Lecture 9 First-Order Logic Reasoning and Transformation to Clausal Form

COMP24412: Symbolic AI

Giles Reger

February 2019

Aim and Learning Outcomes

The aim of this lecture is to:

Look at the saturation-based setting and the first step of transforming to clausal form

Learning Outcomes

By the end of this lecture you will be able to:

1. Recall the saturation loop and the main steps in it
2. Describe the clausal transformation pipeline
3. Transform general first-order formulas into clausal form
4. Identify points where this transformation could be optimised
5. Recall how to run Vampire to perform resolution and clausification

Unification (Recap)

In first-order logic every two terms have a most general unifier

A substitution σ is a unifier for two terms t_1 and t_2 if $\sigma(t1) = \sigma(t_2)$

I avoid giving a very formal definition of more general but a unifier σ_1 is more general than σ_2 if $\sigma_2(t)$ is always an instance of $\sigma_1(t)$

There is a relatively straightforward algorithm for generating most general unifiers called Robinsons Algorithm (does the obvious thing) but in reality we compute unifiers using special data structures.

Clauses and Resolution (Recap)

A literal is an an atom or its negation. A clause is a disjunction of literals.

Clauses are implicitly universally quantified.

We can think of a clause as a conjunction implying a disjunction e.g.

$$(a_1 \wedge \ldots a_n) \rightarrow (b_1 \vee \ldots \vee b_m)$$

An empty clause is false.

Resolution works on clauses

$$\frac{l_1 \vee C \qquad \neg l_2 \vee D}{(C \vee D)\theta} \quad \theta = \mathrm{mgu}(l_1, l_2)$$

Refutational Based Reasoning

We are going to look at a reasoning method that works by refutation.

Recall $\Gamma \models \phi$ if and only if $\Gamma \cup \{\neg\phi\} \models$ *false*

If we want to show that ϕ is entailed by Γ we can show that $\Gamma \cup \{\neg\phi\}$ is inconsistent

Refutational Based Reasoning

We are going to look at a reasoning method that works by refutation.

Recall $\Gamma \models \phi$ if and only if $\Gamma \cup \{\neg\phi\} \models \mathit{false}$

If we want to show that ϕ is entailed by Γ we can show that $\Gamma \cup \{\neg\phi\}$ is inconsistent

This is refutational based reasoning.

We will saturate $\Gamma \cup \{\neg\phi\}$ until there is nothing left to add or we have derived *false*.

If we do not find *false* then $\Gamma \not\models \phi$.

There are some caveats we will meet later.

Resolving to false

$$\begin{array}{l} \neg rich(x) \lor happy(x) \\ rich(giles) \end{array} \models happy(giles)$$

Resolving to false

$$\neg rich(x) \lor happy(x)$$
$$rich(giles)$$
$$\neg happy(giles)$$

Resolving to false

$\neg rich(x) \vee happy(x)$
$rich(giles)$
$\neg happy(giles)$

Resolving to false

$\neg rich(x) \vee happy(x)$
$rich(giles)$
$\neg happy(giles)$
$\neg rich(giles)$

Resolving to false

$\neg rich(x) \vee happy(x)$
$rich(giles)$
$\neg happy(giles)$
$\neg rich(giles)$

Resolving to false

$\neg rich(x) \vee happy(x)$
$rich(giles)$
$\neg happy(giles)$
$\neg rich(giles)$
$false$

Resolving to false

$$\begin{array}{l} \neg rich(x) \vee happy(x) \\ rich(giles) \\ \neg happy(giles) \\ \neg rich(giles) \\ false \end{array}$$

We could have done it in the other order (picked $\neg rich(x)$ first). We'll find out later that it's better to organise proof search to avoid this redundancy.

What about Equality?

$$\begin{array}{ll} \neg rich(father(x)) \vee happy(x) & \\ rich(david) & \models happy(giles) \\ father(giles) = david & \end{array}$$

What about Equality?

$$\neg rich(father(x)) \vee happy(x)$$
$$rich(david)$$
$$father(giles) = david$$
$$\neg happy(giles)$$

What about Equality?

$$\neg rich(father(x)) \vee happy(x)$$
$$rich(david)$$
$$father(giles) = david$$
$$\neg happy(giles)$$

What about Equality?

$$\neg rich(father(x)) \vee happy(x)$$
$$rich(david)$$
$$father(giles) = david$$
$$\neg happy(giles)$$
$$\neg rich(father(giles))$$

What about Equality?

$$\neg rich(father(x)) \lor happy(x)$$
$$rich(david)$$
$$father(giles) = david$$
$$\neg happy(giles)$$
$$\neg rich(father(giles))$$

$father(x)$ and $david$ do not unify.

Paramodulation

The paramodulation rule lifts the idea behind resolution to equality

$$\frac{C \vee s = t \qquad\qquad l[u] \vee D}{(l[t] \vee C \vee D)\theta} \quad \theta = \mathrm{mgu}(s, u)$$

where u is not a variable.

Paramodulation

The paramodulation rule lifts the idea behind resolution to equality

$$\frac{C \vee s = t \qquad\qquad l[u] \vee D}{(l[t] \vee C \vee D)\theta} \quad \theta = \mathrm{mgu}(s, u)$$

where u is not a variable.

For example

$$\frac{\mathit{father}(\mathit{giles}) = \mathit{david} \qquad\qquad \neg \mathit{rich}(\mathit{father}(x)) \vee \mathit{happy}(x)}{\neg \mathit{rich}(\mathit{david}) \vee \mathit{happy}(\mathit{giles})}$$

where $u = \mathit{father}(x)$ and therefore $\theta = \{x \mapsto \mathit{giles}\}$.

Special case: unit equalities

The demodulation rule works with unit equalities

$$\frac{s = t \qquad l[u] \vee D}{(l[t] \vee D)\theta} \quad \theta = \text{matches}(s, u)$$

Note that u must be an instance of s.

Why is this special? The premise $l[u] \vee D$ becomes redundant. We'll find out what that means properly next time but it means we can remove it.

Notice that we could apply this in either direction - this is going to lead to redundancy and later we will see that we should (where possible) order equalities so that we rewrite more complicated things with simpler things.

Equality Resolution

We have another special case, is the following ever true?

$$\forall x.f(x) \neq f(a)$$

e.g. for every input to f the result is not equal to applying f to a.

Equality Resolution

We have another special case, is the following ever true?

$$\forall x. f(x) \neq f(a)$$

e.g. for every input to f the result is not equal to applying f to a.

Functions in first-order logic are always total

Equality Resolution

We have another special case, is the following ever true?

$$\forall x. f(x) \neq f(a)$$

e.g. for every input to f the result is not equal to applying f to a.

Functions in first-order logic are always total

So, no. We have a rule called equality resolution

$$\frac{s \neq t \vee C}{C\theta} \quad \theta = \mathrm{mgu}(s, t)$$

e.g. if we can make two terms equal then any disequality is false.

More Examples Using Vampire

Vampire is a first-order theorem prover

It works on Clauses and implements resolution (and lots of other rules)

It takes problems in either TPTP or STM-LIB format

```
cnf(one,axiom, ~rich(X) | happy(X)).
cnf(two,axiom, rich(giles)).
cnf(three, negated_conjecture, ~happy(giles)).
```

The above is TPTP format already in conjunctive normal form.

Saturation Based Reasoning

To decide $\Gamma \models \varphi$ we are going to follow the below steps

1. Let $NotDone = \Gamma \cup \{\neg\varphi\}$
2. Transform *NotDone* into clauses
3. Let *Done* be an empty set of clauses
4. Select a clause c from *NotDone*
5. Perform all inferences (e.g. resolution) between c and all clauses in *Done* putting any *new* children in *NotDone*
6. Move c to *Done*
7. If one of the children was *false* then **return valid**
8. If *NotDone* is empty stop otherwise go to 4
9. **return not valid**

Saturation Based Reasoning

To decide $\Gamma \models \varphi$ we are going to follow the below steps

1. Let $\mathit{NotDone} = \Gamma \cup \{\neg\varphi\}$
2. Transform *NotDone* into clauses
3. Let *Done* be an empty set of clauses
4. Select a clause c from *NotDone*
5. Perform all inferences (e.g. resolution) between c and all clauses in *Done* putting any *new* children in *NotDone*
6. Move c to *Done*
7. If one of the children was *false* then **return valid**
8. If *NotDone* is empty stop otherwise go to 4
9. **return not valid**

Transformation to Clausal Form

To go from general formula to set of clauses we're going to go through the following steps

1. Rectify the formula
2. Transform to *Negation Normal Form*
3. Eliminate quantifiers
4. Transform into conjunctive normal form

At any stage we might simplify using rules about *true* and *false*, e.g. $\mathit{false} \wedge \phi = \mathit{false}$, and remove tautologies, e.g. $p \vee p$.

Rectification

A formula is rectified if each quantifier binds a different variable and all bound variables are distinct from free ones.

To rectify a formula we identify any name clashes, pick one and rename it consistently. It is important to work out which variables are in the scope of a quantifier e.g.

$$\forall x.(p(x) \vee \exists x.r(x)) \qquad \textit{becomes} \qquad \forall x(p(x) \vee \exists y.r(y))$$

To automate this we typically rename bound variables starting with x_0 etc.

Negation Normal Form

Apply rules in a completely deterministic syntactically-guided way

$$
\begin{array}{rcl}
\neg(F_1 \wedge \ldots \wedge F_n) & \Rightarrow & \neg F_1 \vee \ldots \vee \neg F_n \\
\neg(F_1 \vee \ldots \vee F_n) & \Rightarrow & \neg F_1 \wedge \ldots \wedge \neg F_n \\
F_1 \rightarrow F_2 & \Rightarrow & \neg F_1 \vee F_2 \\
\neg\neg F & \Rightarrow & F \\
\neg\forall x_1, \ldots, x_n F & \Rightarrow & \exists x_1, \ldots, x_n \neg F \\
\neg\exists x_1, \ldots, x_n F & \Rightarrow & \forall x_1, \ldots, x_n \neg F \\
\neg(F_1 \leftrightarrow F_2) & \Rightarrow & F_1 \otimes F_2 \\
\neg(F_1 \otimes F_2) & \Rightarrow & F_1 \leftrightarrow F_2 \\
F_1 \leftrightarrow F_2 & \Rightarrow & (F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1); \\
F_1 \otimes F_2 & \Rightarrow & (F_1 \vee F_2) \wedge (\neg F_1 \vee \neg F_2).
\end{array}
$$

Where $\otimes$ is exclusive or. Can get an exponential increase in size.

Examples

$$(\forall x.p(x)) \rightarrow (\exists x.p(x))$$

$$(\forall x.(p(x) \vee q(x)) \leftrightarrow (\neg\exists x.(\neg p(x) \wedge \neg q(x)))$$

$$\forall x, y, z.((f(x) = y \wedge f(x) = z) \rightarrow y = z)$$

$$\forall x.((\exists y.p(x, y)) \rightarrow q(x)) \wedge p(a, b) \wedge \neg\exists x.q(x)$$

Also, which of the above statements are valid or inconsistent?

Dealing with Existential Quantifiers

There is a best kind of pizza
$\exists x.\forall y.((pizza(x) \wedge pizza(y) \wedge x \neq y) \rightarrow better(x, y))$

There are two different people who live in the same house
$\exists x.\exists y.\exists z.(x \neq y \wedge lives\_in(x, z) \wedge lives\_in(y, z))$

Everybody loves somebody
$\forall x.\exists y.loves(x, y)$

Dealing with Existential Quantifiers

There is a best kind of pizza
$\exists x.\forall y.((pizza(x) \land pizza(y) \land x \neq y) \rightarrow better(x, y))$

There are two different people who live in the same house
$\exists x.\exists y.\exists z.(x \neq y \land lives\_in(x, z) \land lives\_in(y, z))$

Everybody loves somebody
$\forall x.\exists y.loves(x, y)$

Let $\mathcal{I}$ be some interpretation. If $\mathcal{I}(\exists x.\phi[x])$ is true then there must be some domain constant d such that $\mathcal{I}(\phi[d])$ is true.

Dealing with Existential Quantifiers

There is a best kind of pizza
$\exists x.\forall y.((pizza(x) \wedge pizza(y) \wedge x \neq y) \rightarrow better(x, y))$

There are two different people who live in the same house
$\exists x.\exists y.\exists z.(x \neq y \wedge lives\_in(x, z) \wedge lives\_in(y, z))$

Everybody loves somebody
$\forall x.\exists y.loves(x, y)$

Let $\mathcal{I}$ be some interpretation. If $\mathcal{I}(\exists x.\phi[x])$ is true then there must be some domain constant d such that $\mathcal{I}(\phi[d])$ is true.

Let a be a fresh constant symbol (a new name for the object that has to exist). As it is fresh we can freely let $\mathcal{I}(a) = d$.
(This requires the Axiom of Choice)

Dealing with Existential Quantifiers

There is a best kind of pizza
$\exists x.\forall y.((\mathit{pizza}(x) \land \mathit{pizza}(y) \land x \neq y) \rightarrow \mathit{better}(x, y))$

There are two different people who live in the same house
$\exists x.\exists y.\exists z.(x \neq y \land \mathit{lives\_in}(x, z) \land \mathit{lives\_in}(y, z))$

Everybody loves somebody
$\forall x.\exists y.\mathit{loves}(x, y)$

Let $\mathcal{I}$ be some interpretation. If $\mathcal{I}(\exists x.\phi[x])$ is true then there must be some domain constant d such that $\mathcal{I}(\phi[d])$ is true.

Let a be a fresh constant symbol (a new name for the object that has to exist). As it is fresh we can freely let $\mathcal{I}(a) = d$.
(This requires the Axiom of Choice)

Dealing with Existential Quantifiers

There is a best kind of pizza
$\exists x.\forall y.(\neg pizza(x) \vee \neg pizza(y) \vee x = y \vee better(x, y))$

There are two different people who live in the same house
$\exists x.\exists y.\exists z.(x \neq y \wedge lives\_in(x, z) \wedge lives\_in(y, z))$

Everybody loves somebody
$\forall x.\exists y.loves(x, y)$

Let $\mathcal{I}$ be some interpretation. If $\mathcal{I}(\exists x.\phi[x])$ is true then there must be some domain constant d such that $\mathcal{I}(\phi[d])$ is true.

Let a be a fresh constant symbol (a new name for the object that has to exist). As it is fresh we can freely let $\mathcal{I}(a) = d$.
(This requires the Axiom of Choice)

Dealing with Existential Quantifiers

There is a best kind of pizza
$\forall y.(\neg pizza(a) \vee \neg pizza(y) \vee a = y \vee better(a, y))$

There are two different people who live in the same house
$\exists x.\exists y.\exists z.(x \neq y \wedge lives\_in(x, z) \wedge lives\_in(y, z))$

Everybody loves somebody
$\forall x.\exists y.loves(x, y)$

Let $\mathcal{I}$ be some interpretation. If $\mathcal{I}(\exists x.\phi[x])$ is true then there must be some domain constant d such that $\mathcal{I}(\phi[d])$ is true.

Let a be a fresh constant symbol (a new name for the object that has to exist). As it is fresh we can freely let $\mathcal{I}(a) = d$.
(This requires the Axiom of Choice)

Dealing with Existential Quantifiers

There is a best kind of pizza
$\forall y.(\neg pizza(a) \vee \neg pizza(y) \vee a = y \vee better(a, y))$

There are two different people who live in the same house
$\exists x.\exists y.\exists z.(x \neq y \wedge lives\_in(x, z) \wedge lives\_in(y, z))$

Everybody loves somebody
$\forall x.\exists y.loves(x, y)$

Let $\mathcal{I}$ be some interpretation. If $\mathcal{I}(\exists x.\phi[x])$ is true then there must be some domain constant d such that $\mathcal{I}(\phi[d])$ is true.

Let a be a fresh constant symbol (a new name for the object that has to exist). As it is fresh we can freely let $\mathcal{I}(a) = d$.
(This requires the Axiom of Choice)

Dealing with Existential Quantifiers

There is a best kind of pizza
$\forall y.(\neg pizza(a) \vee \neg pizza(y) \vee a = y \vee better(a, y))$

There are two different people who live in the same house
$a \neq b \wedge lives\_in(a, c) \wedge lives\_in(b, c)$

Everybody loves somebody
$\forall x.\exists y.loves(x, y)$

Let $\mathcal{I}$ be some interpretation. If $\mathcal{I}(\exists x.\phi[x])$ is true then there must be some domain constant d such that $\mathcal{I}(\phi[d])$ is true.

Let a be a fresh constant symbol (a new name for the object that has to exist). As it is fresh we can freely let $\mathcal{I}(a) = d$.
(This requires the Axiom of Choice)

Dealing with Existential Quantifiers

There is a best kind of pizza
$\forall y.(\neg pizza(a) \vee \neg pizza(y) \vee a = y \vee better(a, y))$

There are two different people who live in the same house
$a \neq b \wedge lives\_in(a, c) \wedge lives\_in(b, c)$

Everybody loves somebody
$\forall x.\exists y.loves(x, y)$

Dealing with Existential Quantifiers

There is a best kind of pizza
$\forall y.(\neg pizza(a) \vee \neg pizza(y) \vee a = y \vee better(a, y))$

There are two different people who live in the same house
$a \neq b \wedge lives\_in(a, c) \wedge lives\_in(b, c)$

Everybody loves somebody
$\forall x.\exists y.loves(x, y)$

Let $\mathcal{I}$ be some interpretation. If $\mathcal{I}(\forall x.\exists y.\phi[x, y])$ is true then for every domain constant d_1 there must be some other domain constant d_2 such that $\mathcal{I}(\phi[d_1, d_2])$ is true. But how do we find d_2 given d_1?

Let f be a fresh function symbol whose interpretation can be made to *select* the necessary domain constant.

Dealing with Existential Quantifiers

There is a best kind of pizza
$\forall y.(\neg pizza(a) \vee \neg pizza(y) \vee a = y \vee better(a, y))$

There are two different people who live in the same house
$a \neq b \wedge lives\_in(a, c) \wedge lives\_in(b, c)$

Everybody loves somebody
$\forall x.\exists y.loves(x, y)$

Let $\mathcal{I}$ be some interpretation. If $\mathcal{I}(\forall x.\exists y.\phi[x, y])$ is true then for every domain constant d_1 there must be some other domain constant d_2 such that $\mathcal{I}(\phi[d_1, d_2])$ is true. But how do we find d_2 given d_1?

Let f be a fresh function symbol whose interpretation can be made to *select* the necessary domain constant.

Dealing with Existential Quantifiers

There is a best kind of pizza
$\forall y.(\neg pizza(a) \vee \neg pizza(y) \vee a = y \vee better(a, y))$

There are two different people who live in the same house
$a \neq b \wedge lives\_in(a, c) \wedge lives\_in(b, c)$

Everybody loves somebody
$\forall x.loves(x, f(x))$

Let $\mathcal{I}$ be some interpretation. If $\mathcal{I}(\forall x.\exists y.\phi[x, y])$ is true then for every domain constant d_1 there must be some other domain constant d_2 such that $\mathcal{I}(\phi[d_1, d_2])$ is true. But how do we find d_2 given d_1?

Let f be a fresh function symbol whose interpretation can be made to *select* the necessary domain constant.

Dealing with Universal Quantifiers

Forget about them

Skolemisation

This process is called Skolemisation and the new symbols we introduce are called Skolem constants or Skolem functions.

The rules are simply

$$
\begin{aligned}
\forall x_1, \ldots, x_n F &\Rightarrow F \\
\exists x_1, \ldots, x_n F &\Rightarrow F\{x_1 \mapsto f_1(y_1, \ldots, y_m), \ldots, x_n \mapsto f_n(y_1, \ldots, y_m)\},
\end{aligned}
$$

where f_i are fresh function symbols of the correct arrity and $y_1, \ldots y_m$ are the free variables of F e.g. they are the things universally quantified in the larger scope.

Remember that Skolem constants/functions act as witnesses for something that we know has to exist for the formula to be true.

Validity or Satisfiability Preserving?

Do these two formulas have the same models?

$$\exists x.\forall y.p(x,y) \qquad \forall y.p(a,y)$$

Validity or Satisfiability Preserving?

Do these two formulas have the same models?

$$\exists x.\forall y.p(x,y) \qquad \forall y.p(a,y)$$

No - the first one does not need to interpret a

Validity or Satisfiability Preserving?

Do these two formulas have the same models?

$$\exists x. \forall y. p(x, y) \qquad \forall y. p(a, y)$$

No - the first one does not need to interpret a

Do the formulas both have models?

Validity or Satisfiability Preserving?

Do these two formulas have the same models?

$$\exists x.\forall y.p(x, y) \qquad \forall y.p(a, y)$$

No - the first one does not need to interpret a

Do the formulas both have models?

Yes - there's no negation, just make everything true

Validity or Satisfiability Preserving?

Do these two formulas have the same models?

$$\exists x.\forall y.p(x, y) \qquad \forall y.p(a, y)$$

No - the first one does not need to interpret a

Do the formulas both have models?

Yes - there's no negation, just make everything true

When we introduce new symbols we preserve satisfiability (the existence of models) not validity (the same models).

However, most transformations are stronger than this and preserve models on the initial signature.

CNF Transformation

Now the only connectives left should be $\wedge$ and $\vee$ and we need to push the $\vee$ symbols under the $\wedge$ symbols

The associated rule is

$$
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n \quad \Rightarrow \quad
\begin{array}{cc}
(A_1 \vee B_1 \vee \ldots \vee B_n) & \wedge \\
\cdots & \wedge \\
(A_m \vee B_1 \vee \ldots \vee B_n). &
\end{array}
$$

This can lead to exponential growth.

Looking at Vampire's clausification

Run

```
./vampire --mode clausify problem
```

on problem file `problem`

It will sometimes do a lot more than what we've discussed above.

Vampire performs lots of optimisations e.g. naming subformulas, removing pure symbols, or unused definitions.

Examples

$$(\forall x.p(x)) \rightarrow (\exists x.p(x))$$

$$(\forall x.(p(x) \vee q(x)) \leftrightarrow (\neg \exists x.(\neg p(x) \wedge \neg q(x)))$$

$$\forall x, y, z.((f(x) = y \wedge f(x) = z) \rightarrow y = z)$$

$$\forall x.((\exists y.p(x, y)) \rightarrow q(x)) \wedge p(a, b) \wedge \neg \exists x.q(x)$$

Also, which of the above statements are valid or inconsistent?

Optimisation (subformula naming)

To go from general formula to set of clauses we're going to go through the following steps

1. Rectify the formula
2. Transform to *Equivalence Negation Form*
3. Apply *naming* of subformulas
4. Transform to *Negation Normal Form*
5. Eliminate quantifiers
6. Transform into conjunctive normal form

Equivalence Negation Form

Push negations in but preserve equivalences

$$
\begin{array}{rcl}
\neg(F_1 \wedge \ldots \wedge F_n) & \Rightarrow & \neg F_1 \vee \ldots \vee \neg F_n \\
\neg(F_1 \vee \ldots \vee F_n) & \Rightarrow & \neg F_1 \wedge \ldots \wedge \neg F_n \\
F_1 \rightarrow F_2 & \Rightarrow & \neg F_1 \vee F_2 \\
\neg\neg F & \Rightarrow & F \\
\neg\forall x_1, \ldots, x_n F & \Rightarrow & \exists x_1, \ldots, x_n \neg F \\
\neg\exists x_1, \ldots, x_n F & \Rightarrow & \forall x_1, \ldots, x_n \neg F \\
\neg(F_1 \leftrightarrow F_2) & \Rightarrow & F_1 \otimes F_2 \\
\neg(F_1 \otimes F_2) & \Rightarrow & F_1 \leftrightarrow F_2
\end{array}
$$

Only get a linear increase in size.

Subformula Naming

We want to get to a conjunction of disjunctions but this process can 'blow up' in general e.g.

$$p(x,y) \leftrightarrow (q(x) \leftrightarrow (p(y,y) \leftrightarrow q(y))),$$

is equivalent to

$$\begin{gathered}
p(x,y) \vee \neg q(y) \vee \neg p(y,y) \vee \neg q(x)) \\
p(x,y) \vee q(y) \vee p(y,y) \vee \neg q(x)) \\
p(x,y) \vee p(y,y) \vee \neg q(y) \vee q(x)) \\
p(x,y) \vee q(y) \vee \neg p(y,y) \vee q(x)) \\
q(x) \vee \neg q(y) \vee \neg p(y,y) \vee \neg p(x,y)) \\
q(x) \vee q(y) \vee p(y,y) \vee \neg p(x,y)) \\
p(y,y) \vee \neg q(y) \vee \neg q(x) \vee \neg p(x,y)) \\
q(y) \vee \neg p(y,y) \vee \neg q(x) \vee \neg p(x,y))
\end{gathered}$$

Subformula Naming

We can replace

$$p(x,y) \leftrightarrow (q(x) \leftrightarrow (p(y,y) \leftrightarrow q(y))),$$

by

$$\begin{array}{c} p(x,y) \leftrightarrow (q(x) \leftrightarrow n(y)); \\ n(y) \leftrightarrow (p(y,y) \leftrightarrow q(y)). \end{array}$$

to get the same number of clauses but each clause is simpler (better for reasoning).

In the case when the subformula $F(x_1,\ldots,x_k)$ has only positive occurrences in G, one can use the axiom $n(x_1,\ldots,x_k) \rightarrow F(x_1,\ldots,x_k)$ instead of $n(x_1,\ldots,x_k) \leftrightarrow F(x_1,\ldots,x_k)$. This will lead to fewer clauses.

Subformula Naming

Assigning a name n to $F_2 \leftrightarrow F_3$ yields two formulas

$$\begin{array}{l} F_1 \leftrightarrow n; \\ n \leftrightarrow (F_2 \leftrightarrow F_3), \end{array}$$

where the second formula has the same structure as the original formula $F_1 \leftrightarrow (F_2 \leftrightarrow F_3)$.

When to Name Subformulas?

Vampire uses a heuristic that that estimates how many clauses a subformula will produce and names that subformula if that number is above a certain threshold.

Naming can not increase the number of clauses introduced but does not always reduce.

The idea is the same as the optimised structural transformation in the propositional case but we don't always apply it as the cost on reasoning is much higher here.