Monotone bedingte Verzweigungen in Logikprogrammen

Ulrich Neumerkel, TU Wien — Stefan Kral, FH Wiener Neustadt

1,

Monotone bedingte Verzweigungen in Logikprogrammen

Ulrich Neumerkel, TU Wien — Stefan Kral, FH Wiener Neustadt

Langfristiges Ziel: Stärkung von Prologs puren und monotonen Elementen

+ constraintverträglich
+ verbesserte Analysierbarkeit (z.B. Termination)
+ verbesserte Erklärungen (*slices* statt *traces*)

[1, 2,]

Monotone bedingte Verzweigungen in Logikprogrammen

Ulrich Neumerkel, TU Wien — Stefan Kral, FH Wiener Neustadt

Langfristiges Ziel: Stärkung von Prologs puren und monotonen Elementen

+ constraintverträglich
+ verbesserte Analysierbarkeit (z.B. Termination)
+ verbesserte Erklärungen (*slices* statt *traces*)
+ ISO/IEC 13211 Normarbeit in JTC 1 SC 22 WG 17 über DIN & ASI

1, 2, 3,

Monotone bedingte Verzweigungen in Logikprogrammen

Ulrich Neumerkel, TU Wien — Stefan Kral, FH Wiener Neustadt

Langfristiges Ziel: Stärkung von Prologs puren und monotonen Elementen

+ constraintverträglich
+ verbesserte Analysierbarkeit (z.B. Termination)
+ verbesserte Erklärungen (*slices* statt *traces*)
+ ISO/IEC 13211 Normarbeit in JTC 1 SC 22 WG 17 über DIN & ASI
+ Richtung Purheit — ähnlich FP

1, 2, 3, 4,

Monotone bedingte Verzweigungen in Logikprogrammen

Ulrich Neumerkel, TU Wien — Stefan Kral, FH Wiener Neustadt

Langfristiges Ziel: Stärkung von Prologs puren und monotonen Elementen

+ constraintverträglich
+ verbesserte Analysierbarkeit (z.B. Termination)
+ verbesserte Erklärungen (*slices* statt *traces*)
+ ISO/IEC 13211 Normarbeit in JTC 1 SC 22 WG 17 über DIN & ASI
+ Richtung Purheit — ähnlich FP

Was bisher geschah:

... seiteneffektfreie Ein-/Ausgabe, Lambda Audrücke

Higher Order `call/N` über Cor.2:2012 ISO/IEC 13211-1

1, 2, 3, 4, 5,

Monotone bedingte Verzweigungen in Logikprogrammen

Ulrich Neumerkel, TU Wien — Stefan Kral, FH Wiener Neustadt

Langfristiges Ziel: Stärkung von Prologs puren und monotonen Elementen

+ constraintverträglich
+ verbesserte Analysierbarkeit (z.B. Termination)
+ verbesserte Erklärungen (*slices* statt *traces*)
+ ISO/IEC 13211 Normarbeit in JTC 1 SC 22 WG 17 über DIN & ASI
+ Richtung Purheit — ähnlich FP

Was bisher geschah:

... seiteneffektfreie Ein-/Ausgabe, Lambda Audrücke

Higher Order `call/N` über Cor.2:2012 ISO/IEC 13211-1

Jetzt: ein monotones if-then-else Konstrukt

1, 2, 3, 4, 5, 6

If-then-else in Prolog

```
... ( If_0 -> Then_0 ; Else_0 )
```

- `If_0` muss hinreichend instanziert sein
- Nichtmonoton

```
?- ( X = Y -> A = gleich ; A = ungleich ).
```

If-then-else in Prolog

```
... ( If_0 -> Then_0 ; Else_0 )
```

- `If_0` muss hinreichend instanziert sein, sonst unvollständig
- Nichtmonoton

```
?- ( X = Y -> A = gleich ; A = ungleich ).
   X = Y, A = gleich.
% es fehlt:
%  dif(X, Y),  A = ungleich.
```

```
member(X, [E|Es]) :-
    (  X = E
    ;
       member(X, Es)
    ).
```

member/2

```
member(X, [E|Es]) :-
    (  X = E
    ;
       member(X, Es)
    ).
```

- Redundante Antworten, besucht gesamte Liste

```
?- member(1, [1,2,1]).
   true
;  true.
```

- Nichtdeterminismus erfordert Konstrukte wie cut, `once/1`

```
memberchk(X, Xs) :-
    once(member(X, Xs)).
```

member/2 **und** dif/2

```
member(X, [E|Es]) :-
   (  X = E
   ;
      member(X, Es)
   ).
```

```
memberd(X, [E|Es]) :-
   (  X = E
   ;  dif(X, E),
      memberd(X, Es)
   ).
```

member/2 und dif/2

```
member(X, [E|Es]) :-
   (  X = E
   ;
      member(X, Es)
   ).
```

```
?- member(1, [1,X]).
   true
;  X = 1. % redundante Antwort
```

```
memberd(X, [E|Es]) :-
   (  X = E
   ;  dif(X, E),
      memberd(X, Es)
   ).
```

```
?- memberd(1, [1,X]).
   true
;  false. % nichtdet. Scheitern
```

`dif/2` direkt verwendet führt zu umständlichen und ineffizienten Programmen

member/2 und dif/2

```
member(X, [E|Es]) :-
   (  X = E
   ;
      member(X, Es)
   ).
```

```
?- member(1, [1,X]).
   true
;  X = 1. % redundante Antwort
```

```
memberd(X, [E|Es]) :-
   (  X = E
   ;  dif(X, E),
      memberd(X, Es)
   ).
```

```
?- memberd(1, [1,X]).
   true
;  false. % nichtdet. Scheitern
```

`dif/2` direkt verwendet führt zu umständlichen und ineffizienten Programmen
Lösung:

```
memberd(X, [E|Es]) :-
   if_(X = E, true, memberd(X, Es)).

?- memberd(1, [1,X]).
   true.
```

```
if_(If_1, Then_0, Else_0)
```

```
?- if_( X = Y, A = gleich, A = ungleich ).
   X = Y, A = gleich
;  dif(X, Y),  A = ungleich.
```

if_/3

```
if_(If_1, Then_0, Else_0)
```

```
?- if_( X = Y, A = gleich, A = ungleich ).
   X = Y, A = gleich
;  dif(X, Y),  A = ungleich.
```

`If_1` ... partielles Ziel, reifiziert.
`(=)/3` — reifizierte Gleichheit

```
=(X, X, true).
=(X, Y, false) :-
   dif(X,Y).
```

```
if_(If_1, Then_0, Else_0)
```

```
?- if_( X = Y, A = gleich, A = ungleich ).
   X = Y, A = gleich
;  dif(X, Y),  A = ungleich.
```

`If_1` ... partielles Ziel, reifiziert.
`(=)/3` — reifizierte Gleichheit

```
=(X, X, true).
=(X, Y, false) :-
   dif(X,Y).
```

```
=(X, Y, T) :-
   (  X == Y -> T = true
   ;  X \= Y -> T = false
   ;  T = true, X = Y
   ;  T = false,
      dif(X, Y)
   ).
```

Allgemeine Reifikation

```
memberd_t(X, Es, true) :-
   memberd(X, Es).
memberd_t(X, Es, false) :-
   maplist(dif(X), Es).
```

Allgemeine Reifikation

```
memberd_t(X, Es, true) :-
   memberd(X, Es).
memberd_t(X, Es, false) :-
   maplist(dif(X), Es).
```

```
memberd_t(X, Es, T) :-
   l_memberd_t(Es, X, T).

l_memberd_t([], _, false).
l_memberd_t([E|Es], X, T) :-
   if_( X = E
      , T = true
      , l_memberd_t(Es, X, T) ).
```

if_/3

```
if_(If_1, Then_0, Else_0)
```

- pur
- monoton
- effizient (Determinismus)
- ISO-konform

if_/3

```
if_(If_1, Then_0, Else_0)
```

- pur
- monoton
- effizient (Determinismus)
- ISO-konform

Es fehlt noch:

- effizientere Systemunterstützung von `(=)/3`
- Expandierung von `if_/3`