Computational Logic

Logic Programming

Georg Moser

Department of Computer Science @ UIBK

Winter 2016

Summary of Last Lecture

Example

```
no_doubles([],[]).
no_doubles([X|Xs],Ys) :-
    member(X,Xs), !,                cut
    no_doubles(Xs,Ys).
no_doubles([X|Xs],[X|Ys]) :-
    no_doubles(Xs,Ys).
```

Effect of Cut

- ! succeeds
- ! fixes all choices between (and including) moment of matching rule's head with parent goal and cut

  if backtracking reaches !, the cut fails and the search continues from the last choice made before the clause containing ! was chosen

Outline of the Lecture

Monotone Logic Programs

introduction, basic constructs, logic foundations, unification, semantics, database and recursive programming, termination, complexity

Incomplete Data Structures and Constraints

incomplete data structures, definite clause grammars, constraint logic programming, answer set programming

Full Prolog

semantics (revisited), cuts, correctness proofs, meta-logical predicates, cuts nondeterministic programming, efficient programs, complexity

Outline of the Lecture

Monotone Logic Programs

introduction, basic constructs, logic foundations, unification, semantics, database and recursive programming, termination, complexity

Incomplete Data Structures and Constraints

incomplete data structures, definite clause grammars, constraint logic programming, answer set programming

Full Prolog

semantics (revisited), cuts, correctness proofs, meta-logical predicates, cuts nondeterministic programming, efficient programs, complexity

Green vs Red Cuts

Definition

- a cut is green if the addition of the cut doesn't change the meaning of the program; removing it makes the program potentially inefficient, but not wrong

Green vs Red Cuts

Definition

- a cut is green if the addition of the cut doesn't change the meaning of the program; removing it makes the program potentially inefficient, but not wrong
- a cut is red if its presence changes the meaning of the program; removing it, changes the meaning and thus may make the program wrong

Green vs Red Cuts

Definition

- a cut is green if the addition of the cut doesn't change the meaning of the program; removing it makes the program potentially inefficient, but not wrong
- a cut is red if its presence changes the meaning of the program; removing it, changes the meaning and thus may make the program wrong

Example (. . .)

```
delete([X|Ys],X,Zs) :- !, delete(Ys,X,Zs).
delete([Y|Ys],X,[Y|Zs]) :- Y ≠ X, !, delete(Ys,X,Zs).
delete([],X,[]).
```

Green vs Red Cuts

Definition

- a cut is green if the addition of the cut doesn't change the meaning of the program; removing it makes the program potentially inefficient, but not wrong
- a cut is red if its presence changes the meaning of the program; removing it, changes the meaning and thus may make the program wrong

Example (Green Cut)

```
delete([X|Ys],X,Zs) :- !, delete(Ys,X,Zs).
delete([Y|Ys],X,[Y|Zs]) :- Y ≠ X, !, delete(Ys,X,Zs).
delete([],X,[]).
```

Example (...)

```
delete([X|Xs],X,Zs) :- !, delete(Ys,X,Zs).
delete([Y|Ys],X,[Y|Zs]) :- !, delete(Ys,X,Zs).
delete([],X,[]).

:- \+ delete([a,b],b,[a,b]).
```

Example (Red Cut)

```
delete([X|Xs],X,Zs) :- !, delete(Ys,X,Zs).
delete([Y|Ys],X,[Y|Zs]) :- !, delete(Ys,X,Zs).
delete([],X,[]).

:- \+ delete([a,b],b,[a,b]).
```

Example (Red Cut)

```
delete([X|Xs],X,Zs) :- !, delete(Ys,X,Zs).
delete([Y|Ys],X,[Y|Zs]) :- !, delete(Ys,X,Zs).
delete([],X,[]).

:- \+ delete([a,b],b,[a,b]).
```

Example (...)

```
member(X,[X|Xs]) :- !.
member(X,[Y|Ys]) :- member(X,Ys).
```

Example (Red Cut)

```
delete([X|Xs],X,Zs) :- !, delete(Ys,X,Zs).
delete([Y|Ys],X,[Y|Zs]) :- !, delete(Ys,X,Zs).
delete([],X,[]).

:- \+ delete([a,b],b,[a,b]).
```

Example (Red Cut)

```
member(X,[X|Xs]) :- !.
member(X,[Y|Ys]) :- member(X,Ys).
```

Example (Red Cut)

```
delete([X|Xs],X,Zs) :- !, delete(Ys,X,Zs).
delete([Y|Ys],X,[Y|Zs]) :- !, delete(Ys,X,Zs).
delete([],X,[]).
```

```
:- \+ delete([a,b],b,[a,b]).
```

Example (Red Cut)

```
member(X,[X|Xs]) :- !.
member(X,[Y|Ys]) :- member(X,Ys).
```

Example (. . .)

```
minimum(X,Y,X) :- X ⩽ Y, !.
minimum(X,Y,Y).
```

```
:- minimum(2,5,X)
```

$X = 2$

Example (Red Cut)

```
delete([X|Xs],X,Zs) :- !, delete(Ys,X,Zs).
delete([Y|Ys],X,[Y|Zs]) :- !, delete(Ys,X,Zs).
delete([],X,[]).
```

```
:- \+ delete([a,b],b,[a,b]).
```

Example (Red Cut)

```
member(X,[X|Xs]) :- !.
member(X,[Y|Ys]) :- member(X,Ys).
```

Example (...)

```
minimum(X,Y,X) :- X ⩽ Y, .
minimum(X,Y,Y).
```

```
:- minimum(2,5,X)
```

$X = 2$
$X = 5$

Example (Red Cut)

```
delete([X|Xs],X,Zs) :- !, delete(Ys,X,Zs).
delete([Y|Ys],X,[Y|Zs]) :- !, delete(Ys,X,Zs).
delete([],X,[]).

:- \+ delete([a,b],b,[a,b]).
```

Example (Red Cut)

```
member(X,[X|Xs]) :- !.
member(X,[Y|Ys]) :- member(X,Ys).
```

Example (...)

```
minimum(X,Y,X) :- X ⩽ Y, !.
minimum(X,Y,Y).

:- minimum(2,5,5)

true
```

Example (Red Cut)

```
delete([X|Xs],X,Zs) :- !, delete(Ys,X,Zs).
delete([Y|Ys],X,[Y|Zs]) :- !, delete(Ys,X,Zs).
delete([],X,[]).
```

```
:- \+ delete([a,b],b,[a,b]).
```

Example (Red Cut)

```
member(X,[X|Xs]) :- !.
member(X,[Y|Ys]) :- member(X,Ys).
```

Example (Bad Cut)

```
minimum(X,Y,X) :- X ⩽ Y, !.
minimum(X,Y,Y).
```

```
:- minimum(2,5,5)
```

```
true
```

Cuts: Why (Not) Use Them

Fact

the following motivations are given in the literature for the use of cuts

Cuts: Why (Not) Use Them

Fact

the following motivations are given in the literature for the use of cuts

1. *non-overlapping pattern matches*

Cuts: Why (Not) Use Them

Fact

the following motivations are given in the literature for the use of cuts

1. *non-overlapping pattern matches*
2. *increased efficiency due to improved handling of negations*

Cuts: Why (Not) Use Them

Fact

the following motivations are given in the literature for the use of cuts

1. *non-overlapping pattern matches*
2. *increased efficiency due to improved handling of negations*
3. *efficient generate and test*

Cuts: Why (Not) Use Them

Fact

the following motivations are given in the literature for the use of cuts

1. *non-overlapping pattern matches*
2. *increased efficiency due to improved handling of negations*
3. *efficient generate and test*

Example

```
bubblesort(Xs,Ys) :-
    append(As,[X,Y|Bs],Xs),
    X > Y, !,
    append(As,[Y,X|Bs],Xs1),
    bubblesort(Xs1,Ys).
bubblesort(Xs,Xs) :-
    ordered(Xs), !.
```

Cuts: Why (Not) Use Them

Fact

the following motivations are given in the literature for the use of cuts

1. *non-overlapping pattern matches*
2. *increased efficiency due to improved handling of negations*
3. *efficient generate and test*

Example

```
bubblesort(Xs,Ys) :-
    append(As,[X,Y|Bs],Xs),
    X > Y, !,
    append(As,[Y,X|Bs],Xs1),
    bubblesort(Xs1,Ys).
bubblesort(Xs,Xs) :-
    ordered(Xs), !.
```

Fact

the following motivations are given in the literature for the use of cuts

1. *non-overlapping pattern matches*
2. *increased efficiency due to improved handling of negations*
3. *efficient generate and test*

Fact

the following motivations are given in the literature for the use of cuts

1. *non-overlapping pattern matches*
2. *increased efficiency due to improved handling of negations*
3. *efficient generate and test*

Example

```
no_doubles([],[]).
no_doubles([X|Xs],Ys) :-
    member(X,Xs), !,
    no_doubles(Xs,Ys).
no_doubles([X|Xs],[X|Ys]) :-
    no_doubles(Xs,Ys).
```

Fact

the following motivations are given in the literature for the use of cuts

1. *non-overlapping pattern matches*
2. *increased efficiency due to improved handling of negations*
3. *efficient generate and test*

Fact

the following motivations are given in the literature for the use of cuts

1. *non-overlapping pattern matches*
2. *increased efficiency due to improved handling of negations*
3. *efficient generate and test*

Example (integer division with cut)

```
is_integer(0).
is_integer(N) :- is_integer(N1), N is N1 + 1.

divide(N1,N2,Result) :-
    is_integer(Result),
    Product1 is Result * N2,
    Product2 is (Result+1)*N2,
    Product1 =< N1,
    Product2 > N1, !.
```

Reflections

Fact

only the 2nd motivation withstands scrutinised look and can often be prevented by the explicit use of negation:

Reflections

Fact

only the 2nd motivation withstands scrutinised look and can often be prevented by the explicit use of negation:

- *(arithmetic) comparisons are (very) cheap*

Reflections

Fact

only the 2nd motivation withstands scrutinised look and can often be prevented by the explicit use of negation:

- *(arithmetic) comparisons are (very) cheap*
- *generate and test is more efficiently solved by CLP*

Reflections

Fact

only the 2nd motivation withstands scrutinised look and can often be prevented by the explicit use of negation:

- *(arithmetic) comparisons are (very) cheap*
- *generate and test is more efficiently solved by CLP*

Example

```
no_doubles([],[]).
no_doubles([X|Xs],Ys) :-
    member(X,Xs),
    no_doubles(Xs,Ys).
no_doubles([X|Xs],[X|Ys]) :-
    \+ member(X,Xs),
    no_doubles(Xs,Ys).
```

Definition (Prolog Search Tree)

a Prolog search tree is a triple (T, N, U), where

- T is a tree whose nodes are labelled with sequences of goals and partial answer substitutions
- N is the current node of T, which is to be considered in the next step
- U is a set of yet unvisited nodes

Definition (Prolog Search Tree)

a Prolog search tree is a triple (T, N, U), where

- T is a tree whose nodes are labelled with sequences of goals and partial answer substitutions
- N is the current node of T, which is to be considered in the next step
- U is a set of yet unvisited nodes

Example

```
minus(X,0,X).
minus(s(X),s(Y),Z) :- minus(X,Y,Z).

div(X,0,Z,R) :- !, fail.
div(0,_Y,Z,R) :- !, Z=0, R=0.
div(X,Y,s(Z),R) :- minus(X,Y,U), !,
    div(U,Y,Z,R).
div(X,Y,0,X).
```

Definition

let P be a Prolog program and Q be a query; the search tree visit and construction algorithm A generates a search tree (T, N, U) as follows:

Definition

let P be a Prolog program and Q be a query; the search tree visit and construction algorithm A generates a search tree (T, N, U) as follows:

1. initially the root becomes current node N, labelled with Q and ϵ

Definition

let P be a Prolog program and Q be a query; the search tree visit and construction algorithm A generates a search tree (T, N, U) as follows:

1. initially the root becomes current node N, labelled with Q and ϵ
2. if the current sequence of goals Q is `true` backtrack to the first node in U (U is always updated by using a depth-first, leftmost strategy)

Definition

let P be a Prolog program and Q be a query; the search tree visit and construction algorithm A generates a search tree (T, N, U) as follows:

1. initially the root becomes current node N, labelled with Q and ϵ
2. if the current sequence of goals Q is `true` backtrack to the first node in U (U is always updated by using a depth-first, leftmost strategy)
3. otherwise, let T be the first goal in Q

Definition

let P be a Prolog program and Q be a query; the search tree visit and construction algorithm A generates a search tree (T, N, U) as follows:

1. initially the root becomes current node N, labelled with Q and ϵ
2. if the current sequence of goals Q is `true` backtrack to the first node in U (U is always updated by using a depth-first, leftmost strategy)
3. otherwise, let T be the first goal in Q
4. if $T =$ `true`, delete T and goto Step 2

Definition

let P be a Prolog program and Q be a query; the search tree visit and construction algorithm A generates a search tree (T, N, U) as follows:

1. initially the root becomes current node N, labelled with Q and ϵ
2. if the current sequence of goals Q is `true` backtrack to the first node in U (U is always updated by using a depth-first, leftmost strategy)
3. otherwise, let T be the first goal in Q
4. if $T = \texttt{true}$, delete T and goto Step 2
5. if T is user-defined, either expand the tree by n successor nodes, where n is the number of clauses $H_i : -B_i$ such that H_i unifies with T or backtrack; in the former case the successors are labelled by $Q \setminus \{T\} \cup B_i$, the leftmost child becomes the current node, update U

Definition

let P be a Prolog program and Q be a query; the search tree visit and construction algorithm A generates a search tree (T, N, U) as follows:

1. initially the root becomes current node N, labelled with Q and ϵ
2. if the current sequence of goals Q is `true` backtrack to the first node in U (U is always updated by using a depth-first, leftmost strategy)
3. otherwise, let T be the first goal in Q
4. if $T = \texttt{true}$, delete T and goto Step 2
5. if T is user-defined, either expand the tree by n successor nodes, where n is the number of clauses $H_i : -B_i$ such that H_i unifies with T or backtrack; in the former case the successors are labelled by $Q \setminus \{T\} \cup B_i$, the leftmost child becomes the current node, update U
6. if T is built-in, perform the specific side effects of the predicate and goto Step 2

Correctness and Completeness of Prolog

Definition

the intended meaning of a Prolog program is a set of ground facts G

Correctness and Completeness of Prolog

Definition

the intended meaning of a Prolog program is a set of ground facts G

Definition

a program P is called

- correct with respect to the intended meaning M, if the meaning of P is a subset of M
- complete if the intended meaning M is a subset of the meaning of P

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
```

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
```

Lemma

the program is complete wrt the set of facts

$$M := \{\texttt{natural\_number}(s^i(0)) \mid i \geqslant 0\}$$

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
```

Lemma

the program is complete wrt the set of facts

$$M := \{\texttt{natural\_number}(s^i(0)) \mid i \geqslant 0\}$$

Proof of Completeness.

1. let N be a natural number

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
```

Lemma

the program is complete wrt the set of facts

$$M := \{\texttt{natural\_number}(s^i(0)) \mid i \geqslant 0\}$$

Proof of Completeness.

1. let N be a natural number
2. we show that $\texttt{natural\_number}(s^N(0))$ is deducible by given a explicit search tree

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).
```

Lemma

the program is complete wrt the set of facts

$$M := \{\texttt{natural\_number}(s^i(0)) \mid i \geqslant 0\}$$

Proof of Completeness.

1. let N be a natural number
2. we show that $\texttt{natural\_number}(s^N(0))$ is deducible by given a explicit search tree
3. case distinction on $N = 0$ and $N > 0$

Lemma

the program is correct wrt the set of facts

$$M := \{\texttt{natural\_number}(s^i(0)) \mid i \geqslant 0\}$$

Lemma

the program is correct wrt the set of facts

$$M := \{\texttt{natural\_number}(s^i(0)) \mid i \geqslant 0\}$$

Proof of Correctness.

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions

Lemma

the program is correct wrt the set of facts

$$M := \{\texttt{natural\_number}(s^i(0)) \mid i \geqslant 0\}$$

Proof of Correctness.

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions
2. we use induction on n

Lemma

the program is correct wrt the set of facts

$$M := \{\texttt{natural\_number}(s^i(0)) \mid i \geqslant 0\}$$

Proof of Correctness.

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions
2. we use induction on n
3. $n = 1$: then $\texttt{natural\_number}(s^m(0))$ implies $m = 0$

Lemma

the program is correct wrt the set of facts

$$M := \{\texttt{natural\_number}(s^i(0)) \mid i \geqslant 0\}$$

Proof of Correctness.

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions
2. we use induction on n
3. $n = 1$: then $\texttt{natural\_number}(s^m(0))$ implies $m = 0$
4. $n > 0$: the goal must be of form $\texttt{natural\_number}(s(t))$

Lemma

the program is correct wrt the set of facts

$$M := \{\texttt{natural\_number}(s^i(0)) \mid i \geqslant 0\}$$

Proof of Correctness.

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions
2. we use induction on n
3. $n = 1$: then $\texttt{natural\_number}(s^m(0))$ implies $m = 0$
4. $n > 0$: the goal must be of form $\texttt{natural\_number}(s(t))$
5. thus $\texttt{natural\_number}(t)$ is deducible with $n - 1$ deductions

Lemma

the program is correct wrt the set of facts

$$M := \{\texttt{natural\_number}(s^i(0)) \mid i \geqslant 0\}$$

Proof of Correctness.

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions
2. we use induction on n
3. $n = 1$: then $\texttt{natural\_number}(s^m(0))$ implies $m = 0$
4. $n > 0$: the goal must be of form $\texttt{natural\_number}(s(t))$
5. thus $\texttt{natural\_number}(t)$ is deducible with $n - 1$ deductions
6. $t = s^{m'}(0)$ for some $m' \in \mathbb{N}$

Lemma

the program is correct wrt the set of facts

$$M := \{\texttt{natural\_number}(s^i(0)) \mid i \geqslant 0\}$$

Proof of Correctness.

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions
2. we use induction on n
3. $n = 1$: then $\texttt{natural\_number}(s^m(0))$ implies $m = 0$
4. $n > 0$: the goal must be of form $\texttt{natural\_number}(s(t))$
5. thus $\texttt{natural\_number}(t)$ is deducible with $n - 1$ deductions
6. $t = s^{m'}(0)$ for some $m' \in \mathbb{N}$
7. $\texttt{natural\_number}(s^{m'+1}(0)) \in M$ and $m = m' + 1$

Example

is the program is complete wrt the following set?

$$M := \{\texttt{natural\_number}(s^i(0)) \mid 0 \leqslant i \leqslant K\}$$

Example

is the program is correct wrt the following set?

$$M := \{\texttt{natural\_number}(s^i(0)) \mid 0 \leqslant i \leqslant K\}$$

Example

is the program is correct wrt the following set?

$$M := \{\texttt{natural\_number}(s^i(0)) \mid 0 \leqslant i \leqslant K\}$$

Attempted Proof

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions
2. we use induction on n

Example

is the program is correct wrt the following set?

$$M := \{\texttt{natural\_number}(s^i(0)) \mid 0 \leqslant i \leqslant K\}$$

Attempted Proof

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions
2. we use induction on n
3. $n = 0$: then $\texttt{natural\_number}(s^m(0))$ implies $m = 0$

Example

is the program is correct wrt the following set?

$$M := \{\texttt{natural\_number}(s^i(0)) \mid 0 \leqslant i \leqslant K\}$$

Attempted Proof

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions
2. we use induction on n
3. $n = 0$: then $\texttt{natural\_number}(s^m(0))$ implies $m = 0$
4. $n > 0$: the goal must be of form $\texttt{natural\_number}(s(t))$

Example

is the program is correct wrt the following set?

$$M := \{\texttt{natural\_number}(s^i(0)) \mid 0 \leqslant i \leqslant K\}$$

Attempted Proof

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions
2. we use induction on n
3. $n = 0$: then $\texttt{natural\_number}(s^m(0))$ implies $m = 0$
4. $n > 0$: the goal must be of form $\texttt{natural\_number}(s(t))$
5. thus $\texttt{natural\_number}(t)$ is deducible with $n - 1$ deductions

Example

is the program is correct wrt the following set?

$$M := \{\texttt{natural\_number}(s^i(0)) \mid 0 \leqslant i \leqslant K\}$$

Attempted Proof

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions
2. we use induction on n
3. $n = 0$: then $\texttt{natural\_number}(s^m(0))$ implies $m = 0$
4. $n > 0$: the goal must be of form $\texttt{natural\_number}(s(t))$
5. thus $\texttt{natural\_number}(t)$ is deducible with $n - 1$ deductions
6. $t = s^{m'}(0)$ for some $0 \leqslant m' \leqslant K$ and $m = m' + 1$

Example

is the program is correct wrt the following set?

$$M := \{\texttt{natural\_number}(s^i(0)) \mid 0 \leqslant i \leqslant K\}$$

Attempted Proof

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions
2. we use induction on n
3. $n = 0$: then $\texttt{natural\_number}(s^m(0))$ implies $m = 0$
4. $n > 0$: the goal must be of form $\texttt{natural\_number}(s(t))$
5. thus $\texttt{natural\_number}(t)$ is deducible with $n - 1$ deductions
6. $t = s^{m'}(0)$ for some $0 \leqslant m' \leqslant K$ and $m = m' + 1$
7. $\texttt{natural\_number}(s^m(0)) \in M$ iff $m \leqslant K$

Example

is the program is correct wrt the following set?

$$M := \{\texttt{natural\_number}(s^i(0)) \mid 0 \leqslant i \leqslant K\}$$

Attempted Proof

1. suppose $\texttt{natural\_number}(s^m(0))$ is deducible in n deductions
2. we use induction on n
3. $n = 0$: then $\texttt{natural\_number}(s^m(0))$ implies $m = 0$
4. $n > 0$: the goal must be of form $\texttt{natural\_number}(s(t))$
5. thus $\texttt{natural\_number}(t)$ is deducible with $n - 1$ deductions
6. $t = s^{m'}(0)$ for some $0 \leqslant m' \leqslant K$ and $m = m' + 1$
7. $\texttt{natural\_number}(s^m(0)) \in M$ iff $m \leqslant K$
8. what happens for $m > K$?

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).

plus(0,X,X) :- natural_number(X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
```

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).

plus(0,X,X) :- natural_number(X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
```

Lemma

the program is correct and complete wrt to the definition of addition

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).

plus(0,X,X) :- natural_number(X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
```

Lemma

the program is correct and complete wrt to the definition of addition

Proof Sketch.

1. completeness: suppose $X + Y = Z$; then we give a search tree of $\texttt{plus}(X, Y, Z)$

Example

```
natural_number(0).
natural_number(s(X)) :- natural_number(X).

plus(0,X,X) :- natural_number(X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
```

Lemma

the program is correct and complete wrt to the definition of addition

Proof Sketch.

1. completeness: suppose $X + Y = Z$; then we give a search tree of $\texttt{plus}(X, Y, Z)$
2. correctness: suppose $\texttt{plus}(X, Y, Z)$ is deducible; then we prove by induction on the length of this deduction that $X + Y = Z$