

Graphersetzungsregeln

(Theorie und Verwendung)

Motivation:

- intuitive Verständlichkeit
- Ausdrucksmächtigkeit
- problemnahe Modellierung
- formal definierte Semantik

Theorie:**Def. 1: gerichtete, attributierte, knoten- und kantenmarkierte (gakk) Graphen**

$G := (NL, EL, A, N, E, l, av)$ mit

NL endliche Menge von Knotenmarkierungen / -typen / -labeln

EL endliche Menge von Kantenmarkierungen / -typen / -labeln

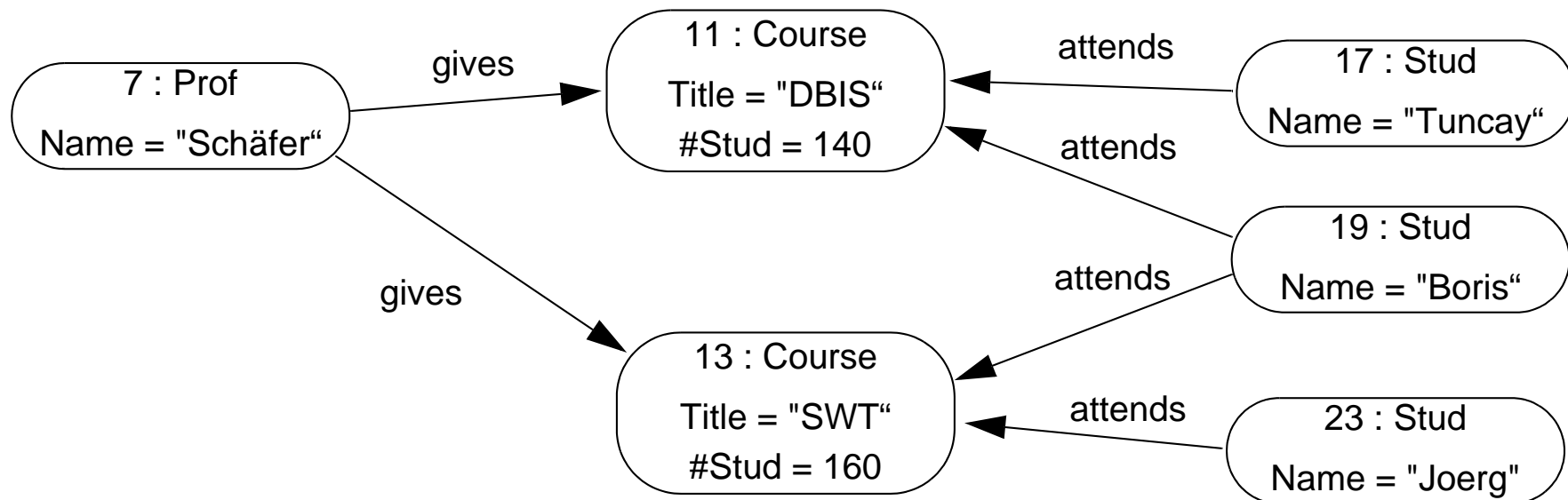
A endliche Menge von Attribut(nam)en

N endliche Menge von Knoten(bezeichnen)

E $\subseteq N \times EL \times N$

l $N \rightarrow NL$

av $N \times A \rightarrow \{\text{true, false}\} \cup N \cup \text{CHAR}^* \cup \dots \cup P(N) \cup P(\text{CHAR}^*) \cup P(\dots)$

Bsp. 1: gakk Graph:

$NL = \{ \text{Prof, Course, Stud} \}$ $EL = \{ \text{gives, attends} \}$ $A = \{ \text{Name, Title, #Stud} \}$

$N = \{ 7, 11, 13, 17, 19, 23 \}$

$E = \{ (7, \text{gives}, 11), (7, \text{gives}, 13), (17, \text{attends}, 11), (19, \text{attends}, 11), (19, \text{attends}, 13), (23, \text{attends}, 13) \}$

$l_G = \{ 7 \rightarrow \text{Prof}, 11 \rightarrow \text{Course}, 13 \rightarrow \text{Course}, 17 \rightarrow \text{Stud}, 19 \rightarrow \text{Stud}, 23 \rightarrow \text{Stud} \}$

$av = \{ (7, \text{Name}) \rightarrow \text{"Schäfer"}, (11, \text{Title}) \rightarrow \text{"DBIS"}, (11, \text{\#Stud}) \rightarrow 140, (13, \text{Title}) \rightarrow \text{"SWT"}, (13, \text{\#Stud}) \rightarrow 160, (17, \text{Name}) \rightarrow \text{"Tuncay"}, (19, \text{Name}) \rightarrow \text{"Boris"}, (23, \text{Name}) \rightarrow \text{"Joerg"} \}$

Def. 4: Abkürzungen

- Mit GraphClass (NL, EL, A) bezeichnen wir die Menge aller Graphen über NL, EL, A
- Im folgenden seien NL, EL, A gegeben und $GC := \text{GraphClass}(NL, EL, A)$
- Ein Graph $G \in GC$ wird vereinfachend nur mit (N, E, l, av) angegeben
- Sei $G=(N,E,l,av) \in GC$, dann bezeichnet
 - $N_G := N$ die Knotenmenge von G
 - $E_G := E$ die Kantenmenge von G
 - $l_G := l$ die Knotenmarkierungsfunktion von G
 - $av_G := av$ die Attributierungsfunktion von G

Def. 5: Basisgraphersetzungsregeln

Eine Graphersetzungsregel ist ein Paar von Graphen (LG, RG) mit $LG, RG \in GC$ und LG und RG sind konsistent markiert,

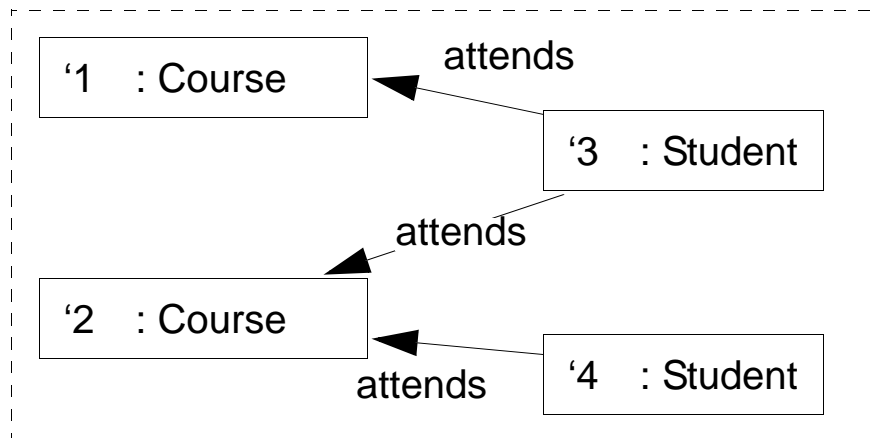
$$\text{d.h. } l_{LG}|_{N_{LG} \cap N_{RG}} = l_{RG}|_{N_{LG} \cap N_{RG}}$$

Sei $grr := (LG, RG)$ gegeben, dann bezeichnen wir mit:

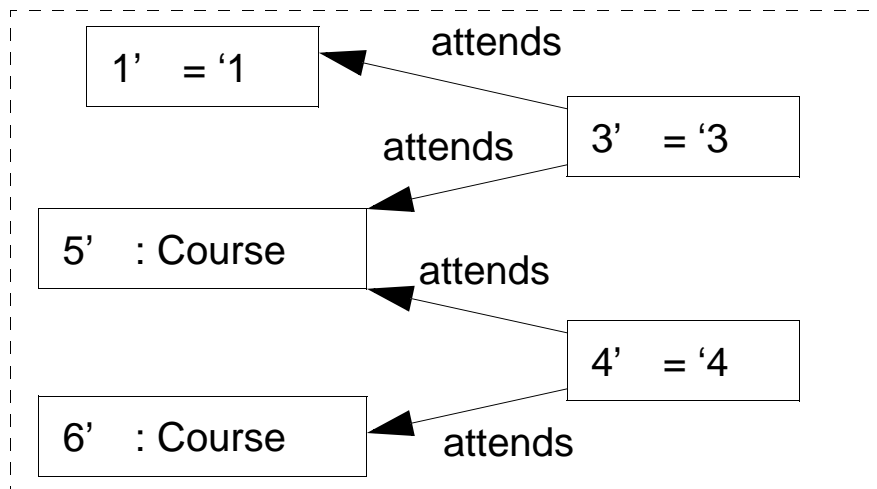
- $DelN_{grr} := N_{LG} - N_{RG}$ die Menge der von grr zu löschenden Knoten
- $DelE_{grr} := E_{LG} - E_{RG}$ die Menge der von grr zu löschenden Kanten
- $CoreN_{grr} := N_{LG} \cap N_{RG}$ die Kern- oder Kontextknoten von grr
- $AddN_{grr} := N_{RG} - N_{LG}$ die Menge der von grr neu zu erzeugenden Knoten
- $AddToAddE_{grr} := E_{RG} \cap (AddN_{grr} \times EL \times AddN_{grr})$
- $AddToCoreE_{grr} := E_{RG} \cap (AddN_{grr} \times EL \times CoreN_{grr})$
- $CoreToAddE_{grr} := \underline{E_{RG}} \cap (CoreN_{grr} \times EL \times AddN_{grr})$
- $CoreToCoreE_{grr} := E_{RG} \cap (CoreN_{grr} \times EL \times CoreN_{grr})$

Bsp. 5: Basisgraphersetzungsregel:

production grr =



::=



condition '1.Title = "DBIS"; '2.Title = "SWT";

transfer 5'.Title := "GT"; 6'.Title := "PSP";

end;

$$N_{LG} = \{ 1, 2, 3, 4 \}$$

$$E_{LG} = \{ (3, \text{attends}, 1), (3, \text{attends}, 2), (4, \text{attends}, 2) \}$$

$$l_{LG} = \{ 1 \rightarrow \text{Course}, 2 \rightarrow \text{Course}, 3 \rightarrow \text{Student}, 4 \rightarrow \text{Student} \}$$

$$av_{LG} = \{ (1, \text{Title}) \rightarrow \text{"DBIS"}, (2, \text{Title}) \rightarrow \text{"SWT"} \}$$

$$N_{RG} = \{ 1, 3, 4, 5, 6 \}$$

$$E_{RG} = \{ (3, \text{attends}, 1), (3, \text{attends}, 5), (4, \text{attends}, 5), (4, \text{attends}, 6) \}$$

$$l_{RG} = \{ 1 \rightarrow \text{Course}, 3 \rightarrow \text{Student}, 4 \rightarrow \text{Student}, 5 \rightarrow \text{Course}, 6 \rightarrow \text{Course} \}$$

$$av_{RG} = \{ (1, \text{Title}) \rightarrow \text{"DBIS"}, (5, \text{Title}) \rightarrow \text{"GT"}, (6, \text{Title}) \rightarrow \text{"PSP"} \}$$

$$\text{DelN}_{\text{grr}} = \{ 2 \}$$

$$\text{DelE}_{\text{grr}} = \{ (3, \text{attends}, 2), (4, \text{attends}, 2) \}$$

$$\text{CoreN}_{\text{grr}} = \{ 1, 3, 4 \}$$

$$\text{AddN}_{\text{grr}} = \{ 5, 6 \}$$

$$\text{AddToAddE}_{\text{grr}} = \{ \}$$

$$\text{AddToCoreE}_{\text{grr}} = \{ \}$$

$$\text{CoreToAddE}_{\text{grr}} = \{ (3, \text{attends}, 5), (4, \text{attends}, 5), (4, \text{attends}, 6) \}$$

$$\text{CoreToCoreE}_{\text{grr}} = \{ (3, \text{attends}, 1) \}$$

Ausführung von Graphersetzungsgesetzen

1. Anwendungsstelle suchen
2. Lösungen
3. Neu erzeugen

1. Suchen:

Suche Teilgraph der zu linker Regelseite passt

Def. 6: Teilgraph

Seien $TG, G \in GC$ gegeben.

TG ist Teilgraph von G (in Zeichen $TG \leq G$)

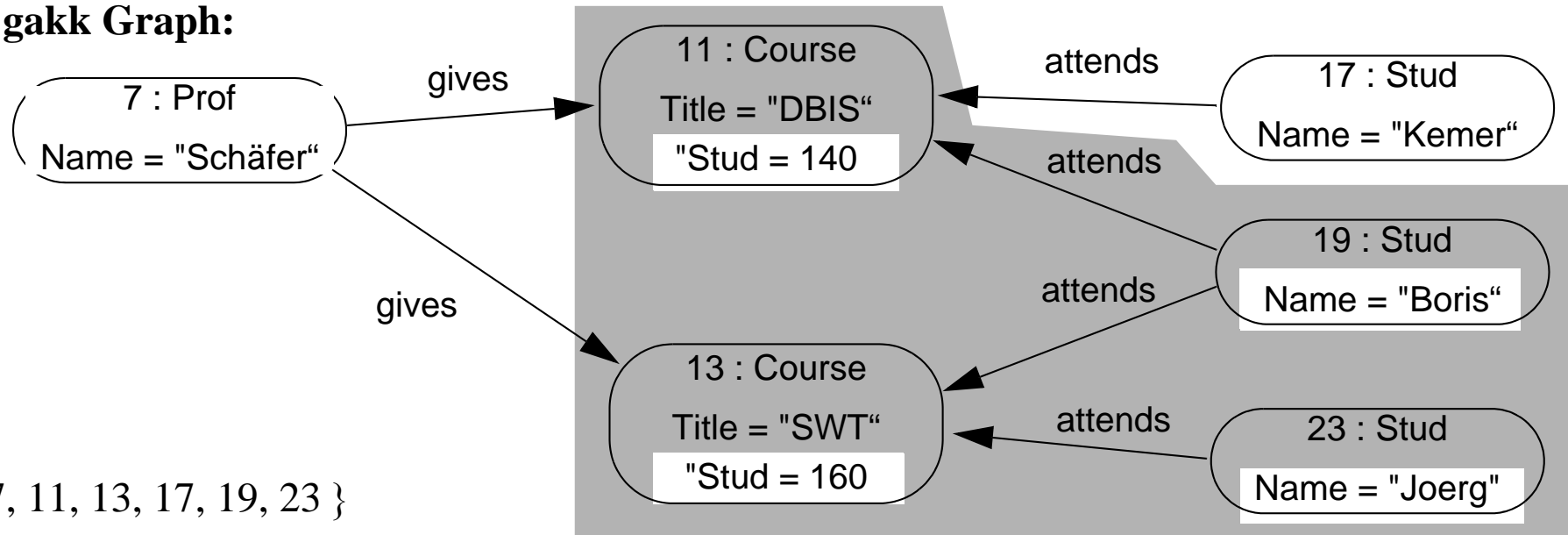
$:\Leftrightarrow$

$$N_{TG} \subseteq N_G$$

$E_{TG} \subseteq (E_G \cap (N_{TG} \times EL \times N_{TG}))$ Teilmenge der Kanten von G , die in TG liegt

$$l_{TG} = l_G|_{N_{TG}} \quad \text{Knotenmarkierungen stimmen überein}$$

$\forall ((n, a) \rightarrow v) \in av_{TG}$ gilt: $n \in N_{TG}$, $av_G(n, a) = v$

Bsp. 1:gakk Graph:

$$N_G = \{ 7, 11, 13, 17, 19, 23 \}$$

$$E_G = \{ (7, \text{gives}, 11), (7, \text{gives}, 13), (17, \text{attends}, 11), (19, \text{attends}, 11), (19, \text{attends}, 13), (23, \text{attends}, 13) \}$$

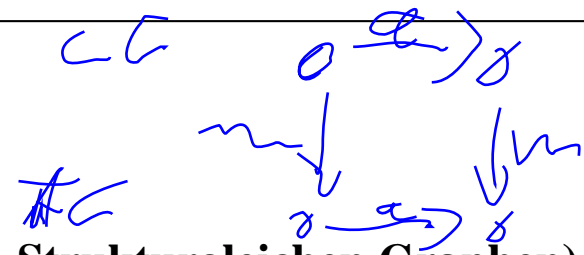
$$l_G = \{ 7 \rightarrow \text{Prof}, 11 \rightarrow \text{Course}, 13 \rightarrow \text{Course}, 17 \rightarrow \text{Stud}, 19 \rightarrow \text{Stud}, 23 \rightarrow \text{Stud} \}$$

$$av_G = \{ (7, \text{Name}) \rightarrow \text{"Schäfer"}, (11, \text{Title}) \rightarrow \text{"DBIS"}, (11, \text{\#Stud}) \rightarrow 140, (13, \text{Title}) \rightarrow \text{"SWT"}, \\ (13, \text{\#Stud}) \rightarrow 160, (17, \text{Name}) \rightarrow \text{"Tuncay"}, (19, \text{Name}) \rightarrow \text{"Boris"}, (23, \text{Name}) \rightarrow \text{"Joerg"} \}$$

$$N_{TG} = \{ 11, 13, 19, 23 \} \quad E_{TG} = \{ (19, \text{attends}, 11), (19, \text{attends}, 13), (23, \text{attends}, 13) \}$$

$$l_{TG} = \{ 11 \rightarrow \text{Course}, 13 \rightarrow \text{Course}, 19 \rightarrow \text{Stud}, 23 \rightarrow \text{Stud} \}$$

$$av_{TG} = \{ (11, \text{Title}) \rightarrow \text{"DBIS"}, (13, \text{Title}) \rightarrow \text{"SWT"} \}$$



Def. 7 Graphisomorphismen (Abbildung zwischen Strukturgleichen Graphen):

Seien $G_1, G_2 \in GC$ gegeben.

Eine **bijektive** Funktion $match: G_1 \rightarrow G_2$ heißt Graphisomorphismus

: \Leftrightarrow

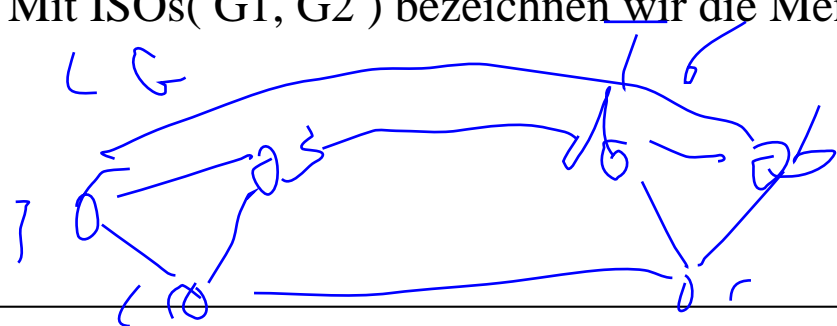
$match(N_{G_1}) = N_{G_2}$ wobei $match(N) := \{ match(n) \mid n \in N \}$

$match(E_{G_1}) = E_{G_2}$ wobei $match(E) := \{ (match(sn), el, match(tn)) \mid (sn, el, tn) \in E \}$

$\forall n \in N_{G_1}$ gilt $l_{G_1}(n) = l_{G_2}(match(n))$ die Knotenmarkierungen bleiben erhalten

$\forall n \in N_{G_1}, a \in A$ mit $av_{G_1}(n, a) = X$ gilt $av_{G_2}(match(n), a) = X$

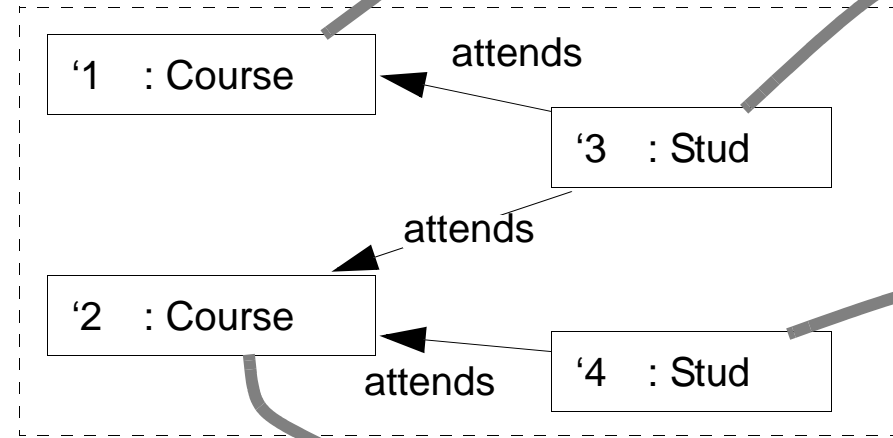
Mit $ISOs(G_1, G_2)$ bezeichnen wir die Menge aller Graphisomorphismen von N_{G_1} zu N_{G_2}



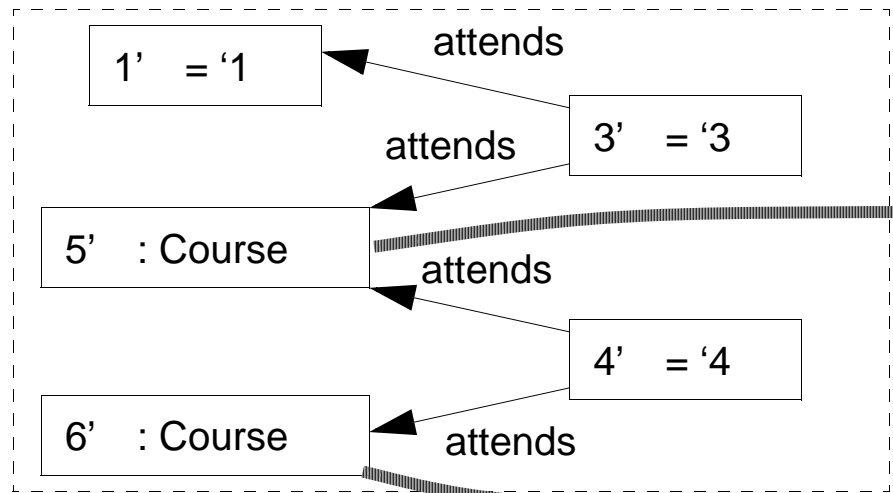
1	a	1	b
2	b	2	c
3	c	3	d

match = { 1 -> 11, 2 -> 13, 3 -> 19, 4 -> 23 }

production grr =

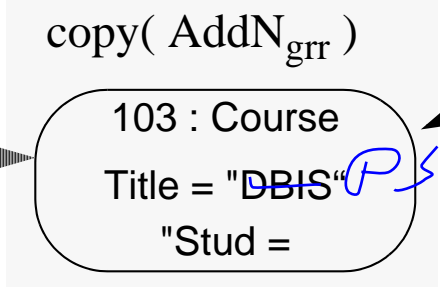
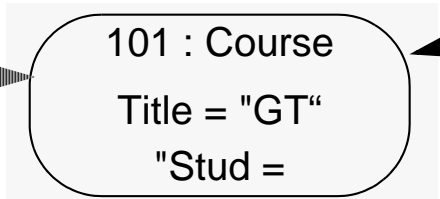
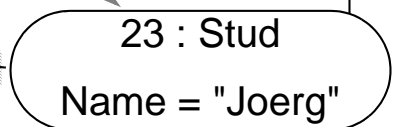
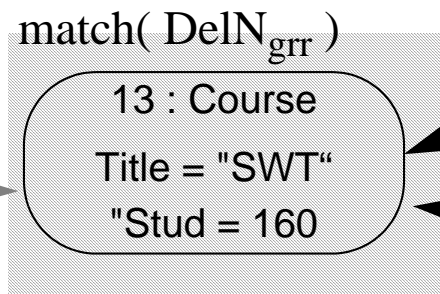
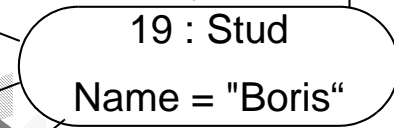
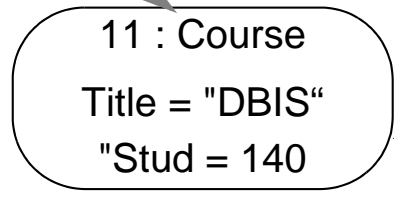


::=



condition '1.Title = "DBIS"; '2.Title = "SWT";
transfer 5'.Title := "GT"; 6'.Title := "PSP";

end;



copy = { 5 -> 101, 6 -> 103 }

Def. 8 Semantik von Graphersetzungsgeln:

$\text{Sem}(\text{grr}) \subseteq \text{GC} \times \text{GC}$ oder auch $\text{Sem}[\text{grr}] : \text{GC} \rightarrow \mathcal{P}(\text{GC})$

mit $(G1, G2) \in \text{Sem}(\text{grr})$ oder auch $G2 \in \text{Sem}[\text{grr}](G1)$

: \Leftrightarrow

1. (Suchen)

$\exists \text{TG} \in \text{GC}$ mit $\text{TG} \leq G1$ und $\exists \text{match} \in \text{ISOs}(\text{LR}, \text{TG})$ (TG heißt Anwendungsstelle)

2. (Löschen)

$\exists \text{IG} \leq G1$ mit

$$N_{\text{IG}} = N_{G1} - \text{match}(\text{Del}N_{\text{grr}})$$

$$E_{\text{IG}} = (E_{G1} - \text{match}(\text{Del}E_{\text{grr}})) \cap (N_{\text{IG}} \times \text{EL} \times N_{\text{IG}})$$

$$l_{\text{IG}} = l_{G1} \Big|_{N_{\text{IG}}}$$

$$\text{av}_{\text{IG}} = \text{av}_{G1} \Big|_{N_{\text{IG}} \times A}$$

3. (Erzeugen)

(neue Knoten)

$\exists N_{\text{new}}$ mit $N_{\text{new}} \cap N_{G1} = \emptyset$ und \exists bijektive Funktion copy mit $\text{copy}(\text{Add}N_{\text{grr}}) = N_{\text{new}}$

wobei copy durch den match eindeutig festgelegt sein soll, d.h.

$$\text{match} = \text{match}' \Rightarrow \text{copy} = \text{copy}'$$

(sonst Menge von Ergebnisgraphen nur mit unterschiedlichen Nummern für N_{new})

und

$$N_{G2} = N_{IG} \cup N_{\text{new}}$$

$$E_{G2} = E_{IG} \cup \text{copy}(\text{AddToAdd}E_{\text{grr}})$$

$$\cup \{ (\text{copy}(sn), el, \text{match}(tn)) \mid (sn, el, tn) \in \text{AddToCore}E_{\text{grr}} \}$$

$$\cup \{ (\text{match}(sn), el, \text{copy}(tn)) \mid (sn, el, tn) \in \text{CoreToAdd}E_{\text{grr}} \}$$

$$\cup \text{match}(\text{CoreToCore}E_{\text{grr}})$$

$$l_{G2}(n) = l_{GR}(nr) \text{ falls } n = \text{copy}(nr) \text{ und } = l_{IG}(n) \text{ sonst}$$

$$av_{G2}(n, a) = av_{GR}(nr, a) \text{ falls } n = \text{copy}(nr) \text{ oder } n = \text{match}(nr) \text{ und } = av_{IG}(n, a) \text{ sonst}$$

Warum $\text{Sem}(\text{grr}) \subseteq \text{GC} \times \text{GC}$?

Bei $\text{Sem}[\text{grr}] : \text{GC} \rightarrow \mathcal{P}(\text{GC})$:

production $\text{grr} = \boxed{} \rightarrow \boxed{} ::= \boxed{} \quad \boxed{} \text{ end}$

$\text{Sem}[\text{grr}](\boxed{} \rightarrow \boxed{} \rightarrow \boxed{} \rightarrow \boxed{}) =$
 $\{ \boxed{} \quad \boxed{} \rightarrow \boxed{} \rightarrow \boxed{},$
 $\boxed{} \rightarrow \boxed{} \quad \boxed{} \rightarrow \boxed{},$
 $\boxed{} \rightarrow \boxed{} \rightarrow \boxed{} \quad \boxed{} \}$

$\text{Sem}(\text{grr} \ \& \ \text{grr}) = ??$

$\text{Sem}(\text{grr}) \subseteq \text{GC} \times \text{GC}$:

$\text{Sem}(\text{grr}) = \{ (\boxed{} \rightarrow \boxed{} \rightarrow \boxed{} \rightarrow \boxed{}, \boxed{} \quad \boxed{} \rightarrow \boxed{} \rightarrow \boxed{}),$
 $(\boxed{} \rightarrow \boxed{} \rightarrow \boxed{} \rightarrow \boxed{}, \boxed{} \rightarrow \boxed{} \quad \boxed{} \rightarrow \boxed{}),$
 $(\boxed{} \rightarrow \boxed{} \rightarrow \boxed{} \rightarrow \boxed{}, \boxed{} \rightarrow \boxed{} \rightarrow \boxed{} \quad \boxed{}),$
 $\dots \}$

Semantik unabhängig von Startgraph, einfachere Beschreibung von Kontrollstrukturen

z.B.: $\text{Sem}(\text{grr}_1 \ \text{or} \ \text{grr}_2) := \text{Sem}(\text{grr}_1) \cup \text{Sem}(\text{grr}_2)$

Einfache Kontrollstrukturen für Graphersetzungsregeln:

Def. 9 Zusammensetzen von Graphersetzungsregeln zu Graphtransformationen

- 1) Jede Graphersetzungsregel ist eine Graphtransformation
- 2) Seien $gt1$ und $gt2$ Graphtransformationen, dann ist auch
 - $gt1 \ \& \ gt2$ eine Graphtransformation mit

$$\text{Sem}(gt1 \ \& \ gt2) := \{ (G1, G3) \mid \exists G2 \text{ mit } (G1, G2) \in \text{Sem}(gt1) \text{ und } (G2, G3) \in \text{Sem}(gt2) \}$$
 - $gt1 \ \text{or} \ gt2$ eine Graphtransformation mit

$$\text{Sem}(gt1 \ \text{or} \ gt2) := \text{Sem}(gt1) \cup \text{Sem}(gt2)$$
 - $\text{def}(gt1)$ eine Graphtransformation mit

$$\text{Sem}[\text{def}(gt1)] := \{ (G, G) \mid \exists G2 \text{ mit } (G, G2) \in \text{Sem}(gt1) \}$$
 - $\text{not def}(gt1)$ eine Graphtransformation mit

$$\text{Sem}[\text{not def}(gt1)] := \{ (G, G) \mid \neg \exists G2 \text{ mit } (G, G2) \in \text{Sem}(gt1) \}$$

Alle anderen Kontrollstrukturen von Progres aus obigen Grundverknüpfungen ableitbar:

- $\text{Sem}(\underline{\text{choose}} \text{ gt1 } \underline{\text{else}} \text{ gt2 } \underline{\text{end}}) := \text{Sem}[\text{gt1 or (not def (gt1) \& gt2)}]$
sequentielles oder
- $\text{Sem}(\text{skip}) := \text{Sem}[\text{def(gt1) or not def(gt1)}]$
noop von Progres
- $\text{Sem}(\text{loop gt1 end}) := \text{Sem}(\text{choose gt1 \& loop gt1 end else skip end})$
rekursive Definition von Iteration
- $\text{Sem}(\text{fail}) := \text{Sem}[\text{not def(skip)}] = \{\}$
halt von Progres

Semantik von Progres, komplexere Sprachkonstrukte:

(Im folgenden Semantikdefinition mit obigen Basiskonstrukten)

Prozedur- und Sichtbarkeitskonzepte der Sprache Progres:

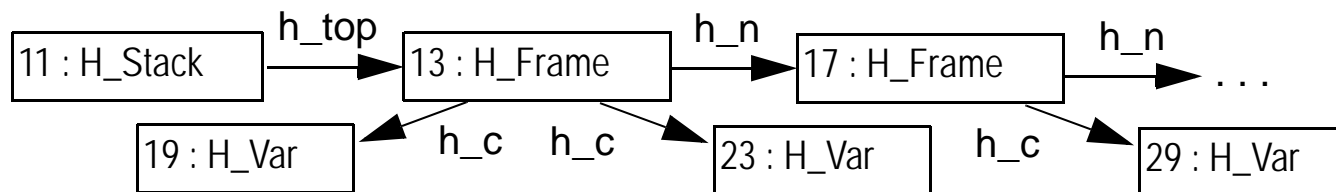
- Benennung von Graphtransformationen durch Transaktionen:
transaction InstallSWT = CreateProf & CreateCourse & SetProfForCourse end;
- Progres kennt lokale Variablen, In- und Out-Parameter für Operationen
- Progres erlaubt Rekursion

=> wir benötigen Prozedurkeller

- Keine geschachtelten Transaktionsdeklarationen (oder anderer Prozeduren)
- keine globalen Variablen (außer implizitem Arbeits-/Wirts-Graphen)

=> Static Link unnötig

Beispiel:



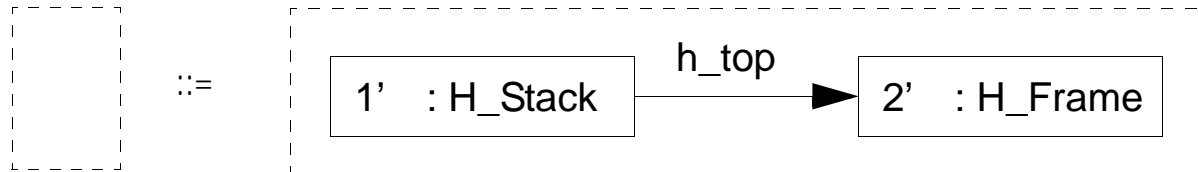
Ausführung einer Progres-Spezifikation (Programmstart):

Sei Spec eine bezügl. statischer Semantik fehlerfreie Progresspezifikation, dann gilt:

- $\text{Sem}(\text{Spec}) := \text{Sem}(\text{H_CreateStack} \ \& \ \text{Main})$

mit

production H_CreateStack =



end;

und

- Main ist eine benutzerdefinierte Operation in Spec
- OBDA. kommen H_CreateStack, H_Stack, h_top, H_Frame in Spec nicht vor

(Prozedurkeller wird gleich für Parameter und lokale Variablen benötigt)

Transaktionen (Prozedurkeller):

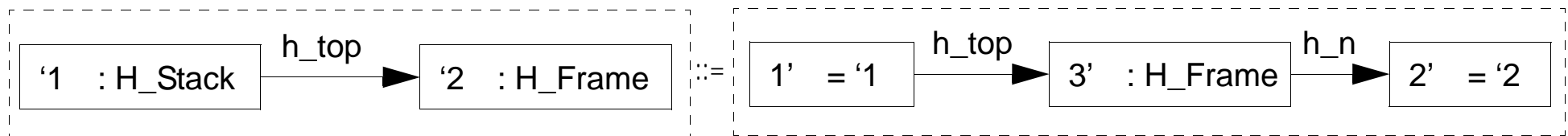
Sei gt eine Graphtransformation und $Spec$ enthalte Deklaration der Form:

transaction $T = gt$ end;

dann gilt:

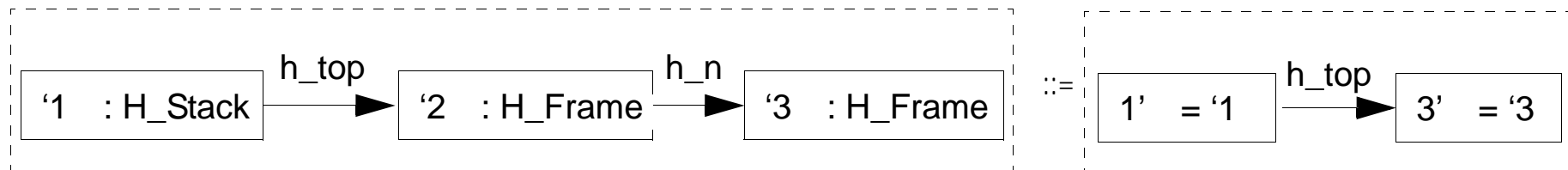
- $Sem(T) := Sem(H_PushFrame \ \& \ gt \ \& \ H_PopFrame)$
mit:

production $H_PushFrame =$



end;

production $H_PopFrame =$



end;

Bemerkung:

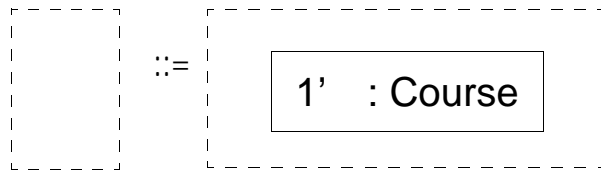
- OBDA gilt im folgenden, daß die zur Simulation eingeführten Knoten- und Kantentypen, Attribute und Operationsnamen in Spec nicht vorkommen (evtl. geeignet umbenennen)

Operationen mit (In-) Parametern:

(Parameterlose Produktionen benötigen keinen eigenen Frame, Aufruf wie Makro)

Spec enthalte folgende Produktion:

production CreateCourse(title : string) =



transfer 1'.Title := title; 1'.NoOfStuds := 0;
end;

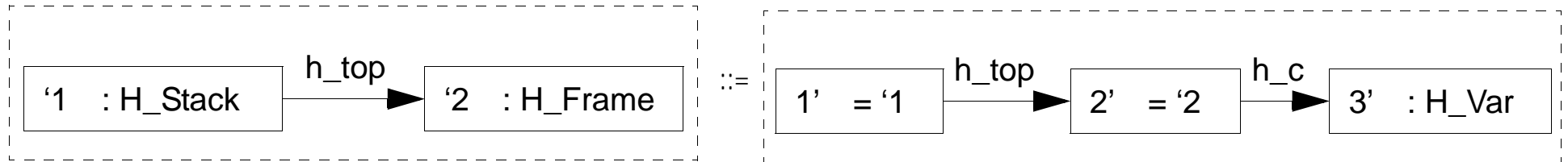
dann gilt:

Sem[CreateCourse("GT")] :=

Sem[H_PushFrame & H_PushStringVar_title_GT & H_CreateCourse_title & H_PopFrame]

mit:

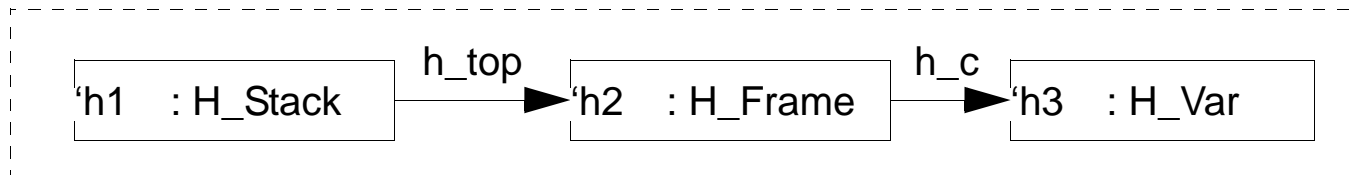
production H_PushStringVar_title_GT =



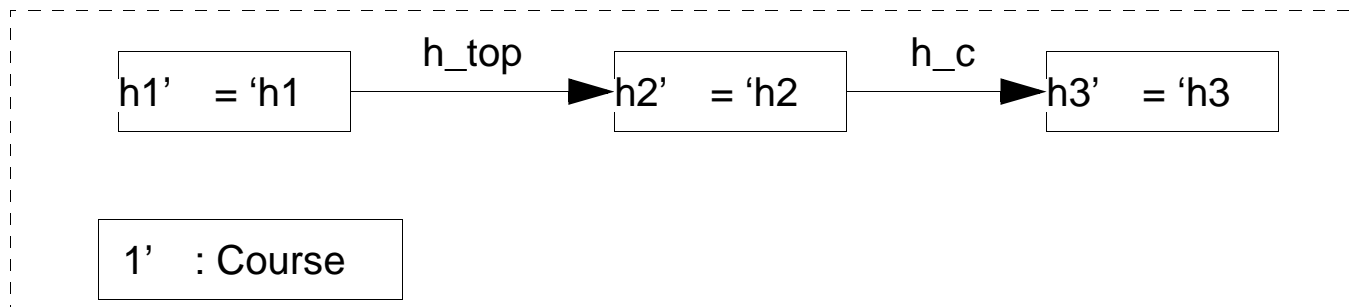
transfer 3'.H_VarName := "title"; 3'.H_StringAttr := "GT";

end;

production H_CreateCourse_title =



::=



condition 'h3.H_VarName = "title"

transfer 1'.Title := 'h3.H_StringAttr; 1'.NoOfStuds := 0;

end;

Allgemeine Behandlung von Parametern:

- Übergabe von Parametern simuliert durch "Parameter-" Knoten auf Prozedurkeller
- Zugriff auf Parameter simuliert durch Attributzugriff auf Parameterknoten.

Bemerkung:

- Obige Organisation des Prozedurkellers wie allgemein im Compilerbau üblich
- Hier vereinfachend: kein Static-Link notwendig
- Obige Simulation benötigt pro Aufrufstelle spezifische PushTVar_x_val-Operation
- Eine Progres Operation kann natürlich eine beliebige Zahl von In-Parametern unterschiedlicher Typen haben, Simulation analog (selbst :)

ACHTUNG !!!:

Attributnamen und Parameternamen haben gemeinsamen Namensraum, d.h. Parameternamen überdecken gleichnamige Attribute: (Gibt ne komische Fehlermeldung)

~~transfer 1'.Title := Title;~~

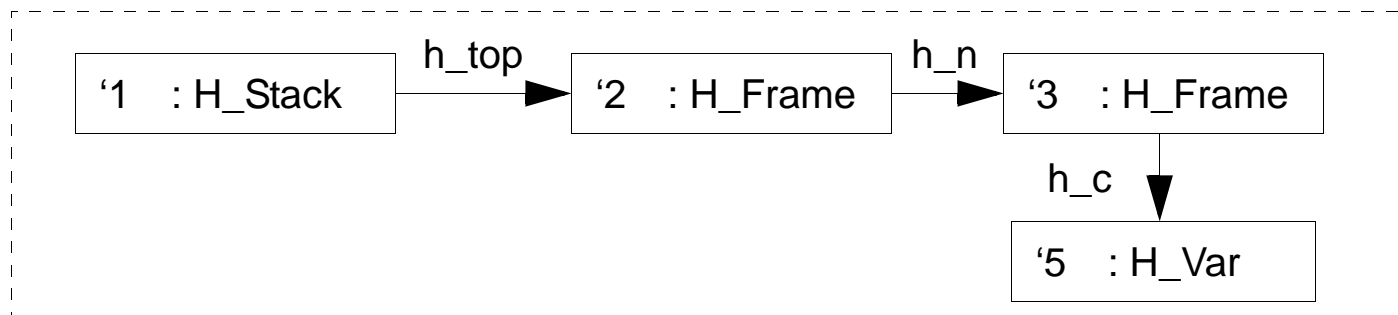
lokale Variablen:

- $\text{Sem}(\text{ use } x : T := \text{val do } \dots \text{Op}(x) \dots \text{end}) :=$

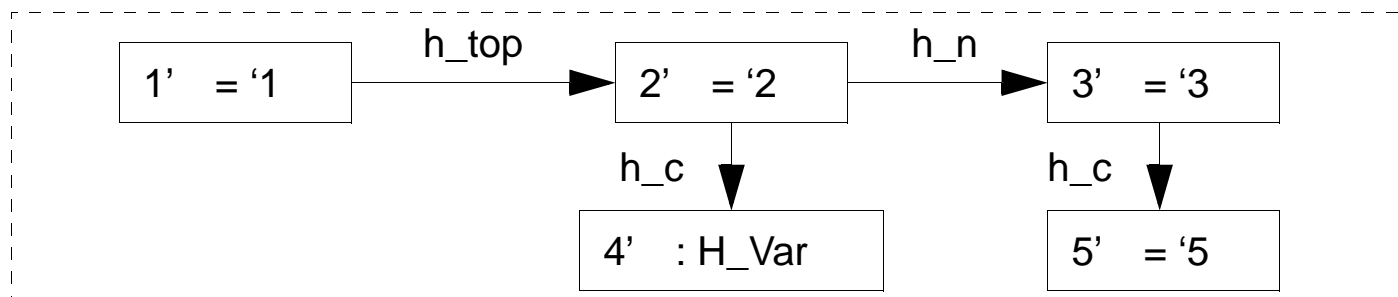
$\text{Sem}(\text{H_PushTVar_x_val} \ \& \ \dots \ (* \leftarrow \text{Behandlung der Variablendeklaration} *)$
 $\ \& \ \text{H_PushFrame} \ \& \ \text{H_PushTVarFromVar}(\text{"param"}, \text{"x"}) \ (* \text{ Variablenzugriff} *)$
 $\ \& \ \text{H_Op_param} \ \& \ \text{H_PopFrame})$

mit:

production $\text{H_PushTVarFromVar}(\text{dest_var}, \text{src_var} : \text{string}) =$



::=



condition '5.H_VarName = src_var;

transfer 4'.H_VarName := dest_var; 4'.H_TAttr := '5.H_TAttr;

end;

Allgemeine Behandlung lokaler Variablen:

- Für jede lokale Variable wird ein Variablen-Knoten im aktuellen Frame angelegt
- Bei Verwendung der Variablen wird auf den zugehörigen Variablen-Knoten zugegriffen

Bemerkung

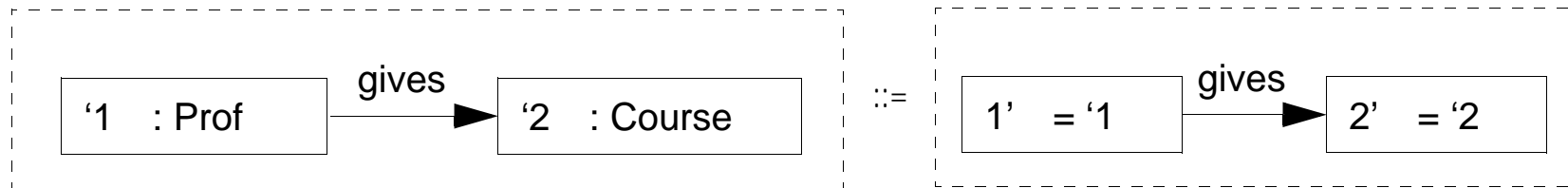
- In einem use-Statemente sind mehrere Variablen zulässig
- der initialisierende Ausdruck ist optional
- Wenn ein initialisierender Ausdruck angegeben ist kann die Typangabe entfallen
- use-Statements können an jeder Stelle in einer Transaktion stehen
- use-Statements können beliebig geschachtelt werden:

```
use    x, y := 0; (* zwei Integer-Variablen x und y *)  
        z : string;  
do  
    ...  
    use i, j : integer := 1 do ... end  
    ...  
end
```

(*eventuelle Namensüberdeckungen werden bei Simulation durch interne Umbenennung behandelt *)

Out-Parameter:

production GetProfOfCourse(title : string ; out name : string) =



condition '2.Title = title;

return name := 1'.Name;

end;

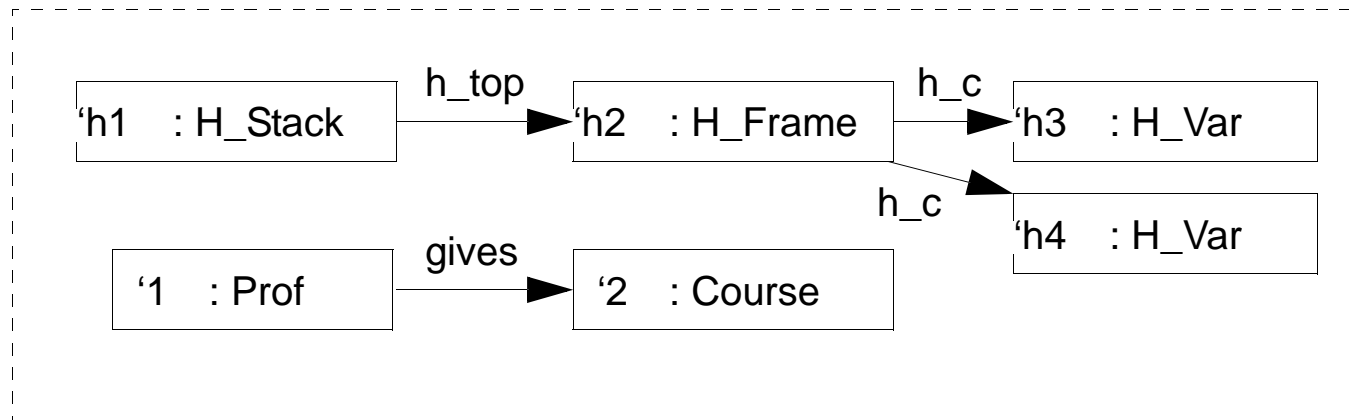
Allgemeine Behandlung von Out-Parametern:

- Wie In-Parameter: zusätzlicher Parameterknoten vor Aufruf erzeugen
- Zugriffe werden Attributzugriffe auf Parameterknoten
- Plus: nach Ausführung Ergebnis in Aktualparameter zurückkopieren (Call-By-Value-Return)

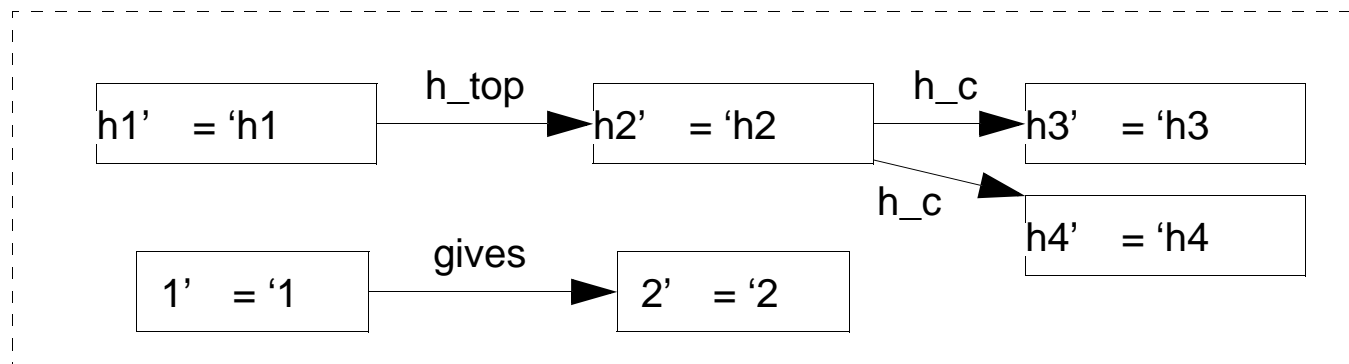
- Sem[use n : string := "" do . . . GetProfOfCourse("SWT", out n) . . . end]
:= Sem[use n: string := "" do
 . . .
 H_PushFrame
 & H_PushStringVar_title_SWT (* Behandlung des In-Parameters *)
 & H_PushStringVarFromVar("name", "n") (* Behandlung des Out-Parameters *)
 & H_GetProfOfCourse_title_name
 & H_ReturnStringOutParam("name", "n") (* Uebergeben des Rückgabewertes *)
 & H_PopFrame
 . . .
end]

mit:

production H_GetProfOfCourse_title_name =



::=



condition '2.Title = 'h3.H_StringAttr;

'h3.H_VarName = "title";

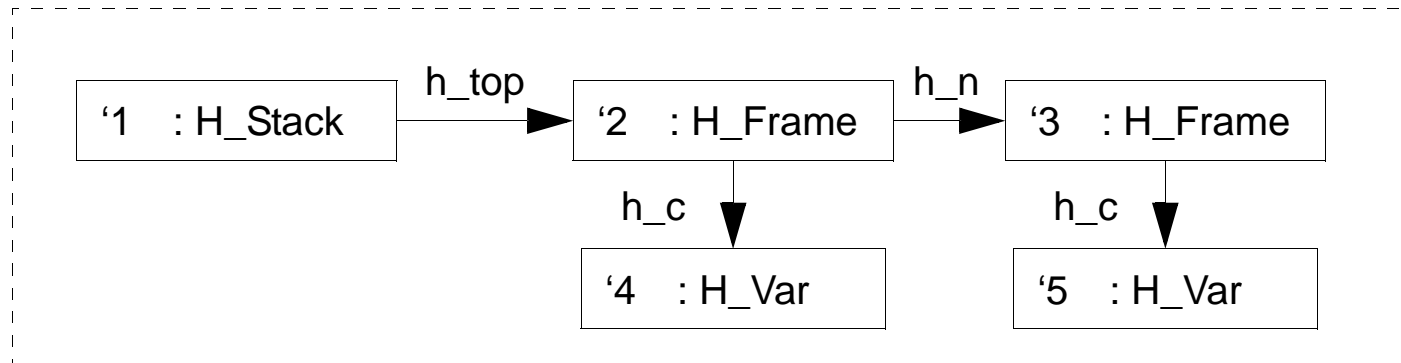
'h4.H_VarName = "name";

transfer h4'.H_StringAttr := '1.Name;

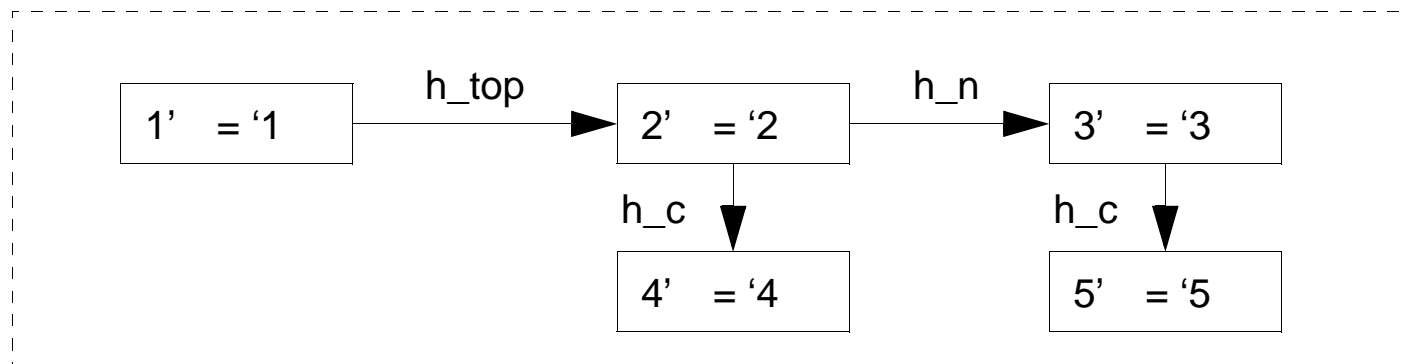
end;

production H_ReturnStringOutParam(src_var, dest_var : string)

=



::=



condition '4.H_VarName = src_var;

'5.H_VarName = dest_var;

transfer '5'.H_StringAttr := '4.H_StringAttr;

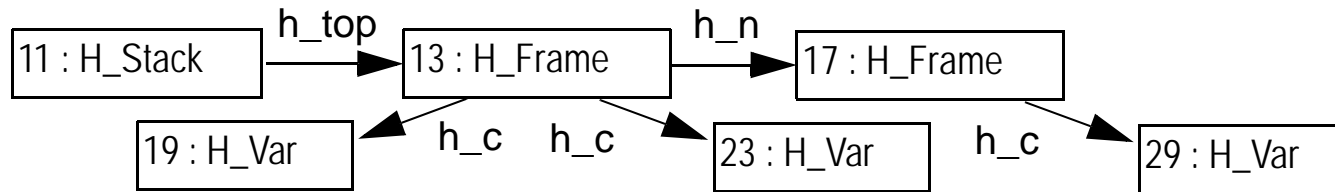
end;

Bemerkungen:

- Hier Call-By-Value-Return (erlaubt uniforme Behandlung von In- und Out-Parametern)
- Progres erlaubt beliebig viele out-Parameter (unterschiedlicher Typen)
-

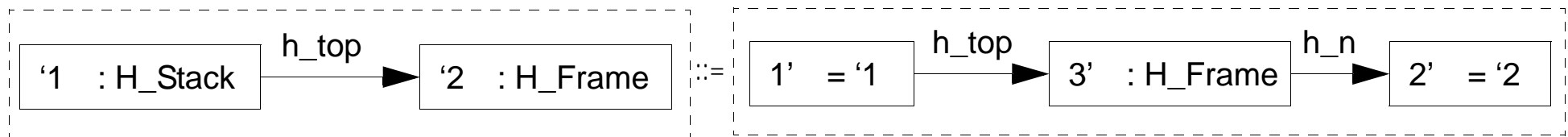
Übungsaufgabe 1:

Graph:



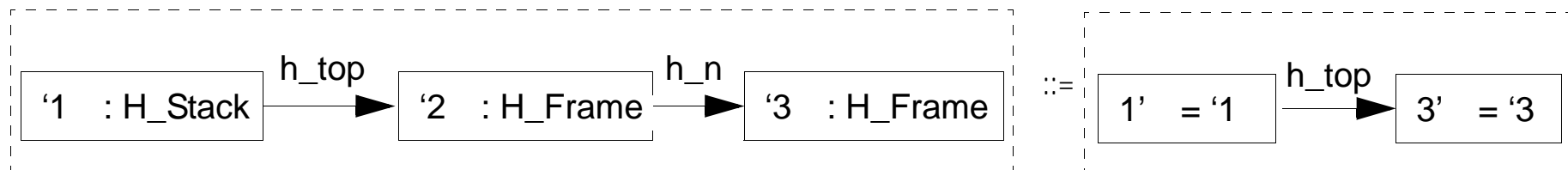
Regeln:

production H_PushFrame =



end;

production H_PopFrame =



end;

Beschreibt formal obigen GAKK-Graphen und obige Regeln, die Anwendung der Regeln und den jeweiligen Intermediate- und Ergebnisgraphen.

Übungsaufgabe 2:

Wir wollen einen Editor und einen Interpreter für (die logischen Datenstrukturen von) Progres mit Fujaba bauen. Überlegt euch Story-Boards für das editieren einer Graphersetzungsregel, d.h. für das

- Anlegen von Knoten- und Kantentypen
- Anlegen von Attributdeklarationen
- Einfügen von Knoten und Kanten in die linke und rechte Regelseite
- Erstellen von Attributbedingungen und -Zuweisungen

Übungsaufgabe 3:

Erstellt Story-Boards für die Ausführung einer Graphersetzungsregel:

- wie soll der Arbeitsgraph modelliert werden
- wie soll die Anwendungsstelle gefunden werden (das machen wir zusammen)
- wie funktioniert die Ausführung

Übungsaufgabe 4:

Erstellt für die Story Boards aus Aufgabe 2 und Aufgabe 3 ein gemeinsames Klassendiagramm mit Hilfe der Fujaba-Umgebung und generiert und kompiliert dafür Java Klassen. Programmiert die Regelausführung aus Aufgabe 3 als Story Diagramme und testet sie mit Fujaba