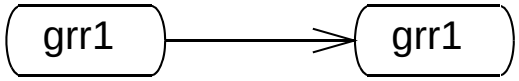
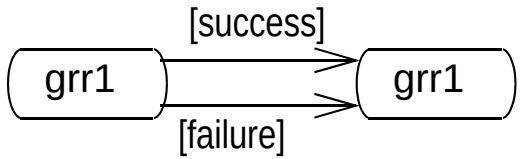
Notationelle Abkürzungen:

- eine Transition ohne guard:



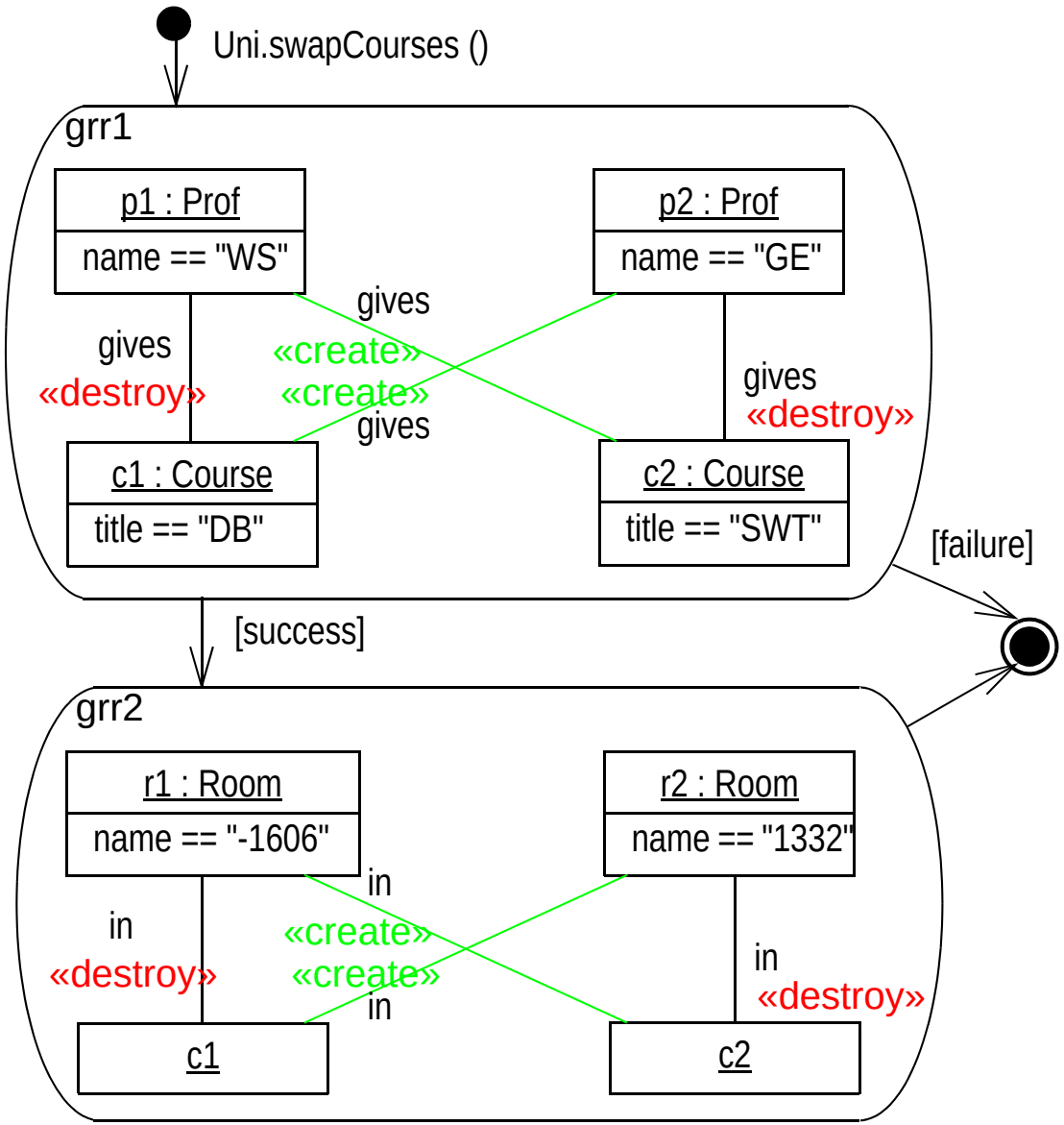
ist eine notationelle Abkürzung für eine [success] und eine [failure] Transitionen:



- wird ein Knoten erzeugt oder gelöscht
  => anhängende Kanten werden auch erzeugt bzw. gelöscht

  => «create» bzw. «destroy» Marker an solchen Kanten können wegfallen

Variablenbindung in Story Pattern



- Story Pattern können leicht sehr komplex werden =>
- aufspalten in mehrere Story Pattern
- die verschiedenen Story Pattern sollen auf der "gleichen Stelle" arbeiten
- häufiges Problem
- spezielles Sprachkonstrukt:

gebundene Variablen

- Zur Kennzeichnung werden bei gebundenen Variablen die Typbezeichnung weggelassen
- in grr2 bezeichnen c1 und c2 genau die Kurse, die in grr1 gefunden wurden

Semantik von gebundenen Variablen:

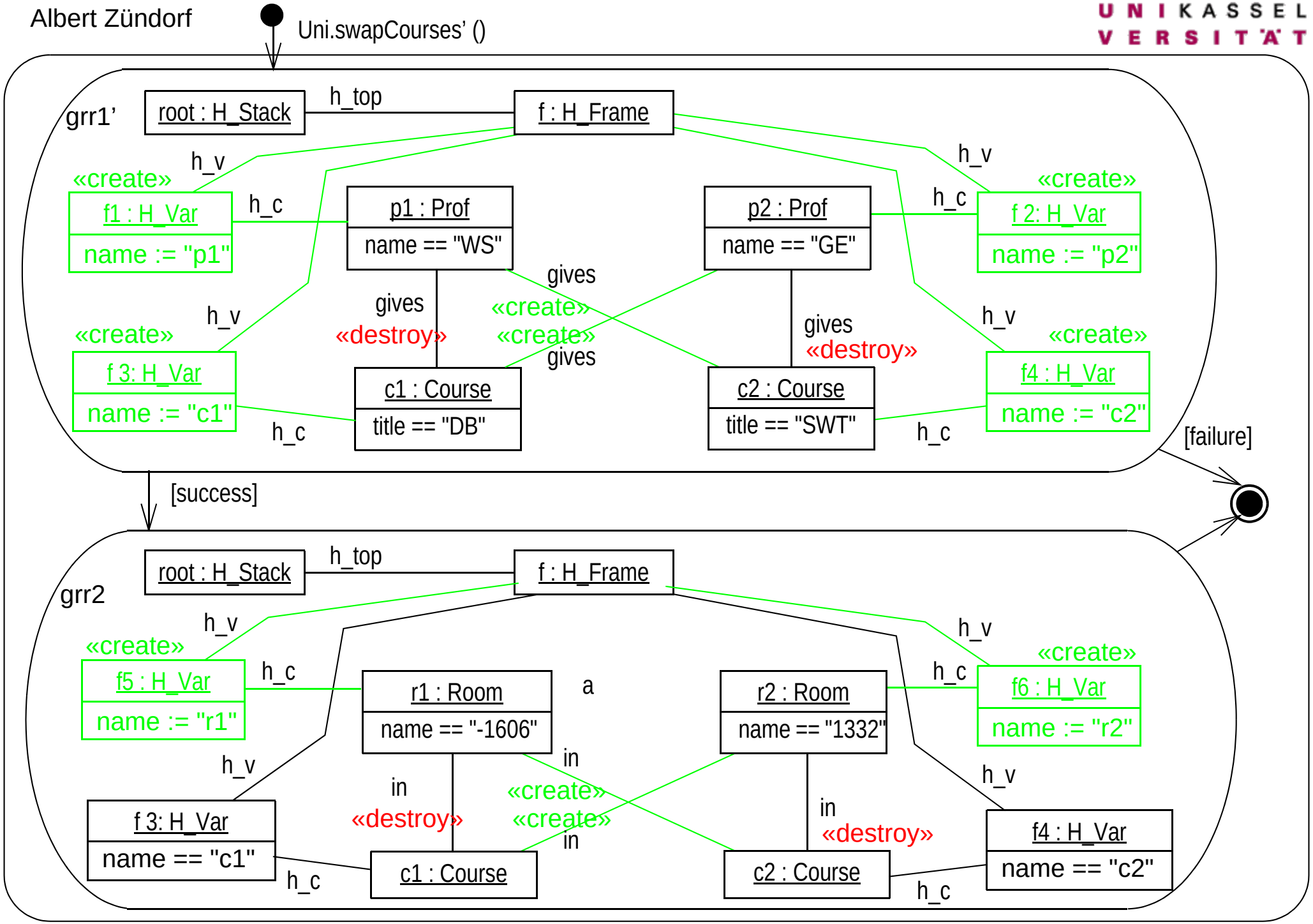
Idee:

- jedes Story Pattern wird implizit um Ausschnitt aus dem Prozedurkeller erweitert.
- für jedes ungebundene Objekt der Regel für das noch kein H_Var-Eintrag auf dem aktuellen Frame existiert, wird die Regel um die Erzeugung eines geeigneten H_Var-Objekts erweitert.
- für jedes gebundene Objekt wird die Regel um die Suche nach dem zugehörigen H_Var-Objekt erweitert

Effekt:

- beim Finden von Objekten wird der Match auf dem Prozedurkeller in einer Variablen vermerkt
- beim wiederverwenden durch gebundene Variablen bezeichnet die zugehörige Variable das gemeinte Objekt eindeutig

Beispiel: (nächste Folie)


Uni.swapCourses' ()
grr1'
root : H_Stack
h_top
f : H_Frame
h_v
«create»
f1 : H_Var
name := "p1"
h_c
p1 : Prof
name == "WS"
p2 : Prof
name == "GE"
f 2: H_Var
name := "p2"
gives
«destroy»
«create»
f 3: H_Var
name := "c1"
c1 : Course
title == "DB"
c2 : Course
title == "SWT"
f4 : H_Var
name := "c2"
[failure]
[success]
grr2
f5 : H_Var
name := "r1"
r1 : Room
name == "-1606"
r2 : Room
name == "1332"
f6 : H_Var
name := "r2"
a
in
f 3: H_Var
name == "c1"
c1 : Course
c2 : Course
f4 : H_Var
name == "c2"

Achtung:

- Variable auf dem Prozedurkeller existiert NUR, wenn sie vorher durch ein anderes Story Pattern oder durch Parameterübergabe angelegt wurde.
  (Da gibt's bei verzweigten Story Diagrammen schnell einen Pfad auf dem die Variable nicht definiert wurde)

- zusätzlich legen wir fest:

  1. das Laufzeitsystem legt auf dem aktuellen Frame per default eine Variable this an, die auf das Objekt zeigt, auf dem die auszuführende Methode aufgerufen wurde

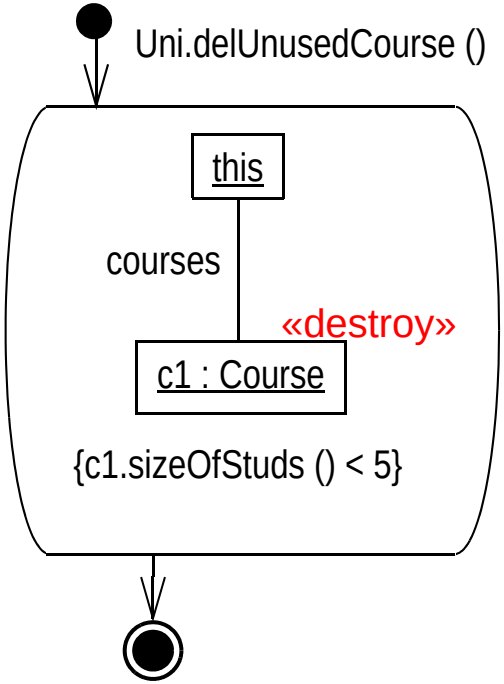
  2. aus Effizienzgründen verlangen wir das in jedem Story Pattern alle ungebundenen Variablen von den gebundenen Variablen über eine Folge von enthaltenen Links erreicht werden können

Forderung 2 macht die explizite Verwaltung von Extensionen überflüssig

(bzw. halst sie dem Modellierer auf :)

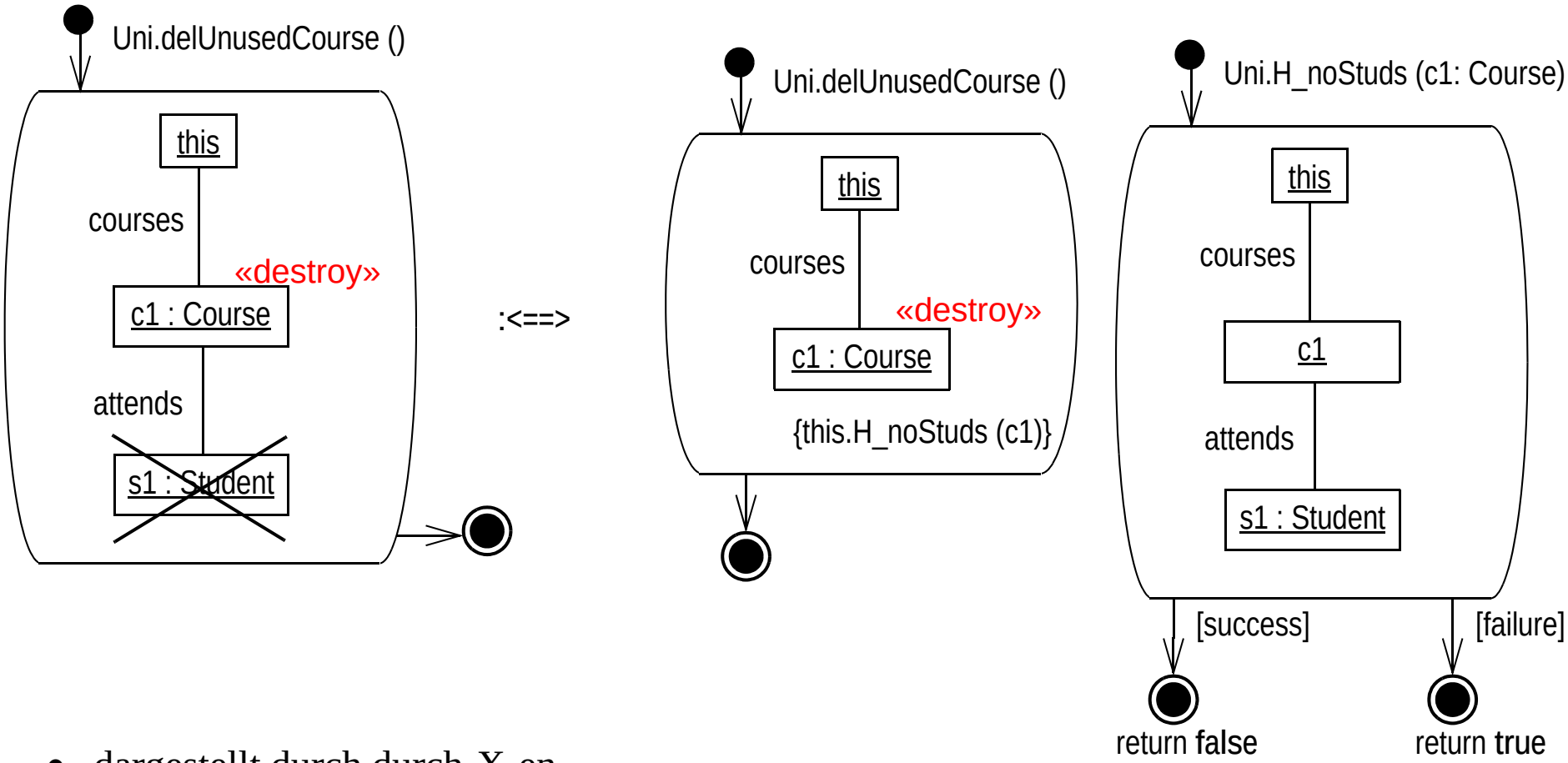
Weitere Sprachelemente von Story Patterns:

boolesche Constraints:



- story patterns dürfen eine beliebige Anzahl von booleschen Bedingungen in geschweiften Klammern enthalten
- ein Teilgraph ist erst dann eine gültige Anwendungsstelle für das Story Pattern, wenn alle Constraints zu true ausgewertet werden können.

negative Knoten:

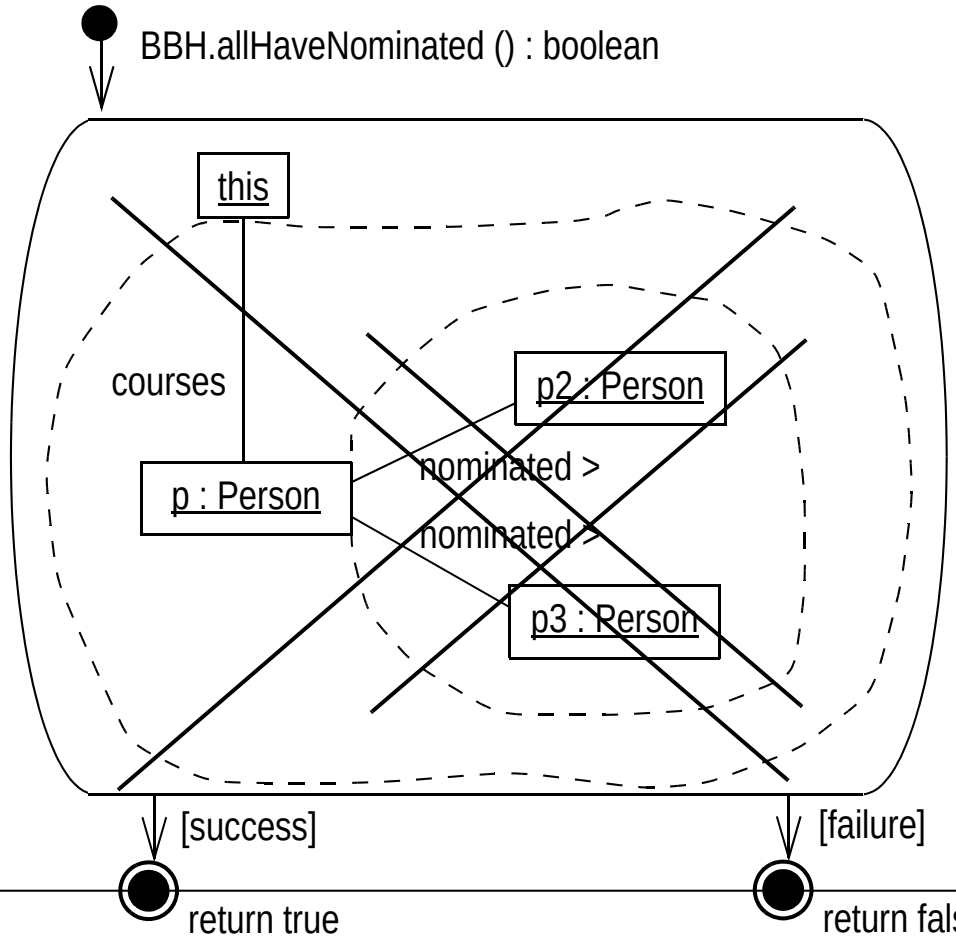


- dargestellt durch durch-X-en
- Übersetzung durch eigene Methode und ein zusätzliches Constraint

mehrere negative Knoten:

- negative Knoten werden einzeln überprüft
- jeder einzelne ergibt eine eigenes spezial Constraint

komplexe negative Anwendungsbedingungen:

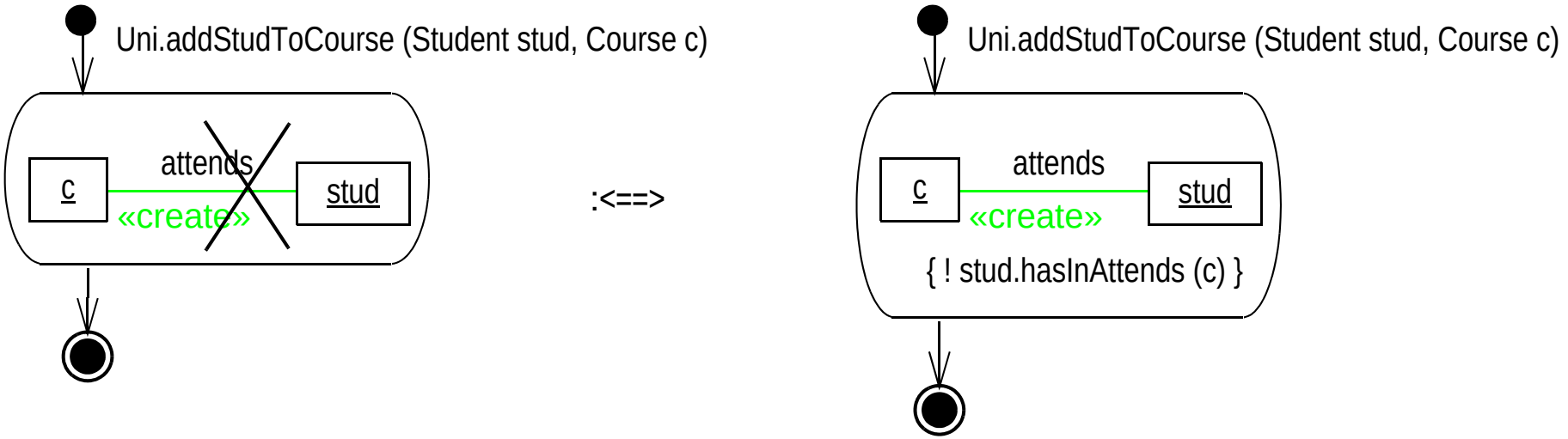


- Regel enthält Teilgraph der nicht vorhanden sein darf
- Markierung des Teilgraphs durch "Lasso"
- Schachtelbar

- ZU UNÜBERSICHTLICH
- nicht in Story Diagrammen enthalten

- vom Benutzer zu simulieren:
  { ! p.hasNominated () }

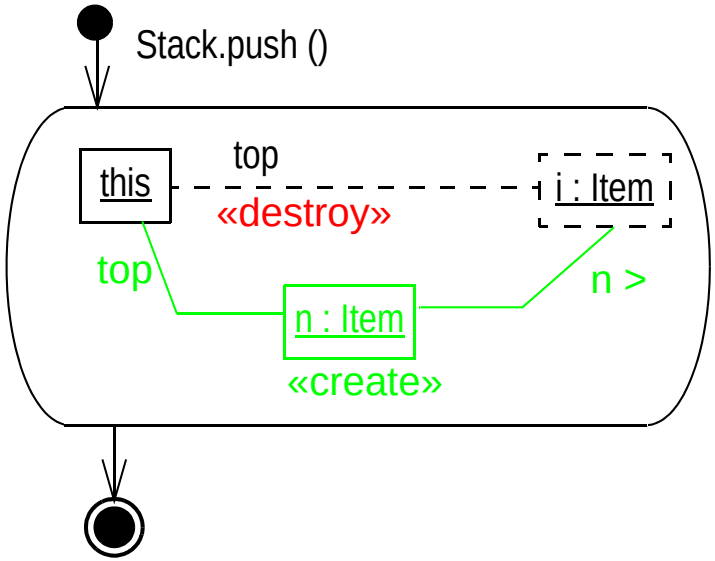
negative Kanten:



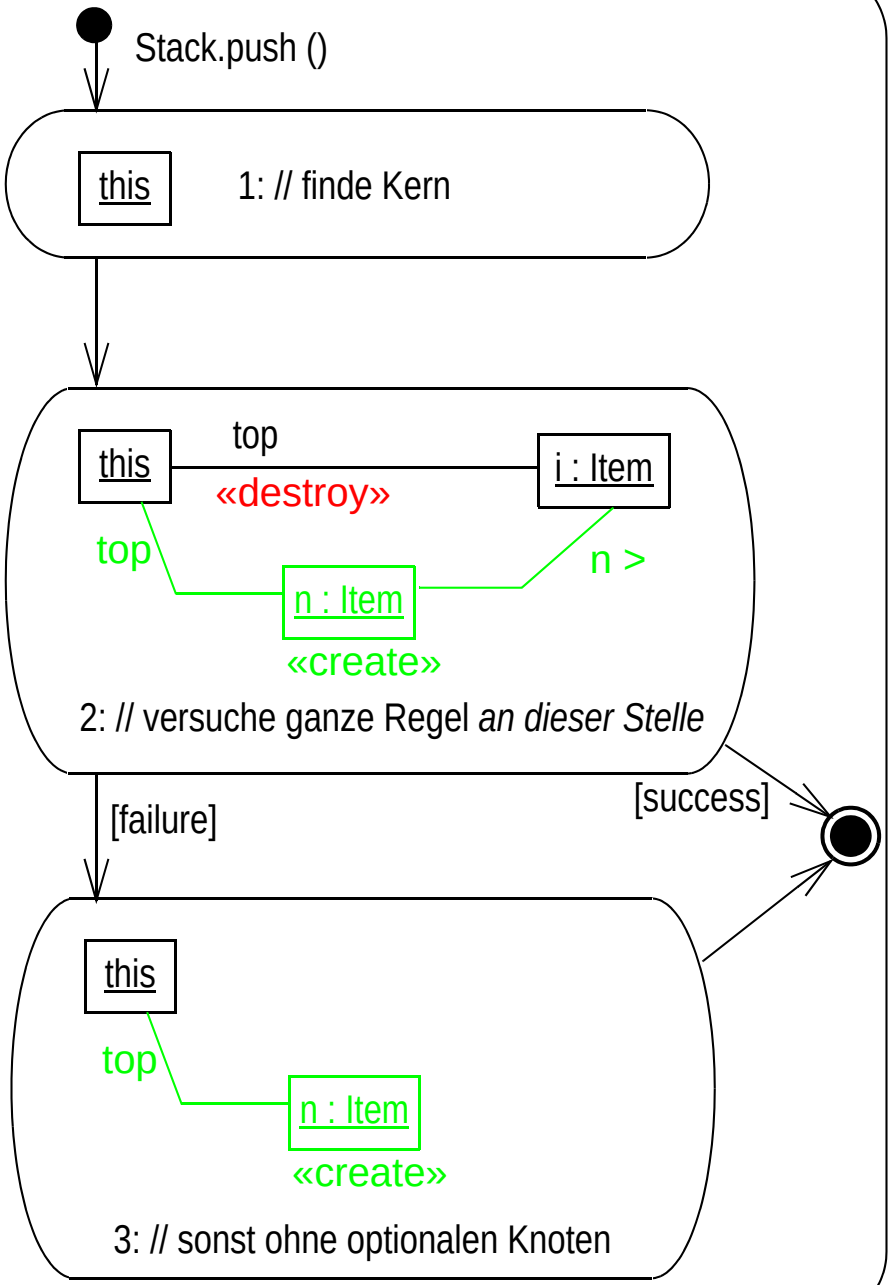
- negative Kante darf nicht vorhanden sein
- wird durch entsprechende Zugriffsmethode realisiert
- jede negative Kante wird einzelnd überprüft

- negative Kante zu einem negativen Knoten ist verboten

optionale Knoten:

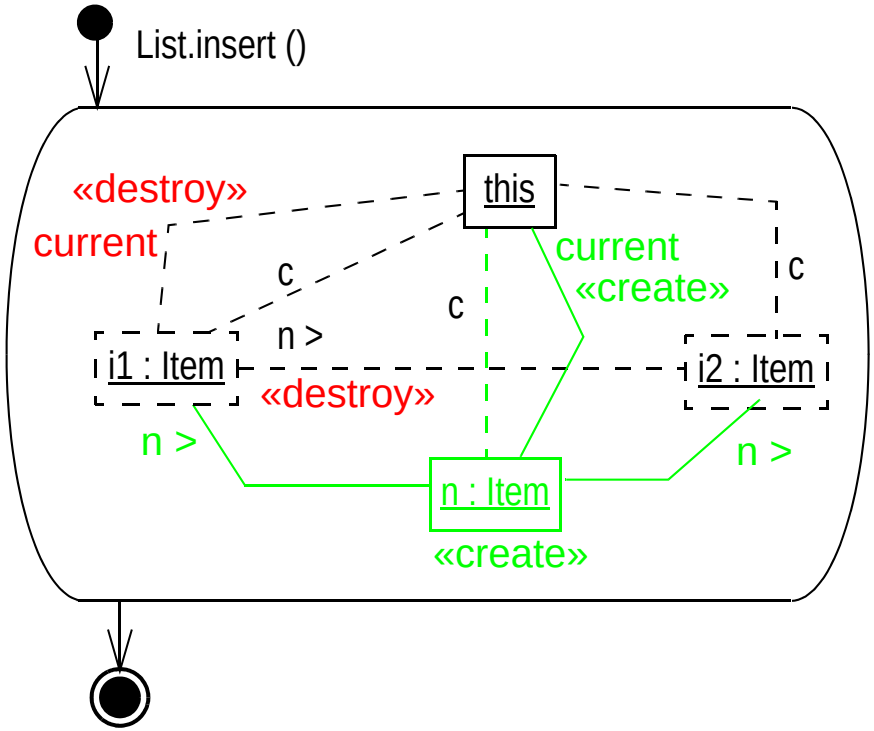


:<==>

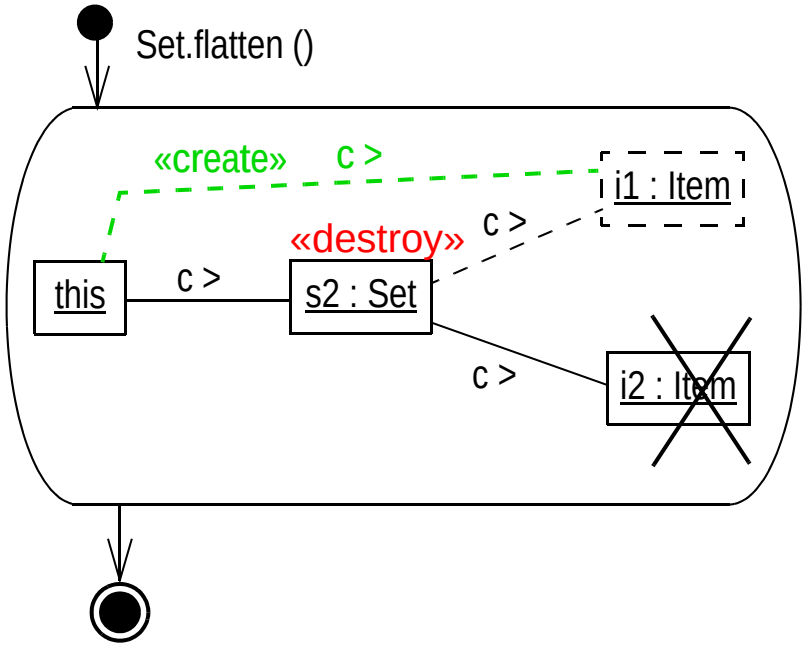


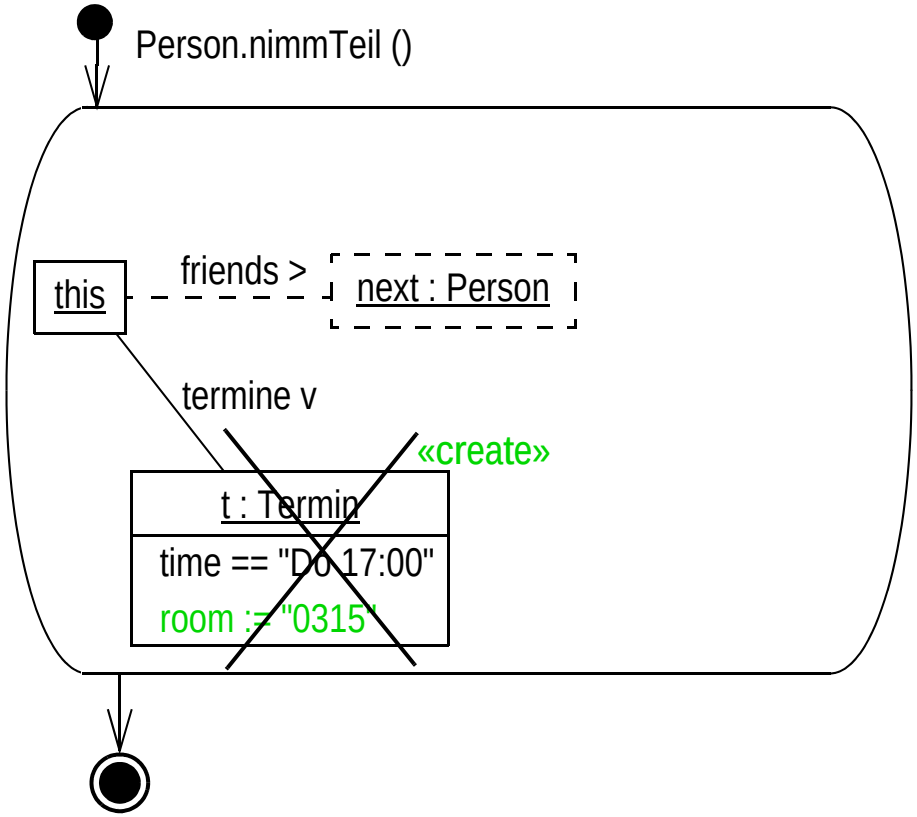
- erst Kernanwendungstelle suchen
- schrittweise um optionale Knoten erweitern, wenn möglich
- Regel für gefundene Teile ausführen
- viele optionale Knoten führen zur kombinatorischen Explosion
- geht auch mit Kanten

Kanten zwischen zwei optionalen Knoten sind VERBOTEN:



- die verschiedenen optionalen Elemente sind zunächst gleichwertig
- der Matcher könnte zuerst i2 über einen beliebigen c Link machen
- dann findet er kein i1 vor i2 mit ’nem current Link zu this
- macht ja nichts, i1 ist ja optional, lassen wir weg
- neues Item n wird eventuell nicht an der aktuellen Listenposition eingefügt wie beabsichtigt
- alles in allem passieren seltsame Dinge
- => dann lassen wir das mal besser
- (ist auch ziemlich schwer zu implementieren :)





optionale und negative Knoten in einer Regel:

- Wieviele Elemente darf s2 enthalten?
- Faustregel> die negativen Elemente zuletzt (also darf s2 bis zu einem Element enthalten)