Komponenten machen Beweise schwieriger

Probleme bei der Verifikation komponentenbasierter Systeme

Felix Gärtner

TU Darmstadt

felix@informatik.tu-darmstadt.de

Die Sache mit den 0190er Nummern

- Eine Telefongesellschaft bietet zwei Dienste an:
  - OCS: *originating call screening*
    "sperrt" ausgewählte Rufnummern
  - CFU: *call forwarding unconditional*
    Weiterleitung eingehender Anrufe z.B. zum Nachbarn

- Eltern sperren 0190er Nummern mittels OCS, doch Kinder können trotzdem dort anrufen. Wie?

Die Sache mit den 0190er Nummern

- Eine Telefongesellschaft bietet zwei Dienste an:
  - OCS: *originating call screening*
    "sperrt" ausgewählte Rufnummern
  - CFU: *call forwarding unconditional*
    Weiterleitung eingehender Anrufe z.B. zum Nachbarn

- Eltern sperren 0190er Nummern mittels OCS, doch Kinder können trotzdem dort anrufen. Wie?
  - Mittels CFU: Anrufe an 0190-x weiterleiten. . .
  - . . . und vom Nachbarn aus anrufen.

"feature interaction"

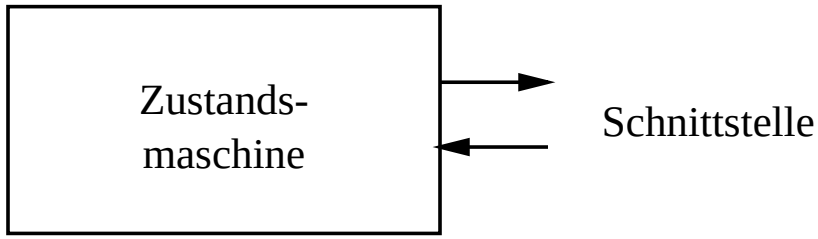
- Alle Dienste funktionieren einwandfrei, wenn man sie zu einem "leeren" System hinzufügt.
- Aber: Eigenschaften von Komponenten beeinflussen sich gegenseitig.

"feature interaction"

- Alle Dienste funktionieren einwandfrei, wenn man sie zu einem "leeren" System hinzufügt.
- Aber: Eigenschaften von Komponenten beeinflussen sich gegenseitig.
- Daumenregeln/Schlußregeln der folgenden Art wünschenswert:

$$\begin{aligned} \text{Wenn} \quad & A_1 \text{ hat Eigenschaft } S_1 \\ \text{und} \quad & A_2 \text{ hat Eigenschaft } S_2 \\ \text{dann} \quad & \text{hat } A_1 \| A_2 \text{ Eigenschaft } C(S_1, S_2). \end{aligned}$$

Was ist eine Komponente?



- Unterschied: Spezifikation/Implementierung.
- Komponentenverhalten läßt sich vollständig an der Schnittstelle beschreiben.
- Benutzen mathematischen Formalismus (z.B. temporale Logik).

Komponentenspezifikation

- *assumption/guarantee* Methode:

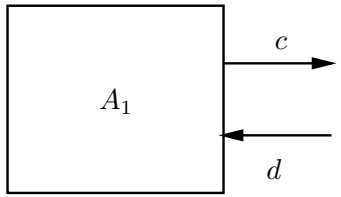
  System erfüllt M, wenn Umgebung E erfüllt.

- Genauer:

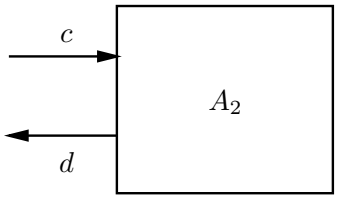
  System A erfüllt M, solange wie Umgebung E erfüllt.

  $$\{E\}A\{M\}$$

Ein einfaches Beispiel

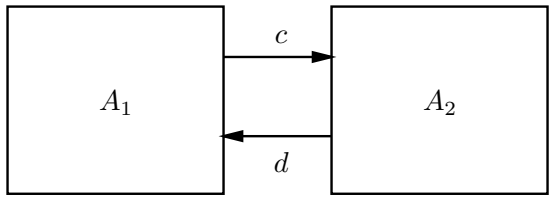


$$E_1 = \Box d = 0$$
$$M_1 = \Box c = 0$$

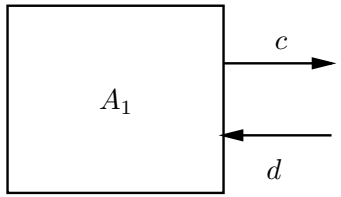


$$E_2 = \Box c = 0$$
$$M_2 = \Box d = 0$$

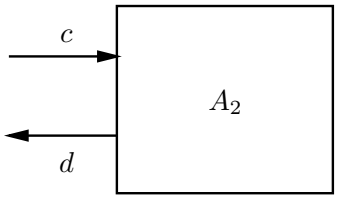
- Was kann man über die Komposition $A_1 \| A_2$ sagen?



Ein einfaches Beispiel

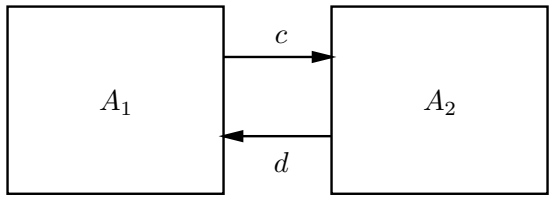


$$E_1 = \Box d = 0$$
$$M_1 = \Box c = 0$$



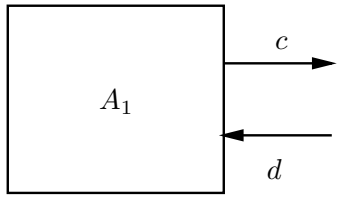
$$E_2 = \Box c = 0$$
$$M_2 = \Box d = 0$$

- Was kann man über die Komposition $A_1 \| A_2$ sagen?

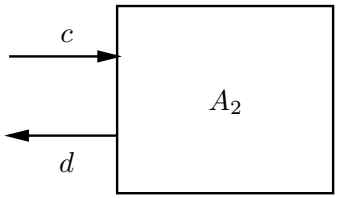


- $\{true\} A_1 \| A_2 \{M_1 \wedge M_2\}$

Noch ein einfaches Beispiel

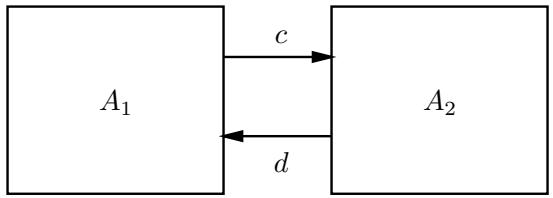


$$E_1 = \Diamond d = 1$$
$$M_1 = \Diamond c = 1$$

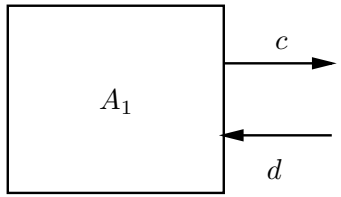


$$E_2 = \Diamond c = 1$$
$$M_2 = \Diamond d = 1$$

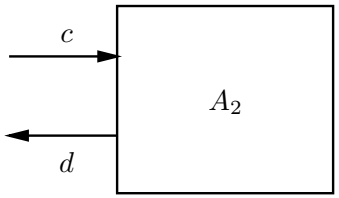
- Gilt hier $\{true\}A_1 \| A_2\{M_1 \wedge M_2\}$?



Noch ein einfaches Beispiel

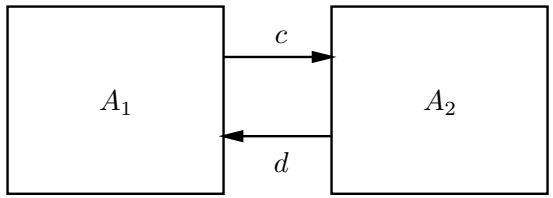


$$E_1 = \Diamond d = 1$$
$$M_1 = \Diamond c = 1$$



$$E_2 = \Diamond c = 1$$
$$M_2 = \Diamond d = 1$$

- Gilt hier $\{true\}A_1 \| A_2 \{M_1 \wedge M_2\}$?



- Nein! $A_1 \| A_2$ erfüllt $\Box c = 0 \wedge \Box d = 0$.

Ansatz zur Erklärung der Probleme

- Spezifikationen beschreiben Eigenschaften.
- Zwei wichtige Klassen von Eigenschaften [3]:
  - Sicherheit (*safety*): z.B. wechselseitiger Ausschluß.
  - Lebendigkeit (*liveness*): z.B. Terminierung.
- $\Box c = 0$ ist eine Sicherheitseigenschaft.
- $\Diamond c = 0$ ist eine Lebendigkeitseigenschaft.
- Lebendigkeitseigenschaften sind im im allgemeinen nicht “kompositionell” [7].

Ähnliche Fragestellung: Dekomposition

- Gegeben $A = A_1 \| A_2$ mit $\{E\}A\{M\}$ (hoffentlich!).
- Beweisführung sehr aufwendig!
- Idee:
  - Zerlegen E, M in E_1, M_1, E_2, M_2.
  - Beweise $\{E_1\}A_1\{M_1\}$ und $\{E_2\}A_2\{M_2\}$ separat.
  - Benutze Dekompositionstheorem, um $\{E\}A\{M\}$ zu folgern.

Dekomposition macht Beweise schwieriger

- Existierende Dekompositionstheoreme behandeln im wesentlichen nur *safety* [5, 1, 2] und sind *unvollständig* [6].
- Man muß E_i "raten" (ähnlich wie beim Finden einer Invariante).
- Beweislänge ist quadratisch in der Länge von E_1, M_1, E_2, M_2 [5].

Fazit

- Wir brauchen mehr Theoreme (insbesondere für Lebendigkeit)!
- Wir brauchen mehr Erfahrung in der formalen Verifikation von “echten” Systemen.
- Beweise werden leichter, wenn Mehraufwand automatisierbar ist (*model checking*) [4].
- Wenn man sich die Literatur anschaut: Komponentenbasierte Verifikation ist (noch) ein Graus!

Literatur

[1] Martín Abadi and Leslie Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, January 1993.

[2] Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, May 1995.

[3] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.

[4] R. P. Kurshan and L. Lamport. Verification of a multiplier: 64 bits and beyond. In C. Courcoubetis, editor, *Proceedings of the 5th International Conference on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 166–179, Elounda, Greece, 1993. Springer-Verlag.

[5] Leslie Lamport. Composition: A way to make proofs harder. In Willem-Paul de Roever, Hans Langmaak, and Amir Pnueli, editors, *Compositionality: The Significant Difference (Proceedings of the COMPOS'97 Symposium)*, number 1536 in Lecture Notes in Computer Science, pages 402–423. Springer-Verlag, 1998.

[6] Kedar S. Namjoshi and Richard J. Trefler. On the completeness of compositional reasoning. In *Proceedings of the 12th Int. Conference on Computer Aided Verification (CAV2000)*, number 1855 in Lecture Notes in Computer Science, pages 139–153. Springer-Verlag, July 2000.

[7] I. S. W. B. Prasetya and S. D. Swierstra. Factorizing fault tolerance. Technical Report UU-CS-2000-02, University of Utrecht, Department of Computer Science, Utrecht, The Netherlands, 2000. appears in special issue of TCS on fault tolerance.

Danksagungen

- Diese Folien wurden hergestellt unter Verwendung von pdfLaTeX und Klaus Guntermanns PPower4.

Zusammenfassung

In der Praxis baut man komplexe Systeme, indem man sie aus (weniger komplexen) Komponenten zusammensetzt. Das Verhalten des zusammengesetzten Systems ergibt sich aus dem zusammengesetzten Verhalten der Komponenten. In der Theorie kann man also ein komplexes System verifizieren, indem man zunaechst seine Komponenten verifiziert und dann daraus auf das Verhalten des Gesamtsystems schliesst. Ich werde kurz umreissen, wie Komponenten in der Theorie formalisiert werden koennen und auf welche Probleme man stoesst, wenn man komponentenbasiert verifizieren moechte. Wie auch immer Komponenten aus praktischer Sicht eingeschaetzt werden, in der Theorie sind sie (noch) ein Graus!