Um algoritmo baseado em programação dinâmica e renomeamento para minimização de formas normais

Matheus Pimenta

Universidade de Brasília

2016

Conteúdo

Conteúdo

Lógica

- Lógicas são utilizadas para representar e raciocinar sobre problemas computacionais.

Lógica

- Lógicas são utilizadas para representar e raciocinar sobre problemas computacionais.
- A representação se dá através de uma linguagem formal, de *fórmulas*.

Lógica

- Lógicas são utilizadas para representar e raciocinar sobre problemas computacionais.
- A representação se dá através de uma linguagem formal, de *fórmulas*.
- Para atribuir um significado a cada fórmula, define-se para a lógica uma *semântica*

Lógica

- Lógicas são utilizadas para representar e raciocinar sobre problemas computacionais.
- A representação se dá através de uma linguagem formal, de *fórmulas*.
- Para atribuir um significado a cada fórmula, define-se para a lógica uma *semântica*, que possui diferentes *interpretações*.

Lógica

- Lógicas são utilizadas para representar e raciocinar sobre problemas computacionais.
- A representação se dá através de uma linguagem formal, de *fórmulas*.
- Para atribuir um significado a cada fórmula, define-se para a lógica uma *semântica*, que possui diferentes *interpretações*.
- Em lógicas clássicas, os significados possíveis são somente *verdadeiro* ou *falso*.

SAT

Satisfatibilidade: Determinar se existe uma interpretação sob a qual uma dada fórmula é verdadeira.

SAT

Satisfatibilidade: Determinar se existe uma interpretação sob a qual uma dada fórmula é verdadeira.

- Possui grande interesse prático:

SAT

Satisfatibilidade: Determinar se existe uma interpretação sob a qual uma dada fórmula é verdadeira.

- Possui grande interesse prático:
  - Síntese [1], otimização [2] e verificação [3] de *hardware*.

SAT

Satisfatibilidade: Determinar se existe uma interpretação sob a qual uma dada fórmula é verdadeira.

- Possui grande interesse prático:
  - Síntese [1], otimização [2] e verificação [3] de *hardware*.
  - Raciocínio automático [4].

SAT

Satisfatibilidade: Determinar se existe uma interpretação sob a qual uma dada fórmula é verdadeira.

- Possui grande interesse prático:
  - Síntese [1], otimização [2] e verificação [3] de *hardware*.
  - Raciocínio automático [4].
  - Biologia e medicina [5].

SAT

Satisfatibilidade: Determinar se existe uma interpretação sob a qual uma dada fórmula é verdadeira.

- Possui grande interesse prático:
  - Síntese [1], otimização [2] e verificação [3] de *hardware*.
  - Raciocínio automático [4].
  - Biologia e medicina [5].
- Interesse teórico fundamental:

SAT

Satisfatibilidade: Determinar se existe uma interpretação sob a qual uma dada fórmula é verdadeira.

- Possui grande interesse prático:
  - Síntese [1], otimização [2] e verificação [3] de *hardware*.
  - Raciocínio automático [4].
  - Biologia e medicina [5].
- Interesse teórico fundamental:
  - Primeiro problema NP-completo [6].

SAT

Satisfatibilidade: Determinar se existe uma interpretação sob a qual uma dada fórmula é verdadeira.

- Possui grande interesse prático:
  - Síntese [1], otimização [2] e verificação [3] de *hardware*.
  - Raciocínio automático [4].
  - Biologia e medicina [5].
- Interesse teórico fundamental:
  - Primeiro problema NP-completo [6].
  - Deu base para formalizar P *versus* NP [6].

VAL

Validade: Determinar se uma dada fórmula é verdadeira sob qualquer interpretação.

VAL

Validade: Determinar se uma dada fórmula é verdadeira sob qualquer interpretação.

SAT e VAL são redutíveis um ao outro!

Algoritmos para SAT e VAL

- Há diversos algoritmos de busca para SAT e VAL [7, 8, 9].

Algoritmos para SAT e VAL

- Há diversos algoritmos de busca para SAT e VAL [7, 8, 9].
- Conjectura-se que todos são exponenciais [6].

Algoritmos para SAT e VAL

- Há diversos algoritmos de busca para SAT e VAL [7, 8, 9].
- Conjectura-se que todos são exponenciais [6].
- Muitos são baseados em *formas normais*: subconjuntos de fórmulas.

Algoritmos para SAT e VAL

- Há diversos algoritmos de busca para SAT e VAL [7, 8, 9].
- Conjectura-se que todos são exponenciais [6].
- Muitos são baseados em *formas normais*: subconjuntos de fórmulas.
- Algoritmos baseados em formas normais precisam de pré-processamento eficiente.

O trabalho

Hipótese

Considerando melhorar a eficiência total de pré-processamento e busca: fórmulas menores produzem respostas mais rápido?

O trabalho

Hipótese

Considerando melhorar a eficiência total de pré-processamento e busca: fórmulas menores produzem respostas mais rápido?

Objetivo

Testar a hipótese experimentalmente.

O trabalho

- Investigamos algoritmos baseados na *forma normal clausal*.

O trabalho

- Investigamos algoritmos baseados na *forma normal clausal*.
- Tentamos obter fórmulas pequenas reduzindo o *número de cláusulas*

O trabalho

- Investigamos algoritmos baseados na *forma normal clausal*.
- Tentamos obter fórmulas pequenas reduzindo o *número de cláusulas*, através de *renomeamento*.

O trabalho

- Investigamos algoritmos baseados na *forma normal clausal*.
- Tentamos obter fórmulas pequenas reduzindo o *número de cláusulas*, através de *renomeamento*.
- Boy de la Tour [10] e Jackson et al. [11] propõem algoritmos para este problema.

O trabalho

- Investigamos algoritmos baseados na *forma normal clausal*.
- Tentamos obter fórmulas pequenas reduzindo o *número de cláusulas*, através de *renomeamento*.
- Boy de la Tour [10] e Jackson et al. [11] propõem algoritmos para este problema.
- Propomos um algoritmo baseado em programação dinâmica para este problema.

O trabalho

- Investigamos algoritmos baseados na *forma normal clausal*.
- Tentamos obter fórmulas pequenas reduzindo o *número de cláusulas*, através de *renomeamento*.
- Boy de la Tour [10] e Jackson et al. [11] propõem algoritmos para este problema.
- Propomos um algoritmo baseado em programação dinâmica para este problema.
- Comparamos experimentalmente o algoritmo que propomos com o de Boy de la Tour.

Conteúdo

Lógica proposicional
Sintaxe

Símbolos proposicionais

$\mathcal{P} = \{a, b, ..., a_1, a_2, ..., b_1, b_2, ...\}$ é dito o conjunto de *símbolos proposicionais*.

Lógica proposicional
Sintaxe

Símbolos proposicionais

$\mathcal{P} = \{a, b, ..., a_1, a_2, ..., b_1, b_2, ...\}$ é dito o conjunto de *símbolos proposicionais*.

Fórmulas

Se $\phi \in \mathcal{P}$, então ϕ é uma *fórmula*. Além disso, se $\phi_1, ..., \phi_n$, $n \in \mathbb{N} \cup \{0\}$, são fórmulas, então também são:

Lógica proposicional
Sintaxe

Símbolos proposicionais

$\mathcal{P} = \{a, b, ..., a_1, a_2, ..., b_1, b_2, ...\}$ é dito o conjunto de *símbolos proposicionais*.

Fórmulas

Se $\phi \in \mathcal{P}$, então ϕ é uma *fórmula*. Além disso, se $\phi_1, ..., \phi_n$, $n \in \mathbb{N} \cup \{0\}$, são fórmulas, então também são:

1. *Negação*: $\neg\phi_1$

Lógica proposicional
Sintaxe

Símbolos proposicionais

$\mathcal{P} = \{a, b, ..., a_1, a_2, ..., b_1, b_2, ...\}$ é dito o conjunto de *símbolos proposicionais*.

Fórmulas

Se $\phi \in \mathcal{P}$, então ϕ é uma *fórmula*. Além disso, se $\phi_1, ..., \phi_n$, $n \in \mathbb{N} \cup \{0\}$, são fórmulas, então também são:

1. *Negação*: $\neg\phi_1$
2. *Conjunção*: $\phi_1 \wedge ... \wedge \phi_n$

Lógica proposicional
Sintaxe

Símbolos proposicionais

$\mathcal{P} = \{a, b, ..., a_1, a_2, ..., b_1, b_2, ...\}$ é dito o conjunto de *símbolos proposicionais*.

Fórmulas

Se $\phi \in \mathcal{P}$, então ϕ é uma *fórmula*. Além disso, se $\phi_1, ..., \phi_n$, $n \in \mathbb{N} \cup \{0\}$, são fórmulas, então também são:

1. *Negação*: $\neg\phi_1$
2. *Conjunção*: $\phi_1 \wedge ... \wedge \phi_n$
3. *Disjunção*: $\phi_1 \vee ... \vee \phi_n$

Lógica proposicional
Sintaxe

Símbolos proposicionais

$\mathcal{P} = \{a, b, ..., a_1, a_2, ..., b_1, b_2, ...\}$ é dito o conjunto de *símbolos proposicionais*.

Fórmulas

Se $\phi \in \mathcal{P}$, então ϕ é uma *fórmula*. Além disso, se $\phi_1, ..., \phi_n$, $n \in \mathbb{N} \cup \{0\}$, são fórmulas, então também são:

1. *Negação*: $\neg\phi_1$
2. *Conjunção*: $\phi_1 \wedge ... \wedge \phi_n$
3. *Disjunção*: $\phi_1 \vee ... \vee \phi_n$
4. *Implicação*: $\phi_1 \to \phi_2$

Lógica proposicional
Sintaxe

Símbolos proposicionais

$\mathcal{P} = \{a, b, ..., a_1, a_2, ..., b_1, b_2, ...\}$ é dito o conjunto de *símbolos proposicionais*.

Fórmulas

Se $\phi \in \mathcal{P}$, então ϕ é uma *fórmula*. Além disso, se $\phi_1, ..., \phi_n$, $n \in \mathbb{N} \cup \{0\}$, são fórmulas, então também são:

1. *Negação*: $\neg\phi_1$
2. *Conjunção*: $\phi_1 \wedge ... \wedge \phi_n$
3. *Disjunção*: $\phi_1 \vee ... \vee \phi_n$
4. *Implicação*: $\phi_1 \rightarrow \phi_2$
5. *Equivalência*: $\phi_1 \leftrightarrow \phi_2$

Lógica proposicional

Sintaxe

Símbolos proposicionais

$\mathcal{P} = \{a, b, ..., a_1, a_2, ..., b_1, b_2, ...\}$ é dito o conjunto de *símbolos proposicionais*.

Fórmulas

Se $\phi \in \mathcal{P}$, então ϕ é uma *fórmula*. Além disso, se $\phi_1, ..., \phi_n$, $n \in \mathbb{N} \cup \{0\}$, são fórmulas, então também são:

1. *Negação*: $\neg\phi_1$
2. *Conjunção*: $\phi_1 \wedge ... \wedge \phi_n$
3. *Disjunção*: $\phi_1 \vee ... \vee \phi_n$
4. *Implicação*: $\phi_1 \rightarrow \phi_2$
5. *Equivalência*: $\phi_1 \leftrightarrow \phi_2$

Denotamos o conjunto de fórmulas por $\mathcal{L}$.

Lógica proposicional
Sintaxe

Subfórmulas imediatas

Na definição anterior, as fórmulas ϕ_i são *subfórmulas imediatas*.

Lógica proposicional
Sintaxe

Subfórmulas imediatas

Na definição anterior, as fórmulas ϕ_i são *subfórmulas imediatas*.

Subfórmulas

Dizemos que ψ é subfórmula de ϕ se ψ é subfórmula imediata de ϕ, ou se ψ é subfórmula de ξ e ξ é subfórmula imediata de ϕ.

Lógica proposicional
Sintaxe

Subfórmulas imediatas

Na definição anterior, as fórmulas ϕ_i são *subfórmulas imediatas*.

Subfórmulas

Dizemos que ψ é subfórmula de ϕ se ψ é subfórmula imediata de ϕ, ou se ψ é subfórmula de ξ e ξ é subfórmula imediata de ϕ.
Notação: $\psi \sqsubset \phi$ e $\{\psi \mid \psi \sqsubset \phi\} = SF(\phi)$

Lógica proposicional
Semântica

Valorações booleanas

Dizemos que $\mathbb{v}_0$ é uma *valoração booleana* se $\mathbb{v}_0 : \mathcal{P} \longmapsto \{V, F\}$.

Lógica proposicional
Semântica

Valorações booleanas

Dizemos que $\mathbb{v}_0$ é uma *valoração booleana* se $\mathbb{v}_0 : \mathcal{P} \longmapsto \{V, F\}$.

Interpretações

Seja $\mathbb{v}_0$ é uma valoração booleana. Dizemos que $\mathbb{v} : \mathcal{L} \longmapsto \{V, F\}$ é uma *interpretação definida por* $\mathbb{v}_0$, se:

Lógica proposicional
Semântica

Valorações booleanas

Dizemos que $\mathbb{v}_0$ é uma *valoração booleana* se $\mathbb{v}_0 : \mathcal{P} \longmapsto \{V, F\}$.

Interpretações

Seja $\mathbb{v}_0$ é uma valoração booleana. Dizemos que $\mathbb{v} : \mathcal{L} \longmapsto \{V, F\}$ é uma *interpretação definida por* $\mathbb{v}_0$, se:

1. Se $\phi_1 \in \mathcal{P}$, então $\mathbb{v}(\phi_1) = \mathbb{v}_0(\phi_1)$.

Lógica proposicional

Semântica

Valorações booleanas

Dizemos que $\mathbb{v}_0$ é uma *valoração booleana* se $\mathbb{v}_0 : \mathcal{P} \longmapsto \{V, F\}$.

Interpretações

Seja $\mathbb{v}_0$ é uma valoração booleana. Dizemos que $\mathbb{v} : \mathcal{L} \longmapsto \{V, F\}$ é uma *interpretação definida por* $\mathbb{v}_0$, se:

1. Se $\phi_1 \in \mathcal{P}$, então $\mathbb{v}(\phi_1) = \mathbb{v}_0(\phi_1)$.
2. $\mathbb{v}(\neg\phi_1) = V$ se, e somente se, $\mathbb{v}(\phi_1) = F$.

Lógica proposicional
Semântica

Valorações booleanas

Dizemos que $\mathbb{v}_0$ é uma *valoração booleana* se $\mathbb{v}_0 : \mathcal{P} \longmapsto \{V, F\}$.

Interpretações

Seja $\mathbb{v}_0$ é uma valoração booleana. Dizemos que $\mathbb{v} : \mathcal{L} \longmapsto \{V, F\}$ é uma *interpretação definida por* $\mathbb{v}_0$, se:

1. Se $\phi_1 \in \mathcal{P}$, então $\mathbb{v}(\phi_1) = \mathbb{v}_0(\phi_1)$.
2. $\mathbb{v}(\neg\phi_1) = V$ se, e somente se, $\mathbb{v}(\phi_1) = F$.
3. $\mathbb{v}(\phi_1 \wedge ... \wedge \phi_n) = V$ se, e somente se, $\mathbb{v}(\phi_i) = V$, para todo i.

Lógica proposicional
Semântica

Valorações booleanas

Dizemos que $\mathbb{v}_0$ é uma *valoração booleana* se $\mathbb{v}_0 : \mathcal{P} \longmapsto \{V, F\}$.

Interpretações

Seja $\mathbb{v}_0$ é uma valoração booleana. Dizemos que $\mathbb{v} : \mathcal{L} \longmapsto \{V, F\}$ é uma *interpretação definida por* $\mathbb{v}_0$, se:

1. Se $\phi_1 \in \mathcal{P}$, então $\mathbb{v}(\phi_1) = \mathbb{v}_0(\phi_1)$.
2. $\mathbb{v}(\neg\phi_1) = V$ se, e somente se, $\mathbb{v}(\phi_1) = F$.
3. $\mathbb{v}(\phi_1 \wedge ... \wedge \phi_n) = V$ se, e somente se, $\mathbb{v}(\phi_i) = V$, para todo i.
4. $\mathbb{v}(\phi_1 \vee ... \vee \phi_n) = V$ se, e somente se, $\mathbb{v}(\phi_i) = V$, para algum i.

Lógica proposicional
Semântica

Valorações booleanas

Dizemos que $\mathbb{v}_0$ é uma *valoração booleana* se $\mathbb{v}_0 : \mathcal{P} \longmapsto \{V, F\}$.

Interpretações

Seja $\mathbb{v}_0$ é uma valoração booleana. Dizemos que $\mathbb{v} : \mathcal{L} \longmapsto \{V, F\}$ é uma *interpretação definida por* $\mathbb{v}_0$, se:

1. Se $\phi_1 \in \mathcal{P}$, então $\mathbb{v}(\phi_1) = \mathbb{v}_0(\phi_1)$.
2. $\mathbb{v}(\neg\phi_1) = V$ se, e somente se, $\mathbb{v}(\phi_1) = F$.
3. $\mathbb{v}(\phi_1 \wedge ... \wedge \phi_n) = V$ se, e somente se, $\mathbb{v}(\phi_i) = V$, para todo i.
4. $\mathbb{v}(\phi_1 \vee ... \vee \phi_n) = V$ se, e somente se, $\mathbb{v}(\phi_i) = V$, para algum i.
5. $\mathbb{v}(\phi_1 \rightarrow \phi_2) = V$ se, e somente se, $\mathbb{v}(\phi_1) = F$ ou $\mathbb{v}(\phi_2) = V$.

Lógica proposicional
Semântica

Valorações booleanas

Dizemos que $\mathbb{v}_0$ é uma *valoração booleana* se $\mathbb{v}_0 : \mathcal{P} \longmapsto \{V, F\}$.

Interpretações

Seja $\mathbb{v}_0$ é uma valoração booleana. Dizemos que $\mathbb{v} : \mathcal{L} \longmapsto \{V, F\}$ é uma *interpretação definida por* $\mathbb{v}_0$, se:

1. Se $\phi_1 \in \mathcal{P}$, então $\mathbb{v}(\phi_1) = \mathbb{v}_0(\phi_1)$.
2. $\mathbb{v}(\neg\phi_1) = V$ se, e somente se, $\mathbb{v}(\phi_1) = F$.
3. $\mathbb{v}(\phi_1 \wedge ... \wedge \phi_n) = V$ se, e somente se, $\mathbb{v}(\phi_i) = V$, para todo i.
4. $\mathbb{v}(\phi_1 \vee ... \vee \phi_n) = V$ se, e somente se, $\mathbb{v}(\phi_i) = V$, para algum i.
5. $\mathbb{v}(\phi_1 \rightarrow \phi_2) = V$ se, e somente se, $\mathbb{v}(\phi_1) = F$ ou $\mathbb{v}(\phi_2) = V$.
6. $\mathbb{v}(\phi_1 \leftrightarrow \phi_2) = V$ se, e somente se, $\mathbb{v}(\phi_1) = \mathbb{v}(\phi_2)$.

Lógica proposicional

Semântica – Algumas definições

1. Se existe $\mathbb{v}$ tal que $\mathbb{v}(\phi) = V$, dizemos que ϕ é *satisfatível*.

Lógica proposicional
Semântica – Algumas definições

1. Se existe $\mathbb{v}$ tal que $\mathbb{v}(\phi) = V$, dizemos que ϕ é *satisfatível*.
2. Se existe $\mathbb{v}$ tal que $\mathbb{v}(\phi) = F$, dizemos que ϕ é *falsificável*.

Lógica proposicional

Semântica – Algumas definições

1. Se existe $\mathbb{v}$ tal que $\mathbb{v}(\phi) = V$, dizemos que ϕ é *satisfatível*.
2. Se existe $\mathbb{v}$ tal que $\mathbb{v}(\phi) = F$, dizemos que ϕ é *falsificável*.
3. Se $\mathbb{v}(\phi) = V$ para toda $\mathbb{v}$, dizemos que ϕ é uma *tautologia*.

Lógica proposicional
Semântica – Algumas definições

1. Se existe $\mathbb{v}$ tal que $\mathbb{v}(\phi) = V$, dizemos que ϕ é *satisfatível*.
2. Se existe $\mathbb{v}$ tal que $\mathbb{v}(\phi) = F$, dizemos que ϕ é *falsificável*.
3. Se $\mathbb{v}(\phi) = V$ para toda $\mathbb{v}$, dizemos que ϕ é uma *tautologia*.
4. Se $\mathbb{v}(\phi) = F$ para toda $\mathbb{v}$, dizemos que ϕ é uma *contradição*, ou que ϕ é *insatisfatível*.

Lógica proposicional

Semântica – Algumas definições

1. Se existe $\mathbb{v}$ tal que $\mathbb{v}(\phi) = V$, dizemos que ϕ é *satisfatível*.
2. Se existe $\mathbb{v}$ tal que $\mathbb{v}(\phi) = F$, dizemos que ϕ é *falsificável*.
3. Se $\mathbb{v}(\phi) = V$ para toda $\mathbb{v}$, dizemos que ϕ é uma *tautologia*.
4. Se $\mathbb{v}(\phi) = F$ para toda $\mathbb{v}$, dizemos que ϕ é uma *contradição*, ou que ϕ é *insatisfatível*.
5. Se ϕ é satisfatível e falsificável, dizemos que ϕ é uma *contingência*.

Problemas da lógica proposicional

Seja $L \subseteq \mathcal{L}$. Se nos referimos a L como um *problema*, referimo-nos ao problema de, dada ϕ qualquer, determinar se $\phi \in L$ ou se $\phi \notin L$.

Problemas da lógica proposicional

Seja $L \subseteq \mathcal{L}$. Se nos referimos a L como um *problema*, referimo-nos ao problema de, dada ϕ qualquer, determinar se $\phi \in L$ ou se $\phi \notin L$.

1. SAT $= \{\phi \in \mathcal{L} \mid \phi$ é satisfatível$\}$

Problemas da lógica proposicional

Seja $L \subseteq \mathcal{L}$. Se nos referimos a L como um *problema*, referimo-nos ao problema de, dada ϕ qualquer, determinar se $\phi \in L$ ou se $\phi \notin L$.

1. $\text{SAT} = \{\phi \in \mathcal{L} \mid \phi \text{ é satisfatível}\}$
2. $\text{UNSAT} = \{\phi \in \mathcal{L} \mid \phi \text{ é insatisfatível}\}$

Problemas da lógica proposicional

Seja $L \subseteq \mathcal{L}$. Se nos referimos a L como um *problema*, referimo-nos ao problema de, dada ϕ qualquer, determinar se $\phi \in L$ ou se $\phi \notin L$.

1. $\mathrm{SAT} = \{\phi \in \mathcal{L} \mid \phi \text{ é satisfatível}\}$
2. $\mathrm{UNSAT} = \{\phi \in \mathcal{L} \mid \phi \text{ é insatisfatível}\} = \overline{\mathrm{SAT}}$

Problemas da lógica proposicional

Seja $L \subseteq \mathcal{L}$. Se nos referimos a L como um *problema*, referimo-nos ao problema de, dada ϕ qualquer, determinar se $\phi \in L$ ou se $\phi \notin L$.

1. $\text{SAT} = \{\phi \in \mathcal{L} \mid \phi \text{ é satisfatível}\}$
2. $\text{UNSAT} = \{\phi \in \mathcal{L} \mid \phi \text{ é insatisfatível}\} = \overline{\text{SAT}}$
3. $\text{VAL} = \{\phi \in \mathcal{L} \mid \phi \text{ é tautologia}\}$

Problemas da lógica proposicional

Seja $L \subseteq \mathcal{L}$. Se nos referimos a L como um *problema*, referimo-nos ao problema de, dada ϕ qualquer, determinar se $\phi \in L$ ou se $\phi \notin L$.

1. $\mathrm{SAT} = \{\phi \in \mathcal{L} \mid \phi \text{ é satisfatível}\}$
2. $\mathrm{UNSAT} = \{\phi \in \mathcal{L} \mid \phi \text{ é insatisfatível}\} = \overline{\mathrm{SAT}}$
3. $\mathrm{VAL} = \{\phi \in \mathcal{L} \mid \phi \text{ é tautologia}\}$

$$\phi \in \mathrm{VAL} \iff \neg\phi \in \mathrm{UNSAT} \implies \phi \in \mathrm{SAT}$$

Formas normais

Regras de reescrita

Uma *regra de reescrita* que transforma ϕ em ψ, escrito $\phi \longmapsto \psi$,

Formas normais

Regras de reescrita

Uma *regra de reescrita* que transforma ϕ em ψ, escrito $\phi \longmapsto \psi$,

1. *preserva equivalência* se, e somente se, $\mathbb{v}(\phi) = \mathbb{v}(\psi), \forall \mathbb{v}$.

Formas normais

Regras de reescrita

Uma *regra de reescrita* que transforma ϕ em ψ, escrito $\phi \longmapsto \psi$,

1. *preserva equivalência* se, e somente se, $\mathbb{v}(\phi) = \mathbb{v}(\psi), \forall \mathbb{v}$.
2. *preserva satisfatibilidade* se, e somente se, $\phi, \psi \in \text{SAT}$ ou $\phi, \psi \notin \text{SAT}$.

Formas normais

Forma normal negada (FNN)

$$(p \wedge \neg q \wedge \neg r) \vee (x \wedge \neg y \wedge (r \vee s))$$

Formas normais

Forma normal negada (FNN)

$$(p \land \neg q \land \neg r) \lor (x \land \neg y \land (r \lor s))$$

As transformações:

1. $\neg\neg\phi_1 \longmapsto \phi_1$ (eliminação de dupla negação)
2. $\neg(\phi_1 \land \ldots \land \phi_n) \longmapsto \neg\phi_1 \lor \ldots \lor \neg\phi_n$ (De Morgan)
3. $\neg(\phi_1 \lor \ldots \lor \phi_n) \longmapsto \neg\phi_1 \land \ldots \land \neg\phi_n$ (De Morgan)
4. $\phi_1 \to \phi_2 \longmapsto \neg\phi_1 \lor \phi_2$
5. $\phi_1 \leftrightarrow \phi_2 \longmapsto (\phi_1 \to \phi_2) \land (\phi_2 \to \phi_1)$

Formas normais

Forma normal negada (FNN)

$$(p \wedge \neg q \wedge \neg r) \vee (x \wedge \neg y \wedge (r \vee s))$$

As transformações:

1. $\neg\neg\phi_1 \longmapsto \phi_1$ (eliminação de dupla negação)
2. $\neg(\phi_1 \wedge \ldots \wedge \phi_n) \longmapsto \neg\phi_1 \vee \ldots \vee \neg\phi_n$ (De Morgan)
3. $\neg(\phi_1 \vee \ldots \vee \phi_n) \longmapsto \neg\phi_1 \wedge \ldots \wedge \neg\phi_n$ (De Morgan)
4. $\phi_1 \rightarrow \phi_2 \longmapsto \neg\phi_1 \vee \phi_2$
5. $\phi_1 \leftrightarrow \phi_2 \longmapsto (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)$

preservam equivalência!

Formas normais

Forma normal clausal (FNC)

$$(p \vee \neg q \vee \neg r) \wedge (x \vee \neg y \vee r \vee s) \wedge (a \vee \neg b \vee c)$$

Formas normais
Forma normal clausal (FNC)

$$(p \vee \neg q \vee \neg r) \wedge (x \vee \neg y \vee r \vee s) \wedge (a \vee \neg b \vee c)$$

A transformação:

$$\phi \vee (\psi \wedge \xi) \longmapsto (\phi \vee \psi) \wedge (\phi \vee \xi) \quad \text{(distribuição)}$$

Formas normais

Forma normal clausal (FNC)

$$(p \vee \neg q \vee \neg r) \wedge (x \vee \neg y \vee r \vee s) \wedge (a \vee \neg b \vee c)$$

A transformação:

$$\phi \vee (\psi \wedge \xi) \longmapsto (\phi \vee \psi) \wedge (\phi \vee \xi) \quad \text{(distribuição)}$$

preserva equivalência!

Formas normais

Forma normal clausal (FNC)

$$(p \vee \neg q \vee \neg r) \wedge (x \vee \neg y \vee r \vee s) \wedge (a \vee \neg b \vee c)$$

A transformação:

$$\phi \vee (\psi \wedge \xi) \longmapsto (\phi \vee \psi) \wedge (\phi \vee \xi) \quad \text{(distribuição)}$$

preserva equivalência!

Geralmente provoca crescimento exponencial!

Renomeamento

1. Escolhemos um conjunto de subfórmulas $R \subseteq SF(\phi)$.

Renomeamento

1. Escolhemos um conjunto de subfórmulas $R \subseteq SF(\phi)$.
2. Para cada $\psi \in R$:

Renomeamento

1. Escolhemos um conjunto de subfórmulas $R \subseteq SF(\phi)$.
2. Para cada $\psi \in R$:
    1. Escolhemos um símbolo proposicional novo $s(\psi) \in \mathcal{P}$.

Renomeamento

1. Escolhemos um conjunto de subfórmulas $R \subseteq SF(\phi)$.
2. Para cada $\psi \in R$:
    1. Escolhemos um símbolo proposicional novo $s(\psi) \in \mathcal{P}$.
    2. Trocamos todas as ocorrências de ψ por $s(\psi)$.

Renomeamento

1. Escolhemos um conjunto de subfórmulas $R \subseteq SF(\phi)$.
2. Para cada $\psi \in R$:
    1. Escolhemos um símbolo proposicional novo $s(\psi) \in \mathcal{P}$.
    2. Trocamos todas as ocorrências de ψ por $s(\psi)$.
    3. Incluímos a definição $s(\psi) \to \psi$ em conjunção.

Renomeamento

1. Escolhemos um conjunto de subfórmulas $R \subseteq SF(\phi)$.
2. Para cada $\psi \in R$:
   1. Escolhemos um símbolo proposicional novo $s(\psi) \in \mathcal{P}$.
   2. Trocamos todas as ocorrências de ψ por $s(\psi)$.
   3. Incluímos a definição $s(\psi) \rightarrow \psi$ em conjunção.

Exemplo: $(\neg p_1 \wedge p_2 \wedge p_3) \vee (\neg q_1 \wedge q_2 \wedge \neg q_3)$

Renomeamento

1. Escolhemos um conjunto de subfórmulas $R \subseteq SF(\phi)$.
2. Para cada $\psi \in R$:
    1. Escolhemos um símbolo proposicional novo $s(\psi) \in \mathcal{P}$.
    2. Trocamos todas as ocorrências de ψ por $s(\psi)$.
    3. Incluímos a definição $s(\psi) \rightarrow \psi$ em conjunção.

Exemplo: $(\neg p_1 \wedge p_2 \wedge p_3) \vee (\neg q_1 \wedge q_2 \wedge \neg q_3)$
Seja $\phi_1 = \neg p_1 \wedge p_2 \wedge p_3$.

Renomeamento

1. Escolhemos um conjunto de subfórmulas $R \subseteq SF(\phi)$.
2. Para cada $\psi \in R$:
    1. Escolhemos um símbolo proposicional novo $s(\psi) \in \mathcal{P}$.
    2. Trocamos todas as ocorrências de ψ por $s(\psi)$.
    3. Incluímos a definição $s(\psi) \to \psi$ em conjunção.

Exemplo: $(\neg p_1 \wedge p_2 \wedge p_3) \vee (\neg q_1 \wedge q_2 \wedge \neg q_3)$
Seja $\phi_1 = \neg p_1 \wedge p_2 \wedge p_3$.
Escolhendo $R = \{\phi_1\}$ e $s(\phi_1) = a$, temos

Renomeamento

1. Escolhemos um conjunto de subfórmulas $R \subseteq SF(\phi)$.
2. Para cada $\psi \in R$:
   1. Escolhemos um símbolo proposicional novo $s(\psi) \in \mathcal{P}$.
   2. Trocamos todas as ocorrências de ψ por $s(\psi)$.
   3. Incluímos a definição $s(\psi) \to \psi$ em conjunção.

Exemplo: $(\neg p_1 \wedge p_2 \wedge p_3) \vee (\neg q_1 \wedge q_2 \wedge \neg q_3)$
Seja $\phi_1 = \neg p_1 \wedge p_2 \wedge p_3$.
Escolhendo $R = \{\phi_1\}$ e $s(\phi_1) = a$, temos

$$(a \vee (\neg q_1 \wedge q_2 \wedge \neg q_3)) \wedge (a \to (\neg p_1 \wedge p_2 \wedge p_3))$$

Renomeamento

1. Escolhemos um conjunto de subfórmulas $R \subseteq SF(\phi)$.
2. Para cada $\psi \in R$:
    1. Escolhemos um símbolo proposicional novo $s(\psi) \in \mathcal{P}$.
    2. Trocamos todas as ocorrências de ψ por $s(\psi)$.
    3. Incluímos a definição $s(\psi) \to \psi$ em conjunção.

Exemplo: $(\neg p_1 \wedge p_2 \wedge p_3) \vee (\neg q_1 \wedge q_2 \wedge \neg q_3)$
Seja $\phi_1 = \neg p_1 \wedge p_2 \wedge p_3$.
Escolhendo $R = \{\phi_1\}$ e $s(\phi_1) = a$, temos

$$(a \vee (\neg q_1 \wedge q_2 \wedge \neg q_3)) \wedge (a \to (\neg p_1 \wedge p_2 \wedge p_3))$$

Não preserva equivalência.

Renomeamento

1. Escolhemos um conjunto de subfórmulas $R \subseteq SF(\phi)$.
2. Para cada $\psi \in R$:
   1. Escolhemos um símbolo proposicional novo $s(\psi) \in \mathcal{P}$.
   2. Trocamos todas as ocorrências de ψ por $s(\psi)$.
   3. Incluímos a definição $s(\psi) \to \psi$ em conjunção.

Exemplo: $(\neg p_1 \wedge p_2 \wedge p_3) \vee (\neg q_1 \wedge q_2 \wedge \neg q_3)$
Seja $\phi_1 = \neg p_1 \wedge p_2 \wedge p_3$.
Escolhendo $R = \{\phi_1\}$ e $s(\phi_1) = a$, temos

$$(a \vee (\neg q_1 \wedge q_2 \wedge \neg q_3)) \wedge (a \to (\neg p_1 \wedge p_2 \wedge p_3))$$

Não preserva equivalência. Mas preserva satisfatibilidade!

Reduzindo o número de cláusulas

Contando cláusulas

Denotamos o *número de cláusulas* geradas por ϕ ao ser colocada na FNC por $p(\phi)$.

Reduzindo o número de cláusulas

Contando cláusulas

Denotamos o *número de cláusulas* geradas por ϕ ao ser colocada na FNC por $p(\phi)$.

Forma de ϕ	$p(\phi)$
$\phi_1 \wedge \ldots \wedge \phi_n$	$p(\phi_1) + \ldots + p(\phi_n)$
$\phi_1 \vee \ldots \vee \phi_n$	$p(\phi_1) \cdot \ldots \cdot p(\phi_n)$
x ou $\neg x, x \in \mathcal{P}$	1

Reduzindo o número de cláusulas

Contando cláusulas

Denotamos o *número de cláusulas* geradas por ϕ ao ser colocada na FNC por $p(\phi)$.

Forma de ϕ	$p(\phi)$
$\phi_1 \wedge \ldots \wedge \phi_n$	$p(\phi_1) + \ldots + p(\phi_n)$
$\phi_1 \vee \ldots \vee \phi_n$	$p(\phi_1) \cdot \ldots \cdot p(\phi_n)$
x ou $\neg x, x \in \mathcal{P}$	1

Exemplo: $\phi = (\neg p_1 \wedge p_2 \wedge p_3) \vee (\neg q_1 \wedge q_2 \wedge \neg q_3) \vee (r_1 \wedge r_2 \wedge \neg r_3)$

Reduzindo o número de cláusulas

Contando cláusulas

Denotamos o *número de cláusulas* geradas por ϕ ao ser colocada na FNC por $p(\phi)$.

Forma de ϕ	$p(\phi)$
$\phi_1 \wedge \ldots \wedge \phi_n$	$p(\phi_1) + \ldots + p(\phi_n)$
$\phi_1 \vee \ldots \vee \phi_n$	$p(\phi_1) \cdot \ldots \cdot p(\phi_n)$
x ou $\neg x, x \in \mathcal{P}$	1

Exemplo: $\phi = (\neg p_1 \wedge p_2 \wedge p_3) \vee (\neg q_1 \wedge q_2 \wedge \neg q_3) \vee (r_1 \wedge r_2 \wedge \neg r_3)$
Temos que

$$p(\phi) = (1+1+1)(1+1+1)(1+1+1) = 3^3 = 27$$

Reduzindo o número de cláusulas

O problema

Problema

Escolher $R \subseteq SF(\phi)$ de modo que o número de cláusulas $p(\phi, R)$ da transformação por renomeamento seja mínimo.

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour

Árvores lineares

Seja ϕ uma fórmula na FNN. Se cada subfórmula de ϕ ocorre somente uma vez, dizemos que ϕ é uma *árvore linear*.

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour

Árvores lineares

Seja ϕ uma fórmula na FNN. Se cada subfórmula de ϕ ocorre somente uma vez, dizemos que ϕ é uma *árvore linear*.

Se ϕ é uma árvore linear, o algoritmo de Boy de la Tour encontra um conjunto $R \subseteq SF(\phi)$ tal que $p(\phi, R)$ é ótimo (mínimo).

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour

Árvores lineares

Seja ϕ uma fórmula na FNN. Se cada subfórmula de ϕ ocorre somente uma vez, dizemos que ϕ é uma *árvore linear*.

Se ϕ é uma árvore linear, o algoritmo de Boy de la Tour encontra um conjunto $R \subseteq SF(\phi)$ tal que $p(\phi, R)$ é ótimo (mínimo).

Seu custo de tempo no pior caso é $O(|SF(\phi)|^2)$.

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour

O algoritmo escreve o número de cláusulas na forma irredutível

$$p(\phi) = c + a_{\psi}^{\phi} \cdot p(\psi)$$

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour

O algoritmo escreve o número de cláusulas na forma irredutível

$$p(\phi) = c + a_{\psi}^{\phi} \cdot p(\psi)$$

Logo, se $R = \{\psi\}$, então

$$p(\phi, R) = c + a_{\psi}^{\phi} + p(\psi)$$

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour

O algoritmo escreve o número de cláusulas na forma irredutível

$$p(\phi) = c + a_\psi^\phi \cdot p(\psi)$$

Logo, se $R = \{\psi\}$, então

$$p(\phi, R) = c + a_\psi^\phi + p(\psi)$$

Assim, é feita em ϕ uma busca em profundidade pré-ordem, incluindo cada $\psi \sqsubset \phi$ que satisfaz

$$a_\psi^\phi \cdot p(\psi) > a_\psi^\phi + p(\psi)$$

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour – Exemplo

$$\phi = (\neg p_1 \wedge p_2 \wedge p_3) \vee (\neg q_1 \wedge q_2 \wedge \neg q_3) \vee (r_1 \wedge r_2 \wedge \neg r_3)$$

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour – Exemplo

$$\phi = (\neg p_1 \wedge p_2 \wedge p_3) \vee (\neg q_1 \wedge q_2 \wedge \neg q_3) \vee (r_1 \wedge r_2 \wedge \neg r_3)$$

$$\begin{tikzcd}
& & & & \vee \arrow[dlll] \arrow[d] \arrow[drrr] & & & & \\
& \wedge \arrow[dl] \arrow[d] \arrow[dr] & & & \wedge \arrow[dl] \arrow[d] \arrow[dr] & & & \wedge \arrow[dl] \arrow[d] \arrow[dr] & \\
\neg p_1 & p_2 & p_3 & \neg q_1 & q_2 & \neg q_3 & r_1 & r_2 & \neg r_3
\end{tikzcd}$$

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour – Exemplo

$$
\begin{tikzcd}
\psi \arrow[rrrr, red] & & & & \lor \arrow[dlll] \arrow[d] \arrow[drrr] & & & & \\
& \land \arrow[dl] \arrow[d] \arrow[dr] & & & \land \arrow[dl] \arrow[d] \arrow[dr] & & & \land \arrow[dl] \arrow[d] \arrow[dr] & \\
\neg p_1 & p_2 & p_3 & \neg q_1 & q_2 & \neg q_3 & r_1 & r_2 & \neg r_3
\end{tikzcd}
$$

$$a_{\psi}^{\phi} = 1, p(\psi) = 27$$

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour – Exemplo

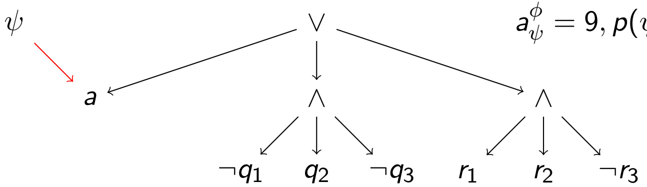
$a_{\psi}^{\phi} = 9, p(\psi) = 3$

$$
\begin{tikzcd}
\psi \arrow[dr] & & & & \vee \arrow[dlll] \arrow[d] \arrow[drrr] & & & & \\
& \wedge \arrow[dl] \arrow[d] \arrow[dr] & & & \wedge \arrow[dl] \arrow[d] \arrow[dr] & & & \wedge \arrow[dl] \arrow[d] \arrow[dr] & \\
\neg p_1 & p_2 & p_3 & \neg q_1 & q_2 & \neg q_3 & r_1 & r_2 & \neg r_3
\end{tikzcd}
$$

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour – Exemplo

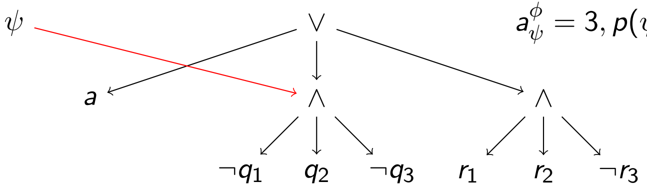
$a_{\psi}^{\phi} = 9, p(\psi) = 1$



$a \rightarrow (\neg p_1 \wedge p_2 \wedge p_3)$

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour – Exemplo



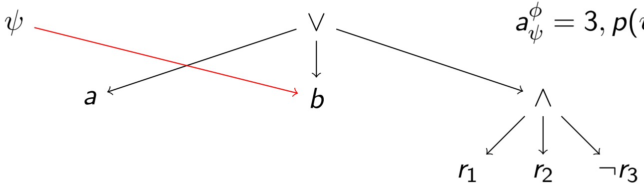
$a_{\psi}^{\phi} = 3, p(\psi) = 3$

$a \rightarrow (\neg p_1 \wedge p_2 \wedge p_3)$

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour – Exemplo

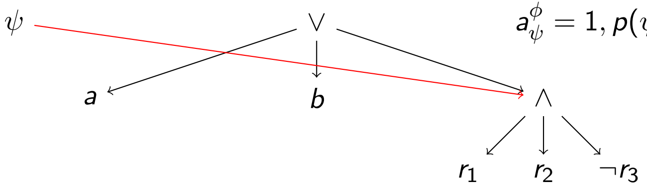
$a_{\psi}^{\phi} = 3, p(\psi) = 1$



$a \rightarrow (\neg p_1 \wedge p_2 \wedge p_3)$
$b \rightarrow (\neg q_1 \wedge q_2 \wedge \neg q_3)$

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour – Exemplo



$$a_\psi^\phi = 1, p(\psi) = 3$$

$$a \rightarrow (\neg p_1 \wedge p_2 \wedge p_3)$$
$$b \rightarrow (\neg q_1 \wedge q_2 \wedge \neg q_3)$$

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour – Exemplo

$$(a \vee b \vee (r_1 \wedge r_2 \wedge \neg r_3)) \wedge (a \rightarrow (\neg p_1 \wedge p_2 \wedge p_3)) \wedge (b \rightarrow (\neg q_1 \wedge q_2 \wedge \neg q_3))$$

Reduzindo o número de cláusulas

Algoritmo de Boy de la Tour – Exemplo

$$(a \vee b \vee (r_1 \wedge r_2 \wedge \neg r_3)) \wedge (a \rightarrow (\neg p_1 \wedge p_2 \wedge p_3)) \wedge (b \rightarrow (\neg q_1 \wedge q_2 \wedge \neg q_3))$$

Número de cláusulas: 9

Conteúdo

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Contraexemplo:

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Contraexemplo:

$\xi_1 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$,

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Contraexemplo:

$\xi_1 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$, $\xi_2 = q_1 \wedge q_2$,

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Contraexemplo:

$\xi_1 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$, $\xi_2 = q_1 \wedge q_2$, $\xi_3 = r_1 \wedge r_2$,

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Contraexemplo:

$\xi_1 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$, $\xi_2 = q_1 \wedge q_2$, $\xi_3 = r_1 \wedge r_2$,
$\xi_4 = s_1 \wedge ... \wedge s_{100}$

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Contraexemplo:

$\xi_1 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$, $\xi_2 = q_1 \wedge q_2$, $\xi_3 = r_1 \wedge r_2$,
$\xi_4 = s_1 \wedge \ldots \wedge s_{100}$ e $\phi = (\xi_1 \vee \xi_2) \wedge (\xi_3 \vee \xi_4)$

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Contraexemplo:
$\xi_1 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$, $\xi_2 = q_1 \wedge q_2$, $\xi_3 = r_1 \wedge r_2$,
$\xi_4 = s_1 \wedge \ldots \wedge s_{100}$ e $\phi = (\xi_1 \vee \xi_2) \wedge (\xi_3 \vee \xi_4)$
Então $p(\phi) = 208$

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Contraexemplo:

$\xi_1 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$, $\xi_2 = q_1 \wedge q_2$, $\xi_3 = r_1 \wedge r_2$,
$\xi_4 = s_1 \wedge \ldots \wedge s_{100}$ e $\phi = (\xi_1 \vee \xi_2) \wedge (\xi_3 \vee \xi_4)$
Então $p(\phi) = 208$ e $R = \{\xi_1, \xi_4\}$ é ótimo, com $p(\phi, R) = 108$.

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j-1$ subfórmulas.

Contraexemplo:
$\xi_1 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$, $\xi_2 = q_1 \wedge q_2$, $\xi_3 = r_1 \wedge r_2$,
$\xi_4 = s_1 \wedge \ldots \wedge s_{100}$ e $\phi = (\xi_1 \vee \xi_2) \wedge (\xi_3 \vee \xi_4)$
Então $p(\phi) = 208$ e $R = \{\xi_1, \xi_4\}$ é ótimo, com $p(\phi, R) = 108$.
Mas $R' = R - \{\xi_4\}$ não é ótimo, pois

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Contraexemplo:
$\xi_1 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$, $\xi_2 = q_1 \wedge q_2$, $\xi_3 = r_1 \wedge r_2$,
$\xi_4 = s_1 \wedge \ldots \wedge s_{100}$ e $\phi = (\xi_1 \vee \xi_2) \wedge (\xi_3 \vee \xi_4)$
Então $p(\phi) = 208$ e $R = \{\xi_1, \xi_4\}$ é ótimo, com $p(\phi, R) = 108$.
Mas $R' = R - \{\xi_4\}$ não é ótimo, pois

$$p(\phi, R')$$

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Contraexemplo:
$\xi_1 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$, $\xi_2 = q_1 \wedge q_2$, $\xi_3 = r_1 \wedge r_2$,
$\xi_4 = s_1 \wedge \ldots \wedge s_{100}$ e $\phi = (\xi_1 \vee \xi_2) \wedge (\xi_3 \vee \xi_4)$
Então $p(\phi) = 208$ e $R = \{\xi_1, \xi_4\}$ é ótimo, com $p(\phi, R) = 108$.
Mas $R' = R - \{\xi_4\}$ não é ótimo, pois

$$p(\phi, R') = p(\phi, \{\xi_1\})$$

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Contraexemplo:
$\xi_1 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$, $\xi_2 = q_1 \wedge q_2$, $\xi_3 = r_1 \wedge r_2$,
$\xi_4 = s_1 \wedge \ldots \wedge s_{100}$ e $\phi = (\xi_1 \vee \xi_2) \wedge (\xi_3 \vee \xi_4)$
Então $p(\phi) = 208$ e $R = \{\xi_1, \xi_4\}$ é ótimo, com $p(\phi, R) = 108$.
Mas $R' = R - \{\xi_4\}$ não é ótimo, pois

$$p(\phi, R') = p(\phi, \{\xi_1\}) = 206$$

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Contraexemplo:
$\xi_1 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$, $\xi_2 = q_1 \wedge q_2$, $\xi_3 = r_1 \wedge r_2$,
$\xi_4 = s_1 \wedge \ldots \wedge s_{100}$ e $\phi = (\xi_1 \vee \xi_2) \wedge (\xi_3 \vee \xi_4)$
Então $p(\phi) = 208$ e $R = \{\xi_1, \xi_4\}$ é ótimo, com $p(\phi, R) = 108$.
Mas $R' = R - \{\xi_4\}$ não é ótimo, pois

$$p(\phi, R') = p(\phi, \{\xi_1\}) = 206 > 110$$

Uma afirmação

Afirmação

Seja $R \subseteq SF(\phi)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas. Então $R - \{\psi\}$ é ótimo entre os que não consideram ψ e contêm no máximo $j - 1$ subfórmulas.

Contraexemplo:

$\xi_1 = p_1 \wedge p_2 \wedge p_3 \wedge p_4$, $\xi_2 = q_1 \wedge q_2$, $\xi_3 = r_1 \wedge r_2$,
$\xi_4 = s_1 \wedge \ldots \wedge s_{100}$ e $\phi = (\xi_1 \vee \xi_2) \wedge (\xi_3 \vee \xi_4)$
Então $p(\phi) = 208$ e $R = \{\xi_1, \xi_4\}$ é ótimo, com $p(\phi, R) = 108$.
Mas $R' = R - \{\xi_4\}$ não é ótimo, pois

$$p(\phi, R') = p(\phi, \{\xi_1\}) = 206 > 110 = p(\phi, \{\xi_3\})$$

Uma afirmação

Logo, a afirmação não é verdadeira.

Uma afirmação

Logo, a afirmação não é verdadeira.
Mas a usaremos como heurística!

Uma fórmula recursiva

Seja $SF(\phi) = \{\phi_1, ..., \phi_n\}$ e denote por $f(i, j)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas e consideram somente as subfórmulas em $\{\phi_1, ..., \phi_i\}$.

Uma fórmula recursiva

Seja $SF(\phi) = \{\phi_1, ..., \phi_n\}$ e denote por $f(i,j)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas e consideram somente as subfórmulas em $\{\phi_1, ..., \phi_i\}$. Então

$$f(i,0) = f(0,j) = \emptyset, \forall i,j$$

Uma fórmula recursiva

Seja $SF(\phi) = \{\phi_1, ..., \phi_n\}$ e denote por $f(i,j)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas e consideram somente as subfórmulas em $\{\phi_1, ..., \phi_i\}$. Então

$$f(i,0) = f(0,j) = \emptyset, \forall i,j$$

e

$$f(i,j) = \begin{cases} f(i-1,j-1) \cup \{\phi_i\} & \text{se } p(\phi, f(i-1,j-1) \cup \{\phi_i\}) < p(\phi, f(i-1,j)) \\ f(i-1,j) & \text{caso contrário} \end{cases}$$

Uma fórmula recursiva

Seja $SF(\phi) = \{\phi_1, ..., \phi_n\}$ e denote por $f(i,j)$ um renomeamento ótimo entre os que contêm no máximo j subfórmulas e consideram somente as subfórmulas em $\{\phi_1, ..., \phi_i\}$. Então

$$f(i,0) = f(0,j) = \emptyset, \forall i, j$$

e

$$f(i,j) = \begin{cases} f(i-1,j-1) \cup \{\phi_i\} & \text{se } p(\phi, f(i-1,j-1) \cup \{\phi_i\}) < p(\phi, f(i-1,j)) \\ f(i-1,j) & \text{caso contrário} \end{cases}$$

Queremos $f(n,n)$!

Uma implementação por computação ascendente

seja $dp[0..n]$ um novo arranjo com $dp[j] = \emptyset$ para todo j
para $i \leftarrow 1$ **até** n **faça**
    para $j \leftarrow n$ **descendo até** 1 **faça**
        $alt \leftarrow dp[j-1] \cup \{\phi_i\}$
        se $p(\phi, alt) < p(\phi, dp[j])$ **então**
            $dp[j] \leftarrow alt$
        fim se
    fim para
fim para

Uma implementação por computação ascendente

seja $dp[0..n]$ um novo arranjo com $dp[j] = \emptyset$ para todo j
para $i \leftarrow 1$ **até** n **faça**
  para $j \leftarrow n$ **descendo até** 1 **faça**
    $alt \leftarrow dp[j-1] \cup \{\phi_i\}$
    se $p(\phi, alt) < p(\phi, dp[j])$ **então**
      $dp[j] \leftarrow alt$
    fim se
  fim para
fim para

O custo de tempo no pior caso é $O(|SF(\phi)|^3)$.

Conjectura para árvores lineares

Conjectura

Se ϕ é uma árvore linear e $SF(\phi) = \{\phi_1, ..., \phi_n\}$, então $f(n, n)$ é ótimo.

Conjectura para árvores lineares

Conjectura

Se ϕ é uma árvore linear e $SF(\phi) = \{\phi_1, ..., \phi_n\}$, então $f(n, n)$ é ótimo.

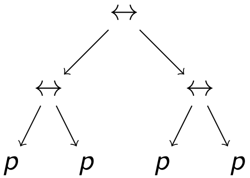
Apresentamos resultados experimentais para a conjectura na próxima seção.

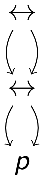
Conteúdo

Metodologia

Representações de fórmulas

$$(p \leftrightarrow p) \leftrightarrow (p \leftrightarrow p)$$



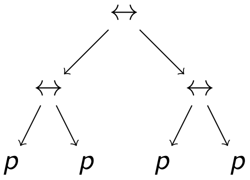


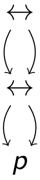
Metodologia

Representações de fórmulas

$$(p \leftrightarrow p) \leftrightarrow (p \leftrightarrow p)$$

Cadeia



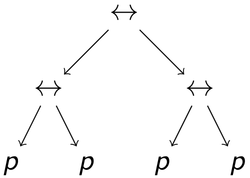


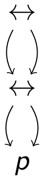
Metodologia

Representações de fórmulas

$$(p \leftrightarrow p) \leftrightarrow (p \leftrightarrow p)$$

Cadeia



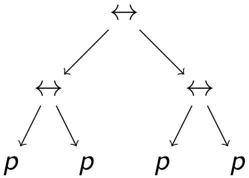


Árvore sintática

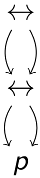
Metodologia
Representações de fórmulas

$$(p \leftrightarrow p) \leftrightarrow (p \leftrightarrow p)$$

Cadeia



Árvore sintática



DAG

Metodologia
Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

Metodologia
Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática

Metodologia
Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)

Metodologia
Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)
2. Conversão para FNN

Metodologia
Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)
2. Conversão para FNN (simplifica

Metodologia

Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)
2. Conversão para FNN (simplifica e permite testar a conjectura)

Metodologia
Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)
2. Conversão para FNN (simplifica e permite testar a conjectura)
3. Aplainamento

Metodologia
Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)
2. Conversão para FNN (simplifica e permite testar a conjectura)
3. Aplainamento $(p \vee (q \vee r)) \longmapsto p \vee q \vee r$

Metodologia

Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)
2. Conversão para FNN (simplifica e permite testar a conjectura)
3. Aplainamento ($p \vee (q \vee r) \longmapsto p \vee q \vee r$, mais simplificações)

Metodologia
Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)
2. Conversão para FNN (simplifica e permite testar a conjectura)
3. Aplainamento ($p \vee (q \vee r) \longmapsto p \vee q \vee r$, mais simplificações)
4. Conversão para DAG

Metodologia

Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)
2. Conversão para FNN (simplifica e permite testar a conjectura)
3. Aplainamento ($p \vee (q \vee r) \longmapsto p \vee q \vee r$, mais simplificações)
4. Conversão para DAG (de árvore para DAG)

Metodologia

Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)
2. Conversão para FNN (simplifica e permite testar a conjectura)
3. Aplainamento ($p \vee (q \vee r) \longmapsto p \vee q \vee r$, mais simplificações)
4. Conversão para DAG (de árvore para DAG)
5. Renomeamento

Metodologia
Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)
2. Conversão para FNN (simplifica e permite testar a conjectura)
3. Aplainamento ($p \vee (q \vee r) \longmapsto p \vee q \vee r$, mais simplificações)
4. Conversão para DAG (de árvore para DAG)
5. Renomeamento (Boy de la Tour e proposto)

Metodologia

Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)
2. Conversão para FNN (simplifica e permite testar a conjectura)
3. Aplainamento ($p \vee (q \vee r) \longmapsto p \vee q \vee r$, mais simplificações)
4. Conversão para DAG (de árvore para DAG)
5. Renomeamento (Boy de la Tour e proposto)
6. Conversão para FNC

Metodologia
Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)
2. Conversão para FNN (simplifica e permite testar a conjectura)
3. Aplainamento ($p \vee (q \vee r) \longmapsto p \vee q \vee r$, mais simplificações)
4. Conversão para DAG (de árvore para DAG)
5. Renomeamento (Boy de la Tour e proposto)
6. Conversão para FNC (opcional: simplificações)

Metodologia

Implementação

Foi implementado um programa em C++ 11 que realiza, em ordem, as seguintes transformações:

1. Análise sintática (de cadeia para árvore)
2. Conversão para FNN (simplifica e permite testar a conjectura)
3. Aplainamento ($p \vee (q \vee r) \longmapsto p \vee q \vee r$, mais simplificações)
4. Conversão para DAG (de árvore para DAG)
5. Renomeamento (Boy de la Tour e proposto)
6. Conversão para FNC (opcional: simplificações)
7. Conversão de DAG para cadeia

Metodologia

Implementação – Renomeamento

No algoritmo de Boy de la Tour, quando $\psi = \psi_1 \vee ... \vee \psi_n$:

Metodologia

Implementação – Renomeamento

No algoritmo de Boy de la Tour, quando $\psi = \psi_1 \vee ... \vee \psi_n$:

$$a^{\phi}_{\psi_i} = a^{\phi}_{\psi} \cdot \prod_{j \neq i} p(\psi_j)$$

Metodologia

Implementação – Renomeamento

No algoritmo de Boy de la Tour, quando $\psi = \psi_1 \vee ... \vee \psi_n$:

$$a^{\phi}_{\psi_i} = a^{\phi}_{\psi} \cdot \prod_{j \neq i} p(\psi_j)$$

Ao processar cada ψ_i, temos as seguintes alternativas:

Metodologia

Implementação – Renomeamento

No algoritmo de Boy de la Tour, quando $\psi = \psi_1 \vee ... \vee \psi_n$:

$$a^{\phi}_{\psi_i} = a^{\phi}_{\psi} \cdot \prod_{j \neq i} p(\psi_j)$$

Ao processar cada ψ_i, temos as seguintes alternativas:

1. Calcular $a^{\phi}_{\psi_i}$ com um laço.

Metodologia

Implementação – Renomeamento

No algoritmo de Boy de la Tour, quando $\psi = \psi_1 \vee ... \vee \psi_n$:

$$a^{\phi}_{\psi_i} = a^{\phi}_{\psi} \cdot \prod_{j \neq i} p(\psi_j)$$

Ao processar cada ψ_i, temos as seguintes alternativas:

1. Calcular $a^{\phi}_{\psi_i}$ com um laço.
2. Calcular $a^{\phi}_{\psi} \cdot \prod_j p(\psi_j)$ antes e dividir por $p(\psi_i)$.

Metodologia

Implementação – Renomeamento

No algoritmo de Boy de la Tour, quando $\psi = \psi_1 \vee ... \vee \psi_n$:

$$a^{\phi}_{\psi_i} = a^{\phi}_{\psi} \cdot \prod_{j \neq i} p(\psi_j)$$

Ao processar cada ψ_i, temos as seguintes alternativas:

1. Calcular $a^{\phi}_{\psi_i}$ com um laço.
2. Calcular $a^{\phi}_{\psi} \cdot \prod_j p(\psi_j)$ antes e dividir por $p(\psi_i)$.
3. Calcular uma tabela de sufixos antes e combinar com um prefixo atualizado após cada iteração.

Metodologia

Implementação – Renomeamento

No algoritmo de Boy de la Tour, quando $\psi = \psi_1 \vee \ldots \vee \psi_n$:

$$a^{\phi}_{\psi_i} = a^{\phi}_{\psi} \cdot \prod_{j \neq i} p(\psi_j)$$

Ao processar cada ψ_i, temos as seguintes alternativas:

1. Calcular $a^{\phi}_{\psi_i}$ com um laço.
2. Calcular $a^{\phi}_{\psi} \cdot \prod_j p(\psi_j)$ antes e dividir por $p(\psi_i)$.
3. Calcular uma tabela de sufixos antes e combinar com um prefixo atualizado após cada iteração. Fazemos assim!

Metodologia

Implementação – Renomeamento

Além disso, pode ocorrer:

Metodologia

Implementação – Renomeamento

Além disso, pode ocorrer:

$$
\begin{tikzcd}
 & \psi \arrow[dl] \arrow[d] \arrow[dr] & \\
\psi_1 \arrow[rr, bend right] & \psi_2 & \psi_3
\end{tikzcd}
$$

Metodologia

Implementação – Renomeamento

Além disso, pode ocorrer:

$$\begin{tikzcd}
 & \psi \arrow[dl] \arrow[d] \arrow[dr] & \\
\psi_1 \arrow[rr, bend right] & \psi_2 & \psi_3
\end{tikzcd}$$

Portanto, é necessária uma ordenação topológica:

Metodologia

Implementação – Renomeamento

Além disso, pode ocorrer:

$$
\begin{tikzcd}
 & \psi \arrow[dl] \arrow[d] \arrow[dr] & \\
\psi_1 \arrow[rr, bend right] & \psi_2 & \psi_3
\end{tikzcd}
$$

Portanto, é necessária uma ordenação topológica:

$$
\begin{tikzcd}
 & \psi \arrow[dl] \arrow[d] \arrow[dr] & \\
\psi_3 & \psi_1 \arrow[l] & \psi_2
\end{tikzcd}
$$

Metodologia

Experimentos propostos

Sobre um *benchmark* tradicional de 1200 fórmulas, foram executadas as seguintes combinações:

Metodologia

Experimentos propostos

Sobre um *benchmark* tradicional de 1200 fórmulas, foram executadas as seguintes combinações:

Combinação	1	2	3	4	5	6	7	8	9	10
Análise sintática	X	X	X	X	X	X	X	X	X	X
Conversão para FNN	X	X	X	X	X	X	X	X	X	X
Aplainamento	X	X	X	X	X	X	X	X	X	X
Conversão para DAG							X	X	X	X
Renomeamento			1	1	2	2	1	1	2	2
Conversão para FNC	1	2	1	2	1	2	1	2	1	2

Metodologia

Experimentos propostos

Sobre um *benchmark* tradicional de 1200 fórmulas, foram executadas as seguintes combinações:

Combinação	1	2	3	4	5	6	7	8	9	10
Análise sintática	X	X	X	X	X	X	X	X	X	X
Conversão para FNN	X	X	X	X	X	X	X	X	X	X
Aplainamento	X	X	X	X	X	X	X	X	X	X
Conversão para DAG							X	X	X	X
Renomeamento			1	1	2	2	1	1	2	2
Conversão para FNC	1	2	1	2	1	2	1	2	1	2

Em seguida, executamos um decisor de VAL baseado em FNC.

Resultados e análise

Combinações sem renomeamento

Combinação 1: sem renomeamento, sem simplificação
Combinação 2: sem renomeamento, com simplificação

Resultados e análise

Combinações sem renomeamento

Combinação 1: sem renomeamento, sem simplificação
Combinação 2: sem renomeamento, com simplificação

- Na Combinação 1, 73% excedeu limite de memória.

Resultados e análise

Combinações sem renomeamento

Combinação 1: sem renomeamento, sem simplificação
Combinação 2: sem renomeamento, com simplificação

- Na Combinação 1, 73% excedeu limite de memória.
- Na Combinação 2, 67% excedeu limite de memória e 1% excedeu limite de tempo.

Resultados e análise

Combinações sem renomeamento

Combinação 1: sem renomeamento, sem simplificação
Combinação 2: sem renomeamento, com simplificação

- Na Combinação 1, 73% excedeu limite de memória.
- Na Combinação 2, 67% excedeu limite de memória e 1% excedeu limite de tempo.
- Nos 27% em que a transformação terminou em C1 e C2, 5 fórmulas (menos de 1%) *não* ficaram menores em C2.

Resultados e análise

Combinações sem renomeamento

Combinação 1: sem renomeamento, sem simplificação
Combinação 2: sem renomeamento, com simplificação

- Na Combinação 1, 73% excedeu limite de memória.
- Na Combinação 2, 67% excedeu limite de memória e 1% excedeu limite de tempo.
- Nos 27% em que a transformação terminou em C1 e C2, 5 fórmulas (menos de 1%) *não* ficaram menores em C2.

Simplificação é bom.

Resultados e análise

Combinações sem renomeamento

Combinação 1: sem renomeamento, sem simplificação
Combinação 2: sem renomeamento, com simplificação

- Na Combinação 1, 73% excedeu limite de memória.
- Na Combinação 2, 67% excedeu limite de memória e 1% excedeu limite de tempo.
- Nos 27% em que a transformação terminou em C1 e C2, 5 fórmulas (menos de 1%) *não* ficaram menores em C2.

Simplificação é bom. Mas não é suficiente!

Resultados e análise

Testando a conjectura para árvores lineares

Combinação 3: árvore, Boy de la Tour, sem simplificação
Combinação 5: árvore, algoritmo proposto, sem simplificação

Resultados e análise

Testando a conjectura para árvores lineares

Combinação 3: árvore, Boy de la Tour, sem simplificação
Combinação 5: árvore, algoritmo proposto, sem simplificação
A transformação terminou em C3 e C5 para 49% das fórmulas.

Resultados e análise

Testando a conjectura para árvores lineares

Combinação 3: árvore, Boy de la Tour, sem simplificação
Combinação 5: árvore, algoritmo proposto, sem simplificação
A transformação terminou em C3 e C5 para 49% das fórmulas.
Nestes 49%, Boy de la Tour e o algoritmo proposto produziram o mesmo número de cláusulas.

Resultados e análise

Comparações entre árvores e DAGs

Combinações $3, 4, 5, 6$: árvore
Combinações $7, 8, 9, 10$: DAG

Resultados e análise
Comparações entre árvores e DAGs

Combinações $3, 4, 5, 6$: árvore
Combinações $7, 8, 9, 10$: DAG

Comparando número de cláusulas.

	$C_3 \times C_7$	$C_4 \times C_8$	$C_5 \times C_9$	$C_6 \times C_{10}$	
C_i foi melhor em	0 (0%)	5 (0%)	0 (0%)	6 (1%)	fórmulas.
C_{i+4} foi melhor em	179 (15%)	177 (15%)	367 (31%)	358 (30%)	fórmulas.

Resultados e análise

Comparações entre árvores e DAGs

Combinações $3, 4, 5, 6$: árvore
Combinações $7, 8, 9, 10$: DAG

Comparando número de cláusulas.

	$C_3 \times C_7$	$C_4 \times C_8$	$C_5 \times C_9$	$C_6 \times C_{10}$	
C_i foi melhor em	0 (0%)	5 (0%)	0 (0%)	6 (1%)	fórmulas.
C_{i+4} foi melhor em	179 (15%)	177 (15%)	367 (31%)	358 (30%)	fórmulas.

Claro, DAGs simplesmente permitem renomeamento global!

Resultados e análise

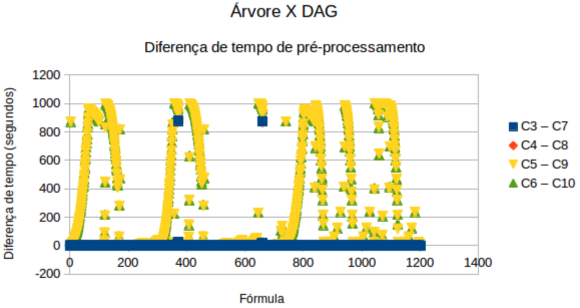
Comparações entre árvores e DAGs

Combinações $3, 4, 5, 6$: árvore
Combinações $7, 8, 9, 10$: DAG

Resultados e análise

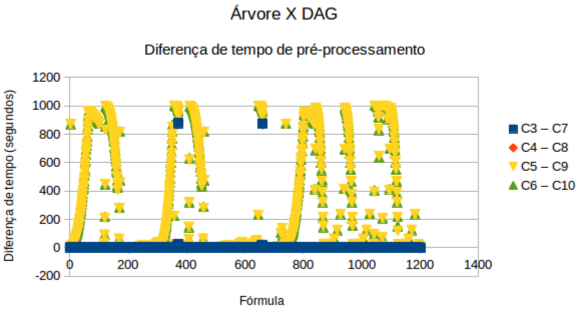
Comparações entre árvores e DAGs

Combinações 3, 4, 5, 6: árvore
Combinações 7, 8, 9, 10: DAG



Resultados e análise
Comparações entre árvores e DAGs

Combinações $3, 4, 5, 6$: árvore
Combinações $7, 8, 9, 10$: DAG



Claro, DAG é uma estrutura mais compacta!

Resultados e análise

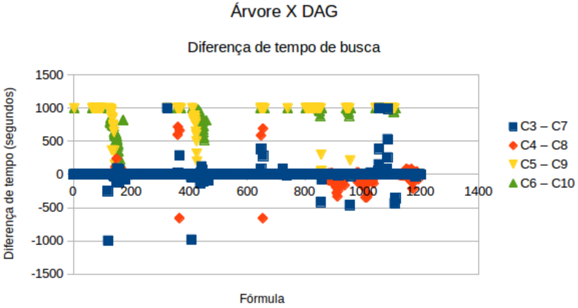
Comparações entre árvores e DAGs

Combinações $3, 4, 5, 6$: árvore
Combinações $7, 8, 9, 10$: DAG

Resultados e análise

Comparações entre árvores e DAGs

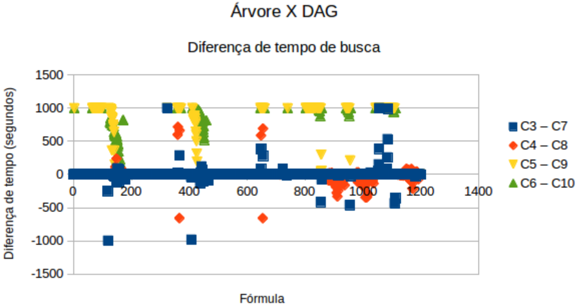
Combinações $3, 4, 5, 6$: árvore
Combinações $7, 8, 9, 10$: DAG



Resultados e análise

Comparações entre árvores e DAGs

Combinações $3, 4, 5, 6$: árvore
Combinações $7, 8, 9, 10$: DAG

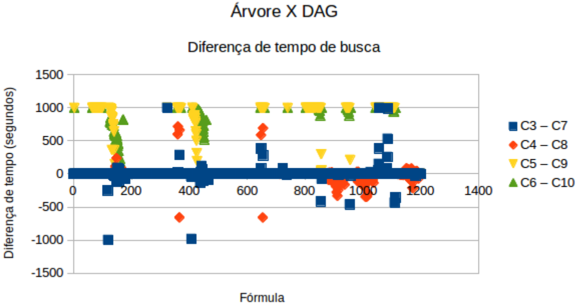


Primeiro indício!

Resultados e análise

Comparações entre árvores e DAGs

Combinações $3, 4, 5, 6$: árvore
Combinações $7, 8, 9, 10$: DAG



Primeiro indício! Converter para DAG é essencial.

Resultados e análise

Comparações entre os algoritmos de renomeamento

Combinações $7, 8$: Boy de la Tour, sem e com simplificação
Combinações $9, 10$: Algoritmo proposto, sem e com simplificação

Resultados e análise

Comparações entre os algoritmos de renomeamento

Combinações $7, 8$: Boy de la Tour, sem e com simplificação
Combinações $9, 10$: Algoritmo proposto, sem e com simplificação

Comparando número de cláusulas.

Resultados e análise
Comparações entre os algoritmos de renomeamento

Combinações $7, 8$: Boy de la Tour, sem e com simplificação
Combinações $9, 10$: Algoritmo proposto, sem e com simplificação

Comparando número de cláusulas.

- A transformação terminou em C7 e C9 para 73% das fórmulas.

Resultados e análise

Comparações entre os algoritmos de renomeamento

Combinações $7, 8$: Boy de la Tour, sem e com simplificação
Combinações $9, 10$: Algoritmo proposto, sem e com simplificação

Comparando número de cláusulas.

- A transformação terminou em C7 e C9 para 73% das fórmulas.
- Em 3%, a transformação terminou em C7 e C9 e C7 produziu menos cláusulas, com $\max\{|C7 - C9|\} = 3$.

Resultados e análise

Comparações entre os algoritmos de renomeamento

Combinações $7, 8$: Boy de la Tour, sem e com simplificação
Combinações $9, 10$: Algoritmo proposto, sem e com simplificação

Comparando número de cláusulas.

- A transformação terminou em C7 e C9 para 73% das fórmulas.
- Em 3%, a transformação terminou em C7 e C9 e C7 produziu menos cláusulas, com $\max\{|C7 - C9|\} = 3$.
- Em 8%, a transformação terminou em C7 e C9 e C9 produziu menos cláusulas, com $\max\{|C7 - C9|\} = 1.572.786$, onde C9 produziu 78 cláusulas.

Resultados e análise

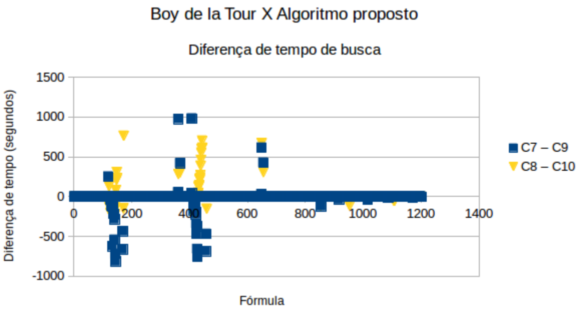
Comparações entre os algoritmos de renomeamento

Combinações $7, 8$: Boy de la Tour, sem e com simplificação
Combinações $9, 10$: Algoritmo proposto, sem e com simplificação

Resultados e análise

Comparações entre os algoritmos de renomeamento

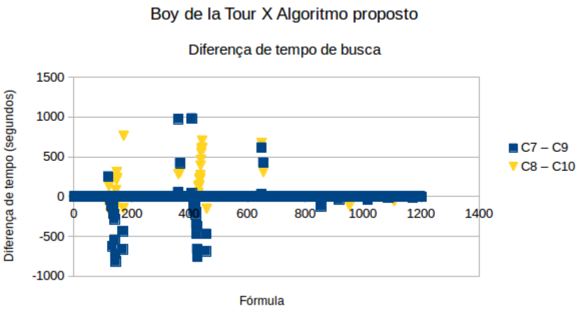
Combinações $7, 8$: Boy de la Tour, sem e com simplificação
Combinações $9, 10$: Algoritmo proposto, sem e com simplificação



Resultados e análise

Comparações entre os algoritmos de renomeamento

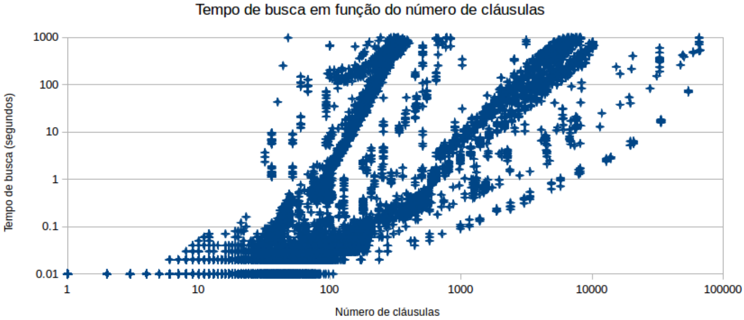
Combinações $7, 8$: Boy de la Tour, sem e com simplificação
Combinações $9, 10$: Algoritmo proposto, sem e com simplificação



Cada algoritmo leva vantagem em famílias de fórmulas específicas.

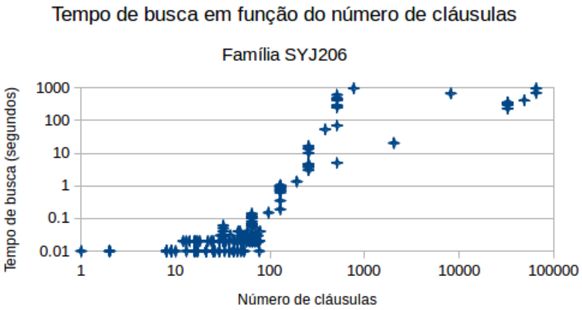
Resultados e análise

Tempo de busca em função do número de cláusulas



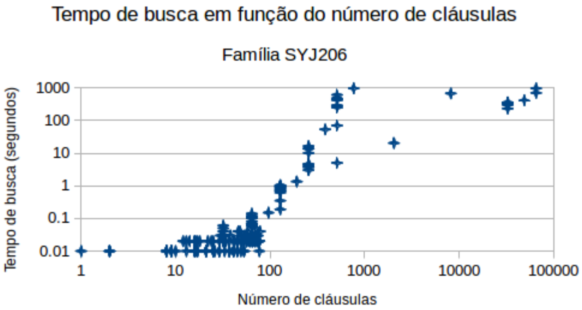
Resultados e análise

Tempo de busca em função do número de cláusulas



Resultados e análise

Tempo de busca em função do número de cláusulas



Então, sim!

Conteúdo

Fórmulas com menos cláusulas produzem respostas mais rápido?

Fórmulas com menos cláusulas produzem respostas mais rápido?

- Revisamos técnicas para reduzir o número de cláusulas [10, 12, 11].

Fórmulas com menos cláusulas produzem respostas mais rápido?

- Revisamos técnicas para reduzir o número de cláusulas [10, 12, 11].
- Desenvolvemos um algoritmo de programação dinâmica para este problema.

Fórmulas com menos cláusulas produzem respostas mais rápido?

- Revisamos técnicas para reduzir o número de cláusulas [10, 12, 11].
- Desenvolvemos um algoritmo de programação dinâmica para este problema.
- Propomos experimentos para:

Fórmulas com menos cláusulas produzem respostas mais rápido?

- Revisamos técnicas para reduzir o número de cláusulas [10, 12, 11].
- Desenvolvemos um algoritmo de programação dinâmica para este problema.
- Propomos experimentos para:
  - Verificar uma propriedade de optimalidade restrita do algoritmo proposto.

Fórmulas com menos cláusulas produzem respostas mais rápido?

- Revisamos técnicas para reduzir o número de cláusulas [10, 12, 11].
- Desenvolvemos um algoritmo de programação dinâmica para este problema.
- Propomos experimentos para:
  - Verificar uma propriedade de optimalidade restrita do algoritmo proposto.
  - Comparar o algoritmo proposto com um outro.

Fórmulas com menos cláusulas produzem respostas mais rápido?

- Revisamos técnicas para reduzir o número de cláusulas [10, 12, 11].
- Desenvolvemos um algoritmo de programação dinâmica para este problema.
- Propomos experimentos para:
  - Verificar uma propriedade de optimalidade restrita do algoritmo proposto.
  - Comparar o algoritmo proposto com um outro.
  - Tentar responder à pergunta.

Principais destaques

- Tentar reduzir o tamanho de uma fórmula compensa.

Principais destaques

- Tentar reduzir o tamanho de uma fórmula compensa.
- Para este problema, DAGs são melhores que árvores.

Principais destaques

- Tentar reduzir o tamanho de uma fórmula compensa.
- Para este problema, DAGs são melhores que árvores.
- É provável que nossa conjectura para árvores lineares esteja correta.

Principais destaques

- Tentar reduzir o tamanho de uma fórmula compensa.
- Para este problema, DAGs são melhores que árvores.
- É provável que nossa conjectura para árvores lineares esteja correta. (trabalho futuro!)

Principais destaques

- Tentar reduzir o tamanho de uma fórmula compensa.
- Para este problema, DAGs são melhores que árvores.
- É provável que nossa conjectura para árvores lineares esteja correta. (trabalho futuro!)
- Diferentes algoritmos de renomeamento levam vantagem em diferentes famílias de fórmulas.

Principais destaques

- Tentar reduzir o tamanho de uma fórmula compensa.
- Para este problema, DAGs são melhores que árvores.
- É provável que nossa conjectura para árvores lineares esteja correta. (trabalho futuro!)
- Diferentes algoritmos de renomeamento levam vantagem em diferentes famílias de fórmulas. Nas famílias *SYJ206* e *SYJ212*, o nosso gera muito menos claúsulas!

Principais destaques

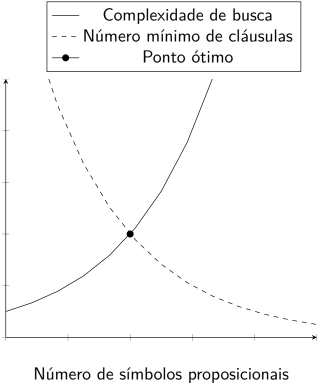
- Tentar reduzir o tamanho de uma fórmula compensa.
- Para este problema, DAGs são melhores que árvores.
- É provável que nossa conjectura para árvores lineares esteja correta. (trabalho futuro!)
- Diferentes algoritmos de renomeamento levam vantagem em diferentes famílias de fórmulas. Nas famílias *SYJ206* e *SYJ212*, o nosso gera muito menos claúsulas!
- Considerando fórmulas parecidas, as com menos cláusulas tendem a dar resposta mais rápido.

Trabalhos futuros

Para melhorar ainda mais o desempenho na etapa de busca:

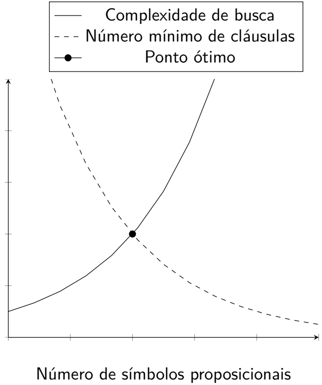
Trabalhos futuros

Para melhorar ainda mais o desempenho na etapa de busca:



Trabalhos futuros

Para melhorar ainda mais o desempenho na etapa de busca:



Também reduz a complexidade do algoritmo proposto!

Trabalhos futuros

$$f(i,j) = \begin{cases} f(i-1,j-1) \cup \{\phi_i\} & \text{se } p(\phi, f(i-1,j-1) \cup \{\phi_i\}) < p(\phi, f(i-1,j)) \\ f(i-1,j) & \text{caso contrário} \end{cases}$$

Trabalhos futuros

$$f(i,j) = \begin{cases} f(i-1,j-1) \cup \{\phi_i\} & \text{se } p(\phi, f(i-1,j-1) \cup \{\phi_i\}) < p(\phi, f(i-1,j)) \\ f(i-1,j) & \text{caso contrário} \end{cases}$$

Adaptar para outras formas normais!

Conteúdo

Referências I

[1] R. Bloem, U. Egly, P. Klampfl, R. Könighofer, and F. Lonsing, "SAT-based methods for circuit synthesis," in *Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design*, pp. 31–34, FMCAD Inc, 2014.

[2] R. Nieuwenhuis and A. Oliveras, "On SAT modulo theories and optimization problems," in *Theory and Applications of Satisfiability Testing-SAT 2006*, pp. 156–169, Springer, 2006.

[3] A. Gupta, M. K. Ganai, and C. Wang, "SAT-based verification methods and applications in hardware verification," in *Formal Methods for Hardware Verification*, pp. 108–143, Springer, 2006.

Referências II

[4] J. Harrison, *Handbook of practical logic and automated reasoning*.
Cambridge University Press, 2009.

[5] E. J. Horvitz, *Automated reasoning for biology and medicine*.
Knowledge Systems Laboratory, Section on Medical Informatics, Stanford University, 1992.

[6] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*, pp. 151–158, ACM, 1971.

Referências III

[7] M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of the ACM (JACM)*, vol. 7, no. 3, pp. 201–215, 1960.

[8] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[9] A. Biere, M. Heule, H. van Maaren, and T. Walsh, "Conflict-driven clause learning SAT solvers," *Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications*, pp. 131–153, 2009.

Referências IV

[10] T. Boy de la Tour, "An optimality result for clause form translation," *Journal of Symbolic Computation*, vol. 14, no. 4, pp. 283–301, 1992.

[11] P. Jackson and D. Sheridan, "Clause form conversions for boolean circuits," in *Theory and applications of satisfiability testing*, pp. 183–198, Springer, 2004.

[12] A. Nonnengart and C. Weidenbach, "Computing small clause normal forms.," *Handbook of automated reasoning*, vol. 1, pp. 335–367, 2001.

Obrigado!

matheuscscp@gmail.com

Problemas da lógica proposicional

Seja A_{UNSAT} um algoritmo para UNSAT e $R_{\text{VAL}} =$
"Sobre a entrada $\phi \in \mathcal{L}$: Dê a resposta de A_{UNSAT} sobre $\neg\phi$."

ϕ	$\neg\phi$	$A_{\text{UNSAT}}(\neg\phi)$	$R_{\text{VAL}}(\phi)$
Tautologia	Contradição	Sim	Sim
Contradição	Tautologia	Não	Não
Contingência	Contingência	Não	Não

Problemas da lógica proposicional

Seja A_{UNSAT} um algoritmo para UNSAT e $R_{\text{VAL}} =$
"Sobre a entrada $\phi \in \mathcal{L}$: Dê a resposta de A_{UNSAT} sobre $\neg\phi$."

ϕ	$\neg\phi$	$A_{\text{UNSAT}}(\neg\phi)$	$R_{\text{VAL}}(\phi)$
Tautologia	Contradição	Sim	Sim
Contradição	Tautologia	Não	Não
Contingência	Contingência	Não	Não

Seja A_{VAL} um algoritmo para VAL e $R_{\text{UNSAT}} =$
"Sobre a entrada $\phi \in \mathcal{L}$: Dê a resposta de A_{VAL} sobre $\neg\phi$."

ϕ	$\neg\phi$	$A_{\text{VAL}}(\neg\phi)$	$R_{\text{UNSAT}}(\phi)$
Tautologia	Contradição	Não	Não
Contradição	Tautologia	Sim	Sim
Contingência	Contingência	Não	Não