Logik – die beliebtesten Klausurfehler und Punktefresser

Inhalt

Allgemein: das Zusammenfassen von Schritten

Was beim privaten manuellen Duchrechnen eines Algorithmus ganz vernünftig sein mag – das **korrekte Zusammenfassen mehrerer Schritte** – bringt in der Klausur oft **Punktverluste**, insbesondere wenn und wo die Schritte ausdrücklich einzeln gewünscht werden.

Meist gibt es Punkte für jeden Schritt, aber **weniger als deren Summe** bei zusammengefassten Schritten. Wenn zu komplexe Gesamtschrittergebnisse vom Himmel fallen, sieht es auch leicht danach aus, als könnten diese aus der Klausur einer Nachbarin oder eines Nachbarn stammen.

Wenn bei mehreren Schritten die Zusammenfassung dann auch noch **misslingt**, verlieren Sie erst recht mehr Punkte als nötig, denn ein falscher von drei zusammengefassten Schritten kostet dann leicht die Punkte für alle drei Schritte.

Beispiel 1

Berechnen Sie den Wahrheitwerteverlauf mittels Wahrheitstafel:

A	B		¬(A→B)∧¬(B→A)
W	W		
W	F		
F	W		
F	F		

nicht elegant (oder rasant) so:

A	B	¬(A→B)	¬(B→A)	¬(A→B)∧¬(B→A)
W	W	F	F	F
W	F	W	F	F
F	W	F	W	F
F	F	F	F	F

sondern so:

A	B	A→B	¬(A→B)	B→A	¬(B→A)	¬(A→B)∧¬(B→A)
W	W	W	F	W	F	F
W	F	F	W	W	F	F
F	W	W	F	F	W	F
F	F	W	F	W	F	F

Beispiel 2

a) Berechnen Sie eine äquivalente KNF-Mengen-Formel für ¬((A→B)∧ ¬B) per syntaktischer Umformung (mit jeweils nur einem Schritt des Algorithmus aus der Vorlesung an jeweils nur einer Stelle).
b) Wie kann die Formel noch äquivalent abgekürzt werden?

nicht elegant so:

(a) ¬((A→B) ∧ ¬B)
¬ (A→B) ∧ B
(A∧¬B) ∧ B
{ {A}, {¬B}, {B} }
(b) { {¬B}, {B} }, oder noch besser: { { } }

sondern ausführlich so:

(a) ¬((A→B) ∧ ¬B)
¬ (A→B) ∧ ¬¬B
¬ (A→B) ∧ B
¬ (¬A∨B) ∧ B
(¬¬A ∧¬ B) ∧ B
(A∧¬B) ∧ B
{ {A}, {¬B}, {B} }
(b) { {¬B}, {B} }, oder noch besser: { { } }

Das gute und das böse Distributivgesetz bei syntaktischer Umformung

Bei syntaktischer Umformung nach KNF

ist

$A \vee (B \wedge C) \rightarrow (A \vee B) \wedge (A \vee C)$

sinnvoll. Es rückt $\vee$ näher an die Literale.

Im Beispiel erzeugt es sogar gerade KNF!

$A \wedge (B \vee C) \rightarrow (A \wedge B) \vee (A \wedge C)$

ist dann dagegen **schädlich**.

Im Beispiel macht es gerade die KNF kaputt!

Bei syntaktischer Umformung nach DNF

ist

$A \wedge (B \vee C) \rightarrow (A \wedge B) \vee (A \wedge C)$

sinnvoll. Es rückt $\wedge$ näher an die Literale.

Im Beispiel erzeugt es sogar gerade DNF!

$A \vee (B \wedge C) \rightarrow (A \vee B) \wedge (A \vee C)$

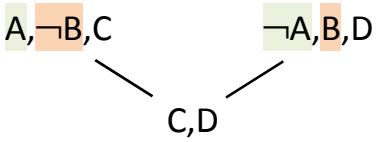
ist dann dagegen **schädlich**.

Im Beispiel macht es gerade die DNF kaputt!

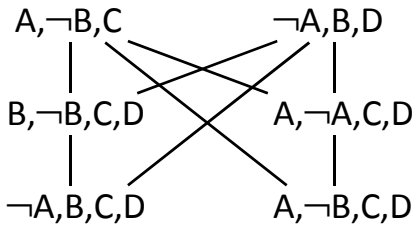
Illegale (also eigentlich gar keine) Resolutionsschritte

Beispiel 1: Gleichzeitiges Verkoppeln über mehr als ein P/¬P-Paar

Ein „Resolutionsschritt“ wie ...



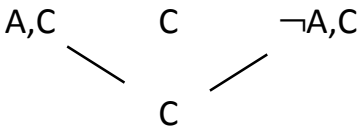
ist verführerisch aber **falsch**. Verschmolzen wird immer **nur über ein** P/¬P-Paar!
Alles was in dieser Situation per Resolution erzeugbar ist, ist dies (prüfen!):



Die Klausel **{C,D}** ist **nicht dabei**! Sie **folgt nicht** aus (A∨¬B∨C)∧(¬A∨B∨D).

Beispiel 2: Neues Erzeugen eines bereits vorhandenen Knotens

Das hier ...



ist auch **falsch**! Unser Resolutions-Algorithmus erzeugt jeweils nur **neue** Klauseln und keine vorhanden erneut.

Verwendung von gemischten und-oder-Ketten: Die gibt's gar nicht!

In etlichen Klausuren sieht man (stets **falsche**) **gemischte Ketten** wie
(?) $(A \vee \neg B \wedge C \vee \neg D)$,
oft auch mit **falschen Äquivalenzen** wie
(??) $\neg (A \vee \neg B \wedge C \vee \neg D) \equiv (\neg A \wedge B \vee \neg C \wedge D)$ ...
Meistens wurden irgendwann vorher schlicht **Klammern vergessen** ...

Wir verwenden höchstens **reine UND-Ketten** oder **reine ODER-Ketten**! Bei denen darf man auch das Antidistributivgesetz für negierte Ketten verwenden und erhält so (nach Streichung von Doppelnegationen) z.B.

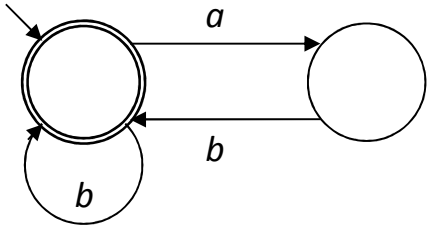
$$\neg (A \vee \neg B \vee C \vee \neg D) \equiv (\neg A \wedge B \wedge \neg C \wedge D) .$$

Für die Leser von Logikbüchern:
Wenn zwischen den logischen Junktoren **Prioritäten** verabredet sind, können "scheinbar (!) gemischte Ketten" erlaubt sein und Klammern eingespart werden. Die falsche Äquivalenz (??) bleibt aber dabei falsch! Außerdem verwenden verschiedene Autoren leider **verschiedene** Prioritäten, so dass dieselbe Formel von Buch zu Buch unterschiedliche Bedeutung haben kann. Deswegen wird in der Vorlesung ganz auf Klammerersparnis durch Prioritäten verzichtet.

Wenn alle Wörter einer Sprache L1 auch zur Sprache L2 gehören, muss noch lange nicht L1=L2 gelten

Zum Beispiel beim Vergleich der ω-Sprachen ...

L1: {*a*,*b*}-Folgen, die akzeptiert werden vom Büchi-Automat



L2: Modelle über {*a*,*b*} für **GF**b

Ist nicht L1 = L2,
wo doch jede Folge in L1 unendlich viele b's enthält,
also auch zu L2 gehört?

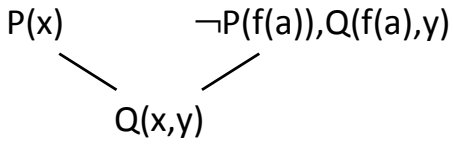
Antwort: L1 ≠ L2 (**obwohl** jede Folge in L1 unendlich viele b's enthält, also auch zu L2 gehört)

Grund: Alle Wörter in L1 haben noch die zusätzliche Eigenschaft, dass jeweils nach jedem *a* sofort ein *b* folgen muss.

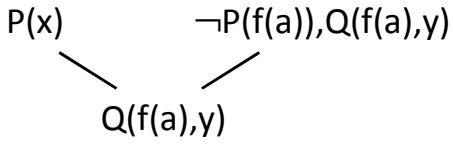
Das macht L1 zu einer **echten** Teilmenge von L2, denn z.B. die Folge $a\,a\,b^{\omega}$ ist Element von L2, aber nicht von L1.

Substitution in PL1 für irgendetwas außer Objektvariablen gibt es nicht

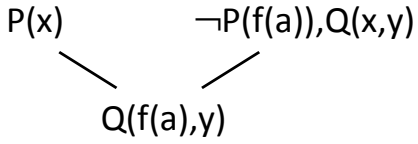
Substituiert wird **nur** für **Objektvariablen** (oft x,y,z).!
Manchmal sieht man **falsche** Unifikationsschritte wie



Dabei wurde fälschlich f(a) durch x ersetzt.
Man darf aber nur z.B. die **Variable** x durch f(a) substituieren, nicht umgekehrt, also richtig:



oder auch



Trennung von Konstanten-, Variablen- und Funktionsnamen in der Prädikatenlogik

Wir verwenden in PL1 Buchstaben ...

- vom Anfang des Alphabets (**a,b,c,...**) für **Objekt**konstanten,
- aus der „Mitte" des Alphabets (**f,g,h,...**) für **Funktions**namen,
- vom Ende des Alphabets (**... x,y,z**) für **Objekt**variablen.

Wenn Sie z.B. ∃a ... schreiben, kann man sich schon denken, dass Sie a als Objektvariable verwenden. Die Gefahr besteht aber, dass Sie das **selbst** später **vergessen**, a als Objektkonstante betrachten und dann evtl. falsche Schlüsse ziehen.