Extraktion von Visual Contracts aus Java-Programmen

Jens Kosiol

22. Mai 2024

Überblick

- Visual Contracts spezifizieren das Verhalten von objektorientierten Systemen.
  - *Sie können das modellbasierte Testen von objektorientierten Programmen unterstützen.*
  - *Die manuelle Erstellung von Visual Contracts ist aufwendig und kann schnell zu Fehlern führen.*
- Kann man Visual Contracts aus einem lauffähigen Java-Programm extrahieren?
- Wie weit kann die Extraktion automatisiert werden?

Automatische Extraktion von Modellen aus Code

- Statischer Ansatz:
  - *Extraktion von Modellen aus Sourcecode oder Bytecode*
  - *Der Code muss kompilierbar, aber nicht lauffähig sein.*
  - *Solche Modelle können dynamisches objektorientiertes (z.B. dynamic binding) Verhalten nicht abbilden.*
- Dynamischer Ansatz:
  - *Extraktion von Modellen aus Testläufen eines Programms*
  - *Dynamisches objektorientiertes Verhalten kann abgebildet werden.*
  - *Extrahierte Modelle enthalten nur das Verhalten, das ausgeführt wurde.*

Anwendungsmöglichkeiten für extrahierte Modelle

- Automatische Dokumentation von
  - *APIs*
  - *Services in serviceorientierten Systemen*
- Visuelles Debuggen
- Anwenden von Verifikationsmethoden (z.B. Model Checking) auf extrahierte Modelle
- (Automatische Ableitung eines Testmodells)

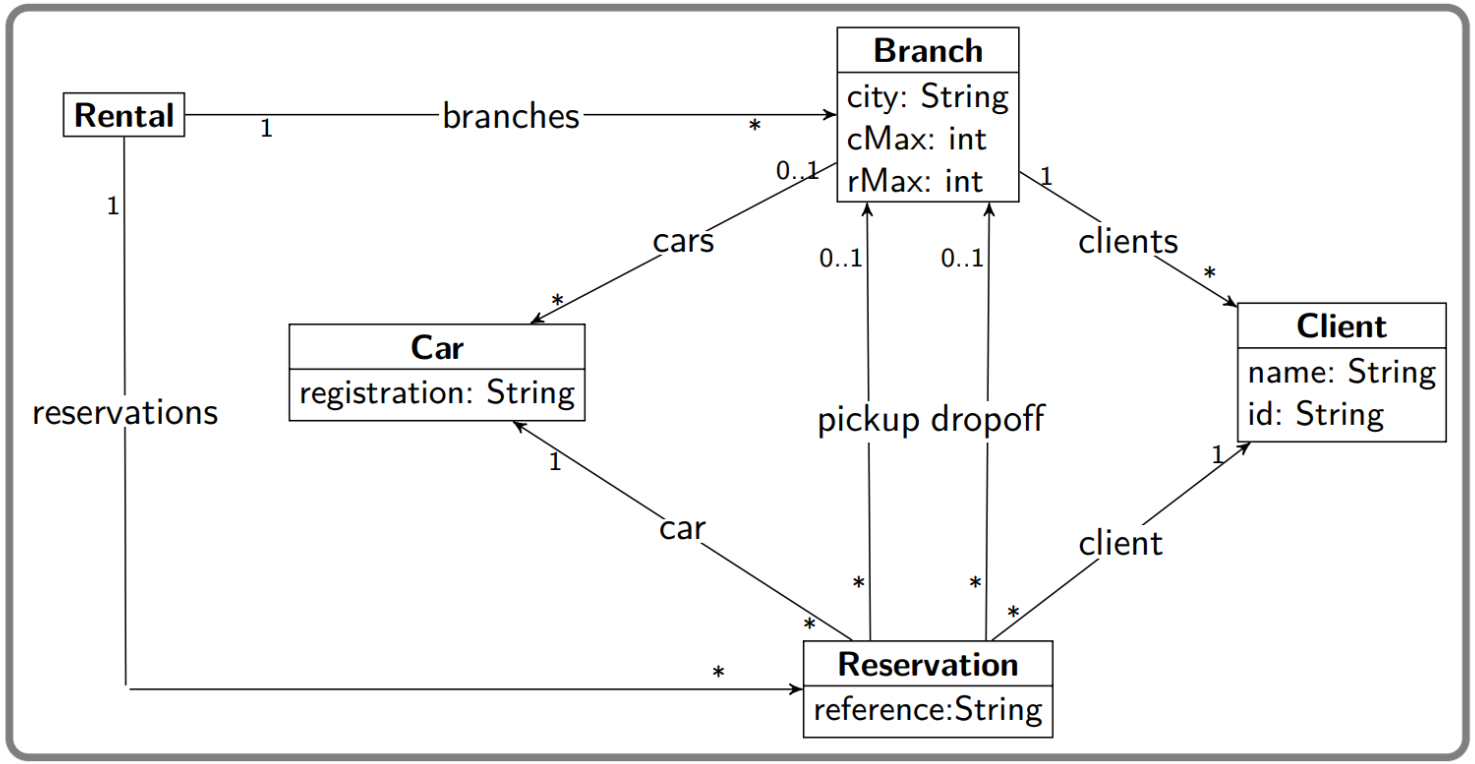
Dynamische Extraktion von Visual Contracts: Grundsätzlicher Ablauf

1. Auswahl von Klassen und Operationen
2. Beobachtung dieser Operationen während der Programmausführung
3. Erstellung von Prä- und Postgraphen für jeden Operationsaufruf: Contract Instances
4. Kombination von Contract Instances in Visual Contracts (Abstraktion von unwesentlichen Inhalten)
5. Weitere Generalisierung durch die Verwendung von Multiobjekten
6. Ableitung von logischen Bedingungen über Attribut- und Parameterwerten

Beispiel: Java-Klasse Rental

```
1  package rentalService;
2
3  public class Rental {
4    public Branch[] branches;
5    public ArrayList<Reservation> reservations;
6    public String registerClient (String city, String clientName) {..}
7    public String makeReservation(String clientId, String pickup, String dropoff) {..}
8    public Boolean cancelReservation(String resId) {..}
9    public Boolean cancelClientReservation (String clientId ) {..}
10   public Boolean pickupCar(String resId) {..}
11   public Boolean dropoffCar(String resId) {..}
12   public ArrayList<Reservation> showReservationsForClient(String clientId ) {..}
13   // further methods to access rental information
14 }
15
16 public class Branch {
17   public ArrayList<Car> at =new ArrayList<Car>();
18   public ArrayList<Client> ofClients=new ArrayList<Client>();
19   public String city=null;
20   public int cMax;
21   public int rMax;
22 }
```

Abgeleiteter Typgraph (inklusive weiterer Klassen)



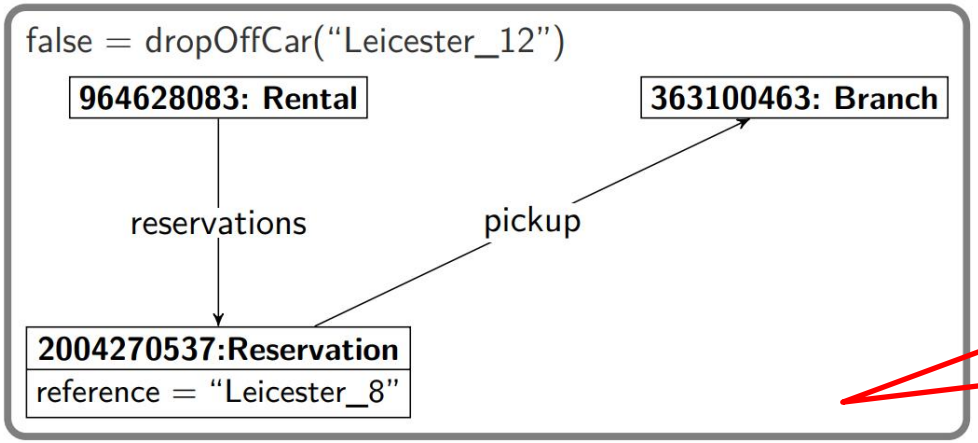
Extraktion von Visual Contract Instanzen

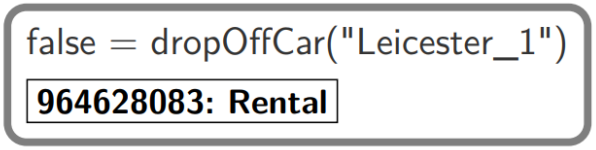
- Menge von Testläufen, die den Code ausgewählter Operationen überdecken sollten.
- Jeder Aufruf einer ausgewählten Operation wird mitprotokolliert.
  - *Es werden konkrete Objektstrukturen, der Zugriff auf Objekte und Felder sowie Parameter protokolliert.*
  - *Jegliche vorgenommene Änderungen werden protokolliert.*

Methode: Rental.dropoffCar()

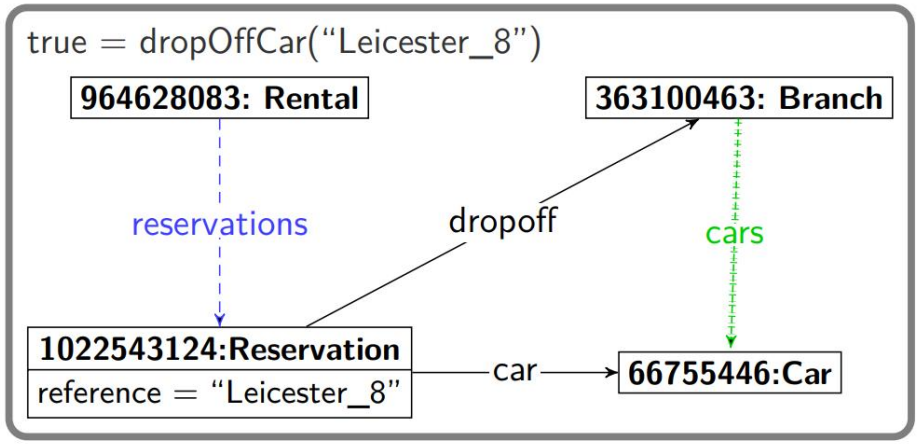
```
1   public Boolean dropoffCar(String resId){
2
3           int resIndex = this.getReservationIndex(resId);
4           if (resIndex==-1){
5               return false;
6           }
7
8           Reservation reservation = this.reservations.get(resIndex);
9           // check if the reserved car has been picked up already
10          if (reservation.pickup!=null){
11              return false;
12          }
13
14          // return reserved car to the dropoff branch
15          reservation.dropoff.cars.add(reservation.car);
16          // remove reservation object
17          this.reservations.remove(resIndex);
18          return true;
19      }
```

Beispiel: Extrahierte Visual Contract Instanzen

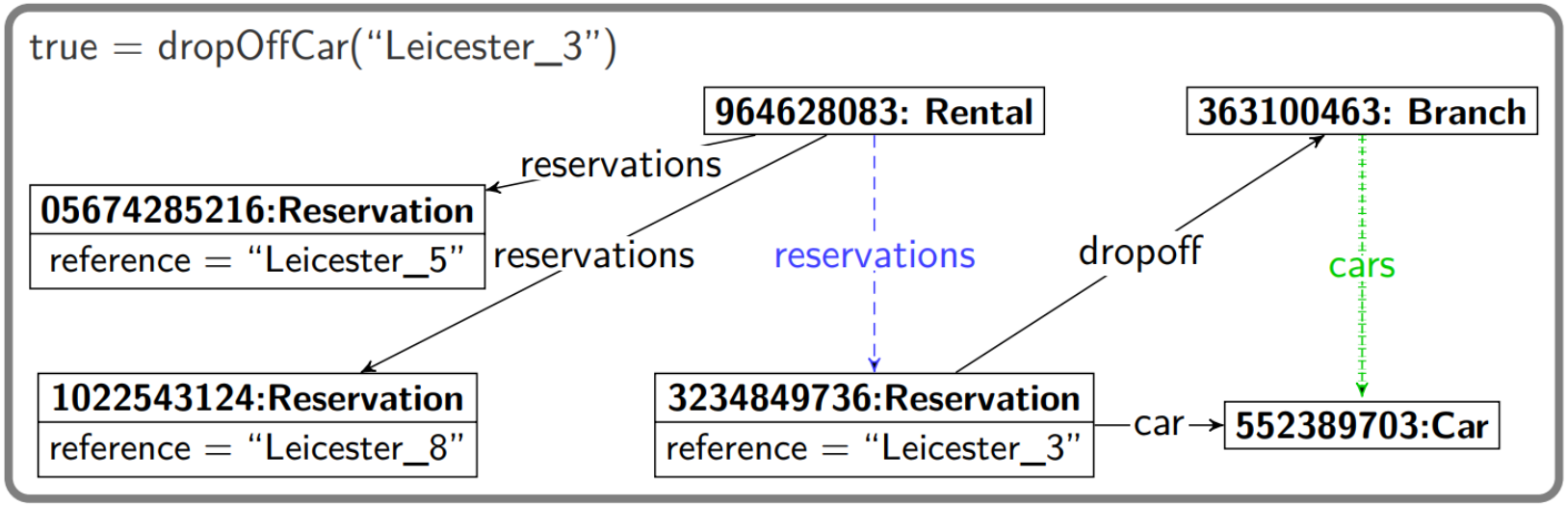




Achtung: Hier muss es heißen: reference = „Leicester_12“



Beispiel: Extrahierte Visual Contract Instanzen



Definition: Visual Contract Instanz

Eine Visual Contract Instanz ist ein Paar von Graphen (mit In- bzw. Output), ein Prägraph und ein Postgraph.
Gegeben einen Test (Methodenaufruf in einem gegebenen Kontext) besteht der abgeleitete Prägraph aus allen Objekten, Referenzen und Attributwerten, die während eines Durchlaufs der Methode beobachtet wurden. Die Inputparameter des Methodenaufrufs sind der Input des Prägraphen.
Der abgeleitete Postgraph entsteht aus dem Prägraph, indem

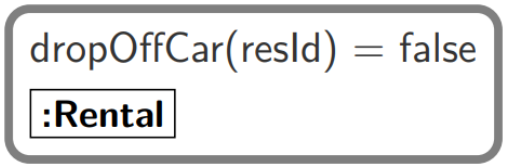
- alle Referenzen auf Objekte bzw. Attributwerte, die während des Durchlaufs gelöscht wurden, auch aus dem Graphen gelöscht werden;
- alle während des Durchlaufs neu erzeugten Objekte und Referenzen auf Objekte und/oder Attributwerte dem Graphen hinzugefügt werden.

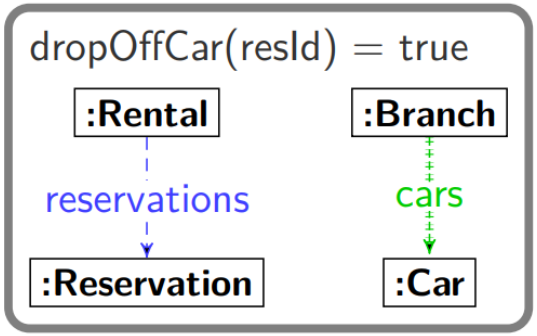
Die Outputparameter des Methodenaufrufs sind der Output des Postgraphen.

Die Visual Contract Instanz kann als Graphtransformationsregel aufgefasst werden, indem man den Prägraphen als linke und den Postgraphen als rechte Regelseite versteht.

Abstraktion von VC Instanzen zu Visual Contracts (Schritt 1)

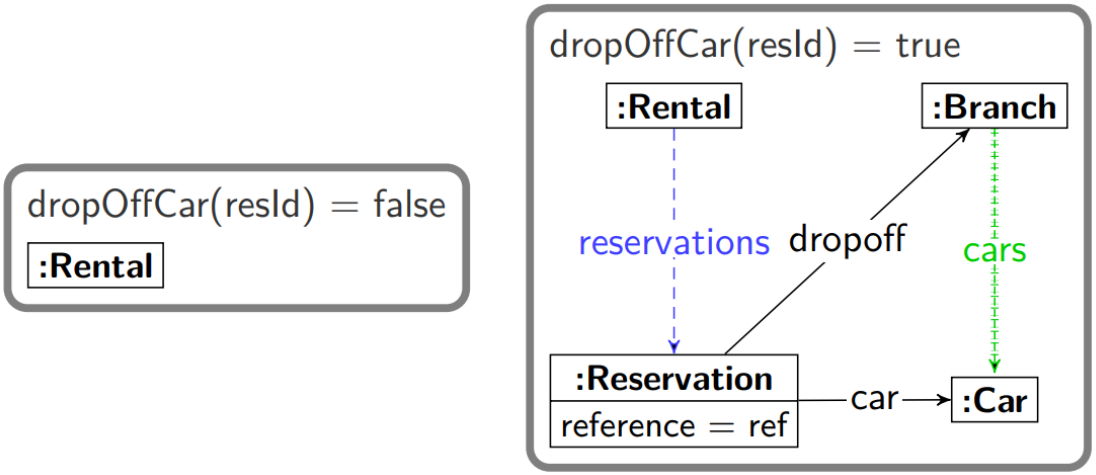
- Minimale Regel:
  - *Kontext in einer VC Instanz, der für die beobachteten Änderungen nicht gebraucht wird, wird gelöscht.*
  - *Intuition: Minimale Regeln klassifizieren die VC Instanzen gemäß ihrer Effekte.*
  - *Beispiel: Minimale Regeln für dropOffCar()*





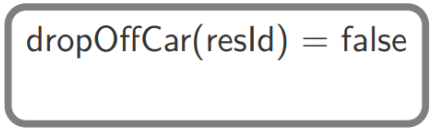
Abstraktion von VC Instanzen zu Visual Contracts (Schritt 2)

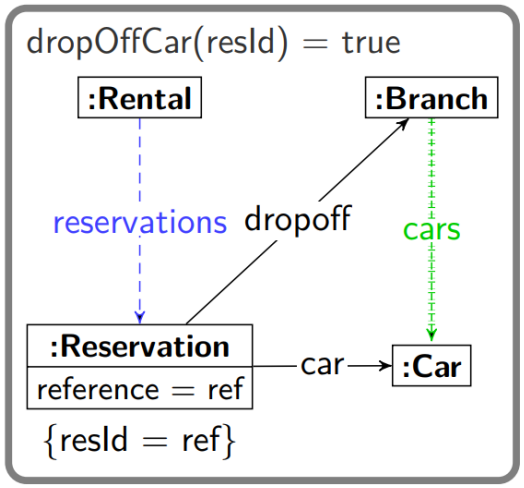
- Maximale Regel:
  - *Eine minimale Regel wird um den Kontext erweitert, den alle VC Instanzen mit derselben minimalen Regel teilen.*
  - *Intuition: Entspricht typischerweise einem Pfad durch den Code*
  - *Beispiel: Maximale Regeln für dropOffCar()*



Abstraktion von VC Instanzen zu Visual Contracts (Schritte 3 und 4)

- Globaler Kontext:
  - *Bedingungen, die in allen VC Instanzen auftreten, bilden Invarianten und werden aus Visual Contracts gelöscht (falls möglich).*
- Attributbedingungen und -zuweisungen
  - *werden durch Machine Learning aus den konkreten Attributwerten gelernt.*





Extraktion von minimalen Regeln

Input: Visual Contract Instanzen

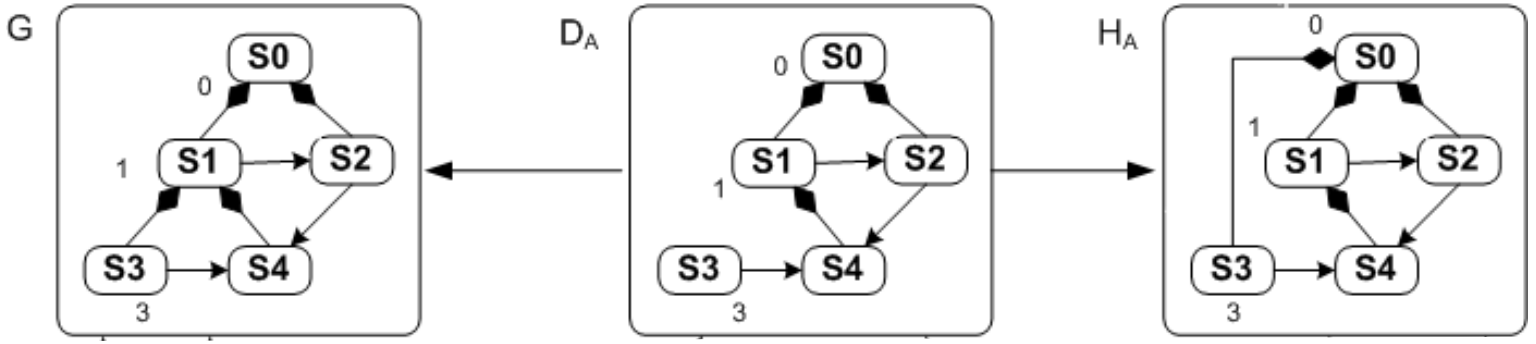
Output: minimale Regel

Vorgehen:

- Extraktion der Effekte (Löschen und Hinzufügen von Objekten bzw. Attributwerten)
- Bestimmung des minimal nötigen Kontexts
- Abstraktion konkreter Werte zu Variablen

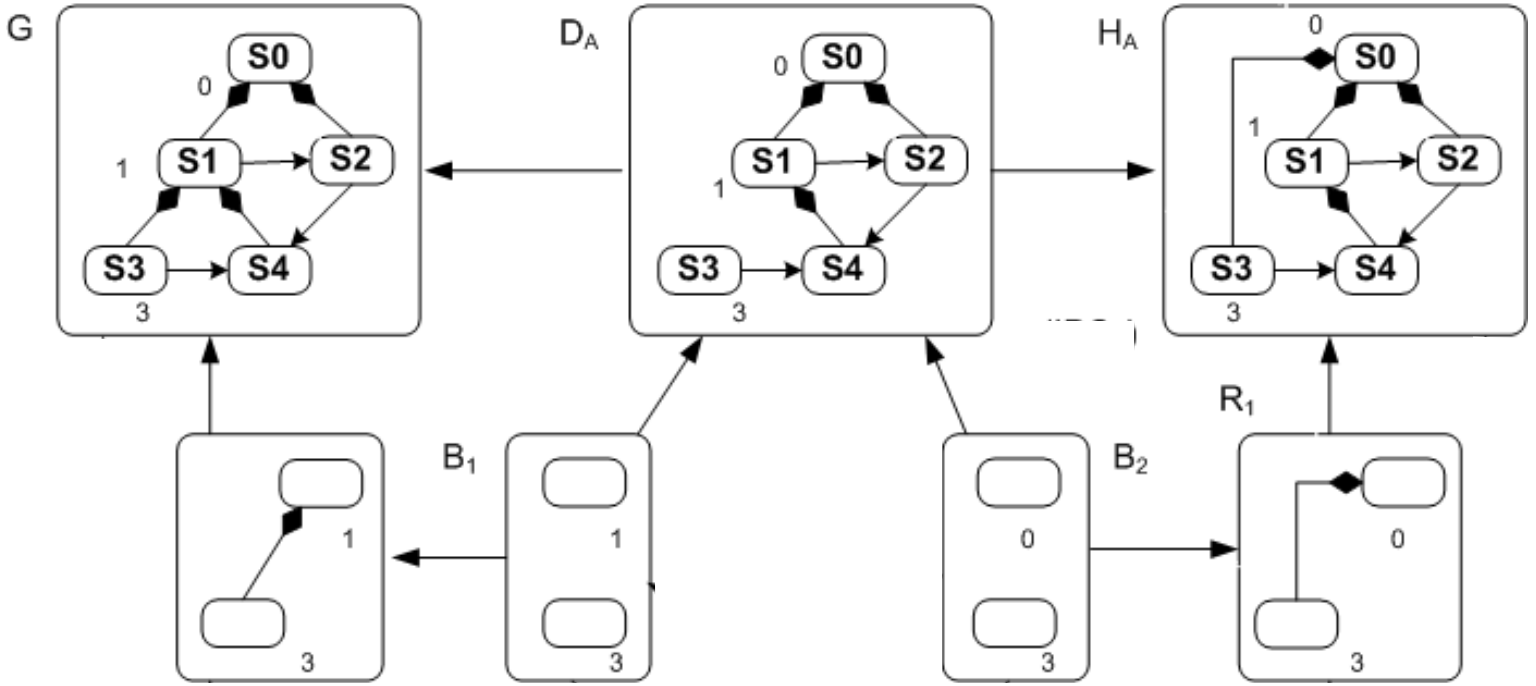
- Eventuell: Zusammenfassen von ähnlichen Visual Contracts zu einem

Beispiel: Extraktion der Effekte



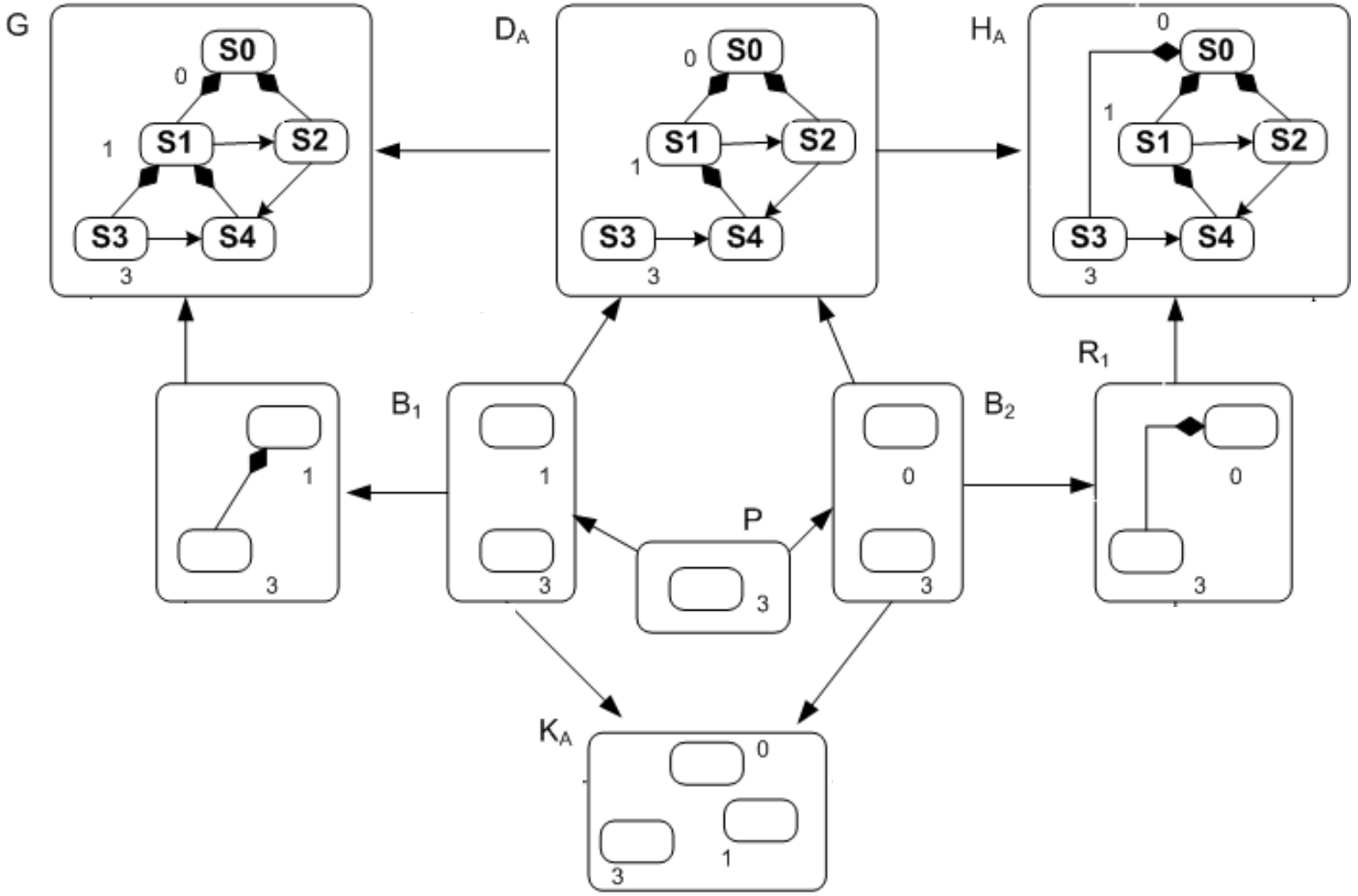
[TELW10]

Beispiel: Extraktion der Effekte



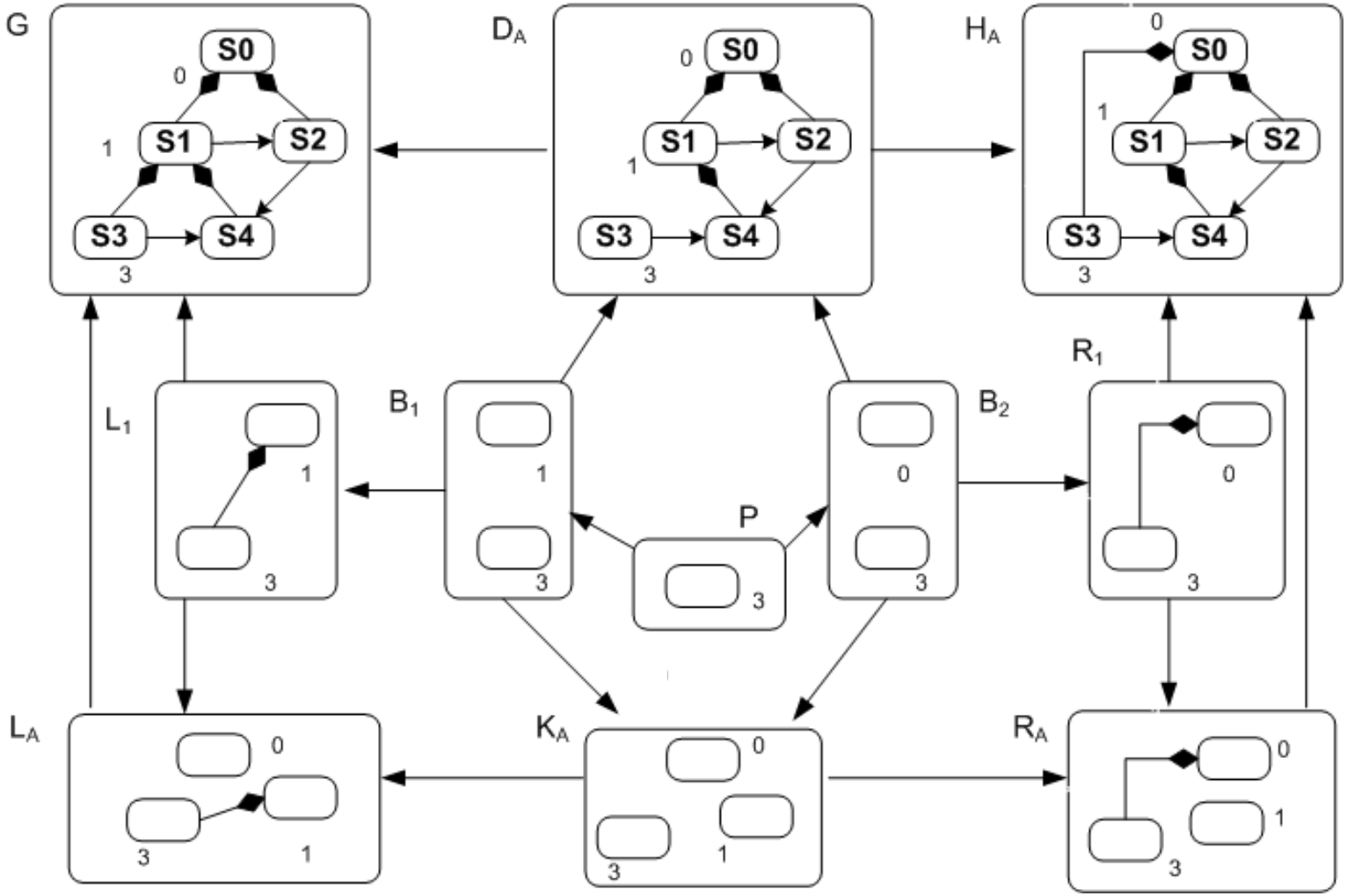
[TELW10]

Beispiel: Extraktion der Effekte



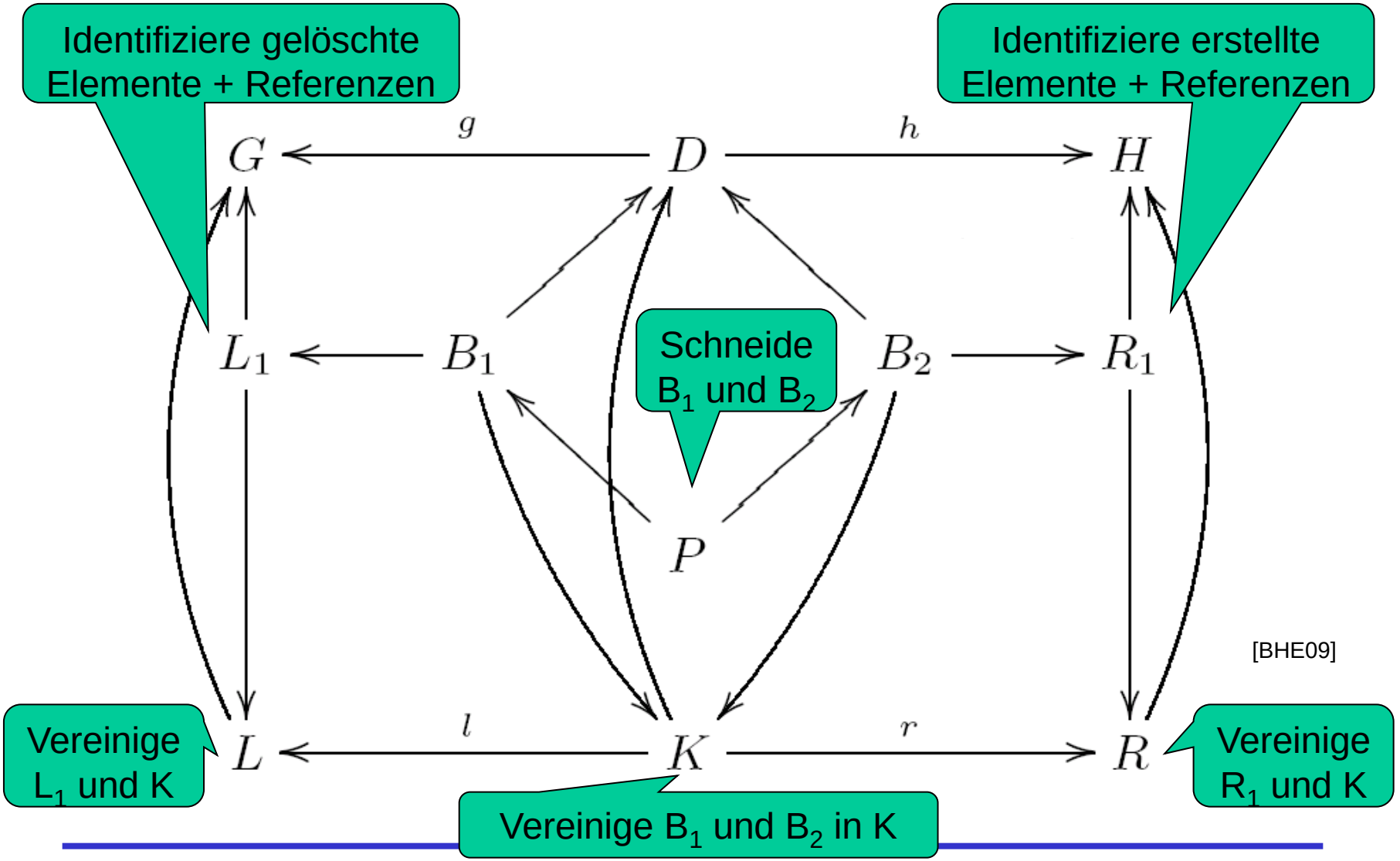
[TELW10]

Beispiel: Extraktion der Effekte



[TELW10]

Schema: Extraktion der Effekte



Minimale Regeln

Gegeben eine Menge von Visual Contract Instanzen wird auf die folgende Art und Weise eine Menge von minimalen Regeln erzeugt:

- Aus jeder Visual Contract Instanz werden die Effekte extrahiert. Hierbei werden allerdings beibehalten:
  - *das this-Objekt*
  - *alle Parameter (Input und Output)*
- In jeder dieser reduzierten Regeln werden konkrete Parameterwerte durch Variablen ersetzt.
- Die entstehende Menge (Duplikate werden ignoriert!) ist die Menge der minimalen Regeln.

Ableitung von Multi-Patterns

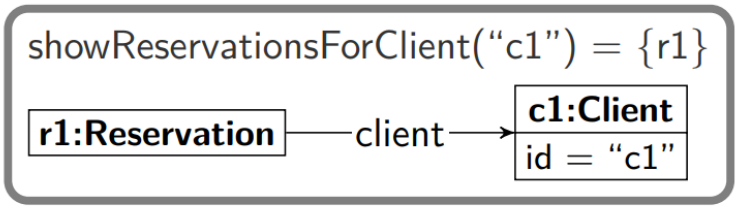
- Erkennen von äquivalenten Objekten:
  - *Zwei Objekte sind äquivalent, wenn sie denselben Typ haben, beide in der minimalen Regel vorkommen und denselben Kontext haben, d.h. die anhängenden Kanten sind von demselben Typ und sind mit denselben Knoten verbunden.*
- Zusammenführen von äquivalenten Objekten:
  - *In jedem Visual Contract werden äquivalente Objekte zu einem Multiobjekt zusammengeführt.*
- Kombination isomorpher Visual Contracts:
  - *Strukturell äquivalente Contracts werden zu einem zusammengeführt.*
- Dieses Verfahren kann auf Patterns verallgemeinert werden. $\rightarrow$ Multi-Patterns

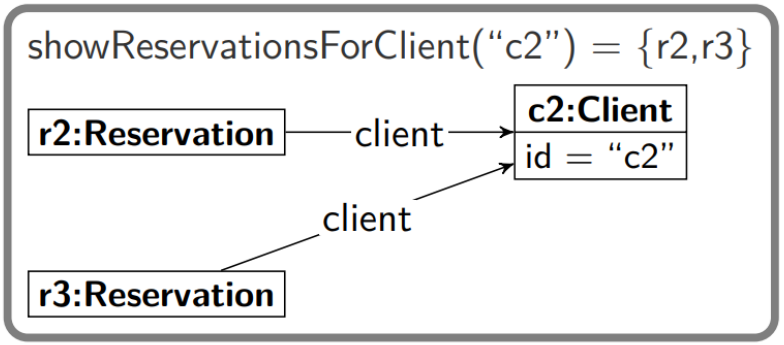
Methode: Rental.showReservationsForClient()

```
1   public ArrayList<Reservation> showReservationsForClient(String clientId ){
2
3           ArrayList<Reservation> cReservations = new ArrayList<Reservation>();
4           for (Reservation reservation : reservations ){
5               if ( reservation . client . cId . equalsIgnoreCase( clientId )){
6                   cReservations . add( reservation );
7               }
8           }
9           return cReservations ;
10      }
```

Beispiel: Abgeleitete Visual Contract Instanzen für showReservations…

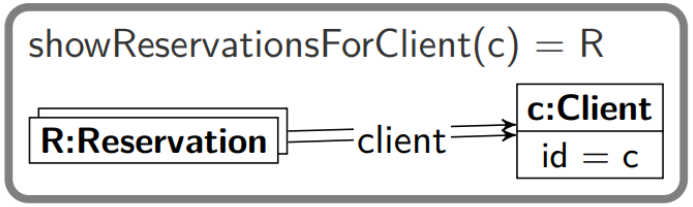
VC Instanzen:





Die Rückgabe ist eine Menge.

Visual Contract:



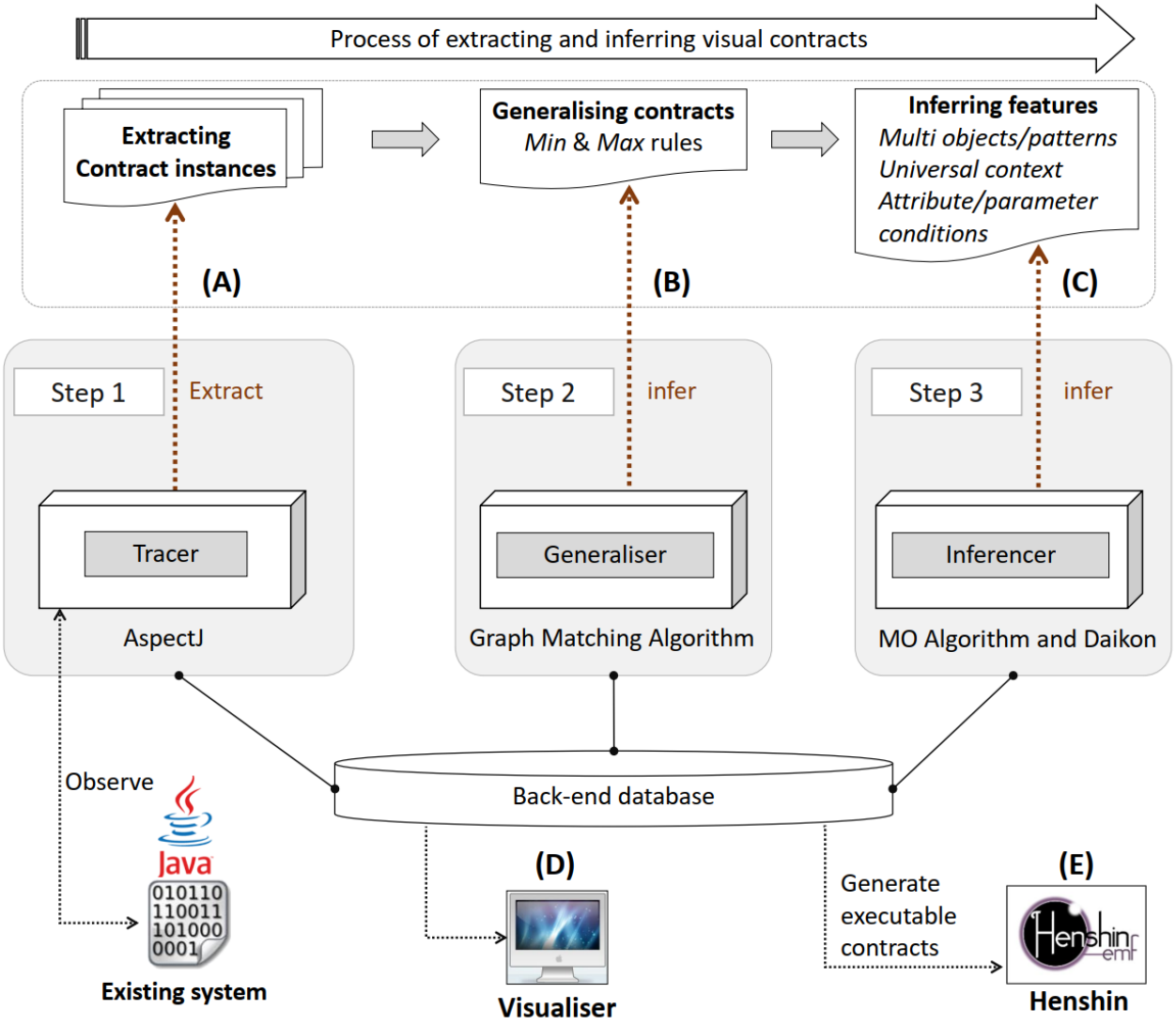
Korrektheit und Vollständigkeit

- Für jeden Zustand s der Implementierung gibt es einen Graphen $G(s)$ im Modell, der die beobachteten Objektstrukturen im gewählten Systemausschnitt zeigt.
- Ein Modell ist korrekt, wenn für jeden validen Zustand s und jede Regelanwendung, die $G(s)$ in einen Graphen H überführt, auch ein Implementierungsschritt von s nach s' mit $H = G(s')$ existiert.
- Ein Modell ist vollständig, wenn für jeden validen Zustand s ein Implementierungsschritt nach Zustand s' zu einem Schritt $G(s)$ nach $G(s')$ im Modell führt.

Korrektheit und Vollständigkeit des vorgestellten Verfahrens

- Extrahierte Modelle sind i.a. nicht korrekt, da nur das beobachtete Verhalten im Modell abgebildet wird.
  - *Bedingungen auf nicht beobachteten Objekten werden z.B. nicht im Modell reflektiert.*
- Extrahierte Modelle sind i.a. nicht vollständig, da nicht das gesamte Implementierungsverhalten beobachtet wird.
  - *Eingeschränkte Vollständigkeit: Das gesamte beobachtete Verhalten soll im Modell abgebildet sein.*
  - *Die Beobachtungen sollten also möglichst vollständig sein, z.B. den Code komplett überdecken.*

Extraktionswerkzeug für Visual Contracts



[AHK18]

Evaluation des Ansatzes

Evaluation des Toolings beschrieben in [AHK18]:

- In einem Beispielszenario *Precision* (Korrektheit) von 1 und *Recall* (Vollständigkeit) zwischen 0,3 und 1 (stark korreliert mit Branch Coverage)
- Studenten konnten Bugs anhand von Visual Contracts signifikant öfter erkennen und lokalisieren als wenn sie die gleiche Information textuell erhalten haben
- Anwendbarkeit auf reale Systeme
  - *NanoXML: 24 Klassen, 5605 Testfälle, > 2000 VC Instanzen*
  - *JHotDraw: 243 Klassen, 405 Testfälle, 135 VC Instanzen*

Zusammenfassung

- Das folgende Verfahren ist weitgehend automatisiert:
  1. *Extraktion von Visual Contract Instanzen aus Java-Programmen*
  2. *Ableitung von Visual Contracts aus der gefundenen Menge von Instanzen*
  3. *Optimierung von Visual Contracts*
- Vorteile:
  - *Modell muss nicht händisch erzeugt werden.*
  - *Modell ist immer konsistent zum beobachteten Verhalten.*
  - *Modell ist ausführbar (anwendbare Transformationsregeln).*
- Nachteile:
  - *Modell bildet das Programmverhalten eventuell nicht korrekt und vollständig ab.*
  - *Visual Contracts reflektieren die Kontrollstruktur nicht vollständig.*

Literatur

- [AH14] Abdullah M. Alshanqiti, Reiko Heckel: Towards Dynamic Reverse Engineering Visual Contracts from Java, ECEASST 67, 2014
- [AH16] Abdullah M. Alshanqiti, Reiko Heckel: Extracting Visual Contracts from Java Programs. ASE 2015: 104–114
- [AHK18] Abdullah M. Alshanqiti, Reiko Heckel, Timo Kehrer: Inferring visual contracts from Java programs. Journal on Automated Software Engineering, 25(4), 745–787, 2018
- [BHE09] Denes Bisztray, Reiko Heckel, Hartmut Ehrig: Verification of Architectural Refactorings: Rule Extraction and Tool Support, ECEASST 16, 2009
- [TELW10] Gabriele Taentzer, Claudia Ermel, Philip Langer, Manuel Wimmer: Conflict Detection for Model Versioning Based on Graph Modifications. Graph Transformations – 5th International Conference, ICGT 2010, Springer, LNCS,171–186