Handout for lecture 8:

Validity in Predicate Logic

1. Predicate logic

1.1 What is predicate logic?

2

Predicate logic

Until now we only reasoned in the world of the domain $\{\mathit{true}, \mathit{false}\}$, together with the usual operations. That, and numeric values through SMT.

In this lecture, out goal is to do automated reasoning in an arbitrary domain D with:

- quantifiers $\forall$ and $\exists$
- `relations`, and
- `functions`.

In SMT we had relations and functions, too; for example $\geq$ and $+$. Here, we will abstract away from the integers, and consider ways to reason about arbitrary domains.

This is called **Predicate logic**.

Running example:

- There is a student that is awake during all lectures.
- During all boring lectures no student keeps awake.

$\Rightarrow$ Then there are no boring lectures.

3

Modeling natural language

In order to express the example by a formula we define relations $\mathtt{S}$, $\mathtt{L}$, $\mathtt{B}$ and $\mathtt{A}$:

- $\mathtt{S}(x)$: x is a student
- $\mathtt{L}(x)$: x is a lecture
- $\mathtt{B}(x)$: x is boring
- $\mathtt{A}(x, y)$: x is awake during y

Hence, we want to prove that:

$$(\exists x[\mathtt{S}(x) \wedge \forall y[\mathtt{L}(y) \rightarrow \mathtt{A}(x, y)]]) \wedge$$

$$(\forall x[(\mathtt{L}(x) \wedge \mathtt{B}(x)) \to \neg\exists y[\mathtt{S}(y) \wedge \mathtt{A}(y,x)]])$$

implies

$$\neg\exists x[\mathtt{L}(x) \wedge \mathtt{B}(x)]$$

4

Formalising predicate logic

But what exactly does that mean?

In these formulas we may have:

- **variables** (here x, y)
- **function symbols** (not here)
- **relation symbols** (here $\mathtt{S}, \mathtt{L}, \mathtt{B}, \mathtt{A}$), also called **predicate symbols**.

Function symbols and relation symbols have an **arity** $= 0, 1, 2, 3, \ldots$

A function symbol of arity 0 is also called a **constant**.

5

Formalising predicate logic

Formally, we introduce **Predicate Logic** as follows.

We inductively define:

- A **term** is:
  - a variable from a set $\mathcal{X}$, or
  - a function symbol of arity n applied on n terms.
- A **predicate (formula)** is:
  - a relation symbol of arity n applied to n terms, or
  - $\forall x[P]$ for a variable $x \in \mathcal{X}$ and a predicate P, or
  - $\exists x[P]$ for a variable $x \in \mathcal{X}$ and a predicate P, or
  - $\neg P$ for a predicate P, or
  - $P \diamond Q$ for predicates P and Q and $\diamond \in \{\vee, \wedge, \to, \leftrightarrow\}$.

1.2 Truth in predicate logic: models

6

Models

If we give meaning to variables, function symbols and relation symbols in a **model**, then a predicate has a boolean value.

More precisely, a **model** is a non-empty set M together with:

- $[\mathtt{f}] : M^n \to M$ for every function symbol $\mathtt{f}$ of arity n
- $[\mathtt{R}] : M^n \to \{\mathit{false}, \mathit{true}\}$ for every relation symbol $\mathtt{R}$ of arity n

For $\alpha : \mathcal{X} \to M$ we define inductively:

- $[\![x]\!]_\alpha = \alpha(x)$ for $x \in \mathcal{X}$
- $[\![\mathtt{f}(t_1, \ldots, t_n)]\!]_\alpha = [\mathtt{f}]([\![t_1]\!]_\alpha, \ldots, [\![t_n]\!]_\alpha)$ for every function symbol $\mathtt{f}$ of arity n and terms $t_1, \ldots, t_n$
- $[\![\mathtt{R}(t_1, \ldots, t_n)]\!]_\alpha = [\mathtt{R}]([\![t_1]\!]_\alpha, \ldots, [\![t_n]\!]_\alpha)$ for every relation symbol $\mathtt{R}$ of arity n and terms $t_1, \ldots, t_n$.

So $[\![\mathtt{f}(t_1, \ldots, t_n)]\!]_\alpha \in M$ and $[\![\mathtt{R}(t_1, \ldots, t_n)]\!]_\alpha \in \{\mathit{false}, \mathit{true}\}$.

7

Modeling connectives and quantifiers

We also define

$$[\![\neg P]\!]_\alpha = \neg [\![P]\!]_\alpha$$

and

$$[\![P \diamond Q]\!]_\alpha = [\![P]\!]_\alpha \diamond [\![Q]\!]_\alpha$$

for $\diamond \in \{\vee, \wedge, \to, \leftrightarrow\}$.

In order to assign a meaning to $\forall$ and $\exists$ we need to modify the valuation α.

We define

$$\begin{aligned} [\![\forall x[P]]\!]_\alpha &= \bigwedge_{m \in M} [\![P]\!]_{\alpha\langle x:=m\rangle} \\ [\![\exists x[P]]\!]_\alpha &= \bigvee_{m \in M} [\![P]\!]_{\alpha\langle x:=m\rangle} \end{aligned}$$

Here, if $\alpha : \mathcal{X} \to M$, $x \in \mathcal{X}$ and $m \in M$ then we define $\alpha\langle x := m\rangle : \mathcal{X} \to M$ by:
$\alpha\langle x := m\rangle(y) = \begin{cases} m & \text{if } y = x \\ \alpha(y) & \text{otherwise} \end{cases}$

We can avoid confusion by disallowing $\forall x$ or $\exists x$ to occur inside P for the same x: choose fresh x for every quantification.

In this way $[\![P]\!]_\alpha \in \{\mathit{false}, \mathit{true}\}$ has been defined for every predicate P and every $\alpha : \mathcal{X} \to M$.

All these definitions are motivated by common knowledge of the usual notions of $\forall$ and $\exists$.

8

Satisfiability and validity

A predicate P is **satisfiable** if there is a model M and $\alpha : \mathcal{X} \to M$ such that $[\![P]\!]_\alpha = \mathit{true}$.

A predicate P is **valid** if there exist no model M and $\alpha : \mathcal{X} \to M$ such that $[\![P]\!]_\alpha = \mathit{false}$.

(That is, if $\neg P$ is unsatisfiable.)

9

Running example

Calling back to our initial problem, we want to prove that **for all models such that**

$$[\![(\exists x[\mathtt{S}(x) \wedge \forall y[\mathtt{L}(y) \to \mathtt{A}(x,y)]])]\!] = \mathit{true}$$
$$[\![(\forall x[(\mathtt{L}(x) \wedge \mathtt{B}(x)) \to \neg\exists y[\mathtt{S}(y) \wedge \mathtt{A}(y,x)]])]\!] = \mathit{true}$$

also

$$[\![\neg\exists x[\mathtt{L}(x) \wedge \mathtt{B}(x)]]\!] = \mathit{true}$$

Put differently, we want to see that

$$(\exists x[\mathtt{S}(x) \wedge \forall y[\mathtt{L}(y) \to \mathtt{A}(x,y)]]) \wedge$$
$$(\forall x[(\mathtt{L}(x) \wedge \mathtt{B}(x)) \to \neg\exists y[\mathtt{S}(y) \wedge \mathtt{A}(y,x)]]) \ \to$$
$$\neg\exists x[\mathtt{L}(x) \wedge \mathtt{B}(x)]$$

holds in all models.

10

Example: a model for the boring lectures

$$(\exists x[\mathtt{S}(x) \wedge \forall y[\mathtt{L}(y) \to \mathtt{A}(x,y)]]) \wedge$$
$$(\forall x[(\mathtt{L}(x) \wedge \mathtt{B}(x)) \to \neg\exists y[\mathtt{S}(y) \wedge \mathtt{A}(y,x)]])$$

Model:

- $M = \{\text{Cynthia, AR}\}$
- $L = [\text{Cynthia} \mapsto \mathit{false}, \text{AR} \mapsto \mathit{true}]$

- $S = [\text{Cynthia} \mapsto \textit{true},\ \text{AR} \mapsto \textit{false}]$
- $B = [\textit{anything} \mapsto \textit{false}]$
- $A = [(\text{Cynthia},\ \text{AR}) \mapsto \textit{true};\ \textit{anything else} \mapsto \textit{false}]$

$$
\begin{aligned}
& \llbracket \exists x[\mathtt{S}(x) \wedge \forall y[\mathtt{L}(y) \to \mathtt{A}(x,y)]]\rrbracket_{[]} \\
\leftrightarrow\ & \textstyle\bigvee_{a\in M} \llbracket \mathtt{S}(x) \wedge \forall y[\mathtt{L}(y) \to \mathtt{A}(x,y)]\rrbracket_{[x:=a]} \\
\leftrightarrow\ & \llbracket \mathtt{S}(x) \wedge \forall y[\mathtt{L}(y) \to \mathtt{A}(x,y)]\rrbracket_{[x:=\text{Cynthia}]} \vee \\
& \llbracket \mathtt{S}(x) \wedge \forall y[\mathtt{L}(y) \to \mathtt{A}(x,y)]\rrbracket_{[x:=\text{AR}]} \\
\leftrightarrow\ & (\llbracket \mathtt{S}(x)\rrbracket_{[x:=\text{Cynthia}]} \wedge \llbracket \forall y[\mathtt{L}(y) \to \mathtt{A}(x,y)]\rrbracket_{[x:=\text{Cynthia}]}) \vee \\
& (\llbracket \mathtt{S}(x)\rrbracket_{[x:=\text{AR}]} \wedge \llbracket \forall y[\mathtt{L}(y) \to \mathtt{A}(x,y)]\rrbracket_{[x:=\text{AR}]}) \\
\leftrightarrow\ & ([\mathtt{S}](\llbracket x\rrbracket_{[x:=\text{Cynthia}]}) \wedge \textstyle\bigwedge_{a\in M} \llbracket \mathtt{L}(y) \to \mathtt{A}(x,y)\rrbracket_{[x:=\text{Cynthia},y:=a]}) \vee \\
& ([\mathtt{S}](\llbracket x\rrbracket_{[x:=\text{AR}]}) \wedge \textstyle\bigwedge_{a\in M} \llbracket \mathtt{L}(y) \to \mathtt{A}(x,y)\rrbracket_{[x:=\text{AR},y:=a]}) \\
\leftrightarrow\ & ([\mathtt{S}](\text{Cynthia}) \wedge \textstyle\bigwedge_{a\in M} \llbracket \mathtt{L}(y) \to \mathtt{A}(x,y)\rrbracket_{[x:=\text{Cynthia},y:=a]}) \vee \\
& ([\mathtt{S}](\text{AR}) \wedge \textstyle\bigwedge_{a\in M} \llbracket \mathtt{L}(y) \to \mathtt{A}(x,y)\rrbracket_{[x:=\text{AR},y:=a]}) \\
\leftrightarrow\ & (\textit{true} \wedge \textstyle\bigwedge_{a\in M} \llbracket \mathtt{L}(y) \to \mathtt{A}(x,y)\rrbracket_{[x:=\text{Cynthia},y:=a]}) \vee \\
& (\textit{false} \wedge \textstyle\bigwedge_{a\in M} \llbracket \mathtt{L}(y) \to \mathtt{A}(x,y)\rrbracket_{[x:=\text{AR},y:=a]}) \\
\leftrightarrow\ & \textstyle\bigwedge_{a\in M} \llbracket \mathtt{L}(y) \to \mathtt{A}(x,y)\rrbracket_{[x:=\text{Cynthia},y:=a]} \\
\leftrightarrow\ & \llbracket \mathtt{L}(y) \to \mathtt{A}(x,y)\rrbracket_{[x:=\text{Cynthia},y:=\text{Cynthia}]} \wedge \\
& \llbracket \mathtt{L}(y) \to \mathtt{A}(x,y)\rrbracket_{[x:=\text{Cynthia},y:=\text{AR}]} \\
\leftrightarrow\ & (\llbracket \mathtt{L}(y)\rrbracket_{[x:=\text{Cynthia},y:=\text{Cynthia}]} \to \llbracket \mathtt{A}(x,y)\rrbracket_{[x:=\text{Cynthia},y:=\text{Cynthia}]}) \wedge \\
& (\llbracket \mathtt{L}(y)\rrbracket_{[x:=\text{Cynthia},y:=\text{AR}]} \to \llbracket \mathtt{A}(x,y)\rrbracket_{[x:=\text{Cynthia},y:=\text{AR}]}) \\
\leftrightarrow\ & ([\mathtt{L}](\text{Cynthia}) \to [\mathtt{A}](\text{Cynthia}, \text{Cynthia})) \wedge \\
& ([\mathtt{L}](\text{AR}) \to [\mathtt{A}](\text{Cynthia}, \text{AR})) \\
\leftrightarrow\ & \textit{true}
\end{aligned}
$$

---

11 Functions

Thus far, we have only used relation symbols. But predicate logic also admits **functions**:

- $\mathtt{S}(x)$: x is a student
- $\mathtt{L}(x)$: x is a lecture
- $\mathtt{B}(x)$: x is boring
- $\mathtt{A}(x,y)$: x is awake during y.
- $\mathtt{F}(x)$: **returns the favourite lecture of x if x is a student**

Formulas may for instance include:

$$\forall x[\mathtt{S}(x) \to \mathtt{A}(x, \mathtt{F}(x))]$$

(All students are awake during their favourite lecture.)

12

Infinite models

Some formulas only have *infinite* models…

$$\begin{gathered}\forall x[\neg \mathtt{L}(x,x)] \wedge \\ \forall x[\mathtt{L}(x,\mathtt{S}(x))] \wedge \\ \forall x[\forall y[\forall z[\mathtt{L}(x,y) \wedge \mathtt{L}(y,z) \rightarrow \mathtt{L}(x,z)]]]\end{gathered}$$

- $M = \mathbb{N}$
- $\mathtt{S}(n) = n+1$
- $\mathtt{L}(n,m) = n < m$

13

Proposition logic

Proposition logic can be seen as a special case of predicate logic in which

- there are no variables,
- there are no function symbols, and
- all relation symbols (corresponding to propositional variables) have arity 0.

1.3 Tools

14

Prover9

Predicate logic can be expressed in SMT, but tools like `Yices` and `Z3` are very weak in quantification.

However, there are some tools specialising in it, e.g., `Prover9`.

Use: `./prover9 -f stud`

where `stud` is a file containing:

```
formulas(assumptions).
(exists x (S(x) & (L(y) -> A(x,y)))).
(L(x) & B(x) -> -(exists y (S(y) & A(y,x)))).
end_of_list.
formulas(goals).
-(exists z (L(z) & B(z))).
```

```
end_of_list.
```

Note: unbound variables are implicitly universally quantified. The second assumption could also be specified as: `(all x ((L(x) & B(x) -> -(exists y (S(y) & A(y,x))))))`.

15

Prover9 syntax

negation: `-`
conjunction: `&`, disjunction: `|`
implication: `->`, bi-implication: `<->`
exists: `exists`, forall: `all`

Terms and variables represent elements of the (unknown) model.
The only (but very useful) built-in predicate on this model is equality: `=`. More about **equational logic** will be discussed in a later lecture!

```
formulas(assumptions).
f(x) = g(y).
h(x) = f(a).
end_of_list.
formulas(goals).
h(a) = f(b).
end_of_list.
```

16

Mace4

In case `Prover9` fails, then maybe the goal does not follow from the assumptions.

One way to prove this is to find a model in which the assumptions hold but the goal does not hold.

Automatically searching for a **finite** model with these properties may be done by the tool `Mace4`, accepting the same format.

If the goal does not follow from the assumptions, then a model exists in which the assumptions hold but the goal does not hold. However, there is not always a *finite* model.

Conversely, it can be the case that the goal follows from the assumptions, but `Prover9` fails to prove this, so if both `Prover9` and `Mace4` fail then no conclusion can be drawn on validity.

1.4 How to prove such implications?

17

Proving implications in predicate logic

So how to prove that

$$\text{assumption}_1 \wedge \cdots \wedge \text{assumption}_n \rightarrow \text{goal}$$

in predicate logic?

The overall approach is exactly the same as what we do in proposition logic!

1. Create the **negation** of the implication:

$$\text{assumption}_1 \wedge \cdots \wedge \text{assumption}_n \wedge \neg\text{goal}$$

2. Transform this formula to **CNF**.

3. Use **resolution** to show that this formula is unsatisfiable!

18

Proving implications in predicate logic

Hence, to prove that

- there is a student that is awake during all lectures;
- during all boring lectures no student keeps awake;

⇒ there are no boring lectures

we want to prove that:

$$(\exists x[\mathtt{S}(x) \wedge \forall y[\mathtt{L}(y) \rightarrow \mathtt{A}(x,y)]]) \wedge$$
$$(\forall x[(\mathtt{L}(x) \wedge \mathtt{B}(x)) \rightarrow \neg\exists y[\mathtt{S}(y) \wedge \mathtt{A}(y,x)]]) \wedge$$
$$\exists x[\mathtt{L}(x) \wedge \mathtt{B}(x)]$$

is unsatisfiable; that is, equivalent to *false*.

2. Transformation to CNF

19

Step 2

As before resolution is only defined for conjunctive normal forms (CNF), where

- a CNF is a conjunction of clauses,
- a clause is a disjunction of literals, and
- a literal is an atomic formula or its negation.

Here an **atomic formula** is an expression of the shape $\mathtt{P}(t_1, \ldots, t_n)$ where $\mathtt{P}$ is a relation symbol of arity n and $t_1, \ldots, t_n$ are terms.

A clause will be interpreted as being universally quantified over all occurring variables.

20

Step 2

Hence, after taking the negation of the formula whose validity we want to prove, our next step is to transform this general predicate into a CNF while maintaining (un)satisfiability.

This transformation consists of:

- **Prenex normal form:** shift all quantifiers $\forall$ and $\exists$ to the front and write the body as a universally quantified CNF.
- **Skolemization:** remove $\exists$ by introducing fresh function symbols.

2.1 Prenex normal form

21

Simplifying predicates

To simplify an arbitrary predicate we apply the following steps:

- Remove $\leftrightarrow$ by applying:

$$A \leftrightarrow B \equiv (A \to B) \wedge (B \to A)$$

- Remove $\to$ by applying:

$$A \to B \equiv (\neg A) \vee B$$

- Remove negations of non-atomic formulas by repeatedly applying:
  - $\neg(A \wedge B) \equiv (\neg A) \vee (\neg B)$
  - $\neg(A \vee B) \equiv (\neg A) \wedge (\neg B)$
  - $\neg(\neg A) \equiv A$
  - $\neg(\forall x[A]) \equiv \exists x[\neg A]$
  - $\neg(\exists x[A]) \equiv \forall x[\neg A]$

22

Moving around quantifiers

The formula obtained at this point is composed of $\vee, \wedge, \exists, \forall$ and literals.

- Rename variables until over every variable there is at most one quantification. For example:
$$\exists x[\mathtt{A}(x)] \vee \forall x[\mathtt{B}(x)] \quad \equiv \quad \exists x[\mathtt{A}(x)] \vee \forall y[\mathtt{B}(y)]$$

- Assuming the domain is non-empty, now all quantors may be moved to the front, for example
$$B \vee \forall x[A] \quad \equiv \quad \forall x[B \vee A]$$
  where x does not occur in B.

  (As a heuristic, move occurrences of $\exists$ before $\forall$, so that, e.g. $\forall x[A] \wedge \exists y[B]$ becomes $\exists y \forall x[A \wedge B]$ rather than $\forall x \exists y[A \wedge B]$.)

  Now the formula consists of a quantor-free body on which a number of quantors is applied.

- Transform the body to CNF using
  - $A \vee (B \wedge C) \quad \equiv \quad (A \vee B) \wedge (A \vee C)$
  - $(A \wedge B) \vee C \quad \equiv \quad (A \vee C) \wedge (B \vee C)$

23

Back to our running example!

Hence, to transform

$$(\exists x[\mathtt{S}(x) \wedge \forall y[\mathtt{L}(y) \rightarrow \mathtt{A}(x, y)]]) \wedge$$
$$(\forall x[(\mathtt{L}(x) \wedge \mathtt{B}(x)) \rightarrow \neg\exists y[\mathtt{S}(y) \wedge \mathtt{A}(y, x)]]) \wedge$$
$$\exists x[\mathtt{L}(x) \wedge \mathtt{B}(x)]$$

We first rename bound variables so they are all unique:

$$(\exists x[\mathtt{S}(x) \land \forall y[\mathtt{L}(y) \to \mathtt{A}(x,y)]]) \land$$
$$(\forall z[(\mathtt{L}(z) \land \mathtt{B}(z)) \to \neg\exists u[\mathtt{S}(u) \land \mathtt{A}(u,z)]]) \land$$
$$\exists v[\mathtt{L}(v) \land \mathtt{B}(v)]$$

Then we remove implications:

$$(\exists x[\mathtt{S}(x) \land \forall y[\neg\mathtt{L}(y) \lor \mathtt{A}(x,y)]]) \land$$
$$(\forall z[\neg(\mathtt{L}(z) \land \mathtt{B}(z)) \lor \neg\exists u[\mathtt{S}(u) \land \mathtt{A}(u,z)]]) \land$$
$$\exists v[\mathtt{L}(v) \land \mathtt{B}(v)]$$

We also remove non-atomic negations:

$$(\exists x[\mathtt{S}(x) \land \forall y[\neg\mathtt{L}(y) \lor \mathtt{A}(x,y)]]) \land$$
$$(\forall z[(\neg\mathtt{L}(z) \lor \neg\mathtt{B}(z)) \lor \forall u[\neg\mathtt{S}(u) \lor \neg\mathtt{A}(u,z)]]) \land$$
$$\exists v[\mathtt{L}(v) \land \mathtt{B}(v)]$$

Next, we move the quantifiers to the left, prioritising existential quantifications:

$$\exists x\exists v\forall y\forall z\forall u[\mathtt{S}(x) \land (\neg\mathtt{L}(y) \lor \mathtt{A}(x,y)) \land$$
$$(\neg\mathtt{L}(z) \lor \neg\mathtt{B}(z)) \lor (\neg\mathtt{S}(u) \lor \neg\mathtt{A}(u,z)) \land$$
$$\mathtt{L}(v) \land \mathtt{B}(v)]$$

We can skip the last step, since the body is already in CNF!

24

Some trickier examples

Challenge:

$$\mathtt{B} \land \forall x[\mathtt{A}(x) \lor \exists y[\mathtt{C}(x,y)]]$$

The heuristic of moving $\exists$ before $\forall$ does **not** allow us to move $\exists$ *out* of a $\forall$ that it is currently inside of.

So this becomes:

$$\forall x[\exists y[\mathtt{B} \land \mathtt{A}(x) \lor \mathtt{C}(x,y)]]$$

and definitely **not**:

$$\exists y[\forall x[\mathtt{B} \land \mathtt{A}(x) \lor \mathtt{C}(x,y)]]$$

Challenge:

$$\forall x[\mathtt{A}(x)] \to \exists x[\mathtt{B}(x)]$$

Don't forget that the simplification step comes **before** the quantifier-moving step! So we first simplify this to:

$$\neg\forall x[\mathtt{A}(x)] \lor \exists x[\mathtt{B}(x)]$$

Which is further simplified to:

$$\exists x[\neg\mathtt{A}(x)] \lor \exists x[\mathtt{B}(x)]$$

And only then do we rename variables and move quantifiers, giving:

$$\exists x[\exists y[\neg\mathtt{A}(x) \lor \mathtt{B}(y)]]$$

If we had instead started to move quantifiers before handling the $\to$, we would incorrectly end up with:

$$\forall x[\exists y[\neg\mathtt{A}(x) \lor \mathtt{B}(x)]]$$

This is clearly not equivalent!

25

Tseitin Transformation?

The result of the process described above is a prenex normal form, i.e., a CNF preceded by a number of quantors, being equivalent to the original predicate.

If the result is too big then Tseitin's transformation (or a variation) can be applied instead. However, usually formulas in predicate logic are not overly large, so that this is typically not necessary.

If we do use the transformation, then variables should be taken into account. For example,

$$\underbrace{\forall z[\underbrace{\exists x[\underbrace{\mathtt{A}(x) \leftrightarrow \underbrace{\forall y[\mathtt{B}(x,y,z)]}_{T_3}]}_{T_2}]}_{T_1} \lor (\underbrace{\mathtt{D}(z) \land \underbrace{\exists y[\mathtt{C}(z,y)]}_{T_5}}_{T_4})]}_{T_0}$$

can be transformed into:

$$\begin{array}{cc}
T_0 & \\
T_0 \leftrightarrow \forall z[T_1(z) \lor T_4(z)] & \forall z[\forall x[T_3(z,x) \leftrightarrow \forall y[\mathtt{B}(x,y,z)]]] \\
\forall z[T_1(z) \leftrightarrow \exists x[T_2(z)]] & \forall z[T_4(Z) \leftrightarrow \mathtt{D}(z) \land T_5(z)] \\
\forall z[\forall x[T_2(z) \leftrightarrow \mathtt{A}(x) \leftrightarrow T_3(z,x)]] & \forall z[T_5(z) \leftrightarrow \exists y[\mathtt{C}(z,y)]]
\end{array}$$

This conjunction of equivalences can then be turned into a prenex normal form by using the simplification rules, and moving quantifiers to the left.

However, in practice this is rarely useful. You will not be required to use a Tseitin Transformation for predicate logic (with quantifiers) on the exam.

2.2 Skolemization

26

Skolemization definition

In order to reach the desired CNF format with implicit universal quantification it remains to eliminate the ∃-symbols.

This is done by **Skolemization**, i.e., replacing

$$\cdots \exists y \cdots (\cdots y \cdots)$$

by

$$\cdots \cdots (\cdots \mathtt{f}(x_1, \ldots, x_n) \cdots)$$

where $\mathtt{f}$ is a fresh function symbol and $x_1, \ldots, x_n$ are the universally quantified variables left of y.

27

Skolemization example

Skolemization applied to

$$\exists z \forall x \exists y \forall u \forall v \exists w [\mathtt{A}(x, y, z, u, v, w)]$$

yields

$$\forall x \forall u \forall v [\mathtt{A}(x, \mathtt{f}(x), \mathtt{c}, u, v, \mathtt{g}(x, u, v))]$$

28

Skolemization correctness

Theorem
A predicate φ is satisfiable if and only if $\mathit{Skolem}(\varphi)$ is satisfiable.

Proof. Deriving a contradiction from $\forall x[\mathtt{A}(x, \mathtt{f}(x))]$ without any knowledge of $\mathtt{f}$ coincides with deriving a contradiction from

$$\exists \mathtt{f} \forall x [\mathtt{A}(x, \mathtt{f}(x))]$$

According to the Axiom of Choice we have

$$\exists \mathtt{f} \forall x [\mathtt{A}(x, \mathtt{f}(x))] \equiv \forall x \exists y [\mathtt{A}(x, y)]$$

□

29

Skolemization overview

By Skolemization all ∃-symbols are removed, introducing a fresh symbol for every removed ∃-symbol.

The result is a CNF for which

- all variables are implicitly universally quantified, and
- satisfiability is equivalent to satisfiability of the original arbitrary predicate formula.

30

Running example

Continuing the example of the boring lectures:

$$\begin{gathered}\exists x \exists v \forall y \forall z \forall u [\mathtt{S}(x) \wedge (\neg \mathtt{L}(y) \vee \mathtt{A}(x, y)) \wedge \\ (\neg \mathtt{L}(z) \vee \neg \mathtt{B}(z) \vee \neg \mathtt{S}(u) \vee \neg \mathtt{A}(u, z)) \wedge \\ \mathtt{L}(v) \wedge \mathtt{B}(v)]\end{gathered}$$

yields the following CNF:

$$\begin{gathered}\forall y \forall z \forall u [\mathtt{S}(\mathtt{a}) \wedge (\neg \mathtt{L}(y) \vee \mathtt{A}(\mathtt{a}, y)) \wedge \\ (\neg \mathtt{L}(z) \vee \neg \mathtt{B}(z) \vee \neg \mathtt{S}(u) \vee \neg \mathtt{A}(u, z)) \wedge \\ \mathtt{L}(\mathtt{e}) \wedge \mathtt{B}(\mathtt{e})]\end{gathered}$$

Since the existential quantifiers all occur at the start, the corresponding functions do not take arguments.

31

Standard form

Typically, we omit the universal quantifiers.

A clause is interpreted as being universally quantified over all the variables occurring in it.

$$\begin{array}{ll} \mathtt{S}(\mathtt{a}) & \wedge \\ (\neg \mathtt{L}(y) \vee \mathtt{A}(\mathtt{a}, y)) & \wedge \\ (\neg \mathtt{L}(z) \vee \neg \mathtt{B}(z) \vee \neg \mathtt{S}(u) \vee \neg \mathtt{A}(u, z)) & \wedge \\ \mathtt{L}(\mathtt{e}) & \wedge \\ \mathtt{B}(\mathtt{e}) & \end{array}$$

3. Resolution

32

Proving validity

Recall: to prove validity of φ:

1. create the **negation** of φ;
2. transform this formula to **prenex normal form**;
3. **Skolemize** it to obtain a CNF;
4. use **resolution** to show that this formula is unsatisfiable!

Remaining step: extend the resolution method to predicate CNFs!

This is fully exploited in for instance the tool `Prover9`.

3.1 Basics

33

Resolution

As before: this is only applicable to formulas in conjunctive normal form.

As before: the goal of resolution is proving that a CNF is unsatisfiable by deriving the empty clause.

Recall the resolution rule:

$$\frac{P \vee V, \quad \neg P \vee W}{V \vee W}$$

Due to terms and variables occurring in atomic formulas and implicit universal quantification of clauses the rule will be slightly more complicated now.

34

Predicate resolution example

$$\begin{array}{ll} \text{(A)} & \mathtt{P}(\mathtt{f}(x), y) \vee \mathtt{Q}(x, y) \\ \text{(B)} & \neg\mathtt{P}(x, \mathtt{g}(y)) \vee \mathtt{R}(x, y) \end{array}$$

Recall that all variables are implicitly universally quantified! So, for instance the variable y in (A) could also have a form $\mathtt{g}(z)$.

A special case of (A) is: $\mathtt{P}(\mathtt{f}(u), \mathtt{g}(v)) \vee \mathtt{Q}(u, \mathtt{g}(v))$.

A special case of (B) is: $\neg\mathtt{P}(\mathtt{f}(u), \mathtt{g}(v)) \vee \mathtt{R}(\mathtt{f}(u), v)$.

On both 'special cases' now resolution yields the new clause

$$\mathtt{Q}(u, \mathtt{g}(v)) \vee \mathtt{R}(\mathtt{f}(u), v)$$

Hence, we want to use resolution to allow deductions such as:

$$\frac{\mathtt{P}(\mathtt{f}(x), y) \vee \mathtt{Q}(x, y), \quad \neg\mathtt{P}(x, \mathtt{g}(y)) \vee \mathtt{R}(x, y)}{\mathtt{Q}(x, \mathtt{g}(y)) \vee \mathtt{R}(\mathtt{f}(x), y)}$$

as a valid resolution step.

3.2 Substitution

35

Substitution

A **substitution** is a map from variables to terms.

A substitution σ can be extended to arbitrary terms and atomic formulas by inductively defining:

Definition

$$x\sigma = \sigma(x)$$

for every variable x and

$$\mathtt{F}(t_1, \dots, t_n)\sigma = \mathtt{F}(t_1\sigma, \dots, t_n\sigma)$$

for every function/relation symbol $\mathtt{F}$.

So $t\sigma$ is obtained from t by replacing every variable x in t by $\sigma(x)$.

For instance, if $\sigma(x) = y$ and $\sigma(y) = \mathtt{g}(x)$ then

$$\mathtt{P}(\mathtt{f}(x), y)\sigma = \mathtt{P}(\mathtt{f}(y), \mathtt{g}(x))$$

36

Composing substitutions

If both σ and τ are substitutions we can define the **composition** $\sigma\tau$:

Definition

$$x(\sigma\tau) = (x\sigma)\tau$$

for all variable $x \in X$, by which we have

$$t(\sigma\tau) = (t\sigma)\tau$$

for all terms t.

Hence $\sigma\tau$ means: first apply σ, then τ.

Here we do not have the usual confusion of composition in prefix notation, where $f \circ g$ means: first apply g, then f.

37

Generalizing the resolution rule

For a clause $V = P_1 \vee P_2 \vee \cdots \vee P_n$ and a substitution σ we write

$$V\sigma = P_1\sigma \vee P_2\sigma \vee \cdots \vee P_n\sigma$$

For every substitution σ we want to consider the clause $V\sigma$ as a special case of the clause V.

Now the general version of the **resolution rule** for predicates reads:

$$\frac{P \vee V \qquad \neg Q \vee W}{V\sigma \vee W\tau} \text{ for } \sigma, \tau \text{ substitutions with } P\sigma = Q\tau$$

38

Unification

To proceed with resolution, we need a way to determine whether substitutions σ, τ exist such that $P\sigma = Q\tau$ for given atomic formulas P and Q.

This algorithm is called **unification**.

Examples:

- $\mathtt{P}(\mathtt{f}(x), y)$ and $\mathtt{P}(x', \mathtt{g}(y'))$ unify by choosing
  $\sigma = [x \mapsto x, y \mapsto \mathtt{g}(y)]$,
  $\tau = [x' \mapsto \mathtt{f}(x), y' \mapsto y]$.
- $\mathtt{P}(\mathtt{f}(x), y)$ and $\mathtt{Q}(x', \mathtt{g}(y'))$ do not unify.
- $\mathtt{P}(\mathtt{f}(x), \mathtt{f}(y))$ and $\mathtt{P}(x', \mathtt{g}(y'))$ do not unify.

- $\mathtt{P}(\mathtt{f}(x), x)$ and $\mathtt{P}(x', x')$ do not unify.

- $\mathtt{P}(x, x)$ and $\mathtt{P}(\mathtt{g}(y'), \mathtt{g}(\mathtt{h}(z, \mathtt{a})))$ unify by choosing
  $\sigma = [x \mapsto \mathtt{g}(\mathtt{h}(z, \mathtt{a}))]$
  $\tau = [y' \mapsto \mathtt{h}(z, \mathtt{a}))]$

In many cases, it is intuitively clear which unifiers we should choose. Hence, for now we will skip the unification algorithm, and continue as though we already have such an algorithm. Later in this handout we will get back to how one can always find these unifiers—even in cases where the same variable appears multiple times, which makes the unification problem significantly harder to do with just your intuition.

3.3 The algorithm

39

Applying resolution steps

Thus, we can use resolution as follows:

- Take two (possibly equal) non-empty clauses.

- Choose a positive literal P in one of the clauses and a negative literal $\neg Q$ in the other.

- Try to find σ, τ with $P\sigma = Q\tau$:

  - If no such σ, τ exist, a corresponding resolution step is not possible.
  - If they do exist, then conclude
    $$V\sigma \vee W\tau$$
    where:
    * $P \vee V$ is the one clause,
    * $\neg Q \vee W$ is the other clause,

3.4 Some examples

40

Examples of resolution sequences

1	$\mathtt{P}(x, \mathtt{f}(y)) \vee \neg\mathtt{P}(x, y)$	
2	$\neg\mathtt{P}(x, \mathtt{f}(\mathtt{f}(y)))$	
3	$\mathtt{P}(\mathtt{a}, \mathtt{g}(y))$	
4	$\mathtt{P}(\mathtt{a}, \mathtt{f}(\mathtt{g}(y)))$	$(1, 3)$
5	$\mathtt{P}(\mathtt{a}, \mathtt{f}(\mathtt{f}(\mathtt{g}(y))))$	$(1, 4)$
	$\perp$	$(2, 5)$

Or alternatively:

1	$\mathtt{P}(x, \mathtt{f}(y)) \lor \neg\mathtt{P}(x, y)$	
2	$\neg\mathtt{P}(x, \mathtt{f}(\mathtt{f}(y)))$	
3	$\mathtt{P}(\mathtt{a}, \mathtt{g}(y))$	
4	$\neg\mathtt{P}(x, \mathtt{f}(y))$	$(1, 2)$
5	$\neg\mathtt{P}(x, y)$	$(1, 4)$
	$\bot$	$(3, 5)$

Completing the running example

We finish our boring lectures example with the following resolution proof:

1	$\mathtt{S}(\mathtt{a})$	
2	$\neg\mathtt{L}(y) \lor \mathtt{A}(\mathtt{a}, y)$	
3	$\neg\mathtt{L}(z) \lor \neg\mathtt{B}(z) \lor \neg\mathtt{S}(u) \lor \neg\mathtt{A}(u, z)$	
4	$\mathtt{L}(\mathtt{e})$	
5	$\mathtt{B}(\mathtt{e})$	
6	$\mathtt{A}(\mathtt{a}, \mathtt{e})$	$(2, 4)$
7	$\neg\mathtt{B}(\mathtt{e}) \lor \neg\mathtt{S}(u) \lor \neg\mathtt{A}(u, \mathtt{e})$	$(3, 4)$
8	$\neg\mathtt{S}(u) \lor \neg\mathtt{A}(u, \mathtt{e})$	$(5, 7)$
9	$\neg\mathtt{A}(\mathtt{a}, \mathtt{e})$	$(1, 8)$
	$\bot$	$(6, 9)$

4. Unification

4.1 Rephrasing the unification problem

42

Unification: one versus two

We have completed the resolution algorithm, save for one rather important step: we did not yet discuss how we find unifiers!

By renaming variables, we can avoid the need for two substitutions, and find just one unifier. For example: to unify

$$\mathtt{P}(x, \mathtt{g}(y)) \quad \text{with} \quad \mathtt{P}(\mathtt{g}(x), y)$$

We first rename variables:

$$\mathtt{P}(x_1, \mathtt{g}(y_1)) \quad \text{with} \quad \mathtt{P}(\mathtt{g}(x_2), y_2)$$

Then we need a single substitution σ such that:

$$\mathtt{P}(x_1, \mathtt{g}(y_1))\sigma \quad \text{with} \quad \mathtt{P}(\mathtt{g}(x_2), y_2)\sigma$$

We also observe that, for the purposes of unification, there is no difference between terms and atomic formulas, nor between function symbols and relation symbols.

Hence, the general unification problem becomes:

Given two terms P and Q, is there a substitution σ such that $P\sigma = Q\sigma$?

If so, find it.

The resulting substitution σ is called a **unifier**.

43

Most general unifier

We make the following observation:
if σ is a unifier for (P, Q)
and τ is an arbitrary substitution
then $\sigma\tau$ is a unifier for (P, Q).

Definition

A unifier σ_0 is called a **most general unifier (mgu)** for (P, Q) if for every unifier σ for (P, Q) a substitution τ exists such that $\sigma = \sigma_0\tau$.

Observation: if both σ_0 and σ_1 are an mgu for (P, Q), then by definition τ_0, τ_1 exist such that

$$\sigma_1 = \sigma_0\tau_0 \quad \text{and} \quad \sigma_0 = \sigma_1\tau_1$$

From this property one can conclude that σ_0 and σ_1 are equal up to renaming, hence an mgu is unique up to renaming.

Hence we will speak about **the** mgu rather than **an** mgu.

4.2 Finding the mgu

44

Key result

Theorem

For all terms s, t.

If (s, t) unify, then there exists a most general unifier for (s, t).

We prove this result by giving an algorithm with two terms as input, and as output:

- whether the terms unify or not, and
- the mgu in case they unify.

The existence of an mgu in case the two terms unify is a consequence of this property of the algorithm.

45

Algorithm ingredients

Write $v(t)$ for the test whether t is a variable.

Write $in(x, t)$ for the test whether the variable x occurs in t; this is called **occur check**.

The unification algorithm has the following invariant:

$$\exists\sigma[P\sigma = Q\sigma] \;\equiv$$

$$\exists\sigma[\forall(t, u) \in S[t\sigma = u\sigma]] \;\wedge\; (\exists\sigma[P\sigma = Q\sigma] \;\rightarrow\; un)$$

where P, Q are the original terms to be unified.

If $S = \emptyset$ then it follows that P and Q unify.

If $\neg un$ then it follows that P and Q do not unify.

Inspired by the invariant and these observations we arrive at the following unification algorithm:

46

Algorithm

```
$S := \{(P, Q)\}$;
$un := true$;
while $(un \wedge S \neq \emptyset)$ do {
    choose $(t, u) \in S$;
    $S := S \setminus \{(t, u)\}$;
    if $t \neq u$ then
        if $v(t) \wedge in(t, u)$ then $un := false$
        else if $v(t) \wedge \neg in(t, u)$ then
            $S := S[t := u]$
        else if $v(u) \wedge in(u, t)$ then
            $un := false$
        else if $v(u) \wedge \neg in(u, t)$ then
            $S := S[u := t]$
        else if $t = \mathtt{f}(t_1, \ldots, t_n) \wedge u = \mathtt{f}(u_1, \ldots, u_n)$ then
            $S := S \cup \{(t_1, u_1), \ldots, (t_n, u_n)\}$
        else if $t = \mathtt{f}(\cdots) \wedge u = \mathtt{g}(\cdots) \wedge \mathtt{f} \neq \mathtt{g}$ then
            $un := false$
}
```

47

Basic idea of the algorithm

- S is inspected and decomposed.
- un is set to *false* if a reason is found that there is no unification, then the algorithm stops.
- if such a reason is not found and the list S of unification requirements is empty, then there is a unifier.

This unification algorithm always terminates, since in every step either

- the total number of variables occurring in S strictly decreases, or
- the total number of variables occurring in S remains the same and the total size of S decreases.

Here the total size of $\{(t_1, u_1), \ldots, (t_n, u_n)\}$ is defined to be

$$\sum_{i=1}^{n} (|t_i| + |u_i|)$$

where $|t|$ is the size of the term t.

After termination the following holds:

- $un = \mathit{false}$ and P and Q are not unifiable, or
- $un = \mathit{true}$ and P and Q are unifiable

Finding the unifier

If P and Q are unifiable, what about the unifier?

We extend the program by building up the unifier in a variable *mgu*.

As only steps are done that are really forced, the resulting unifier *mgu* will be a most general unifier of P and Q.

We write *id* for the substitution mapping every variable on itself.

For a variable x and a term P we write $[x := P]$ for the substitution mapping x on P and every other variable on itself.

This notation will be used for pairs of terms and sets of pairs of terms, meaning that the substitution is applied to every occurring term.

Updated algorithm

```
$S := \{(P,Q)\}$;
$un := true$;
$mgu := id$;
while $(un \wedge S \neq \emptyset)$ do {
    choose $(t,u) \in S$;
    $S := S \setminus \{(t,u)\}$;
    if $t \neq u$ then
     if $v(t) \wedge in(t,u)$ then $un := false$
     else if $v(t) \wedge \neg in(t,u)$ then {
        $S := S[t := u]$;
        $mgu := mgu\ [t := u]$ }
     else if $v(u) \wedge in(u,t)$ then
         $un := false$
     else if $v(u) \wedge \neg in(u,t)$ then {
        $S := S[u := t]$ ;
        $mgu := mgu\ [u := t]$ }
     else if $t = \mathtt{f}(t_1,\ldots,t_n) \wedge u = \mathtt{f}(u_1,\ldots,u_n)$ then
        $S := S \cup \{(t_1,u_1),\ldots,(t_n,u_n)\}$
     else if $t = \mathtt{f}(\cdots) \wedge u = \mathtt{g}(\cdots) \wedge \mathtt{f} \neq \mathtt{g}$ then
        $un := false$
}
```

Example

Unify $\mathtt{P}(\mathtt{f}(x), y)$ and $\mathtt{P}(z, \mathtt{g}(w))$

Start:
$S = \{(\mathtt{P}(\mathtt{f}(x), y), \mathtt{P}(z, \mathtt{g}(w)))\}, mgu = id$

After 1 step:
$S = \{(\mathtt{f}(x), z), (y, \mathtt{g}(w)))\}, mgu = id$

After 2 steps:
$S = \{(y, \mathtt{g}(w))\}, mgu = [z := \mathtt{f}(x)]$

After 3 steps:
$S = \emptyset, mgu = [z := \mathtt{f}(x)][y := \mathtt{g}(w)]$

So the resulting most general unifier σ is given by

$$x\sigma = x,\ y\sigma = \mathtt{g}(w),\ z\sigma = \mathtt{f}(x),\ w\sigma = w$$

Another example

Unify $\mathtt{P}(\mathtt{f}(x), x)$ and $\mathtt{P}(y, \mathtt{g}(y))$

Start:
$S = \{(\mathtt{P}(\mathtt{f}(x), x), \mathtt{P}(y, \mathtt{g}(y)))\}, mgu = id$

After 1 step:
$S = \{(\mathtt{f}(x), y), (x, \mathtt{g}(y)))\}, mgu = id$

After 2 steps:
$S = \{(x, \mathtt{g}(\mathtt{f}(x)))\}, mgu = [y := \mathtt{f}(x)]$

After 3 steps:
x occurs in $\mathtt{g}(\mathtt{f}(x))$, hence no unification.

Remarks

- If two terms unify, then they have an mgu which is unique up to renaming of variables.
- The occur check can be expensive: searching for a particular variable in a big term.
- There are optimizations of this unification algorithm that are linear.
- The unifier can have a size that is exponential in the size of the terms to be unified.
- Efficient algorithms use DAG representation for terms.

53

The difficulty of unification

Unification of

$$\mathtt{P}(x_1, \mathtt{f}(x_2, x_2), x_2, \mathtt{f}(x_3, x_3), x_3)$$

and

$$\mathtt{P}(\mathtt{f}(y_1, y_1), y_1, \mathtt{f}(y_2, y_2), y_2, \mathtt{f}(y_3, y_3))$$

yields an mgu σ in which $x_1\sigma$ is a term containing 32 copies of the same variable and 31 f-symbols.

4.3 Using the mgu in resolution

54

Rephrasing resolution

Recall the resolution rule:

$$\frac{P \vee V \qquad \neg Q \vee W}{V\sigma \vee W\tau} \text{ for } \sigma, \tau \text{ substitutions with } P\sigma = Q\tau$$

Here we can rename variables, so that $P \vee V$ and $\neg Q \vee W$ have no variables in common.

Now we restrict this general version of resolution:

> Instead of allowing the rule for all (infinitely many) unifiers of P and Q we **only** allow the mgu.

So the resolution step can be described as follows:

$$\frac{P \vee V \qquad \neg Q \vee W}{V\tau \vee W\tau} \text{ for } \tau \text{ an } mgu\ \tau \text{ such that } P\tau = Q\tau$$

After all, if σ is a different unifier (that is, also $P\sigma = Q\sigma$), then $V\sigma \vee W\sigma$ is a special case of $V\tau \vee W\tau$! (Since $V\sigma \vee W\sigma = (V\tau \vee W\tau)\rho$ if $\sigma = \tau\rho$.)

55

Resolution with mgu – overview

Summarising the full algorithm for a resolution step:

- Take two (possibly equal) non-empty clauses.
- Rename variables such that they do not have variables in common.
- Choose a positive literal P in one of the clauses and a negative literal $\neg Q$ in the other.

- Try to unify P and Q:
  - If they do not unify a corresponding resolution step is not possible.
  - If they do unify then the clause

    $$(V \vee W)\sigma$$

    can be concluded, where
    * $P \vee V$ is the one clause,
    * $\neg Q \vee W$ is the other clause,
    * σ is the most general unifier of P and Q

5. Concluding remarks

5.1 Completeness and undecidability of resolution

56

Refutation-completeness

Given a CNF, there are only finitely many possibilities of doing a resolution step, and these are computable.

We can use this to obtain the following result:

Theorem

(refutation-completeness of resolution)

A predicate in CNF is equivalent to *false* if and only if there is a sequence of resolution steps ending in the empty clause.

57

Undecidability

However, we also have the following result:

Theorem

(undecidability of predicate logic)

No algorithm exists that can establish in all cases whether a given predicate in CNF is equivalent to *false*.

These two theorems look contradictory, but they are not:

> After an extensive but unsuccessful search for a resolution sequence ending in the empty clause, you are not able to conclude that such a resolution sequence does not exist.

(Compare this to the halting problem!)

We do not prove undedidability. The proof of refutation-completeness will be included in a later handout.

5.2 Overall overview

58

Summary: proving implications

How to prove that $\varphi_1 \wedge \cdots \wedge \varphi_n$ implies ψ?

1. create the **negation** of the implication: $\varphi_1 \wedge \cdots \wedge \varphi_n \wedge \psi$
2. **simplify** the resulting formula to Prenex normal form
3. use **Skolemization** to obtain a CNF
4. use **resolution** to prove *unsatisfiability* of the CNF

5.3 Quiz

59

Quiz: a counterintuitive example?

> Given that there is at least one student, we know: for all lectures there is a student such that: if (s)he understands the lecture, then all students do.

(Because either all students understand the lecture, or you can pick one who doesn't.)

Task: prove this!

1. Present this statement in predicate logic, using the relations:
   - `S` (arity 1): $\texttt{S}(x)$ indicates that x is a student
   - `L` (arity 1): $\texttt{L}(x)$ indicates that x is a lecture
   - `U` (arity 2): $\texttt{U}(x, y)$ indicates that x understands y
2. Find the negation of this formula, and bring it in Prenex normal form.
3. Skolemize this negated formula.
4. Use resolution to derive $\bot$ from the resulting set of clauses.