ROZUMOWANIE, ARGUMENTACJA, DOWÓD

Andrzej Indrzejczak

Katedra Logiki i Metodologii Nauk UŁ

Łódź, semestr letni 2008/2009

JAK ZNALEŹĆ DOWÓD?

Dwa nurty badań:

JAK ZNALEŹĆ DOWÓD?

Dwa nurty badań:

1. algorytmy $\Longrightarrow$ mechanizacja rozumowania, automatyczne dowodzenie twierdzeń

JAK ZNALEŹĆ DOWÓD?

Dwa nurty badań:

1. algorytmy $\Longrightarrow$ mechanizacja rozumowania, automatyczne dowodzenie twierdzeń
2. heurystyka $\Longrightarrow$ badanie strategii szukania dowodu, "reguły" odkrycia

JAK ZNALEŹĆ DOWÓD?

Dwa nurty badań:

1. algorytmy $\Longrightarrow$ mechanizacja rozumowania, automatyczne dowodzenie twierdzeń
2. heurystyka $\Longrightarrow$ badanie strategii szukania dowodu, "reguły" odkrycia

ad. 1 – od lat 60-tych jest to jedna z gałęzi szeroko rozumianej AI

JAK ZNALEŹĆ DOWÓD?

Dwa nurty badań:

1. algorytmy $\Longrightarrow$ mechanizacja rozumowania, automatyczne dowodzenie twierdzeń
2. heurystyka $\Longrightarrow$ badanie strategii szukania dowodu, "reguły" odkrycia

ad. 1 – od lat 60-tych jest to jedna z gałęzi szeroko rozumianej AI
ad. 2 – współcześnie przede wszystkim domena kognitywistyki

JAK ZNALEŹĆ DOWÓD?

Dwa nurty badań:

1. algorytmy $\Longrightarrow$ mechanizacja rozumowania, automatyczne dowodzenie twierdzeń
2. heurystyka $\Longrightarrow$ badanie strategii szukania dowodu, "reguły" odkrycia

ad. 1 – od lat 60-tych jest to jedna z gałęzi szeroko rozumianej AI
ad. 2 – współcześnie przede wszystkim domena kognitywistyki

Uwaga: Szukanie dowodu to szczególny przypadek badań nad rozwiązywaniem problemów.

PROBLEMY ROZWIĄZYWALNE

Rozwiązywalność:

PROBLEMY ROZWIĄZYWALNE

Rozwiązywalność:

Problem (klasa problemów) jest rozwiązywalny (solvable) wtw istnieje efektywna procedura (sposób) jego rozwiązania.

PROBLEMY ROZWIĄZYWALNE

Rozwiązywalność:

Problem (klasa problemów) jest rozwiązywalny (solvable) wtw istnieje efektywna procedura (sposób) jego rozwiązania.
Uwaga: współcześnie dominuje tendencja do sprowadzania kwestii rozwiązywalności problemu do matematycznej kwestii obliczalności funkcji (relacji, zbioru) [por. dalej] ale pojęcie efektywnej metody ma szerszy zasięg, dotyczy m.in.:

PROBLEMY ROZWIĄZYWALNE

Rozwiązywalność:

Problem (klasa problemów) jest rozwiązywalny (solvable) wtw istnieje efektywna procedura (sposób) jego rozwiązania.
Uwaga: współcześnie dominuje tendencja do sprowadzania kwestii rozwiązywalności problemu do matematycznej kwestii obliczalności funkcji (relacji, zbioru) [por. dalej] ale pojęcie efektywnej metody ma szerszy zasięg, dotyczy m.in.:

- metod analizy chemicznej

PROBLEMY ROZWIĄZYWALNE

Rozwiązywalność:

Problem (klasa problemów) jest rozwiązywalny (solvable) wtw istnieje efektywna procedura (sposób) jego rozwiązania.
Uwaga: współcześnie dominuje tendencja do sprowadzania kwestii rozwiązywalności problemu do matematycznej kwestii obliczalności funkcji (relacji, zbioru) [por. dalej] ale pojęcie efektywnej metody ma szerszy zasięg, dotyczy m.in.:

- metod analizy chemicznej
- diagnostyki medycznej

PROBLEMY ROZWIĄZYWALNE

Rozwiązywalność:

Problem (klasa problemów) jest rozwiązywalny (solvable) wtw istnieje efektywna procedura (sposób) jego rozwiązania.
Uwaga: współcześnie dominuje tendencja do sprowadzania kwestii rozwiązywalności problemu do matematycznej kwestii obliczalności funkcji (relacji, zbioru) [por. dalej] ale pojęcie efektywnej metody ma szerszy zasięg, dotyczy m.in.:

- metod analizy chemicznej
- diagnostyki medycznej
- fizycznego pomiaru

PROBLEMY ROZWIĄZYWALNE

Rozwiązywalność:

Problem (klasa problemów) jest rozwiązywalny (solvable) wtw istnieje efektywna procedura (sposób) jego rozwiązania.
Uwaga: współcześnie dominuje tendencja do sprowadzania kwestii rozwiązywalności problemu do matematycznej kwestii obliczalności funkcji (relacji, zbioru) [por. dalej] ale pojęcie efektywnej metody ma szerszy zasięg, dotyczy m.in.:

- metod analizy chemicznej
- diagnostyki medycznej
- fizycznego pomiaru

Efektywność metody jest pojęciem stopniowalnym; najwyższy stopień efektywności to rozstrzygalność.

PROBLEMY ROZWIĄZYWALNE

Rozstrzygalność:

PROBLEMY ROZWIĄZYWALNE

Rozstrzygalność:

Szczególny przypadek rozwiązywania (klas) problemów – odpowiedzi na pytania typu:

PROBLEMY ROZWIĄZYWALNE

Rozstrzygalność:

Szczególny przypadek rozwiązywania (klas) problemów – odpowiedzi na pytania typu:
Czy $x \in K$ ma (nie ma) własność P?

PROBLEMY ROZWIĄZYWALNE

Rozstrzygalność:

Szczególny przypadek rozwiązywania (klas) problemów – odpowiedzi na pytania typu:
Czy $x \in K$ ma (nie ma) własność P?
np. czy dana formuła jest twierdzeniem teorii?

PROBLEMY ROZWIĄZYWALNE

Rozstrzygalność:

Szczególny przypadek rozwiązywania (klas) problemów – odpowiedzi na pytania typu:
Czy $x \in K$ ma (nie ma) własność P?
np. czy dana formuła jest twierdzeniem teorii?
Problem ew. własność P jest rozstrzygalna (decidable) w klasie K wtw. istnieje efektywna procedura znajdowania odpowiedzi na to pytanie.

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

1. skończoność działania

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

1. skończoność działania
2. mechaniczność kroków

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

1. skończoność działania
2. mechaniczność kroków
3. jednoznaczność (i skończoność) opisu

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

1. skończoność działania
2. mechaniczność kroków
3. jednoznaczność (i skończoność) opisu
4. elementarność kroków

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

1. skończoność działania
2. mechaniczność kroków
3. jednoznaczność (i skończoność) opisu
4. elementarność kroków
5. dyskretność kroków

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

6 powtarzalność

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

6 powtarzalność

7 determinizm

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

6. powtarzalność
7. determinizm
8. poprawność

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

6. powtarzalność
7. determinizm
8. poprawność
9. uniwersalność (ogólność, nieselektywność) zastosowań

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

6. powtarzalność
7. determinizm
8. poprawność
9. uniwersalność (ogólność, nieselektywność) zastosowań
10. wykonalność

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 1 daje odpowiedź w skończonej liczbie kroków

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 1 daje odpowiedź w skończonej liczbie kroków (ale algorytmy nie zatrzymujące się)

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 1 daje odpowiedź w skończonej liczbie kroków (ale algorytmy nie zatrzymujące się)
ad 2 przepis realizowalny bez użycia wyobraźni a nawet bez potrzeby rozumienia wykonywanych operacji

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 1 daje odpowiedź w skończonej liczbie kroków (ale algorytmy nie zatrzymujące się)
ad 2 przepis realizowalny bez użycia wyobraźni a nawet bez potrzeby rozumienia wykonywanych operacji
ad 3 i 4 prostota opisu sprowadzonego do kombinacji skończonej ilości elementarnych operacji

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 1 daje odpowiedź w skończonej liczbie kroków (ale algorytmy nie zatrzymujące się)
ad 2 przepis realizowalny bez użycia wyobraźni a nawet bez potrzeby rozumienia wykonywanych operacji
ad 3 i 4 prostota opisu sprowadzonego do kombinacji skończonej ilości elementarnych operacji (uwaga! pojęcie elementarności relatywne – poziomy języków opisu)

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 1 daje odpowiedź w skończonej liczbie kroków (ale algorytmy nie zatrzymujące się)
ad 2 przepis realizowalny bez użycia wyobraźni a nawet bez potrzeby rozumienia wykonywanych operacji
ad 3 i 4 prostota opisu sprowadzonego do kombinacji skończonej ilości elementarnych operacji (uwaga! pojęcie elementarności relatywne – poziomy języków opisu)
ad 5 poszczególne kroki są separowalne

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 1 daje odpowiedź w skończonej liczbie kroków (ale algorytmy nie zatrzymujące się)
ad 2 przepis realizowalny bez użycia wyobraźni a nawet bez potrzeby rozumienia wykonywanych operacji
ad 3 i 4 prostota opisu sprowadzonego do kombinacji skończonej ilości elementarnych operacji (uwaga! pojęcie elementarności relatywne – poziomy języków opisu)
ad 5 poszczególne kroki są separowalne
ad 6 metoda wykonywalna w każdych okolicznościach

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 7 jednoznaczne określenie zarówno kolejności jak i wyniku każdego kroku

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 7 jednoznaczne określenie zarówno kolejności jak i wyniku każdego kroku (ale współcześnie również algorytmy niedeterministyczne i probabilistyczne)

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 7 jednoznaczne określenie zarówno kolejności jak i wyniku każdego kroku (ale współcześnie również algorytmy niedeterministyczne i probabilistyczne)
ad 8 daje tylko poprawne rozwiązania

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 7 jednoznaczne określenie zarówno kolejności jak i wyniku każdego kroku (ale współcześnie również algorytmy niedeterministyczne i probabilistyczne)
ad 8 daje tylko poprawne rozwiązania
ad 9 działa dla każdego $x \in K$ (daje zawsze odpowiedź)

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 7 jednoznaczne określenie zarówno kolejności jak i wyniku każdego kroku (ale współcześnie również algorytmy niedeterministyczne i probabilistyczne)
ad 8 daje tylko poprawne rozwiązania
ad 9 działa dla każdego $x \in K$ (daje zawsze odpowiedź) (ale algorytmy częściowe)

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 7 jednoznaczne określenie zarówno kolejności jak i wyniku każdego kroku (ale współcześnie również algorytmy niedeterministyczne i probabilistyczne)
ad 8 daje tylko poprawne rozwiązania
ad 9 działa dla każdego $x \in K$ (daje zawsze odpowiedź) (ale algorytmy częściowe)
ad 10 podkreśla się, że na efektywną procedurę nie nakłada się realnych ograniczeń wykonania dotyczących czasu czy pamięci – teoretyczna wykonalność

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 7 jednoznaczne określenie zarówno kolejności jak i wyniku każdego kroku (ale współcześnie również algorytmy niedeterministyczne i probabilistyczne)
ad 8 daje tylko poprawne rozwiązania
ad 9 działa dla każdego $x \in K$ (daje zawsze odpowiedź) (ale algorytmy częściowe)
ad 10 podkreśla się, że na efektywną procedurę nie nakłada się realnych ograniczeń wykonania dotyczących czasu czy pamięci – teoretyczna wykonalność (ale współcześnie raczej nacisk na praktyczną realizowalność (tractability, feasibility) i wydajność (efficiency) algorytmów).

KRYTERIA EFEKTYWNOŚCI

Co to znaczy być metodą efektywną?

ad 7 jednoznaczne określenie zarówno kolejności jak i wyniku każdego kroku (ale współcześnie również algorytmy niedeterministyczne i probabilistyczne)
ad 8 daje tylko poprawne rozwiązania
ad 9 działa dla każdego $x \in K$ (daje zawsze odpowiedź) (ale algorytmy częściowe)
ad 10 podkreśla się, że na efektywną procedurę nie nakłada się realnych ograniczeń wykonania dotyczących czasu czy pamięci – teoretyczna wykonalność (ale współcześnie raczej nacisk na praktyczną realizowalność (tractability, feasibility) i wydajność (efficiency) algorytmów).
W teorii nie jest też ważne na czym procedura jest realizowana (materialny aspekt) – choć w praktyce tak.

METODY EFEKTYWNE

Ewolucja problematyki – zasadnicze etapy:

METODY EFEKTYWNE

Ewolucja problematyki – zasadnicze etapy:

1. starożytność – czasy nowożytne: konstrukcja wielu algorytmów

METODY EFEKTYWNE

Ewolucja problematyki – zasadnicze etapy:

1. starożytność – czasy nowożytne: konstrukcja wielu algorytmów
2. czasy nowożytne – powstanie i ewolucja maszyn

METODY EFEKTYWNE

Ewolucja problematyki – zasadnicze etapy:

1. starożytność – czasy nowożytne: konstrukcja wielu algorytmów
2. czasy nowożytne – powstanie i ewolucja maszyn
3. XX w. – program Hilberta, okres dowiedzenia twierdzeń limitacyjnych i intensywnych badań nad rozstrzygalnością (teoretyczną) teorii $\Longrightarrow$ teoria rozstrzygalności i obliczalności

METODY EFEKTYWNE

Ewolucja problematyki – zasadnicze etapy:

1. starożytność – czasy nowożytne: konstrukcja wielu algorytmów
2. czasy nowożytne – powstanie i ewolucja maszyn
3. XX w. – program Hilberta, okres dowiedzenia twierdzeń limitacyjnych i intensywnych badań nad rozstrzygalnością (teoretyczną) teorii $\Longrightarrow$ teoria rozstrzygalności i obliczalności
4. od lat 50tych XX w. – komputery: badanie nad wydajnością algorytmów $\Longrightarrow$ teoria złożoności (rozstrzygalność praktyczna i automatyzacja)

METODY EFEKTYWNE

Trzy współczesne tradycje badań nad efektywnymi procedurami:

METODY EFEKTYWNE

Trzy współczesne tradycje badań nad efektywnymi procedurami:

1. uściślenie pojęcia algorytmu

METODY EFEKTYWNE

Trzy współczesne tradycje badań nad efektywnymi procedurami:

1. uściślenie pojęcia algorytmu
2. zdefiniowanie teoretycznego pojęcia maszyny (automatu)

METODY EFEKTYWNE

Trzy współczesne tradycje badań nad efektywnymi procedurami:

1. uściślenie pojęcia algorytmu
2. zdefiniowanie teoretycznego pojęcia maszyny (automatu)
3. obliczalność – sprowadzenie problemu do jego reprezentacji arytmetycznej

ALGORYTMY

Historyczne przykłady:

ALGORYTMY

Historyczne przykłady:

- algorytm Euklidesa znajdowania największego wspólnego dzielnika

ALGORYTMY

Historyczne przykłady:

- algorytm Euklidesa znajdowania największego wspólnego dzielnika
- IX w. Al Chwarizmi – ojciec chrzestny terminu algorytm

ALGORYTMY

Historyczne przykłady:

- algorytm Euklidesa znajdowania największego wspólnego dzielnika
- IX w. Al Chwarizmi – ojciec chrzestny terminu algorytm
- XIV w. – określenie algorytm jako nazwa przepisu wykonywania 4 działań arytmetycznych w notacji 10-tnej

ALGORYTMY

Historyczne przykłady:

- algorytm Euklidesa znajdowania największego wspólnego dzielnika
- IX w. Al Chwarizmi – ojciec chrzestny terminu algorytm
- XIV w. – określenie algorytm jako nazwa przepisu wykonywania 4 działań arytmetycznych w notacji 10-tnej
- XV w. – Pons asinorum Tartaretusa

ALGORYTMY

Historyczne przykłady:

- algorytm Euklidesa znajdowania największego wspólnego dzielnika
- IX w. Al Chwarizmi – ojciec chrzestny terminu algorytm
- XIV w. – określenie algorytm jako nazwa przepisu wykonywania 4 działań arytmetycznych w notacji 10-tnej
- XV w. – Pons asinorum Tartaretusa
- XV i XVI w. – Tartaglia, Vieta i inni; algorytmy rozwiązywania równań algebraicznych

ALGORYTMY

Opis algorytmu – ważniejsze współczesne konstrukcje teoretyczne:

ALGORYTMY

Opis algorytmu – ważniejsze współczesne konstrukcje teoretyczne:

1. systemy Thuego (1910)

ALGORYTMY

Opis algorytmu – ważniejsze współczesne konstrukcje teoretyczne:

1. systemy Thuego (1910)
2. języki Posta (1920-1943)

ALGORYTMY

Opis algorytmu – ważniejsze współczesne konstrukcje teoretyczne:

1. systemy Thuego (1910)
2. języki Posta (1920-1943)
3. algorytmy normalne Markova (1951)

ALGORYTMY

Współczesne językowe (praktyczne) realizacje algorytmów:

ALGORYTMY

Współczesne językowe (praktyczne) realizacje algorytmów:

Program – algorytm wyrażony w języku przekładalnym na kod maszynowy komputera

ALGORYTMY

Współczesne językowe (praktyczne) realizacje algorytmów:

Program – algorytm wyrażony w języku przekładalnym na kod maszynowy komputera

1. języki niskiego poziomu (asemblery, kompilatory, interpretatory)

ALGORYTMY

Współczesne językowe (praktyczne) realizacje algorytmów:

Program – algorytm wyrażony w języku przekładalnym na kod maszynowy komputera

1. języki niskiego poziomu (asemblery, kompilatory, interpretatory)
2. języki wysokiego poziomu (implementacja algorytmu)

ALGORYTMY

Współczesne językowe (praktyczne) realizacje algorytmów:

Program – algorytm wyrażony w języku przekładalnym na kod maszynowy komputera

1. języki niskiego poziomu (asemblery, kompilatory, interpretatory)
2. języki wysokiego poziomu (implementacja algorytmu)
   - do programowania imperatywnego (Fortran, Algol, Basic, Pascal)

ALGORYTMY

Współczesne językowe (praktyczne) realizacje algorytmów:

Program – algorytm wyrażony w języku przekładalnym na kod maszynowy komputera

1. języki niskiego poziomu (asemblery, kompilatory, interpretatory)
2. języki wysokiego poziomu (implementacja algorytmu)
   - do programowania imperatywnego (Fortran, Algol, Basic, Pascal)
   - do programowania deklaratywnego (Prolog)

ALGORYTMY

Współczesne językowe (praktyczne) realizacje algorytmów:

Program – algorytm wyrażony w języku przekładalnym na kod maszynowy komputera

1. języki niskiego poziomu (asemblery, kompilatory, interpretatory)
2. języki wysokiego poziomu (implementacja algorytmu)
   - do programowania imperatywnego (Fortran, Algol, Basic, Pascal)
   - do programowania deklaratywnego (Prolog)
   - do programowania obiektowego (C^{++})

ALGORYTMY

Współczesne językowe (praktyczne) realizacje algorytmów:

Program – algorytm wyrażony w języku przekładalnym na kod maszynowy komputera

1. języki niskiego poziomu (asemblery, kompilatory, interpretatory)
2. języki wysokiego poziomu (implementacja algorytmu)
   - do programowania imperatywnego (Fortran, Algol, Basic, Pascal)
   - do programowania deklaratywnego (Prolog)
   - do programowania obiektowego (C^{++})
3. praktyczne (specyfikacja algorytmu) – pseudokod, schematy blokowe

MASZYNY

Typy maszyn i ich środowisko:

MASZYNY

Typy maszyn i ich środowisko:

1. materia $\Longleftrightarrow$ obrabiarka

MASZYNY

Typy maszyn i ich środowisko:

1. materia $\Longleftrightarrow$ obrabiarka
2. energia $\Longleftrightarrow$ silnik

MASZYNY

Typy maszyn i ich środowisko:

1. materia $\Longleftrightarrow$ obrabiarka
2. energia $\Longleftrightarrow$ silnik
3. informacja $\Longleftrightarrow$ komputer

MASZYNY

Typy maszyn i ich środowisko:

1. materia $\Longleftrightarrow$ obrabiarka
2. energia $\Longleftrightarrow$ silnik
3. informacja $\Longleftrightarrow$ komputer

Problem Marciszewskego: Czy pojawi się równanie sprowadzające informację do energii analogiczne do równania Einsteina?

MASZYNY

Maszyny liczące – krótka historia:

MASZYNY

Maszyny liczące – krótka historia:

1. XVII w. arytmometry ręczne Schickarda, Pascala i Leibniza

MASZYNY

Maszyny liczące – krótka historia:

1. XVII w. arytmometry ręczne Schickarda, Pascala i Leibniza
2. XIX w. automatyczna maszyna licząca – Ch. Babbage, A. Lovelace

MASZYNY

Maszyny liczące – krótka historia:

1. XVII w. arytmometry ręczne Schickarda, Pascala i Leibniza
2. XIX w. automatyczna maszyna licząca – Ch. Babbage, A. Lovelace
3. 1896 – maszyny tabulujące Holleritha $\Longrightarrow$ IBM

MASZYNY

Maszyny liczące – krótka historia:

1. XVII w. arytmometry ręczne Schickarda, Pascala i Leibniza
2. XIX w. automatyczna maszyna licząca – Ch. Babbage, A. Lovelace
3. 1896 – maszyny tabulujące Holleritha $\Longrightarrow$ IBM
4. lata 30/40-te XX w. maszyny kryptograficzne – Zuse, Turing, von Neumann

MASZYNY

Maszyny liczące – krótka historia:

1. XVII w. arytmometry ręczne Schickarda, Pascala i Leibniza
2. XIX w. automatyczna maszyna licząca – Ch. Babbage, A. Lovelace
3. 1896 – maszyny tabulujące Holleritha $\Longrightarrow$ IBM
4. lata 30/40-te XX w. maszyny kryptograficzne – Zuse, Turing, von Neumann
5. pierwsze komputery – Colossus, ENIAC

MASZYNY

Ważniejsze teoretyczne ujęcia maszyn liczących:

MASZYNY

Ważniejsze teoretyczne ujęcia maszyn liczących:

1. Maszyny Turinga (1936)

MASZYNY

Ważniejsze teoretyczne ujęcia maszyn liczących:

1. Maszyny Turinga (1936)
2. Kanoniczne i normalne systemy Posta (1936, 1943, 1946)

MASZYNY

Ważniejsze teoretyczne ujęcia maszyn liczących:

1. Maszyny Turinga (1936)
2. Kanoniczne i normalne systemy Posta (1936, 1943, 1946)
3. Automaty Rabina i Scotta

MASZYNY

Ważniejsze teoretyczne ujęcia maszyn liczących:

1. Maszyny Turinga (1936)
2. Kanoniczne i normalne systemy Posta (1936, 1943, 1946)
3. Automaty Rabina i Scotta
4. Maszyny rejestrowe Shepherdsona i Sturgisa (1963)

MASZYNY

Ważniejsze teoretyczne ujęcia maszyn liczących:

1. Maszyny Turinga (1936)
2. Kanoniczne i normalne systemy Posta (1936, 1943, 1946)
3. Automaty Rabina i Scotta
4. Maszyny rejestrowe Shepherdsona i Sturgisa (1963)
5. Maszyny RAM

MASZYNY

Maszyna Turinga jest:

MASZYNY

Maszyna Turinga jest:

1. abstrakcyjna (sposób reprezentacji algorytmu)

MASZYNY

Maszyna Turinga jest:

1. abstrakcyjna (sposób reprezentacji algorytmu)
2. dyskretna

MASZYNY

Maszyna Turinga jest:

1. abstrakcyjna (sposób reprezentacji algorytmu)
2. dyskretna
3. deterministyczna

MASZYNY

Budowa maszyny Turinga:

MASZYNY

Budowa maszyny Turinga:

1. nieskończona taśma stanów

MASZYNY

Budowa maszyny Turinga:

1. nieskończona taśma stanów
2. głowica odczytująca aktualny stan taśmy i wykonująca jedną z następujących czynności:

MASZYNY

Budowa maszyny Turinga:

1. nieskończona taśma stanów
2. głowica odczytująca aktualny stan taśmy i wykonująca jedną z następujących czynności:
   - wpisuje symbol

MASZYNY

Budowa maszyny Turinga:

1. nieskończona taśma stanów
2. głowica odczytująca aktualny stan taśmy i wykonująca jedną z następujących czynności:
   - wpisuje symbol
   - kasuje symbol

MASZYNY

Budowa maszyny Turinga:

1. nieskończona taśma stanów
2. głowica odczytująca aktualny stan taśmy i wykonująca jedną z następujących czynności:
   - wpisuje symbol
   - kasuje symbol
   - przechodzi do sąsiedniego stanu

MASZYNY

Budowa maszyny Turinga:

1. nieskończona taśma stanów
2. głowica odczytująca aktualny stan taśmy i wykonująca jedną z następujących czynności:
   - wpisuje symbol
   - kasuje symbol
   - przechodzi do sąsiedniego stanu
3. skończony słownik symboli, np. $\{0, 1\}$

MASZYNY

Budowa maszyny Turinga:

1. nieskończona taśma stanów
2. głowica odczytująca aktualny stan taśmy i wykonująca jedną z następujących czynności:
   - wpisuje symbol
   - kasuje symbol
   - przechodzi do sąsiedniego stanu
3. skończony słownik symboli, np. $\{0, 1\}$
4. skończony zbiór instrukcji postaci – (s_i, S, O, s_j)

MASZYNY

Budowa maszyny Turinga:

1. nieskończona taśma stanów
2. głowica odczytująca aktualny stan taśmy i wykonująca jedną z następujących czynności:
   - wpisuje symbol
   - kasuje symbol
   - przechodzi do sąsiedniego stanu
3. skończony słownik symboli, np. $\{0, 1\}$
4. skończony zbiór instrukcji postaci – (s_i, S, O, s_j)

Uniwersalna maszyna Turinga $\Longrightarrow$ klucz do powstania realnego komputera.

MASZYNY, JĘZYKI, GRAMATYKI

Nie zawsze potrzebujemy całej mocy!

Hierarchia Chomsky'ego:

MASZYNY, JĘZYKI, GRAMATYKI

Nie zawsze potrzebujemy całej mocy!

Hierarchia Chomsky'ego:

1. Maszyny Turinga $\Leftrightarrow$ języki rekurencyjnie przeliczalne $\Leftrightarrow$ gramatyki frazowe (typ 0)

MASZYNY, JĘZYKI, GRAMATYKI

Nie zawsze potrzebujemy całej mocy!

Hierarchia Chomsky'ego:

1. Maszyny Turinga $\Leftrightarrow$ języki rekurencyjnie przeliczalne $\Leftrightarrow$ gramatyki frazowe (typ 0)
2. Automaty linowo ograniczone $\Leftrightarrow$ języki kontekstowe $\Leftrightarrow$ gramatyki kontekstowe (typ 1)

MASZYNY, JĘZYKI, GRAMATYKI

Nie zawsze potrzebujemy całej mocy!

Hierarchia Chomsky'ego:

1. Maszyny Turinga $\Leftrightarrow$ języki rekurencyjnie przeliczalne $\Leftrightarrow$ gramatyki frazowe (typ 0)
2. Automaty linowo ograniczone $\Leftrightarrow$ języki kontekstowe $\Leftrightarrow$ gramatyki kontekstowe (typ 1)
3. Automaty ze stosem $\Leftrightarrow$ języki bezkontekstowe $\Leftrightarrow$ gramatyki bezkontekstowe (typ 2)

MASZYNY, JĘZYKI, GRAMATYKI

Nie zawsze potrzebujemy całej mocy!

Hierarchia Chomsky'ego:

1. Maszyny Turinga $\Leftrightarrow$ języki rekurencyjnie przeliczalne $\Leftrightarrow$ gramatyki frazowe (typ 0)
2. Automaty linowo ograniczone $\Leftrightarrow$ języki kontekstowe $\Leftrightarrow$ gramatyki kontekstowe (typ 1)
3. Automaty ze stosem $\Leftrightarrow$ języki bezkontekstowe $\Leftrightarrow$ gramatyki bezkontekstowe (typ 2)
4. Automaty skończenie stanowe $\Leftrightarrow$ języki regularne $\Leftrightarrow$ gramatyki regularne (typ 3)

MASZYNY, JĘZYKI, GRAMATYKI

Nie zawsze potrzebujemy całej mocy!

Hierarchia Chomsky'ego:

1. Maszyny Turinga $\Leftrightarrow$ języki rekurencyjnie przeliczalne $\Leftrightarrow$ gramatyki frazowe (typ 0)
2. Automaty linowo ograniczone $\Leftrightarrow$ języki kontekstowe $\Leftrightarrow$ gramatyki kontekstowe (typ 1)
3. Automaty ze stosem $\Leftrightarrow$ języki bezkontekstowe $\Leftrightarrow$ gramatyki bezkontekstowe (typ 2)
4. Automaty skończenie stanowe $\Leftrightarrow$ języki regularne $\Leftrightarrow$ gramatyki regularne (typ 3)

OBLICZALNOŚĆ

Tradycja matematyczna – efektywność jako obliczalność:

OBLICZALNOŚĆ

Tradycja matematyczna – efektywność jako obliczalność:

- funkcja (n-argumentowa) liczb naturalnych jest obliczalna wtw istnieje metoda efektywna, która pozwala dla każdego ciągu argumentów ustalić jej wartość

OBLICZALNOŚĆ

Tradycja matematyczna – efektywność jako obliczalność:

- funkcja (n-argumentowa) liczb naturalnych jest obliczalna wtw istnieje metoda efektywna, która pozwala dla każdego ciągu argumentów ustalić jej wartość
- relacja (n-argumentowa) na l. nat. jest obliczalna wtw istnieje metoda efektywna, która pozwala ustalić czy relacja ta zachodzi dla każdej n-ki uporządkowanej l. nat.

OBLICZALNOŚĆ

Tradycja matematyczna – efektywność jako obliczalność:

- funkcja (n-argumentowa) liczb naturalnych jest obliczalna wtw istnieje metoda efektywna, która pozwala dla każdego ciągu argumentów ustalić jej wartość
- relacja (n-argumentowa) na l. nat. jest obliczalna wtw istnieje metoda efektywna, która pozwala ustalić czy relacja ta zachodzi dla każdej n-ki uporządkowanej l. nat.
- zbiór l. nat. jest obliczalny wtw istnieje metoda efektywna, która pozwala dla dowolnej liczby ustalić czy jest elementem tego zbioru czy nie

OBLICZALNOŚĆ

Tradycja matematyczna – efektywność jako obliczalność:

- funkcja (n-argumentowa) liczb naturalnych jest obliczalna wtw istnieje metoda efektywna, która pozwala dla każdego ciągu argumentów ustalić jej wartość
- relacja (n-argumentowa) na l. nat. jest obliczalna wtw istnieje metoda efektywna, która pozwala ustalić czy relacja ta zachodzi dla każdej n-ki uporządkowanej l. nat.
- zbiór l. nat. jest obliczalny wtw istnieje metoda efektywna, która pozwala dla dowolnej liczby ustalić czy jest elementem tego zbioru czy nie

Uwaga: poszczególne pojęcia obliczalności są wzajemnie definiowalne

OBLICZALNOŚĆ

Czy każdy zbiór liczb naturalnych jest obliczalny?

OBLICZALNOŚĆ

Czy każdy zbiór liczb naturalnych jest obliczalny?

1. liczba różnych zbiorów l. naturalnych wynosi $c = Card(\mathcal{P}(N))$

OBLICZALNOŚĆ

Czy każdy zbiór liczb naturalnych jest obliczalny?

1. liczba różnych zbiorów l. naturalnych wynosi $c = Card(\mathcal{P}(N))$
2. zbiorów obliczalnych jest tylko przeliczalnie wiele

OBLICZALNOŚĆ

Czy każdy zbiór liczb naturalnych jest obliczalny?

1. liczba różnych zbiorów l. naturalnych wynosi $c = Card(\mathcal{P}(N))$
2. zbiorów obliczalnych jest tylko przeliczalnie wiele

Zatem: istnieje nieprzeliczalnie wiele zbiorów (funkcji, relacji), które nie są obliczalne.

OBLICZALNOŚĆ

Czy każdy zbiór liczb naturalnych jest obliczalny?

1. liczba różnych zbiorów l. naturalnych wynosi $c = Card(\mathcal{P}(N))$
2. zbiorów obliczalnych jest tylko przeliczalnie wiele

Zatem: istnieje nieprzeliczalnie wiele zbiorów (funkcji, relacji), które nie są obliczalne.

Uwaga: aby sprowadzić rozważanie wszelkich metod efektywnych do rozważania obiektów obliczalnych niezbędna jest arytmetyzacja dziedzin nienumerycznych.

OBLICZALNOŚĆ

Arytmetyzacja – historia problemu:

OBLICZALNOŚĆ

Arytmetyzacja – historia problemu:

1. Kartezjusz – geometria analityczna

OBLICZALNOŚĆ

Arytmetyzacja – historia problemu:

1. Kartezjusz – geometria analityczna
2. Leibniz – algebra logiki

OBLICZALNOŚĆ

Arytmetyzacja – historia problemu:

1. Kartezjusz – geometria analityczna
2. Leibniz – algebra logiki
3. Gödel – arytmetyzacja języka logiki

OBLICZALNOŚĆ

Arytmetyzacja – zasady:

OBLICZALNOŚĆ

Arytmetyzacja – zasady:

Musi dla każdego wyrażenia w efektywnie ustalić jego unikalny gödlowski numer $gn(w)$, a dla każdego numeru, przyporządkowaną mu formułę, czyli spełniać warunki:

OBLICZALNOŚĆ

Arytmetyzacja – zasady:

Musi dla każdego wyrażenia w efektywnie ustalić jego unikalny gödlowski numer $gn(w)$, a dla każdego numeru, przyporządkowaną mu formułę, czyli spełniać warunki:

- $w_i = w_j$ wtw $gn(w_i) = gn(w_j)$

OBLICZALNOŚĆ

Arytmetyzacja – zasady:

Musi dla każdego wyrażenia w efektywnie ustalić jego unikalny gödlowski numer $gn(w)$, a dla każdego numeru, przyporządkowaną mu formułę, czyli spełniać warunki:

- $w_i = w_j$ wtw $gn(w_i) = gn(w_j)$
- istnieje algorytm obliczania dla dowolnego w jego gn w skończonym czasie

OBLICZALNOŚĆ

Arytmetyzacja – zasady:

Musi dla każdego wyrażenia w efektywnie ustalić jego unikalny gödlowski numer $gn(w)$, a dla każdego numeru, przyporządkowaną mu formułę, czyli spełniać warunki:

- $w_i = w_j$ wtw $gn(w_i) = gn(w_j)$
- istnieje algorytm obliczania dla dowolnego w jego gn w skończonym czasie
- dla dowolnej l. nat. n jest rozstrzygalne czy jest ona gn dla jakiegoś wyrażenia

OBLICZALNOŚĆ

Arytmetyzacja – zasady:

Musi dla każdego wyrażenia w efektywnie ustalić jego unikalny gödlowski numer $gn(w)$, a dla każdego numeru, przyporządkowaną mu formułę, czyli spełniać warunki:

- $w_i = w_j$ wtw $gn(w_i) = gn(w_j)$
- istnieje algorytm obliczania dla dowolnego w jego gn w skończonym czasie
- dla dowolnej l. nat. n jest rozstrzygalne czy jest ona gn dla jakiegoś wyrażenia
- jeżeli n jest numerem gödlowskim, to algorytm pozwala w skończonym czasie ustlić jego wartość

OBLICZALNOŚĆ

Arytmetyzacja – przykład:

OBLICZALNOŚĆ

Arytmetyzacja – przykład:

Alfabet $= \{a, b, c\}$.

OBLICZALNOŚĆ

Arytmetyzacja – przykład:

Alfabet $= \{a, b, c\}$.
Dla dowolnego słowa $x_1 ... x_n$, gdzie $x_i \in \{a, b, c\}$ przypisujemy liczbę $2^{d_0} \times 3^{d_1} \times ... \times p_n^{d_n}$, gdzie p_i to i-ta liczba pierwsza, a

OBLICZALNOŚĆ

Arytmetyzacja – przykład:

Alfabet $= \{a, b, c\}$.
Dla dowolnego słowa $x_1 ... x_n$, gdzie $x_i \in \{a, b, c\}$ przypisujemy liczbę $2^{d_0} \times 3^{d_1} \times ... \times p_n^{d_n}$, gdzie p_i to i-ta liczba pierwsza, a

$$d^i = \begin{cases} 1 & \text{jeżeli } x_i = a \\ 2 & \text{jeżeli } x_i = b \\ 3 & \text{jeżeli } x_i = c \end{cases}$$

OBLICZALNOŚĆ

Arytmetyzacja – przykład:

Alfabet $= \{a, b, c\}$.
Dla dowolnego słowa $x_1 ... x_n$, gdzie $x_i \in \{a, b, c\}$ przypisujemy liczbę $2^{d_0} \times 3^{d_1} \times ... \times p_n^{d_n}$, gdzie p_i to i-ta liczba pierwsza, a

$$d^i = \begin{cases} 1 & \text{jeżeli } x_i = a \\ 2 & \text{jeżeli } x_i = b \\ 3 & \text{jeżeli } x_i = c \end{cases}$$

np. $gn(cba) = 2^3 \times 3^2 \times 5^1 = 360$

OBLICZALNOŚĆ

Arytmetyzacja – przykład:

Alfabet $= \{a, b, c\}$.
Dla dowolnego słowa $x_1 ... x_n$, gdzie $x_i \in \{a, b, c\}$ przypisujemy liczbę $2^{d_0} \times 3^{d_1} \times ... \times p_n^{d_n}$, gdzie p_i to i-ta liczba pierwsza, a

$$d^i = \begin{cases} 1 & \text{jeżeli } x_i = a \\ 2 & \text{jeżeli } x_i = b \\ 3 & \text{jeżeli } x_i = c \end{cases}$$

np. $gn(cba) = 2^3 \times 3^2 \times 5^1 = 360$

OBLICZALNOŚĆ

Ważniejsze współczesne definicje funkcji obliczalnych:

OBLICZALNOŚĆ

Ważniejsze współczesne definicje funkcji obliczalnych:

1. definiowalność w logice kombinatorycznej – Schönfinkel (1924), Curry (1929)

OBLICZALNOŚĆ

Ważniejsze współczesne definicje funkcji obliczalnych:

1. definiowalność w logice kombinatorycznej – Schönfinkel (1924), Curry (1929)
2. funcje rekurencyjne – Gödel (1931, 1934 – ogólne f. rekurencyjne), Herbrand (1932)

OBLICZALNOŚĆ

Ważniejsze współczesne definicje funkcji obliczalnych:

1. definiowalność w logice kombinatorycznej – Schönfinkel (1924), Curry (1929)
2. funcje rekurencyjne – Gödel (1931, 1934 – ogólne f. rekurencyjne), Herbrand (1932)
3. definiowalność w rachunku lambda – Church (1936)

OBLICZALNOŚĆ

Ważniejsze współczesne definicje funkcji obliczalnych:

1. definiowalność w logice kombinatorycznej – Schönfinkel (1924), Curry (1929)
2. funcje rekurencyjne – Gödel (1931, 1934 – ogólne f. rekurencyjne), Herbrand (1932)
3. definiowalność w rachunku lambda – Church (1936)
4. reprezentowalność w arytmetyce – Kleene (1952)

OBLICZALNOŚĆ

definicja zbioru funkcji rekurencyjnych:

OBLICZALNOŚĆ

definicja zbioru funkcji rekurencyjnych:

Jest to najmniejszy zbiór funkcji zawierający stałą funkcję zerowania, funkcję następnika, funkcje projekcji oraz domknięty na operacje podstawiania, rekursji prostej i minimum efektywnego.

OBLICZALNOŚĆ

definicja zbioru funkcji rekurencyjnych:

Jest to najmniejszy zbiór funkcji zawierający stałą funkcję zerowania, funkcję następnika, funkcje projekcji oraz domknięty na operacje podstawiania, rekursji prostej i minimum efektywnego.
Funkcje wyjściowe:

OBLICZALNOŚĆ

definicja zbioru funkcji rekurencyjnych:

Jest to najmniejszy zbiór funkcji zawierający stałą funkcję zerowania, funkcję następnika, funkcje projekcji oraz domknięty na operacje podstawiania, rekursji prostej i minimum efektywnego.
Funkcje wyjściowe:

1. stała funkcja zero – $Z(x) = 0$

OBLICZALNOŚĆ

definicja zbioru funkcji rekurencyjnych:

Jest to najmniejszy zbiór funkcji zawierający stałą funkcję zerowania, funkcję następnika, funkcje projekcji oraz domknięty na operacje podstawiania, rekursji prostej i minimum efektywnego.
Funkcje wyjściowe:

1. stała funkcja zero – $Z(x) = 0$
2. funkcja następnika – $S(x) = x + 1$

OBLICZALNOŚĆ

definicja zbioru funkcji rekurencyjnych:

Jest to najmniejszy zbiór funkcji zawierający stałą funkcję zerowania, funkcję następnika, funkcje projekcji oraz domknięty na operacje podstawiania, rekursji prostej i minimum efektywnego. Funkcje wyjściowe:

1. stała funkcja zero – $Z(x) = 0$
2. funkcja następnika – $S(x) = x + 1$
3. n-argumentowe funkcje projekcji – $P_i^n(x_1, ..., x_n) = x_i$, dla $1 \leq i \leq n$, np. funkcja $id(x) = x$

OBLICZALNOŚĆ

operacja podstawiania (kompozycji, superpozycji) funkcji

OBLICZALNOŚĆ

operacja podstawiania (kompozycji, superpozycji) funkcji

$f(x_1, ..., x_n) = g(h_1(x_1, ..., x_n), ..., h_m(x_1, ..., x_n))$

OBLICZALNOŚĆ

operacja podstawiania (kompozycji, superpozycji) funkcji

$f(x_1, ..., x_n) = g(h_1(x_1, ..., x_n), ..., h_m(x_1, ..., x_n))$
gdzie g jest rekurencyjną funkcją m-argumentową a $h_1, ..., h_m$ to rekurencyjne funkcje n-argumentowe.

OBLICZALNOŚĆ

operacja podstawiania (kompozycji, superpozycji) funkcji

$f(x_1, ..., x_n) = g(h_1(x_1, ..., x_n), ..., h_m(x_1, ..., x_n))$
gdzie g jest rekurencyjną funkcją m-argumentową a $h_1, ..., h_m$ to rekurencyjne funkcje n-argumentowe.
Szczególny przypadek to złożenie dwóch funkcji jednoargumentowych $f(x) = g(h(x))$

OBLICZALNOŚĆ

operacja rekursji prostej

OBLICZALNOŚĆ

operacja rekursji prostej

$f(x_1, ..., x_n, 0) = g(x_1, ..., x_n)$

OBLICZALNOŚĆ

operacja rekursji prostej

$f(x_1, ..., x_n, 0) = g(x_1, ..., x_n)$
$f(x_1, ..., x_n, S(y)) = h(x_1, ..., x_n, y, f(x_1, ..., x_n, y))$

OBLICZALNOŚĆ

operacja rekursji prostej

$f(x_1, ..., x_n, 0) = g(x_1, ..., x_n)$
$f(x_1, ..., x_n, S(y)) = h(x_1, ..., x_n, y, f(x_1, ..., x_n, y))$
gdzie g i h to rekurencyjne funkcje

OBLICZALNOŚĆ

operacja rekursji prostej

$f(x_1, ..., x_n, 0) = g(x_1, ..., x_n)$
$f(x_1, ..., x_n, S(y)) = h(x_1, ..., x_n, y, f(x_1, ..., x_n, y))$
gdzie g i h to rekurencyjne funkcje
np. definicja silni:

OBLICZALNOŚĆ

operacja rekursji prostej

$f(x_1, ..., x_n, 0) = g(x_1, ..., x_n)$
$f(x_1, ..., x_n, S(y)) = h(x_1, ..., x_n, y, f(x_1, ..., x_n, y))$
gdzie g i h to rekurencyjne funkcje
np. definicja silni:
$0! = 1$

OBLICZALNOŚĆ

operacja rekursji prostej

$f(x_1, ..., x_n, 0) = g(x_1, ..., x_n)$
$f(x_1, ..., x_n, S(y)) = h(x_1, ..., x_n, y, f(x_1, ..., x_n, y))$
gdzie g i h to rekurencyjne funkcje
np. definicja silni:
$0! = 1$
$(n+1)! = (n+1)(n!)$

OBLICZALNOŚĆ

operacja minimum efektywnego

OBLICZALNOŚĆ

operacja minimum efektywnego

$f(x_1, ..., x_n) = \mu_y g(x_1, ..., x_n, y) = 0$

OBLICZALNOŚĆ

operacja minimum efektywnego

$f(x_1, ..., x_n) = \mu_y g(x_1, ..., x_n, y) = 0$
gdzie g to rekurencyjna funkcja spełniająca warunek:

OBLICZALNOŚĆ

operacja minimum efektywnego

$f(x_1, ..., x_n) = \mu_y g(x_1, ..., x_n, y) = 0$
gdzie g to rekurencyjna funkcja spełniająca warunek:
$\forall x_1, ..., \forall x_n, \exists y g(x_1, ..., x_n, y) = 0$

OBLICZALNOŚĆ

operacja minimum efektywnego

$f(x_1, ..., x_n) = \mu_y g(x_1, ..., x_n, y) = 0$
gdzie g to rekurencyjna funkcja spełniająca warunek:
$\forall x_1, ..., \forall x_n, \exists y g(x_1, ..., x_n, y) = 0$
a μ_y czytamy "najmniejszy taki y, że"

OBLICZALNOŚĆ

Zbiory rekurencyjne i rekurencyjnie przeliczalne:

OBLICZALNOŚĆ

Zbiory rekurencyjne i rekurencyjnie przeliczalne:

- Zbiór jest rekurencyjny wtw funkcja charakterystyczna tego zbioru jest rekurencyjna

OBLICZALNOŚĆ

Zbiory rekurencyjne i rekurencyjnie przeliczalne:

- Zbiór jest rekurencyjny wtw funkcja charakterystyczna tego zbioru jest rekurencyjna
- Zbiór jest rekurencyjnie przeliczalny wtw istnieje efektywna metoda, która dla każdego elementu tego zbioru pozwala pokazać, że do niego należy

OBLICZALNOŚĆ

Zbiory rekurencyjne i rekurencyjnie przeliczalne:

- Zbiór jest rekurencyjny wtw funkcja charakterystyczna tego zbioru jest rekurencyjna
- Zbiór jest rekurencyjnie przeliczalny wtw istnieje efektywna metoda, która dla każdego elementu tego zbioru pozwala pokazać, że do niego należy

Przykłady:

OBLICZALNOŚĆ

Zbiory rekurencyjne i rekurencyjnie przeliczalne:

- Zbiór jest rekurencyjny wtw funkcja charakterystyczna tego zbioru jest rekurencyjna
- Zbiór jest rekurencyjnie przeliczalny wtw istnieje efektywna metoda, która dla każdego elementu tego zbioru pozwala pokazać, że do niego należy

Przykłady:

- zbiory rekurencyjne – np. w KRK: słownik, zbiór poprawnych wyrażeń, zbiór aksjomatów, zbiór dowodów

OBLICZALNOŚĆ

Zbiory rekurencyjne i rekurencyjnie przeliczalne:

- Zbiór jest rekurencyjny wtw funkcja charakterystyczna tego zbioru jest rekurencyjna
- Zbiór jest rekurencyjnie przeliczalny wtw istnieje efektywna metoda, która dla każdego elementu tego zbioru pozwala pokazać, że do niego należy

Przykłady:

- zbiory rekurencyjne – np. w KRK: słownik, zbiór poprawnych wyrażeń, zbiór aksjomatów, zbiór dowodów
- zbiór rekurencyjnie przeliczalny – zbiór tez KRK

OBLICZALNOŚĆ

Zbiory rekurencyjne i rekurencyjnie przeliczalne:

- Zbiór jest rekurencyjny wtw funkcja charakterystyczna tego zbioru jest rekurencyjna
- Zbiór jest rekurencyjnie przeliczalny wtw istnieje efektywna metoda, która dla każdego elementu tego zbioru pozwala pokazać, że do niego należy

Przykłady:

- zbiory rekurencyjne – np. w KRK: słownik, zbiór poprawnych wyrażeń, zbiór aksjomatów, zbiór dowodów
- zbiór rekurencyjnie przeliczalny – zbiór tez KRK
- zbiór rekurencyjnie nieprzeliczalny – zbiór nietez KRK

OBLICZALNOŚĆ

Zbiory rekurencyjne i rekurencyjnie przeliczalne:

- Zbiór jest rekurencyjny wtw funkcja charakterystyczna tego zbioru jest rekurencyjna
- Zbiór jest rekurencyjnie przeliczalny wtw istnieje efektywna metoda, która dla każdego elementu tego zbioru pozwala pokazać, że do niego należy

Przykłady:

- zbiory rekurencyjne – np. w KRK: słownik, zbiór poprawnych wyrażeń, zbiór aksjomatów, zbiór dowodów
- zbiór rekurencyjnie przeliczalny – zbiór tez KRK
- zbiór rekurencyjnie nieprzeliczalny – zbiór nietez KRK

Twierdzenie: Zbiór jest rekurencyjny wtw zarówno on jak i jego dopełnienie jest rek. przeliczalne

OBLICZALNOŚĆ

Teza Churcha-Turinga (1936)

OBLICZALNOŚĆ

Teza Churcha-Turinga (1936)

Jeżeli funkcja zdefiniowana na liczbach naturalnych jest efektywna, to:

OBLICZALNOŚĆ

Teza Churcha-Turinga (1936)

Jeżeli funkcja zdefiniowana na liczbach naturalnych jest efektywna, to:

a) jest funkcją rekurencyjną (Church)

OBLICZALNOŚĆ

Teza Churcha-Turinga (1936)

Jeżeli funkcja zdefiniowana na liczbach naturalnych jest efektywna, to:

a) jest funkcją rekurencyjną (Church)

b) jest obliczalna przez maszynę Turinga (Turing)

OBLICZALNOŚĆ

Teza Churcha-Turinga (1936)

Jeżeli funkcja zdefiniowana na liczbach naturalnych jest efektywna, to:
a) jest funkcją rekurencyjną (Church)
b) jest obliczalna przez maszynę Turinga (Turing)
Częściowe poparcie tezy C-T – dowody równoważności rozmaitych charakterystyk procedury efektywnej (the most amazing fact)

OBLICZALNOŚĆ

Teza Churcha-Turinga (1936)

Jeżeli funkcja zdefiniowana na liczbach naturalnych jest efektywna, to:
a) jest funkcją rekurencyjną (Church)
b) jest obliczalna przez maszynę Turinga (Turing)
Częściowe poparcie tezy C-T – dowody równoważności rozmaitych charakterystyk procedury efektywnej (the most amazing fact)
Obiekcje – np. Post, Kalmar (1959)

WYNIKI

Program Hilberta 1900:

WYNIKI

Program Hilberta 1900:

Ugruntowanie podstaw matematyki oraz znalezienie uniwersalnej i mechanicznej procedury rozwiązywania wszystkich problemów arytmetycznych

WYNIKI

Program Hilberta 1900:

Ugruntowanie podstaw matematyki oraz znalezienie uniwersalnej i mechanicznej procedury rozwiązywania wszystkich problemów arytmetycznych
Narzędzie realizacji: formalizacja (syntaktyczna) teorii matematycznych, metody finitystyczne stosowane do konkretnych obiektów syntaktycznych.

WYNIKI

Program Hilberta 1900:

Ugruntowanie podstaw matematyki oraz znalezienie uniwersalnej i mechanicznej procedury rozwiązywania wszystkich problemów arytmetycznych
Narzędzie realizacji: formalizacja (syntaktyczna) teorii matematycznych, metody finitystyczne stosowane do konkretnych obiektów syntaktycznych.
Zadania: dowody niesprzeczności, zupełności i roztrzygalności (Entscheidungsproblem) teorii matematycznych.

WYNIKI

Program Hilberta 1900:

Ugruntowanie podstaw matematyki oraz znalezienie uniwersalnej i mechanicznej procedury rozwiązywania wszystkich problemów arytmetycznych
Narzędzie realizacji: formalizacja (syntaktyczna) teorii matematycznych, metody finitystyczne stosowane do konkretnych obiektów syntaktycznych.
Zadania: dowody niesprzeczności, zupełności i roztrzygalności (Entscheidungsproblem) teorii matematycznych.
$\Longrightarrow$ Hilbert, Ackermann (1928) – problem rozstrzygalności to zasadniczy problem logiki matematycznej

WYNIKI

Program Hilberta 1900:

Ugruntowanie podstaw matematyki oraz znalezienie uniwersalnej i mechanicznej procedury rozwiązywania wszystkich problemów arytmetycznych
Narzędzie realizacji: formalizacja (syntaktyczna) teorii matematycznych, metody finitystyczne stosowane do konkretnych obiektów syntaktycznych.
Zadania: dowody niesprzeczności, zupełności i roztrzygalności (Entscheidungsproblem) teorii matematycznych.
$\Longrightarrow$ Hilbert, Ackermann (1928) – problem rozstrzygalności to zasadniczy problem logiki matematycznej
Oryginalność programu Hilberta: poszukiwanie zwiększonej ścisłości nie tylko narzędzi dowodzenia w matematyce ale przede wszystkim o matematyce $\Longrightarrow$ metamatematyka

WYNIKI

Program Hilberta 1900:

Ugruntowanie podstaw matematyki oraz znalezienie uniwersalnej i mechanicznej procedury rozwiązywania wszystkich problemów arytmetycznych
Narzędzie realizacji: formalizacja (syntaktyczna) teorii matematycznych, metody finitystyczne stosowane do konkretnych obiektów syntaktycznych.
Zadania: dowody niesprzeczności, zupełności i roztrzygalności (Entscheidungsproblem) teorii matematycznych.
$\Longrightarrow$ Hilbert, Ackermann (1928) – problem rozstrzygalności to zasadniczy problem logiki matematycznej
Oryginalność programu Hilberta: poszukiwanie zwiększonej ścisłości nie tylko narzędzi dowodzenia w matematyce ale przede wszystkim o matematyce $\Longrightarrow$ metamatematyka
Wady: niedostateczne sprecyzowanie pojecia metody finitystycznej.

WYNIKI

Roztrzygalność teorii:

WYNIKI

Roztrzygalność teorii:

- Teoria jest rozstrzygalna wtw zbiór (numerów gödlowskich) jej twierdzeń jest rekurencyjny

WYNIKI

Roztrzygalność teorii:

- Teoria jest rozstrzygalna wtw zbiór (numerów gödlowskich) jej twierdzeń jest rekurencyjny
- Teoria jest nierozstrzygalna wtw zbiór jej twierdzeń nie jest rekurencyjny

WYNIKI

Roztrzygalność teorii:

- Teoria jest rozstrzygalna wtw zbiór (numerów gödlowskich) jej twierdzeń jest rekurencyjny
- Teoria jest nierozstrzygalna wtw zbiór jej twierdzeń nie jest rekurencyjny
- Teoria jest istotnie nierozstrzygalna wtw każde jej niesprzeczne rozszerzenie jest nierozstrzygalne

WYNIKI

Roztrzygalność teorii:

- Teoria jest rozstrzygalna wtw zbiór (numerów gödlowskich) jej twierdzeń jest rekurencyjny
- Teoria jest nierozstrzygalna wtw zbiór jej twierdzeń nie jest rekurencyjny
- Teoria jest istotnie nierozstrzygalna wtw każde jej niesprzeczne rozszerzenie jest nierozstrzygalne
- Teoria jest półrozstrzygalna wtw zbiór jej twierdzeń jest rekurencyjnie przeliczalny

WYNIKI

Zupełność teorii:

WYNIKI

Zupełność teorii:

- Teoria jest zupełna wtw dla dowolnego zdania albo ono albo jego negacja należy do teorii, w przeciwnym wypadku jest niezupełna

WYNIKI

Zupełność teorii:

- Teoria jest zupełna wtw dla dowolnego zdania albo ono albo jego negacja należy do teorii, w przeciwnym wypadku jest niezupełna
- Teoria jest Post-zupełna wtw staje się sprzeczna po dołączeniu do niej dowolnego zdania nie będącego jej twierdzeniem

WYNIKI

Zupełność teorii:

- Teoria jest zupełna wtw dla dowolnego zdania albo ono albo jego negacja należy do teorii, w przeciwnym wypadku jest niezupełna
- Teoria jest Post-zupełna wtw staje się sprzeczna po dołączeniu do niej dowolnego zdania nie będącego jej twierdzeniem
- Teoria jest istotnie niezupełna wtw każde jej niesprzeczne rozszerzenie jest niezupełne

WYNIKI

Zupełność teorii:

- Teoria jest zupełna wtw dla dowolnego zdania albo ono albo jego negacja należy do teorii, w przeciwnym wypadku jest niezupełna
- Teoria jest Post-zupełna wtw staje się sprzeczna po dołączeniu do niej dowolnego zdania nie będącego jej twierdzeniem
- Teoria jest istotnie niezupełna wtw każde jej niesprzeczne rozszerzenie jest niezupełne

Twierdzenie: Dowolna teoria jest istotnie nierozstrzygalna wtw jest istotnie niezupełna

WYNIKI

Twierdzenia limitacyjne:

WYNIKI

Twierdzenia limitacyjne:

- 1931 – I tw. Gödla o (istotnej) niezupełności arytmetyki I. nat.

WYNIKI

Twierdzenia limitacyjne:

- 1931 – I tw. Gödla o (istotnej) niezupełności arytmetyki l. nat.
- 1931 – II tw Gödla o nieudowadnialności niesprzeczności arytmetyki

WYNIKI

Twierdzenia limitacyjne:

- 1931 – I tw. Gödla o (istotnej) niezupełności arytmetyki l. nat.
- 1931 – II tw Gödla o nieudowadnialności niesprzeczności arytmetyki
- 1936 – tw. Churcha o nierozstrzygalności KRK

WYNIKI

Twierdzenia limitacyjne:

- 1931 – I tw. Gödla o (istotnej) niezupełności arytmetyki l. nat.
- 1931 – II tw Gödla o nieudowadnialności niesprzeczności arytmetyki
- 1936 – tw. Churcha o nierozstrzygalności KRK
- 1936 – tw. Churcha o istotnej nierozstrzygalności arytmetyki

WYNIKI POZYTYWNE

Poziomy możliwości formalizacyjnych w logice:

WYNIKI POZYTYWNE

Poziomy możliwości formalizacyjnych w logice:

- KRZ – niezupełny, Post-zupełny, pełny, rozstrzygalny

WYNIKI POZYTYWNE

Poziomy możliwości formalizacyjnych w logice:

- KRZ – niezupełny, Post-zupełny, pełny, rozstrzygalny
- KRK – pełny, nierozstrzygalny ale półrozstrzygalny

WYNIKI POZYTYWNE

Poziomy możliwości formalizacyjnych w logice:

- KRZ – niezupełny, Post-zupełny, pełny, rozstrzygalny
- KRK – pełny, nierozstrzygalny ale półrozstrzygalny
- logika 2-rzędu – niepełna, nierozstrzygalna

WYNIKI POZYTYWNE

Poziomy możliwości formalizacyjnych w logice:

- KRZ – niezupełny, Post-zupełny, pełny, rozstrzygalny
- KRK – pełny, nierozstrzygalny ale półrozstrzygalny
- logika 2-rzędu – niepełna, nierozstrzygalna

ale wiele fragmentów KRK jest rozstrzygalnych! np. monadyczny KRK (Löwenheim 1915).

WYNIKI POZYTYWNE

Przykłady teorii (elementarnych) rozstrzygalnych:

WYNIKI POZYTYWNE

Przykłady teorii (elementarnych) rozstrzygalnych:

1. Teoria równości (Löwenheim 1915)

WYNIKI POZYTYWNE

Przykłady teorii (elementarnych) rozstrzygalnych:

1. Teoria równości (Löwenheim 1915)
2. Teorie niektórych relacji porządkujących (np. gęstego porządku liniowego bez elementu pierwszego i ostatniego, porządku dyskretnego (Langford 1927))

WYNIKI POZYTYWNE

Przykłady teorii (elementarnych) rozstrzygalnych:

1. Teoria równości (Löwenheim 1915)
2. Teorie niektórych relacji porządkujących (np. gęstego porządku liniowego bez elementu pierwszego i ostatniego, porządku dyskretnego (Langford 1927))
3. Teoria algebr Boole'a (Tarski 1949)

WYNIKI POZYTYWNE

Przykłady teorii (elementarnych) rozstrzygalnych:

1. Teoria równości (Löwenheim 1915)
2. Teorie niektórych relacji porządkujących (np. gęstego porządku liniowego bez elementu pierwszego i ostatniego, porządku dyskretnego (Langford 1927))
3. Teoria algebr Boole'a (Tarski 1949)
4. Teoria grup abelowych (Szmielew 1955)

WYNIKI POZYTYWNE

Przykłady teorii (elementarnych) rozstrzygalnych:

1. Teoria równości (Löwenheim 1915)
2. Teorie niektórych relacji porządkujących (np. gęstego porządku liniowego bez elementu pierwszego i ostatniego, porządku dyskretnego (Langford 1927))
3. Teoria algebr Boole'a (Tarski 1949)
4. Teoria grup abelowych (Szmielew 1955)
5. Arytmetyka Presburgera liczb naturalnych z $+$ (1929)

WYNIKI POZYTYWNE

Przykłady teorii (elementarnych) rozstrzygalnych:

1. Teoria równości (Löwenheim 1915)
2. Teorie niektórych relacji porządkujących (np. gęstego porządku liniowego bez elementu pierwszego i ostatniego, porządku dyskretnego (Langford 1927))
3. Teoria algebr Boole'a (Tarski 1949)
4. Teoria grup abelowych (Szmielew 1955)
5. Arytmetyka Presburgera liczb naturalnych z $+$ (1929)
6. Arytmetyka Skolema liczb naturalnych z $\times$ (1930)

WYNIKI POZYTYWNE

Przykłady teorii (elementarnych) rozstrzygalnych:

1. Teoria równości (Löwenheim 1915)
2. Teorie niektórych relacji porządkujących (np. gęstego porządku liniowego bez elementu pierwszego i ostatniego, porządku dyskretnego (Langford 1927))
3. Teoria algebr Boole'a (Tarski 1949)
4. Teoria grup abelowych (Szmielew 1955)
5. Arytmetyka Presburgera liczb naturalnych z $+$ (1929)
6. Arytmetyka Skolema liczb naturalnych z $\times$ (1930)
7. Teoria liczb rzeczywistych (Tarski 1951)

WYNIKI POZYTYWNE

Przykłady teorii (elementarnych) rozstrzygalnych – komentarze:

WYNIKI POZYTYWNE

Przykłady teorii (elementarnych) rozstrzygalnych – komentarze:

ad 3 i 4: zarówno teoria krat jak i teoria grup jest nierozstrzygalna ale nie jest istotnie nierozstrzygalna!

WYNIKI POZYTYWNE

Przykłady teorii (elementarnych) rozstrzygalnych – komentarze:

ad 3 i 4: zarówno teoria krat jak i teoria grup jest nierozstrzygalna ale nie jest istotnie nierozstrzygalna!

ad 5 i 6: pokazuje to niezależność operacji dodawania i mnożenia

WYNIKI POZYTYWNE

Przykłady teorii (elementarnych) rozstrzygalnych – komentarze:

ad 3 i 4: zarówno teoria krat jak i teoria grup jest nierozstrzygalna ale nie jest istotnie nierozstrzygalna!

ad 5 i 6: pokazuje to niezależność operacji dodawania i mnożenia

as 7: fakt ten nie jest w konflikcie z istotną nierozstrzygalnością arytmetyki liczb naturalnych!

WYNIKI

Jak dowodzimy rozstrzygalności?

WYNIKI

Jak dowodzimy rozstrzygalności?

Wybrane metody:

WYNIKI

Jak dowodzimy rozstrzygalności?

Wybrane metody:

- metodą eliminacji kwantyfikatorów (Löwenheim 1915, Skolem 1919)

WYNIKI

Jak dowodzimy rozstrzygalności?

Wybrane metody:

- metodą eliminacji kwantyfikatorów (Löwenheim 1915, Skolem 1919)
- przez metody teoriomodelowe (np. wykazanie kategoryczności teorii lub własności skończonych modeli)

WYNIKI

Jak dowodzimy rozstrzygalności?

Wybrane metody:

- metodą eliminacji kwantyfikatorów (Löwenheim 1915, Skolem 1919)
- przez metody teoriomodelowe (np. wykazanie kategoryczności teorii lub własności skończonych modeli)
- przez interpretację w rozstrzygalnych teoriach

WYNIKI

Jak dowodzimy rozstrzygalności?

Wybrane metody:

- metodą eliminacji kwantyfikatorów (Löwenheim 1915, Skolem 1919)
- przez metody teoriomodelowe (np. wykazanie kategoryczności teorii lub własności skończonych modeli)
- przez interpretację w rozstrzygalnych teoriach
- przez konstrukcję poprawnego algorytmu decyzyjnego

WYNIKI

Jak dowodzimy nierozstrzygalności?

WYNIKI

Jak dowodzimy nierozstrzygalności?

Wybrane metody:

WYNIKI

Jak dowodzimy nierozstrzygalności?

Wybrane metody:

- przez wykazanie, że zawiera w sobie istotnie nierozstrzygalną teorię

WYNIKI

Jak dowodzimy nierozstrzygalności?

Wybrane metody:

- przez wykazanie, że zawiera w sobie istotnie nierozstrzygalną teorię
- przez wykazanie, że zawiera się w jakiejś teorii nierozstrzygalnej w tym samym języku

TEORIA ZŁOŻONOŚCI OBLICZENIOWEJ

Co zmieniły komputery?

TEORIA ZŁOŻONOŚCI OBLICZENIOWEJ

Co zmieniły komputery?

- dawniej: teoria obliczalności $\Longrightarrow$ główny problem: czy X jest obliczalne?

TEORIA ZŁOŻONOŚCI OBLICZENIOWEJ

Co zmieniły komputery?

- dawniej: teoria obliczalności $\Longrightarrow$ główny problem: czy X jest obliczalne?
- obecnie: teoria złożoności obliczeniowej $\Longrightarrow$ główny problem: czy X jest praktycznie obliczalne ?

TEORIA ZŁOŻONOŚCI OBLICZENIOWEJ

Co zmieniły komputery? – niektóre konsekwencje:

TEORIA ZŁOŻONOŚCI OBLICZENIOWEJ

Co zmieniły komputery? – niektóre konsekwencje:

- algorytmy z obiektów idealnych zmieniły się w konkretne

TEORIA ZŁOŻONOŚCI OBLICZENIOWEJ

Co zmieniły komputery? – niektóre konsekwencje:

- algorytmy z obiektów idealnych zmieniły się w konkretne (implementacja zastąpiła kontemplację

TEORIA ZŁOŻONOŚCI OBLICZENIOWEJ

Co zmieniły komputery? – niektóre konsekwencje:

- algorytmy z obiektów idealnych zmieniły się w konkretne (implementacja zastąpiła kontemplację
- wiele teorii uznawanych za mechaniczne (bo rozstrzygalne) i zaniedbanych ponownie stało się przedmiotem intensywnych badań (np. KRZ i jego rola w teorii NP-zupełności)

TEORIA ZŁOŻONOŚCI OBLICZENIOWEJ

Co zmieniły komputery? – niektóre konsekwencje:

- algorytmy z obiektów idealnych zmieniły się w konkretne (implementacja zastąpiła kontemplację
- wiele teorii uznawanych za mechaniczne (bo rozstrzygalne) i zaniedbanych ponownie stało się przedmiotem intensywnych badań (np. KRZ i jego rola w teorii NP-zupełności)
- problemy lekceważone (jako inżynierskie) zostały docenione (problemy optymalizacji)

TEORIA ZŁOŻONOŚCI OBLICZENIOWEJ

Co zmieniły komputery? – niektóre konsekwencje:

- algorytmy z obiektów idealnych zmieniły się w konkretne (implementacja zastąpiła kontemplację
- wiele teorii uznawanych za mechaniczne (bo rozstrzygalne) i zaniedbanych ponownie stało się przedmiotem intensywnych badań (np. KRZ i jego rola w teorii NP-zupełności)
- problemy lekceważone (jako inżynierskie) zostały docenione (problemy optymalizacji)
- teorie nierozstrzygalne stymulują badania nad metodami niepełnymi i wydajnymi (np. algorytmy dla klauzul Horne'a)

MIARY ZŁOŻONOŚCI

Jak mierzyć wydajność algorytmu?

MIARY ZŁOŻONOŚCI

Jak mierzyć wydajność algorytmu?

Co mierzymy?

MIARY ZŁOŻONOŚCI

Jak mierzyć wydajność algorytmu?

Co mierzymy?

- czas potrzebny na wykonanie obliczeń

MIARY ZŁOŻONOŚCI

Jak mierzyć wydajność algorytmu?

Co mierzymy?

- czas potrzebny na wykonanie obliczeń
- pamięć niezbędna do ich magazynowania

MIARY ZŁOŻONOŚCI

Jak mierzyć wydajność algorytmu?

Co mierzymy?

- czas potrzebny na wykonanie obliczeń
- pamięć niezbędna do ich magazynowania

Co ustalamy?

MIARY ZŁOŻONOŚCI

Jak mierzyć wydajność algorytmu?

Co mierzymy?

- czas potrzebny na wykonanie obliczeń
- pamięć niezbędna do ich magazynowania

Co ustalamy?

- ograniczenie górne – notacja $O(f(n))$

MIARY ZŁOŻONOŚCI

Jak mierzyć wydajność algorytmu?

Co mierzymy?

- czas potrzebny na wykonanie obliczeń
- pamięć niezbędna do ich magazynowania

Co ustalamy?

- ograniczenie górne – notacja $O(f(n))$
- ograniczenie dolne – notacja $\Omega(f(n))$

MIARY ZŁOŻONOŚCI

Jak mierzyć wydajność algorytmu?

Co mierzymy?

- czas potrzebny na wykonanie obliczeń
- pamięć niezbędna do ich magazynowania

Co ustalamy?

- ograniczenie górne – notacja $O(f(n))$
- ograniczenie dolne – notacja $\Omega(f(n))$
- średnia wydajność

gdzie f to funkcja wyznaczona przez analizowany algorytm, a n to rozmiar danych

MIARY ZŁOŻONOŚCI

Ograniczenie górne czasowe:

MIARY ZŁOŻONOŚCI

Ograniczenie górne czasowe:

Niech f i g to funkcje jednoargumentowe na l. nat.:

MIARY ZŁOŻONOŚCI

Ograniczenie górne czasowe:

Niech f i g to funkcje jednoargumentowe na l. nat.:
f jest ograniczone przez g wtw $\exists j \forall k(k \geq j \rightarrow f(k) \leq g(k))$

MIARY ZŁOŻONOŚCI

Ograniczenie górne czasowe:

Niech f i g to funkcje jednoargumentowe na l. nat.:
f jest ograniczone przez g wtw $\exists j \forall k(k \geq j \rightarrow f(k) \leq g(k))$
np. $f(n) = n + 50$ a $g(n) = 3n$ dla $j = 25$

MIARY ZŁOŻONOŚCI

Ograniczenie górne czasowe:

Niech f i g to funkcje jednoargumentowe na l. nat.:
f jest ograniczone przez g wtw $\exists j \forall k(k \geq j \rightarrow f(k) \leq g(k))$
np. $f(n) = n + 50$ a $g(n) = 3n$ dla $j = 25$

- każda funkcja na n ograniczona przez funkcję postaci n^k (k ustalone) jest wielomianowa

MIARY ZŁOŻONOŚCI

Ograniczenie górne czasowe:

Niech f i g to funkcje jednoargumentowe na l. nat.:
f jest ograniczone przez g wtw $\exists j \forall k(k \geq j \rightarrow f(k) \leq g(k))$
np. $f(n) = n + 50$ a $g(n) = 3n$ dla $j = 25$

- każda funkcja na n ograniczona przez funkcję postaci n^k (k ustalone) jest wielomianowa
- jeżeli funkcja ustalająca dla danego algorytmu związek wielkości danych na wejściu do ilości czasu potrzebnego na rozwiązanie problemu jest wielomianowa, to algorytm pracuje w czasie wielomianowym

MIARY ZŁOŻONOŚCI

Ograniczenie górne czasowe:

Niech f i g to funkcje jednoargumentowe na l. nat.:
f jest ograniczone przez g wtw $\exists j \forall k(k \geq j \rightarrow f(k) \leq g(k))$
np. $f(n) = n + 50$ a $g(n) = 3n$ dla $j = 25$

- każda funkcja na n ograniczona przez funkcję postaci n^k (k ustalone) jest wielomianowa
- jeżeli funkcja ustalająca dla danego algorytmu związek wielkości danych na wejściu do ilości czasu potrzebnego na rozwiązanie problemu jest wielomianowa, to algorytm pracuje w czasie wielomianowym
- problem rozwiązywalny przez (deterministyczny) algorytm wielomianowy należy do klasy P (albo $PTIME$)

MIARY ZŁOŻONOŚCI

Ograniczenie górne czasowe:

Niech f i g to funkcje jednoargumentowe na l. nat.:
f jest ograniczone przez g wtw $\exists j \forall k(k \geq j \rightarrow f(k) \leq g(k))$
np. $f(n) = n + 50$ a $g(n) = 3n$ dla $j = 25$

- każda funkcja na n ograniczona przez funkcję postaci n^k (k ustalone) jest wielomianowa
- jeżeli funkcja ustalająca dla danego algorytmu związek wielkości danych na wejściu do ilości czasu potrzebnego na rozwiązanie problemu jest wielomianowa, to algorytm pracuje w czasie wielomianowym
- problem rozwiązywalny przez (deterministyczny) algorytm wielomianowy należy do klasy P (albo $PTIME$)
- P = klasa problemów praktycznie rozwiązywalnych (tractable)

MIARY ZŁOŻONOŚCI

Problemy praktycznie rozwiązywalne – hierarchia:

MIARY ZŁOŻONOŚCI

Problemy praktycznie rozwiązywalne – hierarchia:

1. logarytmiczne – $O(log\ n)$, np. $f(n) = log_2 n$

MIARY ZŁOŻONOŚCI

Problemy praktycznie rozwiązywalne – hierarchia:

1. logarytmiczne – $O(log\ n)$, np. $f(n) = log_2 n$
2. liniowe – $O(n)$, np. $f(n) = 3n + 5$

MIARY ZŁOŻONOŚCI

Problemy praktycznie rozwiązywalne – hierarchia:

1. logarytmiczne – $O(log\ n)$, np. $f(n) = log_2 n$
2. liniowe – $O(n)$, np. $f(n) = 3n + 5$
3. kwadratowe – $O(n^2)$, np. $f(n) = 4n^2 + 5n + 3$

MIARY ZŁOŻONOŚCI

Problemy praktycznie rozwiązywalne – hierarchia:

1. logarytmiczne – $O(log\ n)$, np. $f(n) = log_2 n$
2. liniowe – $O(n)$, np. $f(n) = 3n + 5$
3. kwadratowe – $O(n^2)$, np. $f(n) = 4n^2 + 5n + 3$

generalnie: problemy wielomianowe są rzędu $O(n^k)$, dla $f(n) = a_1 n^k + a_2 n^{k-1} + ... + a_k$, gdzie $a_1, a_2, ..., a_k$ to dowolne liczby, w tym $a_1 \neq 0$

MIARY ZŁOŻONOŚCI

Problemy praktycznie rozwiązywalne – hierarchia:

1. logarytmiczne – $O(log\ n)$, np. $f(n) = log_2 n$
2. liniowe – $O(n)$, np. $f(n) = 3n + 5$
3. kwadratowe – $O(n^2)$, np. $f(n) = 4n^2 + 5n + 3$

generalnie: problemy wielomianowe są rzędu $O(n^k)$, dla $f(n) = a_1 n^k + a_2 n^{k-1} + ... + a_k$, gdzie $a_1, a_2, ..., a_k$ to dowolne liczby, w tym $a_1 \neq 0$

Uwaga! przy określaniu rzędu złożoności pomijamy czynniki stałe, choć w praktyce mogą mieć znaczny wpływ na różnice w pracy konkretnych algorytmów. Podobnie lekceważymy różnice wykładników; n^2 i n^{200} są rzędu $O(n^k)$ choć w praktyce różnica jest drastyczna.

MIARY ZŁOŻONOŚCI

Problemy a algorytmy:

MIARY ZŁOŻONOŚCI

Problemy a algorytmy:

- Należy odróżnić problem od algorytmów służących do jego rozwiązania!

MIARY ZŁOŻONOŚCI

Problemy a algorytmy:

- Należy odróżnić problem od algorytmów służących do jego rozwiązania!
- Aktualne algorytmy wyznaczają górne ograniczenie wydajności.

MIARY ZŁOŻONOŚCI

Problemy a algorytmy:

- Należy odróżnić problem od algorytmów służących do jego rozwiązania!
- Aktualne algorytmy wyznaczają górne ograniczenie wydajności.
- W niektórych przypadkach można udowodnić dolne ograniczenie złożoności problemu.

MIARY ZŁOŻONOŚCI

Problemy a algorytmy:

- Należy odróżnić problem od algorytmów służących do jego rozwiązania!
- Aktualne algorytmy wyznaczają górne ograniczenie wydajności.
- W niektórych przypadkach można udowodnić dolne ograniczenie złożoności problemu.
- Jeżeli jest między nimi różnica, to określamy ją jako lukę algorytmiczną dla danego problemu; w przeciwnym wypadku problem jest algorytmicznie zamknięty.

MIARY ZŁOŻONOŚCI

Problemy a algorytmy:

- Należy odróżnić problem od algorytmów służących do jego rozwiązania!
- Aktualne algorytmy wyznaczają górne ograniczenie wydajności.
- W niektórych przypadkach można udowodnić dolne ograniczenie złożoności problemu.
- Jeżeli jest między nimi różnica, to określamy ją jako lukę algorytmiczną dla danego problemu; w przeciwnym wypadku problem jest algorytmicznie zamknięty.
- Problemy z luką algorytmiczną lub z nieustalonym ograniczeniem dolnym są algorytmicznie otwarte.

MIARY ZŁOŻONOŚCI

Przykłady problemów algorytmicznych:

MIARY ZŁOŻONOŚCI

Przykłady problemów algorytmicznych:

- wieża Hanoi – problem zamknięty $O(2^n)$

MIARY ZŁOŻONOŚCI

Przykłady problemów algorytmicznych:

- wieża Hanoi – problem zamknięty $O(2^n)$
- sortowanie par – metoda naiwna $O(n^2)$; metoda sprawna $O(n)$

MIARY ZŁOŻONOŚCI

Przykłady problemów algorytmicznych:

- wieża Hanoi – problem zamknięty $O(2^n)$
- sortowanie par – metoda naiwna $O(n^2)$; metoda sprawna $O(n)$
- przeszukiwanie listy uporządkowanej – metoda naiwna $O(n)$; wyszukiwanie binarne $O(log\ n)$

MIARY ZŁOŻONOŚCI

Problemy trudno rozwiązywalne (niepodatne) – hierarchia:

MIARY ZŁOŻONOŚCI

Problemy trudno rozwiązywalne (niepodatne) – hierarchia:

1. $O(2^n)$, np. $f(n) = 2^n + 5$

MIARY ZŁOŻONOŚCI

Problemy trudno rozwiązywalne (niepodatne) – hierarchia:

1. $O(2^n)$, np. $f(n) = 2^n + 5$
2. $O(n!)$

MIARY ZŁOŻONOŚCI

Problemy trudno rozwiązywalne (niepodatne) – hierarchia:

1. $O(2^n)$, np. $f(n) = 2^n + 5$
2. $O(n!)$
3. $O(n^n)$

MIARY ZŁOŻONOŚCI

Problemy trudno rozwiązywalne (niepodatne) – hierarchia:

1. $O(2^n)$, np. $f(n) = 2^n + 5$
2. $O(n!)$
3. $O(n^n)$

Generalnie są to problemy wykładnicze rzędu $O(k^n)$, dla ustalonego k (zazwyczaj dajemy $k = 2$, gdyż liczby o innej podstawie można sprowadzić do takiej postaci).

MIARY ZŁOŻONOŚCI

Problemy trudno rozwiązywalne (niepodatne) – hierarchia:

1. $O(2^n)$, np. $f(n) = 2^n + 5$
2. $O(n!)$
3. $O(n^n)$

Generalnie są to problemy wykładnicze rzędu $O(k^n)$, dla ustalonego k (zazwyczaj dajemy $k = 2$, gdyż liczby o innej podstawie można sprowadzić do takiej postaci).
Ich klasa to *EXP* (lub *EXPTIME*).

MIARY ZŁOŻONOŚCI

Problemy trudno rozwiązywalne (niepodatne) – hierarchia:

1. $O(2^n)$, np. $f(n) = 2^n + 5$
2. $O(n!)$
3. $O(n^n)$

Generalnie są to problemy wykładnicze rzędu $O(k^n)$, dla ustalonego k (zazwyczaj dajemy $k = 2$, gdyż liczby o innej podstawie można sprowadzić do takiej postaci).
Ich klasa to *EXP* (lub *EXPTIME*).
Uwaga: jeszcze gorsze są problemy wielowykładnicze typu 2^{2^n} itd., np. arytmetyka Presburgera.

MIARY ZŁOŻONOŚCI

Dlaczego problemy trudno rozwiązywalne są takie trudne?

MIARY ZŁOŻONOŚCI

Dlaczego problemy trudno rozwiązywalne są takie trudne?

n	10	100	1000
$1 + log_2 n$	4	7	10
$5n$	50	500	5000
$n\ log\ n$	33	665	9966
n^2	100	10000	1 mln
n^3	1000	1 mln	1 mld
2^n	1024	liczba 31-cyfrowa	
$n!$	3,6 mln	liczba 161-cyfrowa	
n^n	10 mld	liczba 201-cyfrowa	

MIARY ZŁOŻONOŚCI

Czy zawsze trudne algorytmy są aż takie trudne?

MIARY ZŁOŻONOŚCI

Czy zawsze trudne algorytmy są aż takie trudne?

W praktyce algorytm wielomianowy o dużym wykładniku może dla danych o rozsądnej wielkości pracować znacznie gorzej niż algorytm wykładniczy! np. dla $n \leq 1165$ n^{1000} daje gorszy wynik niż $n!$.

MIARY ZŁOŻONOŚCI

Czy zawsze trudne algorytmy są aż takie trudne?

W praktyce algorytm wielomianowy o dużym wykładniku może dla danych o rozsądnej wielkości pracować znacznie gorzej niż algorytm wykładniczy! np. dla $n \leq 1165$ n^{1000} daje gorszy wynik niż $n!$.
Ale zazwyczaj algorytmy są ograniczone przez wielomiany niskiego stopnia.

MIARY ZŁOŻONOŚCI

Zagadkowa klasa NP:

MIARY ZŁOŻONOŚCI

Zagadkowa klasa NP:

Czy niedeterminizm wprowadzony do algorytmów coś zmienia?

MIARY ZŁOŻONOŚCI

Zagadkowa klasa NP:

Czy niedeterminizm wprowadzony do algorytmów coś zmienia?

W sensie obliczalności nic, ale w sensie wydajności bardzo wiele!

MIARY ZŁOŻONOŚCI

Zagadkowa klasa NP:

Czy niedeterminizm wprowadzony do algorytmów coś zmienia?

W sensie obliczalności nic, ale w sensie wydajności bardzo wiele!

NP (NondeterministicPolynomial) – klasa problemów o zasadniczej luce algorytmicznej: znane algorytmy deterministyczne są wykładnicze, ale najlepsze znane ograniczenia dolne są logarytmiczne!

MIARY ZŁOŻONOŚCI

Zagadkowa klasa NP:

Czy niedeterminizm wprowadzony do algorytmów coś zmienia?

W sensie obliczalności nic, ale w sensie wydajności bardzo wiele!

NP (NondeterministicPolynomial) – klasa problemów o zasadniczej luce algorytmicznej: znane algorytmy deterministyczne są wykładnicze, ale najlepsze znane ograniczenia dolne są logarytmiczne!
Co więcej, dla wielu problemów NP istnieją proste wielomianowe algorytmy niedeterministyczne.

MIARY ZŁOŻONOŚCI

Zagadkowa klasa NP:

MIARY ZŁOŻONOŚCI

Zagadkowa klasa NP:

Przykłady:

MIARY ZŁOŻONOŚCI

Zagadkowa klasa NP:

Przykłady:

- Sat-problem dla KRZ

MIARY ZŁOŻONOŚCI

Zagadkowa klasa NP:

Przykłady:

- Sat-problem dla KRZ
- Problem komiwojażera

MIARY ZŁOŻONOŚCI

Zagadkowa klasa NP:

Przykłady:

- Sat-problem dla KRZ
- Problem komiwojażera
- Problem ścieżki Hamiltona

MIARY ZŁOŻONOŚCI

Zagadkowa klasa NP:

Przykłady:

- Sat-problem dla KRZ
- Problem komiwojażera
- Problem ścieżki Hamiltona

i jeszcze ponad 1000 innych....

MIARY ZŁOŻONOŚCI

Zasadniczy problem (od 1971 – Cook):

MIARY ZŁOŻONOŚCI

Zasadniczy problem (od 1971 – Cook):

Czy $P = NP$?

MIARY ZŁOŻONOŚCI

Zasadniczy problem (od 1971 – Cook):

Czy $P = NP$?

Dokładniej: czy $NP \subseteq P$? albo czy NP-problemy są praktycznie rozwiązywalne?

MIARY ZŁOŻONOŚCI

Zasadniczy problem (od 1971 – Cook):

$$\text{Czy } P = NP?$$

Dokładniej: czy $NP \subseteq P$? albo czy NP-problemy są praktycznie rozwiązywalne?
Jeden z 7 tzw. milenijnych problemów matematycznych.

MIARY ZŁOŻONOŚCI

Zasadniczy problem (od 1971 – Cook):

Czy $P = NP$?

Dokładniej: czy $NP \subseteq P$? albo czy NP-problemy są praktycznie rozwiązywalne?
Jeden z 7 tzw. milenijnych problemów matematycznych.

Uwaga: gdyby $NP = P$, to można łatwo złamać wiele szyfrów!

MIARY ZŁOŻONOŚCI

Zasadniczy problem (od 1971 – Cook):

Czy $P = NP$?

Dokładniej: czy $NP \subseteq P$? albo czy NP-problemy są praktycznie rozwiązywalne?
Jeden z 7 tzw. milenijnych problemów matematycznych.

Uwaga: gdyby $NP = P$, to można łatwo złamać wiele szyfrów!
ale są mocne podstawy by wierzyć, że $NP \neq P$

MIARY ZŁOŻONOŚCI

Najtrudniejsze NP-problemy:

MIARY ZŁOŻONOŚCI

Najtrudniejsze NP-problemy:

Problem P_1 jest wielomianowo redukowalny do problemu P_2 ($P_1 \prec P_2$) wtw istnieje wielomianowa funkcja f z języka P_1 w język P_2, taka, że $x \in P_1$ wtw $f(x) \in P_2$.

MIARY ZŁOŻONOŚCI

Najtrudniejsze NP-problemy:

Problem P_1 jest wielomianowo redukowalny do problemu P_2 ($P_1 \prec P_2$) wtw istnieje wielomianowa funkcja f z języka P_1 w język P_2, taka, że $x \in P_1$ wtw $f(x) \in P_2$.
Problem A jest NP-zupełny wtw:

MIARY ZŁOŻONOŚCI

Najtrudniejsze NP-problemy:

Problem P_1 jest wielomianowo redukowalny do problemu P_2 ($P_1 \prec P_2$) wtw istnieje wielomianowa funkcja f z języka P_1 w język P_2, taka, że $x \in P_1$ wtw $f(x) \in P_2$.
Problem A jest NP-zupełny wtw:

- jest NP-trudny tj. $B \prec A$ dla dowolnego $B \in NP$

MIARY ZŁOŻONOŚCI

Najtrudniejsze NP-problemy:

Problem P_1 jest wielomianowo redukowalny do problemu P_2 ($P_1 \prec P_2$) wtw istnieje wielomianowa funkcja f z języka P_1 w język P_2, taka, że $x \in P_1$ wtw $f(x) \in P_2$.
Problem A jest NP-zupełny wtw:

- jest NP-trudny tj. $B \prec A$ dla dowolnego $B \in NP$
- $A \in NP$

MIARY ZŁOŻONOŚCI

Najtrudniejsze NP-problemy:

Problem P_1 jest wielomianowo redukowalny do problemu P_2 ($P_1 \prec P_2$) wtw istnieje wielomianowa funkcja f z języka P_1 w język P_2, taka, że $x \in P_1$ wtw $f(x) \in P_2$.
Problem A jest NP-zupełny wtw:

- jest NP-trudny tj. $B \prec A$ dla dowolnego $B \in NP$
- $A \in NP$

Zatem dla pozytywnego rozstrzygnięcia problemu $P = NP$? wystarczy wykazać, że jeden (dowolny) problem NP-zupełny należy do klasy P.

MIARY ZŁOŻONOŚCI

Najtrudniejsze NP-problemy:

Problem P_1 jest wielomianowo redukowalny do problemu P_2 ($P_1 \prec P_2$) wtw istnieje wielomianowa funkcja f z języka P_1 w język P_2, taka, że $x \in P_1$ wtw $f(x) \in P_2$.
Problem A jest NP-zupełny wtw:

- jest NP-trudny tj. $B \prec A$ dla dowolnego $B \in NP$
- $A \in NP$

Zatem dla pozytywnego rozstrzygnięcia problemu $P = NP$? wystarczy wykazać, że jeden (dowolny) problem NP-zupełny należy do klasy P.
Wszystkie podane dotąd przykłady NP-problemów są NP-zupełne (Cook 1971), ale np. problem czy x jest liczbą pierwszą? jest NP ale nie NP-zupełny (wiadomo też, że nie należy do klasy P).

MIARY ZŁOŻONOŚCI

A co z pamięcią?

MIARY ZŁOŻONOŚCI

A co z pamięcią?

1. $NP \subseteq PSPACE \supseteq coNP$, ale:

MIARY ZŁOŻONOŚCI

A co z pamięcią?

1. $NP \subseteq PSPACE \supseteq coNP$, ale:
    1. $NP \subset PSPACE$?

MIARY ZŁOŻONOŚCI

A co z pamięcią?

1. $NP \subseteq PSPACE \supseteq coNP$, ale:
   1. $NP \subset PSPACE$?
   2. $coNP \subset PSPACE$?

MIARY ZŁOŻONOŚCI

A co z pamięcią?

1. $NP \subseteq PSPACE \supseteq coNP$, ale:
   1. $NP \subset PSPACE$?
   2. $coNP \subset PSPACE$?
   3. $NP = coNP$?

MIARY ZŁOŻONOŚCI

A co z pamięcią?

1. $NP \subseteq PSPACE \supseteq coNP$, ale:
   1. $NP \subset PSPACE$?
   2. $coNP \subset PSPACE$?
   3. $NP = coNP$?
2. $PSPACE = NPSPACE = coPSPACE$ – Tw. Savitcha

MIARY ZŁOŻONOŚCI

Hierarchia problemów:

MIARY ZŁOŻONOŚCI

Hierarchia problemów:

Co wiemy na pewno:

MIARY ZŁOŻONOŚCI

Hierarchia problemów:

Co wiemy na pewno:

1. $LOG \subseteq LOGSPACE \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXP \subseteq NEXP \subseteq EXPSPACE...$

MIARY ZŁOŻONOŚCI

Hierarchia problemów:

Co wiemy na pewno:

1. $LOG \subseteq LOGSPACE \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXP \subseteq NEXP \subseteq EXPSPACE...$
2. $P \neq EXP$

MIARY ZŁOŻONOŚCI

Jak zwiększyć wydajność?

MIARY ZŁOŻONOŚCI

Jak zwiększyć wydajność?

- komputery równoległe i procesy współbieżne

MIARY ZŁOŻONOŚCI

Jak zwiększyć wydajność?

- komputery równoległe i procesy współbieżne
- komputery kwantowe

MIARY ZŁOŻONOŚCI

Jak zwiększyć wydajność?

- komputery równoległe i procesy współbieżne
- komputery kwantowe
- heurystyka – strategie rozwiązywania problemu szybko, choć bez gwarancji:

MIARY ZŁOŻONOŚCI

Jak zwiększyć wydajność?

- komputery równoległe i procesy współbieżne
- komputery kwantowe
- heurystyka – strategie rozwiązywania problemu szybko, choć bez gwarancji:
  - pełności

MIARY ZŁOŻONOŚCI

Jak zwiększyć wydajność?

- komputery równoległe i procesy współbieżne
- komputery kwantowe
- heurystyka – strategie rozwiązywania problemu szybko, choć bez gwarancji:
  - pełności
  - optymalności

HEURYSTYKA

Dla wydajnego rozwiązywania problemu należy skonstruować jego formalną reprezentację, czyli:

HEURYSTYKA

Dla wydajnego rozwiązywania problemu należy skonstruować jego formalną reprezentację, czyli:

1. zdefiniować adekwatną przestrzeń stanów (możliwe konfiguracje obiektów) – przestrzeń poszukiwania rozwiązania

HEURYSTYKA

Dla wydajnego rozwiązywania problemu należy skonstruować jego formalną reprezentację, czyli:

1. zdefiniować adekwatną przestrzeń stanów (możliwe konfiguracje obiektów) – przestrzeń poszukiwania rozwiązania
2. specyfikacja stanu(ów) wyjściowego(ych), z którego proces rozwiązania problemu może wystartować

HEURYSTYKA

Dla wydajnego rozwiązywania problemu należy skonstruować jego formalną reprezentację, czyli:

1. zdefiniować adekwatną przestrzeń stanów (możliwe konfiguracje obiektów) – przestrzeń poszukiwania rozwiązania
2. specyfikacja stanu(ów) wyjściowego(ych), z którego proces rozwiązania problemu może wystartować
3. specyfikacja stanu(ów) końcowego, akceptowanego jako rozwiązanie problemu

HEURYSTYKA

Dla wydajnego rozwiązywania problemu należy skonstruować jego formalną reprezentację, czyli:

1. zdefiniować adekwatną przestrzeń stanów (możliwe konfiguracje obiektów) – przestrzeń poszukiwania rozwiązania
2. specyfikacja stanu(ów) wyjściowego(ych), z którego proces rozwiązania problemu może wystartować
3. specyfikacja stanu(ów) końcowego, akceptowanego jako rozwiązanie problemu
4. określenie reguł przejść między stanami

HEURYSTYKA

Dla wydajnego rozwiązywania problemu należy skonstruować jego formalną reprezentację, czyli:

1. zdefiniować adekwatną przestrzeń stanów (możliwe konfiguracje obiektów) – przestrzeń poszukiwania rozwiązania
2. specyfikacja stanu(ów) wyjściowego(ych), z którego proces rozwiązania problemu może wystartować
3. specyfikacja stanu(ów) końcowego, akceptowanego jako rozwiązanie problemu
4. określenie reguł przejść między stanami
5. określenie strategii kontroli dla stosowania reguł (wędrówki po przestrzeni stanów – poszukiwanie rozwiązania)

HEURYSTYKA

Dla wydajnego rozwiązywania problemu należy skonstruować jego formalną reprezentację, czyli:

1. zdefiniować adekwatną przestrzeń stanów (możliwe konfiguracje obiektów) – przestrzeń poszukiwania rozwiązania
2. specyfikacja stanu(ów) wyjściowego(ych), z którego proces rozwiązania problemu może wystartować
3. specyfikacja stanu(ów) końcowego, akceptowanego jako rozwiązanie problemu
4. określenie reguł przejść między stanami
5. określenie strategii kontroli dla stosowania reguł (wędrówki po przestrzeni stanów – poszukiwanie rozwiązania)

ad. 1 np. drzewo albo graf skierowany

HEURYSTYKA

Właściwa strategia kontroli musi:

HEURYSTYKA

Właściwa strategia kontroli musi:

- prowadzić do zmian

HEURYSTYKA

Właściwa strategia kontroli musi:

- prowadzić do zmian
- być systematyczna

HEURYSTYKA

Właściwa strategia kontroli musi:

- prowadzić do zmian
- być systematyczna
- być wydajna – dawać niekoniecznie najlepsze rozwiązanie, ale wystarczająco dobre

HEURYSTYKA

Właściwa strategia kontroli musi:

- prowadzić do zmian
- być systematyczna
- być wydajna – dawać niekoniecznie najlepsze rozwiązanie, ale wystarczająco dobre

ad 3 np. znane szybkie algorytmy dla problemu komiwojażera

HEURYSTYKA

Aby wybrać dobrą heurystykę trzeba określić charakter problemu, m.in:

HEURYSTYKA

Aby wybrać dobrą heurystykę trzeba określić charakter problemu, m.in:

1. czy jest rozkładalny na podproblemy

HEURYSTYKA

Aby wybrać dobrą heurystykę trzeba określić charakter problemu, m.in:

1. czy jest rozkładalny na podproblemy
2. czy możliwe jest wycofywanie się z (lub ignorowanie) błędnych operacji

HEURYSTYKA

Aby wybrać dobrą heurystykę trzeba określić charakter problemu, m.in:

1. czy jest rozkładalny na podproblemy
2. czy możliwe jest wycofywanie się z (lub ignorowanie) błędnych operacji
3. czy dopuszcza przewidywanie konsekwencji operacji

HEURYSTYKA

Aby wybrać dobrą heurystykę trzeba określić charakter problemu, m.in:

1. czy jest rozkładalny na podproblemy
2. czy możliwe jest wycofywanie się z (lub ignorowanie) błędnych operacji
3. czy dopuszcza przewidywanie konsekwencji operacji
4. czy ma tylko jedno dobre rozwiązanie

HEURYSTYKA

Aby wybrać dobrą heurystykę trzeba określić charakter problemu, m.in:

1. czy jest rozkładalny na podproblemy
2. czy możliwe jest wycofywanie się z (lub ignorowanie) błędnych operacji
3. czy dopuszcza przewidywanie konsekwencji operacji
4. czy ma tylko jedno dobre rozwiązanie
5. czy dane są niesprzeczne

HEURYSTYKA

Aby wybrać dobrą heurystykę trzeba określić charakter problemu, m.in:

1. czy jest rozkładalny na podproblemy
2. czy możliwe jest wycofywanie się z (lub ignorowanie) błędnych operacji
3. czy dopuszcza przewidywanie konsekwencji operacji
4. czy ma tylko jedno dobre rozwiązanie
5. czy dane są niesprzeczne
6. czy zakłada udział dodatkowej wiedzy

HEURYSTYKA

Aby wybrać dobrą heurystykę trzeba określić charakter problemu, m.in:

1. czy jest rozkładalny na podproblemy
2. czy możliwe jest wycofywanie się z (lub ignorowanie) błędnych operacji
3. czy dopuszcza przewidywanie konsekwencji operacji
4. czy ma tylko jedno dobre rozwiązanie
5. czy dane są niesprzeczne
6. czy zakłada udział dodatkowej wiedzy
7. czy wymaga interakcji z człowiekiem

HEURYSTYKA

ad 2 – trzy rodzaje problemów:

HEURYSTYKA

ad 2 – trzy rodzaje problemów:

- ignorowalne – dowolne operacje mogą być zignorowane (np. dowodzenie twierdzeń w aks. czy w systemach dowodzenia zbieżnych)

HEURYSTYKA

ad 2 – trzy rodzaje problemów:

- ignorowalne – dowolne operacje mogą być zignorowane (np. dowodzenie twierdzeń w aks. czy w systemach dowodzenia zbieżnych)
- odzyskiwalne – można z błędnych operacji się wycofać (ale potrzebny backtracking – dowolne systemy dowodzenia)

HEURYSTYKA

ad 2 – trzy rodzaje problemów:

- ignorowalne – dowolne operacje mogą być zignorowane (np. dowodzenie twierdzeń w aks. czy w systemach dowodzenia zbieżnych)
- odzyskiwalne – można z błędnych operacji się wycofać (ale potrzebny backtracking – dowolne systemy dowodzenia)
- nieodzyskiwalne – bez możliwości wycofania np. gra w szachy

HEURYSTYKA

Strategie poszukiwania rozwiązania, wg. kierunku:

HEURYSTYKA

Strategie poszukiwania rozwiązania, wg. kierunku:

- do przodu

HEURYSTYKA

Strategie poszukiwania rozwiązania, wg. kierunku:

- do przodu
- do tyłu (goal-directed, backchaining)

HEURYSTYKA

Strategie poszukiwania rozwiązania, wg. kierunku:

- do przodu
- do tyłu (goal-directed, backchaining)
- łączona (ale należy uważać, żeby ścieżki się zeszły!)

HEURYSTYKA

O wyborze właściwego kierunku decydujemy wg. trzech kryteriów:

HEURYSTYKA

O wyborze właściwego kierunku decydujemy wg. trzech kryteriów:

- czego jest więcej: stanów wyjściowych czy docelowych? – lepiej iść od mniejszego zbioru do większego

HEURYSTYKA

O wyborze właściwego kierunku decydujemy wg. trzech kryteriów:

- czego jest więcej: stanów wyjściowych czy docelowych? – lepiej iść od mniejszego zbioru do większego
- na którym końcu przestrzeni poszukiwań jest mniejszy czynnik rozgałęzienia

HEURYSTYKA

O wyborze właściwego kierunku decydujemy wg. trzech kryteriów:

- czego jest więcej: stanów wyjściowych czy docelowych? – lepiej iść od mniejszego zbioru do większego
- na którym końcu przestrzeni poszukiwań jest mniejszy czynnik rozgałęzienia
- czy program musi dostarczyć uzasadnienia dla procesu wnioskowania

HEURYSTYKA

Strategie poszukiwania rozwiązania:

HEURYSTYKA

Strategie poszukiwania rozwiązania:

- depth-first search – zazwyczaj niezbędny backtracking

HEURYSTYKA

Strategie poszukiwania rozwiązania:

- depth-first search – zazwyczaj niezbędny backtracking
- breadth-first search – zazwyczaj wymaga więcej pamięci i pracy

HEURYSTYKA

Strategie poszukiwania rozwiązania:

- depth-first search – zazwyczaj niezbędny backtracking
- breadth-first search – zazwyczaj wymaga więcej pamięci i pracy
- best-first search

HEURYSTYKA

Strategie poszukiwania rozwiązania:

- depth-first search – zazwyczaj niezbędny backtracking
- breadth-first search – zazwyczaj wymaga więcej pamięci i pracy
- best-first search

ad 3 np. programy do gry w szachy analizujące zazwyczaj 5 do 7 sekwencji ruchów do przodu i wybierające najbardziej obiecującą ścieżkę (kompletne przestrzeń stanów dla rozgrywki szachowej wynosi ok. $36^{80} \approx 10^{124}$! – 36 - czynnik rozgałęzienia (średnia ilość możliwych ruchów w danym momencie), 80 - głębokość drzewa (przeciętna ilość ruchów w rozgrywce))

AUTOMATYCZNE DOWODZENIE

Wprowadzenie:

AUTOMATYCZNE DOWODZENIE

Wprowadzenie:

Automatyczne dowodzenie twierdzeń (ADT) jest współcześnie traktowane jako jedna z gałęzi AI, ale powstało wcześniej niż AI w sensie samodzielnej dyscypliny.

AUTOMATYCZNE DOWODZENIE

Wprowadzenie:

Automatyczne dowodzenie twierdzeń (ADT) jest współcześnie traktowane jako jedna z gałęzi AI, ale powstało wcześniej niż AI w sensie samodzielnej dyscypliny.

- lata 40/50-te – pierwsze komputery tylko liczyły

AUTOMATYCZNE DOWODZENIE

Wprowadzenie:

Automatyczne dowodzenie twierdzeń (ADT) jest współcześnie traktowane jako jedna z gałęzi AI, ale powstało wcześniej niż AI w sensie samodzielnej dyscypliny.

- lata 40/50-te – pierwsze komputery tylko liczyły
- lata 50/60-te – pierwsze programy do dowodzenia i gier

AUTOMATYCZNE DOWODZENIE

Wprowadzenie:

Automatyczne dowodzenie twierdzeń (ADT) jest współcześnie traktowane jako jedna z gałęzi AI, ale powstało wcześniej niż AI w sensie samodzielnej dyscypliny.

- lata 40/50-te – pierwsze komputery tylko liczyły
- lata 50/60-te – pierwsze programy do dowodzenia i gier

Uzasadnienie teoretyczne – Turing (1948):

AUTOMATYCZNE DOWODZENIE

Wprowadzenie:

Automatyczne dowodzenie twierdzeń (ADT) jest współcześnie traktowane jako jedna z gałęzi AI, ale powstało wcześniej niż AI w sensie samodzielnej dyscypliny.

- lata 40/50-te – pierwsze komputery tylko liczyły
- lata 50/60-te – pierwsze programy do dowodzenia i gier

Uzasadnienie teoretyczne – Turing (1948):
każdy problem można sprowadzić do szukania pewnego obiektu, a szukanie do wnioskowania.

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) a obliczalność:

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) a obliczalność:

Automatyzacja $\neq$ rozstrzygalność, gdyż:

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) a obliczalność:

Automatyzacja $\neq$ rozstrzygalność, gdyż:

- automatyzować można dowodzenie w dowolnych teoriach rekurencyjnie przeliczalnych

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) a obliczalność:

Automatyzacja $\neq$ rozstrzygalność, gdyż:

- automatyzować można dowodzenie w dowolnych teoriach rekurencyjnie przeliczalnych
- sama rozstrzygalność nie zapewnia praktycznej realizowalności

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) a obliczalność:

Automatyzacja $\neq$ rozstrzygalność, gdyż:

- automatyzować można dowodzenie w dowolnych teoriach rekurencyjnie przeliczalnych
- sama rozstrzygalność nie zapewnia praktycznej realizowalności
- nawet w teoriach nierozstrzygalnych można wydzielić fragmenty dopuszczające wydajną algorytmizację

AUTOMATYCZNE DOWODZENIE

Konstrukcja programu ADT wymaga:

AUTOMATYCZNE DOWODZENIE

Konstrukcja programu ADT wymaga:

1. określenia dziedziny problemu (ogólny, czy szczegółowe zadania)

AUTOMATYCZNE DOWODZENIE

Konstrukcja programu ADT wymaga:

1. określenia dziedziny problemu (ogólny, czy szczegółowe zadania)
2. określenia sposobu reprezentacji (np. pełny język 1-go rzędu, język klauzul, rachunek lambda)

AUTOMATYCZNE DOWODZENIE

Konstrukcja programu ADT wymaga:

1. określenia dziedziny problemu (ogólny, czy szczegółowe zadania)
2. określenia sposobu reprezentacji (np. pełny język 1-go rzędu, język klauzul, rachunek lambda)
3. wyboru systemu dedukcyjnego (np. rezolucja, TAB)

AUTOMATYCZNE DOWODZENIE

Konstrukcja programu ADT wymaga:

1. określenia dziedziny problemu (ogólny, czy szczegółowe zadania)
2. określenia sposobu reprezentacji (np. pełny język 1-go rzędu, język klauzul, rachunek lambda)
3. wyboru systemu dedukcyjnego (np. rezolucja, TAB)
4. zaprojektowania stosownego algorytmu

AUTOMATYCZNE DOWODZENIE

Konstrukcja programu ADT wymaga:

1. określenia dziedziny problemu (ogólny, czy szczegółowe zadania)
2. określenia sposobu reprezentacji (np. pełny język 1-go rzędu, język klauzul, rachunek lambda)
3. wyboru systemu dedukcyjnego (np. rezolucja, TAB)
4. zaprojektowania stosownego algorytmu
5. określenia strategii zwiększających wydajność

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – rodzaje programów:

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – rodzaje programów:

ze względu na zadania:

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – rodzaje programów:

ze względu na zadania:

- prover, np. ITP, HADES, PROTHEO, Otter, Vampire, THINKER, Oscar

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – rodzaje programów:

ze względu na zadania:

- prover, np. ITP, HADES, PROTHEO, Otter, Vampire, THINKER, Oscar
- checker, np. Mizar, LogiCola

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – rodzaje programów:

ze względu na zadania:

- prover, np. ITP, HADES, PROTHEO, Otter, Vampire, THINKER, Oscar
- checker, np. Mizar, LogiCola
- proof assistant, np. MacLogic, Tarski World, Heterogenious Logic

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – historia:

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – historia:

1. Metoda eliminacji kwantyfikatorów (skolemizacja) – 1919

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – historia:

1. Metoda eliminacji kwantyfikatorów (skolemizacja) – 1919
2. Procedura Herbranda – 1930

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – historia:

1. Metoda eliminacji kwantyfikatorów (skolemizacja) – 1919
2. Procedura Herbranda – 1930
3. Gentzen – rozstrzygalność RS – 1934

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – historia:

1. Metoda eliminacji kwantyfikatorów (skolemizacja) – 1919
2. Procedura Herbranda – 1930
3. Gentzen – rozstrzygalność RS – 1934
4. Davis – pierwszy prover dla arytmetyki Presburgera – 1954

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – historia:

1. Metoda eliminacji kwantyfikatorów (skolemizacja) – 1919
2. Procedura Herbranda – 1930
3. Gentzen – rozstrzygalność RS – 1934
4. Davis – pierwszy prover dla arytmetyki Presburgera – 1954
5. Newell, Shaw, Simon – Logic Theorist – 1956

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – historia:

1. Metoda eliminacji kwantyfikatorów (skolemizacja) – 1919
2. Procedura Herbranda – 1930
3. Gentzen – rozstrzygalność RS – 1934
4. Davis – pierwszy prover dla arytmetyki Presburgera – 1954
5. Newell, Shaw, Simon – Logic Theorist – 1956
6. Gelernter – Geometry Theorem-proving Machine – 1959

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – historia:

1. Metoda eliminacji kwantyfikatorów (skolemizacja) – 1919
2. Procedura Herbranda – 1930
3. Gentzen – rozstrzygalność RS – 1934
4. Davis – pierwszy prover dla arytmetyki Presburgera – 1954
5. Newell, Shaw, Simon – Logic Theorist – 1956
6. Gelernter – Geometry Theorem-proving Machine – 1959
7. Gilmore – pierwszy prover dla KRK (oparty na TAB) – 1959

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – historia:

1. Metoda eliminacji kwantyfikatorów (skolemizacja) – 1919
2. Procedura Herbranda – 1930
3. Gentzen – rozstrzygalność RS – 1934
4. Davis – pierwszy prover dla arytmetyki Presburgera – 1954
5. Newell, Shaw, Simon – Logic Theorist – 1956
6. Gelernter – Geometry Theorem-proving Machine – 1959
7. Gilmore – pierwszy prover dla KRK (oparty na TAB) – 1959
8. Hao Wang – pierwszy prover dla KRK (na RS) – 1958-1963

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – historia:

1. Metoda eliminacji kwantyfikatorów (skolemizacja) – 1919
2. Procedura Herbranda – 1930
3. Gentzen – rozstrzygalność RS – 1934
4. Davis – pierwszy prover dla arytmetyki Presburgera – 1954
5. Newell, Shaw, Simon – Logic Theorist – 1956
6. Gelernter – Geometry Theorem-proving Machine – 1959
7. Gilmore – pierwszy prover dla KRK (oparty na TAB) – 1959
8. Hao Wang – pierwszy prover dla KRK (na RS) – 1958-1963
9. Davis, Putnam – oryginalny prover dla KRK – 1960

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – historia:

1. Metoda eliminacji kwantyfikatorów (skolemizacja) – 1919
2. Procedura Herbranda – 1930
3. Gentzen – rozstrzygalność RS – 1934
4. Davis – pierwszy prover dla arytmetyki Presburgera – 1954
5. Newell, Shaw, Simon – Logic Theorist – 1956
6. Gelernter – Geometry Theorem-proving Machine – 1959
7. Gilmore – pierwszy prover dla KRK (oparty na TAB) – 1959
8. Hao Wang – pierwszy prover dla KRK (na RS) – 1958-1963
9. Davis, Putnam – oryginalny prover dla KRK – 1960
10. Prawitz – unifikacja termów – 1960

AUTOMATYCZNE DOWODZENIE

Automatyczne dowodzenie twierdzeń (ADT) – historia:

1. Metoda eliminacji kwantyfikatorów (skolemizacja) – 1919
2. Procedura Herbranda – 1930
3. Gentzen – rozstrzygalność RS – 1934
4. Davis – pierwszy prover dla arytmetyki Presburgera – 1954
5. Newell, Shaw, Simon – Logic Theorist – 1956
6. Gelernter – Geometry Theorem-proving Machine – 1959
7. Gilmore – pierwszy prover dla KRK (oparty na TAB) – 1959
8. Hao Wang – pierwszy prover dla KRK (na RS) – 1958-1963
9. Davis, Putnam – oryginalny prover dla KRK – 1960
10. Prawitz – unifikacja termów – 1960
11. Robinson – rezolucja (z unifikacją) – 1965

AUTOMATYCZNE DOWODZENIE

Przykład – prosty algorytm dla TAB Hintikki w KRZ:

AUTOMATYCZNE DOWODZENIE

Przykład – prosty algorytm dla TAB Hintikki w KRZ:

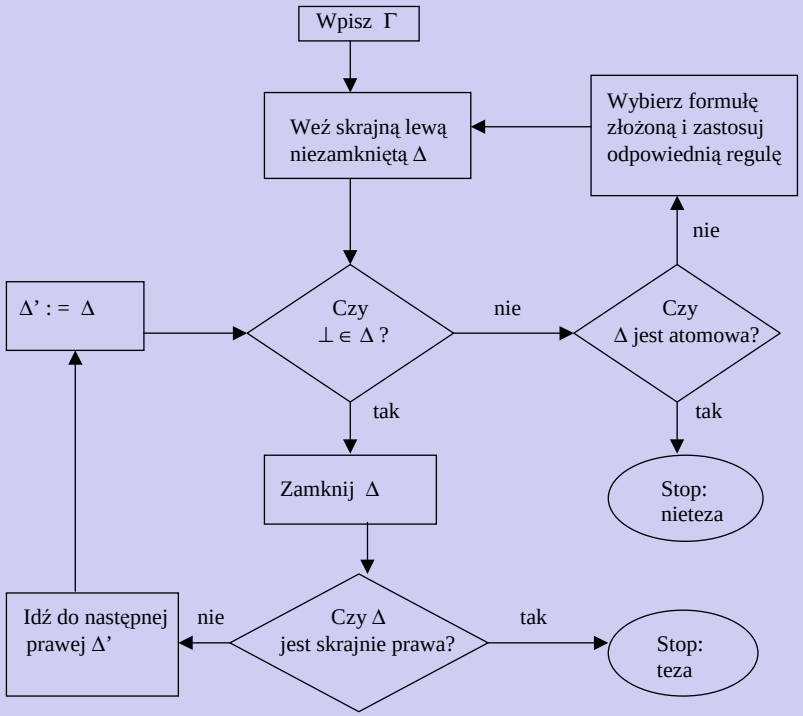
0. **Start** Wpisz Γ $(= \{\neg\varphi\})$ jako korzeń drzewa.
1. **Select** skrajny lewy niezamknięty liść Δ.
2. **If** $\bot \notin \Delta$, **then**
   2.1. **If** Δ jest atomowa, **then**
   stop: $\nvdash \varphi$
   else
   zastosuj regułę, **goto** 1.
   else zamknij Δ.
3. **If** Δ jest skrajnym prawym liściem, **then**
   stop: $\vdash \varphi$
   else
   3.1. przejdź do następnego liścia Δ'
   3.2. $\Delta' := \Delta$ **goto** 2.

AUTOMATYCZNE DOWODZENIE

Prosty algorytm dla TAB Hintikki w KRZ – schemat blokowy:

AUTOMATYCZNE DOWODZENIE

Prosty algorytm dla TAB Hintikki w KRZ – schemat blokowy:



AUTOMATYCZNE DOWODZENIE

Prosty algorytm dla TAB Hintikki w KRZ – komentarze:

AUTOMATYCZNE DOWODZENIE

Prosty algorytm dla TAB Hintikki w KRZ – komentarze:

1. drzewo budowane wg. strategii depth-first

AUTOMATYCZNE DOWODZENIE

Prosty algorytm dla TAB Hintikki w KRZ – komentarze:

1. drzewo budowane wg. strategii depth-first
2. wczesne zastosowanie testu sprzeczności

AUTOMATYCZNE DOWODZENIE

Prosty algorytm dla TAB Hintikki w KRZ – komentarze:

1. drzewo budowane wg. strategii depth-first
2. wczesne zastosowanie testu sprzeczności
3. ograniczony niedeterminizm

AUTOMATYCZNE DOWODZENIE

Prosty algorytm dla TAB Hintikki w KRZ – komentarze:

1. drzewo budowane wg. strategii depth-first
2. wczesne zastosowanie testu sprzeczności
3. ograniczony niedeterminizm
4. brak strategii preferencji

AUTOMATYCZNE DOWODZENIE

Rezolucja w KRK:

AUTOMATYCZNE DOWODZENIE

Rezolucja w KRK:

Jest to najbardziej rozpowszechniony i wydajny system dedukcyjny stosowany w ADT.

AUTOMATYCZNE DOWODZENIE

Rezolucja w KRK:

Jest to najbardziej rozpowszechniony i wydajny system dedukcyjny stosowany w ADT.
Reguła rezolucji w KRK zakłada wykorzystanie:

AUTOMATYCZNE DOWODZENIE

Rezolucja w KRK:

Jest to najbardziej rozpowszechniony i wydajny system dedukcyjny stosowany w ADT.
Reguła rezolucji w KRK zakłada wykorzystanie:

- skolemizacji termów

AUTOMATYCZNE DOWODZENIE

Rezolucja w KRK:

Jest to najbardziej rozpowszechniony i wydajny system dedukcyjny stosowany w ADT.
Reguła rezolucji w KRK zakłada wykorzystanie:

- skolemizacji termów
- unifikacji termów

REZOLUCJA

Rezolucja w KRK – skolemizacja:

REZOLUCJA

Rezolucja w KRK – skolemizacja:

Aby zastosować rezolucję w KRK trzeba każdą formułę φ odpowiednio przekształcić wg. następującej procedury:

REZOLUCJA

Rezolucja w KRK – skolemizacja:

Aby zastosować rezolucję w KRK trzeba każdą formułę φ odpowiednio przekształcić wg. następującej procedury:

1. Sprowadź φ do $PN(\varphi)$

REZOLUCJA

Rezolucja w KRK – skolemizacja:

Aby zastosować rezolucję w KRK trzeba każdą formułę φ odpowiednio przekształcić wg. następującej procedury:

1. Sprowadź φ do $PN(\varphi)$
2. Sprowadź $PN(\varphi)$ do $SK(\varphi)$

REZOLUCJA

Rezolucja w KRK – skolemizacja:

Aby zastosować rezolucję w KRK trzeba każdą formułę φ odpowiednio przekształcić wg. następującej procedury:

1. Sprowadź φ do $PN(\varphi)$
2. Sprowadź $PN(\varphi)$ do $SK(\varphi)$
3. Przekształć $SK(\varphi)$ w zbiór klauzul

REZOLUCJA

Rezolucja w KRK – skolemizacja:

Aby zastosować rezolucję w KRK trzeba każdą formułę φ odpowiednio przekształcić wg. następującej procedury:

1. Sprowadź φ do $PN(\varphi)$
2. Sprowadź $PN(\varphi)$ do $SK(\varphi)$
3. Przekształć $SK(\varphi)$ w zbiór klauzul

gdzie: $PN(\varphi)$ to normalna postać preneksowa φ, a $SK(\varphi)$ to postać skolemowa φ.

REZOLUCJA

Postacie preneksowe – definicja:

REZOLUCJA

Postacie preneksowe – definicja:

Formuła jest w normalnej postaci preneksowej wtw gdy ma postać $Q_1, ..., Q_n\psi$, gdzie $Q_1, ..., Q_n$ to ciąg kwantyfikatorów (tzw. prefiks) a ψ (tzw. matryca) to formuła, która składa się z koniunkcji i alternatyw literałów (tzn. nie zawiera kwantyfikatorów i innych spójników, a negacja występuje tylko przed formułami atomowymi).

REZOLUCJA

Postacie preneksowe – definicja:

Formuła jest w normalnej postaci preneksowej wtw gdy ma postać $Q_1, ..., Q_n\psi$, gdzie $Q_1, ..., Q_n$ to ciąg kwantyfikatorów (tzw. prefiks) a ψ (tzw. matryca) to formuła, która składa się z koniunkcji i alternatyw literałów (tzn. nie zawiera kwantyfikatorów i innych spójników, a negacja występuje tylko przed formułami atomowymi).
Twierdzenie: Każda formuła φ jest sprowadzalna do równoważnej normalnej postaci prefiksowej $PN(\varphi)$

REZOLUCJA

Postacie preneksowe – algorytm:

REZOLUCJA

Postacie preneksowe – algorytm:

1. Zastosuj reguły pozbywania się $\leftrightarrow$ i $\rightarrow$

REZOLUCJA

Postacie preneksowe – algorytm:

1. Zastosuj reguły pozbywania się $\leftrightarrow$ i $\rightarrow$
2. Zastosuj reguły DeMorgana

REZOLUCJA

Postacie preneksowe – algorytm:

1. Zastosuj reguły pozbywania się $\leftrightarrow$ i $\rightarrow$
2. Zastosuj reguły DeMorgana
3. użyj następujących równoważności i (ewentualnie) przemianowania zmiennych:

REZOLUCJA

Postacie preneksowe – algorytm:

1. Zastosuj reguły pozbywania się $\leftrightarrow$ i $\rightarrow$
2. Zastosuj reguły DeMorgana
3. użyj następujących równoważności i (ewentualnie) przemianowania zmiennych:

$$\begin{aligned} Qx\varphi(x) \star \psi &\leftrightarrow Qx(\varphi(x) \star \psi) \\ \forall x\varphi(x) \wedge \forall x\psi(x) &\leftrightarrow \forall x(\varphi(x) \wedge x\psi(x)) \\ \exists x\varphi(x) \vee \exists x\psi(x) &\leftrightarrow \exists x(\varphi(x) \vee x\psi(x)) \\ Q_1 x\varphi(x) \star Q_2 y\psi(y) &\leftrightarrow Q_1 x Q_2 z(\varphi(x) \star \psi(z)) \end{aligned}$$

gdzie $\star \in \{\vee, \wedge\}$, $Q, Q_1, Q_2 \in \{\forall, \exists\}$ a z jest zmienną, która nie występuje w φ.

REZOLUCJA

Skolemizacja PN-formuły polega na:

REZOLUCJA

Skolemizacja PN-formuły polega na:

1. eliminacji wszystkich kwantyfikatorów egzystencjalnych (od lewej do prawej), przy czym:

REZOLUCJA

Skolemizacja PN-formuły polega na:

1. eliminacji wszystkich kwantyfikatorów egzystencjalnych (od lewej do prawej), przy czym:
   - jeżeli przed $\exists x_i$ nie ma żadnych $\forall$, to za wszystkie wystąpienia x_i w matrycy wstawiamy nową stałą nazwową

REZOLUCJA

Skolemizacja PN-formuły polega na:

1. eliminacji wszystkich kwantyfikatorów egzystencjalnych (od lewej do prawej), przy czym:
   - jeżeli przed $\exists x_i$ nie ma żadnych $\forall$, to za wszystkie wystąpienia x_i w matrycy wstawiamy nową stałą nazwową
   - jeżeli przed $\exists x_i$ jest k wystąpień $\forall$, to za wszystkie wystąpienia x_i w matrycy wstawiamy nową stałą funkcyjną k-argumentową, której argumentami są te zmienne skwantyfikowane ogólnie, które występują w prefiksie przed $\exists x_i$

REZOLUCJA

Skolemizacja PN-formuły polega na:

1. eliminacji wszystkich kwantyfikatorów egzystencjalnych (od lewej do prawej), przy czym:
   - jeżeli przed $\exists x_i$ nie ma żadnych $\forall$, to za wszystkie wystąpienia x_i w matrycy wstawiamy nową stałą nazwową
   - jeżeli przed $\exists x_i$ jest k wystąpień $\forall$, to za wszystkie wystąpienia x_i w matrycy wstawiamy nową stałą funkcyjną k-argumentową, której argumentami są te zmienne skwantyfikowane ogólnie, które występują w prefiksie przed $\exists x_i$
2. eliminacji wszystkich kwantyfikatorów ogólnych z pozostawieniem zmiennych jako wolnych w matrycy

REZOLUCJA

Skolemizacja PN-formuły:

REZOLUCJA

Skolemizacja PN-formuły:

Przykład:

REZOLUCJA

Skolemizacja PN-formuły:

Przykład:

$\varphi := \exists x \forall y \forall z \exists v \forall w (Rxy \wedge Qyv \vee Pzx \wedge Svwz)$

REZOLUCJA

Skolemizacja PN-formuły:

Przykład:

$\varphi := \exists x \forall y \forall z \exists v \forall w (Rxy \wedge Qyv \vee Pzx \wedge Svwz)$
$SK(\varphi) := Ray \wedge Qyf(yz) \vee Pza \wedge Sf(yz)wz$

REZOLUCJA

Skolemizacja PN-formuły:

Przykład:

$\varphi := \exists x \forall y \forall z \exists v \forall w (Rxy \wedge Qyv \vee Pzx \wedge Svwz)$
$SK(\varphi) := Ray \wedge Qyf(yz) \vee Pza \wedge Sf(yz)wz$

Twierdzenie: φ jest spełnialne wtw $SK(\varphi)$ jest spełnialne

REZOLUCJA

Skolemizacja PN-formuły:

Przykład:

$\varphi := \exists x \forall y \forall z \exists v \forall w (Rxy \wedge Qyv \vee Pzx \wedge Svwz)$
$SK(\varphi) := Ray \wedge Qyf(yz) \vee Pza \wedge Sf(yz)wz$

Twierdzenie: φ jest spełnialne wtw $SK(\varphi)$ jest spełnialne

Uwaga: przekształcenie $SK(\varphi)$ w zbiór klauzul odbywa się tak samo jak w KRZ

REZOLUCJA

Unifikacja:

REZOLUCJA

Unifikacja:

Jest to operacja ujednolicania termów (formuł) poprzez znajdowanie dla nich wspólnego podstawienia.

REZOLUCJA

Unifikacja:

Jest to operacja ujednolicania termów (formuł) poprzez znajdowanie dla nich wspólnego podstawienia.
Podstawienie to dowolna funkcja $\sigma : Var \longrightarrow Term$, gdzie Var to skończony zbiór zmiennych nazwowych, a $Term$ to zbiór dowolnych wyrażeń nazwowych nie zawierających zmienych z Var.

REZOLUCJA

Unifikacja:

Jest to operacja ujednolicania termów (formuł) poprzez znajdowanie dla nich wspólnego podstawienia.
Podstawienie to dowolna funkcja $\sigma : Var \longrightarrow Term$, gdzie Var to skończony zbiór zmiennych nazwowych, a $Term$ to zbiór dowolnych wyrażeń nazwowych nie zawierających zmienych z Var.
np. $\sigma = \{x/f(y), y/g(x)\}$ nie jest podstawieniem.

REZOLUCJA

Unifikacja:

Jest to operacja ujednolicania termów (formuł) poprzez znajdowanie dla nich wspólnego podstawienia.
Podstawienie to dowolna funkcja $\sigma : Var \longrightarrow Term$, gdzie Var to skończony zbiór zmiennych nazwowych, a $Term$ to zbiór dowolnych wyrażeń nazwowych nie zawierających zmienych z Var.
np. $\sigma = \{x/f(y), y/g(x)\}$ nie jest podstawieniem.
Definicję podstawiania można poszerzyć na dowolne termy, formuły i ich zbiory:

REZOLUCJA

Unifikacja:

Jest to operacja ujednolicania termów (formuł) poprzez znajdowanie dla nich wspólnego podstawienia.
Podstawienie to dowolna funkcja $\sigma : Var \longrightarrow Term$, gdzie Var to skończony zbiór zmiennych nazwowych, a $Term$ to zbiór dowolnych wyrażeń nazwowych nie zawierających zmienych z Var.
np. $\sigma = \{x/f(y), y/g(x)\}$ nie jest podstawieniem.
Definicję podstawiania można poszerzyć na dowolne termy, formuły i ich zbiory:

$$\sigma(\tau^n(t_1, ..., t_n)) = \tau^n(\sigma(t_1), ..., \sigma(t_n))$$
$$\sigma(\varphi^n(t_1, ..., t_n)) = \varphi^n(\sigma(t_1), ..., \sigma(t_n)) \text{ itd.}$$

REZOLUCJA

Unifikacja:

REZOLUCJA

Unifikacja:

Dowolne podstawienie ujednolicające zbiór termów (formuł) to unifikator tego zbioru.

REZOLUCJA

Unifikacja:

Dowolne podstawienie ujednolicające zbiór termów (formuł) to unifikator tego zbioru.
O termach (formułach) posiadających unifikatory powiemy że są unifikowalne.

REZOLUCJA

Unifikacja:

Dowolne podstawienie ujednolicające zbiór termów (formuł) to unifikator tego zbioru.
O termach (formułach) posiadających unifikatory powiemy że są unifikowalne.
Przykłady par nieunifikowalnych:

REZOLUCJA

Unifikacja:

Dowolne podstawienie ujednolicające zbiór termów (formuł) to unifikator tego zbioru.
O termach (formułach) posiadających unifikatory powiemy że są unifikowalne.
Przykłady par nieunifikowalnych:

- Aa i Ba

REZOLUCJA

Unifikacja:

Dowolne podstawienie ujednolicające zbiór termów (formuł) to unifikator tego zbioru.
O termach (formułach) posiadających unifikatory powiemy że są unifikowalne.
Przykłady par nieunifikowalnych:

- *Aa* i *Ba*
- *Rxa* i *Rxb*

REZOLUCJA

Unifikacja:

Dowolne podstawienie ujednolicające zbiór termów (formuł) to unifikator tego zbioru.
O termach (formułach) posiadających unifikatory powiemy że są unifikowalne.
Przykłady par nieunifikowalnych:

- *Aa* i *Ba*
- *Rxa* i *Rxb*
- *Ra* i *Rax*

REZOLUCJA

Unifikacja:

Dowolne podstawienie ujednolicające zbiór termów (formuł) to unifikator tego zbioru.
O termach (formułach) posiadających unifikatory powiemy że są unifikowalne.
Przykłady par nieunifikowalnych:

- Aa i Ba
- Rxa i Rxb
- Ra i Rax
- $f(xx)$ i $f(yg(y))$

REZOLUCJA

Unifikacja:

REZOLUCJA

Unifikacja:

σ jest najogólniejszym unifikatorem zbioru X (mgu(X)) wtw dla dowolnego unifikatora σ' istnieje unifikator σ'' taki, że $\sigma' = \sigma \circ \sigma''$

REZOLUCJA

Unifikacja:

σ jest najogólniejszym unifikatorem zbioru X (mgu(X)) wtw dla dowolnego unifikatora σ' istnieje unifikator σ'' taki, że $\sigma' = \sigma \circ \sigma''$
Przykład:

REZOLUCJA

Unifikacja:

σ jest najogólniejszym unifikatorem zbioru X (mgu(X)) wtw dla dowolnego unifikatora σ' istnieje unifikator σ'' taki, że $\sigma' = \sigma \circ \sigma''$
Przykład:
$\sigma = \{y/h(x), z/a\}$ jest mgu($\{Ryh(a), Rh(x)h(z)\}$) a
$\sigma' = \{x/g(w), y/h(g(w)), z/a\}$ nie jest mgu chociaż też jest unifikatorem dla $\{Ryh(a), Rh(x)h(z)\}$.

REZOLUCJA

Unifikacja:

σ jest najogólniejszym unifikatorem zbioru X (mgu(X)) wtw dla dowolnego unifikatora σ' istnieje unifikator σ'' taki, że $\sigma' = \sigma \circ \sigma''$
Przykład:
$\sigma = \{y/h(x), z/a\}$ jest mgu($\{Ryh(a), Rh(x)h(z)\}$) a
$\sigma' = \{x/g(w), y/h(g(w)), z/a\}$ nie jest mgu chociaż też jest unifikatorem dla $\{Ryh(a), Rh(x)h(z)\}$.
$\sigma = \{y/a, x/c, z/c\}$ jest mgu($\{Raxx, Ryzc\}$) ale np.
$\{Raxx, Ryyb\}$ nie są unifikowalne.

REZOLUCJA

Algorytm unifikacji:

REZOLUCJA

Algorytm unifikacji:

Wejście: dwa termy τ_1, τ_2 takie, że $\tau_1 \neq \tau_2$ i $\mid \tau_1 \mid = \mid \tau_2 \mid$

REZOLUCJA

Algorytm unifikacji:

Wejście: dwa termy τ_1, τ_2 takie, że $\tau_1 \neq \tau_2$ i $\mid \tau_1 \mid = \mid \tau_2 \mid$
Wyjście: mgu($\{\tau_1, \tau_2\}$) albo zaprzeczenie jego istnienia (**fail**)

REZOLUCJA

Algorytm unifikacji:

Wejście: dwa termy τ_1, τ_2 takie, że $\tau_1 \neq \tau_2$ i $\mid \tau_1 \mid = \mid \tau_2 \mid$
Wyjście: mgu$(\{\tau_1, \tau_2\})$ albo zaprzeczenie jego istnienia (**fail**)

0. **Start** Wpisz $\sigma = \varnothing$ jako mgu$(\{\tau_1, \tau_2\})$.
1. **While** $\sigma(\tau_1) \neq \sigma(\tau_2)$ **do**
2. **Select** pierwszą od lewej niezgodną parę wyrażeń t_1, t_2.
   - 2.1. **If** ani t_1 ani t_2 nie są zmiennymi, **then stop: fail**
   - 2.2. **If** t_1 (lub t_2) jest zmienną, **then**
     if: t_1 występuje w t_2 (lub odwrotnie) **then stop: fail**
     else
     przyjmij, że $\sigma(t_1) = t_2$.

REZOLUCJA

Reguła rezolucji w KRK:

REZOLUCJA

Reguła rezolucji w KRK:

$$\Gamma \vee \varphi,\ \Delta \vee \neg\psi \ \vdash\ \Gamma' \vee \Delta',$$

REZOLUCJA

Reguła rezolucji w KRK:

$$\Gamma \vee \varphi,\ \Delta \vee \neg\psi \ \vdash\ \Gamma' \vee \Delta',$$

gdzie: σ jest mgu$(\{\varphi, \psi\})$, a Γ' i Δ' to faktoryzacje $\sigma(\Gamma)$ i $\sigma(\Delta)$.

REZOLUCJA

Reguła rezolucji w KRK:

$$\Gamma \vee \varphi,\ \Delta \vee \neg\psi \ \vdash\ \Gamma' \vee \Delta',$$

gdzie: σ jest mgu($\{\varphi, \psi\}$), a Γ' i Δ' to faktoryzacje $\sigma(\Gamma)$ i $\sigma(\Delta)$. Faktoryzacja to zredukowanie powtarzających się literałów do jednego wystąpienia.

REZOLUCJA

Reguła rezolucji w KRK:

$$\Gamma \vee \varphi,\ \Delta \vee \neg\psi \ \vdash\ \Gamma' \vee \Delta',$$

gdzie: σ jest mgu$(\{\varphi, \psi\})$, a Γ' i Δ' to faktoryzacje $\sigma(\Gamma)$ i $\sigma(\Delta)$.
Faktoryzacja to zredukowanie powtarzających się literałów do jednego wystąpienia.
Przykład:
$Ay \vee Rxf(y) \vee Bya,\ Qyz \vee Qya \vee \neg Baz \ \vdash\ Aa \vee Rxf(a) \vee Qaa$

REZOLUCJA

Zastosowania rezolucji:

REZOLUCJA

Zastosowania rezolucji:

- wykazywania nie/sprzeczności zbioru klauzul

REZOLUCJA

Zastosowania rezolucji:

- wykazywania nie/sprzeczności zbioru klauzul
- automatyczne wnioskowanie

REZOLUCJA

Zastosowania rezolucji:

- wykazywania nie/sprzeczności zbioru klauzul
- automatyczne wnioskowanie
- znajdowanie odpowiedzi na pytania (queries) do bazy danych

REZOLUCJA

Strategie zwiększania wydajności:

REZOLUCJA

Strategie zwiększania wydajności:

- saturacja (nasycanie) zbioru klauzul

REZOLUCJA

Strategie zwiększania wydajności:

- saturacja (nasycanie) zbioru klauzul
- strategie usuwania zbędnych klauzul:

REZOLUCJA

Strategie zwiększania wydajności:

- saturacja (nasycanie) zbioru klauzul
- strategie usuwania zbędnych klauzul:
  - z literałami niekomplementarnymi
  - tautologicznych
  - redundantnych (tzw. pochłoniętych)

REZOLUCJA

Strategie zwiększania wydajności:

- saturacja (nasycanie) zbioru klauzul
- strategie usuwania zbędnych klauzul:
  - z literałami niekomplementarnymi
  - tautologicznych
  - redundantnych (tzw. pochłoniętych)
- strategie sterowania procesem dedukcji:

REZOLUCJA

Strategie zwiększania wydajności:

- saturacja (nasycanie) zbioru klauzul
- strategie usuwania zbędnych klauzul:
  - z literałami niekomplementarnymi
  - tautologicznych
  - redundantnych (tzw. pochłoniętych)
- strategie sterowania procesem dedukcji:
  - rezolucja linearna
  - hiperrezolucja
  - zbiory uzasadnień

SZTUCZNA INTELIGENCJA

Co to właściwie znaczy?

SZTUCZNA INTELIGENCJA

Co to właściwie znaczy?

Sztuczna Inteligencja (Artificial Intelligence) termin wprowadzony przez J. McCarthy'ego w 1956. Wcześniej termin "inteligencja maszynowa" u A. Turinga w 1948.

SZTUCZNA INTELIGENCJA

Co to właściwie znaczy?

Sztuczna Inteligencja (Artificial Intelligence) termin wprowadzony przez J. McCarthy'ego w 1956. Wcześniej termin "inteligencja maszynowa" u A. Turinga w 1948.
Obecne użycie terminu AI jest wieloznaczne, oznacza zarówno:

SZTUCZNA INTELIGENCJA

Co to właściwie znaczy?

Sztuczna Inteligencja (Artificial Intelligence) termin wprowadzony przez J. McCarthy'ego w 1956. Wcześniej termin "inteligencja maszynowa" u A. Turinga w 1948.
Obecne użycie terminu AI jest wieloznaczne, oznacza zarówno:

1. dyscyplinę badawczą oraz zespół technik i metod stosowanych w tych badaniach

SZTUCZNA INTELIGENCJA

Co to właściwie znaczy?

Sztuczna Inteligencja (Artificial Intelligence) termin wprowadzony przez J. McCarthy'ego w 1956. Wcześniej termin "inteligencja maszynowa" u A. Turinga w 1948.
Obecne użycie terminu AI jest wieloznaczne, oznacza zarówno:

1. dyscyplinę badawczą oraz zespół technik i metod stosowanych w tych badaniach
2. poglądy filozoficzne dotyczące szeroko rozumianych relacji pomiędzy człowiekiem a maszyną

SZTUCZNA INTELIGENCJA

Co to właściwie znaczy?

Sztuczna Inteligencja (Artificial Intelligence) termin wprowadzony przez J. McCarthy'ego w 1956. Wcześniej termin "inteligencja maszynowa" u A. Turinga w 1948.
Obecne użycie terminu AI jest wieloznaczne, oznacza zarówno:

1. dyscyplinę badawczą oraz zespół technik i metod stosowanych w tych badaniach
2. poglądy filozoficzne dotyczące szeroko rozumianych relacji pomiędzy człowiekiem a maszyną
3. obiekty konstruowane w ramach AI1 i rozważane w ramach AI2

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina) – wybrane dziedziny badań:

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina) – wybrane dziedziny badań:

- dowodzenie twierdzeń

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina) – wybrane dziedziny badań:

- dowodzenie twierdzeń
- ogólne rozwiązywanie problemów

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina) – wybrane dziedziny badań:

- dowodzenie twierdzeń
- ogólne rozwiązywanie problemów
- szczegółowe rozwiązywanie problemów

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina) – wybrane dziedziny badań:

- dowodzenie twierdzeń
- ogólne rozwiązywanie problemów
- szczegółowe rozwiązywanie problemów
  - matematyka

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina) – wybrane dziedziny badań:

- dowodzenie twierdzeń
- ogólne rozwiązywanie problemów
- szczegółowe rozwiązywanie problemów
  - matematyka
  - diagnostyka medyczna

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina) – wybrane dziedziny badań:

- dowodzenie twierdzeń
- ogólne rozwiązywanie problemów
- szczegółowe rozwiązywanie problemów
  - matematyka
  - diagnostyka medyczna
  - analiza chemiczna

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina) – wybrane dziedziny badań:

- dowodzenie twierdzeń
- ogólne rozwiązywanie problemów
- szczegółowe rozwiązywanie problemów
  - matematyka
  - diagnostyka medyczna
  - analiza chemiczna
  - projektowanie inżynieryjne

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina) – wybrane dziedziny badań:

- dowodzenie twierdzeń
- ogólne rozwiązywanie problemów
- szczegółowe rozwiązywanie problemów
  - matematyka
  - diagnostyka medyczna
  - analiza chemiczna
  - projektowanie inżynieryjne
- gry

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina) – wybrane dziedziny badań:

- dowodzenie twierdzeń
- ogólne rozwiązywanie problemów
- szczegółowe rozwiązywanie problemów
  - matematyka
  - diagnostyka medyczna
  - analiza chemiczna
  - projektowanie inżynieryjne
- gry
- percepcja

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina) – wybrane dziedziny badań:

- dowodzenie twierdzeń
- ogólne rozwiązywanie problemów
- szczegółowe rozwiązywanie problemów
  - matematyka
  - diagnostyka medyczna
  - analiza chemiczna
  - projektowanie inżynieryjne
- gry
- percepcja
  - widzenie

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina) – wybrane dziedziny badań:

- dowodzenie twierdzeń
- ogólne rozwiązywanie problemów
- szczegółowe rozwiązywanie problemów
  - matematyka
  - diagnostyka medyczna
  - analiza chemiczna
  - projektowanie inżynieryjne
- gry
- percepcja
  - widzenie
  - mowa

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina) – wybrane dziedziny badań:

- dowodzenie twierdzeń
- ogólne rozwiązywanie problemów
- szczegółowe rozwiązywanie problemów
  - matematyka
  - diagnostyka medyczna
  - analiza chemiczna
  - projektowanie inżynieryjne
- gry
- percepcja
  - widzenie
  - mowa
- rozumienie języka naturalnego

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina):

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina):

Dwa podejścia:

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina):

Dwa podejścia:

1. symbolizm

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina):

Dwa podejścia:

1. symbolizm
2. koneksjonizm

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina):

Dwa podejścia:

1. symbolizm
2. koneksjonizm

ad 1: Hipoteza Newella i Simona (1976): Fizyczny system symboliczny ma konieczne i wystarczające środki dla wykonywania inteligentnych działań.

SZTUCZNA INTELIGENCJA

AI w sensie technicznym (dyscyplina):

Dwa podejścia:

1. symbolizm
2. koneksjonizm

ad 1: Hipoteza Newella i Simona (1976): Fizyczny system symboliczny ma konieczne i wystarczające środki dla wykonywania inteligentnych działań.
ad 2: Związek z badaniami nad procesami współbieżnymi, sieciami neuronowymi itp. W ADT uwzględnianie rozumowań oglądowych (np. Heterogenous Logic Barwise i Etchemendy'ego).

SZTUCZNA INTELIGENCJA

AI w sensie filozoficznym – kilka pytań:

SZTUCZNA INTELIGENCJA

AI w sensie filozoficznym – kilka pytań:

1. Czym jest inteligencja?

SZTUCZNA INTELIGENCJA

AI w sensie filozoficznym – kilka pytań:

1. Czym jest inteligencja?
2. Czym różnić się ma szuczna inteligencja od naturalnej?

SZTUCZNA INTELIGENCJA

AI w sensie filozoficznym – kilka pytań:

1. Czym jest inteligencja?
2. Czym różnić się ma szuczna inteligencja od naturalnej?
3. Czy AI już jest czy dopiero się pojawi?

SZTUCZNA INTELIGENCJA

AI w sensie filozoficznym – kilka pytań:

1. Czym jest inteligencja?
2. Czym różnić się ma szuczna inteligencja od naturalnej?
3. Czy AI już jest czy dopiero się pojawi?
4. Jak się mają do siebie terminy inteligencja, umysł, myślenie i świadomość?

SZTUCZNA INTELIGENCJA

AI w sensie filozoficznym – kilka pytań:

1. Czym jest inteligencja?
2. Czym różnić się ma szuczna inteligencja od naturalnej?
3. Czy AI już jest czy dopiero się pojawi?
4. Jak się mają do siebie terminy inteligencja, umysł, myślenie i świadomość?
5. Czy maszyny mogą myśleć? (Turing 1950)

SZTUCZNA INTELIGENCJA

AI w sensie filozoficznym – kilka pytań:

1. Czym jest inteligencja?
2. Czym różnić się ma szuczna inteligencja od naturalnej?
3. Czy AI już jest czy dopiero się pojawi?
4. Jak się mają do siebie terminy inteligencja, umysł, myślenie i świadomość?
5. Czy maszyny mogą myśleć? (Turing 1950)
6. Czy maszyny mogą być świadome?

SZTUCZNA INTELIGENCJA

Inteligencja – kilka elementarnych uwag:

SZTUCZNA INTELIGENCJA

Inteligencja – kilka elementarnych uwag:

- Inteligencja może być scharakteryzowana jako zdolność do rozwiązywania problemów.

SZTUCZNA INTELIGENCJA

Inteligencja – kilka elementarnych uwag:

- Inteligencja może być scharakteryzowana jako zdolność do rozwiązywania problemów.
- Sam termin wiele razy zmieniał znaczenie, np. w scholastyce oznaczał zdolność pojmowania pierwszych zasad (cechującą zwłaszcza anioły).

SZTUCZNA INTELIGENCJA

Inteligencja – kilka elementarnych uwag:

- Inteligencja może być scharakteryzowana jako zdolność do rozwiązywania problemów.
- Sam termin wiele razy zmieniał znaczenie, np. w scholastyce oznaczał zdolność pojmowania pierwszych zasad (cechującą zwłaszcza anioły).
- Inteligencja jest relatywna do gatunku, epoki, kręgu kulturowego.

SZTUCZNA INTELIGENCJA

Inteligencja – kilka elementarnych uwag:

- Inteligencja może być scharakteryzowana jako zdolność do rozwiązywania problemów.
- Sam termin wiele razy zmieniał znaczenie, np. w scholastyce oznaczał zdolność pojmowania pierwszych zasad (cechującą zwłaszcza anioły).
- Inteligencja jest relatywna do gatunku, epoki, kręgu kulturowego.
- Można wyróżnić kilka rodzajów inteligencji.

SZTUCZNA INTELIGENCJA

Świadome maszyny – 4 stanowiska wg. Penrose'a:

SZTUCZNA INTELIGENCJA

Świadome maszyny – 4 stanowiska wg. Penrose'a:

1. hard AI – umysł to po prostu oprogramowanie; świadomość jest funkcją jego złożoności; myślenie to proces algorytmiczny (obliczanie) zachodzący w układzie fizycznym (Turing)

SZTUCZNA INTELIGENCJA

Świadome maszyny – 4 stanowiska wg. Penrose'a:

1. hard AI – umysł to po prostu oprogramowanie; świadomość jest funkcją jego złożoności; myślenie to proces algorytmiczny (obliczanie) zachodzący w układzie fizycznym (Turing)
2. świadomość jest własnością fizjologicznej aktywności mózgu więc nie może być odtworzona w maszynie, ale nie jest to konieczne; charakterystyka myślenia jak w 1.

SZTUCZNA INTELIGENCJA

Świadome maszyny – 4 stanowiska wg. Penrose'a:

1. hard AI – umysł to po prostu oprogramowanie; świadomość jest funkcją jego złożoności; myślenie to proces algorytmiczny (obliczanie) zachodzący w układzie fizycznym (Turing)
2. świadomość jest własnością fizjologicznej aktywności mózgu więc nie może być odtworzona w maszynie, ale nie jest to konieczne; charakterystyka myślenia jak w 1.
3. myślenie i świadomość są wprawdzie zjawiskami fizycznymi ale wykraczają poza sferę obliczalności (Penrose)

SZTUCZNA INTELIGENCJA

Świadome maszyny – 4 stanowiska wg. Penrose'a:

1. hard AI – umysł to po prostu oprogramowanie; świadomość jest funkcją jego złożoności; myślenie to proces algorytmiczny (obliczanie) zachodzący w układzie fizycznym (Turing)
2. świadomość jest własnością fizjologicznej aktywności mózgu więc nie może być odtworzona w maszynie, ale nie jest to konieczne; charakterystyka myślenia jak w 1.
3. myślenie i świadomość są wprawdzie zjawiskami fizycznymi ale wykraczają poza sferę obliczalności (Penrose)
4. dualizm i spirytualizm – świadomość i myślenie nie są redukowalne do materii i procesów obliczalnych

SZTUCZNA INTELIGENCJA

Argumenty za poszczególnymi stanowiskami:

SZTUCZNA INTELIGENCJA

Argumenty za poszczególnymi stanowiskami:

ad 1. sukcesy symulacji komputerowych i kreowania rzeczywistości wirtualnej

SZTUCZNA INTELIGENCJA

Argumenty za poszczególnymi stanowiskami:

ad 1. sukcesy symulacji komputerowych i kreowania rzeczywistości wirtualnej

ad 2. wystarczająca jest możliwość symulacji (wpływ behawioryzmu – test Turinga)

SZTUCZNA INTELIGENCJA

Argumenty za poszczególnymi stanowiskami:

ad 1. sukcesy symulacji komputerowych i kreowania rzeczywistości wirtualnej

ad 2. wystarczająca jest możliwość symulacji (wpływ behawioryzmu – test Turinga)

ad 3. teoria kwantów implikuje niealgorytmiczność procesów mózgowych

SZTUCZNA INTELIGENCJA

Argumenty za poszczególnymi stanowiskami:

ad 1. sukcesy symulacji komputerowych i kreowania rzeczywistości wirtualnej

ad 2. wystarczająca jest możliwość symulacji (wpływ behawioryzmu – test Turinga)

ad 3. teoria kwantów implikuje niealgorytmiczność procesów mózgowych

ad 4. argument z twierdzenia Gödla

SZTUCZNA INTELIGENCJA

ad 1, 2: Test Turinga i chiński pokój Searle'a:

SZTUCZNA INTELIGENCJA

ad 1, 2: Test Turinga i chiński pokój Searle'a:

- Przewidywania Turinga nie spełniły się, nawet programy specjalistyczne łatwo rozszyfrować (np. Eliza)

SZTUCZNA INTELIGENCJA

ad 1, 2: Test Turinga i chiński pokój Searle'a:

- Przewidywania Turinga nie spełniły się, nawet programy specjalistyczne łatwo rozszyfrować (np. Eliza)
- Searle – nawet gdyby jakiś program zdał pomyślnie test Turinga, to i tak nie oznacza to posiadania inteligancji (argument chińskiego pokoju)

SZTUCZNA INTELIGENCJA

ad 4: czy twierdzenie Gödla coś przesądza?

SZTUCZNA INTELIGENCJA

ad 4: czy twierdzenie Gödla coś przesądza?

- Turing – człowiek nie jest w sytuacji bardziej uprzywilejowanej niż maszyna w poznaniu tego co niedowiedlne

SZTUCZNA INTELIGENCJA

ad 4: czy twierdzenie Gödla coś przesądza?

- Turing – człowiek nie jest w sytuacji bardziej uprzywilejowanej niż maszyna w poznaniu tego co niedowiedlne
- Gödel (Popper, Eccles) – zakłada, że człowiek może poznać prawdy niealgorytmizowalne, ale skoro materia podlega prawom obliczalnym, to musi istnieć umysł niematerialny

SZTUCZNA INTELIGENCJA

ad 4: czy twierdzenie Gödla coś przesądza?

- Turing – człowiek nie jest w sytuacji bardziej uprzywilejowanej niż maszyna w poznaniu tego co niedowiedlne
- Gödel (Popper, Eccles) – zakłada, że człowiek może poznać prawdy niealgorytmizowalne, ale skoro materia podlega prawom obliczalnym, to musi istnieć umysł niematerialny
- Penrose – zgadza się z Gödlem w pierwszej części ale przyjmuje, że w materii też prawa niealgorytmiczne, więc nie ma potrzeby postulować istnienia umysłu niematerialnego.

Literatura

W. Bibel, Deduction, Academic Press 1993
G. Boolos, R. Jeffrey, Computability and Logic, Cambridge 1974
C. Chang, R. Lee, Symbolic Logic and Mechanical Theorem Proving, Academic Press 1973
R. Davis, Truth, Deduction and Computation, New York 1989
K. Devlin, Żegnaj Kartezjuszu, Warszawa 1999
R. Epstein, W. Carnielli, Computability, Wadsworth 1989
M. Fitting, First-order Logic and Automated Theorem Proving, Springer 1996
J. Gallier, Logic for Computer Science, New York 1986
A. Grzegorczyk, Zagadnienie rozstrzygalności, PWN 1957

Literatura

A. Grzegorczyk, Zarys logiki matematycznej, wiele wydań
D. Harel, Rzecz o istocie informatyki, WNT 1992
W. Hillis, Wzory na krzemowej płytce, Warszawa 2000
J. Hopcroft, J. Ullman, Wprowadzenie do teorii automatów, języków i obliczeń, PWN 1994
R. Kowalski, Logika w rozwiązywaniu zadań, WNT 1989
S. Krajewski, Twierdzenie Gödla i jego interpretacje filozoficzne, Warszawa 2003
T. Krasiński, Automaty i języki formalne, Łódź 2007
W. Marciszewski, Sztuczna Inteligencja, Znak 1998
E. Mendelson, Introduction to Mathematical Logic, Van Nostrand 1964

Literatura

A. Mostowski, Z. Pawlak, Logika dla inżynierów, PWN 1970
R. Murawski, Funkcje rekurencyjne i elementy metamatematyki, Poznań 1990
E. Nagel, J. Newman, Twierdzenie Gödla, PWN 1966
Z. Pawlak, Automatyczne dowodzenie twierdzeń, PZWS 1965
R. Penrose, Nowy umysł cesarza, PWN 1995
R. Penrose, Cienie umysłu, Warszawa 2000
S. Pinker, Jak działa umysł?, Warszawa 2002
G. Polya, Jak to rozwiązać?, PWN 1993
A. Ramsay, Formal Methods in AI, Cambridge 1988

Literatura

A. Rich, Artificial Intelligence, McGraw-Hill 1983
A. Scott, Schody do umysłu, WNT 1999
J. Searle, Umysł na nowo odkryty, PIW 1999
A. Szałas, Zarys dedukcyjnych metod automatycznego wnioskowania, Warszawa 1992
R. Turner, Logics for AI, Ellis Horwood Ltd 1984
M. Wójcik, Zasada rezolucji, PWN 1991