# Logik für Informatiker: PROLOG
# Part 2: Logic

Andreas Karwath

&

Wolfram Burgard

(original slides by Peter Flach)

- For
  - propositional clausal logic
  - relational clausal logic
  - full clausal logic
- we present:
  - the *syntax*: defining the logical language
  - the *semantics*: defining the meaning of words and sentences *truth-functional*
  - the *proof theory*: how new true sentences can be derived from known true sentences by rules of inference
  - the *meta-theory*: theory about logic; statements about completeness, correctness, decidability.

- ***Propositional* clausal logic**
  - expressions that can be true or false
- ***Relational* clausal logic**
  - constants and variables refer to objects
- ***Full* clausal logic**
  - functors aggregate objects
- ***Definite clause* logic = pure Prolog**
  - no disjunctive heads



# Clausal logic

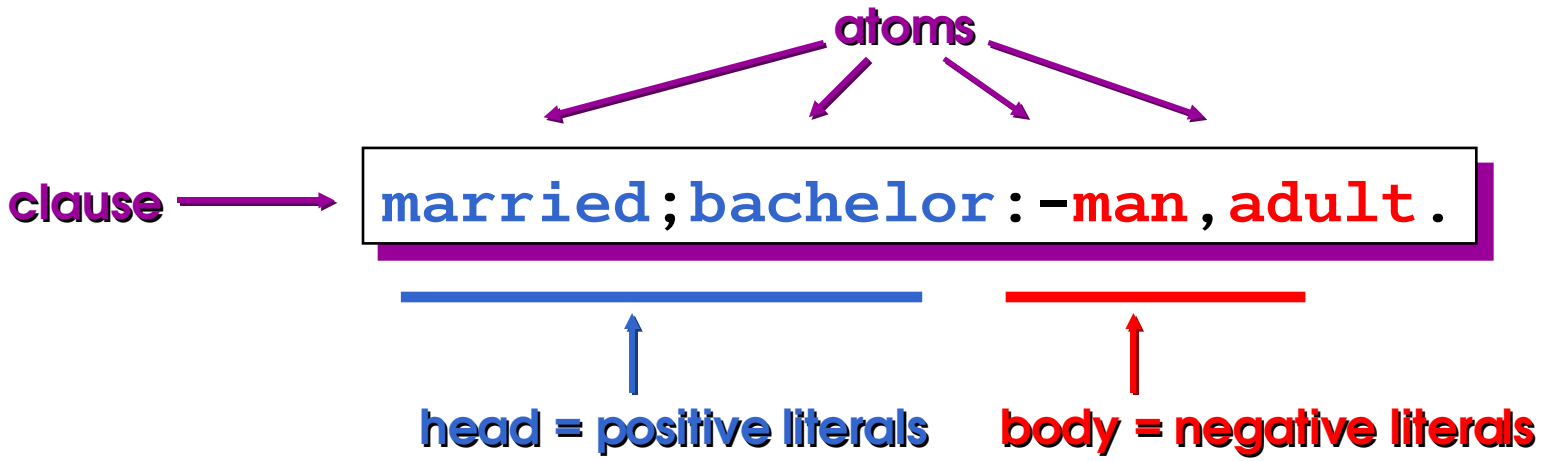
# Propositional Clausal Logic

- *Proposition*
- Statement that can be true or false.
- Propositions are represented by so-called *atoms*. E.g., `married` is an *atom* denoting the statement „he/she is married“
- Symbols used:
  :- if
  ; or
  , and
- Program: set of clauses, each finished with a „.“
  Read conjunctively
- ```
  woman; man :- human.
  human :- woman.
  human :- man.
  ```



# Propositional clausal logic: syntax (1)
  ```

"Somebody is married **or** a bachelor **if** he is a man **and** an adult."




$$married \vee bachelor \vee \neg man \vee \neg adult$$



# Propositional clausal logic: syntax (2)

- Persons are happy or sad

```
happy;sad:-person.
```

- No person is both happy and sad

```
:-person,happy,sad.
```

- Sad persons are not happy

```
:-person,sad,happy.
```

- Non-happy persons are sad

```
sad;happy:-person.
```



# Exercise 2.1

- ***Herbrand base***: set of atoms
  - `{married,bachelor,man,adult}`
- ***Herbrand interpretation***: set of **true** atoms
  - `{married,man,adult}`
- A clause is **false** in an interpretation if all body-literals are **true** and all head-literals are **false**…
  - `bachelor:-man,adult.`
- …and **true** otherwise: the interpretation is a ***model*** of the clause.
  - `:-married,bachelor.`



# Propositional clausal logic: semantics

- Herbrand base: set of atoms occurring in program *P*
- Herbrand interpretation of *P*: mapping of Herbrand base to `{true, false}`
- E.g., `{woman -> true, man -> false, human -> true}`
- Shorthand notation: only subset of Herbrand base that is true: `{woman, human}`
- Herbrand interpretation describes a state of affairs in the universe of discourse.

- Truth value of clause:
  (i) true if body true and head true
  (ii) false if body true and head false
  (iii) true if body false and head true
  (iv) true if body false and head false

- (iii) and (iv): every clause should have a truth value under any interpretation

- Clause „if *body* then *head*“ equivalent to „*head* or not *body*“.

- If a clause is true under an interpretation, then this interpretation is said to be a *model* of the clause.

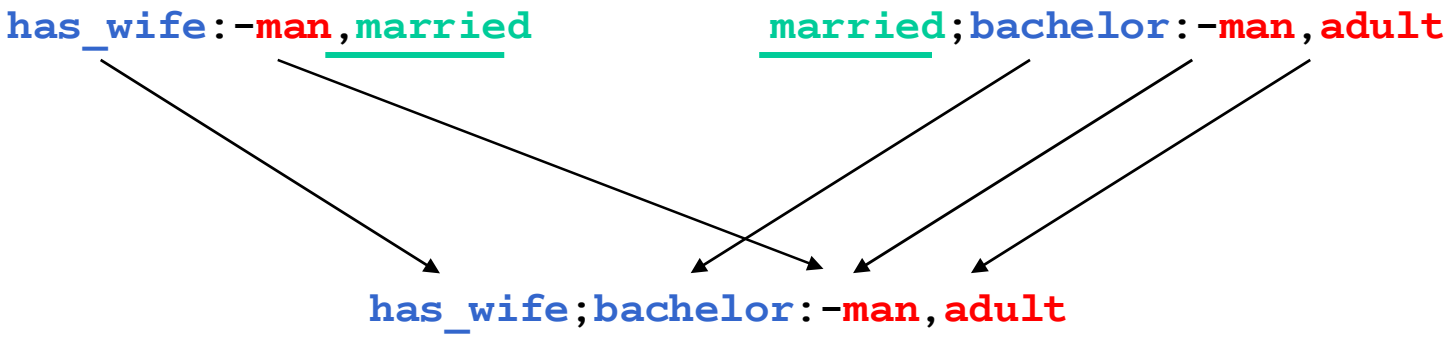


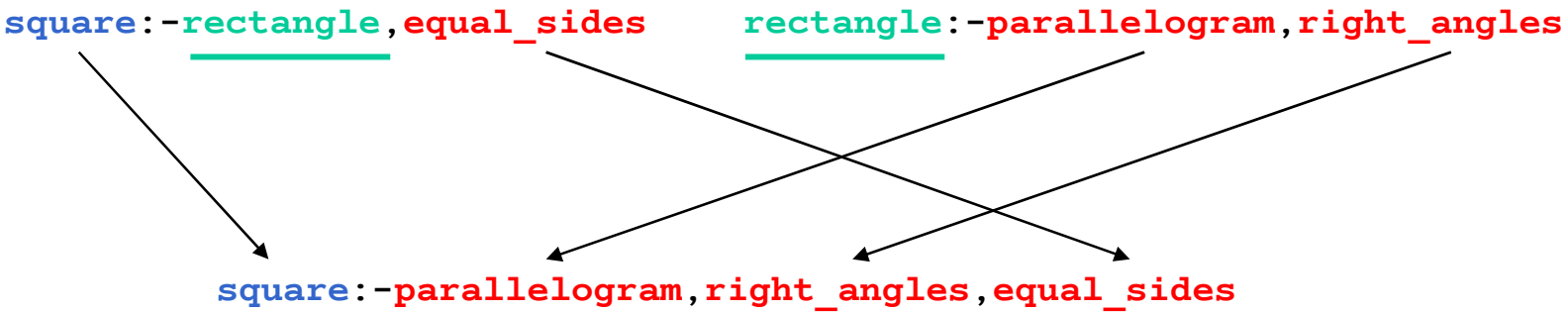
# Propositional clausal logic: semantics (2)

- A clause ***C*** is a ***logical consequence*** of a program (set of clauses) ***P*** iff every model of ***P*** is a model of ***C***.
- Let ***P*** be
  - `married;bachelor:-man,adult.`
  - `man.`
  - `:-bachelor.`
- **`married:-adult`** is a logical consequence of ***P***;
- **`married:-bachelor`** is a logical consequence of ***P***;
- **`bachelor:-man`** is not a logical consequence of ***P***;
- **`bachelor:-bachelor`** is a logical consequence of ***P***.



# Exercise 2.2






# Propositional resolution

- Propositional resolution is
  - ***sound***: it derives only logical consequences.
  - ***incomplete***: it cannot derive arbitrary tautologies like `a:-a`…
  - …but ***refutation-complete***: it derives the empty clause from any inconsistent set of clauses.
- ***Proof by refutation***: add the negation of the assumed logical consequence to the program, and prove inconsistency by deriving the empty clause.

- Derive `friendly` from the following the program:

```
happy; friendly :- teacher.

friendly :- teacher, happy.

teacher; wise.

teacher :- wise.
```

- Find a proof by refutation that

```
friendly :- has_friends.
```

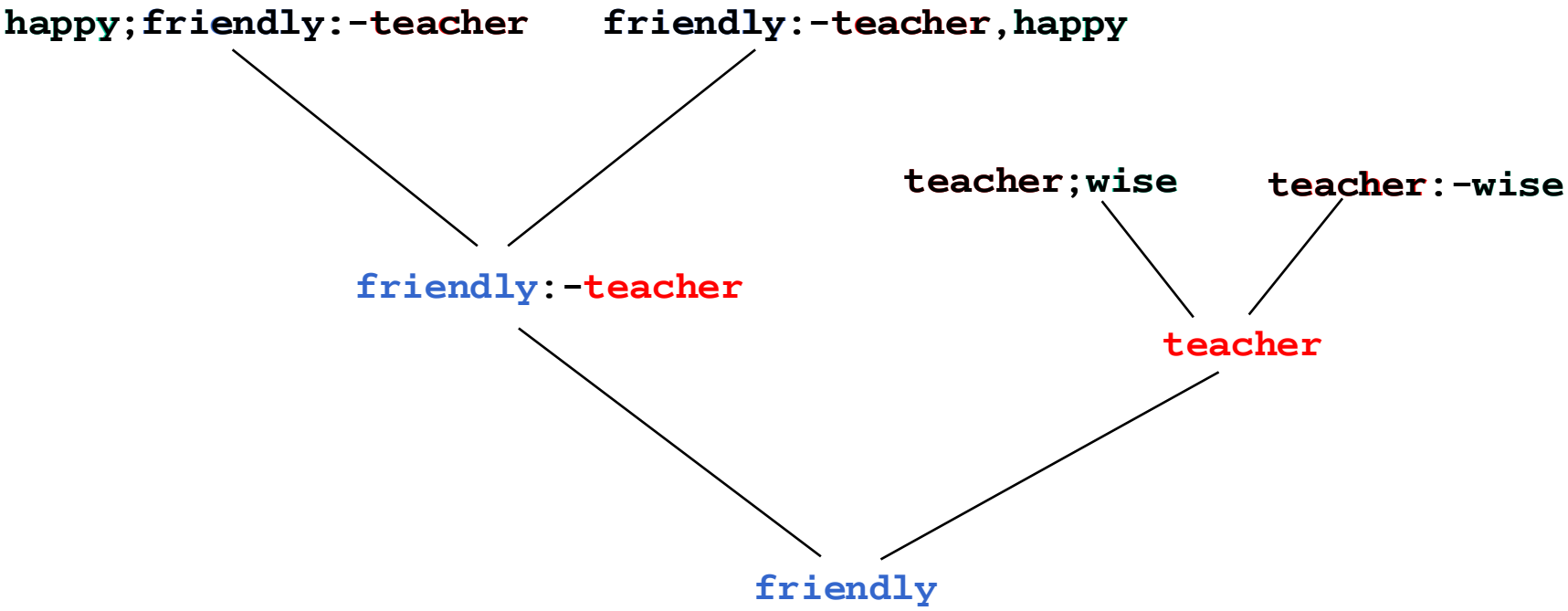
is a logical consequence of the following clauses:

```
happy :- has_friends.

friendly :- happy.
```



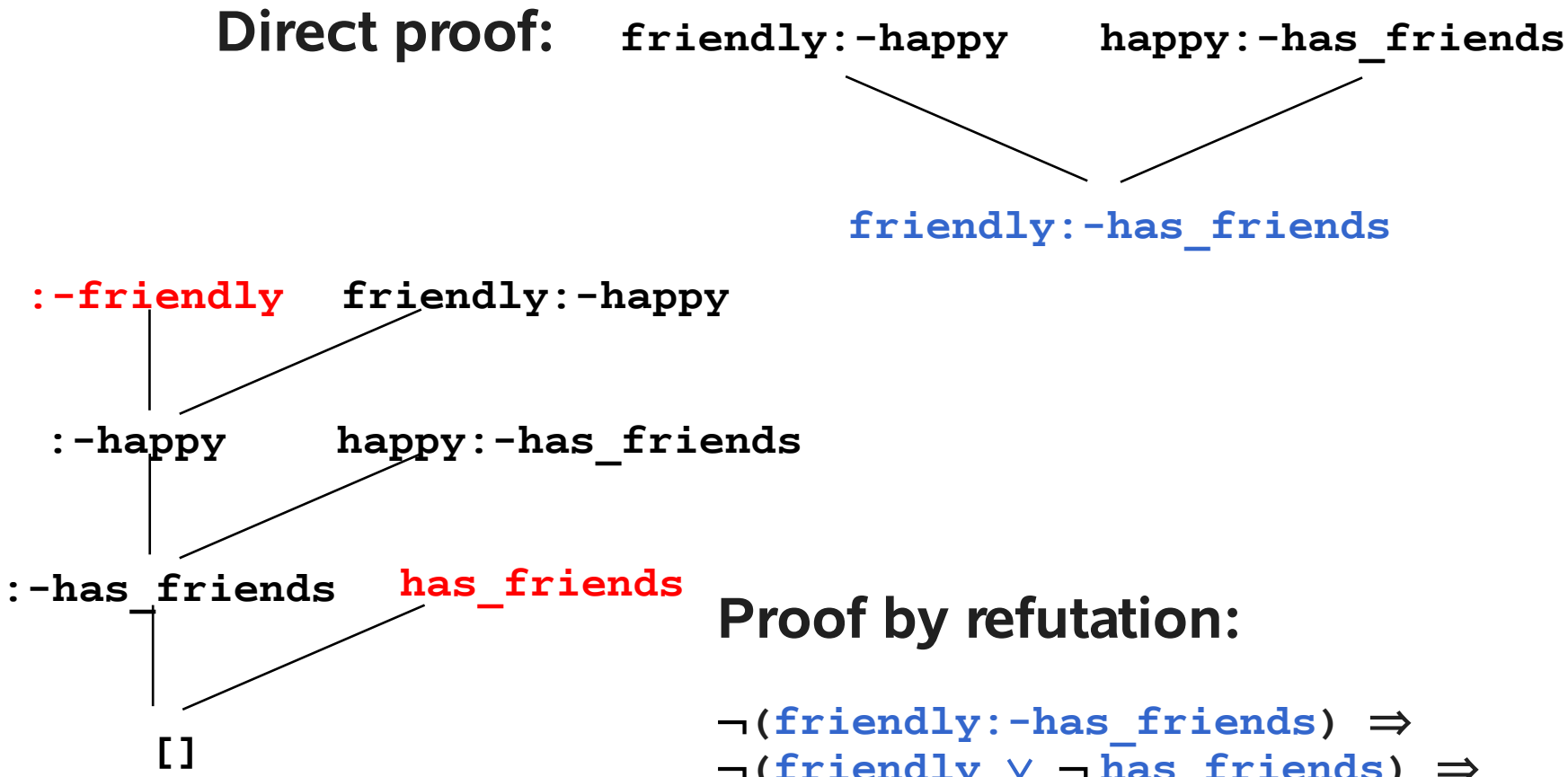
# Exercises


happy;friendly:-teacher
friendly:-teacher,happy
teacher;wise
teacher:-wise
friendly:-teacher
teacher
friendly



# Exercise 2.4

## Direct proof:




## Proof by refutation:

¬(friendly:-has_friends) ⇒
¬(friendly ∨ ¬has_friends) ⇒
(¬friendly) ∧ (has_friends) ⇒
:-friendly and has_friends



Exercise 2.5

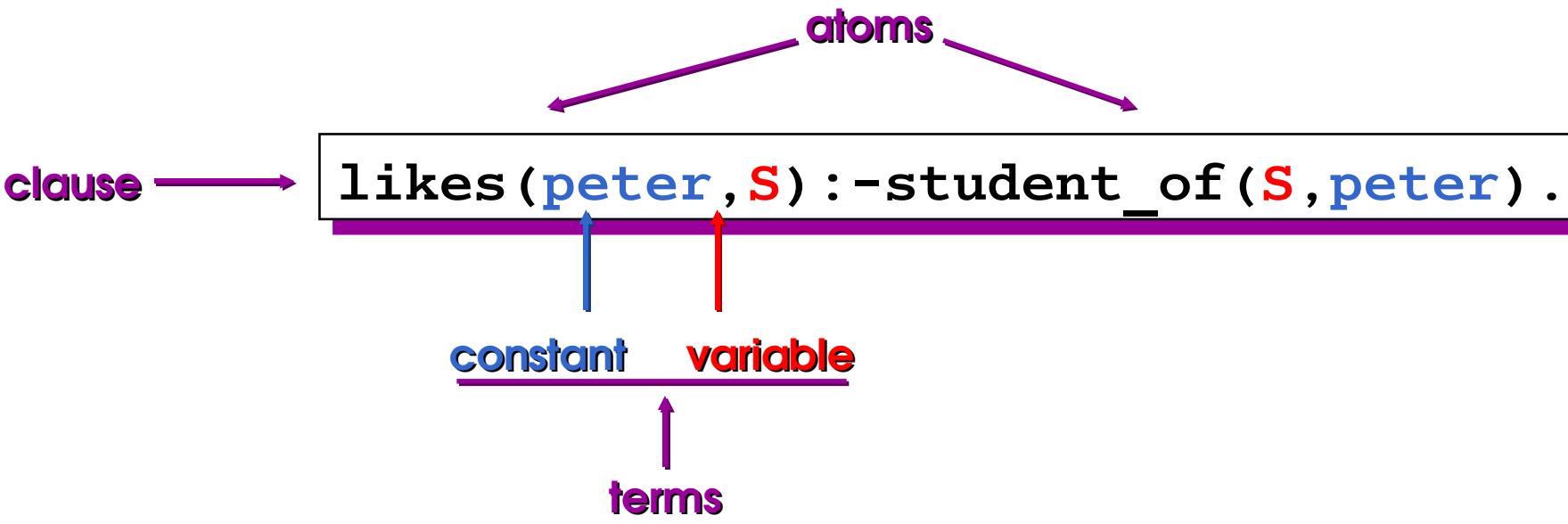
# Relational Clausal Logic

- Atoms consist of predicate symbol and *arguments*
- Number of arguments: *arity*
- *Ground atoms* are atoms without variables
- Variables are
  - *logical variables*
  - *scope:* within the same clause
- „Grounding“ substitutions are applied to clauses, not to programs.
- Renaming such that names different: *standardizing clauses apart*



# Relational clausal logic: syntax

"Peter likes anybody who is his student."






# Relational clausal logic: syntax

- A ***substitution*** maps variables to terms:
  - `{S->maria}`
- A substitution can be ***applied*** to a clause:
  - `likes(peter,maria):-student_of(maria,peter).`
- The resulting clause is said to be an ***instance*** of the original clause, and a ***ground instance*** if it does not contain variables.
- Each instance of a clause is among its logical consequences.

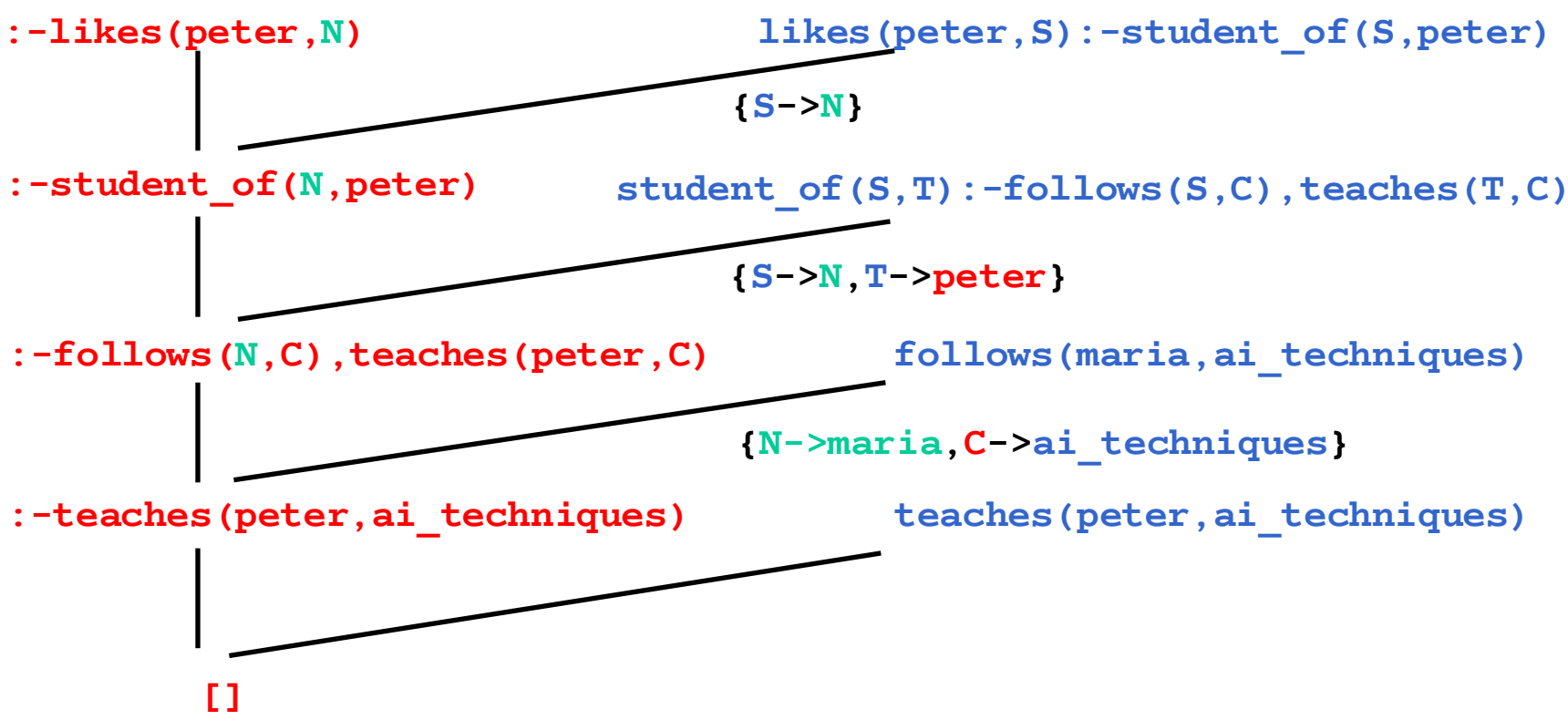


# Substitutions

- ***Herbrand universe***: set of ground terms (i.e. constants)
  - `{peter,maria}`
- ***Herbrand base***: set of ground atoms
  - `{likes(peter,peter),likes(peter,maria),likes(maria,peter), likes(maria,maria),student_of(peter,peter),student_of(peter,maria), student_of(maria,peter),student_of(maria,maria)}`
- ***Herbrand interpretation***: set of **true** ground atoms
  - `{likes(peter,maria),student_of(maria,peter)}`
- An interpretation is a **model** for a clause if it makes all of its ground instances **true**
  - `likes(peter,maria):-student_of(maria,peter).`
  - `likes(peter,peter):-student_of(peter,peter).`



# Relational clausal logic: semantics






# Relational resolution

- Resolution as in propositional clausal logic, but *variables* have to be taken into account.
- Usually, clauses are not exactly the same during resolution. Thus, they are made equal by substitutions.
- Substitutions needed in a resolution proof are derived directly from the clauses involved.
- *Unification*/*Unifiers* (need not be grounding)
- One *unifier* can be more general than another (partial order).
- Used in resolution proof: *most general unifier (mgu)*
- If contradiction (empty clause): clauses are inconsistent under the substitutions that were used for the unification of the literals.



# Relational clausal logic: proof theory

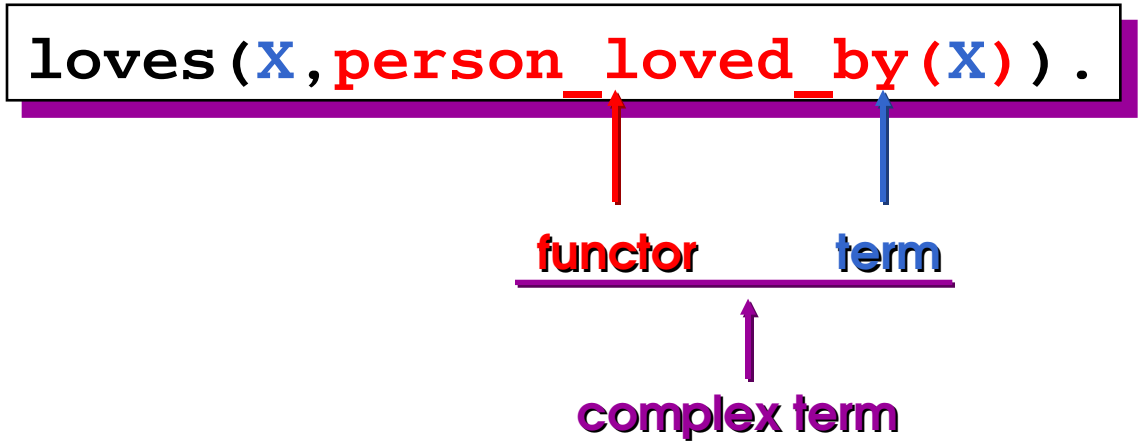
- *sound*
- *refutation complete*
- Herbrand universe finite, thus decidable



# Relational clausal logic: proof theory

# Full Clausal Logic

"Everybody loves somebody."




```
loves(peter,person_loved_by(peter)).
loves(anna,person_loved_by(anna)).
loves(paul,person_loved_by(paul)).
…
```



# Full clausal logic: syntax

Translate into *Clausal Logic*:

(a) Every mouse has a tail.

(b) Somebody loves everybody.

(c) Every two numbers have a maximum.



# Exercise

- Every mouse has a tail
  - `tail_of(tail(X),X):-mouse(X).`
- Somebody loves everybody
  - `loves(person_who_loves_everybody,X).`
- Every two numbers have a maximum
  - `maximum_of(X,Y,max(X,Y)):-number(X),number(Y).`



# Exercise 2.9

- ***Herbrand universe***: set of ground terms
  - `{0,s(0),s(s(0)),s(s(s(0))),…}`
- ***Herbrand base***: set of ground atoms
  - `{plus(0,0,0), plus(s(0),0,0), …,`
    `plus(0,s(0),0), plus(s(0),s(0),0), …,`
    `…,`
    `plus(s(0),s(s(0)),s(s(s(0)))), …}`
- ***Herbrand interpretation***: set of **true** ground atoms
  - `{plus(0,0,0), plus(s(0),0,s(0)), plus(0,s(0),s(0))}`
- Some programs have only infinite models
  - `plus(0,X,X).`
  - `plus(s(X),Y,s(Z)):-plus(X,Y,Z).`



# Full clausal logic: semantics

- Unification now has to take into account subterms.
- Before unification the atoms do not have common variables.
- `plus(s(0), X, s(X))`
  `plus(s(Y), s(0), s(s(Y)))`

  *mgu* (most general unifier) is
  `{Y -> 0, X -> s(0)}`

# Unifikation

Ein *Unifikator* zweier Terme ist eine Substitution, die beide Terme identisch macht.

Zwei Terme heißen *unifizierbar*, wenn es einen solchen Unifikator gibt.

Der *allgemeinste Unifikator* (MGU: most general unifier) zweier Terme ist ein Unifikator, dessen zugeordnete gemeinsame Instanz die allgemeinste ist. Sind zwei Terme unifizierbar, dann gibt es - bis auf das Umbenennen von Variablen - einen eindeutigen allgemeinsten Unifikator. Analog hierzu haben zwei Terme eine bis auf alphabetische Varianten eindeutige allgemeinste Instanz.

Ein Unifikationsalgorithmus berechnet den allgemeinsten Unifikator zweier Terme, wenn dieser existiert; ansonsten wird *fail* zurückgegeben.

# Unifikation (2)

**Eingabe:** Zwei zu unifizierende Terme $T_1$ und $T_2$
**Ausgabe:** $\sigma$, der MGU von $T_1$ und $T_2$, oder $fail$
**Algorithmus:**

Initialisiere die Substitution $\sigma$ zu leer, den Keller
mit der Gleichung $T_1 = T_2$, und $OK$ mit $true$.
solange der Keller nicht leer ist:
hole $X = Y$ vom Keller
  falls
  $X$ eine Variable ist, die nicht in $Y$ vorkommt:
    ersetze $X$ im Keller und in $\sigma$ durch $Y$
    füge $X = Y$ zu $\sigma$ hinzu
  $Y$ eine Variable ist, die nicht in $X$ vorkommt:
    ersetze $Y$ im Keller und in $\sigma$ durch $X$
    füge $Y = X$ zu $\sigma$ hinzu
  $X$ und $Y$ identische Konstanten oder Variablen sind:
    fahre fort
  $X$ gleich $f(X_1, \ldots, X_n)$ und $Y$ gleich $f(Y_1, \ldots, Y_n)$ für
  einen Funktor $f$ und $n > 0$ ist:
    schreibe $X_i = Y_i$, $i = 1, \ldots, n$ auf den Keller
  ansonsten:
    $OK = fail$

wenn $OK = fail$, dann gebe $fail$ aus, ansonsten gebe $\sigma$ aus.

# Unifikation (3)

Wurde der Keller vollständig abgearbeitet, so sind die Terme $T_1$ und $T_2$ unifizierbar, $\sigma$ enthält den allgemeinsten Unifikator (MGU).

Der Occur-Check wird durch den Ausdruck "kommt nicht vor in ..." realisiert. Er ist notwendig, um die Unifikation von Termen wie $s(X)$ und $X$ zu verhindern, zu denen es keine gemeinsame endliche Instanz gibt.

*Die meisten PROLOG-Implementationen lassen den Occur-Check aus dem Unifikationsalgorithmus weg!*

- In the process of unification it may happen that a variable occurs in both atoms.
  Before substituting a variable by a term, we have to check whether the variable already occurs in the term.

- *occurs check*

- Example:

```
loves(X, person_loved_by(X)).
loves(Y, Y).
```



# Occurs check

Unify, if possible, the following pairs of terms:

(a) `plus(X, Y, s(Y))` and
`plus(s(V), W, s(s(V)))`

(b) `length([X|Y], s(0))` and
`length([V], V)`

(c) `larger(s(s(X)), X)` and
`larger(V, s(V))`



# Exercise

`plus(X,Y,s(Y))`
and
`plus(s(V),W,s(s(V)))`
unify to
`plus(s(V),s(V),s(s(V)))`

`length([X|Y],s(0))`
and
`length([V],V)`
unify to
`length([s(0)],s(0))`

`larger(s(s(X),X)`
and
`larger(V,s(V))`
do **not** unify (occur check!)



# Exercise 2.11

- *sound*
- *refutation-complete*
- *semi-decidable*:
  if *A* follows from *B*, then there exists an algorithm that can prove this in finite time.
  if *A* does *not* follow from *B*, then there exists no algorithm that can prove this in finite time for arbitrary *A* and *B*.



# Full clausal logic: meta-theory

| | Propositional — | Relational — | Full clausal logic |
|---|---|---|---|
| **Herbrand universe** | — | `{a, b}`<br>(finite) | `{a, f(a), f(f(a)),…}`<br>(infinite) |
| **Herbrand base** | `{p, q}` | `{p(a,a), p(b,a),…}`<br>(finite) | `{p(a,f(a)), p(f(a), …}`<br>(infinite) |
| **clause** | `p:-q.` | `p(X,Z):-q(X,Y),p(Y,Z).` | `p(X,f(X)):-q(X).` |
| **Herbrand models** | ∅ ∅ ∅<br>`{p}`<br>`{p, q}` | `{p(a,a)}`<br>`{p(a,a), p(b,a), q(b,a)}`<br>…<br>(finite number of finite models) | `{p(a,f(a)), q(a)}`<br>`{p(f(a), f(f(a))), q(f(a))}`<br>…<br>(infinite number of finite or infinite models) |
| **Meta-theory** | sound<br>refutation-complete<br>decidable | sound<br>refutation-complete<br>decidable | sound (if unifying with occur check)<br>refutation-complete<br>semi-decidable |



# Summary