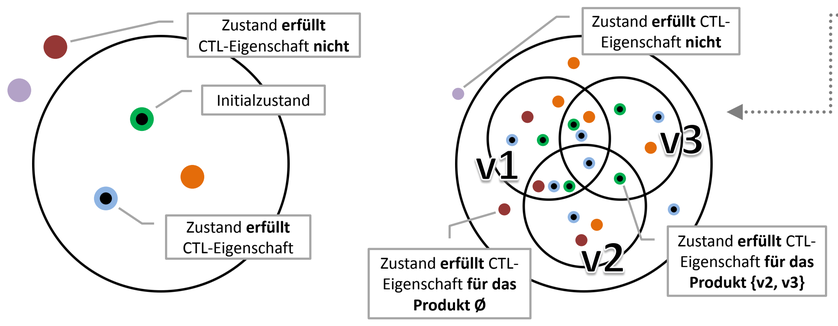


PL-Verhaltensspezifikation mit integrierter Variabilitätsdefinition

Mögliche Produktausprägungen der variablen Verhaltensspezifikation

II. Ansatz





Semantik einer symbolischen Zustandsmenge, die eine CTL-Eigenschaft erfüllt

Semantik einer symbolischen Menge von Zustandsmengen, die eine CTL-Eigenschaft abhängig von der PL-Variabilität erfül-

$$ValidProducts(f, v_{prop}) = \{ V^* \mid V^* \in \mathcal{P}(V) \wedge \forall (s_i \in I).(\quad SAT(\neg v_{prop} \vee (e_{OVM} \wedge \wedge \bigwedge_{v \in V^*} (v) \wedge \bigwedge_{v \in (V - V^*)} (\neg v) \wedge evaluate(f) \wedge inv(s_i))) \quad)\}$$

*Logische Definition des Algorithmus zur Berechnung der Produktmenge **V***, die die variable CTL-Eigenschaft **f** erfüllt, die mit der Variante v_{prop} relationiert ist*

II-1. Zielsetzung

Entwicklung eines effizienten **Model-Checking-Verfahrens** für verhaltensbasierte PLS, das **hohe Skalierbarkeitseigenschaften** aufweist.

Der Ansatz hat folgende Kriterien zu erfüllen:

- **Eingabe** ist (1) eine **variable CTL-Spezifikation,** die die zu prüfenden temporalen Eigenschaften der Verhaltensspezifikation beschreibt, (2) die **variable Verhaltensspezifikation** in Form einer Kripke-Struktur mit variablen Transitionen sowie (3) das **Variabilitätsmodell** der Produktlinie.
- **Ausgabe** ist die **Menge der in Bezug auf die CTL-Spezifikation validen Produkte** der geprüften Produktlinie, dargestellt als aussagenlogische Formel über die Varianten des Variabilitätsmodells.
- Das Model-Checking-Verfahren nutzt eine **symbolische Repräsentation des Zustandsraumes** der Verhaltensspezifikation, die durch **Binary Decision Diagrams** dargestellt wird.

II-2. Erarbeitung der Lösung

Auf Basis des **symbolischen Model-Checking-Verfahrens für Einzelsysteme** nach [McMillan92] wurden folgende **Anpassungen** zur Umsetzung der geforderten Anforderungen vorgenommen:

- Die **symbolische Repräsentation des Zustandsraumes** wurde um **Variabilität** erweitert. Damit kann eine Menge variabler Zustandsmengen kompakt dargestellt und so für jedes Produkt der Produktlinie die Gültigkeit einer CTL-Eigenschaft abgebildet werden.
- Die **Evaluationsalgorithmen** für die CTL-Operatoren $EX\varphi$, $E\varphi U\mu$ und $EG\varphi$ wurden zur Verarbeitung der variablen Zustandsmengen **angepasst,** allerdings algorithmisch nicht verändert.
- Zusätzliche **Algorithmen zur Extraktion der validen Produktmenge** aus dem Ergebnis der CTL-Evaluation wurden spezifiziert.

II-3. Korrektheitsbeweis

Die **algorithmische Korrektheit** der Lösung wurde durch prosaische Korrektheitsargumente unter Verwendung von Beispielmodellen bewiesen.

Zudem wurde eine **Laufzeitabschätzung** zur Bestimmung der Komplexität aller Algorithmen unter der Annahme der Repräsentation des Zustandsraumes mit Binary Decision Diagrams durchgeführt.

II-4. Prototypische Implementierung

Das entwickelte Verfahren wurde auf Basis der *Java*-Plattform und unter Verwendung der Binary-Decision-Diagramm-Bibliothek *JavaBDD* **prototypisch implementiert**.

Der Prototyp akzeptiert das verbreitete **UPPAAL-Format** zur Eingabe der Verhaltensspezifikation einer Produktlinie.

III. Validierung

Die **Skalierbarkeit** des entwickelten Verfahrens wurde durch die Anwendung des Prototypen auf eine Produktlinienspezifikation industrieller Größenordnung (375 Mrd. Produkte, 89.000 Zustände) gezeigt.

Der **Vergleich** der **Laufzeiten** der Prüfung zwölf verschiedener CTL-Eigenschaften **mit dem Prototypen des expliziten Model-Checking-Ansatzes von [Toehning09]** zeigte, dass die symbolische Prüfung mit einem Faktor von **bis zu 500x schneller** prüft.

Zudem zeigten die Laufzeitmessungen einen **geringeren Speicherbedarf** des entwickelten Prototypen. Einige Prüfungen, die der explizite Model Checker von [Toehning09] aufgrund eines **Speicherüberlaufes** nicht prüfen konnte, wurden durch den in dieser Arbeit entwickelten Prototypen erfolgreich durchgeführt.