4CCS1ELA ELEMENTARY LOGIC WITH APPLICATIONS

REVISION LECTURE – PROGRAMMING WITH LOGIC

What we have covered = what you need to know

1. Propositional definite clause programming
   – Prop. formula -> CNF -> Definite Clauses -> Definite Rules -> (tree)
2. Predicate definite clause programming
   – Pred. formula -> PNF -> CNF -> Definite Clauses -> Definite Rules -> (tree)
3. Logic programming
   – Trees with backtracking, negation as failure and recursion

Propositional definite clause programming

Exercise (Jan. 2016)

Transform the formula $P \rightarrow Q \wedge R$ into propositional definite rules

Prop. formula -> CNF -> Definite Clauses -> Definite Rules

1. transform it into CNF using equivalence laws

$P \rightarrow Q \wedge R$

$\neg P \vee (Q \wedge R)$

$(\neg P \vee Q) \wedge (\neg P \vee R)$

2. identify definite clauses. Not all clauses are definite clauses (a definite clause has exactly one positive literal)!

$(\neg P \vee Q) \wedge (\neg P \vee R)$

$(\neg P \vee Q)$ is a definite clause

$(\neg P \vee R)$ is a definite clause

3. we represent each of the definite clauses as definite rules.

$(\neg P \vee Q)$ can be represented as rule $\boldsymbol{P \rightarrow Q}$

$(\neg P \vee R)$ can be represented as rule $\boldsymbol{P \rightarrow R}$

Predicate definite clause programming

Pred. formula -> PNF -> CNF -> Definite Clauses -> Definite Rules

1) Transform predicate formula into **PNF** formula F

2) Transform matrix of F to **CNF**

3) If

   i) every quantifier in prefix is **universal**

   ii) every clause in CNF of matrix contains **exactly one positive atom**, **0 or more negative atoms** and all variables in the clause are in the scope of a universal quantifier in the prefix (which means that each clause is a **definite clause**)

Then represent each clause as a **definite rule**

Exercise (Aug. 2016)

Transform the formula into definite rules:

$(\exists y\ P(y) \land \neg\forall z\ R(z)) \rightarrow \neg\exists y\ S(y)$

Algorithm to **transform a formula into PNF**

1. Remove $\rightarrow$ and $\leftrightarrow$
2. Move negations inward

   - such that, in the end, negations only appear in front of atoms.
3. Rename variables so that variables of each quantifier are unique, if necessary (this is called standardisation).
4. Move quantifiers to the front of the formula

   - preserving order of quantifiers as they appear in formula.

1) Formula to PNF

$(\exists y\ P(y) \land \neg\forall z\ R(z)) \rightarrow \neg\exists y\ S(y)$

$\neg(\exists y\ P(y) \land \neg\forall z\ R(z)) \lor \neg\exists y\ S(y)$

$(\neg\exists y\ P(y) \lor \neg\neg\forall z\ R(z)) \lor \neg\exists y\ S(y)$

$\neg\exists y\ P(y) \lor \neg\neg\forall z\ R(z) \lor \neg\exists y\ S(y)$

$\neg\exists y\ P(y) \lor \forall z\ R(z) \lor \neg\exists y\ S(y)$

$\forall y\ \neg P(y) \lor \forall z\ R(z) \lor \forall y\ \neg S(y)$

$\forall y\ \neg P(y) \lor \forall z\ R(z) \lor \forall w\ \neg S(w)$

$\forall y\ \forall z\ \forall w\ \neg P(y) \lor R(z) \lor \neg S(w)$

2) PNF to CNF

$\forall y\ \forall z\ \forall w\ \mathbf{\neg P(y) \lor R(z) \lor \neg S(w)}$

- Matrix is already in CNF

3) CNF to Definite Clause

- $\forall y \; \forall z \; \forall w \; \neg \mathbf{P(y)} \vee \mathbf{R(z)} \vee \neg \mathbf{S(w)}$

- One definite clause

Definite Clauses to **Definite RULES**

- **A definite clause:**

$$\forall x_1,\ldots,\forall x_n \ \neg\alpha_1 \vee \ldots \vee \neg\alpha_m \vee \alpha$$

- **Represented as definite rule:**

$$\forall x_1,\ldots,\forall x_n \ \alpha_1 \wedge \ldots \wedge \alpha_m \rightarrow \alpha$$

- Now drop ∧:

$$\forall x_1,\ldots,\forall x_n \ \alpha_1 ,\ldots , \alpha_m \rightarrow \alpha$$

4) Definite Clause to definite rules

$\forall y\ \forall z\ \forall w\ \neg \mathbf{P(y)} \lor \mathbf{R(z)} \lor \neg \mathbf{S(w)}$

- Can be represented as the rule

$\forall y\ \forall z\ \forall w\ \mathbf{P(y)} \land \mathbf{S(w)} \rightarrow \mathbf{R(z)}$

$\forall y\ \forall z\ \forall w\ \mathbf{P(y)}\ ,\ \mathbf{S(w)} \rightarrow \mathbf{R(z)}$

Logic programming

Example

1) *bird(x)*, not *can_fly(x)* → *flightless_bird(x)*
2) *bird(eagle)*
3) *can_fly(eagle)*
4) *bird(chicken)*

$\mathscr{P}_1$

every attempt to prove *can_fly(chicken)* failed (that is, there is **no successful derivation tree for** *can_fly(chicken)*). So **not** ***can_fly(chicken)*** must be true.

? *flightless_bird (chicken)*

1) { (x/chicken) }

? *bird(chicken)*, not *can_fly(chicken)*

4)

? not *can_fly(chicken)*

? *can_fly(chicken)*

Exercise (Jan. 2016)

Using negation as failure, specify the following as a predicate logic program:

1) if x is an enemy of z and z is an enemy of y, x is a friend of y.
2) if x is at war with y, x is an enemy of y
3) if x is a trade rival of y and x does **not** have a peace treaty with y, x is an enemy of y.
4) winterfell is at war with kings_landing.
5) kings_landing is a trade rival with braavos.

Exercise (Jan. 2016)

1) if x is an enemy of z and z is an enemy of y, x is a friend of y.

enemy(x,z), enemy (z,y) → friend (x,y)

2) if x is at war with y, x is an enemy of y

war(x,y) → enemy (x,y)

3) if x is a trade rival of y and x does not have a peace treaty with y, x is an enemy of y.

rival (x,y), **not** *treaty (x,y) → enemy (x,y)*

3) winterfell is at war with kings_landing.

war(winterfell, kings_landing)

4) kings_landing is a trade rival with braavos.

rival(kings_landing, braavos)

Try a Query

1) *enemy(x,z), enemy (z,y) → friend (x,y)*
2) *war(x,y) → enemy (x,y)*
3) *rival (x,y) ,* **not** *treaty (x,y) → enemy (x,y)*
4) *war(winterfell, kings_landing)*
5) *rival(kings_landing, braavos)*

$\mathscr{P}_1$

? friend(winterfell, braavos)

1) { (x/winterfell), (y/braavos)}

? enemy(winterfell,z), enemy(z,braavos)

2) { (x/winterfell), (y/z)}

? war(winterfell,z), enemy(z,braavos)

4) { (z/kings_landing)}

? enemy(kings_landing,braavos)

2) { (x/kings_landing), (y/braavos)}

? war(kings_landing,braavos)

■

Failure! Backtrack to last choice point

1) *enemy(x,z), enemy (z,y) → friend (x,y)*
2) *war(x,y) → enemy (x,y)*
3) *rival (x,y) ,* **not** *treaty (x,y) → enemy (x,y)*
4) *war(winterfell, kings_landing)*
5) *rival(kings_landing, braavos)*

$\mathcal{P}1$

every attempt to prove *treaty(k_lan,br)* failed (that is, there is **no successful derivation tree for** *treaty(k_lan,br))*. So **not** *treaty(k_lan,br)* must be true.

? friend(winterfell, braavos)

1) { (x/winterfell), (y/braavos)}

? enemy(winterfell,z), enemy(z,braavos)

2) { (x/winterfell), (y/z)}

? war(winterfell,z), enemy(z,braavos)

4) { (z/kings_landing)}

? enemy(kings_landing,braavos)

3) { (x/kings_landing), (y/braavos)}

? rival(kings_landing,braavos) , **not** *treaty(k_lan,br)*

5)

? **not** *treaty(k_lan,br)*

? treaty (k_lan,br)

Good Luck!