Lógica para Computação

Forma Normal Conjuntiva

Thiago Alves Rocha

thiagoalvesifce@gmail.com

Tópicos

Tópicos

Introdução

- Vamos definir uma formato padrão de fórmulas
- Deve ser possível representar qualquer fórmula no formato
- Algoritmos podem assumir que as fórmulas estão no formato
- As fórmulas podem ser convertidas ou dadas no formato

Forma Normal Conjuntiva

- Primeiro vamos conhecer a forma normal conjuntiva (FNC)
- Chamada de Conjunctive Normal Form (CNF) em inglês
- Também chamada de forma normal clausal

Tópicos

Cláusulas

- O elemento básico da CNF é o literal
- Um literal L é uma fórmula atômica p ou negação de atômica $\neg p$
- Um literal da forma p é chamado de positivo
- Um literal da forma $\neg p$ é chamado de negativo

Definição

Uma cláusula é uma disjunção de literais $L_1 \vee \ldots \vee L_n$ em que n é o tamanho da cláusula.

- $(\neg p \vee r)$ é cláusula
- $(\neg q \vee p \vee r)$ é cláusula

Cláusulas

- Se $n = 1$ dizemos que a cláusula é unitária
- Uma cláusula também pode ser representada por um conjunto $\{L_1, ..., L_n\}$.
- Uma cláusula da forma $\neg q_1 \vee ... \vee \neg q_k \vee p_1 \vee ... \vee p_l$ é equivalente a $(q_1 \wedge ... \wedge q_k) \rightarrow (p_1 \vee ... \vee p_l)$

Forma Normal Conjuntiva

Definição

Uma fórmula ϕ está na forma normal conjuntiva ou forma normal clausal se for uma conjunção de cláusulas $\phi = C_1 \wedge ... \wedge C_m$.

Exemplos

Definição

Uma fórmula ϕ está na forma normal conjuntiva ou forma normal clausal se for uma conjunção de cláusulas $\phi = C_1 \wedge \ldots \wedge C_m$.

- $(\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$ está na CNF
- $(p \vee r) \wedge (\neg p \vee r) \wedge (p \vee \neg r)$ está na CNF
- $(\neg(q \vee p) \vee r) \wedge (q \vee r)$ não está na CNF

Tópicos

Conversão para CNF

- Qualquer fórmula tem uma equivalente na CNF
- A prova é feita através de um algoritmo de conversão para CNF

Conversão para CNF

- A ideia do algoritmo é usar equivalências lógicas para modificar a fórmula ϕ de entrada
- Isso vai garantir que a fórmula resultante ϕ' seja equivalente à fórmula de entrada ϕ
- Primeiro retiramos todas as implicações usando a equivalência $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$
- Veja que este procedimento deve ser recursivo pois φ e ψ também podem possuir $\rightarrow$

Conversão para CNF

function IMPL_FREE (ϕ) :
/* computes a formula without implication equivalent to ϕ */
begin function
  case
    ϕ is an atom: **return** ϕ
    ϕ is $\neg\phi_1$: **return** $\neg$IMPL_FREE (ϕ_1)
    ϕ is $\phi_1 \wedge \phi_2$: **return** IMPL_FREE (ϕ_1) $\wedge$ IMPL_FREE (ϕ_2)
    ϕ is $\phi_1 \vee \phi_2$: **return** IMPL_FREE (ϕ_1) $\vee$ IMPL_FREE (ϕ_2)
    ϕ is $\phi_1 \rightarrow \phi_2$: **return** IMPL_FREE ($\neg\phi_1 \vee \phi_2$)
  end case
end function

Conversão para CNF

```
function IMPL_FREE (φ) :
/* computes a formula without implication equivalent to φ */
begin function
  case
    φ is an atom: return φ
    φ is ¬φ1: return ¬IMPL_FREE (φ1)
    φ is φ1 ∧ φ2: return IMPL_FREE (φ1) ∧ IMPL_FREE (φ2)
    φ is φ1 ∨ φ2: return IMPL_FREE (φ1) ∨ IMPL_FREE (φ2)
    φ is φ1 → φ2: return IMPL_FREE (¬φ1 ∨ φ2)
  end case
end function
```

- Executar $IMPL\_FREE((\neg p \wedge q) \rightarrow (p \wedge (r \rightarrow q)))$

Conversão para CNF

- Depois eliminamos a dupla negação com a equivalência $\neg\neg\varphi \equiv \varphi$
- E internalizamos as negações através das equivalências $\neg(\varphi \land \psi) \equiv \neg\varphi \lor \neg\psi$ e $\neg(\varphi \lor \psi) \equiv \neg\varphi \land \neg\psi$

Conversão para CNF

function NNF (ϕ):

/* precondition: ϕ is implication free */

/* postcondition: NNF (ϕ) computes a NNF for ϕ */

begin function

  case

    ϕ is a literal: **return** ϕ

    ϕ is $\neg\neg\phi_1$: **return** NNF (ϕ_1)

    ϕ is $\phi_1 \wedge \phi_2$: **return** NNF (ϕ_1) $\wedge$ NNF (ϕ_2)

    ϕ is $\phi_1 \vee \phi_2$: **return** NNF (ϕ_1) $\vee$ NNF (ϕ_2)

    ϕ is $\neg(\phi_1 \wedge \phi_2)$: **return** NNF ($\neg\phi_1$) $\vee$ NNF ($\neg\phi_2$)

    ϕ is $\neg(\phi_1 \vee \phi_2)$: **return** NNF ($\neg\phi_1$) $\wedge$ NNF ($\neg\phi_2$)

  end case

end function

```
function NNF(φ):
/* precondition: φ is implication free */
/* postcondition: NNF(φ) computes a NNF for φ */
begin function
  case
    φ is a literal: return φ
    φ is ¬¬φ1 : return NNF(φ1)
    φ is φ1 ∧ φ2 : return NNF(φ1) ∧ NNF(φ2)
    φ is φ1 ∨ φ2 : return NNF(φ1) ∨ NNF(φ2)
    φ is ¬(φ1 ∧ φ2): return NNF(¬φ1) ∨ NNF(¬φ2)
    φ is ¬(φ1 ∨ φ2): return NNF(¬φ1) ∧ NNF(¬φ2)
  end case
end function
```

- Executar $NNF(\neg(\neg p \wedge q) \vee (p \wedge (\neg r \vee q)))$

Conversão para CNF

- Depois aplicamos a equivalência $\varphi_1 \vee (\varphi_2 \wedge \varphi_3) \equiv (\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3)$ para colocar os conectivos $\vee$ para dentro das cláusulas e os conectivos $\wedge$ para fora

Conversão para CNF

function $\texttt{CNF}\,(\phi)$:

/* precondition: ϕ implication free and in NNF */

/* postcondition: $\texttt{CNF}\,(\phi)$ computes an equivalent CNF for ϕ */

begin function

  case

    ϕ is a literal: **return** ϕ

    ϕ is $\phi_1 \wedge \phi_2$: **return** $\texttt{CNF}\,(\phi_1) \wedge \texttt{CNF}\,(\phi_2)$

    ϕ is $\phi_1 \vee \phi_2$: **return** $\texttt{DISTR}\,(\texttt{CNF}\,(\phi_1), \texttt{CNF}\,(\phi_2))$

  end case

end function

Conversão para CNF

function $\texttt{DISTR}(\eta_1, \eta_2)$:
/* precondition: η_1 and η_2 are in CNF */
/* postcondition: $\texttt{DISTR}(\eta_1, \eta_2)$ computes a CNF for $\eta_1 \vee \eta_2$ */
begin function
  case
    η_1 is $\eta_{11} \wedge \eta_{12}$: **return** $\texttt{DISTR}(\eta_{11}, \eta_2) \wedge \texttt{DISTR}(\eta_{12}, \eta_2)$
    η_2 is $\eta_{21} \wedge \eta_{22}$: **return** $\texttt{DISTR}(\eta_1, \eta_{21}) \wedge \texttt{DISTR}(\eta_1, \eta_{22})$
    otherwise (= no conjunctions): **return** $\eta_1 \vee \eta_2$
  end case
end function

Conversão para CNF

function $\texttt{DISTR}\,(\eta_1, \eta_2)$:
/* precondition: η_1 and η_2 are in CNF */
/* postcondition: $\texttt{DISTR}\,(\eta_1, \eta_2)$ computes a CNF for $\eta_1 \vee \eta_2$ */
begin function
  case
    η_1 is $\eta_{11} \wedge \eta_{12}$: **return** $\texttt{DISTR}\,(\eta_{11}, \eta_2) \wedge \texttt{DISTR}\,(\eta_{12}, \eta_2)$
    η_2 is $\eta_{21} \wedge \eta_{22}$: **return** $\texttt{DISTR}\,(\eta_1, \eta_{21}) \wedge \texttt{DISTR}\,(\eta_1, \eta_{22})$
    otherwise (= no conjunctions): **return** $\eta_1 \vee \eta_2$
  end case
end function

- Executar $CNF((p \vee \neg q) \vee (p \wedge (\neg r \vee q)))$

Exemplos

- Converter $r \rightarrow (s \rightarrow ((t \land s) \rightarrow r))$ para CNF