COMP30026 Models of Computation

Predicate Logic: Unification and Resolution

Lecture Week 5 Part 1 (Zo

Semester 2, 2021

This Lecture is Being Recorded

Notation for Variables and Constants

Recall again our convention: We use letters from the start of the alphabet (a, b, c, ...) for constants, and letters from the end of the alphabet (u, v, x, y ...) for variables.

This disti

We also us ction symbols, and, of course, upper case letters for predi ls.

In some contexts it is important to distinguish funct s and predicate symbols. As far as unification e is no difference—the unification algorithm treats $f(x, a)$ and $P(f(x, a), x)$ the same way, so for now, let us just consider both "terms".

Substitutions

A substitution is a finite set of replacements of variables by terms, that is, a set θ of the form $\{x_1 \mapsto t_1, x_2 \mapsto t_2, \ldots, x_n \mapsto t_n\}$, where the x_i are variables and the t_i are terms.

We can als

atomic fo

simulta i i

Example. If F is $P(f(x), g(y, y, b))$ an

$\theta = \{x \mapsto h(u), y \mapsto a, z \mapsto c\}$ then $\theta($ $, b))$.

Note: Similar to a valuation, but a substitution maps a variable to a term, and, by natural extension, terms to terms.

Most General Unifiers

A unifier of two terms s and t is a substitution θ such that $\theta(s) = \theta(t)$.

The terms s and t are unifiable iff there exists a unifier for s and t.

A most g[illegible] such that

1. θ i[illegible]
2. every other unifier σ of s and t ca[illegible] for some substitution τ.

(The composition $\tau \circ \theta$ is the substitution we get by first using θ, and then using τ on the result produced by θ.)

Theorem. If s and t are unifiable, they have a most general unifier.

Unifier Examples

1. $P(x, a)$ and $P(b, c)$ are not unifiable.
2. $P(f(x), y)$ and $P(a, w)$ are not unifiable.
3. $P(x, c)$ and $P(a, y)$ are unifiable using $\{x \mapsto a, y \mapsto c\}$.
4. $P($ … $, y \mapsto c\}$.
5. **No**

The last case relies on a principle that a variable (such a … t unifiable with any term containing x (a

If we were allowed to have a substitution $\{x \mapsto f(f(f(\ldots)))\}$, that would be a unifier for the last example. But we cannot have that, as terms must be finite.

More Unifier Examples

Now consider $P(f(x), g(y, a))$ and $P(f(a), g(z, a))$.

The following are all unifiers, so which is "best"?

- $\{x \mapsto$ [illegible]
- $\{x \mapsto$ [illegible]
- $\{x \mapsto$ [illegible]
- $\{x \mapsto a, z \mapsto y\}$

More Unifier Examples

Now consider $P(f(x), g(y, a))$ and $P(f(a), g(z, a))$.

The following are all unifiers, so which is "best"?

- $\{x \mapsto$
- $\{x \mapsto$
- $\{x \mapsto$
- $\{x \mapsto a, z \mapsto y\}$

The first and the last are mgus. They avoi commitments. The second commits y and z to take the value a, which was not really needed in order to unify the two formulas.

Note that $\{x \mapsto a, y \mapsto a, z \mapsto a\} = \{z \mapsto a\} \circ \{x \mapsto a, y \mapsto z\}$.

A Unification Algorithm

In the following, let x be a variable, let F and G be function or predicate names, and let s and t be arbitrary terms.

Input:

Output ... and t otherwi

Algorithm: Start with the set of equati
(This is a singleton set: it has one element.)

As long as some equation in the set has one of the six forms listed on the next slide, perform the corresponding action.

Unification: Solving Term Equations

1. $F(s_1, \dots, s_n) = F(t_1, \dots, t_n)$:
   - Replace the equation by the n equations $s_1 = t_1, \dots, s_n = t_n$.

2. $F(s_1, \dots, s_n) = G(t_1, \dots, t_m)$ where $F \neq G$ or $n \neq m$:
   -
3. x
   -

4. $t = x$ where t is not a variable:
   - Replace the equation by $x = t$.

5. $x = t$ where t is not x but x occu
   - Halt, returning 'failure'.
6. $x = t$ where t contains no x but x occurs in other equations:
   - Replace x by t in those other equations.

Starting from

$$f(h(y), g(y, a), z) = f(x, g(v, v), b)$$

we rewrit

$$\xRightarrow{(1,4)} \quad = h(a) \quad = a \quad = a$$

$$z = b$$

$$z$$

The last set is in normal form and corres

$$\theta = \{x \mapsto h(a), y \mapsto a, v \mapsto a, z \mapsto b\}$$

which indeed unifies the two original terms.

Starting from

$$f(x, a, x) = f(h(z, b), y, h(z, y))$$

we rewrite:

$$\xRightarrow{(1,4)} \begin{array}{l} x \\ y \\ x \end{array} \qquad \begin{array}{l} h(z, b) \\ a \\ z \end{array}$$

$$\xRightarrow{(3)} \begin{array}{lcl} x & = & h(z, b) \\ y & = & a \\ y & = & b \end{array} \qquad \xRightarrow{(6)} \begin{array}{lcl} x & & \\ y & & \\ a & = & b \end{array}$$

So the two original terms are not unifiable.

Solving Term Equations: Example 3

Starting from

$$f(x, g(v, v), x) = f(h(y), g(y, z), z)$$

we rewrite:

$z)$

$$\stackrel{(1)}{\Longrightarrow} \begin{array}{rcl} x & = & h(y) \\ v & = & y \\ v & = & z \\ z & = & h(y) \end{array} \quad \stackrel{(6)}{\Longrightarrow} \begin{array}{rcl} x & = & h(y) \\ z & = & y \\ v & = & z \\ z & = & h(y) \end{array} \quad \begin{array}{rcl} v & & y \\ y & = & h(y) \end{array}$$

failure

This is "failure by occurs check": The algorithm fails, as soon as we discover the equation $y = h(y)$.

Term Equations as Substitutions

The process of solving term equations always halts.

When it halts without reporting 'failure', the term equation system is left in a normal form: On the left-hand sides we have variables only, and they a owhere in the right-

If the normal form is $\{x_1 = t_1, \ldots, x_n$

$$\{x_1 \mapsto t_1, \ldots, x_n$$

is a most general unifier for the input terms s and t.

If the result is 'failure', no unifier exists.

Resolvents

Recall how we defined resolvents for propositional logic:

- Two literals are complementary if one is L and the other is $\neg L$.
- Th ls
  L,

For predi

- Two literals L and $\neg L'$ are comple able.

- Let C_1 and C_2 be clauses, rename
  complementary literals $\{L, \neg L'\}$ with L a literal in C_1 and $\neg L'$ a literal in C_2. Then the resolvent of C_1 and C_2 is the union $\theta(C_1 \setminus \{L\}) \cup \theta(C_2 \setminus \{\neg L'\})$.

- Every shark eats a tadpole

$$\forall x(S(x) \Rightarrow \exists y(T(y) \land E(x, y)))$$

- All large white fish are sharks

- Col

$$W(colin) \land$$

- Any tadpole eaten by a deep water fish is misera

$$\forall z((T(z) \land \exists y(D(y) \land E(y, z))) \Rightarrow M(z))$$

- Therefore some tadpole is miserable

$$\therefore \exists z(T(z) \land M(z))$$

Tadpoles in Clausal Form

Every shark eats a tadpole

- $\forall x \, (S(x) \Rightarrow \exists y \, (T(y) \land E(x, y)))$ $\{\neg S(x), T(f(x))\}$
- $\{\neg S(x), E(x, f(x))\}$

All large w

- $\forall x$ ($(x), S(x)\}$

Colin is a lar

- $W(co$ $\{W(colin)\}$
- $colin)\}$

Any tadpole eaten by a deep water fish is miserable

- $\forall z \, ((T(z) \land \exists y \, (D(y) \land E(y, z))) \Rightarrow$
- $\{\neg T(z), \neg D(y), \neg E(y, z), M(z)\}$

Negation of: Some tadpole is miserable

- $\neg \exists z \, (T(z) \land M(z))$ $\{\neg T(z), \neg M(z)\}$

A Refutation

Let us find a refutation of the set of seven clauses.

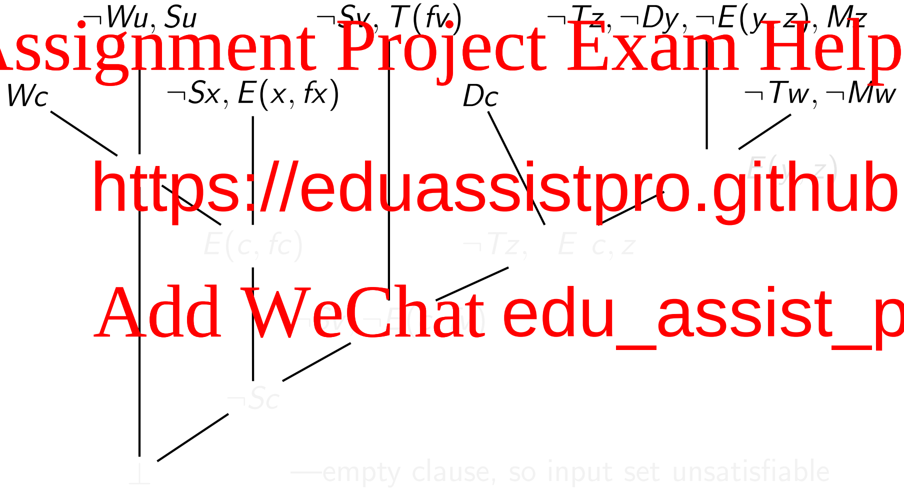
To save space, we leave out braces and some parentheses, for example

$$y, z), Mz$$

$$Wc \qquad \neg Sx, E(x, fx) \qquad Dc \qquad , \neg Mv$$

Many different resolution proofs are possible—the next slides show one.

A Refutation for the Tadpole Example



A Refutation for the Tadpole Example

$\neg Wu, Su$ $\quad$ $\neg Sv, T(fv)$ $\quad$ $\neg Tz, \neg Dy, \neg E(y, z), Mz$

Wc $\quad$ $\neg Sx, E(x, fx)$ $\quad$ Dc $\quad$ $\neg Tw, \neg Mw$

$\neg Tz, \neg Dy, \neg E(y, z)$

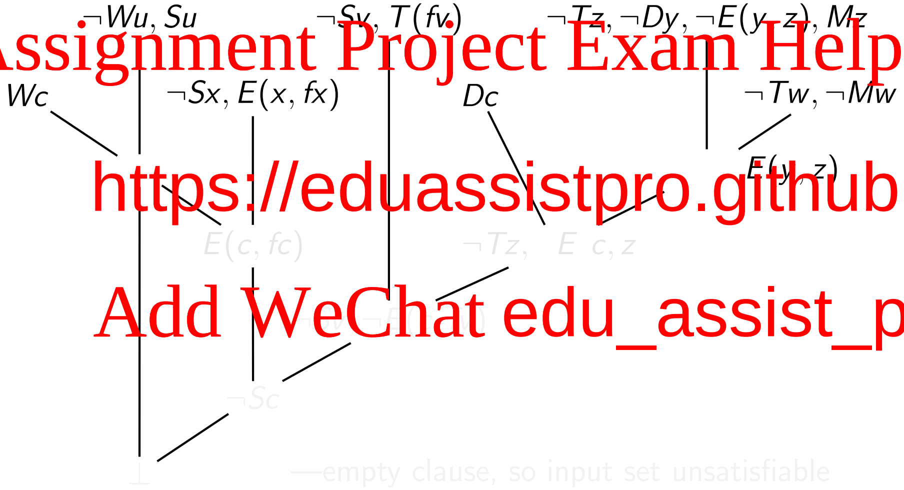
$E(c, fc)$ $\quad$ $\neg Tz, \neg E(c, z)$

$\neg Sv, \neg E(c, fv)$

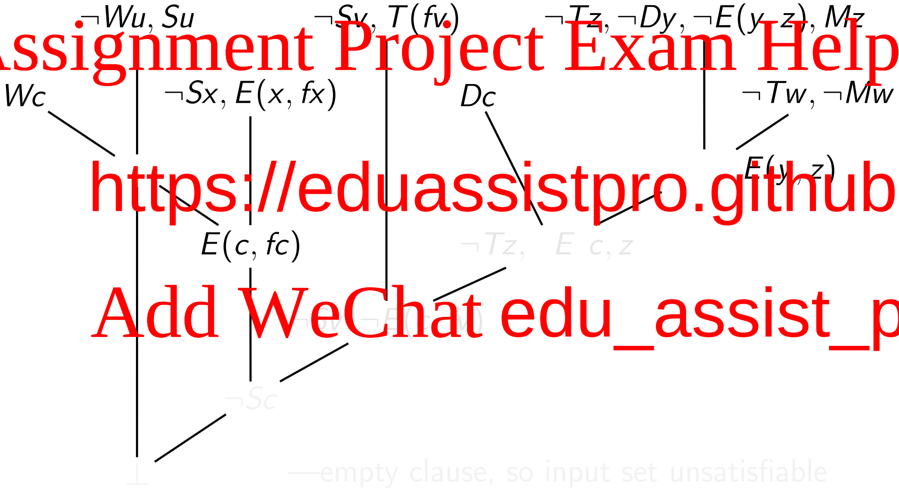
$\neg Sc$

$\bot$ —empty clause, so input set unsatisfiable

A Refutation for the Tadpole Example

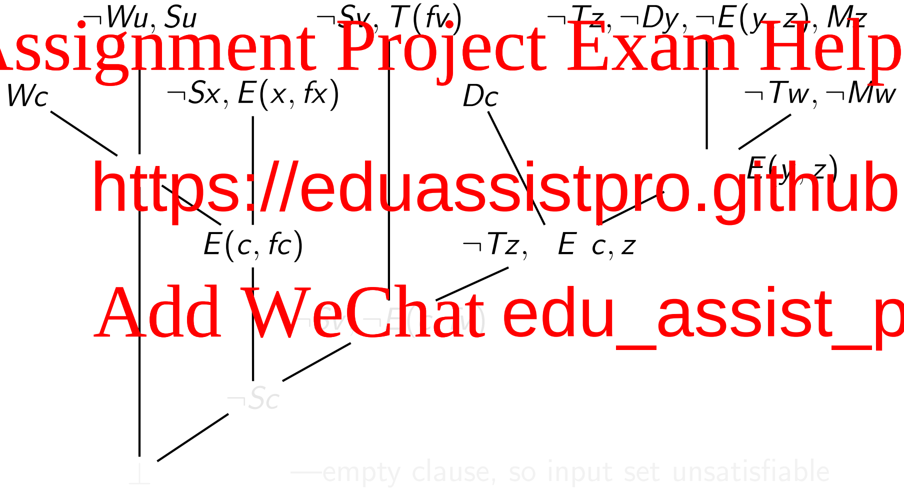


A Refutation for the Tadpole Example

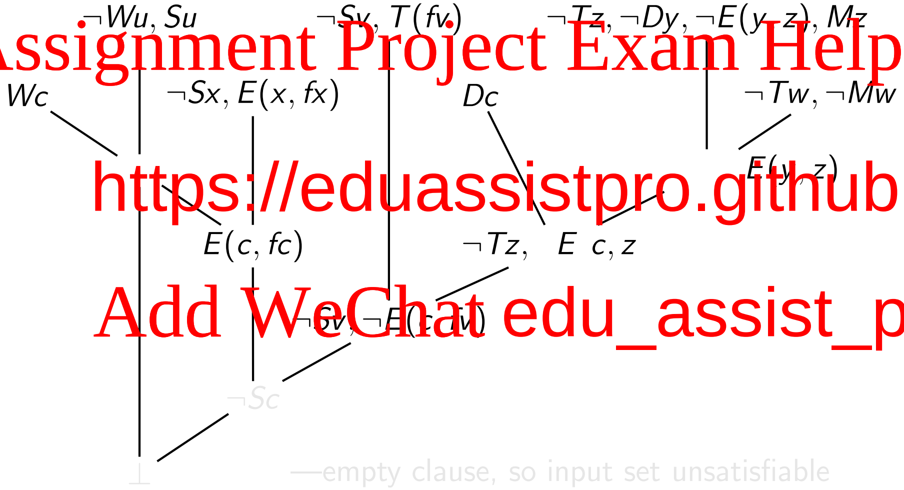


—empty clause, so input set unsatisfiable

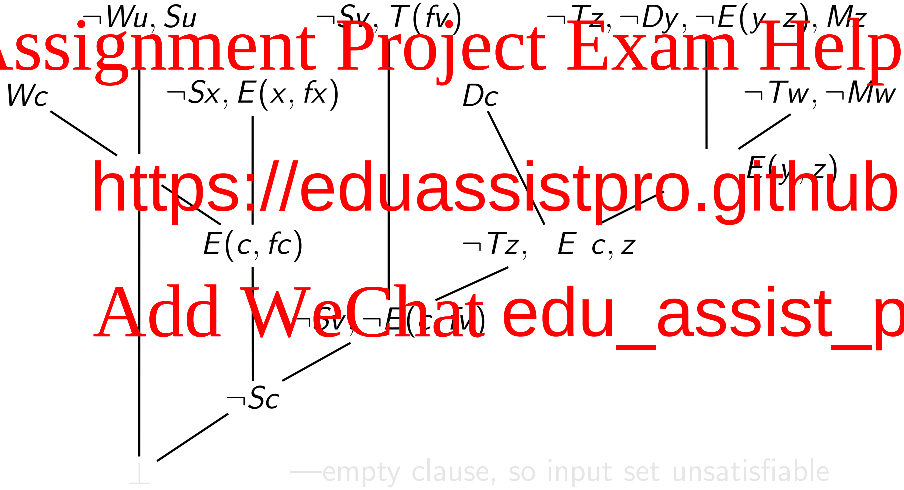
A Refutation for the Tadpole Example



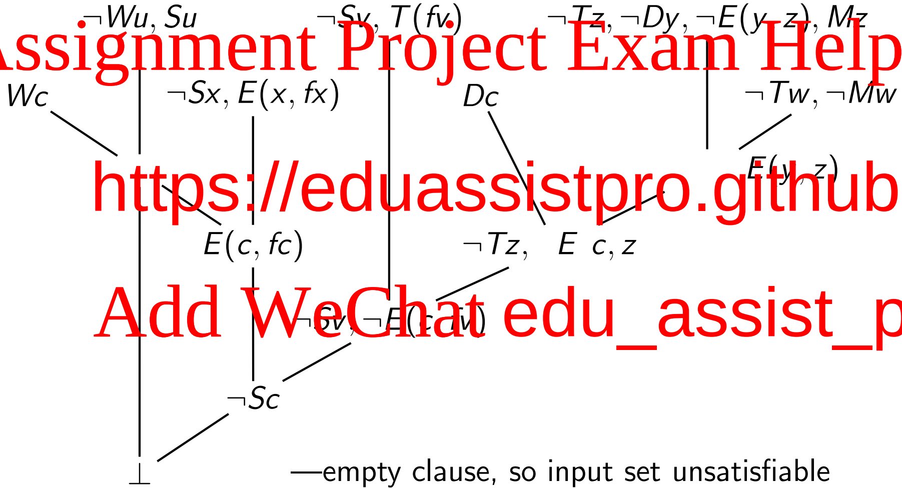
A Refutation for the Tadpole Example



A Refutation for the Tadpole Example



A Refutation for the Tadpole Example



Resolution Exercise

Using resolution, justify this argument:

- All philosophers are wise $\forall x\,(P(x) \Rightarrow W(x))$
- Some Greeks are philosophers $\exists x\,(G(x) \land P(x))$
- Th $(x))$

In addition to resolution, there is one more valid rewriting of clauses, called factoring.

Let C be a clause and let $A_1, A_2 \in C$. If A_1 and A_2 are unifiable with mgu θ, a

$\{D(f(y), y), \neg P(f(y$

$$\{D(f(y), y), \neg P(f(y))\}$$

The Need for Factoring

Factoring is sometimes crucial:

$$
\begin{array}{ccc}
\{P(x)\} & & \{\neg P(u)\} \\
 & \perp &
\end{array}
$$

How to Use Clauses

A resolution step uses two clauses (or two "copies" of the same clause). A factoring step uses one clause.

A given cla … art in many diff

But recal … ed.

Hence we really should rename all variable … e we use the clause, using fresh variable names.

Sometimes this renaming is essential for correctness, especially when resolution uses two "copies" of the same clause.

Leibniz's Dream

The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us calculate, without further ado, to see who is right.

[5]

Start wit

While

add t

or a resolvent of some $C_1, C_2 \in \mathcal{C}$

Does this process always terminate?

The Power of Resolution

Theorem. $\mathcal{C}$ is unsatisfiable iff the resolution method can add $\perp$ after a finite number of steps.

We say that resolution is sound and complete.

It gives us a

Note, however, that resolution does not give a decision procedure for unsatisfiability (or validity) of first-order predic .

Indeed, it can be shown that there are no such proof pro s.

We say that validity and unsatisfiability are semi-decidable) properties.

Proof Search

Resolution only works if we apply a sensible search strategy:

$\{Q\} \quad \{\neg Q\} \quad \{\neg P(x), P(f(x))\}$

$\}$

$\{\neg P(x), P(f($

$\vdots$

Moreover, search can be expensive.

* Here we used two copies of the same clause for resolution—the second using fresh variables, say $\{\neg P(y), P(f(y))\}$. When the resolvent is later used, it too is first renamed.

Proof Search Strategies

There is a chapter on resolution proofs on the LMS under “Readings Online”.

It covers di
it also has a lo
resolutio
(For the lat
Herbrand interpretation and semantic tree.)

These parts are not examinable.

Jacques Herbrand, 1908–1931

Horn Clauses and Prolog

A Horn clause is a clause with at most one positive literal.

All seven clauses used in the tadpole example were Horn.

A clause su e

thought

In the logic programming language Prolo


```
miserable(Z) :- tadpole(Z),
                deepwaterfish(Y),
                eats(Y,Z).
```

For logic p es the

search pr

For pro

time. (For arbitrary propositional CNF it is NP-co

Sidebar: Unification and Type Inference

Another important use of unification is in type checking/inference. Here variables are type variables and the function symbols are type constructors. For example, think of the list type constructor as 'List', so we write '`list(x)`' instead of the Haskell type '`[x]`', and let us write '


```
map  :: fu
null :: fun(list(z), bool)
```

If we use the expression `map null "abc"`
effectively sets up these equations:


```
fun(x,y) = fun(list(z), bool) -- for map null
list(x)  = list(char)         -- for (map null) "abc"
```

Sidebar: Unification and Type Inference

The unification algorithm then produces these equations:


```
x = list
y = bool
x = char
```


and hence

```
list(z) = char
```


Unification failure shows that `map null "abc"` is not well-typed.

Next Up

Next we'l

Next wee cepts
and results: sets, relations and functions.