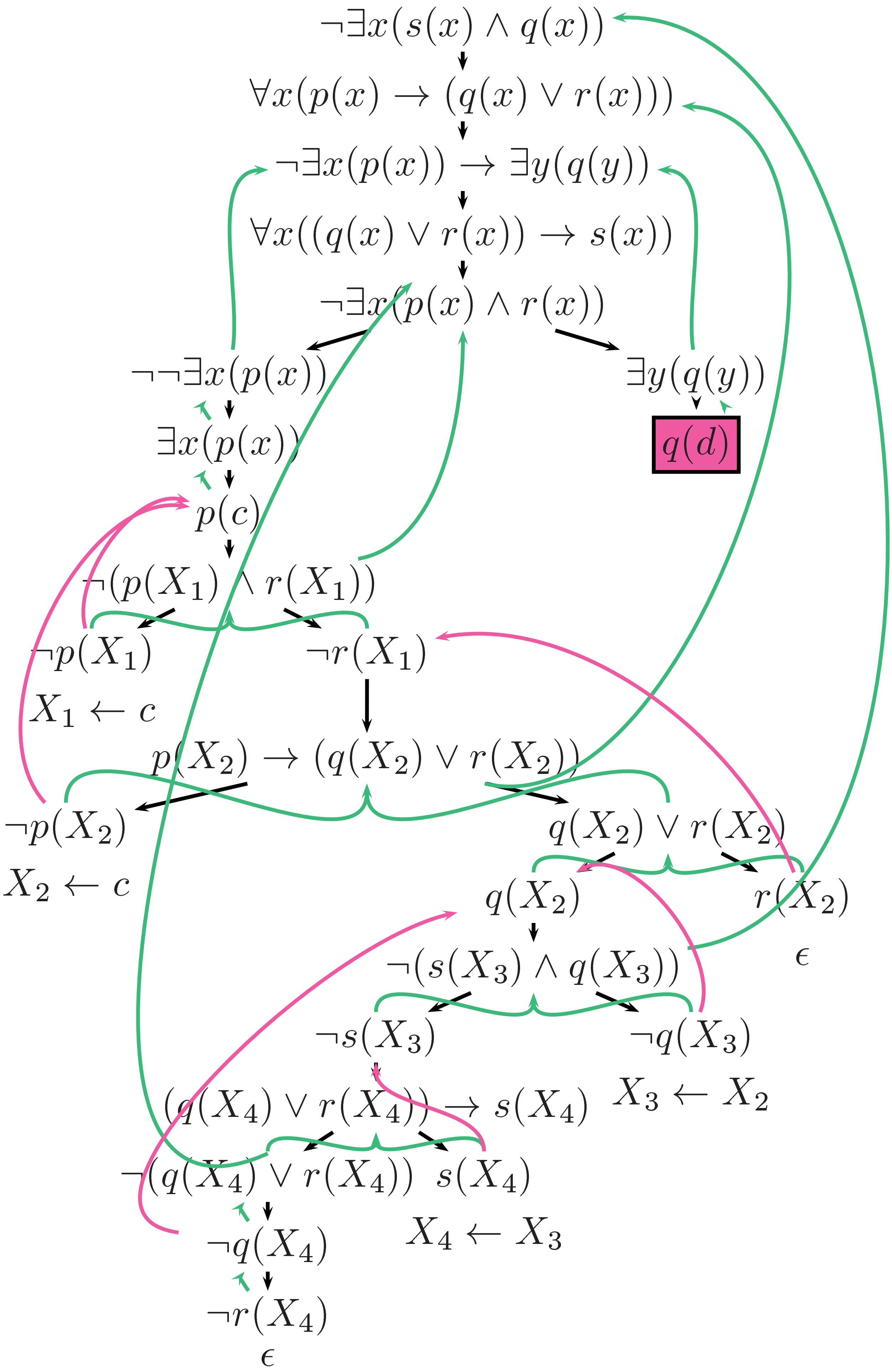


$\neg\exists x(s(x) \land q(x))$
$\forall x(p(x) \to (q(x) \lor r(x)))$
$\neg\exists x(p(x)) \to \exists y(q(y))$
$\forall x((q(x) \lor r(x)) \to s(x))$
$\neg\exists x(p(x) \land r(x))$
$\neg\neg\exists x(p(x))$
$\exists y(q(y))$
$\exists x(p(x))$
$q(d)$
$p(c)$
$\neg(p(X_1) \land r(X_1))$
$\neg p(X_1)$
$\neg r(X_1)$
$X_1 \leftarrow c$
$p(X_2) \to (q(X_2) \lor r(X_2))$
$\neg p(X_2)$
$q(X_2) \lor r(X_2)$
$X_2 \leftarrow c$
$q(X_2)$
$r(X_2)$
ϵ
$\neg(s(X_3) \land q(X_3))$
$\neg s(X_3)$
$\neg q(X_3)$
$(q(X_4) \lor r(X_4)) \to s(X_4)$
$X_3 \leftarrow X_2$
$\neg(q(X_4) \lor r(X_4))$
$s(X_4)$
$\neg q(X_4)$
$X_4 \leftarrow X_3$
$\neg r(X_4)$
ϵ

Vom Kalkül zur Beweissuchprozedur

Vollständigkeit der Tableauregeln garantiert nur **Existenz** eines geschlossenen Tableaus.
Eine systematische Beweissuchprozedur ist notwendig, um einen Beweis zu finden.

Tableaukonstruktion ist nicht-deterministisch

Choice points:

1. Nächster zu betrachtender Ast?
2. Abschluß- oder Erweiterung?
3. Falls Abschluß: Mit welchem Literal-Paar?
4. Falls Erweiterung: Mit welcher Formel?

Schlechte Wahl kann das Finden eines Beweises verhindern!

Drei Arten von Choice points

- Harmlose
- Kritische – mit Fairneßstrategie handhabbar
- Kritische – (z.Zt.) nicht mit Fairneß handhabbar

Eine abstrakte Tableauprozedur

```
/* declare */

type branch = list of formula;
type tableau = list of branch;

var H: {extend,close};
var P,Q: formula;
var B,C: branch;
var T,U: tableau;
var S: substitution;

function read_next_formula(): formula;
function append(L,M: list):list;
function delete(X: elem; L: list):list;
function rest(L: list):list;
function first(L: list):elem;
function mgu(P,Q: formula): substitution;
function apply_subst(S: substitution;
                     T: tableau): tableau;
function apply_rule(P: formula): tableau;

/* initialize */

B := [];
while ¬eof  do
  B := append(B,[read_next_formula]);
T := [B];
```

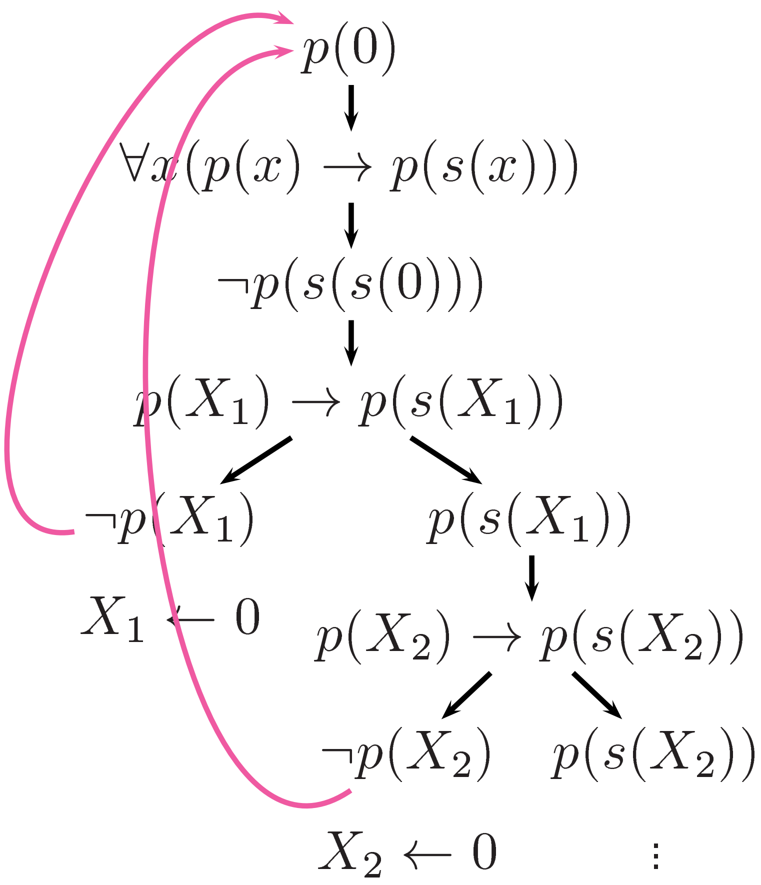
```
/* prove */

while T≠[] do begin
  B := select_branch(T);
  H := select_mode(B);
  T := delete(B,T);
  if H=close then begin
    [P,Q] := select_pair(B);
    S := mgu(P,Q);
    T := apply_subst(S,T)
    end
  else begin
    P := select_formula(B);
    if type(P) ∈ {alpha,beta,delta} then
      delete(P,B);
    U := apply_rule(P)
    while U≠[] do begin
      C := append(B,first(U));
      U := rest(U);
      T := append(T,[C])
    end
  end
end
writeln('Proof.')
```

Prozedur nicht für alle Auswahlfunktionen vollständig!

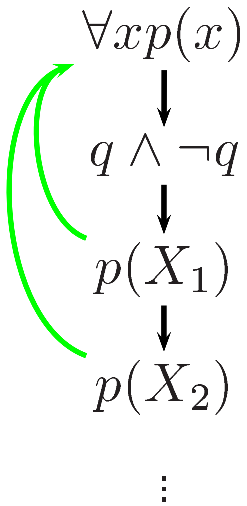
```
function select_branch(T: tableau): branch;
begin
  return(first(T))
end;

function
  select_mode(B: branch): {close,extend};
begin
  if ∃σ:Pσ,Qσ komplementäres Paar then
    return(close)
  elseif B enthält Nichtliteral
    return(extend)
  else halt
end;

function
  select_formula(B: branch): formula;
begin
  P:= das erste Nichtliteral P∈B;
  return(P)
end;

function
  select_pair(B:branch): list of formula;
begin
  [P,Q] := die ersten P,Q∈B so daß
               ∃σ:Pσ,Qσ komplementäres Paar;
  return([P,Q])
end;
```

Führt zu Unvollständigkeit!

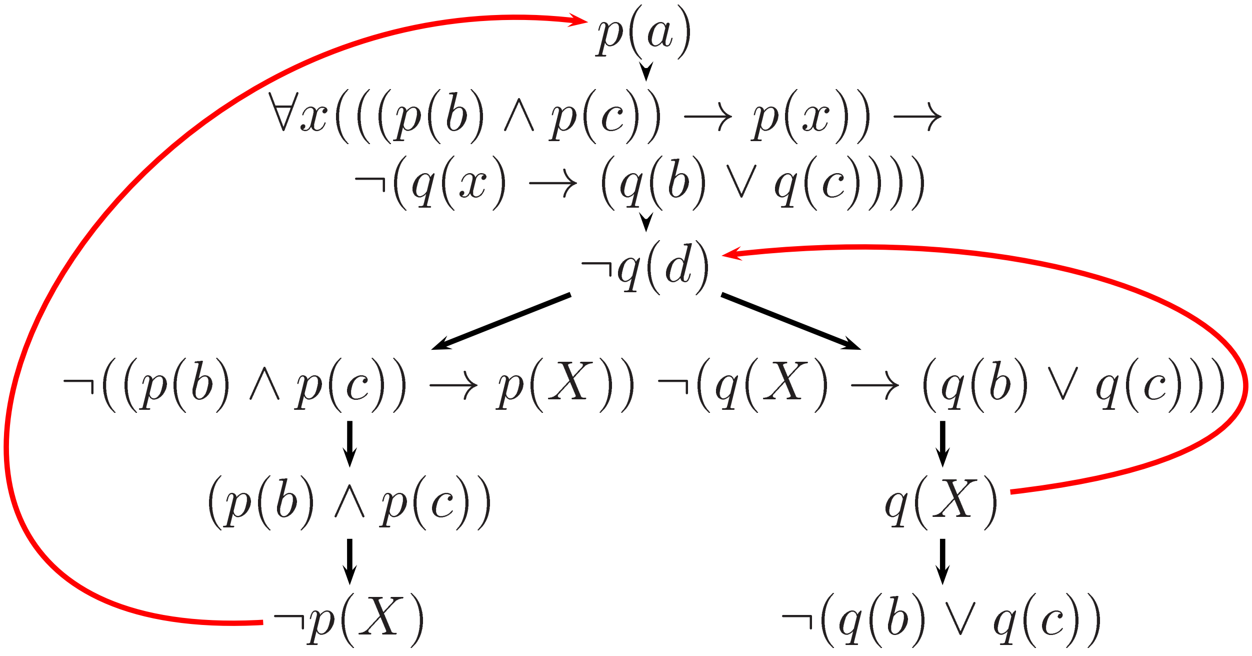
Bevorzugung einer Abschlußformel—`select_pair`



Bevorzugung einer Formel—`select_formula`



Bevorzugung eines Modus—`select_mode`

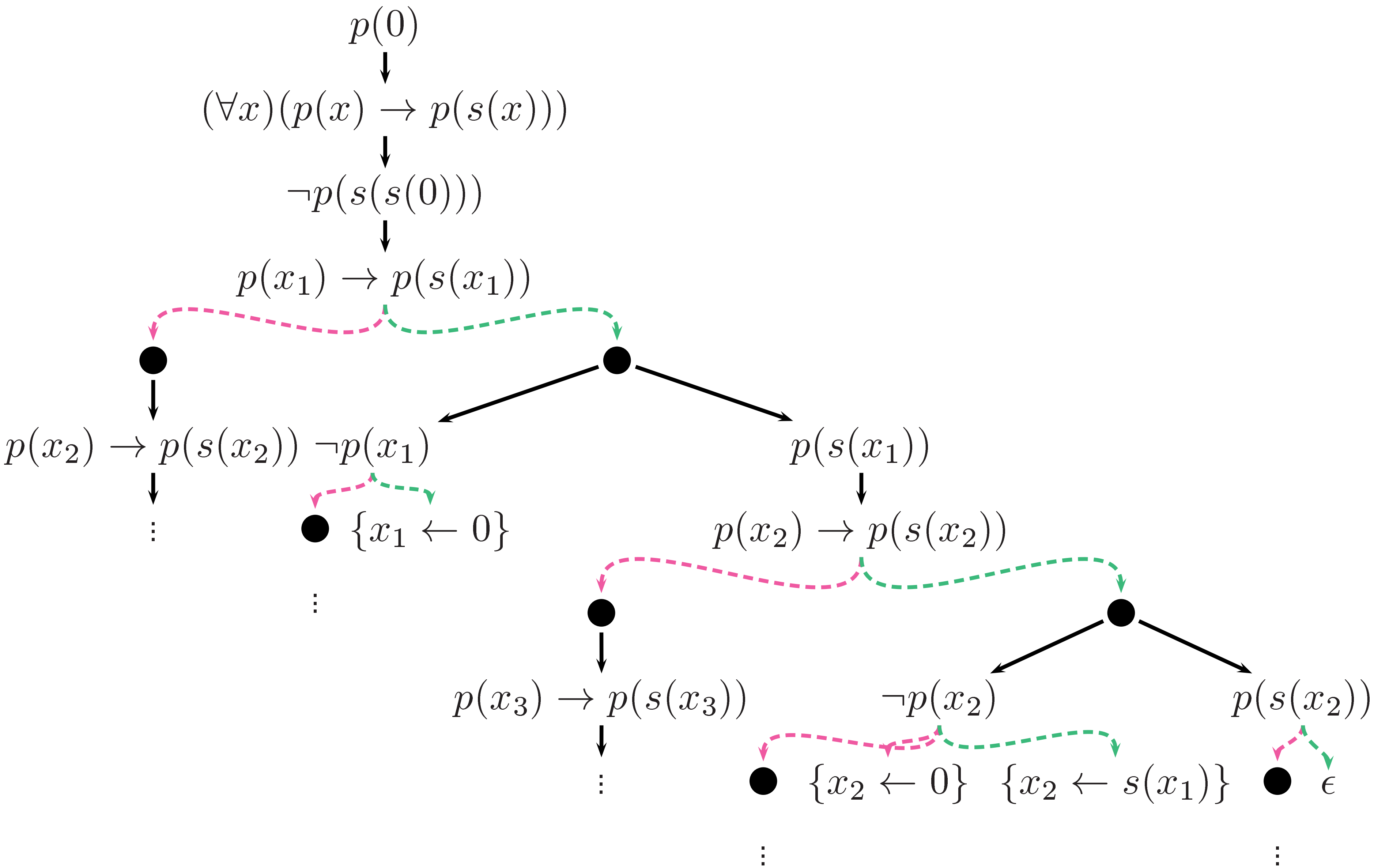


`select_branch` erzeugt kein Bevorzugungsproblem: Ohnehin müssen alle Äste geschlossen werden.

Erste Lösung des Bevorzugungsproblems:

Breitensuche.
`select_formula`, `select_mode`, `select_pair` stellen kritische Alternativen dar.

Repräsentiere die Suche nach Tableaubeweis als UND-ODER-Suchbaum, wobei die möglichen Werte der `select_...` Funktionen die ODER-Knoten definieren.


$p(0)$
$(\forall x)(p(x) \to p(s(x)))$
$\neg p(s(s(0)))$
$p(x_1) \to p(s(x_1))$
$p(x_2) \to p(s(x_2))$
$\neg p(x_1)$
$p(s(x_1))$
$\{x_1 \leftarrow 0\}$
$p(x_2) \to p(s(x_2))$
$p(x_3) \to p(s(x_3))$
$\neg p(x_2)$
$p(s(x_2))$
$\{x_2 \leftarrow 0\}$
$\{x_2 \leftarrow s(x_1)\}$
ϵ

Zweite Lösung des Bevorzugungsproblems:

– Ersetze Anwendungen der Abschlußregel durch eine einzige Anwendung der Substitutionsregel (ganz am Schluß), und

– implementiere `select_formula` **fair**.

```
function select_mode(T: tableau):
                    [{close,extend},substitution];
begin
  if ∃ Substitution σ so daß Tσ geschlossen then
    return([close,σ])
  elseif T enthält Nichtliteral
    return([extend,ε])
  else
    halt
end;

function select_pair(B:branch,
              σ:substitution): list of formula;
begin
  [P,Q] := die ersten P,Q ∈ B so daß
              Pσ,Qσ komplementäres Paar;
  return([P,Q])
end;
```

Fairneß

`select_formula` heißt **fair**, falls für alle $\Phi \subset For_{\Sigma^*}$ jede Folge von Tableaus $(T_i)_{i \geq 0}$ für Φ folgende Eigenschaften hat:

1. Auf alle Nichtliterale in allen Ästen aller T_i, wird in einem T_{i+k} eine passende Regel angewandt

2. Für jedes $k \geq 0$ gibt es ein i_k, so daß auf alle γ-Formeln in allen Ästen von T_{i_k} mindestens k-mal eine γ-Regel angewandt wird.

Nachteile:

- Gesamtes Tableau muß gespeichert werden, da für Abschlußtest benötigt.
- Testen auf Abschluß teuer.
- Effiziente Implementierung aufwendig.
- Das gesamte Tableau u.U. dennoch zu groß.
- Redundanz.

Vorteil: kein Backtracking, nur ein Tableau.

Beweiskonfluenz

Definition. Ein Kalkül heißt **beweiskonfluent**, falls jeder angefangene Beweis für eine unerfüllbare Formelmenge vervollständigt werden kann. ■

Satz. Freie-Variablen-Tableaus sind beweiskonfluent, falls

– Anwendungen der Abschlußregel in eine einzige Anwendung der Substitutionsregel am Schluß verschoben werden und

– die Implementierung der Funktion `select_formula` *fair* ist.

■

Beweiskonfluente nicht-destruktive Kalküle erfordern keine Breitensuche und kein Backtracking.

Daher verträglich mit sogenannten *Redundanz*-Strategien.

Noch besser wäre:

Fairneß von `select_pair`.

Dann:

– Abschlußregel kann verwendet werden,
– keine Speicherung geschlossener Äste nötig.

Aber:
Bisher nur theoretische Ergebnisse.

Dritte Lösung des Bevorzugungsproblems:

Implizite Beweisaufzählung durch Backtracking.

Wann immer `select_ ...` indeterministisch erzeuge *Choicepoint*.

m sei **Vollständigkeitsmodus**,
d.h. monotone Abbildung m von $\mathbb{N}$ auf endliche Mengen von Tableaus für eine Formelmenge Φ,
so daß $\bigcup_{i \in \mathbb{N}} m(i)$ alle Tableaus für Φ umfaßt.

Beispiele:

- Tableaus mit höchstens i Knoten.
- Tableaus, in denen γ-Regel höchstens i-mal auf eine Formel im Tableau/in jedem Ast angewandt wird.
- Tableaus mit Ästen höchstens der Länge i.
- Tableaus, in denen höchstens Terme der Länge bzw. Tiefe i vorkommen.
- Kombinierte Maße.

$M(i)$ sei Teil des UND-ODER Suchbaums, der Tableaus in $m(i)$ aber nicht solche in $m(j)$, $j < i$ umfaßt.

Breitensuche rät z.B. i_0 und durchsucht $M(i_0)$.

Tiefenbeschränkte Breitensuche als Tiefensuche mit Backtracking (über ODER-Knoten) implementiert (dynamische Programmierung).

$|m(i)|$ wächst exponentiell—exaktes Maß wichtig.

Inkrementelle Tiefensuche, depth-first iterative deepening search [Stickel, 1992]: durchsuche $\bigcup_{j<i} M(i)$ für $i = 0, 1, \ldots$.

Nur polynomiell teurer als Suche in „richtigem“ $M(i_0)$ [Korf, 1985].

- Genau analysieren, welche Choicepoints für Vollständigkeit benötigt.

- Reihenfolge, in der Suchbaum durchlaufen wird, hat großen Einfluß auf tatsächlichen Suchraum.

Sprachkonstrukte für dynamische Programmierung (Backtracking):

```
function or(L: list of X):X;
begin
  if L=[] then
    fail
  else
    lege choicepoint an mit Code-Fortsetzung
      L := rest(L); return(or(L));
    return(first(L))
end;

procedure fail;
begin
  if es gibt noch choicepoints then
    gehe zum zuletzt angelegten choicepoint
  else
    halt
end;
```

Zur Beschränkung inkrementeller Tiefensuche an geeigneter Stelle *fail* erzeugen.

`select_ ...` mit Choicepoints

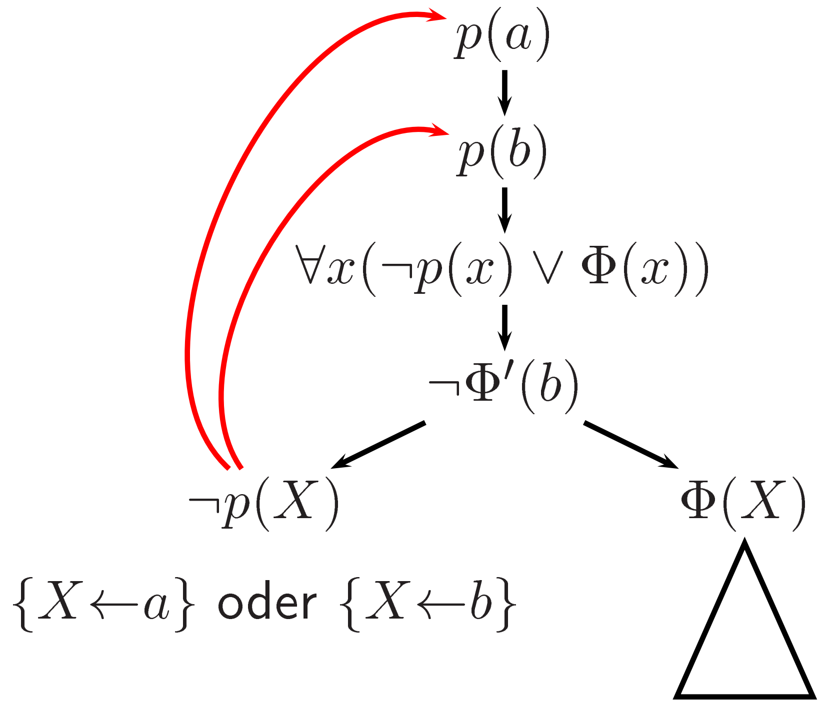
```
function
  select_mode(B: branch):{close,extend};
begin
  if ∃σ:Pσ,Qσ komplementäres Paar then
    return(or([close,extend]))
  elseif B enthält Nichtliteral
    return(extend)
  else
    fail
end;

function
  select_formula(B: branch): formula;
begin
  L := Liste aller Nichtliterale P∈B;
  return(or(L))
end;

function
  select_pair(B:branch):list of formula;
begin
  L := Liste aller Paare P,Q∈B so daß
          ∃σ:Pσ,Qσ komplementäres Paar;
  return(or(L))
end;
```

Verzicht of Choice points in `select_branch`,
immer, da Reihenfolge des Astabschlusses keinen Einfluß hat.

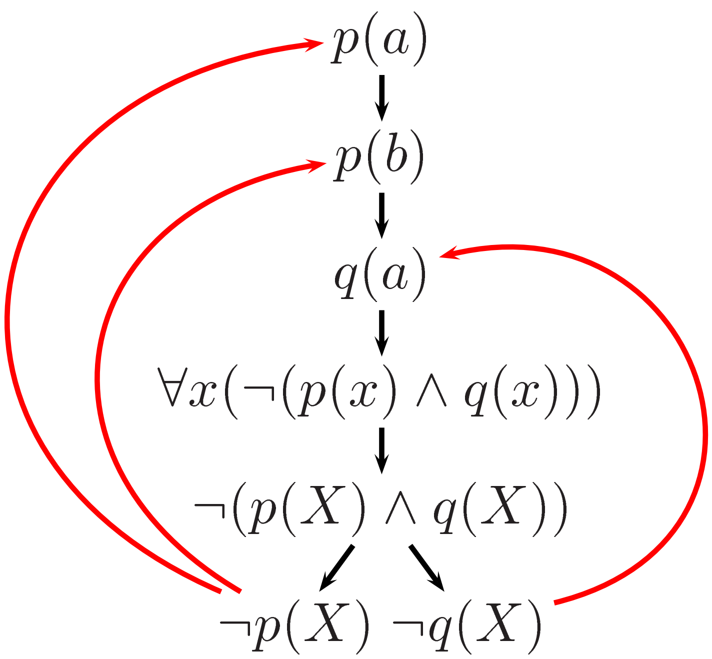
Verzicht auf Choice points in `select_formula`
möglich bei Fairneß.

Nachteil: potentiell längere Beweise (Redundanz)
Vorteil: weniger Choice points

Beispiel für Sensibilität des tatsächlichen Suchraums auf Reihenfolge von L in `select_pair`.



Interaktion von `select_branch` mit `select_pair`:



Übliche Auswahlregeln für `select_branch`

Least recent: der „älteste“ noch offene Ast

Left/right-first: „linkester/rechtester“ offener Ast

Depth-first: offener Ast maximaler Länge

Depth-last: offener Ast mit minimaler Länge

Depth-left-first: „linkester“ offener Ast maximaler Länge