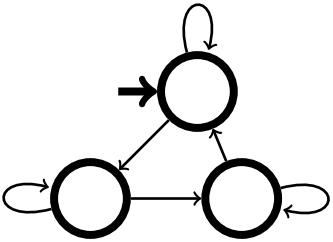
Algorithmische Einführung ins Automata Learning

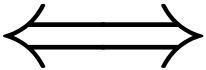
Wolffhardt Schwabe

1. Februar 2024

Überblick

Software Engineering





Spezifikation
(Modell)

Executable
(Blackbox)

Problem: Fehlender Quellcode

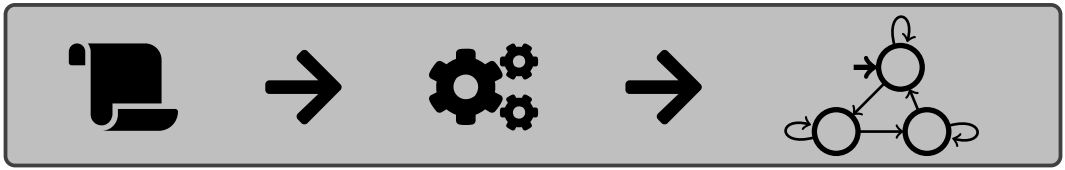
- Legacy-Software
- Manchmal fehlt sogar Spezifikation (oder ist unvollständig):
  - Unzureichend dokumentierte Third-Party-Komponenten
  - Allgemein: Reverse Engineering

Idee:
Konstruiere Verhaltensmodell durch Beobachtung der Ein- und Ausgaben des ausgeführten Systems
$\Rightarrow$ **Automata Learning**

Variante 1: Passives Lernen

Idee:

- Analysiere vorhandene Logs von Ein- und Ausgaben
- Extrapoliere Daten zu minimalem kompatiblen Modell *(evt. mehrdeutig!)*
  $\Rightarrow$ **Occam's Razor**: Datenlücke wächst mit Modellkomplexität



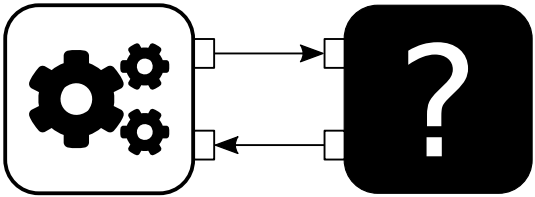
Nachteile:

- Daten liegen nicht immer vor
- Datenqualität limitiert Modellqualität
  $\Rightarrow$ Wichtige Edge Cases fehlen potenziell

Variante 2: Aktives Lernen

Bilde Modell durch Interaktion mit *System under Learning* (SUL):

- Starte ohne Annahmen über das Systemverhalten
- Schließe Wissenslücken durch strukturierte Systemausführung
  → Wähle nächste Eingabe jeweils abhängig von bisherigen Ein- und Ausgaben
  → Iterative Vervollständigung des Modells



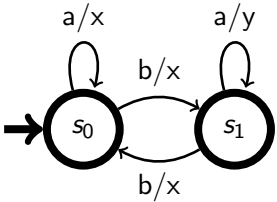
Voraussetzungen

- Programm isoliert ausführbar
- Programmzustand zurücksetzbar
- Ein-/Ausgabe-Schnittstelle bekannt
- Programmlogik mit ausgewählter Modellklasse beschreibbar
  - Jeder Input produziert immer genau einen Output
  - Zeitliche Abstände der Inputs egal
  - Verhalten ist deterministisch
  - Endlicher Zustandsraum

Mealy-Maschinen: Syntax

Mealy-Maschine $\mathcal{M} = (S, s_0, \mathcal{I}, O, \delta, \lambda)$:

- Zustandsmenge S
- Startzustand $s_0 \in S$
- Eingabemenge $\mathcal{I}$
- Ausgabemenge O
- Übergangsfunktion $\delta : S \times \mathcal{I} \to S$
- Ausgabefunktion $\lambda : S \times \mathcal{I} \to O$



Für Anzahl von Zuständen $\boldsymbol{n}$, Eingaben $\boldsymbol{p}$ und Ausgaben $\boldsymbol{q}$:

$\Rightarrow \mathcal{O}((np)^{nq})$ mögliche Modelle

$\Rightarrow$ Modellraum ist abzählbar

Mealy-Maschinen: Semantik

- Erweiterung der Ausgabefunktion: $\lambda^* : S \times \mathcal{I}^* \to O^*$

$$\lambda^*(s, a \cdot v) = \lambda(s, a) \cdot \lambda^*(\delta(s, a), v)$$
$$\lambda^*(s, \varepsilon) = \varepsilon$$

- Verhalten der Maschine für $w \in \mathcal{I}^*$: $\mathcal{M}(w) = \lambda^*(s_0, w)$

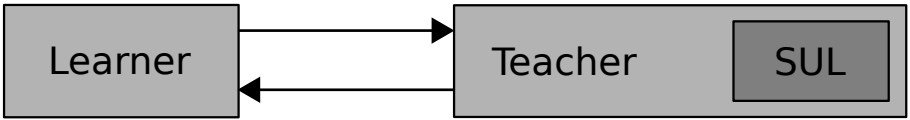
Äquivalenz zweier Maschinen $\mathcal{M}_1 \approx \mathcal{M}_2$:

- $\mathcal{I}_{\mathcal{M}_1} = \mathcal{I}_{\mathcal{M}_2} = \mathcal{I}$
- $\mathcal{M}_1(w) = \mathcal{M}_2(w) \quad \forall w \in \mathcal{I}^*$

Minimalität: Keine äquivalente Maschine mit weniger Zuständen

MAT-Framework [1]

- Algorithmisches Framework von Dana Angluin[1]: **Minimally Adequate Teacher**
  → Wesentlicher Meilenstein
- Zweiteilt Lernproblem:
  1. Aufbau einer Hypothese
  2. Prüfen und Widerlegen der Hypothese
- Lernprozess als Kommunikation zwischen *Learner* und *Teacher*
  → Learner interagiert mit Zielsystem nur indirekt



[1] D. Angluin. *Learning Regular Sets from Queries and Counterexamples. Information and Computation 75, Academic Press 1987, pp. 87–106.*

MAT-Framework [2]

Definition: SUL implementiert minimale Mealy-Maschine $\mathcal{M}$

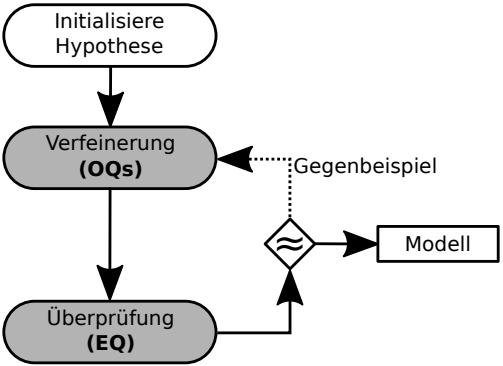
Teacher beantwortet zwei Arten von Anfragen:

- **Output Query (OQ):**
  Liefert $\mathcal{M}(w)$ für $w \in \mathcal{I}_{\mathcal{M}}^*$
- **Equivalence Query (EQ):**
  Für eine Mealy-Maschine $\mathcal{H}$, sagt ob $\mathcal{H} \approx \mathcal{M}$
  $\rightarrow$ Falls nein, liefert $w \in \mathcal{I}_{\mathcal{M}}^*$ mit $\mathcal{H}(w) \neq \mathcal{M}(w)$ *(Gegenbeispiel)*

$\Longrightarrow$ Teacher dient als Orakel

MAT-Framework [3]

Lernschleife:



Regel für Learner:

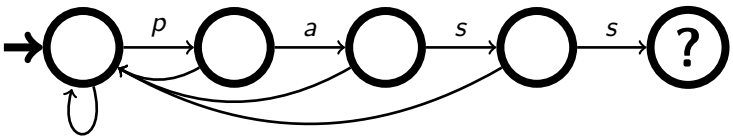
Hypothese vor jeder EQ eindeutig minimal

$\Rightarrow$ Streng monotones Wachstum

$\Rightarrow$ Konvergenz zu korrektem Modell

$+$ Terminierung der Schleife

Grenzen der Erlernbarkeit



- Für **begrenzte** Zustands- und Eingabemenge:
  endliche, aber **exponentielle** Anzahl von Kombinationen
  $\Rightarrow$ Äquivalenzprüfung für Blackbox braucht Exponentialzeit[2]
- In der Praxis gibt es nicht mal obere Schranke
  $\rightarrow$ Lernen ohne Korrektheitsgarantie
- Ergebnisse trotzdem nützlich, z.B. zum Finden reproduzierbarer Fehler

[2] D. Peled, M.Y. Vardi und M. Yannakakis. *Black Box Checking. PSTV 1999, FORTE 1999, pp 225–240.*

Diskriminatoren

- Gleiche Input-Sequenz führt immer zu gleichem Zustand
- Wie können wir Zustände einer Blackbox unterscheiden?
  $\rightarrow$ Einziges Kriterium sind die Ausgaben

Nerode-Kongruenz definiert Äquivalenz von Zuständen:

$$\forall s_x, s_y \in S : s_x \sim_{\mathcal{M}} s_y \iff \forall w \in \mathcal{I}^* : \lambda^*(s_x, w) = \lambda^*(s_y, w)$$

$\Rightarrow$ Für zwei unterschiedliche Zustände gibt es immer mind. einen **Diskriminator** (Suffix, der jeweils unterschiedliche Ausgaben produziert)

EQs in der Praxis

- Blackbox gilt auch für Teacher
  $\rightarrow$ Approximation durch differenzielles Testen
- Verschiedene Strategien möglich[3]:
  - Zufällige Wörter (mit Maximallänge)
  - RandomWp:
    1. Nehme Präfix von zufälligem Zustand in $\mathcal{H}$
    2. Hänge daran zufällige Sequenz
    3. Füge Diskriminator ans Ende
       $\Rightarrow$ Testet, ob Zielzustand der Zufallssequenz Erwartung entspricht
  - Viele andere Heuristiken ($\rightarrow$ **Conformance Testing**)
- Testparameter (bspw. maximale Wortlänge und -anzahl) sind zielspezifisch
  $\Rightarrow$ Trade-Off: Performance vs. Accuracy

[3] B.K. Aichernig, M. Tappler und F. Wallner. *Benchmarking Combinations of Learning and Testing Algorithms for Active Automata Learning. TAP 2020, LNCS 12165, pp. 3–22.*

Tabellenbasiertes Lernen[4]

Ziel: Lerne Mealy-Maschine $\mathcal{M}$ über $\mathcal{I}$

Observation Table $\mathcal{T}$

- Jede Zeile hat einen Präfix $u \in \mathcal{I}^*$
- Jede Spalte hat einen Suffix $v \in \mathcal{I}^*$
- Jede Zelle (u, v) enthält die $|v|$ letzten Symbole von $\mathcal{M}(u \cdot v)$

Closedness

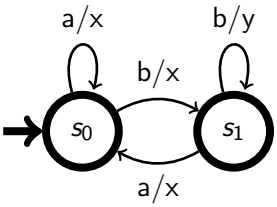
- Sei $\mathcal{U} = \mathcal{U}_\mathcal{S} \cup \mathcal{U}_\mathcal{L}$ die Präfixmenge und $\mathcal{V}$ die Suffixmenge
  - $\mathcal{U}_\mathcal{S}$ ist die präfixgeschlossene Menge der **Kurzpräfixe**
  - $\mathcal{U}_\mathcal{L} = (\mathcal{U}_\mathcal{S} \cdot \mathcal{I}) \setminus \mathcal{U}_\mathcal{S}$ ist die Menge der **Langpräfixe**
- Tabelle ist geschlossen *gdw.* $\forall u_\mathcal{L} \in \mathcal{U}_\mathcal{L} : [\exists u_\mathcal{S} \in \mathcal{U}_\mathcal{S} : \mathcal{T}(u_\mathcal{S}) \equiv \mathcal{T}(u_\mathcal{L})]$

[4] M. Shahbaz und R. Groz. *Inferring Mealy Machines. FM 2009, LNCS 5850, pp. 207–222.*

Table Closing

- Initial: $\mathcal{U}_\mathcal{S} = \{\varepsilon\}$, $\mathcal{U}_\mathcal{L} = \mathcal{I}$ und $\mathcal{V} = \{a\}$ mit $a \in \mathcal{I}$
- Solange $\exists u_\mathcal{L} \in \mathcal{U}_\mathcal{L} : \mathcal{T}(u_\mathcal{L}) \notin \mathcal{T}[\mathcal{U}_\mathcal{S}]$
  1. $\mathcal{U}_\mathcal{S} = \mathcal{U}_\mathcal{S} \cup \{u_\mathcal{L}\}$
  2. $\mathcal{U}_\mathcal{L} = (\mathcal{U}_\mathcal{S} \cdot \mathcal{I}) \setminus \mathcal{U}_\mathcal{S}$
  3. Fülle fehlende Zellen durch **OQs**
- Intuition:
  - $\mathcal{T}[\mathcal{U}_\mathcal{S}]$ charakterisiert Zustände
  - $\mathcal{T}[\mathcal{U}_\mathcal{L}]$ identifiziert Transitionsziele

Beispiel



	a	b
ε	x	x
b	x	y
a	x	x
ba	x	x
bb	x	y

Hypothesenkonstruktion

Verschiedene Kurzpräfixe haben unterschiedliche Zeileninhalte

$\Rightarrow$ Jeder Kurzpräfix identifiziert eindeutigen Zustand in $\mathcal{M}$

Geschlossene Tabelle definiert eindeutige Hypothese $\mathcal{H}$:

- Jeder Kurzpräfix entspricht einem Zustand und umgekehrt ($s_0 \leftrightarrow \varepsilon$)
- Transitionsziel für Eingabesymbol a in Zustand $u_{\mathcal{S}}$ ergibt sich aus $\mathcal{T}(u_{\mathcal{S}} \cdot a)$
- Zweite Tabelle für Transitionsausgaben
  $\rightarrow$ speichert Ausgaben für alle $u_{\mathcal{S}} \cdot a \in \mathcal{U}_{\mathcal{S}} \cdot \mathcal{I}$

Hypothesenverfeinerung

Empfang von Gegenbeispiel g mit $\mathcal{H}(g) \neq \mathcal{M}(g)$

1. Es gibt Zerlegung $g = u \cdot v$, wo Zustand u in $\mathcal{H}$ für v falsche Ausgabe produziert
2. $\mathcal{V} = \mathcal{V} \cup \{v\}$ sorgt für ungeschlossene Tabelle
3. Erneutes Table Closing führt zu mind. einem neuen Zustand

Zerlegung kann per Trick auf binäre Suche reduziert werden![5]

[5] R. L. Rivest und R. E. Schapire. *Inference of Finite Automata Using Homing Sequences. STOC 21, ACM 1989, pp. 411–420.*

Komplexitätsmaß

- Laufzeit dominiert von Ausführung des SUL
- Komplexität misst Grad der Interaktion mit SUL:
  - **Query Complexity:** Anzahl der OQs
    $\rightarrow$ erfasst konstanten Overhead
  - **Symbol Complexity:** Summe der Eingabelängen
    $\rightarrow$ skaliert mit Ausführungsdauer
- Komplexität der EQs abhängig von Testalgorithmus
  $\rightarrow$ bei asymptotischer Analyse ausgeblendet
  $\rightarrow$ Praxis: Ermittlung der Gesamtkomplexität per Messung für konkrete Systeme

Komplexität des Tabellen-Algorithmus

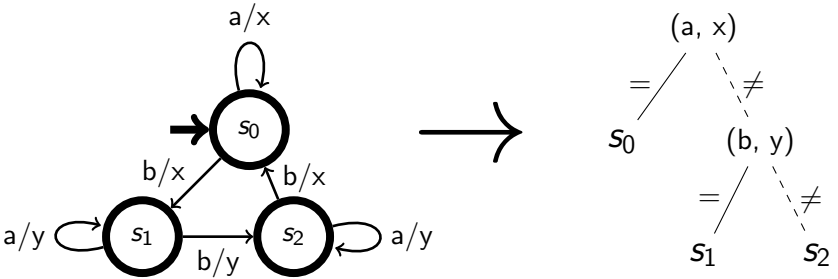
$n = |S_{\mathcal{M}}|$, $c = |\mathcal{I}_{\mathcal{M}}|$, $k =$ Länge des längsten Gegenbeispiels

- Bis zu $n - 1$ Gegenbeispiele (1 pro neuem Zustand)
  $\rightarrow$ Analyse erfordert $\mathcal{O}(n \log k)$ OQs
  $\rightarrow |\mathcal{V}| \leq n$
- Tabelle: $|\mathcal{U}_{\mathcal{S}}| = n$
  $\rightarrow |\mathcal{U}_{\mathcal{L}}| \leq nc$
  $\rightarrow \mathcal{O}(n^2 c)$ OQs
- Transitionsausgaben: $\mathcal{O}(nc)$ OQs

$\Longrightarrow$ Insgesamt $\mathcal{O}(n^2 c + n \log k)$ OQs

Baumbasiertes Lernen [1]

Dynamische Zustandsdiskriminierung[6] statt fixer Tabelle:



[6] M. Isberner, F. Howar und B. Steffen. *The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning. RV 2014, LNCS 8734, pp. 307–322.*

Baumbasiertes Lernen [2]

- Transitionsziel von $c \in \mathcal{I}_{\mathcal{M}}$ in Zustand $u \in \mathcal{I}_{\mathcal{M}}^*$ bestimmen:
  1. Starte bei Wurzel
  2. Bei innerem Knoten (v, o):
     - Erfrage $\mathcal{M}(u \cdot c \cdot v)$ per OQ
     - Vergleiche Ausgabesuffix mit o, um Nachfolgeknoten zu bestimmen
  3. Erreichtes Blatt zeigt Zustand an
- Bei Gegenbeispiel muss Baum erweitert werden

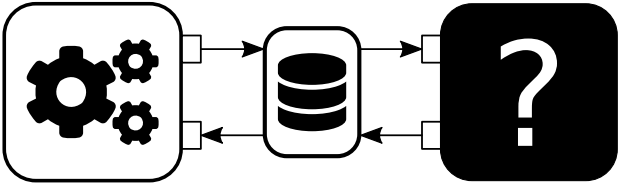
$\Rightarrow$ Balancierter Baum ermöglicht logarithmischen Lookup

Caching

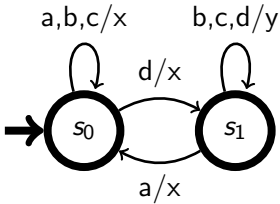
Sei $u \cdot v \in \mathcal{I}^*_{\mathcal{M}}$ und $\mathcal{M}(u \cdot v)$ aus vorheriger OQ bekannt

$\Rightarrow \mathcal{M}(u)$ ohne weitere OQ als Präfix ableitbar

$\longrightarrow$ Caching der OQs vermeidet redundante Ausführungen des SUL



Alphabet Abstraction [1]



- Oft differenzieren Zustände nur einen Teil der Symbole
  → Trotzdem müssen alle Transitionsziele und -ausgaben identifiziert werden

- In s_0 und s_1 würden jeweils zwei Stellvertreter-Symbole ausreichen
  → Erfordert korrekte Zuordnung der anderen Symbole

Alphabet Abstraction [2]

- Idee[7]: Beginne in jedem Zustand mit einem einzigen Symbol als Repräsentant
- Bei Gegenbeispiel:
  Prüfe bei jedem Symbol den Repräsentant auf abweichendes Verhalten im SUL
  $\rightarrow$ Zustand erhält ggf. zusätzliches Repräsentantensymbol
- Abbildung von Symbolen auf Repräsentanten erfolgt per Kodierungsbaum
  (funktioniert ähnlich wie Diskriminierungsbaum)
- Abstraktion hat Overhead, reduziert in Praxis aber oft Gesamtkomplexität

[7] M. Isberner, F. Howar und B. Steffen. *Inferring Automata with State-Local Alphabet Abstractions. NFM 2013, LNCS 7871, pp. 124–138.*

Implementierung

Großes Java-Framework: LearnLib[8]

- Open-Source-Projekt der TU Dortmund
- Implementiert viele bekannte Algorithmen
- Verschiedene Bausteine modular kombinierbar
  (Learner, Abstraktion, EQ-Orakel etc.)
- Bietet auch Datenstrukturen (bspw. Diskriminatorenbaum)
- Interaktion mit Anwendungen durch manuelles Interfacing

→ Ermöglicht einfaches Experimentieren

[8] learnlib.de

Case Studies

- Reverse Engineering von Smartcards[9]
- Aufdeckung von Bugs in Protokoll-Implementierungen (z.B. SSH[10], MQTT[11])
- Re-Engineering einer Legacy-Komponente eines Radiologie-Gerätes[12]
- Einiges mehr...

[9] G. Chalupar u. a. *Automated Reverse Engineering using Lego. WOOT 14, USENIX 2014.*

[10] P. Fiterău-Broștean u. a. *Model Learning and Model Checking of SSH Implementations. SPIN 24, ACM 2017, pp. 142–151.*

[11] M. Tappler, B. Aichernig und R. Bloem. *Model-Based Testing IoT Communication via Active Automata Learning. ICST 2017, IEEE, pp. 276–287.*

[12] M. Schuts, J. Hooman und F. Vaandrager. *Refactoring of Legacy Software using Model Learning and Equivalence Checking: an Industrial Experience Report. IFM 12, LNCS 9681, pp. 311–325.*

Aktuelle Forschung

- Erweiterung auf mächtigere Modelle
  - $\rightarrow$ zeitabhängiges Verhalten[13]
  - $\rightarrow$ zustandsübergreifender Speicher[14]
- Erweiterung auf Greybox-Szenarien[15]
- Verknüpfung mit Machine Learning[16]

[13] P. Kogel, V. Klös und S. Glesner. *Learning Mealy Machines with Local Timers. ICFEM 24, LNCS 14308, pp. 47–64.*

[14] S. Dierl u. a. *Scalable Tree-based Register Automata Learning. arXiv preprint arXiv:2401.14324 (2024).*

[15] B. Garhewal u. a. *Grey-Box Learning of Register Automata. IFM 2020, LNCS 12546, pp. 22–40.*

[16] G. Weiss, Y. Goldberg und E. Yahav. *Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples. ICML 35, PMLR 80, pp. 5247–5256.*

Zusammenfassung

- Blackbox-Lernen ist schwer!
  $\rightarrow$ Exaktheitsanspruch bedingt Komplexität
- Problem nur durch Reduktion auf Teilprobleme beherrschbar
- Praktischer Erfolg hängt auch von guten Heuristiken ab
  $\rightarrow$ Algorithm Engineering
- Enorme Fortschritte seit Feldentstehung!